pt

# Ti*k*Z-MEC

v.0.5a – 2012/10/21

Francesco Clemente*

## 1 Introduction

This library widens the application field of the Ti*k*Z package defining some new *shapes* used in structural mechanics' two-dimensional diagrams, useful in the description of mechanical problems related to (internally and externaly) loaded structures. The library defines the shapes of various *constraints* used in the above mentioned diagrams, providing a Ti*k*Z-user-friendly interface for their usage.

If similar features are provided by other libraries or packages, please let me know that through an e-mail to the address specified at the bottom of the page. Advices, corrections and demans are welcomed too. I'll try to answer as soon as possible.

### 1.1 License



### 1.2 Installation

Since the library is currently in development and that it can go towards huge changes, it is better to install it simply copying the file `tikzlibrarymec.code.tex` in the working directory, using it for single documents. The above mentioned file is generated compiling the file `tikz-mec.ins` with `latex`. To generate this documentation file in .pdf format, you need to compile the file `tikz-mec.dtx` with `pdflatex`.

### 1.3 Loading the library

Once installed, to be able to use the Ti*k*Z-MEC features just add in the preamble of your document the following line:

```
\usetikzlibrary{mec}
```

Ti*k*Z-MEC is based on Ti*k*Z and uses the `decorations.markings` library, then it is necessary to load both the package and the library too.

*e-mail: `francesco dot clemente00 at gmail dot com`

## 2 Shapes

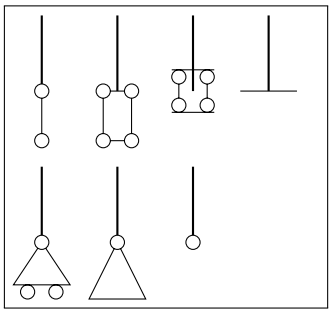
The library defines the following constraints:

- Support (`support`): 
- Hinge A (`hinge`): 
- Hinge B (`hinge b`): 
- Pendulum (`pendulum`): 
- Double pendulum A (`double pendulum`): 
- Double pendulum B (`double pendulum b`): 
- Fixed support (`fixed support`): 
- Ground (`ground`): 

## 3 Usage examples

In this section we will see some code examples which show the Ti*k*Z-MEC features. As we will see, all the defined constraints are *node shapes*. So drawing a constraint is the same as drawing a Ti*k*Z node, with all advantages and disadvantages (discussed in the Ti*k*Z documentation) that this implies.

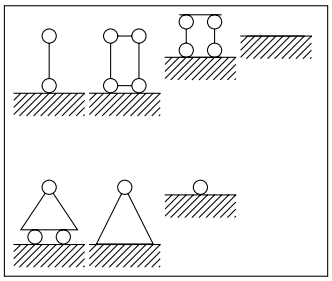
All of the defined constraints connected to a rod which shows their default anchor points:



```
\begin{tikzpicture}
  \node[draw,support](A) at (0,0){};
  \draw[thick] (A) -- +(0,1);
  \node[draw,hinge](A) at (1,0){};
  \draw[thick] (A) -- +(0,1);
  \node[draw,hinge b](A) at (2,0){};
  \draw[thick] (A) -- +(0,1);
  \node[draw,pendulum](A) at (0,2){};
  \draw[thick] (A) -- +(0,1);
  \node[draw,double pendulum](A) at (1,2){};
  \draw[thick] (A) -- +(0,1);
  \node[draw,double pendulum b](A) at (2,2){};
  \draw[thick] (A) -- +(0,1);
  \node[draw,fixed support](A) at (3,2){};
  \draw[thick] (A) -- +(0,1);
\end{tikzpicture}
```

The previous example shows that the default anchor point of the double pendulum version B is at the center of the shape. This leads to a bad graphic result. A future example will show the correct usage of the constraint which takes advantage of its `north` and `south` anchors.

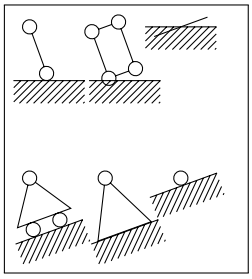
Grounded constraints:



```
\begin{tikzpicture}
  \node[draw,support,grounded] (A) at (0,0){};
  \node[draw,hinge,grounded] (B) at (1,0){};
  \node[draw,hinge b,grounded] (C) at (2,0){};
  \node[draw,pendulum,grounded] (D) at (0,2){};
  \node[draw,double pendulum,grounded] (E) at (1,2){};
  \node[draw,double pendulum b,grounded] (F) at (2,2){};
  \node[draw,fixed support,grounded] (G) at (3,2){};
\end{tikzpicture}
```

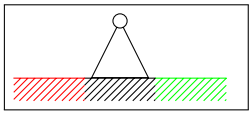
Even if it is possible to draw a ground at the bottom of the double pendulum B like any other shape, it is better to avoid that because it is an *internal* (to the structure) constraint. The equivalent external constraint (from the mechanics point of view) is the double pendulum.

When you need to *rotate* a grounded constraint you don't have to use the standard `rotate` key, but you should specify the rotation angle to the `grounded` key. Doing so you will be able to rotate both the constraint and the ground:



```
\begin{tikzpicture}
  % right
  \node[draw,support,grounded=20] (A) at (0,0){};
  \node[draw,hinge,grounded=20] (B) at (1,0){};
  \node[draw,hinge b,grounded=20] (C) at (2,0){};
  % wrong
  \node[draw,pendulum,grounded,rotate=20] (D) at (0,2){};
  \node[draw,double pendulum,grounded,rotate=20] (E) at (1,2){};
  \node[draw,fixed support,grounded,rotate=20] (F) at (2,2){};
\end{tikzpicture}
```

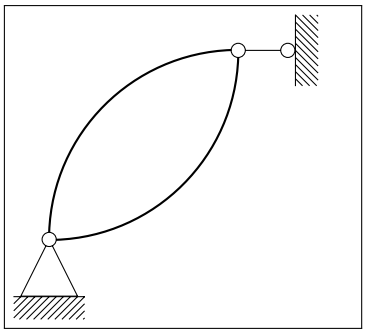
When you define a grounded constraint, the ground is automatically labeled in the form ⟨*node label*⟩`ground` where ⟨*node label*⟩ is the label given to the constraint on which the `ground` is connected. Thanks to that it is possible to refer external objects to the ground through the anchors shown in section **??**[1]. Of course, if you don't define a name for the node, you'll not be able to use the ground anchors:



```
\begin{tikzpicture}
  \coordinate (a) at (0,0);
  \node[draw,hinge,grounded] (A) at (a){};
  \node[draw,ground,red,anchor=east] (l) at (Aground.west){};
  \node[draw,ground,green,anchor=west] (r) at (Aground.east){};
\end{tikzpicture}
```

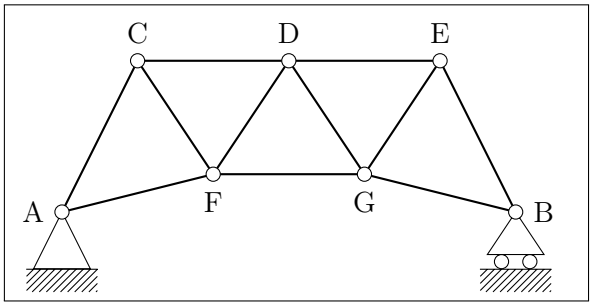
When you need to draw a diagram in which are present curved rods, you *shouldn't* refer to the nodes ending with a circle (i.e. all of them excluding the fixed support and both the versions of the double pendulum) using the usual way. You should instead consider the real starting point and set the arc starting angle accordingly (with default dimensions I suggest to correct the angle by 2°). It is possible to use the same syntax used in the standard *circle* shape:

[1] The author wishes to thank Claudio Fiandrino for the suggestion of the feature and for helping in the code implementation



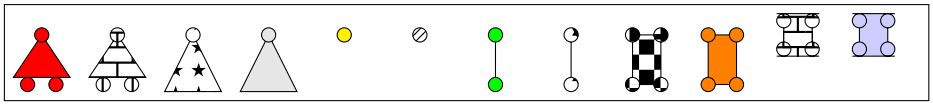
```
\begin{tikzpicture}
  \coordinate (a) at (0,0);
  \coordinate (b) at (2.5,2.5);
  \node[draw,hinge,grounded] (A) at (a){};
  \node[draw,pendulum,grounded=90] (B) at (b){};
  \draw[thick] (A.90) arc[radius=2.5,%
    start angle=178, end angle=92];
  \draw[thick] (B.180) arc[radius=2.5,%
    start angle=358, end angle=272];
\end{tikzpicture}
```

A complete structure which shows where could be useful the B version of the hinge:



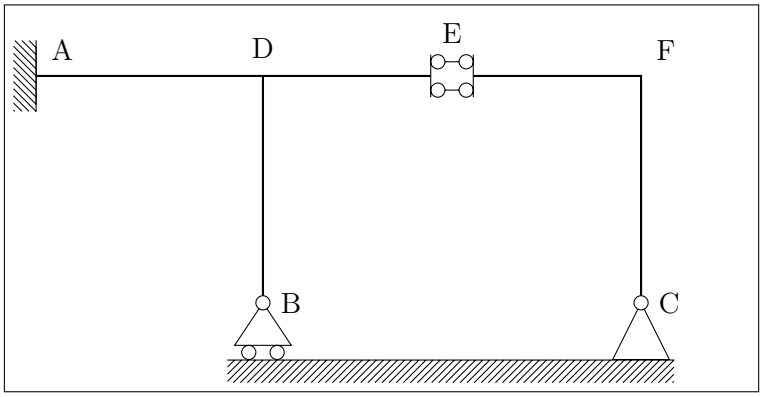

```
\usetikzlibrary{positioning}
\begin{tikzpicture}[node distance=1mm]
  \coordinate (a) at (0,0) node[left=of a] {A};
  \coordinate (b) at (6,0) node[right=of b] {B};
  \coordinate (c) at (1,2) node[above=of c] {C};
  \coordinate (d) at (3,2) node[above=of d] {D};
  \coordinate (e) at (5,2) node[above=of e] {E};
  \coordinate (f) at (2,0.5) node[below=of f] {F};
  \coordinate (g) at (4,0.5) node[below=of g] {G};
  \node[hinge,draw,grounded] (A) at (a){};
  \node[support,draw,grounded] (B) at (b){};
  \node[hinge b,draw] (C) at (c){};
  \node[hinge b,draw] (D) at (d){};
  \node[hinge b,draw] (E) at (e){};
  \node[hinge b,draw] (F) at (f){};
  \node[hinge b,draw] (G) at (g){};
  \draw[thick] (A) -- (C) -- (F) -- (D) -- (G) -- (E) -- (B) -- (G)
    -- (F) -- (A) (C) -- (D) -- (E);
\end{tikzpicture}
```

An example showing the possibility to use Ti*k*Z fill and pattern operations on the various constraints:



```
\usetikzlibrary{patterns}
\begin{tikzpicture}
  \node[draw,support,fill=red] (A) at (0,0){};
  \node[draw,support,pattern=bricks] (B) at (1,0){};
  \node[draw,hinge,pattern=fivepointed stars] (C) at (2,0){};
  \node[draw,hinge,fill=gray!20] (D) at (3,0){};
  \node[draw,hinge b,fill=yellow] (E) at (4,0){};
  \node[draw,hinge b,pattern=north east lines] (F) at (5,0){};
  \node[draw,pendulum,fill=green] (G) at (6,0){};
  \node[draw,pendulum,pattern=sixpointed stars] (G) at (7,0){};
  \node[draw,double pendulum,pattern=checkerboard] (H) at (8,0){};
  \node[draw,double pendulum,fill=orange] (I) at (9,0){};
  \node[draw,double pendulum b,pattern=bricks] (J) at (10,0){};
  \node[draw,double pendulum b,fill=blue!20] (K) at (11,0){};
\end{tikzpicture}
```

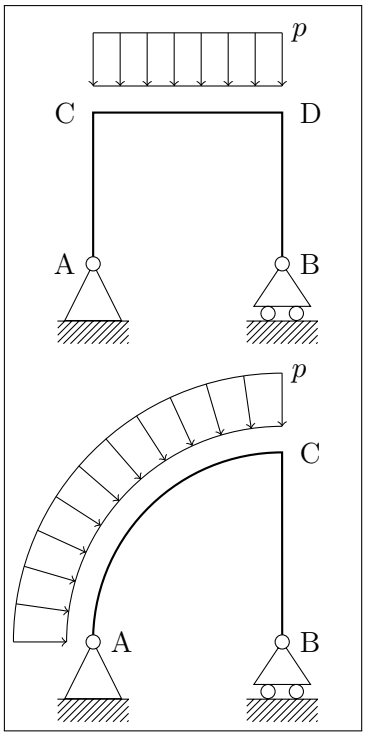
A diagram in which the `double pend b` is correctly used. The key `stretch factor` is also used: it allows to lenghten the ground (the width is not modified by the key). As you can see from the example below, the use of the `stretch factor` key requires to define the ground separately from the constraint. The `left` and `right` anchors come in help when you need to refer a stretched ground to a particular constraint (see section **??**).




```
\usetikzlibrary{positioning}
\begin{tikzpicture}[node distance=1mm]
  \coordinate (a) at (0,3) node[above right=of a] {A};
  \coordinate (b) at (3,0) node[right=of b] {B};
  \coordinate (c) at (8,0) node[right=of c] {C};
  \coordinate (d) at (3,3) node[above=of d] {D};
  \coordinate (e) at (5.5,3) node[above=3mm of e] {E};
  \coordinate (f) at (8,3) node[above right=of f] {F};
  \node[fixed support,draw,grounded=-90] (A) at (a){};
  \node[support,draw] (B) at (b){};
  \node[hinge,draw] (C) at (c){};
  \node[draw,ground,stretch factor=6.3,anchor=left] (GR) at (B.bottom){};
  \node[double pendulum b,draw,rotate=90] (E) at (e){};
  \draw[thick] (A) -- (d) -- (E.north) (E.south) -- (f) -- (C) (d) -- (B);
\end{tikzpicture}
```

## 3.1 Adding loads

Ti*k*Z-MEC allows to add distributed loads on parts (i.e. links or part of them) of a structure. Lets see some examples:




```
\usetikzlibrary{positioning}
\begin{tikzpicture}[node distance=1mm]
  % coordinates
  \coordinate (a) at (0,0)   node[left=of a] {A};
  \coordinate (b) at (2.5,0) node[right=of b] {B};
  \coordinate (c) at (0,2)   node[left=of c] {C};
  \coordinate (d) at (2.5,2) node[right=of d] {D};
  % constraints and structure
  \node[hinge,draw,grounded]   (A) at (a){};
  \node[support,grounded,draw] (B) at (b){};
  \draw[thick]   (A) -- (c) -- (d) -- (B);
  % load
  \path[draw straight load]    (c) -- (d);
  \begin{scope}[shift={(0,-5)}]
   % coordinates
   \coordinate (a) at (0,0)     node[right=of a] {A};
   \coordinate (b) at (2.5,0)   node[right=of b] {B};
   \coordinate (c) at (2.5,2.5) node[right=of c] {C};
   % constraints and structure
   \node[hinge,draw,grounded]   (A) at (a){};
   \node[support,grounded,draw] (B) at (b){};
   \draw[thick] (A) arc[start angle=178,end angle=90,
     radius=2.5cm] -- (B);
   % load
   \path[start angle=180,end angle=90,radius=2.5cm,
     draw curved load] (a) arc;
  \end{scope}
\end{tikzpicture}
```

As you can see, the actual version of the library allows to add loads on straight and curved links using two different keys: `draw straight load` and `draw curved load`. Every load is generated using an indipendent path and, by default, it is labeled with a $p$ positioned on the top right of it.

For straight links, you don't need to specify anything else. On the countrary, for curved links, you need to let the library know in time which are the `start angle`, `end angle` and (eventually `x` and `y`) `radius` of the loaded link. This is why those keys need to be anticipated as options to the path. If not, an error will occur because the library will not find their values.

You can personalize a load appearance, using some keys:

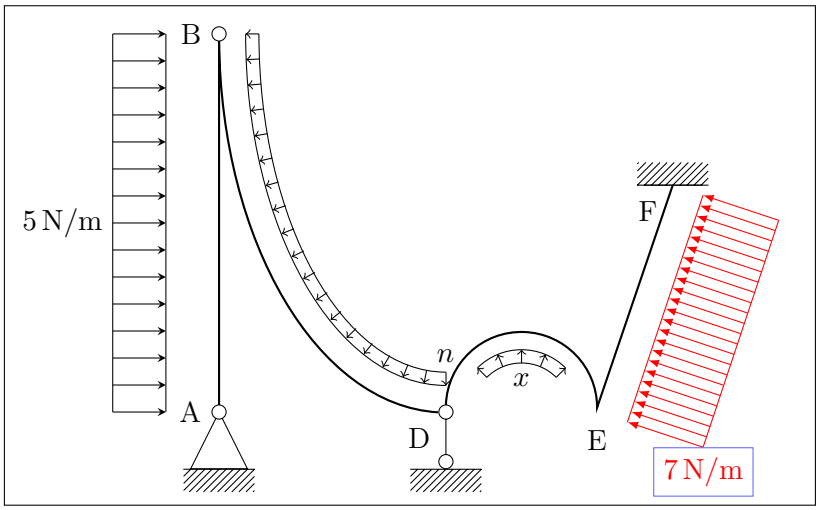
`draw` ⟨*load type*⟩ `load={[`⟨*options*⟩`]{`⟨*label*⟩`}}{`⟨*position*⟩`}` as previously explained, ⟨*load type*⟩ is either `straight` or `curved`. ⟨*label*⟩ is the label attached to the load and the optional ⟨*options*⟩ can be used to personalize it (just as in a normal node). The value stored as ⟨*position*⟩ can vary between 0 and 1 and it changes the position of the label along the farther (from the structure) line of the load.

`load distance` it sets the distance between the load and the structure. It is set to 10 by default;

`forces distance` this changes the distance between the single arrows (the forces) that compose the load and is useful to make visible the differences of modulus between different loads (default is 10);

`forces length` its purpose is the same as for `forces distance`, but it allows to change the length of the forces. Its default value is 20.

Using those keys you can easily get results like this:




```
\usetikzlibrary{positioning}
\begin{tikzpicture}[node distance=1mm]
\coordinate (a) at (0,0) node[left= of a]{A};
\coordinate (b) at (0,5) node[left= of b]{B};
\coordinate (d) at (3,0) node[below left= of d]{D};
\coordinate (e) at (5,0) node[below= of e]{E};
\coordinate (f) at (6,3) node[below left= of f]{F};
% constraints
\node[draw,hinge,grounded] (A) at (a){};
\node[draw,pendulum, grounded] (D) at (d){};
\node[draw,fixed support,grounded=180] (F) at (f){};
\node[draw,hinge b] (B) at (b){};
% structure
\draw[thick] (A) -- (B) (B.270)
  arc[start angle=181,end angle=268,x radius=3,y radius=5] (D.90)
  arc[start angle=178,end angle=0,radius=1cm] -- (F);
% loads
\path[>=stealth,load distance=20pt,
  draw straight load={[left]{\SI{5}{\newton/\meter}}}{0.5}] (a) -- (b);
\path[>=latex,red,forces length=30pt,forces distance=4pt, load distance=12pt,
  draw straight load={[{draw=blue!60,below}]{\SI{7}{\newton/\meter}}}{1}]
  (f) -- (e);
\path (4,0)+(45:1cm) coordinate (S);
\path[forces length=5pt,load distance=5pt,start angle=45,end angle=135,%
  radius=1cm,draw curved load={[below]{$x$}}{0.5}] (S) arc;
\path[forces length=5pt,start angle=180,end angle=270,%
  x radius=3cm,y radius=5cm,draw curved load={[above]{$n$}}{0.999}]
  (b) arc;
\end{tikzpicture}
```

# 4 TODO List

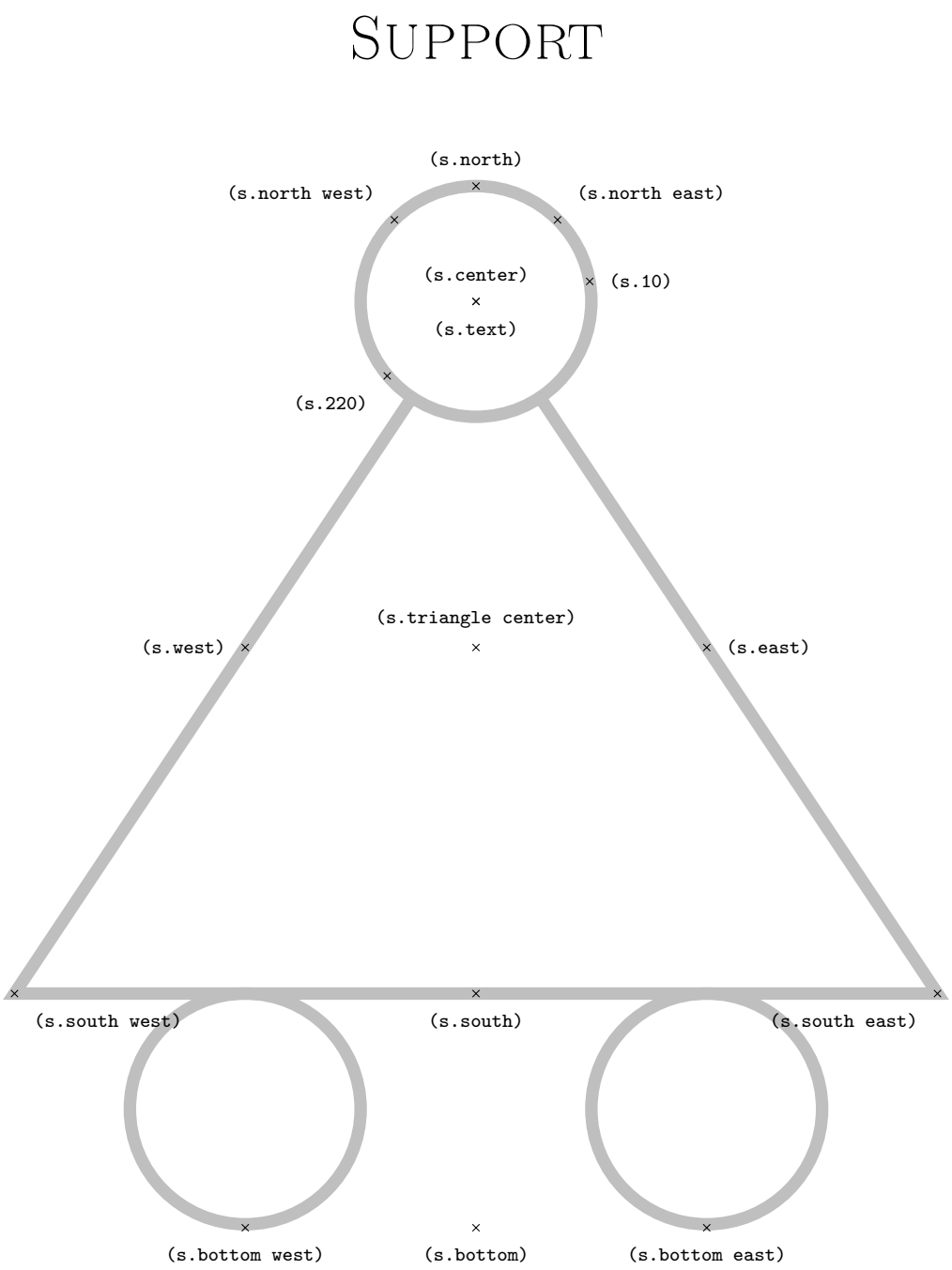
The following is a (unordered and incomplete) list of goals needed to improve the library. If you think that something is missing or something has to be deleted from the list, please let me know using the e-mail address in the first page. Well, if you really want to delete some items in the list, you should send me a solution too.

- Verify if the defined shapes don't violate a ISO/UNI standard (if it exists)
- Check the anchors defined for the various shapes and, if necessary, define new and delete others of them
- Modify the shading in the pendulum which now leads to a too dark lower circle
- Find other shared parameters between constraints and the ground
- Create a good interface to draw structures loads (in progress)
- Create a good interface to draw internal stresses distributions
- Solve precision problems affecting (mostly) the appearance of distributed loads on elliptical arcs

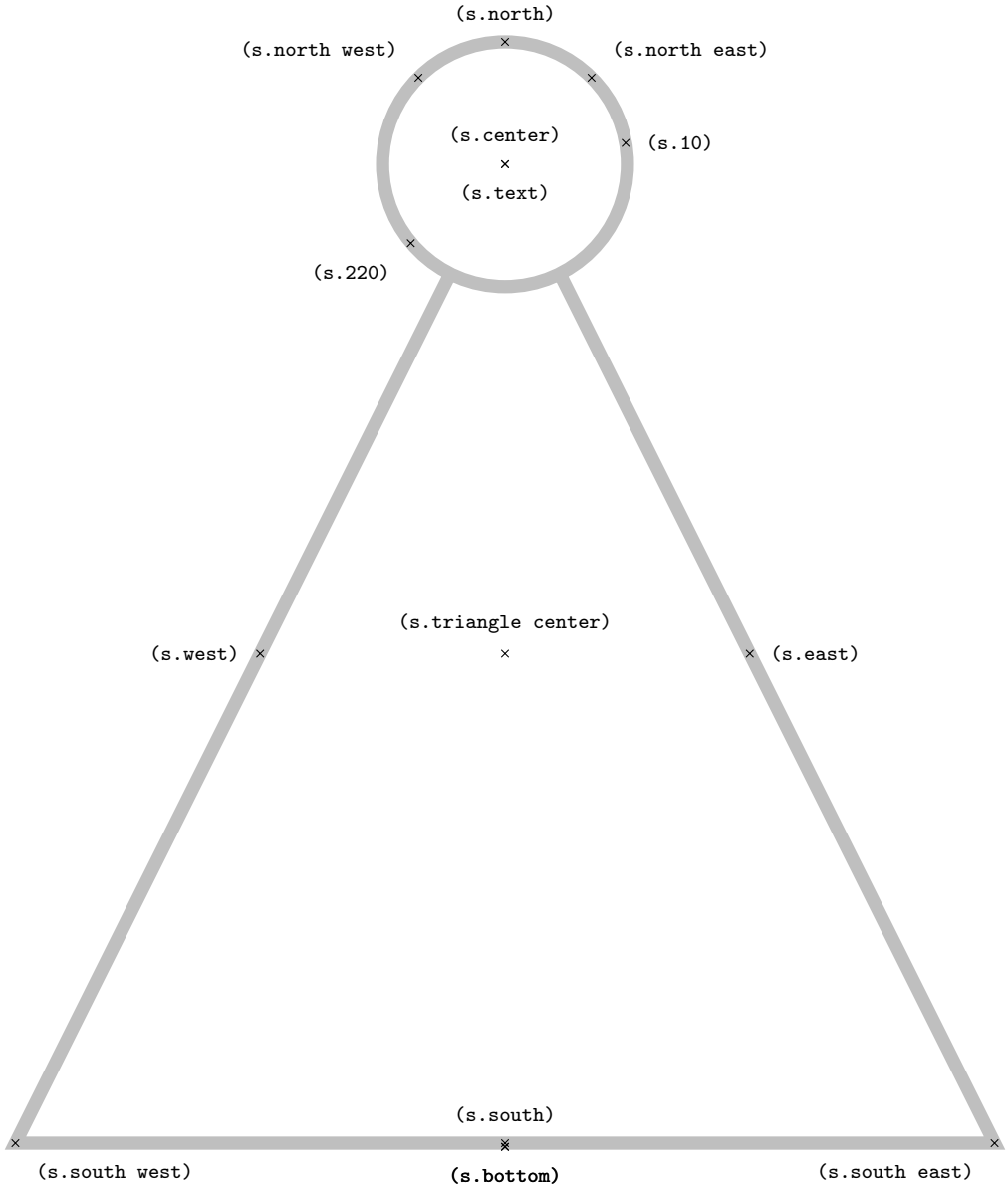
# Change History

## 5 Anchors

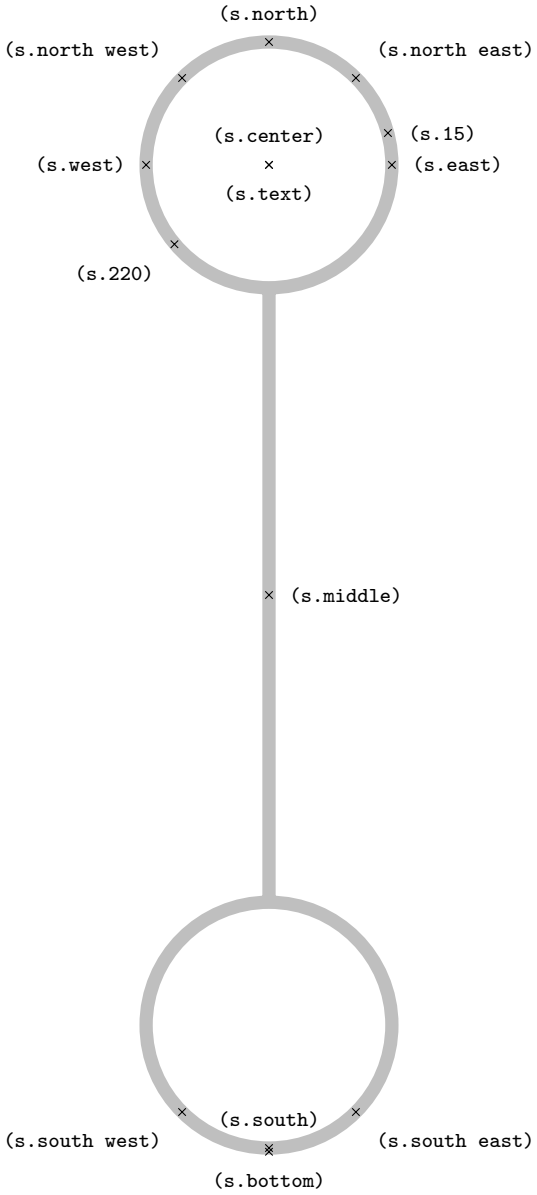
In this section you'll find all the defined anchors for all the defined shapes. They can be used to refer external objects (like rods and forces) to the constraints and to the ground.



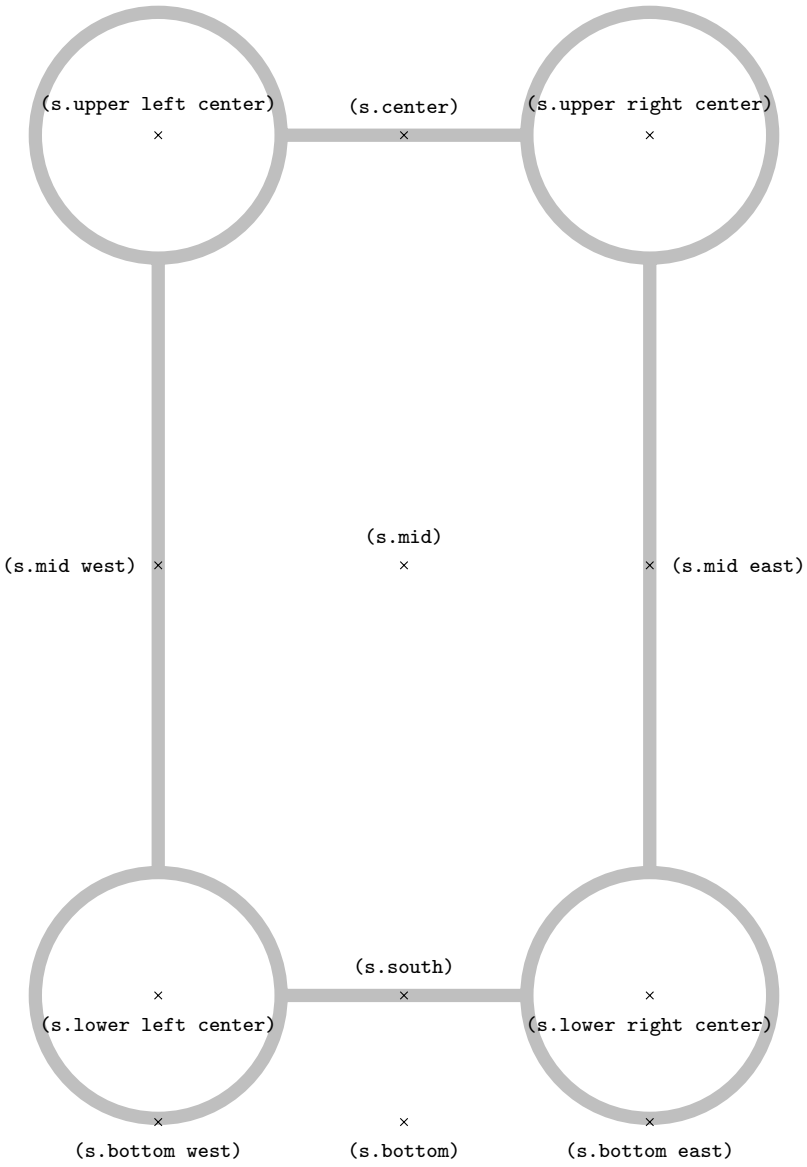
# Hinge


(s.north)
(s.north west)
(s.north east)
(s.center)
(s.text)
(s.10)
(s.220)
(s.triangle center)
(s.west)
(s.east)
(s.south)
(s.south west)
(s.bottom)
(s.south east)

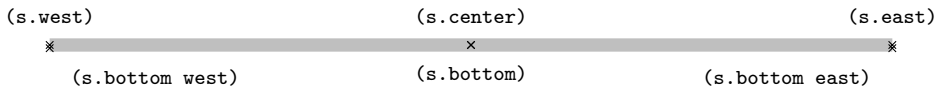
# Pendulum


(s.north)
(s.north west)
(s.north east)
(s.center)
(s.15)
(s.west)
(s.east)
(s.text)
(s.220)
(s.middle)
(s.south)
(s.south west)
(s.south east)
(s.bottom)

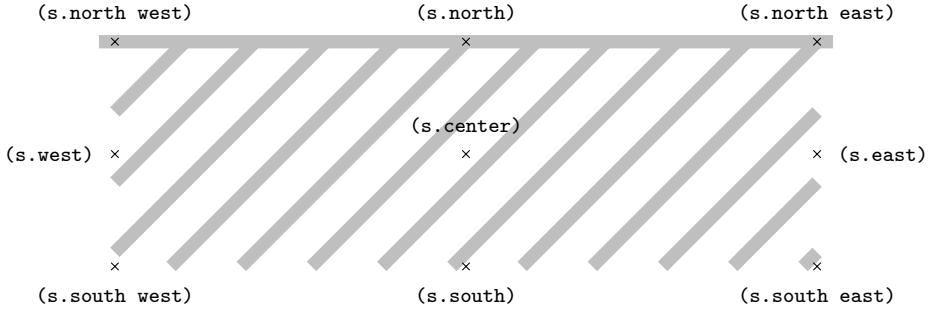
# Double pendulum



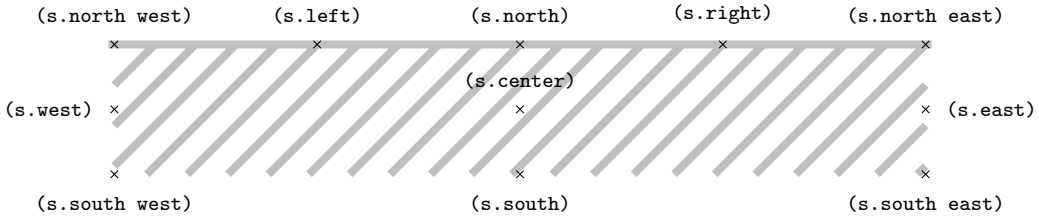
# Fixed support



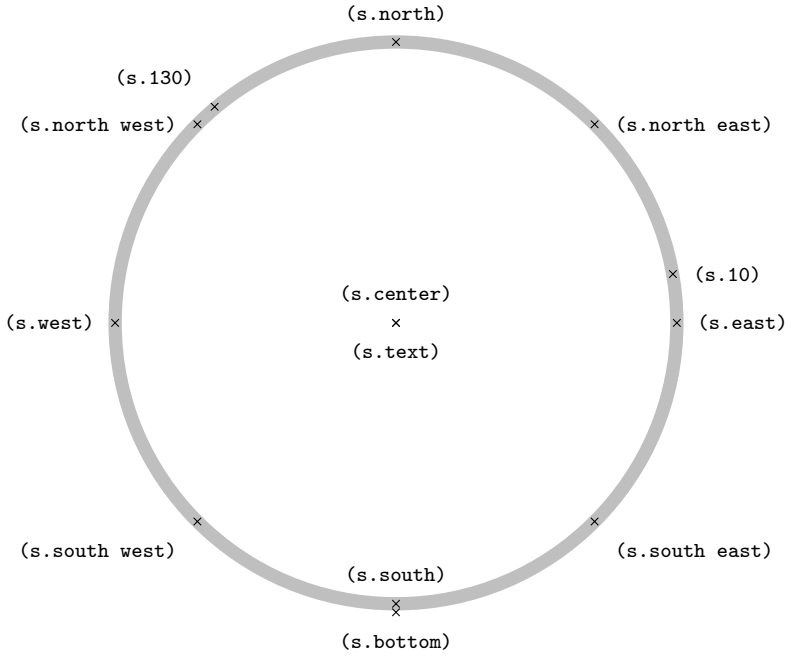

# Ground



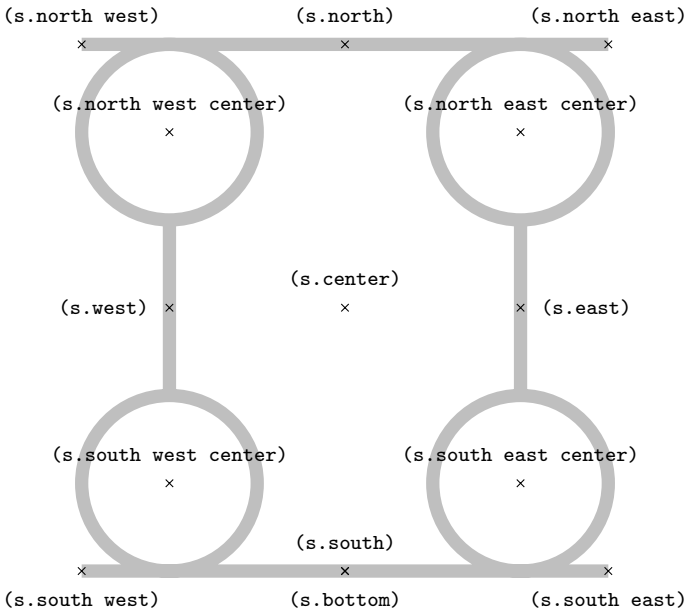

# Stretched ground



# Hinge B




# Double Pendulum B



# 6 Library Code

First of all, some styles which simplify the user interface are defined.

`grounded` `grounded` adds a node (shaped *ground*) at the baseline of a constraint allowing its rotation accordingly to it. The rotation angle has to be specified as a parameter of the key (i.e. `grounded=20`)

```
\tikzset{%
  Ground/.style={%
    append after command={;%
      \node[draw,ground,anchor=north,rotate=#1] (\tikzlastnode ground) at
      (\tikzlastnode.bottom){}}},%
  Ground/.default=0,%
  grounded/.style={Ground=#1,rotate=#1},%
  grounded/.default=0}%
```

`loads` Here we define useful keys and decorations to easily draw distributed loads on a structure or truss. It is possible to add such loads on straigth and curved (i.e. circular and elliptical arcs) paths.

Define decoration that draws the arrows that composes a load on a straight path

```
\pgfdeclaredecoration{mec@draw@straight@load@forces}{initial}{%
```

The width of each decoration is equal to the path length (stored in `\pgfdecoratedpathlength`) divided by the number of arrows. The number of arrows is calculated dividing the length of the path by the distance between the arrows (contained in the `force distance` key) and rounding the result to assure an integer number

```
  \state{initial}[width=\pgfdecoratedpathlength/%
     round(\pgfdecoratedpathlength/\pgfkeysvalueof{/tikz/forces distance})]{%
```

Loading parameters used to personalize the load appearance

```
    \pgf@xb=\pgfkeysvalueof{/tikz/load distance}
    \pgf@xa=\pgfkeysvalueof{/tikz/forces length}
```

`\pgf@xa` gets the distance between the path and the beginning of the arrows

```
    \advance\pgf@xa by\pgf@xb
```

Draw the arrow perpendicular to the path. No need to add an arrow tip (using, for example, `\pgfsetarrows{}`) at this level. It will be added at higher level in the style definition (this allows customization at user level defining `>`)

```
    \pgfpathmoveto{\pgfpoint{0pt}{\pgf@xa}}
    \pgfpathlineto{\pgfpoint{0pt}{\pgf@xb}}
    \pgfusepath{stroke}
  }
  \state{final}{%
```

The last arrow is not drawed in the `initial` state, so it is needed to add it. Same operations as above, but starting from the last point of the path

```
    \pgf@xb=\pgfkeysvalueof{/tikz/load distance}
    \pgf@xa=\pgfkeysvalueof{/tikz/forces length}
    \advance\pgf@xa by\pgf@xb
    \pgfpathmoveto{%
      \pgfpointadd{\pgfpoint{0pt}{\pgf@xa}}{\pgfpointdecoratedpathlast}}
    \pgfpathlineto{%
      \pgfpointadd{\pgfpoint{0pt}{\pgf@xb}}{\pgfpointdecoratedpathlast}}
    \pgfusepath{stroke}
  }
}
```

Define a decoration that draws the box containing the arrows of a load on a straight path

```
\pgfdeclaredecoration{mec@draw@straight@load@box}{initial}{%
```

Here we use a fake width and draw only once (thanks to the `next state` option) the two paths (above and below the arrows)

```
  \state{initial}[width=1pt,next state=final]{
    \pgf@xa=\pgfkeysvalueof{/tikz/load distance}
    \pgf@xb=\pgfkeysvalueof{/tikz/forces length}
    \advance\pgf@xb by\pgf@xa
```

Draw the path far from the structure

```
    \pgfpathmoveto{\pgfpoint{0pt}{\pgf@xa}}
    \pgfpathlineto{%
      \pgfpointadd{\pgfpoint{0pt}{\pgf@xa}}{\pgfpointdecoratedpathlast}}%
```

Draw the path near the structure

```
    \pgfpathmoveto{\pgfpoint{0pt}{\pgf@xb}}
    \pgfpathlineto{%
      \pgfpointadd{\pgfpoint{0pt}{\pgf@xb}}{\pgfpointdecoratedpathlast}}%
    \pgfusepath{stroke}
  }
  \state{final}{%
    \pgfpathmoveto{\pgfpointdecoratedpathlast}
  }
}
```

Define a decoration that draws the arrows on a curved (i.e. circular or elliptical) path. Everything is quite similar to the straight case

```
\pgfdeclaredecoration{mec@draw@curved@load@forces}{initial}{%
```

Same width as for the straigth one. The only difference is the presence of the multiplier 0.999 to avoid artifacts affecting the last arrow (bad positioning)

```
  \state{initial}[width=0.999*\pgfdecoratedpathlength/%
     round(\pgfdecoratedpathlength/\pgfkeysvalueof{/tikz/forces distance})]{%
    \pgf@xb=\pgfkeysvalueof{/tikz/load distance}
    \pgf@xa=\pgfkeysvalueof{/tikz/forces length}
    \advance\pgf@xa by\pgf@xb
    \pgfpathmoveto{\pgfpoint{0pt}{\pgf@xa}}
    \pgfpathlineto{\pgfpoint{0pt}{\pgf@xb}}
    \pgfusepath{stroke}
  }
  \state{final}{%
    \pgf@xb=\pgfkeysvalueof{/tikz/load distance}
    \pgf@xa=\pgfkeysvalueof{/tikz/forces length}
    \advance\pgf@xa by\pgf@xb
    \pgfpathmoveto{%
      \pgfpointadd{\pgfpoint{0pt}{\pgf@xa}}{\pgfpointdecoratedpathlast}}
    \pgfpathlineto{%
      \pgfpointadd{\pgfpoint{0pt}{\pgf@xb}}{\pgfpointdecoratedpathlast}}
    \pgfusepath{stroke}
  }
}
```

Define a decoration that draws the box containing the arrows of a load on a curved path

```
\pgfdeclaredecoration{mec@draw@curved@load@box}{initial}{%
  \state{initial}[width=1pt,next state=final]{%
```

It is needed to store the values of the $x$ radius and $y$ radius and to add or subtract (depending on the position of the load) to them the distance between the load and the structure

```
    \pgf@xa=\pgfkeysvalueof{/tikz/load distance}
    \pgf@ya=\pgf@xa
    \pgf@xb=\pgfkeysvalueof{/tikz/x radius}
    \pgf@yb=\pgfkeysvalueof{/tikz/y radius}
    \pgf@xc=\pgfkeysvalueof{/tikz/forces length}
```

If the load is drawn internally (i.e. `start angle` is smaller than `end angle`) the final radius of the box lines are both smaller than the one of the truss. So `\mec@adjust@factor` is set to $-1$ (using the `\pgfmathtruncatemacro` I can automatically discard the decimal part) and is used as a multiplier of the lengths

```
    \pgfmathtruncatemacro\mec@adjust@factor{%
      \pgfkeysvalueof{/tikz/start angle} <%
      \pgfkeysvalueof{/tikz/end angle} ? -1 : 1}
    \multiply\pgf@xa by\mec@adjust@factor
    \multiply\pgf@xc by\mec@adjust@factor
    \advance\pgf@xb by\pgf@xa
    \advance\pgf@yb by\pgf@xa
    \pgfpathmoveto{\pgfpoint{0pt}{\pgf@ya}}
```

It is needed to adjust the `start angle` and `end angle` to correctly draw the arcs adding to it a different factor based on the position (internal or external) of the load

```
    \pgfmathtruncatemacro\mec@adjust@angle{%
      (90*\mec@adjust@factor)-\pgfkeysvalueof{/tikz/start angle}}
    \pgfpatharc{\pgfkeysvalueof{/tikz/start angle}+\mec@adjust@angle}%
               {\pgfkeysvalueof{/tikz/end angle}+\mec@adjust@angle}%
               {\pgf@yb and \pgf@xb}% switched
    \advance\pgf@ya by\mec@adjust@factor\pgf@xc
    \advance\pgf@xb by\pgf@xc
    \advance\pgf@yb by\pgf@xc
    \pgfpathmoveto{\pgfpoint{0pt}{\pgf@ya}}
    \pgfpatharc{\pgfkeysvalueof{/tikz/start angle}+\mec@adjust@angle}%
               {\pgfkeysvalueof{/tikz/end angle}+\mec@adjust@angle}%
               {\pgf@yb and \pgf@xb}% switched
    \pgfusepath{stroke}
  }
  \state{final}{%
    \pgfpathmoveto{\pgfpointdecoratedpathlast}
  }
}
```

Now some styles are defined to allow an easy use of the above defined decorations.

```
\tikzset{load distance/.initial=10pt,
         forces distance/.initial=10pt,
         forces length/.initial=20pt,
         total distance/.initial=30pt,
         draw curved load/.style 2 args={%
           total distance={\pgfkeysvalueof{/tikz/forces length}+%
             \pgfkeysvalueof{/tikz/load distance}},%
           postaction={->,decorate,decoration={mec@draw@curved@load@forces}},%
           postaction={decorate,decoration=mec@draw@curved@load@box},%
           postaction={decorate,decoration={markings,%
             raise=\pgfkeysvalueof{/tikz/total distance},%
             mark=at position #2 with \node #1;}}},
```

```
        draw curved load/.default={[right]{$p$}}{0.999},
        draw straight load/.style 2 args={%
          total distance={\pgfkeysvalueof{/tikz/forces length}+%
            \pgfkeysvalueof{/tikz/load distance}},%
          postaction={->,decorate,decoration={mec@draw@straight@load@forces}},%
          postaction={decorate,decoration=mec@draw@straight@load@box},%
          postaction={decorate,decoration={markings,%
            raise=\pgfkeysvalueof{/tikz/total distance},%
            mark=at position #2 with \node #1;}}},
        draw straight load/.default={[right]{$p$}}{0.999}
}
```

stretch factor — The value passed to `stretch factor` will be used to compute the final lenght of the interested `ground`. The default value is 1 (no stretch present)

```
\tikzset{stretch factor/.initial=1}
```

Here starts the library's shapes definitions. All the defined shapes are based on the standard shape `circle`, using as reference dimension its radius (saved in `\radius`)

support — Starting defining the support

```
\pgfdeclareshape{support}{%
```

Load the "saved anchors" from the circle (`\centerpoint` and `\radius`)

```
  \inheritsavedanchors[from=circle]
```

Create a new dimension which is one half of the `circle` radius (the "last" saved anchor)

```
  \saveddimen{\halfradius}{%
    \pgf@x=.5\pgf@x}
```

Load some anchors from the circle and redefine others to adapt them to the new dimension `\halfradius`

```
  \anchorborder{
    \pgf@xa=\pgf@x%
    \pgf@ya=\pgf@y%
    \edef\pgf@marshal{%
      \noexpand\pgfpointborderellipse
      {\noexpand\pgfqpoint{\the\pgf@xa}{\the\pgf@ya}}
      {\noexpand\pgfqpoint{\halfradius}{\halfradius}}% <- edited here
    }%
    \pgf@marshal%
    \pgf@xa=\pgf@x%
    \pgf@ya=\pgf@y%
    \centerpoint%
    \advance\pgf@x by\pgf@xa%
    \advance\pgf@y by\pgf@ya%
  }
  \inheritanchor[from=circle]{center}
  \inheritanchor[from=circle]{mid}
  \inheritanchor[from=circle]{base}
  \anchor{north}{\centerpoint\advance\pgf@y by\halfradius}
  \anchor{south}{%
    \centerpoint
    \pgf@xa=\radius
    \advance\pgf@y by-3\pgf@xa}
  \anchor{west}{%
    \centerpoint
    \pgf@xa=\radius
```

```
    % y = -1.5*r*tan(alpha)
    \advance\pgf@x by-1\pgf@xa
    \advance\pgf@y by-1.5\pgf@xa
  }
  \anchor{east}{%
    \centerpoint
    \pgf@xa=\radius
    % y = 1.5*r*tan(alpha)
    \advance\pgf@x by1\pgf@xa
    \advance\pgf@y by-1.5\pgf@xa
  }
  \anchor{mid west}{%
    \centerpoint
    \advance\pgf@x by-\halfradius\pgfmathsetlength\pgf@y{.5ex}
  }
  \anchor{mid east}{%
    \centerpoint
    \advance\pgf@x by\halfradius\pgfmathsetlength\pgf@y{.5ex}
  }
  \anchor{base west}{\centerpoint\advance\pgf@x by-\halfradius\pgf@y=0pt}
  \anchor{base east}{\centerpoint\advance\pgf@x by\halfradius\pgf@y=0pt}
  \anchor{north west}{%
    \centerpoint
    \pgf@xa=\halfradius
    \advance\pgf@x by-0.707107\pgf@xa
    \advance\pgf@y by0.707107\pgf@xa
  }
  \anchor{south west}{%
    \centerpoint
    \pgf@xa=\radius
    \advance\pgf@x by-2\pgf@xa
    \advance\pgf@y by-3\pgf@xa
  }
  \anchor{north east}{%
    \centerpoint
    \pgf@xa=\halfradius
    \advance\pgf@x by0.707107\pgf@xa
    \advance\pgf@y by0.707107\pgf@xa
  }
  \anchor{south east}{%
    \centerpoint
    \pgf@xa=\radius
    \advance\pgf@x by2\pgf@xa
    \advance\pgf@y by-3\pgf@xa
  }
  \anchor{triangle center}{%
    \centerpoint
    \pgf@xa=\radius
    \advance\pgf@y by-1.5\pgf@xa
    }
```

Add anchors at the base of the shape, useful to correctly place the ground (considering one half of the line width)

```
  \anchor{bottom}{%
    \centerpoint
    \pgf@xa=\radius
```

```
    \pgf@xb=\halfradius
    \advance\pgf@y by-3\pgf@xa
    \advance\pgf@y by-2\pgf@xb
    \advance\pgf@y by-.5\pgflinewidth
  }
  \anchor{bottom west}{%
    \centerpoint
    \pgf@xa=\radius
    \pgf@xb=\halfradius
    \advance\pgf@y by-3\pgf@xa
    \advance\pgf@y by-2\pgf@xb
    \advance\pgf@y by-.5\pgflinewidth
    \advance\pgf@x by-\pgf@xa
  }
  \anchor{bottom east}{%
    \centerpoint
    \pgf@xa=\radius
    \pgf@xb=\halfradius
    \advance\pgf@y by-3\pgf@xa
    \advance\pgf@y by-2\pgf@xb
    \advance\pgf@y by-.5\pgflinewidth
    \advance\pgf@x by\pgf@xa
  }
```

Draw the shape

```
  \backgroundpath{%
      \pgf@x=\radius
```

Radius is also containing the "minimum width" and "minimum height". This ensures that even with no text the shape will be drawn, unless of course that minimums are set to 0 pt. Now save `\radius` and `\halfradius` (that will be the radius of the drawed circles)

```
      \pgfutil@tempdima=\pgf@x%
      \pgfutil@tempdimb=\halfradius%
```

Define the west triangle corner "b" and the east triangle corner "c"

```
      \pgf@xb=-2\pgf@x%
      \pgf@yb=-3\pgf@x%
      \pgf@xc= 2\pgf@x%
      \pgf@yc=-3\pgf@x%
      % (half inner angle is 33.69)
```

If text is present, shift shape to the center

```
      \centerpoint
      \advance\pgf@xb by\pgf@x
      \advance\pgf@yb by\pgf@y
      \advance\pgf@xc by\pgf@x
      \advance\pgf@yc by\pgf@y
```

Save centerpoint in "a" (triangle top corner)

```
      \pgf@xa=\pgf@x
      \pgf@ya=\pgf@y
```

Below are good for debugging purposes

```
      %\message{^^JTop : \the\pgf@xa,\the\pgf@ya}
      %\message{^^JWest: \the\pgf@xb,\the\pgf@yb}
      %\message{^^JEast: \the\pgf@xc,\the\pgf@yc}
      %\message{^^JCent: \the\pgf@x,\the\pgf@y}
      %\message{^^JR: \the\pgfutil@tempdima}
```

Draw the triangle (considering overimposed upper circle). The starting point is $0.5$\radius$\cos(\alpha)$ below and $0.5$\radius$\sin(\alpha)$ on the left of the center point (because \halfradius $= 0.5$\radius). Then the shift is $(0.5547, 0.832)$\halfradius

```
      \advance\pgf@xa by-0.5547\pgfutil@tempdimb%
      \advance\pgf@ya by-0.832\pgfutil@tempdimb%
      \pgfpathmoveto{\pgfpoint{\pgf@xa}{\pgf@ya}}%
      \pgfpathlineto{\pgfpoint{\pgf@xb}{\pgf@yb}}%
      \pgfpathlineto{\pgfpoint{\pgf@xc}{\pgf@yc}}%
```

Shifting "a" to upper right edge and back to upper left edge to close the path

```
      \advance\pgf@xa by1.1094\pgfutil@tempdimb%
      \pgfpathlineto{\pgfpoint{\pgf@xa}{\pgf@ya}}%
      \advance\pgf@xa by-1.1094\pgfutil@tempdimb%
      \pgfpathmoveto{\pgfpoint{\pgf@xa}{\pgf@ya}}%
      \pgfpathclose%
```

Restore "a" in \centerpoint (triangle top corner)

```
      \advance\pgf@xa by0.5547\pgfutil@tempdimb%
      \advance\pgf@ya by0.832\pgfutil@tempdimb%
```

Move "b" to west circle center and "c" to east circle center

```
      \advance\pgf@xb by 1\pgfutil@tempdima
      \advance\pgf@yb by -\pgfutil@tempdimb
      \advance\pgf@xc by-1\pgfutil@tempdima
      \advance\pgf@yc by -\pgfutil@tempdimb
```

Drawing the three circles

```
      \edef\pgf@marshal{%
          \noexpand\pgfpathcircle{%
              \noexpand\pgfqpoint{\the\pgf@xa}{\the\pgf@ya}}
          {\the\pgfutil@tempdimb}%
          \noexpand\pgfpathcircle{%
              \noexpand\pgfqpoint{\the\pgf@xb}{\the\pgf@yb}}
          {\the\pgfutil@tempdimb}%
          \noexpand\pgfpathcircle{%
              \noexpand\pgfqpoint{\the\pgf@xc}{\the\pgf@yc}}
          {\the\pgfutil@tempdimb}%
      }\pgf@marshal
  }%
}
```

`hinge` Start of hinge definition. Similar operations on anchors to what has been made for the support

```
\pgfdeclareshape{hinge}{%
  \inheritsavedanchors[from=circle]
  \saveddimen{\halfradius}{%
    \pgf@x=.5\pgf@x}
  \anchorborder{
    \pgf@xa=\pgf@x%
    \pgf@ya=\pgf@y%
    \edef\pgf@marshal{%
      \noexpand\pgfpointborderellipse
      {\noexpand\pgfqpoint{\the\pgf@xa}{\the\pgf@ya}}
      {\noexpand\pgfqpoint{\halfradius}{\halfradius}}% <- edited here
    }%
    \pgf@marshal%
    \pgf@xa=\pgf@x%
```

```
    \pgf@ya=\pgf@y%
    \centerpoint%
    \advance\pgf@x by\pgf@xa%
    \advance\pgf@y by\pgf@ya%
  }
  \inheritanchor[from=circle]{center}
  \inheritanchor[from=circle]{mid}
  \inheritanchor[from=circle]{base}
  \anchor{north}{\centerpoint\advance\pgf@y by\halfradius}
  \anchor{south}{%
    \centerpoint
    \pgf@xa=\radius
    \advance\pgf@y by-4\pgf@xa
  }
  \anchor{west}{%
    \centerpoint
    \pgf@xa=\radius
    % y = -2*r*tan(alpha)
    \advance\pgf@x by-1\pgf@xa
    \advance\pgf@y by-2\pgf@xa
  }
  \anchor{east}{%
    \centerpoint
    \pgf@xa=\radius
    % y = 2*r*tan(alpha)
    \advance\pgf@x by1\pgf@xa
    \advance\pgf@y by-2\pgf@xa
  }
  \anchor{mid west}{%
    \centerpoint
    \advance\pgf@x by-\halfradius\pgfmathsetlength\pgf@y{.5ex}}
  \anchor{mid east}{%
    \centerpoint
    \advance\pgf@x by\halfradius\pgfmathsetlength\pgf@y{.5ex}}
  \anchor{base west}{\centerpoint\advance\pgf@x by-\halfradius\pgf@y=0pt}
  \anchor{base east}{\centerpoint\advance\pgf@x by\halfradius\pgf@y=0pt}
  \anchor{north west}{%
    \centerpoint
    \pgf@xa=\halfradius
    \advance\pgf@x by-0.707107\pgf@xa
    \advance\pgf@y by0.707107\pgf@xa
  }
  \anchor{south west}{%
    \centerpoint
    \pgf@xa=\radius
    \advance\pgf@x by-2\pgf@xa
    \advance\pgf@y by-4\pgf@xa
  }
  \anchor{north east}{%
    \centerpoint
    \pgf@xa=\halfradius
    \advance\pgf@x by0.707107\pgf@xa
    \advance\pgf@y by0.707107\pgf@xa
  }
  \anchor{south east}{%
    \centerpoint
```

```
    \pgf@xa=\radius
    \advance\pgf@x by2\pgf@xa
    \advance\pgf@y by-4\pgf@xa
  }
  \anchor{bottom}{%
    \centerpoint
    \pgf@xa=\radius
    \advance\pgf@y by-4\pgf@xa
    \advance\pgf@y by-.5\pgflinewidth
  }
  \anchor{triangle center}{%
    \centerpoint
    \pgf@xa=\radius
    \advance\pgf@y by-2\pgf@xa
  }
```

Defining the shape

```
  \backgroundpath{%
      \pgf@x=\radius
      \pgfutil@tempdima=\pgf@x%
      \pgfutil@tempdimb=\halfradius%
      % west triangle corner "b"
      \pgf@xb=-2\pgf@x%
      \pgf@yb=-4\pgf@x%
      % east triangle corner "c"
      \pgf@xc= 2\pgf@x%
      \pgf@yc=-4\pgf@x%
```

Half angle is 26.565°

```
      % If text is present shift shape to center
      \centerpoint
      \advance\pgf@xb by\pgf@x
      \advance\pgf@yb by\pgf@y
      \advance\pgf@xc by\pgf@x
      \advance\pgf@yc by\pgf@y
      % Save centerpoint in "a" (triangle top corner)
      \pgf@xa=\pgf@x
      \pgf@ya=\pgf@y
```

Draw the triangle (considering upper circle). See the shape `support` for details. The shift is $(0.4472, 0.8944)$`\radius`. Starting from upper left edge

```
      \advance\pgf@xa by-0.4472\pgfutil@tempdimb%
      \advance\pgf@ya by-0.8944\pgfutil@tempdimb%
      \pgfpathmoveto{\pgfpoint{\pgf@xa}{\pgf@ya}}%
      \pgfpathlineto{\pgfpoint{\pgf@xb}{\pgf@yb}}%
      \pgfpathlineto{\pgfpoint{\pgf@xc}{\pgf@yc}}%
      % changing "a" to upper right edge...
      \advance\pgf@xa by0.8944\pgfutil@tempdimb%
      \pgfpathlineto{\pgfpoint{\pgf@xa}{\pgf@ya}}%
      % ...and back to upper left to close the path
      \advance\pgf@xa by-0.8944\pgfutil@tempdimb%
      \pgfpathmoveto{\pgfpoint{\pgf@xa}{\pgf@ya}}%
      \pgfpathclose%
```

Restore the center point in "a" (triangle top corner) and draw the circle

```
      \advance\pgf@xa by0.4472\pgfutil@tempdimb%
      \advance\pgf@ya by0.8944\pgfutil@tempdimb%
```

```
      \edef\pgf@marshal{%
          \noexpand\pgfpathcircle{%
              \noexpand\pgfqpoint{\the\pgf@xa}{\the\pgf@ya}}
              {\the\pgfutil@tempdimb}%
      }\pgf@marshal
  }%
}
```

`hinge b` Starting the definition of the alternative shape for the hinge, useful as an internal constraint. Again same operations on anchors

```
\pgfdeclareshape{hinge b}{%
  \inheritsavedanchors[from=circle]
  \saveddimen{\halfradius}{%
    \pgf@x=.5\pgf@x}
  \anchorborder{
    \pgf@xa=\pgf@x%
    \pgf@ya=\pgf@y%
    \edef\pgf@marshal{%
      \noexpand\pgfpointborderellipse
      {\noexpand\pgfqpoint{\the\pgf@xa}{\the\pgf@ya}}
      {\noexpand\pgfqpoint{\halfradius}{\halfradius}}% <- edited here
    }%
    \pgf@marshal%
    \pgf@xa=\pgf@x%
    \pgf@ya=\pgf@y%
    \centerpoint%
    \advance\pgf@x by\pgf@xa%
    \advance\pgf@y by\pgf@ya%
  }
  \inheritanchor[from=circle]{center}
  \inheritanchor[from=circle]{mid}
  \inheritanchor[from=circle]{base}
  \anchor{north}{\centerpoint\advance\pgf@y by\halfradius}
  \anchor{south}{\centerpoint\advance\pgf@y by-\halfradius}
  \anchor{west}{\centerpoint\advance\pgf@x by-\halfradius}
  \anchor{east}{\centerpoint\advance\pgf@x by\halfradius}
  \anchor{mid west}{%
    \centerpoint
    \advance\pgf@x by-\halfradius\pgfmathsetlength\pgf@y{.5ex}
  }
  \anchor{mid east}{%
    \centerpoint
    \advance\pgf@x by\halfradius\pgfmathsetlength\pgf@y{.5ex}
  }
  \anchor{base west}{%
    \centerpoint
    \advance\pgf@x by-\halfradius\pgf@y=0pt
  }
  \anchor{base east}{%
    \centerpoint
    \advance\pgf@x by\halfradius\pgf@y=0pt
  }
  \anchor{north west}{%
    \centerpoint
    \pgf@xa=\halfradius
    \advance\pgf@x by-0.707107\pgf@xa
```

```
    \advance\pgf@y by0.707107\pgf@xa
  }
  \anchor{south west}{%
    \centerpoint
    \pgf@xa=\halfradius
    \advance\pgf@x by-0.707107\pgf@xa
    \advance\pgf@y by-0.707107\pgf@xa
  }
  \anchor{north east}{%
    \centerpoint
    \pgf@xa=\halfradius
    \advance\pgf@x by0.707107\pgf@xa
    \advance\pgf@y by0.707107\pgf@xa
  }
  \anchor{south east}{%
    \centerpoint
    \pgf@xa=\halfradius
    \advance\pgf@x by0.707107\pgf@xa
    \advance\pgf@y by-0.707107\pgf@xa
  }
  \anchor{bottom}{%
    \centerpoint
    \advance\pgf@y by-\halfradius
    \advance\pgf@y by-0.5\pgflinewidth
  }
```

The shape is just a circle with the radius equal to `\halfradius`

```
  \backgroundpath{%
    \pgfutil@tempdima=\halfradius%
    \pgfpathcircle{\centerpoint}{\pgfutil@tempdima}%
  }%
}
```

`pendulum` Starting the definition of the pendulum. Same operations on anchors

```
\pgfdeclareshape{pendulum}{%
  \inheritsavedanchors[from=circle]
  \saveddimen{\halfradius}{%
    \pgf@x=.5\pgf@x}
  \anchorborder{
    \pgf@xa=\pgf@x%
    \pgf@ya=\pgf@y%
    \edef\pgf@marshal{%
      \noexpand\pgfpointborderellipse
      {\noexpand\pgfqpoint{\the\pgf@xa}{\the\pgf@ya}}
      {\noexpand\pgfqpoint{\halfradius}{\halfradius}}% <- edited here
    }%
    \pgf@marshal%
    \pgf@xa=\pgf@x%
    \pgf@ya=\pgf@y%
    \centerpoint%
    \advance\pgf@x by\pgf@xa%
    \advance\pgf@y by\pgf@ya%
  }
  \inheritanchor[from=circle]{center}
  \inheritanchor[from=circle]{mid}
  \inheritanchor[from=circle]{base}
```

```
\anchor{north}{\centerpoint\advance\pgf@y by\halfradius}
\anchor{south}{%
  \centerpoint
  \pgf@xa=\radius
  \advance\pgf@y by-4\pgf@xa
}
\anchor{west}{\centerpoint\advance\pgf@x by-\halfradius}
\anchor{east}{\centerpoint\advance\pgf@x by\halfradius}
\anchor{mid west}{%
  \centerpoint
  \advance\pgf@x by-\halfradius\pgfmathsetlength\pgf@y{.5ex}
}
\anchor{mid east}{%
  \centerpoint
  \advance\pgf@x by\halfradius\pgfmathsetlength\pgf@y{.5ex}
}
\anchor{base west}{\centerpoint\advance\pgf@x by-\halfradius\pgf@y=0pt}
\anchor{base east}{\centerpoint\advance\pgf@x by\halfradius\pgf@y=0pt}
\anchor{north west}{%
  \centerpoint
  \pgf@xa=\halfradius
  \advance\pgf@x by-0.707107\pgf@xa
  \advance\pgf@y by0.707107\pgf@xa
}
\anchor{south west}{%
  \centerpoint
  \pgf@xa=\radius
  \pgf@xb=\halfradius
  \advance\pgf@x by-0.707107\pgf@xb
  \advance\pgf@y by-3\pgf@xa
  \advance\pgf@y by-1.707107\pgf@xb
}
\anchor{north east}{%
  \centerpoint
  \pgf@xa=\halfradius
  \advance\pgf@x by0.707107\pgf@xa
  \advance\pgf@y by0.707107\pgf@xa
}
\anchor{south east}{%
  \centerpoint
  \pgf@xa=\radius
  \pgf@xb=\halfradius
  \advance\pgf@x by0.707107\pgf@xb
  \advance\pgf@y by-3\pgf@xa
  \advance\pgf@y by-1.707107\pgf@xb
}
\anchor{bottom}{%
  \centerpoint
  \pgf@xa=\radius
  \advance\pgf@y by-4\pgf@xa
  \advance\pgf@y by-.5\pgflinewidth
}
\anchor{middle}{%
  \centerpoint
  \pgf@xa=\radius
  \pgf@xb=\halfradius
```

```
    \advance\pgf@y by-1\pgf@xa
    \advance\pgf@y by-1.5\pgf@xb
  }
```

Define the shape

```
  \backgroundpath{%
      \pgf@x=\radius
      \pgfutil@tempdima=\pgf@x%
      \pgfutil@tempdimb=\halfradius%
```

Define the lower circle center "b"

```
      \pgf@xb=0\pgfutil@tempdima%
      \pgf@yb=-3\pgfutil@tempdima%
      \advance\pgf@yb by-\pgfutil@tempdimb
      % If text is present shift shape to center
      \centerpoint
      \advance\pgf@xb by\pgf@x
      \advance\pgf@yb by\pgf@y
      % Save centerpoint in "a" (top circle center)
      \pgf@xa=\pgf@x
      \pgf@ya=\pgf@y
```

Draw the vertical axis of the shape (considering upper and lower circle) starting from the upper edge

```
      \advance\pgf@ya by-\pgfutil@tempdimb%
      \advance\pgf@yb by\pgfutil@tempdimb%
      \pgfpathmoveto{\pgfpoint{\pgf@xa}{\pgf@ya}}%
      \pgfpathlineto{\pgfpoint{\pgf@xb}{\pgf@yb}}%
      \pgfpathclose
```

Restore "a" in center point (top circle center) and "b" in the center of the lower circle. After that, draw the two circles

```
      \advance\pgf@ya by\pgfutil@tempdimb%
      \advance\pgf@yb by-\pgfutil@tempdimb%
      \edef\pgf@marshal{%
          \noexpand\pgfpathcircle{%
              \noexpand\pgfqpoint{\the\pgf@xa}{\the\pgf@ya}}
              {\the\pgfutil@tempdimb}%
      \noexpand\pgfpathcircle{%
              \noexpand\pgfqpoint{\the\pgf@xb}{\the\pgf@yb}}
              {\the\pgfutil@tempdimb}%
      }\pgf@marshal
  }%
}
```

`louble pendulum` Beginning of definition of the double pendulum. Same operations on anchors

```
\pgfdeclareshape{double pendulum}{%
  \inheritsavedanchors[from=circle]
  \saveddimen{\halfradius}{%
    \pgf@x=.5\pgf@x}
  \inheritanchor[from=circle]{center}
  \anchor{mid}{% y = centerpoint - 1\radius - 1.5\halfradius
    \centerpoint
    \pgf@xa=\radius
    \pgf@xb=\halfradius
    \advance\pgf@y by-1\pgf@xa
    \advance\pgf@y by-1.5\pgf@xb
```

```
  }
  \anchor{mid west}{%
    % y = centerpoint - 1\radius - 1.5\halfradius
    % x = centerpoint - radius
    \centerpoint
    \pgf@xa=\radius
    \pgf@xb=\halfradius
    \advance\pgf@x by-1\pgf@xa
    \advance\pgf@y by-1\pgf@xa
    \advance\pgf@y by-1.5\pgf@xb
  }
  \anchor{mid east}{%
    % y = centerpoint - 1\radius - 1.5\halfradius
    % x = centerpoint + radius
    \centerpoint
    \pgf@xa=\radius
    \pgf@xb=\halfradius
    \advance\pgf@x by\pgf@xa
    \advance\pgf@y by-1\pgf@xa
    \advance\pgf@y by-1.5\pgf@xb
  }
  \anchor{south}{%
   \centerpoint
    \pgf@xa=\radius
    \pgf@xb=\halfradius
    \advance\pgf@y by-3\pgf@xa
    \advance\pgf@y by-\pgf@xb
  }
  \anchor{upper left center}{%
    \centerpoint
    \advance\pgf@x by-\radius
  }
  \anchor{upper right center}{%
    \centerpoint
    \advance\pgf@x by\radius
  }
  \anchor{lower left center}{%
    \centerpoint
    \pgf@xa=\radius
    \pgf@xb=\halfradius
    \advance\pgf@x by-\pgf@xa
    \advance\pgf@y by-3\pgf@xa
    \advance\pgf@y by-\pgf@xb
  }
  \anchor{lower right center}{%
    \centerpoint
    \pgf@xa=\radius
    \pgf@xb=\halfradius
    \advance\pgf@x by\pgf@xa
    \advance\pgf@y by-3\pgf@xa
    \advance\pgf@y by-\pgf@xb
  }
  \anchor{bottom}{%
    \centerpoint
    \pgf@xa=\radius
    \pgf@xb=\halfradius
```

```
    \advance\pgf@y by-3\pgf@xa
    \advance\pgf@y by-2\pgf@xb
    \advance\pgf@y by-.5\pgflinewidth
  }
  \anchor{bottom west}{%
    \centerpoint
    \pgf@xa=\radius
    \pgf@xb=\halfradius
    \advance\pgf@y by-3\pgf@xa
    \advance\pgf@y by-2\pgf@xb
    \advance\pgf@y by-.5\pgflinewidth
    \advance\pgf@x by-\pgf@xa
  }
  \anchor{bottom east}{%
    \centerpoint
    \pgf@xa=\radius
    \pgf@xb=\halfradius
    \advance\pgf@y by-3\pgf@xa
    \advance\pgf@y by-2\pgf@xb
    \advance\pgf@y by-.5\pgflinewidth
    \advance\pgf@x by\pgf@xa
  }
```

Define the shape

```
  \backgroundpath{%
      \pgf@x=\radius
      \pgfutil@tempdima=\pgf@x%
      \pgfutil@tempdimb=\halfradius%
```

Define the rectangle upper right corner “b” and lower left corner “c”

```
      \pgf@xb=\pgfutil@tempdima%
      \pgf@yb=0\pgfutil@tempdima%
      \pgf@xc=-\pgfutil@tempdima%
      \pgf@yc=-3\pgfutil@tempdima%
      \advance\pgf@yc by-\pgfutil@tempdimb%
      % If text is present shift shape to center
      % You need to shift more, but to get the idea
      \centerpoint
      \advance\pgf@xb by\pgf@x
      \advance\pgf@yb by\pgf@y
      \advance\pgf@xc by\pgf@x
      \advance\pgf@yc by\pgf@y
      % Save centerpoint in "a" (top rectangle centerpoint)
      \pgf@xa=\pgf@x
      \pgf@ya=\pgf@y
```

The drawed rectangle is made of multiple paths so it is necessary to create another unique path that will be filled correctly when needed (patterns are working too). This is because the fill operation is defined only for single paths

```
      \tikz@mode
      \iftikz@mode@fill
      \pgfpathmoveto{\pgfpoint{\pgf@xb}{\pgf@yb}}
      \pgfpathlineto{\pgfpoint{\pgf@xb}{\pgf@yc}}
      \pgfpathlineto{\pgfpoint{\pgf@xc}{\pgf@yc}}
      \pgfpathlineto{\pgfpoint{\pgf@xc}{\pgf@yb}}
      \pgfpathclose
      \pgfusepath{fill}\fi
```

Draw sides of rectangle (considering circles; this leads to multiple paths) starting from upper left edge and going counterclockwise

```
      \advance\pgf@xc by\pgfutil@tempdimb%
      \pgfpathmoveto{\pgfpoint{\pgf@xc}{\pgf@yb}}%
      % moving to upper right edge
      \advance\pgf@xb by-\pgfutil@tempdimb%
      \pgfpathlineto{\pgfpoint{\pgf@xb}{\pgf@yb}}%
      % moving to lower right edge
      \advance\pgf@xb by\pgfutil@tempdimb%
      \advance\pgf@yb by-\pgfutil@tempdimb%
      \pgfpathmoveto{\pgfpoint{\pgf@xb}{\pgf@yb}}%
      \advance\pgf@yc by\pgfutil@tempdimb%
      \pgfpathlineto{\pgfpoint{\pgf@xb}{\pgf@yc}}%
      % moving to lower left edge
      \advance\pgf@yc by-\pgfutil@tempdimb%
      \advance\pgf@xb by-\pgfutil@tempdimb%
      \pgfpathmoveto{\pgfpoint{\pgf@xb}{\pgf@yc}}%
      \pgfpathlineto{\pgfpoint{\pgf@xc}{\pgf@yc}}%
      % moving to upper right edge
      \advance\pgf@xc by-\pgfutil@tempdimb%
      \advance\pgf@yc by\pgfutil@tempdimb%
      \pgfpathmoveto{\pgfpoint{\pgf@xc}{\pgf@yc}}%
      \pgfpathlineto{\pgfpoint{\pgf@xc}{\pgf@yb}}%
      % closing the path
      \advance\pgf@xc by\pgfutil@tempdimb%
      \advance\pgf@yb by\pgfutil@tempdimb%
      \pgfpathmoveto{\pgfpoint{\pgf@xc}{\pgf@yb}}%
      \pgfpathclose%
```

Restore “b” and “c” and draw the circles

```
      \advance\pgf@xb by\pgfutil@tempdimb%
      \advance\pgf@xc by-\pgfutil@tempdimb%
      \advance\pgf@yc by-\pgfutil@tempdimb%
      \edef\pgf@marshal{%
          \noexpand\pgfpathcircle{%
              \noexpand\pgfqpoint{\the\pgf@xc}{\the\pgf@yb}}
              {\the\pgfutil@tempdimb}%
          \noexpand\pgfpathcircle{%
              \noexpand\pgfqpoint{\the\pgf@xb}{\the\pgf@yb}}
              {\the\pgfutil@tempdimb}%
          \noexpand\pgfpathcircle{%
              \noexpand\pgfqpoint{\the\pgf@xb}{\the\pgf@yc}}
              {\the\pgfutil@tempdimb}%
          \noexpand\pgfpathcircle{%
              \noexpand\pgfqpoint{\the\pgf@xc}{\the\pgf@yc}}
              {\the\pgfutil@tempdimb}%
      }\pgf@marshal
  }%
}
```

`fixed support` Starting defining the fixed support. Same operations on anchors

```
\pgfdeclareshape{fixed support}{%
  \inheritsavedanchors[from=circle]
  \saveddimen{\halfradius}{%
    \pgf@x=.5\pgf@x}
  \inheritanchor[from=circle]{center}
```

```
  \anchor{west}{%
    \centerpoint
    \pgf@xa=\radius
    \advance\pgf@x by-2\pgf@xa
  }
  \anchor{east}{%
    \centerpoint
    \pgf@xa=\radius
    \advance\pgf@x by2\pgf@xa
  }
  \anchor{bottom}{%
    \centerpoint
  }
  \anchor{bottom west}{%
    \centerpoint
    \pgf@xa=\radius
    \advance\pgf@x by-2\pgf@xa
    \advance\pgf@y by-.5\pgflinewidth
  }
  \anchor{bottom east}{%
    \centerpoint
    \pgf@xa=\radius
    \advance\pgf@x by2\pgf@xa
    \advance\pgf@y by-.5\pgflinewidth
  }
```

Define the shape

```
  \backgroundpath{%
      \pgf@x=\radius
      \pgfutil@tempdima=\pgf@x%
      \pgfutil@tempdimb=\halfradius%
```

Define the line edges starting from the left edge "b" and ending to the right edge "c"

```
      \pgf@xb=-2\pgfutil@tempdima
      \pgf@yb=0\pgfutil@tempdima%
      \pgf@xc=2\pgfutil@tempdima
      \pgf@yc=0\pgfutil@tempdima%
      % If text is present shift shape to center
      \centerpoint
      \advance\pgf@xb by\pgf@x
      \advance\pgf@yb by\pgf@y
      \advance\pgf@xc by\pgf@x
      \advance\pgf@yc by\pgf@y
      % Save centerpoint in "a" (middle line point)
      \pgf@xa=\pgf@x
      \pgf@ya=\pgf@y
```

Draw the line from left edge to right edge

```
      \pgfpathmoveto{\pgfpoint{\pgf@xb}{\pgf@yb}}%
      \pgfpathlineto{\pgfpoint{\pgf@xc}{\pgf@yc}}%
      \pgfpathclose
  }%
}
```

`ground` Beginning of the ground definition in such a way that allows to stretch its length via the `stretch factor` key. Same operations for anchors

```
\pgfdeclareshape{ground}{%
```

```
\inheritsavedanchors[from=circle]
\saveddimen{\halfradius}{%
  \pgf@x=.5\pgf@x}
```

Get the multiplied value of `\radius`, which is (`stretch factor`)`\radius`. Using a *saved* dimension it is possible to correctly update anchors when the ground is longer than default

```
\saveddimen{\multipliedradius}{%
  \pgf@x=2\pgf@x% now equal to \radius
  \pgfmathsetlength{\pgf@x}{\pgfkeysvalueof{/tikz/stretch factor}*\pgf@x}
  }
\inheritanchor[from=circle]{center}
\anchor{north}{%
  \centerpoint
  \pgf@xa=\radius%
  \advance\pgf@y by0.8\pgf@xa
}
\anchor{south}{%
  \centerpoint
  \pgf@xa=\radius%
  \advance\pgf@y by-0.8\pgf@xa
}
\anchor{west}{%
  \centerpoint
  \pgf@xa=\multipliedradius%
  \advance\pgf@x by-2.5\pgf@xa
}
\anchor{east}{%
  \centerpoint
  \pgf@xa=\multipliedradius%
  \advance\pgf@x by2.5\pgf@xa
}
\anchor{north west}{%
  \centerpoint
  \pgf@xa=\radius%
  \pgf@xb=\multipliedradius%
  \advance\pgf@y by0.8\pgf@xa
  \advance\pgf@x by-2.5\pgf@xb
}
\anchor{north east}{%
  \centerpoint
  \pgf@xa=\radius%
  \pgf@xb=\multipliedradius%
  \advance\pgf@y by0.8\pgf@xa
  \advance\pgf@x by2.5\pgf@xb
}
\anchor{south west}{%
  \centerpoint
  \pgf@xa=\radius%
  \pgf@xb=\multipliedradius%
  \advance\pgf@y by-0.8\pgf@xa
  \advance\pgf@x by-2.5\pgf@xb
}
\anchor{south east}{%
  \centerpoint
  \pgf@xa=\radius%
  \pgf@xb=\multipliedradius%
```

```
    \advance\pgf@y by-0.8\pgf@xa
    \advance\pgf@x by2.5\pgf@xb
  }
  \anchor{left}{%
    \centerpoint
    \pgf@xa=\radius%
    \pgf@xb=\multipliedradius%
    \advance\pgf@y by0.8\pgf@xa
    \advance\pgf@x by-2.5\pgf@xb
    \advance\pgf@x by2.5\pgf@xa
  }
  \anchor{right}{%
    \centerpoint
    \pgf@xa=\radius%
    \pgf@xb=\multipliedradius%
    \advance\pgf@y by0.8\pgf@xa
    \advance\pgf@x by2.5\pgf@xb
    \advance\pgf@x by-2.5\pgf@xa
  }
```

Define the shape

```
  \backgroundpath{%
      \pgf@x=\radius
      \pgfutil@tempdima=\pgf@x%
```

The horizontal distance between lines has been chosen equal to 5\radius/10 (ten lines every five \radiuses), that is equal to \radius/2

```
      \divide\pgf@x by2\relax
      \pgfutil@tempdimb=\pgf@x%
```

Calculating beginning and end of lines starting from the first one on the left. The $x$ coordinate of the start point of first line is at left of the center point by a distance equal to 2.5(stretch factor)\radius$-$\radius/2. Its $y$ coordinate is 0.8\radius above of the center point. The $x$ coordinate of the end point of the first line is 2.5(stretch factor)\radius on the left of the center point and the $y$ coordinate is 0.8\radius $-$ \radius/2 above of it (considering lines slanted by 45°; this leads to $\tan 45 = 1$)

```
      % start of first line
      \pgf@xb=-2.5\pgfutil@tempdima%
      \pgfmathsetlength{\pgf@xb}{\pgfkeysvalueof{/tikz/stretch factor}*\pgf@xb}
      \advance\pgf@xb by\pgfutil@tempdimb
      \pgf@yb=0.8\pgfutil@tempdima%
      % end of first line
      \pgf@xc=-2.5\pgfutil@tempdima%
      \pgfmathsetlength{\pgf@xc}{\pgfkeysvalueof{/tikz/stretch factor}*\pgf@xc}
      \pgf@yc=\pgf@yb%
      \advance\pgf@yc by-1\pgfutil@tempdimb% tan(45)=1
      % If text is present shift shape to center
      \centerpoint
      \advance\pgf@xb by\pgf@x
      \advance\pgf@yb by\pgf@y
      \advance\pgf@xc by\pgf@x
      \advance\pgf@yc by\pgf@y
      % Save centerpoint in "a" (rectangle center)
      \pgf@xa=\pgf@x
      \pgf@ya=\pgf@y
```

Draw the sloped lines starting from upper left edge. This can be divided in 3 phases. Phase 1: I know start and end point of the lines.

1:

```
\pgfpathmoveto{\pgfpoint{\pgf@xb}{\pgf@yb}}%
\pgfpathlineto{\pgfpoint{\pgf@xc}{\pgf@yc}}%
```

2: shift the $x$ coordinate on the right by `\radius`/2 and the $y$ coordinate below by the same amount (again $\tan 45 = 1$) and then draw

```
\advance\pgf@xb by\pgfutil@tempdimb
\advance\pgf@yc by-\pgfutil@tempdimb
\pgfpathmoveto{\pgfpoint{\pgf@xb}{\pgf@yb}}%
\pgfpathlineto{\pgfpoint{\pgf@xc}{\pgf@yc}}%
```

3: same as 2

```
\advance\pgf@xb by\pgfutil@tempdimb
\advance\pgf@yc by-\pgfutil@tempdimb
\pgfpathmoveto{\pgfpoint{\pgf@xb}{\pgf@yb}}%
\pgfpathlineto{\pgfpoint{\pgf@xc}{\pgf@yc}}%
```

Phase 2: I know the start point and lenght of the lines, so I can calculate the end point. The number of lines depends on the value of `stretch factor`. If this is equal to 1, the number of lines in all phases is equal to 13. If `stretch factor` is not equal to 1 the number of lines in phase 2 is equal to 10(`stretch factor`) − 3.

4:

```
\advance\pgf@xb by\pgfutil@tempdimb
\pgf@xc=\pgf@xb%
\advance\pgf@xc by-1.6\pgfutil@tempdima
\pgf@yc=\pgf@yb%
\advance\pgf@yc by-1.6\pgfutil@tempdima
\pgfpathmoveto{\pgfpoint{\pgf@xb}{\pgf@yb}}%
\pgfpathlineto{\pgfpoint{\pgf@xc}{\pgf@yc}}%
```

5 to $n$: create a new macro called `\mec@centrallines` to store the number of lines to be drawn in phase 2 (minus one, the number 4, that is drawn separately). Then use a `\foreach` statement to draw in a compact way all of the lines. Since the cycle creates a group, transformations on coordinates need to be preceeded by `\global`, otherwise the effect would be lost at the end of each cycle

```
\pgfmathsetmacro{\mec@centrallines}{%
  10*\pgfkeysvalueof{/tikz/stretch factor}-4}
\foreach \x in {1,...,\mec@centrallines}{%
  \global\advance\pgf@xb by\pgfutil@tempdimb
  \global\advance\pgf@xc by\pgfutil@tempdimb
  \pgfpathmoveto{\pgfpoint{\pgf@xb}{\pgf@yb}}%
  \pgfpathlineto{\pgfpoint{\pgf@xc}{\pgf@yc}}%
}
```

Phase 3: I know the end point of the lines because it lies on the right margin of the shape.

$n + 1$:

```
\advance\pgf@xc by\pgfutil@tempdimb
\pgf@xb=2.5\pgfutil@tempdima%
\pgfmathsetlength{\pgf@xb}{\pgfkeysvalueof{/tikz/stretch factor}*\pgf@xb}
\pgf@yb=0.8\pgfutil@tempdima%
\advance\pgf@yb by\pgf@xa
\advance\pgf@yb by-1.6\pgfutil@tempdima
\advance\pgf@yb by\pgf@xb % should be equal to l
\advance\pgf@yb by-\pgf@xc
```

Reconsider the presence of text on redefined coordinates and draw the line

```
      \advance\pgf@xb by\pgf@xa
      \advance\pgf@yb by\pgf@ya
      \pgfpathmoveto{\pgfpoint{\pgf@xb}{\pgf@yb}}%
      \pgfpathlineto{\pgfpoint{\pgf@xc}{\pgf@yc}}%
```

$n+2$:

```
      \advance\pgf@xc by\pgfutil@tempdimb
      \advance\pgf@yb by-\pgfutil@tempdimb
      \pgfpathmoveto{\pgfpoint{\pgf@xb}{\pgf@yb}}%
      \pgfpathlineto{\pgfpoint{\pgf@xc}{\pgf@yc}}%
```

$n+3$

```
      \advance\pgf@xc by\pgfutil@tempdimb
      \advance\pgf@yb by-\pgfutil@tempdimb
      \pgfpathmoveto{\pgfpoint{\pgf@xb}{\pgf@yb}}%
      \pgfpathlineto{\pgfpoint{\pgf@xc}{\pgf@yc}}%
```

Additional phase (4): draw the horizontal line (restoring the default values of points "b" and "c"). The line is lengthened in both directions by 0.3`\pgflinewidth` to avoid edges of the sloped lines to be visible

```
      \pgf@xb=0pt\relax%
      \advance\pgf@xb by-2.5\pgfutil@tempdima
      \pgfmathsetlength{\pgf@xb}{\pgfkeysvalueof{/tikz/stretch factor}*\pgf@xb}
      \advance\pgf@xb by-0.3\pgflinewidth
      \pgf@yb=0pt\relax%
      \advance\pgf@yb by0.8\pgfutil@tempdima
      \pgf@xc=0pt\relax%
      \advance\pgf@xc by2.5\pgfutil@tempdima
      \pgfmathsetlength{\pgf@xc}{\pgfkeysvalueof{/tikz/stretch factor}*\pgf@xc}
      \advance\pgf@xc by0.3\pgflinewidth
      \pgf@yc=0pt\relax%
      \advance\pgf@yc by0.8\pgfutil@tempdima
      % Reconsidering the presence of text on redefined coordinates
      \advance\pgf@xb by\pgf@xa
      \advance\pgf@yb by\pgf@ya
      \advance\pgf@xc by\pgf@xa
      \advance\pgf@yc by\pgf@ya
      % draw the horizontal line
      \pgfpathmoveto{\pgfpoint{\pgf@xb}{\pgf@yb}}%
      \pgfpathlineto{\pgfpoint{\pgf@xc}{\pgf@yc}}%
      \pgfpathclose
  }%
}
```

`ble pendulum b` Starting defining the double pendulum internal variant. Same operations for anchors

```
\pgfdeclareshape{double pendulum b}{%
  \inheritsavedanchors[from=circle]
  \saveddimen{\halfradius}{%
    \pgf@x=.5\pgf@x}
  \inheritanchor[from=circle]{center}
  \anchor{north}{%
    \centerpoint
    \pgf@xa=\radius%
    \pgf@xb=\halfradius%
    \advance\pgf@y by\pgf@xa
```

```
    \advance\pgf@y by\pgf@xb
  }
  \anchor{north east}{%
    \centerpoint
    \pgf@xa=\radius%
    \pgf@xb=\halfradius%
    \advance\pgf@x by\pgf@xa
    \advance\pgf@x by\pgf@xb
    \advance\pgf@y by\pgf@xa
    \advance\pgf@y by\pgf@xb
  }
  \anchor{north west}{%
    \centerpoint
    \pgf@xa=\radius%
    \pgf@xb=\halfradius%
    \advance\pgf@x by-\pgf@xa
    \advance\pgf@x by-\pgf@xb
    \advance\pgf@y by\pgf@xa
    \advance\pgf@y by\pgf@xb
  }
  \anchor{south}{%
    \centerpoint
    \pgf@xa=\radius%
    \pgf@xb=\halfradius%
    \advance\pgf@y by-\pgf@xa
    \advance\pgf@y by-\pgf@xb
  }
  \anchor{south east}{%
    \centerpoint
    \pgf@xa=\radius%
    \pgf@xb=\halfradius%
    \advance\pgf@x by\pgf@xa
    \advance\pgf@x by\pgf@xb
    \advance\pgf@y by-\pgf@xa
    \advance\pgf@y by-\pgf@xb
  }
  \anchor{south west}{%
    \centerpoint
    \pgf@xa=\radius%
    \pgf@xb=\halfradius%
    \advance\pgf@x by-\pgf@xa
    \advance\pgf@x by-\pgf@xb
    \advance\pgf@y by-\pgf@xa
    \advance\pgf@y by-\pgf@xb
  }
  \anchor{west}{%
    \centerpoint
    \pgf@xa=\radius%
    \advance\pgf@x by-\pgf@xa
  }
  \anchor{east}{%
    \centerpoint
    \pgf@xa=\radius%
    \advance\pgf@x by\pgf@xa
  }
  \anchor{north east center}{%
```

```
    \centerpoint
    \pgf@xa=\radius%
    \advance\pgf@x by\pgf@xa
    \advance\pgf@y by\pgf@xa
  }
  \anchor{north west center}{%
    \centerpoint
    \pgf@xa=\radius%
    \advance\pgf@x by-\pgf@xa
    \advance\pgf@y by\pgf@xa
  }
  \anchor{south east center}{%
    \centerpoint
    \pgf@xa=\radius%
    \advance\pgf@x by\pgf@xa
    \advance\pgf@y by-\pgf@xa
  }
  \anchor{south west center}{%
    \centerpoint
    \pgf@xa=\radius%
    \advance\pgf@x by-\pgf@xa
    \advance\pgf@y by-\pgf@xa
  }
```

Even if it shouldn't be necessary, define the `bottom` anchor, which is used to correctly place the ground

```
  \anchor{bottom}{%
    \centerpoint
    \pgf@xa=\radius%
    \pgf@xb=\halfradius%
    \advance\pgf@y by-\pgf@xa
    \advance\pgf@y by-\pgf@xb
  }
```

Define the shape

```
  \backgroundpath{%
      \pgf@x=\radius
      \pgfutil@tempdima=\pgf@x%
      \pgfutil@tempdimb=\halfradius%
```

Define the rectangle box upper right corner "b" and lower left corner "c". The lenght of one side is 3`\radius`

```
      \pgf@xb=\pgfutil@tempdima%
      \advance\pgf@xb by\pgfutil@tempdimb
      \pgf@yb=\pgfutil@tempdima%
      \advance\pgf@yb by\pgfutil@tempdimb
      \pgf@xc=-\pgfutil@tempdima%
      \advance\pgf@xc by-\pgfutil@tempdimb
      \pgf@yc=-\pgfutil@tempdima%
      \advance\pgf@yc by-\pgfutil@tempdimb
      % If text is present shift shape to center
      \centerpoint
      \advance\pgf@xb by\pgf@x
      \advance\pgf@yb by\pgf@y
      \advance\pgf@xc by\pgf@x
      \advance\pgf@yc by\pgf@y
      % Save centerpoint in "a" (top rectangle centerpoint)
```

```
        \pgf@xa=\pgf@x
        \pgf@ya=\pgf@y
```

Just like the double pendulum, the shape is made of multiple paths so it is necessary to create another unique path that will be filled correctly when needed (patterns are working too)

```
        \tikz@mode
        \iftikz@mode@fill
        \advance\pgf@xb by-\pgfutil@tempdimb
        \advance\pgf@xc by\pgfutil@tempdimb
        \pgfpathmoveto{\pgfpoint{\pgf@xb}{\pgf@yb}}
        \pgfpathlineto{\pgfpoint{\pgf@xb}{\pgf@yc}}
        \pgfpathlineto{\pgfpoint{\pgf@xc}{\pgf@yc}}
        \pgfpathlineto{\pgfpoint{\pgf@xc}{\pgf@yb}}
        \pgfpathclose
        \pgfusepath{fill}
        \advance\pgf@xb by\pgfutil@tempdimb
        \advance\pgf@xc by-\pgfutil@tempdimb
        \fi
```

Draw upper and lower sides of rectangle box starting from upper side

```
        \pgfpathmoveto{\pgfpoint{\pgf@xb}{\pgf@yb}}%
        \pgfpathlineto{\pgfpoint{\pgf@xc}{\pgf@yb}}%
        \pgfpathmoveto{\pgfpoint{\pgf@xb}{\pgf@yc}}%
        \pgfpathlineto{\pgfpoint{\pgf@xc}{\pgf@yc}}%
```

Draw left and right side of the inner box shifting “b” to upper right inner corner and “c” to lower left inner corner

```
        \advance\pgf@xb by-\pgfutil@tempdimb
        \advance\pgf@yb by-\pgfutil@tempdima
        %\advance\pgf@yb by-.5\pgflinewidth
        \advance\pgf@xc by\pgfutil@tempdimb
        \advance\pgf@yc by\pgfutil@tempdima
        %\advance\pgf@yb by.5\pgflinewidth
        \pgfpathmoveto{\pgfpoint{\pgf@xb}{\pgf@yb}}
        \pgfpathlineto{\pgfpoint{\pgf@xb}{\pgf@yc}}
        \pgfpathmoveto{\pgfpoint{\pgf@xc}{\pgf@yc}}
        \pgfpathlineto{\pgfpoint{\pgf@xc}{\pgf@yb}}
```

Transform coordinates to fit the circle centers and draw them

```
        \advance\pgf@yb by\pgfutil@tempdimb
        \advance\pgf@yc by-\pgfutil@tempdimb
        \edef\pgf@marshal{%
            \noexpand\pgfpathcircle{%
                \noexpand\pgfqpoint{\the\pgf@xb}{\the\pgf@yb}}
                {\the\pgfutil@tempdimb}%
            \noexpand\pgfpathcircle{%
                \noexpand\pgfqpoint{\the\pgf@xb}{\the\pgf@yc}}
                {\the\pgfutil@tempdimb}%
            \noexpand\pgfpathcircle{%
                \noexpand\pgfqpoint{\the\pgf@xc}{\the\pgf@yc}}
                {\the\pgfutil@tempdimb}%
            \noexpand\pgfpathcircle{%
                \noexpand\pgfqpoint{\the\pgf@xc}{\the\pgf@yb}}
                {\the\pgfutil@tempdimb}%
        }\pgf@marshal
    }%
```

```
1133 }
```