The triangletools package

Vu Van Dung

May 8, 2020

Contents

1 Introduction

This package aims to help you construct special points in a triangle directly in a short and easy way. Using this package, you can construct most important points listed in Clark Kimberling's Encyclopedia of Triangle Centers (ETC). Currently, all points numbered from X_1 and X_{10}, as well as the excenter, are supported; however with other utilities in this package (see Section 3.4) and a bit of knowledge in geometry and expl3 programming, you can construct even more.

2 Loading the package

This package can be loaded as usual.

```
\usepackage{triangletools}
```

It will load Ti*k*Z and expl3 automatically.

3 User interface

The user interface of this package, including that of the utilities, is provided as pgf keys under the tree `/tikz/triangletools`.

Note that, in the following sections, a *coordinate* means a *named* Ti*k*Z coordinate. That is, in the following example,

```
\begin{tikzpicture}
  \draw (0,0) -- (3,0) coordinate (a);
\end{tikzpicture}
```

`a` is a named coordinate, while `0,0` or `3,0` are *not* named coordinates. The current implementation of this package only allows named coordinates in the user interface. It is like the `angles` Ti*k*Z library.

3.1 Accessing the keys

`/tikz/trt`

`/tikz/trt={`⟨*keys*⟩`}`

It executes *keys* with the key path set to `/tikz/triangletools`, which is the main key tree of this package.

This key is used to access all other keys in the user interface.

3.2 Circles associated with triangle centers

`\trtradius`

Some points, for example the incenter and the circumcenter, are associated with some special circles. If the requested point is associated with a circle, this macro stores the radius of that circle, in points (`pt`).

This macro is assigned *globally* every time a point is requested. Therefore, it stores the radius related to the last point that has a circle. So beware that while it always gives you some values once you have drawn such points, that value might not be what you want. It is recommended to use this macro *immediately after* the execution of triangle center keys.

In Section 3.3, if a point has a `\trtradius` associated to it, the circle will be drawn in the code example. Currently X_1, the excenter, X_3, X_5 and X_{10} can change the value of `\trtradius`.

If the macro is used before any center with a circle is constructed, an error message will be issued.

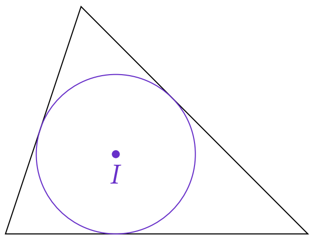
3.3 Triangle centers

`incenter`
`\trt_sp_incenter:nnnn`

`/tikz/triangletools/incenter=(`⟨*coor 1*⟩`)(`⟨*coor 2*⟩`)(`⟨*coor 3*⟩`)`
`\trt_sp_incenter:nnnn {`⟨*coor 1*⟩`}{`⟨*coor 2*⟩`}{`⟨*coor 3*⟩`}{`⟨*name*⟩`}`

Find the incenter X_1 of the triangle joining Ti*k*Z coordinates ⟨*coor 1*⟩, ⟨*coor 2*⟩ and ⟨*coor 3*⟩. The incenter is saved to Ti*k*Z coordinate ⟨*name*⟩.

If you use the key (why do you use the function anyway), ⟨*name*⟩ is set to `trt output` by default. You can change that using `output name`, see Section 3.5.

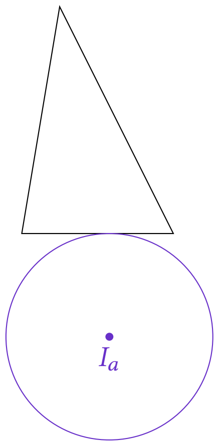


```
\begin {tikzpicture}
  \draw (1,3) coordinate (a) --
        (0,0) coordinate (b) --
        (4,0) coordinate (c) -- cycle;
  \fill[maincolor,trt={incenter=(a)(b)(c)}] (trt output)
    circle[radius=1.5pt] node[below] {$I$};
  \draw[maincolor] (trt output) circle (\trtradius);
\end{tikzpicture}
```

`excenter`
`\trt_sp_excenter:nnnn`

`/tikz/triangletools/excenter=(`⟨*coor 1*⟩`)(`⟨*coor 2*⟩`)(`⟨*coor 3*⟩`)`
`\trt_sp_excenter:nnnn {`⟨*coor 1*⟩`}{`⟨*coor 2*⟩`}{`⟨*coor 3*⟩`}{`⟨*name*⟩`}`

Find the excenter of the triangle. The returned point will be on the internal angular bisector at ⟨*coor 1*⟩. Note that the order matters: `excenter=(a)(b)(c)` is *different* from `excenter=(b)(a)(c)`.

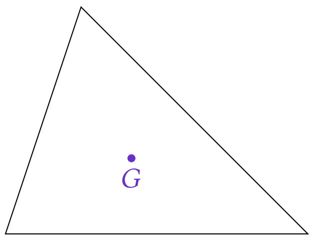


```
\begin {tikzpicture}
  \draw (.5,3) coordinate (a) --
        (0 ,0) coordinate (b) --
        (2 ,0) coordinate (c) -- cycle;
  \fill[maincolor,trt={excenter=(a)(b)(c)}] (trt output)
    circle[radius=1.5pt] node[below] {$I_a$};
  \draw[maincolor] (trt output) circle (\trtradius);
\end{tikzpicture}
```

`centroid`
`\trt_sp_centroid:nnnn`

`/tikz/triangletools/centroid=(`⟨*coor 1*⟩`)(`⟨*coor 2*⟩`)(`⟨*coor 3*⟩`)`
`\trt_sp_centroid:nnnn {`⟨*coor 1*⟩`}{`⟨*coor 2*⟩`}{`⟨*coor 3*⟩`}{`⟨*name*⟩`}`

Find the centroid X_2 of the triangle.

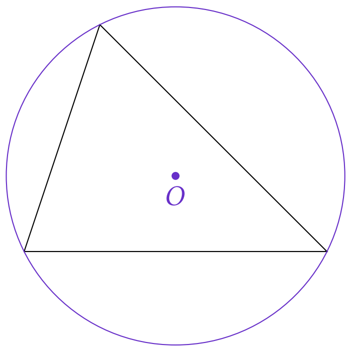


```
\begin {tikzpicture}
  \draw (1,3) coordinate (a) --
        (0,0) coordinate (b) --
        (4,0) coordinate (c) -- cycle;
  \fill[maincolor,trt={centroid=(a)(b)(c)}] (trt output)
    circle[radius=1.5pt] node[below] {$G$};
\end{tikzpicture}
```

`circumcenter`
`\trt_sp_circumcenter:nnnn`

`/tikz/triangletools/circumcenter=(`⟨*coor 1*⟩`)(`⟨*coor 2*⟩`)(`⟨*coor 3*⟩`)`
`\trt_sp_circumcenter:nnnn {`⟨*coor 1*⟩`}{`⟨*coor 2*⟩`}{`⟨*coor 3*⟩`}{`⟨*name*⟩`}`

Find the circumcenter X_3 of the triangle.

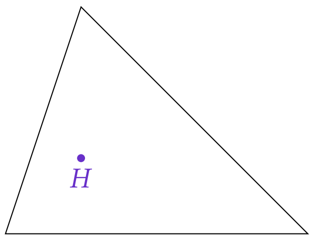


```
\begin {tikzpicture}
  \draw (1,3) coordinate (a) --
        (0,0) coordinate (b) --
        (4,0) coordinate (c) -- cycle;
  \fill[maincolor,trt={circumcenter=(a)(b)(c)}] (trt output)
    circle[radius=1.5pt] node[below] {$O$};
  \draw[maincolor] (trt output) circle (\trtradius);
\end{tikzpicture}
```

`orthocenter`
`\trt_sp_orthocenter:nnnn`

`/tikz/triangletools/orthocenter=(`⟨*coor 1*⟩`)(`⟨*coor 2*⟩`)(`⟨*coor 3*⟩`)`
`\trt_sp_orthocenter:nnnn {`⟨*coor 1*⟩`}{`⟨*coor 2*⟩`}{`⟨*coor 3*⟩`}{`⟨*name*⟩`}`

Find the orthocenter X_4 of the triangle.



```
\begin {tikzpicture}
  \draw (1,3) coordinate (a) --
        (0,0) coordinate (b) --
        (4,0) coordinate (c) -- cycle;
  \fill[maincolor,trt={orthocenter=(a)(b)(c)}] (trt output)
    circle[radius=1.5pt] node[below] {$H$};
\end{tikzpicture}
```

`triangle_center`

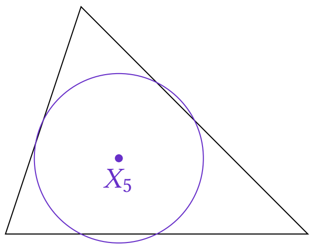
`/tikz/triangletools/triangle center=(`⟨*coor 1*⟩`)(`⟨*coor 2*⟩`)(`⟨*coor 3*⟩`)(`⟨*index*⟩`)`

Find the point $X_{\langle index\rangle}$ of the triangle. Currently ⟨*index*⟩ can be any integer between and including 1 and 10.

`\trt_sp_ninepointcenter:nnnn`

`/tikz/triangletools/triangle center=(`⟨*coor 1*⟩`)(`⟨*coor 2*⟩`)(`⟨*coor 3*⟩`)(5)`
`\trt_sp_ninepointcenter:nnnn {`⟨*coor 1*⟩`}{`⟨*coor 2*⟩`}{`⟨*coor 3*⟩`}{`⟨*name*⟩`}`

Find the nine-point center X_5 of the triangle.

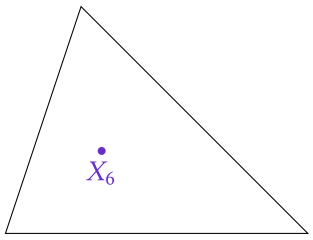


```
\begin {tikzpicture}
  \draw (1,3) coordinate (a) --
        (0,0) coordinate (b) --
        (4,0) coordinate (c) -- cycle;
  \fill[maincolor,trt={triangle center=(a)(b)(c)(5)}] (trt output)
    circle[radius=1.5pt] node[below] {$X_5$};
  \draw[maincolor] (trt output) circle (\trtradius);
\end{tikzpicture}
```

`\trt_sp_symmedian:nnnn`

`/tikz/triangletools/triangle center=(`*⟨coor 1⟩*`)(`*⟨coor 2⟩*`)(`*⟨coor 3⟩*`)(6)`
`\trt_sp_symmedian:nnnn {`*⟨coor 1⟩*`}{`*⟨coor 2⟩*`}{`*⟨coor 3⟩*`}{`*⟨name⟩*`}`

Find the symmedian point X_6 (*aka.* the Lemoine point or Grebe point) of the triangle.

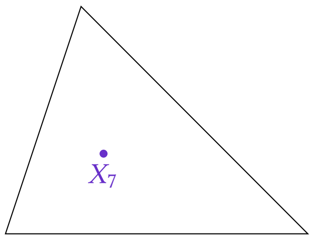


```
\begin {tikzpicture}
  \draw (1,3) coordinate (a) --
        (0,0) coordinate (b) --
        (4,0) coordinate (c) -- cycle;
  \fill[maincolor,trt={triangle center=(a)(b)(c)(6)}] (trt output)
    circle[radius=1.5pt] node[below] {$X_6$};
\end{tikzpicture}
```

`\trt_sp_gergonne:nnnn`

`/tikz/triangletools/triangle center=(`*⟨coor 1⟩*`)(`*⟨coor 2⟩*`)(`*⟨coor 3⟩*`)(7)`
`\trt_sp_gergonne:nnnn {`*⟨coor 1⟩*`}{`*⟨coor 2⟩*`}{`*⟨coor 3⟩*`}{`*⟨name⟩*`}`

Find the Gergonne point X_7 of the triangle.

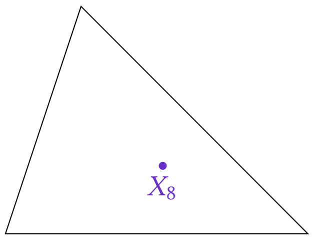


```
\begin {tikzpicture}
  \draw (1,3) coordinate (a) --
        (0,0) coordinate (b) --
        (4,0) coordinate (c) -- cycle;
  \fill[maincolor,trt={triangle center=(a)(b)(c)(7)}] (trt output)
    circle[radius=1.5pt] node[below] {$X_7$};
\end{tikzpicture}
```

`\trt_sp_nagel:nnnn`

`/tikz/triangletools/triangle center=(`*⟨coor 1⟩*`)(`*⟨coor 2⟩*`)(`*⟨coor 3⟩*`)(8)`
`\trt_sp_nagel:nnnn {`*⟨coor 1⟩*`}{`*⟨coor 2⟩*`}{`*⟨coor 3⟩*`}{`*⟨name⟩*`}`

Find the Nagel point X_8 of the triangle.

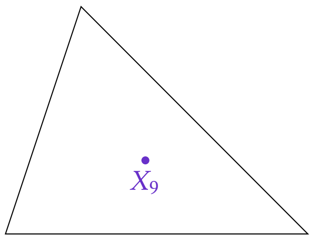


```
\begin {tikzpicture}
  \draw (1,3) coordinate (a) --
        (0,0) coordinate (b) --
        (4,0) coordinate (c) -- cycle;
  \fill[maincolor,trt={triangle center=(a)(b)(c)(8)}] (trt output)
    circle[radius=1.5pt] node[below] {$X_8$};
\end{tikzpicture}
```

`\trt_sp_mittenpunkt:nnnn`

`/tikz/triangletools/triangle center=(`*⟨coor 1⟩*`)(`*⟨coor 2⟩*`)(`*⟨coor 3⟩*`)(9)`
`\trt_sp_mittenpunkt:nnnn {`*⟨coor 1⟩*`}{`*⟨coor 2⟩*`}{`*⟨coor 3⟩*`}{`*⟨name⟩*`}`

Find the *mittenpunkt* X_9 of the triangle.

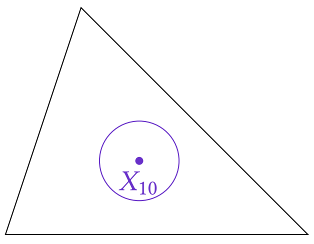


```
\begin {tikzpicture}
  \draw (1,3) coordinate (a) --
        (0,0) coordinate (b) --
        (4,0) coordinate (c) -- cycle;
  \fill[maincolor,trt={triangle center=(a)(b)(c)(9)}] (trt output)
    circle[radius=1.5pt] node[below] {$X_9$};
\end{tikzpicture}
```

`\trt_sp_spieker:nnnn`

`/tikz/triangletools/triangle center=(`*⟨coor 1⟩*`)(`*⟨coor 2⟩*`)(`*⟨coor 3⟩*`)(10)`
`\trt_sp_spieker:nnnn {`*⟨coor 1⟩*`}{`*⟨coor 2⟩*`}{`*⟨coor 3⟩*`}{`*⟨name⟩*`}`

Find the Spieker center X_{10} of the triangle.



```
\begin {tikzpicture}
  \draw (1,3) coordinate (a) --
        (0,0) coordinate (b) --
        (4,0) coordinate (c) -- cycle;
  \fill[maincolor,trt={triangle center=(a)(b)(c)(10)}] (trt output)
    circle[radius=1.5pt] node[below] {$X_{10}$};
  \draw[maincolor] (trt output) circle (\trtradius);
\end{tikzpicture}
```

3.4 Other utilities

3.4.1 Line tools

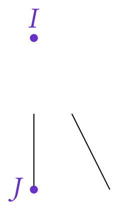
The line tools utility can helps you play with some (very basic) operations related to lines.

`intersection`

`/tikz/triangletools/intersection=(⟨coor 1⟩)(⟨coor 2⟩)--(⟨coor 3⟩)(⟨coor 4⟩)`

There are two lines, the first joins ⟨*coor 1*⟩ and ⟨*coor 2*⟩, and the other joins ⟨*coor 3*⟩ and ⟨*coor 4*⟩. This finds the intersection of these lines.

If the two lines are parallel, `trt output` is set to `(0,0)`, and the package will report a warning.

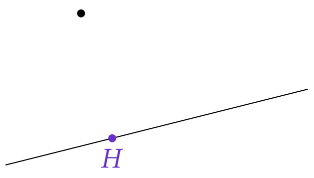


```
\begin {tikzpicture}
  \draw (0,0) coordinate (a) -- (0,1) coordinate (b);
  \draw (1,0) coordinate (c) -- (.5,1) coordinate (d);
  \coordinate (e) at (1,1);
  \fill[maincolor,trt={intersection=(a)(b)--(c)(d)}] (trt output)
    circle (1.5pt) node[above] {$I$};
  \fill[maincolor,trt={intersection=(a)(b)--(c)(e)}] (trt output)
    circle (1.5pt) node[left] {$J$};
\end{tikzpicture}
```

`foot_of_perpendicular`

`/tikz/triangletools/foot of perpendicular=(⟨coor 1⟩)--(⟨coor 2⟩)(⟨coor 3⟩)`

Find the foot of perpendicular of point ⟨*coor 1*⟩ to the line joining ⟨*coor 2*⟩ and ⟨*coor 3*⟩.



```
\begin {tikzpicture}
  \draw (0,0) coordinate (b) -- (4,1) coordinate (c);
  \fill (1,2) circle (1.5pt) coordinate (a);
  \fill[maincolor,trt={foot of perpendicular=(a)--(b)(c)}] (trt output)
    circle (1.5pt) node[below] {$H$};
\end{tikzpicture}
```

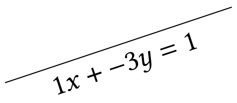
You can do much more using these expl3 functions.

`\trt_lt_get_line_equation:nnNNN`

`\trt_lt_get_line_equation:nnNNN{⟨coor 1⟩}{⟨coor 2⟩}⟨a⟩⟨b⟩⟨c⟩`

Find the equation of the line joining ⟨*coor 1*⟩ and ⟨*coor 2*⟩, in the form of $ax + by = c$. The l3fp *local* variables ⟨*a*⟩, ⟨*b*⟩ and ⟨*c*⟩ will be set accordingly.

Note that for any pair of points ⟨*coor 1*⟩ and ⟨*coor 2*⟩, there are infinitely many solutions for ⟨*a*⟩, ⟨*b*⟩ and ⟨*c*⟩. This function will produce one of such solution. While the solution is likely to be the simplest of all possible ones, this is not guaranteed.



```
\begin {tikzpicture}
  \coordinate (a) at (1,0);
  \coordinate (b) at (4,1);
  \ExplSyntaxOn
  \fp_new:N \l_foo_tmpa_fp
  \fp_new:N \l_foo_tmpb_fp
  \fp_new:N \l_foo_tmpc_fp
  \trt_lt_get_line_equation:nnNNN {a} {b}
    \l_foo_tmpa_fp \l_foo_tmpb_fp \l_foo_tmpc_fp
  \fp_new:N \c_foo_cm_fp
  \fp_set:Nn \c_foo_cm_fp {28.45275590551181}
  \def \resultequation {
    $\fp_eval:n {round (\l_foo_tmpa_fp / \c_foo_cm_fp)}x +
     \fp_eval:n {round (\l_foo_tmpb_fp / \c_foo_cm_fp)}y
    =\fp_eval:n {round (\l_foo_tmpc_fp / (\c_foo_cm_fp * \c_foo_cm_fp))}$
  }
  \ExplSyntaxOff
  \draw (a) -- (b) node[midway,sloped,below] {\resultequation};
\end{tikzpicture}
```

`\trt_lt_get_intersection_line:NNNNNNNN`

`\trt_lt_get_intersection_line:NNNNNNNN`⟨*a1*⟩⟨*b1*⟩⟨*c1*⟩⟨*a2*⟩⟨*b2*⟩⟨*c2*⟩⟨*x*⟩⟨*y*⟩

This function finds the intersection of lines $a_1x + b_1y = c_1$ and $a_2x + b_2y = c_2$, afterwards store the dimensions of the intersection in variables ⟨x⟩ and ⟨y⟩.

All arguments are floating points variables, ⟨x⟩ and ⟨y⟩ needs to be local variables.

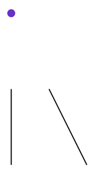
`\trt_lt_get_intersection_coordinate:nnnnNN`

`\trt_lt_get_intersection_coordinate:nnnnNN`
`{`⟨*coor 1*⟩`}{`⟨*coor 2*⟩`}{`⟨*coor 3*⟩`}{`⟨*coor 4*⟩`}`⟨*x*⟩⟨*y*⟩

This function is a wrapper of `\trt_lt_get_intersection_line:NNNNNNNN`. It finds the intersection of the line joining ⟨*coor 1*⟩, ⟨*coor 2*⟩ and the line joining ⟨*coor 3*⟩, ⟨*coor 4*⟩. The dimensions of the returned point is stored in ⟨x⟩ and ⟨y⟩, which are local l3fp variables.

A warning will be raised if the lines are parallel, in that case ⟨x⟩ and ⟨y⟩ are set to zero.

This is the base of `intersection`.

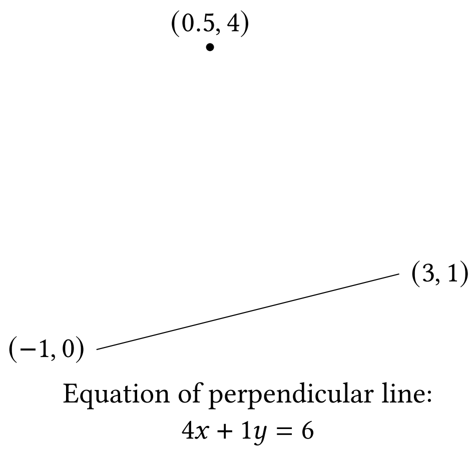


```
\begin {tikzpicture}
  \draw (0,0) coordinate (a) -- (0,1) coordinate (b);
  \draw (1,0) coordinate (c) -- (.5,1) coordinate (d);
  \ExplSyntaxOn
  \fp_new:N \l_foo_tmpa_fp
  \fp_new:N \l_foo_tmpb_fp
  \trt_lt_get_intersection_coordinate:nnnnNN {a} {b} {c} {d}
    \l_foo_tmpa_fp \l_foo_tmpb_fp
  \coordinate (i) at (\fp_to_dim:N \l_foo_tmpa_fp,
                      \fp_to_dim:N \l_foo_tmpb_fp);
  \ExplSyntaxOff
  \fill[maincolor] (i) circle (1.5pt);
\end{tikzpicture}
```

`\trt_lt_get_perpendicular_equation:nNNNNNN`

`\trt_lt_get_perpendicular_equation:nNNNNNN{`⟨*coor 1*⟩`}`⟨*a1*⟩⟨*b1*⟩⟨*c1*⟩⟨*a2*⟩⟨*b2*⟩⟨*c2*⟩

This function finds the line of equation $a_2x + b_2y = c_2$ that passes coordinate ⟨*coor 1*⟩ and is perpendicular to $a_1x + b_1y = c_1$.



```
\begin {tikzpicture}
  \draw (-1,0) coordinate (b) node[left] {$(-1,0)$} --
        (3,1)  coordinate (c) node[right] {$(3,1)$};
  \fill (0.5,4) circle (1.5pt) coordinate (a) node[above] {$(0.5,4)$};
  \ExplSyntaxOn
  \fp_new:N \l_foo_tmpa_fp
  \fp_new:N \l_foo_tmpb_fp
  \fp_new:N \l_foo_tmpc_fp
  \fp_new:N \l_foo_tmpd_fp
  \fp_new:N \l_foo_tmpe_fp
  \fp_new:N \l_foo_tmpf_fp
  \trt_lt_get_line_equation:nnNNN {b} {c}
    \l_foo_tmpa_fp \l_foo_tmpb_fp \l_foo_tmpc_fp
  \trt_lt_get_perpendicular_equation:nNNNNNN {a}
    \l_foo_tmpa_fp \l_foo_tmpb_fp \l_foo_tmpc_fp
    \l_foo_tmpd_fp \l_foo_tmpe_fp \l_foo_tmpf_fp
  \fp_new:N \c_foo_cm_fp
  \fp_set:Nn \c_foo_cm_fp {28.45275590551181}
  \def \resultequation {
    $\fp_eval:n {round (\l_foo_tmpd_fp / \c_foo_cm_fp)}x +
     \fp_eval:n {round (\l_foo_tmpe_fp / \c_foo_cm_fp)}y
    =\fp_eval:n {round (\l_foo_tmpf_fp / (\c_foo_cm_fp * \c_foo_cm_fp))}$
  }
  \ExplSyntaxOff
  \path (1,0) node[below=3mm,align=center]
    {Equation of perpendicular line:\\\resultequation};
\end{tikzpicture}
```

`\trt_lt_get_perpendicular_coordinate:nnnNN`

`\trt_lt_get_perpendicular_coordinate:nnnNN`{⟨*coor 1*⟩}{⟨*coor 2*⟩}{⟨*coor 3*⟩}⟨*x*⟩⟨*y*⟩

Find the dimensions of the foot of perpendicular from ⟨*coor 1*⟩ to the line joining ⟨*coor 2*⟩ and ⟨*coor 3*⟩. Afterwards store the dimensions found in ⟨*x*⟩ and ⟨*y*⟩.

This is the base of `foot of perpendicular`.

3.4.2 The barycentric coordinate system

`initialize_barycentric`

`/tikz/triangletools/initialize barycentric=(`⟨*coor 1*⟩`)(`⟨*coor 2*⟩`)(`⟨*coor 3*⟩`)`

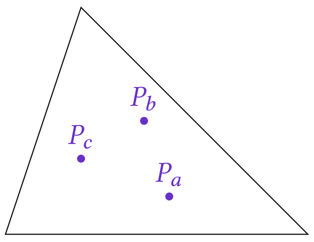
Use the three coordinates as "anchors" of the barycentric coordinate system.

`bc3`

`(bc3 cs:`⟨*l1*⟩`,`⟨*l2*⟩`,`⟨*l3*⟩`)`

Using the barycentric coordinate system. Note that the system needs to be initialized in advance using `initialize barycentric`, and an error message will be reported if you do otherwise.

The sum of ⟨*l1*⟩, ⟨*l2*⟩ and ⟨*l3*⟩ is not necessarily 1 – the package will take care of that internally.



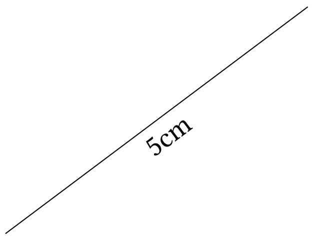
```
\begin {tikzpicture}
  \draw (1,3) coordinate (a) --
        (0,0) coordinate (b) --
        (4,0) coordinate (c) -- cycle;
  \fill[maincolor,trt={initialize barycentric=(a)(b)(c)}] (bc3 cs:1,2,3)
    circle[radius=1.5pt] node[above] {$P_a$};
  \fill[maincolor,trt={initialize barycentric=(b)(c)(a)}] (bc3 cs:1,2,3)
    circle[radius=1.5pt] node[above] {$P_b$};
  \fill[maincolor,trt={initialize barycentric=(c)(a)(b)}] (bc3 cs:1,2,3)
    circle[radius=1.5pt] node[above] {$P_c$};
\end{tikzpicture}
```

3.4.3 Distance-finding utility

`\trt_distance:nnN`

`\trt_distance:nnN {`⟨*coor 1*⟩`} {`⟨*coor 2*⟩`}` ⟨*fp var*⟩

Find distance between ⟨*coor 1*⟩ and ⟨*coor 2*⟩, and store that value to ⟨*fp var*⟩.



```
\begin {tikzpicture}
  \path (0,0) coordinate (a) (4,3) coordinate (b);
  \ExplSyntaxOn
  \fp_new:N \l_foo_tmpa_fp
  \trt_distance:nnN {a} {b} \l_foo_tmpa_fp
  \fp_new:N \c_foo_cm_fp
  \fp_set:Nn \c_foo_cm_fp {28.45275590551181}
  \draw (a) -- (b) node[midway,sloped,below]
    { \fp_eval:n {round(\l_foo_tmpa_fp / \c_foo_cm_fp)} cm };
  \ExplSyntaxOff
\end{tikzpicture}
```

`\trt_distance_triangle:nnnNNN`

`\trt_distance_triangle:nnnNNN{`⟨*coor 1*⟩`}{`⟨*coor 2*⟩`}{`⟨*coor 3*⟩`}`⟨*a*⟩⟨*b*⟩⟨*c*⟩

`\trt_distance:nnN` is needed to find the side lengths in a triangle (these side lengths are very helpful in many areas, for instance in this package to find special points based on the barycentric system). However, using it three times in a row is not quite elegant; this function is defined to automate that process.

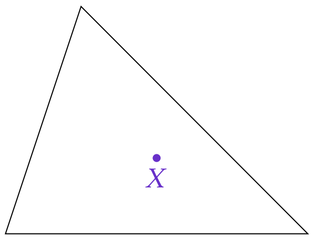
⟨*a*⟩ is set to the distance between ⟨*coor 2*⟩ and ⟨*coor 3*⟩, similar things happen for ⟨*b*⟩ and ⟨*c*⟩.

3.5 Customization

`output_name`

`/tikz/triangletools/output name=`⟨*name*⟩

This key can be used to change the name of the returned coordinates. The initial value of this key is `trt output`.



```
\begin {tikzpicture}[trt={output name=hello world}]
  \draw (1,3) coordinate (a) --
        (0,0) coordinate (b) --
        (4,0) coordinate (c) -- cycle;
  \fill[maincolor,trt={circumcenter=(a)(b)(c)}] (hello world)
    circle[radius=1.5pt] node[below] {$X$};
\end{tikzpicture}
```

4 Implementation

```
⟨@@=trt⟩
```

4.1 The main package file

```
⟨*triangletools⟩
\RequirePackage{tikz}
\RequirePackage{expl3}
\ProvidesExplPackage {triangletools} {2020/04/30} {0.1}
  {TikZ support for triangular geometry}
```

`\trt@tmp@i`
`\trt@tmp@ii`

We will use these dimensions many times to extract the dimensions of a Ti*k*Z coordinate.

```
\newdimen\trt@tmp@i
\newdimen\trt@tmp@ii
```

(*End definition for* `\trt@tmp@i` *and* `\trt@tmp@ii`*. These functions are documented on page* **??**.)

Let's load the necessary subpackage files.

```
\input {trtmessages.code.tex}
\input {trtlinetools.code.tex}
\input {trtbarycentric.code.tex}
\input {trtdistance.code.tex}
\input {trtspecialpoints.code.tex}
\input {trtfrontend.code.tex}
⟨/triangletools⟩
```

4.2 Errors and warnings

```
⟨*messages⟩
\ProvidesExplFile {trtmessages.code.tex} {2020/04/30} {0.1}
  {The ~ triangletools ~ package: ~ Messages}
```

We also need to declare some helpful messages that we will use later on.

In `trtlinetools.code.tex`, when we find the intersection of two lines, a warning will be shown if the lines are parallel. The warning is based on `intersection-not-found`.

```
\msg_new:nnnn {triangletools} {intersection-not-found}
  {
    Intersection ~ not ~ found.
  }
  {
    You ~ told ~ me ~ to ~ find ~ the ~ intersection ~ of ~ the ~ line ~
    joining ~ #1 ~ and ~ #2 ~ and ~ the ~ line ~ joining ~ #3 ~ and ~ #4, ~
    however ~ these ~ lines ~ are ~ parallel ~ so ~ I ~ can't ~ find ~ any ~
    intersection. ~ The ~ return ~ point ~ is ~ set ~ to ~ the ~ origin ~
    (0, ~ 0).
  }
```

When the barycentric coordinate system, implemented in `trtbarycentric.code.tex`, is used, it should already be initialized, *i.e.* we should already know what are the three "anchor" coordinates of the system. If the coordinate system is not yet initialized, this error will be shown.

```
\msg_new:nnnn {triangletools} {uninitialized}
  {
    Barycentric ~ coordinate ~ system ~ not ~ initialized.
  }
  {
    You ~ have ~ not ~ initialized ~ the ~ three ~ anchor ~ points ~ for ~
    the ~ coordinate ~ system. ~ Please ~ initialize ~ the ~ points ~
    before ~ using ~ the ~ 'bc3' ~ coordinate ~ system.
  }
```

We do let the user to find triangle center X_i for any i. However this package obviously can't implement all points in ETC (in fact, I will implement only some most important points). An error will be raised if the user tries to use an unimplemented point.

```
\msg_new:nnnn {triangletools} {center-not-found}
  {
    Triangle ~ center ~ not ~ found.
  }
  {
    I ~ can't ~ find ~ the ~ requested ~ triangle ~ center, ~ because ~
    point ~ X(#1) ~ is ~ not ~ yet ~ implemented ~ in ~ the ~ triangletools ~
    package. ~ Try ~ to ~ construct ~ it ~ yourself.
  }
```

We need to guard against using `\trtradius` before the macro stores something.

```
\msg_new:nnnn {triangletools} {no-radius-found}
  {
    No ~ circles ~ can ~ be ~ constructed.
  }
  {
    I ~ can’t ~ construct ~ the ~ requested ~ circle, ~ because ~ you ~ have ~
    not ~ request ~ me ~ to ~ construct ~ any ~ triangle ~ centers ~ that ~
    are ~ associated ~ to ~ a ~ circle. ~ I ~ will ~ set ~
    \protect\trtradius\space to ~ zero ~ now.
  }
⟨/messages⟩
```

4.3 The backend layer

4.3.1 The line tools utility

```
⟨*linetools⟩
\ProvidesExplFile {trtlinetools.code.tex} {2020/04/30} {0.1}
  {The ~ triangletools ~ package: ~ Utilities ~ for ~ lines}
```

In `trtlinetools.code.tex`, we will implement the necessary functions to handle lines in a mathematical way.

Firstly, let’s declare some internal variables that we will use later.

`\l__trt_lt_pointi_x_fp`
`\l__trt_lt_pointi_y_fp`
`\l__trt_lt_pointii_x_fp`
`\l__trt_lt_pointii_y_fp`

These variables are used to store coordinates of the two vertices of a segment.

```
\fp_new:N \l__trt_lt_pointi_x_fp
\fp_new:N \l__trt_lt_pointi_y_fp
\fp_new:N \l__trt_lt_pointii_x_fp
\fp_new:N \l__trt_lt_pointii_y_fp
```

(*End definition for* `\l__trt_lt_pointi_x_fp` *and others.*)

`\l__trt_lt_linei_a_fp`
`\l__trt_lt_linei_b_fp`
`\l__trt_lt_linei_c_fp`
`\l__trt_lt_lineii_a_fp`
`\l__trt_lt_lineii_b_fp`
`\l__trt_lt_lineii_c_fp`

We will store the line equation under the format of $ax + by = c$, because this is the most generic format. These six variables will do that job.

```
\fp_new:N \l__trt_lt_linei_a_fp
\fp_new:N \l__trt_lt_linei_b_fp
\fp_new:N \l__trt_lt_linei_c_fp
\fp_new:N \l__trt_lt_lineii_a_fp
\fp_new:N \l__trt_lt_lineii_b_fp
\fp_new:N \l__trt_lt_lineii_c_fp
```

(*End definition for* `\l__trt_lt_linei_a_fp` *and others.*)

`\l__trt_lt_tmp_fp`
`\l__trt_lt_tmpa_fp`
`\l__trt_lt_tmpb_fp`

Some additional temporary variables.

```
\fp_new:N \l__trt_lt_tmp_fp
\fp_new:N \l__trt_lt_tmpa_fp
\fp_new:N \l__trt_lt_tmpb_fp
```

(*End definition for* `\l__trt_lt_tmp_fp` *,* `\l__trt_lt_tmpa_fp` *, and* `\l__trt_lt_tmpb_fp`*.*)

`\trt_lt_get_line_equation:nnNNN`

Find the equation of the line passing #1 and #2, and store the values of a, b, c found to #3, #4 and #5, which are floating point variables, respectively.

```
\cs_new:Npn \trt_lt_get_line_equation:nnNNN #1 #2 #3 #4 #5
  {
    \pgfextractx {\trt@tmp@i} {\pgfpointanchor{#1}{center}}
    \pgfextracty {\trt@tmp@ii} {\pgfpointanchor{#1}{center}}
    \fp_set:Nn \l__trt_i_pointi_x_fp {\trt@tmp@i}
    \fp_set:Nn \l__trt_i_pointi_y_fp {\trt@tmp@ii}
    \pgfextractx {\trt@tmp@i} {\pgfpointanchor{#2}{center}}
    \pgfextracty {\trt@tmp@ii} {\pgfpointanchor{#2}{center}}
    \fp_set:Nn \l__trt_i_pointii_x_fp {\trt@tmp@i}
    \fp_set:Nn \l__trt_i_pointii_y_fp {\trt@tmp@ii}
```

There is a simple hack here. We have $ax_1 + by_1 = c = ax_2 + by_2$, which is equivalent to $a(x_1 - x_2) = b(y_2 - y_1)$. Therefore $a = y_2 - y_1$ and $b = x_1 - x_2$ can be used.

```
    \fp_set:Nn #3
      {
        \l__trt_i_pointii_y_fp - \l__trt_i_pointi_y_fp
      }
    \fp_set:Nn #4
      {
        \l__trt_i_pointi_x_fp - \l__trt_i_pointii_x_fp
      }
    \fp_set:Nn #5
      {
        #3 * \l__trt_i_pointi_x_fp + #4 * \l__trt_i_pointi_y_fp
      }
  }
```

(*End definition for* `\trt_lt_get_line_equation:nnNNN`. *This function is documented on page 6.*)

`\trt_lt_get_intersection_line:NNNNNNNN` Find the intersection of two lines with given equation, after that store the intersection coordinate to floating point variables #7 and #8.

```
\cs_new:Npn \trt_lt_get_intersection_line:NNNNNNNN #1 #2 #3 #4 #5 #6 #7 #8
  {
    \fp_set:Nn \l__trt_lt_tmp_fp { #1 * #5 - #4 * #2 }
```

If `\l__trt_lt_tmp_fp` is zero, the two lines are parallel. In that case, we will issue a warning, and set the intersection coordinate to $(0, 0)$. Otherwise, continue computing as usual.

```
    \fp_compare:nNnTF {\l__trt_lt_tmp_fp} = {0}
      {
        \msg_warning:nnnnnn {triangletools} {intersection-not-found}
          {(#1)} {(#2)} {(#3)} {(#4)}
        \fp_set:Nn #7 {0}
        \fp_set:Nn #8 {0}
      }
      {
        \fp_set:Nn #7 { ( #5 * #3 - #2 * #6 ) / \l__trt_lt_tmp_fp }
        \fp_set:Nn #8 { ( #1 * #6 - #4 * #3 ) / \l__trt_lt_tmp_fp }
      }
  }
```

(*End definition for* `\trt_lt_get_intersection_line:NNNNNNNN`. *This function is documented on page 7.*)

`\trt_lt_get_intersection_coordinate:nnnnNN` Let's generalize `\trt_lt_get_intersection_line:NNNNNNNN`. The following function finds the intersection of two lines between #1, #2 and #3, #4, and store the coordinates to #5 and #6. We still use floating point variables here, as they might be useful in the future.

```
\cs_new:Npn \trt_lt_get_intersection_coordinate:nnnnNN #1 #2 #3 #4 #5 #6
  {
    \trt_lt_get_line_equation:nnNNN {#1} {#2}
      \l__trt_lt_linei_a_fp \l__trt_lt_linei_b_fp \l__trt_lt_linei_c_fp
    \trt_lt_get_line_equation:nnNNN {#3} {#4}
      \l__trt_lt_lineii_a_fp \l__trt_lt_lineii_b_fp \l__trt_lt_lineii_c_fp
    \trt_lt_get_intersection_line:NNNNNNNN
      \l__trt_lt_linei_a_fp \l__trt_lt_linei_b_fp \l__trt_lt_linei_c_fp
      \l__trt_lt_lineii_a_fp \l__trt_lt_lineii_b_fp \l__trt_lt_lineii_c_fp
      #5 #6
  }
```

(*End definition for* `\trt_lt_get_intersection_coordinate:nnnnNN`*. This function is documented on page 7.*)

`\__trt_lt_return_intersection:nnnnn`

Now, let's Ti*k*Zify the above function! Note that I use `overlay` because I don't want to affect the bounding box. The user can use the returned coordinate to change the bounding box in whatever way he wants to.

```
\cs_new:Npn \__trt_lt_return_intersection:nnnnn #1 #2 #3 #4 #5
  {
    \trt_lt_get_intersection_coordinate:nnnnNN {#1} {#2} {#3} {#4}
      \l__trt_lt_tmpa_fp \l__trt_lt_tmpb_fp
    \coordinate[overlay] (#5) at
      (\fp_to_dim:N \l__trt_lt_tmpa_fp, \fp_to_dim:N \l__trt_lt_tmpb_fp);
  }
```

(*End definition for* `\__trt_lt_return_intersection:nnnnn`*.*)

Next, let's make some implementation regarding perpendicularity.

`\trt_lt_get_perpendicular_equation:nNNNNNN`

This function finds the equation of the line passing point and being perpendicular to a line having a given equation. The task is not quite complicated: note that lines $ax + by = c$ and $ay - bx = d$ are perpendicular.

```
\cs_new:Npn \trt_lt_get_perpendicular_equation:nNNNNNN #1 #2 #3 #4 #5 #6 #7
  {
    \fp_set:Nn #5 { -#3 }
    \fp_set:Nn #6 { #2 }
    \pgfextractx {\trt@tmp@i} {\pgfpointanchor{#1}{center}}
    \pgfextracty {\trt@tmp@ii} {\pgfpointanchor{#1}{center}}
    \fp_set:Nn \l__trt_lt_tmpa_fp {\trt@tmp@i}
    \fp_set:Nn \l__trt_lt_tmpb_fp {\trt@tmp@ii}
    \fp_set:Nn #7 { #5 * \l__trt_lt_tmpa_fp + #6 * \l__trt_lt_tmpb_fp }
  }
```

(*End definition for* `\trt_lt_get_perpendicular_equation:nNNNNNN`*. This function is documented on page 7.*)

`\trt_lt_get_perpendicular_coordinate:nnnNN`

The base implemented, let's find the foot of perpendicular from a point to a segment.

```
\cs_new:Npn \trt_lt_get_perpendicular_coordinate:nnnNN #1 #2 #3 #4 #5
  {
    \trt_lt_get_line_equation:nnNNN {#2} {#3}
      \l__trt_lt_linei_a_fp \l__trt_lt_linei_b_fp \l__trt_lt_linei_c_fp

    \trt_lt_get_perpendicular_equation:nNNNNNN {#1}
      \l__trt_lt_linei_a_fp \l__trt_lt_linei_b_fp \l__trt_lt_linei_c_fp
      \l__trt_lt_lineii_a_fp \l__trt_lt_lineii_b_fp \l__trt_lt_lineii_c_fp

    \trt_lt_get_intersection_line:NNNNNNNN
      \l__trt_lt_linei_a_fp \l__trt_lt_linei_b_fp \l__trt_lt_linei_c_fp
      \l__trt_lt_lineii_a_fp \l__trt_lt_lineii_b_fp \l__trt_lt_lineii_c_fp
      #4 #5
  }
```

(*End definition for* `\trt_lt_get_perpendicular_coordinate:nnnNN`*. This function is documented on page 8.*)

`rt_lt_return_perpendicular_coordinate:nnnn`

This is just a wrapper of `\trt_lt_get_perpendicular_coordinate:nnnNN`.

```
\cs_new:Npn \__trt_lt_return_perpendicular_coordinate:nnnn #1 #2 #3 #4
  {
    \trt_lt_get_perpendicular_coordinate:nnnNN {#1} {#2} {#3}
      \l__trt_lt_tmpa_fp \l__trt_lt_tmpb_fp
    \coordinate[overlay] (#4) at
```

```
      (\fp_to_dim:N \l__trt_lt_tmpa_fp, \fp_to_dim:N \l__trt_lt_tmpb_fp);
  }
⟨/linetools⟩
```

(*End definition for* `\__trt_lt_return_perpendicular_coordinate:nnnn`.)

4.3.2 The barycentric coordinate system utility

```
⟨*barycentric⟩
\ProvidesExplFile {trtbarycentric.code.tex} {2020/04/30} {0.1}
  {
    The ~ triangletools ~ package: ~ Utilities ~ for ~ the ~ barycentric ~
    coordinate ~ system.
  }
```

In `trtbarycentric.code.tex`, we will implement the three-point barycentric coordinate system, which is essential in constructing many special points in a triangle.

`\l__trt_bc_anchor_ix_fp`
`\l__trt_bc_anchor_iy_fp`
`\l__trt_bc_anchor_iix_fp`
`\l__trt_bc_anchor_iiy_fp`
`\l__trt_bc_anchor_iiix_fp`
`\l__trt_bc_anchor_iiiy_fp`

We use six variables to store the 'anchors' of the barycentric coordinate system.

```
\fp_new:N \l__trt_bc_anchor_ix_fp
\fp_new:N \l__trt_bc_anchor_iy_fp
\fp_new:N \l__trt_bc_anchor_iix_fp
\fp_new:N \l__trt_bc_anchor_iiy_fp
\fp_new:N \l__trt_bc_anchor_iiix_fp
\fp_new:N \l__trt_bc_anchor_iiiy_fp
```

(*End definition for* `\l__trt_bc_anchor_ix_fp` *and others.*)

`\l__trt_bc_lambda_i_fp`
`\l__trt_bc_lambda_ii_fp`
`\l__trt_bc_lambda_iii_fp`

We use these variables to store the user input coordinate. Note that our system is a three-point one, hence exactly three number is required.

Why lambda λ? Well, I don't know. Wikipedia uses that, so I do the same.

```
\fp_new:N \l__trt_bc_lambda_i_fp
\fp_new:N \l__trt_bc_lambda_ii_fp
\fp_new:N \l__trt_bc_lambda_iii_fp
```

(*End definition for* `\l__trt_bc_lambda_i_fp`, `\l__trt_bc_lambda_ii_fp`, *and* `\l__trt_bc_lambda_iii_fp`.)

`\l__trt_bc_initialized_bool`

We need to guard against using the system before initializing. This boolean variable does that job: if it is set to `false` (default), do nothing.

```
\bool_new:N \l__trt_bc_initialized_bool
```

(*End definition for* `\l__trt_bc_initialized_bool`.)

`\l__trt_bc_tmp_fp`

A temporary variable.

```
\fp_new:N \l__trt_bc_tmp_fp
```

(*End definition for* `\l__trt_bc_tmp_fp`.)

`\__trt_bc_initialize:nnn`

Initialize the barycentric coordinate system. This is the only place where `\l__trt_bc_initialized_bool` can be set to true, so this function must be executed before everything else in this file.

```
\cs_new:Npn \__trt_bc_initialize:nnn #1 #2 #3
  {
    \bool_set_true:N \l__trt_bc_initialized_bool
    \pgfextractx {\trt@tmp@i} {\pgfpointanchor{#1}{center}}
    \pgfextracty {\trt@tmp@ii} {\pgfpointanchor{#1}{center}}
    \fp_set:Nn \l__trt_bc_anchor_ix_fp {\trt@tmp@i}
    \fp_set:Nn \l__trt_bc_anchor_iy_fp {\trt@tmp@ii}
    \pgfextractx {\trt@tmp@i} {\pgfpointanchor{#2}{center}}
    \pgfextracty {\trt@tmp@ii} {\pgfpointanchor{#2}{center}}
    \fp_set:Nn \l__trt_bc_anchor_iix_fp {\trt@tmp@i}
    \fp_set:Nn \l__trt_bc_anchor_iiy_fp {\trt@tmp@ii}
    \pgfextractx {\trt@tmp@i} {\pgfpointanchor{#3}{center}}
    \pgfextracty {\trt@tmp@ii} {\pgfpointanchor{#3}{center}}
    \fp_set:Nn \l__trt_bc_anchor_iiix_fp {\trt@tmp@i}
    \fp_set:Nn \l__trt_bc_anchor_iiiy_fp {\trt@tmp@ii}
  }
```

(*End definition for* `\__trt_bc_initialize:nnn`.)

`bc3` The `bc3` coordinate system implementation. We will guard against using it when `\__trt_bc_initialize:nnn` is not yet executed – in that case, `uninitialized` error will be raised.

We will receive arguments of `bc3` as `#1,#2,#3`, so a simple parser is needed. All interesting things will be done with that parser.

```
\tikzdeclarecoordinatesystem {bc3}
  {
    \bool_if:NTF \l__trt_bc_initialized_bool
      {
        \__trt_bc_parse:w #1 \q_stop
      }
      {
        \msg_error:nn {triangletools} {uninitialized}
      }
  }
```

(*End definition for* `bc3`. *This function is documented on page 8.*)

`\__trt_bc_parse:w` This is the parser we use for `bc3`.

The conversion from λ_i to the Cartesian format is pretty simple, we have $x = \lambda_1 x_1 + \lambda_2 x_2 + \lambda_3 x_3$ and the formula for y is similar. However, first we have to change the value of λ_i so that $\lambda_1 + \lambda_2 + \lambda_3 = 1$.

```
\cs_new:Npn \__trt_bc_parse:w #1,#2,#3 \q_stop
  {
    \fp_set:Nn \l__trt_bc_tmp_fp { (#1) + (#2) + (#3) }
    \fp_set:Nn \l__trt_bc_lambda_i_fp { (#1) / (\l__trt_bc_tmp_fp) }
    \fp_set:Nn \l__trt_bc_lambda_ii_fp { (#2) / (\l__trt_bc_tmp_fp) }
    \fp_set:Nn \l__trt_bc_lambda_iii_fp { (#3) / (\l__trt_bc_tmp_fp) }
    \fp_set:Nn \l__trt_tmp_a_fp
      {
        \l__trt_bc_anchor_ix_fp * \l__trt_bc_lambda_i_fp +
        \l__trt_bc_anchor_iix_fp * \l__trt_bc_lambda_ii_fp +
        \l__trt_bc_anchor_iiix_fp * \l__trt_bc_lambda_iii_fp
      }
    \fp_set:Nn \l__trt_tmp_b_fp
      {
        \l__trt_bc_anchor_iy_fp * \l__trt_bc_lambda_i_fp +
        \l__trt_bc_anchor_iiy_fp * \l__trt_bc_lambda_ii_fp +
        \l__trt_bc_anchor_iiiy_fp * \l__trt_bc_lambda_iii_fp
      }
```

Floating point variables are not TeX dimensions, hence `\fp_to_dim:N` is used.

```
    \pgf@x = \fp_to_dim:N \l__trt_tmp_a_fp
    \pgf@y = \fp_to_dim:N \l__trt_tmp_b_fp
  }
⟨/barycentric⟩
```

(*End definition for* `\__trt_bc_parse:w`.)

4.3.3 Distance-finding utility

```
⟨*distance⟩
\ProvidesExplFile {trtdistance.code.tex} {2020/04/30} {0.1}
  {The ~ triangletools ~ package: ~ Utilities ~ for ~ 2d ~ distance}
```

This file implements functions to find the distance between (2d) Ti*k*Z coordinates.

\l__trt_d_pointi_x_fp
\l__trt_d_pointi_y_fp
\l__trt_d_pointii_x_fp
\l__trt_d_pointii_y_fp

These variables are used to store the coordinates of the points between which we are finding the distance.

```
\fp_new:N \l__trt_d_pointi_x_fp
\fp_new:N \l__trt_d_pointii_x_fp
\fp_new:N \l__trt_d_pointi_y_fp
\fp_new:N \l__trt_d_pointii_y_fp
```

(*End definition for* \l__trt_d_pointi_x_fp *and others.*)

\trt_distance:nnN

Find the distance between Ti*k*Z coordinates #1 and #2.

```
\cs_new:Npn \trt_distance:nnN #1 #2 #3
  {
    \pgfextractx {\trt@tmp@i} {\pgfpointanchor{#1}{center}}
    \pgfextracty {\trt@tmp@ii} {\pgfpointanchor{#1}{center}}
    \fp_set:Nn \l__trt_d_pointi_x_fp {\trt@tmp@i}
    \fp_set:Nn \l__trt_d_pointi_y_fp {\trt@tmp@ii}
    \pgfextractx {\trt@tmp@i} {\pgfpointanchor{#2}{center}}
    \pgfextracty {\trt@tmp@ii} {\pgfpointanchor{#2}{center}}
    \fp_set:Nn \l__trt_d_pointii_x_fp {\trt@tmp@i}
    \fp_set:Nn \l__trt_d_pointii_y_fp {\trt@tmp@ii}
    \fp_set:Nn #3
      {
        sqrt((
          (\l__trt_d_pointi_x_fp - \l__trt_d_pointii_x_fp) *
          (\l__trt_d_pointi_x_fp - \l__trt_d_pointii_x_fp)
        ) + (
          (\l__trt_d_pointi_y_fp - \l__trt_d_pointii_y_fp) *
          (\l__trt_d_pointi_y_fp - \l__trt_d_pointii_y_fp)
        ))
      }
  }
```

(*End definition for* \trt_distance:nnN*. This function is documented on page 9.*)

\trt_distance_triangle:nnnNNN

We mainly need the above function to find the side length in a triangle. Let's create a function that do so automatically.

```
\cs_new:Npn \trt_distance_triangle:nnnNNN #1 #2 #3 #4 #5 #6
  {
    \trt_distance:nnN {#2} {#3} #4
    \trt_distance:nnN {#3} {#1} #5
    \trt_distance:nnN {#1} {#2} #6
  }
⟨/distance⟩
```

(*End definition for* \trt_distance_triangle:nnnNNN*. This function is documented on page 9.*)

4.4 Construction of triangle centers

```
⟨*specialpoints⟩
\ProvidesExplFile {trtspecialpoints.code.tex} {2020/04/30} {0.1}
  {The ~ triangletools ~ package: ~ Triangle ~ center ~ construction}
```

This file will use the utility implemented in the above sections to find some most important triangle centers described in the ETC.

\l__trt_sp_a_fp
\l__trt_sp_b_fp
\l__trt_sp_c_fp

We will need the side length of the triangle for some centers.

```
\fp_new:N \l__trt_sp_a_fp
\fp_new:N \l__trt_sp_b_fp
\fp_new:N \l__trt_sp_c_fp
```

(*End definition for* `\l__trt_sp_a_fp`, `\l__trt_sp_b_fp`, *and* `\l__trt_sp_c_fp`.)

\l__trt_sp_coordinatei_x_fp
\l__trt_sp_coordinatei_y_fp
\l__trt_sp_coordinateii_x_fp
\l__trt_sp_coordinateii_y_fp
\l__trt_sp_coordinateiii_x_fp
\l__trt_sp_coordinateiii_y_fp
\l__trt_sp_linei_a_fp
\l__trt_sp_linei_b_fp
\l__trt_sp_linei_c_fp
\l__trt_sp_lineii_a_fp
\l__trt_sp_lineii_b_fp
\l__trt_sp_lineii_c_fp

These variables may also be helpful for triangle centers for which a simple formula doesn't exist, e.g. the circumcenter.

```
\fp_new:N \l__trt_sp_coordinatei_x_fp
\fp_new:N \l__trt_sp_coordinatei_y_fp
\fp_new:N \l__trt_sp_coordinateii_x_fp
\fp_new:N \l__trt_sp_coordinateii_y_fp
\fp_new:N \l__trt_sp_coordinateiii_x_fp
\fp_new:N \l__trt_sp_coordinateiii_y_fp
\fp_new:N \l__trt_sp_linei_a_fp
\fp_new:N \l__trt_sp_linei_b_fp
\fp_new:N \l__trt_sp_linei_c_fp
\fp_new:N \l__trt_sp_lineii_a_fp
\fp_new:N \l__trt_sp_lineii_b_fp
\fp_new:N \l__trt_sp_lineii_c_fp
```

(*End definition for* `\l__trt_sp_coordinatei_x_fp` *and others.*)

\l__trt_sp_tmpa_fp
\l__trt_sp_tmpb_fp
\l__trt_sp_tmpc_fp

Some additional temporary variables.

```
\fp_new:N \l__trt_sp_tmpa_fp
\fp_new:N \l__trt_sp_tmpb_fp
\fp_new:N \l__trt_sp_tmpc_fp
```

(*End definition for* `\l__trt_sp_tmpa_fp`, `\l__trt_sp_tmpb_fp`, *and* `\l__trt_sp_tmpc_fp`.)

4.4.1 X_1 – The incenter

Each center will have a function taking four arguments. The first three arguments are the Ti*k*Z coordinates of the triangle vertices; the last argument is the name of the return Ti*k*Z coordinate.

To prevent conflict between these sister functions when they are used together, I put each of them inside a TeX group.

\trt_sp_incenter:nnnn

Return the incenter. It is based on the barycentric coordinate of the incenter, (a, b, c).

```
\cs_new:Npn \trt_sp_incenter:nnnn #1 #2 #3 #4
  {
    \group_begin:
      \__trt_bc_initialize:nnn {#1} {#2} {#3}
      \trt_distance_triangle:nnnNNN {#1} {#2} {#3}
        \l__trt_sp_a_fp \l__trt_sp_b_fp \l__trt_sp_c_fp
      \path[overlay] (bc3 ~ cs \c_colon_str
        \fp_eval:n {\l__trt_sp_a_fp},
        \fp_eval:n {\l__trt_sp_b_fp},
        \fp_eval:n {\l__trt_sp_c_fp}) coordinate (#4);
    \group_end:
  }
```

(*End definition for* `\trt_sp_incenter:nnnn`. *This function is documented on page 3.*)

\trt_sp_excenter:nnnn

Return the excenter of the triangle, with respect to vertex `#1`. This center is just a derivation of the incenter; also it is not unique, so it is not assigned a number.

Barycentric coordinate of the excenter is $(-a, b, c)$, where a is the length of the side joining #2 and #3.

Note that this is the only function in this series in which argument order is important.

```
\cs_new:Npn \trt_sp_excenter:nnnn #1 #2 #3 #4
  {
    \group_begin:
      \__trt_bc_initialize:nnn {#1} {#2} {#3}
      \trt_distance_triangle:nnnNNN {#1} {#2} {#3}
        \l__trt_sp_a_fp \l__trt_sp_b_fp \l__trt_sp_c_fp
      \path[overlay] (bc3 ~ cs \c_colon_str
        \fp_eval:n {- \l__trt_sp_a_fp},
        \fp_eval:n {\l__trt_sp_b_fp},
        \fp_eval:n {\l__trt_sp_c_fp}) coordinate (#4);
    \group_end:
  }
```

(*End definition for* `\trt_sp_excenter:nnnn`. *This function is documented on page 3.*)

4.4.2 X_2 – The centroid

`\trt_sp_centroid:nnnn` This is perhaps the simplest of all. Barycentric coordinate: $(1, 1, 1)$.

```
\cs_new:Npn \trt_sp_centroid:nnnn #1 #2 #3 #4
  {
    \group_begin:
      \__trt_bc_initialize:nnn {#1} {#2} {#3}
      \path[overlay] (bc3 ~ cs \c_colon_str 1, 1, 1) coordinate (#4);
    \group_end:
  }
```

(*End definition for* `\trt_sp_centroid:nnnn`. *This function is documented on page 3.*)

4.4.3 X_3 – The circumcenter

`\trt_sp_circumcenter:nnnn` This is opposite to X_2: perhaps this is the most complex of all. The barycentric coordinate formula is not simple enough for me, so I construct this point purely manually: find the intersection of the perpendicular bisectors.

```
\cs_new:Npn \trt_sp_circumcenter:nnnn #1 #2 #3 #4
  {
    \group_begin:
```

Firstly, let's store the coordinate of the vertices.

```
      \pgfextractx {\trt@tmp@i} {\pgfpointanchor{#1}{center}}
      \pgfextracty {\trt@tmp@ii} {\pgfpointanchor{#1}{center}}
      \fp_set:Nn \l__trt_sp_coordinatei_x_fp {\trt@tmp@i}
      \fp_set:Nn \l__trt_sp_coordinatei_y_fp {\trt@tmp@ii}
      \pgfextractx {\trt@tmp@i} {\pgfpointanchor{#2}{center}}
      \pgfextracty {\trt@tmp@ii} {\pgfpointanchor{#2}{center}}
      \fp_set:Nn \l__trt_sp_coordinateii_x_fp {\trt@tmp@i}
      \fp_set:Nn \l__trt_sp_coordinateii_y_fp {\trt@tmp@ii}
      \pgfextractx {\trt@tmp@i} {\pgfpointanchor{#3}{center}}
      \pgfextracty {\trt@tmp@ii} {\pgfpointanchor{#3}{center}}
      \fp_set:Nn \l__trt_sp_coordinateiii_x_fp {\trt@tmp@i}
      \fp_set:Nn \l__trt_sp_coordinateiii_y_fp {\trt@tmp@ii}
```

Now, let's change point #2 to the midpoint between #1 and #2, and do the same for #3.

```
      \fp_set:Nn \l__trt_sp_coordinateii_x_fp
```

```
          {
            (\l__trt_sp_coordinatei_x_fp + \l__trt_sp_coordinateii_x_fp) / 2
          }
        \fp_set:Nn \l__trt_sp_coordinateii_y_fp
          {
            (\l__trt_sp_coordinatei_y_fp + \l__trt_sp_coordinateii_y_fp) / 2
          }
        \coordinate[overlay] (trt@tmp@ii) at (
          \fp_to_dim:N \l__trt_sp_coordinateii_x_fp,
          \fp_to_dim:N \l__trt_sp_coordinateii_y_fp);
        \fp_set:Nn \l__trt_sp_coordinateiii_x_fp
          {
            (\l__trt_sp_coordinatei_x_fp + \l__trt_sp_coordinateiii_x_fp) / 2
          }
        \fp_set:Nn \l__trt_sp_coordinateiii_y_fp
          {
            (\l__trt_sp_coordinatei_y_fp + \l__trt_sp_coordinateiii_y_fp) / 2
          }
        \coordinate[overlay] (trt@tmp@iii) at (
          \fp_to_dim:N \l__trt_sp_coordinateiii_x_fp,
          \fp_to_dim:N \l__trt_sp_coordinateiii_y_fp);
```

All we have to do now is to find the equations of the bisectors and their intersection.

```
        \trt_lt_get_line_equation:nnNNN {#1} {#2}
          \l__trt_sp_tmpa_fp \l__trt_sp_tmpb_fp \l__trt_sp_tmpc_fp
        \trt_lt_get_perpendicular_equation:nNNNNNN {trt@tmp@ii}
          \l__trt_sp_tmpa_fp \l__trt_sp_tmpb_fp \l__trt_sp_tmpc_fp
          \l__trt_sp_linei_a_fp \l__trt_sp_linei_b_fp \l__trt_sp_linei_c_fp
        \trt_lt_get_line_equation:nnNNN {#1} {#3}
          \l__trt_sp_tmpa_fp \l__trt_sp_tmpb_fp \l__trt_sp_tmpc_fp
        \trt_lt_get_perpendicular_equation:nNNNNNN {trt@tmp@iii}
          \l__trt_sp_tmpa_fp \l__trt_sp_tmpb_fp \l__trt_sp_tmpc_fp
          \l__trt_sp_lineii_a_fp \l__trt_sp_lineii_b_fp \l__trt_sp_lineii_c_fp
        \trt_lt_get_intersection_line:NNNNNNNN
          \l__trt_sp_linei_a_fp \l__trt_sp_linei_b_fp \l__trt_sp_linei_c_fp
          \l__trt_sp_lineii_a_fp \l__trt_sp_lineii_b_fp \l__trt_sp_lineii_c_fp
          \l__trt_sp_tmpa_fp \l__trt_sp_tmpb_fp
        \coordinate[overlay] (#4) at (
          \fp_to_dim:N \l__trt_sp_tmpa_fp, \fp_to_dim:N \l__trt_sp_tmpb_fp);
      \group_end:
    }
```

Quite surprisingly, the function is still very fast after all this. On my machine it never exceeds 10ms in execution time.

(*End definition for* `\trt_sp_circumcenter:nnnn`. *This function is documented on page 3.*)

4.4.4 X_4 – The orthocenter

`\trt_sp_orthocenter:nnnn` Return the orthocenter of the triangle. This point is also constructed manually instead of using a proved formula. However, the utilities help making the construction look very simple.

```
\cs_new:Npn \trt_sp_orthocenter:nnnn #1 #2 #3 #4
  {
    \group_begin:
      \__trt_lt_return_perpendicular_coordinate:nnnn {#1} {#2} {#3} {trt@tmp@i}
      \__trt_lt_return_perpendicular_coordinate:nnnn {#2} {#1} {#3} {trt@tmp@ii}
      \__trt_lt_return_intersection:nnnnn
        {#1} {trt@tmp@i} {#2} {trt@tmp@ii} {#4}
    \group_end:
  }
```

(*End definition for* `\trt_sp_orthocenter:nnnn`. *This function is documented on page 4.*)

4.4.5 X_5 – The nine-point center

\trt_sp_ninepointcenter:nnnn Return the center of the nine-point circle.

```
\cs_new:Npn \trt_sp_ninepointcenter:nnnn #1 #2 #3 #4
  {
    \group_begin:
```

X_5 is is the midpoint of X_3 and X_4. Therefore, for simplicity, X_3 and X_4 are constructed first. This causes some run-time overhead, however the overall execution time is still below 15ms, which is, in my opinion, still good.

Note that we already used `trt@tmp@i` and `trt@tmp@ii` coordinates in the construction of X_3 and X_4, so to prevent conflict, `trt@tmp@iii` and `trt@tmp@iv` are used.

```
      \trt_sp_circumcenter:nnnn {#1} {#2} {#3} {trt@tmp@iii}
      \trt_sp_orthocenter:nnnn {#1} {#2} {#3} {trt@tmp@iv}
      \pgfextractx {\trt@tmp@i} {\pgfpointanchor{trt@tmp@iii}{center}}
      \pgfextracty {\trt@tmp@ii} {\pgfpointanchor{trt@tmp@iii}{center}}
      \fp_set:Nn \l__trt_sp_tmpa_fp {\trt@tmp@i}
      \fp_set:Nn \l__trt_sp_tmpb_fp {\trt@tmp@ii}
      \pgfextractx {\trt@tmp@i} {\pgfpointanchor{trt@tmp@iv}{center}}
      \pgfextracty {\trt@tmp@ii} {\pgfpointanchor{trt@tmp@iv}{center}}
      \fp_set:Nn \l__trt_sp_tmpa_fp { (\trt@tmp@i + \l__trt_sp_tmpa_fp) / 2 }
      \fp_set:Nn \l__trt_sp_tmpb_fp { (\trt@tmp@ii + \l__trt_sp_tmpb_fp) / 2 }
      \coordinate[overlay] (#4) at (\fp_to_dim:N \l__trt_sp_tmpa_fp,
        \fp_to_dim:N \l__trt_sp_tmpb_fp);
    \group_end:
  }
```

(*End definition for* `\trt_sp_ninepointcenter:nnnn`. *This function is documented on page 4.*)

4.4.6 X_6 – The symmedian point

\trt_sp_symmedian:nnnn Return the symmedian point (*aka.* the Lemoine point or Grebe point). The barycentric coordinate of the point is (a^2, b^2, c^2).

```
\cs_new:Npn \trt_sp_symmedian:nnnn #1 #2 #3 #4
  {
    \group_begin:
      \__trt_bc_initialize:nnn {#1} {#2} {#3}
      \trt_distance_triangle:nnnNNN {#1} {#2} {#3}
        \l__trt_sp_a_fp \l__trt_sp_b_fp \l__trt_sp_c_fp
      \path[overlay] (bc3 ~ cs \c_colon_str
        \fp_eval:n {\l__trt_sp_a_fp * \l__trt_sp_a_fp},
        \fp_eval:n {\l__trt_sp_b_fp * \l__trt_sp_b_fp},
        \fp_eval:n {\l__trt_sp_c_fp * \l__trt_sp_c_fp}) coordinate (#4);
    \group_end:
  }
```

(*End definition for* `\trt_sp_symmedian:nnnn`. *This function is documented on page 4.*)

4.4.7 X_7 – The Gergonne point

\trt_sp_gergonne:nnnn Return the Gergonne point of the triangle. The barycentric coordinate of the point is $(\frac{1}{b+c-a}, \frac{1}{c+a-b}, \frac{1}{a+b-c})$.

```
\cs_new:Npn \trt_sp_gergonne:nnnn #1 #2 #3 #4
  {
    \group_begin:
      \__trt_bc_initialize:nnn {#1} {#2} {#3}
      \trt_distance_triangle:nnnNNN {#1} {#2} {#3}
        \l__trt_sp_a_fp \l__trt_sp_b_fp \l__trt_sp_c_fp
      \path[overlay] (bc3 ~ cs \c_colon_str
        \fp_eval:n { 1/(\l__trt_sp_b_fp + \l__trt_sp_c_fp - \l__trt_sp_a_fp) },
        \fp_eval:n { 1/(\l__trt_sp_c_fp + \l__trt_sp_a_fp - \l__trt_sp_b_fp) },
        \fp_eval:n { 1/(\l__trt_sp_a_fp + \l__trt_sp_b_fp - \l__trt_sp_c_fp) }
      ) coordinate (#4);
    \group_end:
  }
```

(*End definition for* `\trt_sp_gergonne:nnnn`*. This function is documented on page 4.*)

4.4.8 X_8 – The Nagel point

`\trt_sp_nagel:nnnn` Return the Nagel point. The barycentric coordinate of the point is $(b+c-a, c+a-b, a+b-c)$.

```
\cs_new:Npn \trt_sp_nagel:nnnn #1 #2 #3 #4
  {
    \group_begin:
      \__trt_bc_initialize:nnn {#1} {#2} {#3}
      \trt_distance_triangle:nnnNNN {#1} {#2} {#3}
        \l__trt_sp_a_fp \l__trt_sp_b_fp \l__trt_sp_c_fp
      \path[overlay] (bc3 ~ cs \c_colon_str
        \fp_eval:n { \l__trt_sp_b_fp + \l__trt_sp_c_fp - \l__trt_sp_a_fp },
        \fp_eval:n { \l__trt_sp_c_fp + \l__trt_sp_a_fp - \l__trt_sp_b_fp },
        \fp_eval:n { \l__trt_sp_a_fp + \l__trt_sp_b_fp - \l__trt_sp_c_fp }
      ) coordinate (#4);
    \group_end:
  }
```

(*End definition for* `\trt_sp_nagel:nnnn`*. This function is documented on page 5.*)

4.4.9 X_9 – The *mittenpunkt*

`\trt_sp_mittenpunkt:nnnn` Return the *mittenpunkt* of the triangle – its barycentric coordinate is $(a\times(b+c-a), b\times(c+a-b), c\times(a+b-c))$.

```
\cs_new:Npn \trt_sp_mittenpunkt:nnnn #1 #2 #3 #4
  {
    \group_begin:
      \__trt_bc_initialize:nnn {#1} {#2} {#3}
      \trt_distance_triangle:nnnNNN {#1} {#2} {#3}
        \l__trt_sp_a_fp \l__trt_sp_b_fp \l__trt_sp_c_fp
      \path[overlay] (bc3 ~ cs \c_colon_str
        \fp_eval:n
          {
            \l__trt_sp_a_fp * (
              \l__trt_sp_b_fp + \l__trt_sp_c_fp - \l__trt_sp_a_fp
            )
          },
        \fp_eval:n
          {
            \l__trt_sp_b_fp * (
              \l__trt_sp_c_fp + \l__trt_sp_a_fp - \l__trt_sp_b_fp
            )
          },
        \fp_eval:n
          {
            \l__trt_sp_c_fp * (
              \l__trt_sp_a_fp + \l__trt_sp_b_fp - \l__trt_sp_c_fp
            )
          }
      ) coordinate (#4);
    \group_end:
  }
```

(*End definition for* `\trt_sp_mittenpunkt:nnnn`. *This function is documented on page 5.*)

4.4.10 X_{10} – The Spieker point

`\trt_sp_spieker:nnnn` Return the Spieker point. The barycentric coordinate of the point is $(b+c, c+a, a+b)$.

```
\cs_new:Npn \trt_sp_spieker:nnnn #1 #2 #3 #4
  {
    \group_begin:
      \__trt_bc_initialize:nnn {#1} {#2} {#3}
      \trt_distance_triangle:nnnNNN {#1} {#2} {#3}
        \l__trt_sp_a_fp \l__trt_sp_b_fp \l__trt_sp_c_fp
      \path[overlay] (bc3 ~ cs \c_colon_str
        \fp_eval:n { \l__trt_sp_b_fp + \l__trt_sp_c_fp },
        \fp_eval:n { \l__trt_sp_c_fp + \l__trt_sp_a_fp },
        \fp_eval:n { \l__trt_sp_a_fp + \l__trt_sp_b_fp }
      ) coordinate (#4);
    \group_end:
  }
⟨/specialpoints⟩
```

(*End definition for* `\trt_sp_spieker:nnnn`. *This function is documented on page 5.*)

4.5 The frontend layer

```
⟨*frontend⟩
\ProvidesExplFile {trtfrontend.code.tex} {2020/04/30} {0.1}
  {The ~ triangletools ~ package: ~ The ~ front-end ~ layer}
```

The user interface of the package, which consists solely of pgf keys, will be implemented in this file.

`\l__trt_fr_output_name_tl` Store the name of the output coordinate. Default to `trt␣output`.

```
\tl_new:N \l__trt_fr_output_name_tl
\tl_set:Nn \l__trt_fr_output_name_tl {trt ~ output}
```

(*End definition for* `\l__trt_fr_output_name_tl`.)

`\l__trt_fr_center_number_int` We only provide specific key for X_1, X_2, X_3 and X_4. All other points can be referenced using a single generic key. We need to store the index of that point so that we can choose the right function for the point.

```
\int_new:N \l__trt_fr_center_number_int
```

(*End definition for* `\l__trt_fr_center_number_int`.)

`\trtradius` `\l__trt_fr_radius_fp` This macro will store the radius if a circle is associated. Of course firstly we need a floating point variable specified for that purpose.

```
\fp_new:N \l__trt_fr_radius_fp
\cs_gset_nopar:Npn \trtradius
  {
    \msg_error:nn {triangletools} {no-radius-found} 0pt
  }
```

(*End definition for* `\trtradius` *and* `\l__trt_fr_radius_fp`. *This function is documented on page 2.*)

`/tikz/trt` Now it's time for the keys. They will be stored under `/tikz/triangletools` and can be accessed at `trt={⟨keys⟩}`.

```
\tikzset {
  triangletools/.is ~ family,
  trt/.code={\pgfkeys{/tikz/triangletools/.cd,#1}},
  triangletools/.cd,
```

(*End definition for* `/tikz/trt`. *This function is documented on page 2.*)

`output␣name` Change the output name of all returned coordinates.

```
output ~ name/.code={
  \tl_set:Nn \l__trt_fr_output_name_tl {#1}
},
```

(*End definition for* `output name`. *This function is documented on page 9.*)

`intersection` `foot␣of␣perpendicular` The front-end of the line tools utility.

```
intersection/.code ~ args={(#1)(#2)--(#3)(#4)}{
  \__trt_lt_return_intersection:nnnnn {#1} {#2} {#3} {#4}
    {\tl_use:N \l__trt_fr_output_name_tl}
},
foot ~ of ~ perpendicular/.code ~ args={(#1)--(#2)(#3)}{
  \__trt_lt_return_perpendicular_coordinate:nnnn {#1} {#2} {#3}
    {\tl_use:N \l__trt_fr_output_name_tl}
},
```

(*End definition for* `intersection` *and* `foot of perpendicular`. *These functions are documented on page 6.*)

`initialize␣barycentric` The front-end of the barycentric coordinate system.

```
initialize ~ barycentric/.code ~ args={(#1)(#2)(#3)}{
  \__trt_bc_initialize:nnn {#1} {#2} {#3}
},
```

(*End definition for* `initialize barycentric`. *This function is documented on page 8.*)

`incenter` `excenter` `centroid` `circumcenter` `orthocenter` The front-end of the triangle centers X_1 to X_4 and the excenter.

```
incenter/.code ~ args={(#1)(#2)(#3)}{
  \trt_sp_incenter:nnnn {#1} {#2} {#3} {trt@tmp@center}
  \pgfkeysalso{foot ~ of ~ perpendicular=(trt@tmp@center)--(#1)(#2)}
  \trt_distance:nnN {\tl_use:N \l__trt_fr_output_name_tl} {trt@tmp@center}
    \l__trt_fr_radius_fp
  \cs_gset_nopar:Npx \trtradius { \fp_to_dim:N \l__trt_fr_radius_fp }
  \coordinate (\tl_use:N \l__trt_fr_output_name_tl) at (trt@tmp@center);
},
excenter/.code ~ args={(#1)(#2)(#3)}{
  \trt_sp_excenter:nnnn {#1} {#2} {#3} {trt@tmp@center}
  \pgfkeysalso{foot ~ of ~ perpendicular=(trt@tmp@center)--(#2)(#3)}
  \trt_distance:nnN {\tl_use:N \l__trt_fr_output_name_tl} {trt@tmp@center}
    \l__trt_fr_radius_fp
  \cs_gset_nopar:Npx \trtradius { \fp_to_dim:N \l__trt_fr_radius_fp }
  \coordinate (\tl_use:N \l__trt_fr_output_name_tl) at (trt@tmp@center);
},
centroid/.code ~ args={(#1)(#2)(#3)}{
  \trt_sp_centroid:nnnn {#1} {#2} {#3}
    {\tl_use:N \l__trt_fr_output_name_tl}
},
circumcenter/.code ~ args={(#1)(#2)(#3)}{
  \trt_sp_circumcenter:nnnn {#1} {#2} {#3}
    {\tl_use:N \l__trt_fr_output_name_tl}
  \trt_distance:nnN {\tl_use:N \l__trt_fr_output_name_tl} {#1}
    \l__trt_fr_radius_fp
  \cs_gset_nopar:Npx \trtradius { \fp_to_dim:N \l__trt_fr_radius_fp }
},
orthocenter/.code ~ args={(#1)(#2)(#3)}{
  \trt_sp_orthocenter:nnnn {#1} {#2} {#3}
    {\tl_use:N \l__trt_fr_output_name_tl}
},
```

(*End definition for* `incenter` *and others. These functions are documented on page 3.*)

`triangle␣center` This key is used to access all centers. I don't give any centers from X_5 a key – this key is necessary to construct them.

```
triangle ~ center/.code ~ args={(#1)(#2)(#3)(#4)}{
  \int_case:nnF {#4}
    {
      {1} {
        \pgfkeysalso{incenter=(#1)(#2)(#3)}
      }
      {2} {
        \pgfkeysalso{centroid=(#1)(#2)(#3)}
      }
      {3} {
        \pgfkeysalso{circumcenter=(#1)(#2)(#3)}
      }
      {4} {
        \pgfkeysalso{orthocenter=(#1)(#2)(#3)}
      }
      {5} {
        \trt_sp_ninepointcenter:nnnn {#1} {#2} {#3}
          {\tl_use:N \l__trt_fr_output_name_tl}
        \group_begin:
          \__trt_bc_initialize:nnn {#1} {#2} {#3}
          \coordinate (trt@tmp@mid) at (bc3 ~ cs \c_colon_str 1,1,0);
        \group_end:
        \trt_distance:nnN {\tl_use:N \l__trt_fr_output_name_tl} {trt@tmp@mid}
          \l__trt_fr_radius_fp
        \cs_gset_nopar:Npx \trtradius { \fp_to_dim:N \l__trt_fr_radius_fp }
      }
      {6} {
        \trt_sp_symmedian:nnnn {#1} {#2} {#3}
          {\tl_use:N \l__trt_fr_output_name_tl}
      }
      {7} {
        \trt_sp_gergonne:nnnn {#1} {#2} {#3}
          {\tl_use:N \l__trt_fr_output_name_tl}
      }
      {8} {
        \trt_sp_nagel:nnnn {#1} {#2} {#3}
          {\tl_use:N \l__trt_fr_output_name_tl}
      }
      {9} {
        \trt_sp_mittenpunkt:nnnn {#1} {#2} {#3}
          {\tl_use:N \l__trt_fr_output_name_tl}
      }
      {10} {
        \trt_sp_spieker:nnnn {#1} {#2} {#3} {trt@tmp@center}
        \group_begin:
          \__trt_bc_initialize:nnn {#1} {#2} {#3}
          \coordinate (trt@tmp@midi)  at (bc3 ~ cs \c_colon_str 1,1,0);
          \coordinate (trt@tmp@midii) at (bc3 ~ cs \c_colon_str 0,1,1);
        \group_end:
        \pgfkeysalso{
          foot~of~perpendicular=(trt@tmp@center)--(trt@tmp@midi)(trt@tmp@midii)
        }
        \trt_distance:nnN {\tl_use:N \l__trt_fr_output_name_tl} {trt@tmp@center}
          \l__trt_fr_radius_fp
        \cs_gset_nopar:Npx \trtradius { \fp_to_dim:N \l__trt_fr_radius_fp }
        \coordinate (\tl_use:N \l__trt_fr_output_name_tl) at (trt@tmp@center);
      }
    }
    {
        \msg_error:nnn {triangletools} {center-not-found} {#4}
      }
  }
}
⟨/frontend⟩
```

(*End definition for* `triangle center`*. This function is documented on page 4.*)

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.