1 3D Tools

Ti*k*Z Library 3dtools

```
\usetikzlibrary{3dtools} % LaTeX and plain TeX
\usetikzlibrary[3dtools] % ConTeXt
```

This library provides additional tools to create 3d-like pictures. It is a collection of reasonably working tools, which however is not streamlined, and may be subject to substantial changes if the library ever happens to get further developed or published.

Ti*k*Z has the `3d` and `tpp` libraries which deal with the projections of three-dimensional drawings. In addition there exist excellent packages like `tikz-3dplot`. The purpose of this library is to provide some means to manipulate the coordinates. It supports linear combinations of vectors, vector products and scalar products.
Note: Hopefully this library is only temporary and its contents will be absorbed in slightly extended versions of the `3d` and `calc` libraries. The cleanest way will be to record a screen depth when "saving" a coordinate with Ti*k*Z. Some limited support of such a functionality is provided in this library, see section 1.4. However, a full-fledged realization would require changes at the level of `tikz.code.tex` and can only be done consistently if the maintainer(s) of Ti*k*Z make some fundamental changes. Even then dealing with node anchors and so on, which so far are intrinsically two-dimensional (see, however, https://tex.stackexchange.com/q/516316). Note also that it is quite conceivable that the viewers in the future will be able to achieve 3d ordering, so, in a way, recording the screen depth (see the `screendepth` function below) will become almost mandatory at a given point.
Filing a bug report or placing a feature request. This library is currently hosted under `https://github.com/marmotghost/tikz-3dtools`. The author is also active at the *noncommercial* Q & A site `https://topanswers.xyz/tex`, which offers, apart from the possibility of asking questions and browsing through the posts a chat in which one can discuss problems.
Why is this library not on CTAN? First of all, ideally this library is only temporary. Secondly, the author does not have a law degree, and finds it very hard to select the appropriate license for submitting the library to CTAN. The library is written as a fun project, everyone is free to use it, and trying to understand all the legal stuff that seems to be required to perform a successful submission to CTAN is not the kind of fun the author is after. This is not to say that CTAN does not have a point in requiring all these data.

1.1 Coordinate computations

The `3dtools` library has some options and styles for coordinate computations and manipulations.

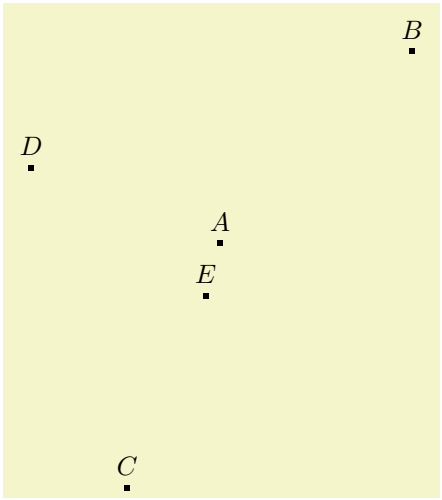
`/tikz/3d parse` (no value)

Parses an expression and inserts the result in form of a coordinate.

`/tikz/3d coordinate` (no value)

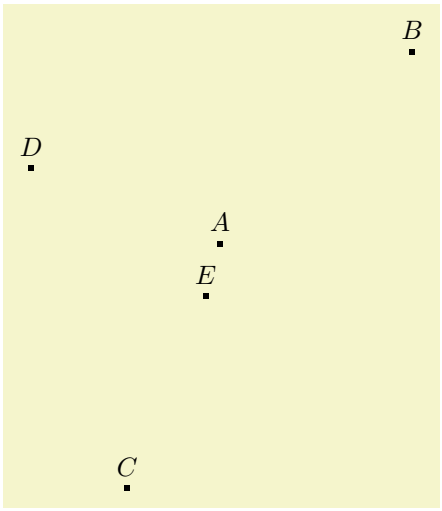
Allow one to define a 3d coordinate from other coordinates.

Both keys support both symbolic and explicit coordinates.




```
\begin{tikzpicture}
 \path (1,2,3) coordinate (A)
  (2,3,-1) coordinate (B)
  (-1,-2,1) coordinate (C)
  [3d parse={0.25*(1,2,3)x(B)}]
  coordinate(D)
  [3d parse={0.25*(C)x(B)}]
  coordinate(E);
 \path foreach \X in {A,...,E}
 {(\X) node[fill,inner sep=1pt,
 label=above:$\X$]{}};
\end{tikzpicture}
```

Notice that, as of now, only the syntax `\path (1,2,3) coordinate (A);` works, i.e. `\coordinate (A) at (1,2,3);` does *not* work, but leads to error messages.




```
\begin{tikzpicture}
 \path (1,2,3) coordinate (A)
  (2,3,-1) coordinate (B)
  (-1,-2,1) coordinate (C)
  [3d coordinate={(D)=0.25*(1,2,3)x(B)},
  3d coordinate={(E)=0.25*(C)x(B)},
  3d coordinate={(F)=(A)-(B)}];
 \path foreach \X in {A,...,E}
 {(\X) node[fill,inner sep=1pt,
 label=above:$\X$]{}};
\end{tikzpicture}
```

The actual parsings are done by the function `\pgfmathtdparse` that allows one to parse 3d expressions. The supported vector operations are `+` (addition $+$), `-` (subtraction $-$), `*` (multiplication of the vector by a scalar), `x` (vector product $\times$) and `o` (scalar product).

`\pgfmathtdparse{`$\langle x \rangle$`}`

Parses 3d expressions.

`TDx("`*vector*`")`

Yields the x-component of a 3d expression.

`TDy("`*vector*`")`

Yields the y-component of a 3d expression.

`TDz("vector")`

Yields the z-component of a 3d expression.

`screendepth("vector")`

Yields distance a coordinate is above (positive) or below (negative) the screen. The values are only really meaningful if the user has installed some reasonable view. The larger the screen depth of a point is, the closer is the point to the observer. The function reconstructs the 3-bein[1] from lengths like `\textbackslash pgf@xy` and so on, so the function is independent of the tool that is employed to install a view (cf. section 1.2). The screen depth is crucial to decide which objects are in front of other objects.

In order to pretty-print the result one may want to use `\pgfmathprintvector`, and use the math function `TD` for parsing.

`\pgfmathprintvector{`$\langle x \rangle$`}`

Pretty-prints vectors.

$0.2\,\vec{A} - 0.3\,\vec{B} + 0.6\,\vec{C} = (-1, -1.7, 1.5)$

```
\pgfmathparse{TD("0.2*(A)
-0.3*(B)+0.6*(C)")}%
$0.2\,\vec A-0.3\,\vec B+0.6\,\vec C
=(\pgfmathprintvector\pgfmathresult)$
```

The alert reader may wonder why this works, i.e. how would Ti*k*Z "know" what the coordinates A, B and C are. It works because the coordinates in Ti*k*Z are global, so they get remembered from the above example.

Warning. The expressions that are used in the coordinates will only be evaluated when they are retrieved. So, if you use, say, random numbers, you will get each time a *different* result. This is because this library is working with `\tikz@dcl@coord@`*coord*, where *coord* is the name of the coordinate, say `A`. This is the string with which the coordinate was generatated, e.g. `(1,2,3)`. However, the coordinate will always be at the same location. So you may want to avoid using functions that change their values when declaring coordinates that get used later in coordinate calculations.

$\vec{R} = (0.96, 0.69, 0.82)$

$\vec{R} = (0.2, 0.4, 0.92)$

```
\begin{tikzpicture}
 \path[overlay] (0.1+rnd,0.1+rnd,0.1+rnd)
 coordinate (R);
 \def\ppv{\pgfmathprintvector\pgfmathresult}
 \draw[red] (R) circle[radius=0.1];
 \node at (0,1)
  {\pgfmathparse{TD("(R)")}%
  $\vec R=(\ppv)$};
 \draw[blue] (R) circle[radius=0.2];
 \node at (0,0)
  {\pgfmathparse{TD("(R)")}%
  $\vec R=(\ppv)$};
\end{tikzpicture}
```

[1] A 3-bein or dreibein is a German word that stands for a local frame, its literal translation is something like three-leg. Like the term eigenvector this is a foreign word that, to the best of my knowledge, has no commonly used English counterpart.

Its main usage is to strip off unnecessary zeros, which emerge since the internal computations are largely done with the `fpu` library.

$(1,0,0)^T \times (0,1,0)^T = (0,0,1)^T$

```
\pgfmathparse{TD("(1,0,0)x(0,1,0)")}%
$(1,0,0)^T\times(0,1,0)^T=
(\pgfmathprintvector\pgfmathresult)^T$
```

$\vec{A}\cdot\vec{B}=5$

```
\pgfmathparse{TD("(A)o(B)")}%
$\vec A\cdot \vec B=
\pgfmathprintnumber\pgfmathresult$
```

Notice that, as of now, the only purpose of brackets `(...)` is to delimit vectors. Further, the addition `+` and subtraction `-` have a *higher* precedence than vector products `x` and scalar products `o`. That is, `(A)+(B)o(C)` gets interpreted as $(\vec{A}+\vec{B})\cdot\vec{C}$, and `(A)+(B)x(C)` as $(\vec{A}+\vec{B})\times\vec{C}$.

$(\vec{A}+\vec{B})\cdot\vec{C}=-11$

```
\pgfmathparse{TD("(A)+(B)o(C)")}%
$(\vec A+\vec B)\cdot\vec C=
\pgfmathprintnumber\pgfmathresult$
```

$(\vec{A}+\vec{B})\times\vec{C}=(9,-5,-1)$

```
\pgfmathparse{TD("(A)+(B)x(C)")}%
$(\vec A+\vec B)\times\vec C=
(\pgfmathprintvector\pgfmathresult)$
```

Moreover, any expression can only have either one `o` or one `x`, or none of these. Expressions with more of these can be accidentally right.

`axisangles("`*vector*`")`

Yields the the rotation angles that transforms the vector in the z-axis. Since an axis has a residual rotation symmetry, namely the rotation around this axis, only two angles are required, and thus returned. In the conventions of section 1.2, these are the angles ϕ and ψ. It corresponds to the macro `\tdplotgetpolarcoords{`$\langle x\rangle$`}{`$\langle y\rangle$`}{`$\langle z\rangle$`}` from the `tikz-3dplot` package.

$\sphericalangle(\vec{A})=-116.57,-36.7$

```
\pgfmathparse{axisangles("(A)")}%
$\sphericalangle(\vec A)=
\pgfmathprintvector\pgfmathresult$
```

Computations in 3d sometimes involve projections of a point on a line or a plane. In 2d, projections of a point on a line can be conveniently done with the `calc` library, but this does in general not yield the correct projection in 3d. In order to make these projections more user friendly, `3dtools` offers the possibility to define extended objects such as lines or planes. This is the same concept as what is offered by plain Ti*k*Z, where the user can name coordinates and nodes, and, if the `intersections` library is loaded, also paths.

`/tikz/3d/plane through=`*point 1* and *point 2* and *point 3* named *name* (no default)

Defines a plane of name *name* by the requirement that it goes through the three specified points. Internally the plane is stored in terms of its normal and a point it goes through.

`/tikz/3d/plane with normal`=*normal* through *point 1* named *name* (no default)

Defines a plane of name *name* by the requirement that it goes through *point 1* and has the normal *normal*. Internally the plane is stored in terms of this normal and point.

`/tikz/3d/line through`=*point 1* and *point 2* named *name* (no default)

Defines a line of name *name* by the requirement that it goes through *point 1* and *point 1*.

`/tikz/3d/intersection of`=*object 1* with *object 2* (no default)

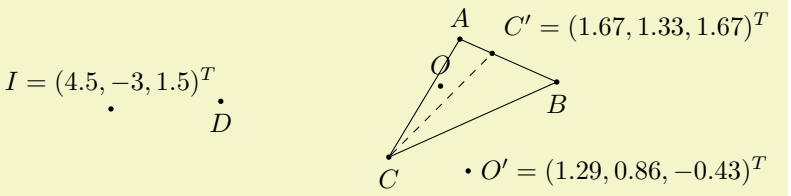
Computes the intersection of a named object *object 1* with another object *object 2*. Currently only intersections of planes with lines are supported.

These extended objects can be used in projections.

`/tikz/3d/project`=*point* on *named object* (no default)

Computes the projection of a point on some extended object, and inserts the projection in the path.

The following code example illustrates the usage. It also makes use of the `install view` key, which we describe in section 1.2.



```
\begin{tikzpicture}[3d/install view={phi=110,psi=0,theta=60},
dot/.style={circle,inner sep=0pt,
minimum size=2pt,fill}]
 \draw[every coordinate node/.append style={dot}]
   (2,1,2) coordinate[label=above:{$A$}] (A) --
   (1,2,1) coordinate[label=below:{$B$}] (B) --
   (2,0,0) coordinate[label=below:{$C$}] (C) -- cycle
   (0,0,0) coordinate[label=above:{$O$}] (O)
   (3,-2,1) coordinate[label=below:{$D$}] (D);
 \path[3d/plane through={(A) and (B) and (C) named pABC},
3d/plane with normal={(1,1,1) through (C) named ptwo},
3d/line through={(A) and (B) named lAB},
3d/line through={(O) and (D) named lOD}];
 % project point on plane
 \path[3d/project={(O) on pABC}] coordinate (O');
 % project point on line
 \path[3d/project={(C) on lAB}] coordinate (C');
 % intersection of plane and line
 \path[3d/intersection of={lOD with pABC}] coordinate (I);
 \draw[dashed] (C) -- (C')
  coordinate[dot,label=above right:{$C'=\pgfmathparse{TD("(C')")}%
  (\pgfmathprintvector\pgfmathresult)^T$}];
 \path (O') coordinate[dot,label=right:{$O'=\pgfmathparse{TD("(O')")}%
  (\pgfmathprintvector\pgfmathresult)^T$}];
 \path (I) coordinate[dot,label=above:{$I=\pgfmathparse{TD("(I)")}%
  (\pgfmathprintvector\pgfmathresult)^T
  $}];
\end{tikzpicture}
```

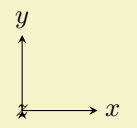
1.2 Orthonormal projections

The 3dtools library can be used together with the tikz-3dplot package and/or the tpp library. It also has its own means to install orthonormal projections. Orthonormal projections emerge from subjecting 3-dimensional vectors to orthogonal transformations and projecting them to 2 dimensions. They are not to be confused with the perspective projections, which are more realistic and supported by the tpp library. Orthonormal projections may be thought of a limit of perspective projections at large distances, where large means that the distance of the observer is much larger than the dimensions of the objects that get depicted.

`/tikz/3d/install view` (no value)

Installs a 3d orthonormal projection.

The initial projection is such that x is right an y is up, as if we had no third direction.




```
\begin{tikzpicture}[3d/install view]
 \draw[-stealth] (0,0,0) -- (1,0,0)
  node[pos=1.2] {$x$};
 \draw[-stealth] (0,0,0) -- (0,1,0)
  node[pos=1.2] {$y$};
 \draw[-stealth] (0,0,0) -- (0,0,1)
  node[pos=1.2] {$z$};
\end{tikzpicture}
```

The 3d-like pictures emerge by rotating the view. The conventions for the parametrization of the orthogonal rotations in terms of three rotation angles ϕ, ψ

and θ are

$$O(\phi,\psi,\theta) = \begin{pmatrix} c_\phi\, c_\psi & s_\phi\, c_\psi & -s_\psi \\ c_\phi\, s_\psi\, s_\theta - s_\phi\, c_\theta & s_\phi\, s_\psi\, s_\theta + c_\phi\, c_\theta & c_\psi\, s_\theta \\ c_\phi\, s_\psi\, c_\theta + s_\phi\, s_\theta & s_\phi\, s_\psi\, c_\theta - c_\phi\, s_\theta & c_\psi\, c_\theta \end{pmatrix} . \tag{1}$$

Here, $c_\phi := \cos\phi$, $s_\phi := \sin\phi$ and so on.

`/tikz/3d/phi` (initially `0`)

3d rotation angle.

`/tikz/3d/psi` (initially `0`)

3d rotation angle.

`/tikz/3d/theta` (initially `0`)

3d rotation angle.

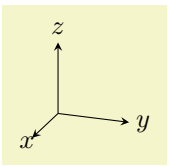
The rotation angles can be used to define the view. The conventions are chosen in such a way that they resemble those of the `tikz-3dplot` package, which gets widely used. This matrix can be written as

$$O(\phi,\psi,\theta) = R_x(\theta)\cdot R_y(\psi)\cdot R_z(\phi)\ ,$$

where

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) \\ 0 & -\sin(\theta) & \cos(\theta) \end{pmatrix}\ , \quad R_y(\psi) = \begin{pmatrix} \cos(\psi) & 0 & -\sin(\psi) \\ 0 & 1 & 0 \\ \sin(\psi) & 0 & \cos(\psi) \end{pmatrix}\ ,$$
$$R_z(\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & \sin(\phi) \\ 0 & -\sin(\phi) & \cos(\phi) \end{pmatrix}$$

are rotations about the x, y and z axis, respectively. For $\psi = 0$, $O(\phi = \phi_d, \psi = 0, \theta = \theta_d) = R^d(\theta_d, \phi_d)$ from the `tikz-3dplot` package. Note, however, that there seems to be an inconsistency in equation (2.1) of that package.[2]




```
\begin{tikzpicture}[3d/install view={phi=110,psi=0,theta=70}]
 \draw[-stealth] (0,0,0) -- (1,0,0)
  node[pos=1.2] {$x$};
 \draw[-stealth] (0,0,0) -- (0,1,0)
  node[pos=1.2] {$y$};
 \draw[-stealth] (0,0,0) -- (0,0,1)
  node[pos=1.2] {$z$};
\end{tikzpicture}
```

`/tikz/3d/define orthonormal dreibein=`⟨*options*⟩ (no default, initially `empty`)

Defines a local dreibein from three input points. The initial choice of these points are `A=(A)`, `B=(B)`, `C=(C)`, and the initial choice for the basis vectors is `ex=(ex)`, `ey=(ey)`, `ez=(ez)`. In terms of these points, (ex) is parallel to $(B)-(A)$, (ey) is in the plane that contains (A), (B) and (C), and is orthogonal to (ex), and (ez) is orthogonal to (ex) and (ey). All the basis vectors are normalized to length 1.

[2] I do not know how to contact the author.

1.3 Visibility considerations

When drawing objects in 3d, some parts are "visible", i.e. in the foreground, while other parts are "hidden", i.e. in the background. In what follows, we discuss some basic means that allow one to determine whether some stretch is visible or hidden. Very often this amounts to find critical values of some parameter(s) (such as angles) at which some curve changes from visible to hidden, or vice versa. One of the major players in this game is the normal on the screen, which is given by the last row of the orthogonal matrix O of (1). For $\psi = 0$, this normal is also the normal in the so-called "main" coordinates of `tikz-3dplot` with the identifications $\theta =$ `\tdplotmaintheta` and $\phi =$ `\tdplotmainphi`.

Here is an example. Suppose we wish to draw a latitude circle on a sphere for $\psi = 0$. Since the system has a rotational symmetry around the z axis, we can set $\phi = 0$ as well. The normal on the screen then becomes

$$\vec{n}_{\text{screen}} = \big(0, -\sin(\theta), \cos(\theta)\big)^T \,.$$

The latitude circle can be parametrized as

$$\vec{\gamma}_{\text{lat}}(\beta) = \big(r\,\cos(\beta), r\,\sin(\beta), h\big)^T \,,$$

where $r = R\,\cos(\lambda)$ and $h = R\,\sin(\lambda)$ with R being the radius of the sphere and λ the latitude angle of the latitude circle. The critical angles β_{crit} at which the curve transitions from hidden to visible are then given by the solutions of

$$\vec{n}_{\text{screen}} \cdot \vec{\gamma}_{\text{lat}}(\beta_{\text{crit}}) = -r\,\sin(\beta)\,\sin(\theta) + h\,\cos(\theta) \stackrel{!}{=} 0 \,.$$

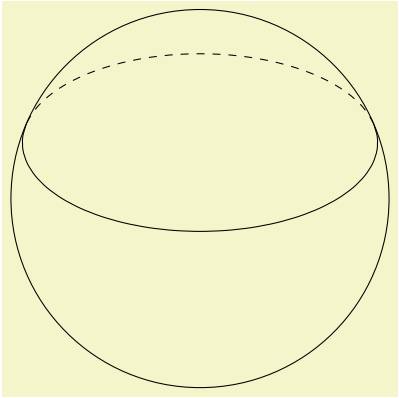
Since $-1 \le \sin(\beta) \le 1$, this equation only has solutions if

$$\left|\frac{h}{r}\cot(\theta)\right| \le 1 \,. \tag{2}$$

The solutions are then given by

$$\beta^{(1)}_{\text{crit}} = \arcsin\left(\frac{h}{r}\cot(\theta)\right) \quad \text{and} \quad \beta^{(2)}_{\text{crit}} = 180 - \beta^{(1)}_{\text{crit}} \,,$$

and $\beta^{(1)}_{\text{crit}} = \beta^{(2)}_{\text{crit}}$ if the inequality (2) is saturated. The following code example illustrates this. Notice that it could be made more concise with the `3d` library.



```
\begin{tikzpicture}[declare function={%
R=2.5;lambda=20;theta=60;
r=R*cos(lambda);h=R*sin(lambda);
betacrit=asin(h/r*cot(theta));}]
 \draw (0,0) circle[radius=R];
 \begin{scope}[smooth,
 3d/install view={phi=0,psi=0,theta=theta}]
  \draw[dashed]
   plot[variable=\t,
   domain=betacrit:180-betacrit]
   ({r*cos(\t)},{r*sin(\t)},h);
  \draw
   plot[variable=\t,
   domain=betacrit:-180-betacrit]
   ({r*cos(\t)},{r*sin(\t)},h);
 \end{scope}
\end{tikzpicture}
```

Another type of visibility considerations concerns the contours of shapes. To be specific, consider a surface parametrized by

$$\vec{f}(u,v) = \begin{pmatrix} u \\ R(u)\,\cos(v) \\ R(u)\,\sin(v) \end{pmatrix} \tag{3}$$

with some function $R(u)$. What is the contour of an orthonormal projection of the surface on the screen? It is the boundary between the hidden and visible parts. What distinguishes the visible from the hidden parts? It is the projection of the normal of the surface on the screen, $F(u,v) := n_S(u,v) \cdot n_{\text{screen}}$. The hidden and visible stretches are separated by the zeros of this projection. In the case of our surface (3),

$$\vec{n}_S(u,v) = \vec{\nabla}_u f(u,v) \times \vec{\nabla}_v f(u,v) = R(u) \begin{pmatrix} -R'(u) \\ \cos(v) \\ \sin(v) \end{pmatrix} , \tag{4}$$

and for $\psi = 0$

$$\vec{n}_{\text{screen}} = \begin{pmatrix} \sin(\theta)\,\sin(\phi) \\ -\sin(\theta)\,\cos(\phi) \\ \cos(\theta) \end{pmatrix} . \tag{5}$$

So we need to find the zeros of

$$\begin{aligned} F(u,v) =& -\cos(\theta)\,\sin(v) + \sin(\theta)\,\cos(\phi)\,\cos(v) + R'(u)\,\sin(\theta)\,\sin(\phi) \\ =:& \; a\,\sin(v) + b\,\cos(v) + c\,. \end{aligned} \tag{6}$$

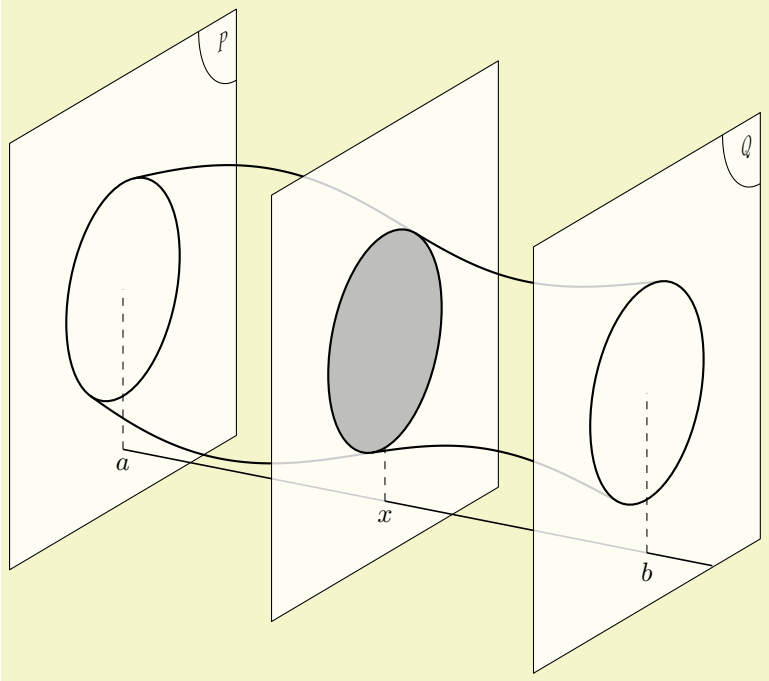
This equation can be solved for $x := \sin(v)$ using $\cos(v) = \pm\sqrt{1-\sin^2(v)} = \pm\sqrt{1-x^2}$. It is clear that, since $\sin(v)$ is bounded (and so is $\cos(v)$), this equation does not always have a solution. In particular, if $R'(u)$ is too large, there might not be a solution. This just means that for a given u all points are hidden or visible. This happens e.g. when you look at a cone from the top or bottom. A more advanced code could deal with these cases, the one presented below (which is taken from topanswers.xyz) does not. Note also that these are local visibility conditions, a seemingly visible stretch can always be covered by some other stretch that is a finite distance away in u direction.

However, for "reasonable" configurations, there are two solutions of the quadratic equation, as they should, they correspond to the upper and lower contour in the code example below. They are a bit unilluminating and given by

$$[\sin(v)]_{1,2} = -\frac{a\sqrt{a^2+b^2-c^2} \pm b\,c}{a^2+b^2} , \tag{7a}$$

$$[\cos(v)]_{1,2} = -\frac{b\sqrt{a^2+b^2-c^2} \mp a\,c}{a^2+b^2} , \tag{7b}$$

where a, b and c are implicitly defined in (6). Hence, for "reasonable" configurations, it is straightforward to plot the contour of the surface with LaTeX.


P
Q
a
x
b

```
\begin{tikzpicture}[declare function={phi=30;theta=70;},
3d/install view={phi=phi,psi=0,theta=theta},
line cap=butt,line join=round,
declare function={%
  R(\u)=1.5+sin(\u*45)/(2+\u/8);%<- input
  Rprime(\u)=(R(\u+0.01)-R(\u-0.01))/0.02;%<- numerical derivative
  a=cos(phi)*sin(theta);% see equation {eq:ucrit}
  b=-1*cos(theta);%
  c(\u)=sin(theta)*sin(phi)*Rprime(\u);%
      sv1(\u)=-((b*c(\u)+sqrt(a*a+b*b-c(\u)*c(\u))*abs(a))/(a*a+b*b));%
  sv2(\u)=(-(b*c(\u))+sqrt(a*a+b*b-c(\u)*c(\u))*abs(a))/(a*a+b*b);%
  cv1(\u)=-((a*c(\u)+sqrt(a*a+b*b-c(\u)*c(\u))*abs(b))/(a*a+b*b));%
  cv2(\u)=(-(a*c(\u))+sqrt(a*a+b*b-c(\u)*c(\u))*abs(b))/(a*a+b*b);%
},
icircle/.style={thick},
iplane/.style={fill=white,fill opacity=0.8},
iline/.style={semithick},
 extra/.code={},annot/.code={\tikzset{extra/.code={#1}}}]
 \newcommand\DrawIntersectingPlane[2][]{%
  \begin{scope}[canvas is yz plane at x=#2,transform shape,#1]
 \draw[iplane] (3,3) -- (-3,3) -- (-3,-3)-- (3,-3) -- cycle;
 \draw[icircle]  (0,0) circle[radius={R(#2)}];
 \tikzset{extra}
  \end{scope}}
 %
 \DrawIntersectingPlane[annot={%
 \path (2.9,2.9) node[below left,transform shape]{$P$};
\draw (2,3) arc[start angle=180,end angle=270,radius=1];
\draw[dashed] (0,-2.25) node[below,transform shape=false]{$a$} -- (0,0);
}]{0}
 \draw[iline] (0,0,-2.25) -- (4,0,-2.25);
 \draw[thick,smooth,variable=\u,domain=0:4]
 plot  (\u,{R(\u)*cv1(\u)},{R(\u)*sv1(\u)})
 plot  (\u,{R(\u)*cv2(\u)},{R(\u)*sv2(\u)});
 \DrawIntersectingPlane[icircle/.append style={fill=gray,fill opacity=0.5},
   annot={%
\draw[dashed] (0,-2.25) node[below,transform shape=false]{$x$} -- (0,-1.5);
}]{4}
 \draw[iline] (4,0,-2.25) -- (8,0,-2.25);
 \draw[thick,smooth,variable=\u,domain=4:8]
 plot  (\u,{R(\u)*cv1(\u)},{R(\u)*sv1(\u)})
 plot  (\u,{R(\u)*cv2(\u)},{R(\u)*sv2(\u)});
 ;
 \DrawIntersectingPlane[annot={%
 \path (2.9,2.9) node[below left,transform shape]{$Q$};
\draw (2,3) arc[start angle=180,end angle=270,radius=1];
\draw[dashed] (0,-2.25) node[below,transform shape=false]{$b$} -- (0,0);
}]{8}
 \draw[iline] (8,0,-2.25) -- (9,0,-2.25) ;
\end{tikzpicture}
```

There are some special cases that may be of particular interest: $R'(u) = 0$ for a cylinder and $R'(u) = -r/h$ for a cone of height h and base radius r.

1.4 "Physical" coordinates

It is quite conceivable that users may want to use different coordinate systems for defining different coordinates. The following functions aim at supporting this. It should be said, though, that these functions are even more "experimental" than what has been discussed thus far. The main point is that, apart from the "physical"

x and y coordinates, i.e. the coordinates of an point on the screen, one can also keep track of the physical z coordinate, or screen depth, which indicates how far a given point is above ($z > 0$) or below ($z < 0$) the screen. To accomplish this, one needs to swich on the recording of physical components.

`/tikz/3d/record physical components` (no value)

Switches on recording of physical components of coordinates.

The physical components can then be retrieved with various functions.

`TDphysx("`*vector*`")`

Yields the physical x-component of a 3d named coordinate.

`TDphysy("`*vector*`")`

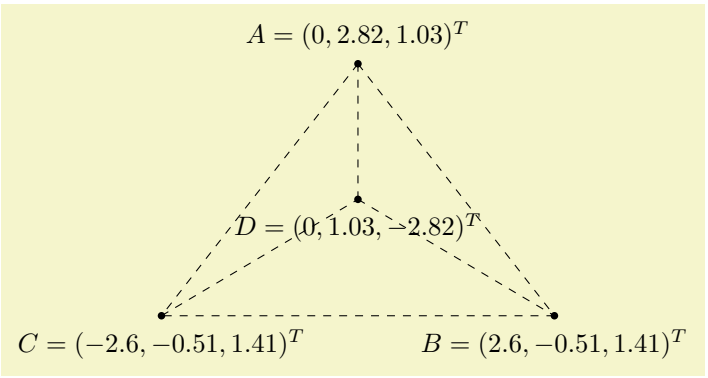
Yields the physical y-component of a 3d named coordinate.

`TDphysz("`*vector*`")`

Yields the physical z-component of a 3d named coordinate.

`TDphys("`*vector*`")`

Yields an array containgin physical x, y and z-components of a 3d named coordinate.




```
\begin{tikzpicture}[3d/record physical components,
dot/.style={circle,inner sep=1pt,fill}]
 \begin{scope}[3d/install view={phi=0,psi=0,theta=70}]
  \path (0,0,3) coordinate (A);
  \path[3d/install view={phi=30}] (3,0,0) coordinate (B);
  \path[3d/install view={phi=150}] (3,0,0) coordinate (C);
  \path[3d/install view={phi=270}] (3,0,0) coordinate (D);
 \end{scope}
 \draw[dashed] foreach \X [remember=\X as \Y (initially D)]
 in {B,C,D}
  {(A) -- (\X) (\Y) -- (\X)
 coordinate[dot,label=below:{$\X=\pgfmathparse{TDphys("\X")}%
 (\pgfmathprintvector\pgfmathresult)^T
 $}]}
 (A) coordinate[dot,label=above:{$A=\pgfmathparse{TDphys("A")}%
 (\pgfmathprintvector\pgfmathresult)^T
 $}];
\end{tikzpicture}
```

Once one has defined physical coordinates, one can use them to compute e.g. distances between coordinates defined in different frames.

$\vec{A}_{\text{phys}} = (0, 2.82, 1.03)$,
$\vec{B}_{\text{phys}} = (2.6, -0.51, 1.41)$,
$d_{\text{phys}}(\vec{A}, \vec{B}) = 4.24$

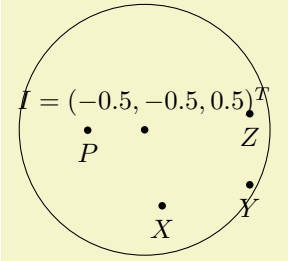
```
$\pgfmathsetmacro{\vecA}{TDphys("A")}%
\pgfmathsetmacro{\vecB}{TDphys("B")}%
\pgfmathsetmacro{\physdist}{%
sqrt(TD("(\vecA)-(\vecB)o(\vecA)-(\vecB)"))}%
\begin{array}{l}
\vec A_\mathrm{phys}=(\pgfmathprintvector\vecA),\\
\vec B_\mathrm{phys}=(\pgfmathprintvector\vecB),\\
d_\mathrm{phys}(\vec A,\vec B)=%
\pgfmathprintnumber\physdist
\end{array}$
```

These are the things that work. However, there are, unfortunately, more than enough things that do not work. They include shifted frames, other coordinate systems such as spherical ones, all coordinates that are defined in `canvas is xy plane at z=0` or relatives, etc.

1.5 Predefined path constructions and pics

`/tikz/3d/circumsphere center=`⟨*options*⟩ (no default, initially `empty`)

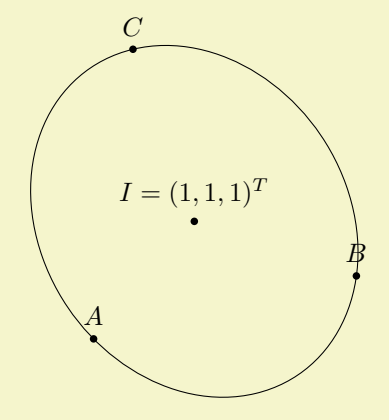
Computes the center of a sphere for a given set of four non-coplanar points. The initial choice of these points are `A=(A)`, `B=(B)`, `C=(C)`, `D=(D)`. The underlying maths can be found at https://topanswers.xyz/tex?q=1233#a1467.




```
\begin{tikzpicture}[dot/.style={circle,inner sep=1pt,fill}]
 \begin{scope}[3d/install view={phi=100,psi=0,theta=70}]
  \path (1,0,0) coordinate (X)
   (0,1,0) coordinate (Y)
   (0,1,1) coordinate (Z)
   (1,-1,1) coordinate (P);
  \path[3d/circumsphere center={A={(X)},B={(Y)},C={(Z)},D={(P)}}]
  coordinate (I);
 \end{scope}
  \pgfmathsetmacro{\csr}{sqrt(TD("(X)-(I)o(X)-(I)"))}
  \draw (I) circle[radius=\csr];
  \path foreach \X in {X,Y,Z,P}
 {(\X) coordinate[dot,label=below:{$\X$}]}
 (I) (I) coordinate[dot,label=above:{$I=\pgfmathparse{TD("(I)")}%
  (\pgfmathprintvector\pgfmathresult)^T$}];
\end{tikzpicture}
```

`/tikz/3d/circumcircle center=`⟨*options*⟩ (no default, initially `empty`)

Computes the center of a circle for a given set of three non-collinear points. The initial choice of these points are `A=(A)`, `B=(B)`, `C=(C)`.




```
\begin{tikzpicture}[declare function={a=3;},
   dot/.style={circle,inner sep=1pt,fill},
   3d/install view={phi=100,psi=0,theta=70}]
  \path (a,0,0) coordinate (A)
   (0,a,0) coordinate (B)
   (0,0,a) coordinate (C);
  \path[3d/circumcircle center]
  coordinate (I);
  \pgfmathsetmacro{\myr}{%
     sqrt(TD("(A)-(I)o(A)-(I)"))}
  \tikzset{3d/define orthonormal dreibein}
  \begin{scope}[x={(ex)},y={(ey)}]
   \draw (I) circle[radius=\myr];
  \end{scope}
  \path foreach \X in {A,B,C}
  {(\X) coordinate[dot,label=above:{$\X$}]}
  (I) (I) coordinate[dot,label=above:{$I=%
  \pgfmathparse{TD("(I)")}%
   (\pgfmathprintvector\pgfmathresult)^T$}];
\end{tikzpicture}
```

`/tikz/pics/3d circle through 3 points`=⟨*options*⟩ (no default, initially `empty`)

Draws a circle through 3 points in 3 dimensions. If the three coordinates are close to linearly dependent, the circle will not be drawn. *This pic will most likely be removed from the documentation and no longer be supported/developed.*

`/tikz/3d circle through 3 points/A` (initially `(1,0,0)`)

First coordinate. Can be either symbolic or explicit. Symbolic coordinates need to be defined via `\path (x,y,z) coordinate (name);`.

`/tikz/3d circle through 3 points/B` (initially `(0,1,0)`)

Second coordinate, like above.

`/tikz/3d circle through 3 points/C` (initially `(0,0,1)`)

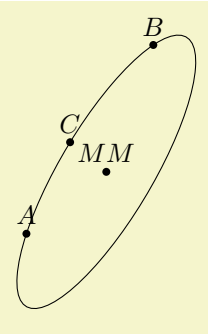
Third coordinate, like above.

`/tikz/3d circle through 3 points/center name` (initially `M`)

Name of the center coordinate that will be derived.

`/tikz/3d circle through 3 points/auxiliary coordinate prefix` (initially `tmp`)

In Ti*k*Z the coordinates are global. The code for the circle is more comprehensible if named coordinates are introduced. Their names will begin with this prefix. Changing the prefix will allow users to avoid overwriting existing coordinates.




```
\begin{tikzpicture}[3d/install view={%
    phi=30,psi=0,theta=70}]
 \foreach \X in {A,B,C}
 {\pgfmathsetmacro{\myx}{3*(rnd-1/2)}
 \pgfmathsetmacro{\myy}{3*(rnd-1/2)}
 \pgfmathsetmacro{\myz}{3*(rnd-1/2)}
 \path (\myx,\myy,\myz) coordinate (\X);}
 \path pic{3d circle through 3 points={%
 A={(A)},B={(B)},C={(C)},center name=MM}};
  \foreach \X in {A,B,C,MM}
  {\fill (\X) circle[radius=1.5pt]
  node[above]{$\X$};}
\end{tikzpicture}
```

`/tikz/pics/3d incircle=`⟨*options*⟩ (no default, initially `empty`)

Inscribes a circle in a triangle in 3 dimensions.

`/tikz/3d incircle/A` (initially `(1,0,0)`)

First coordinate. Can be either symbolic or explicit.

`/tikz/3d incircle/B` (initially `(0,1,0)`)

Second coordinate, like above.

`/tikz/3d incircle/C` (initially `(0,0,1)`)

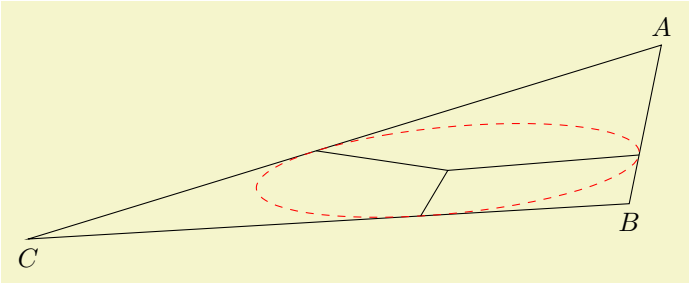
Third coordinate, like above.

`/tikz/3d incircle/center name` (initially `I`)

Name of the center coordinate that will be derived.

`/tikz/3d incircle/auxiliary coordinate prefix` (initially `tmpp`)

In Ti*k*Z the coordinates are global. The code for the circle is more comprehensible if named coordinates are introduced. Their names will begin with this prefix. Changing the prefix will allow users to avoid overwriting existing coordinates.




```
\begin{tikzpicture}[3d/install view={phi=110,psi=0,theta=70}]
 \draw
   (8,5,5) coordinate[label=above:{$A$}] (A) --
   (1,2,0) coordinate[label=below:{$B$}] (B) --
   (5,-5,0) coordinate[label=below:{$C$}] (C) -- cycle;
 \path pic[red,dashed]{3d incircle={%
 A={(A)},B={(B)},C={(C)},center name=I}};
 \draw (I) -- (tmppa) (I) -- (tmppb) (I)  -- (tmppc);
\end{tikzpicture}
```

/tikz/pics/ycylinder (initially empty)

A cylinder in the y-direction. This pic requires the `calc` library. As of now it does only work for psi=0.

/tikz/pics/3d/r (initially 1)

Key for radii, e.g. of cylinders.

/tikz/pics/3d/h (initially 1)

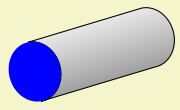
Key for heights, e.g. of cylinders.

/tikz/pics/3d/mantle (initially draw)

Style for cylinder mantle. If no fill option is specified, it will be shaded.

/tikz/pics/3d/top (initially draw)

Style for cylinder top.



```
\begin{tikzpicture}[3d/install view={phi=30,psi=0,theta=80}]
 \pic{ycylinder={r=0.4,h=3,
 top/.style={fill=blue}}};
\end{tikzpicture}
```

To do:

- transform to plane given by three non-degenerate coordinates
- transform to plane given by normal and one point
- maybe layering/visibility

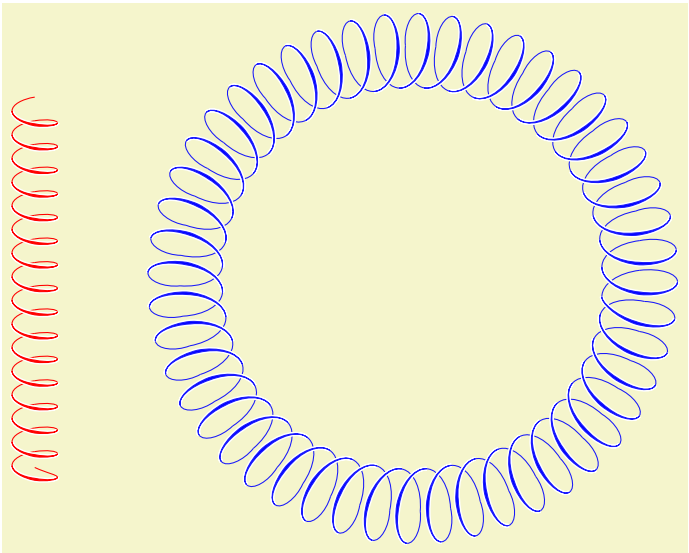
1.6 3D-like decorations

/tikz/decorations/3d complete coil (no value)

3d-like coil where the front is thicker than the back.

/tikz/decorations/3d coil closed (no value)

Indicates that the coil is closed.



```
\begin{tikzpicture}
\draw[decoration={3d coil color=red,aspect=0.35, segment length=3.1mm,
amplitude=3mm,3d complete  coil},
decorate] (0,1) -- (0,6);
\draw[decoration={3d coil color=blue,3d coil opacity=0.9,aspect=0.5,
segment length={2*pi*3cm/50}, amplitude=5mm,3d complete coil,
3d coil closed},
decorate] (5,3.5) circle[radius=3cm];
\end{tikzpicture}
```