1 3D Tools

TikZ 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.

TikZ has the `3d` and `tpp` libraries which deal with the projections of three-dimensional drawings. This library provides some means to manipulate the coordinates. It supports linear combinations of vectors, vector 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 TikZ. However, this would require changes at the level of `tikz.code.tex` and can only be done consistently if the maintainer(s) of TikZ support this. Note also that it is quite conceivable that the viewers will in the future will be able to achieve 3d ordering, so, in a way, recording the screen depth will become almost mandatory at a given point.

1.1 Coordinate computations

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

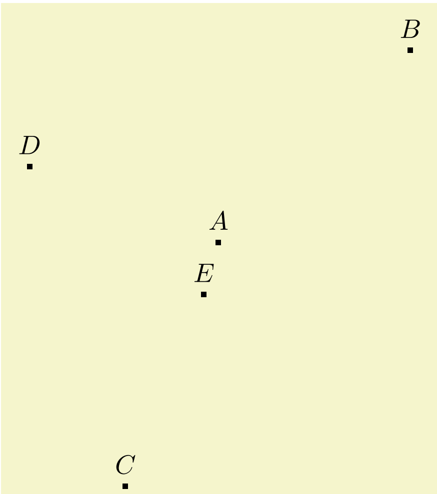
`/tikz/3d parse` (no value)

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

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

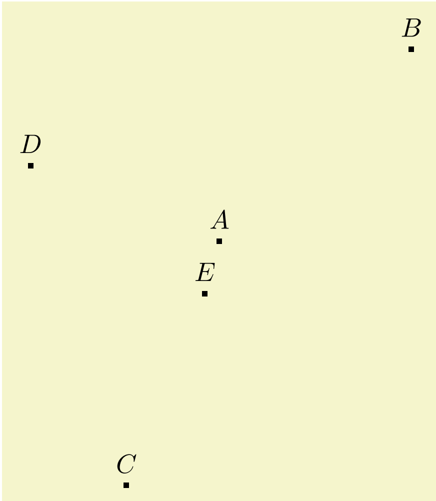
Allow one to define a 3d coordinate from other coordinates.

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.

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.

$\vec{R} = (0.91, 0.5, 2.54 \cdot 10^{-2})$

$\vec{R} = (0.81, 0.29, 0.32)$

```
\begin{tikzpicture}
 \path[overlay] (rnd,rnd,rnd)
 coordinate (R);
 \node at (0,1)
  {\pgfmathparse{TD("(R)")}%
  $\vec R=(\pgfmathprintvector\pgfmathresult)$};
 \node at (0,0)
  {\pgfmathparse{TD("(R)")}%
  $\vec R=(\pgfmathprintvector\pgfmathresult)$};
\end{tikzpicture}
```

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

```
\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{\marg{x}}{\marg{y}}{\marg{z}}` from the `tikz-3dplot` package.

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

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

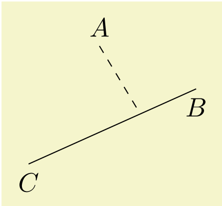
$\sphericalangle(\vec{A}) = -116.57, -36.7$

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

`/tikz/3d/projection of` (no value)

Allows one to compute the projection of a point on a line.

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}]
 \draw
   (2,1,2) coordinate[label=above:{$A$}] (A)
   (1,2,1) coordinate[label=below:{$B$}] (B) --
   (2,0,0) coordinate[label=below:{$C$}] (C);
 \path[3d/projection of={(A) on (B)--(C)}]
   coordinate (P);
 \draw[dashed] (A) -- (P);
\end{tikzpicture}
```

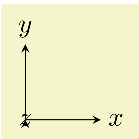
1.2 Orthonormal projections

This library can be used together with the `tikz-3dplot` package. 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 picture 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

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

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.

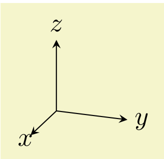
The rotation angles can be used to define the view. The conventions are chosen in such a way that they resemble those of the `tikz-3dplot` package, which gets widely used. This matrix can be written as

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

where

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

are rotations about the x, y and z axis, respectively. For $\psi = 0$, $O(\phi,0,\theta) = R^d(\theta_d,\phi_d)$ from the 'tikz-3dplot' package. Note, however, that there seems to be an inconsistency in equation (2.1) of that package.[1]




```
\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}
```

1.3 Predefined pics

`/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.

`/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.

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

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

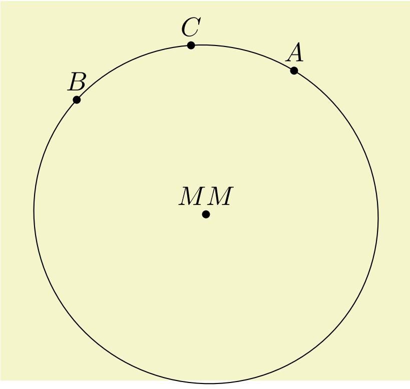
Third coordinate, like above.

/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))

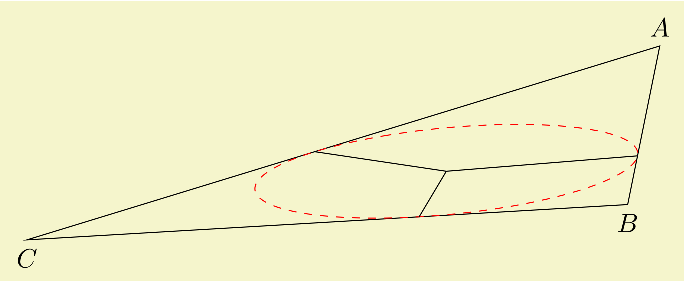
Third coordinate, like above.

/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 TikZ 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`)

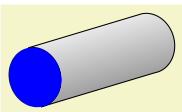
Key for heights, e.g. of cylinders.

/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

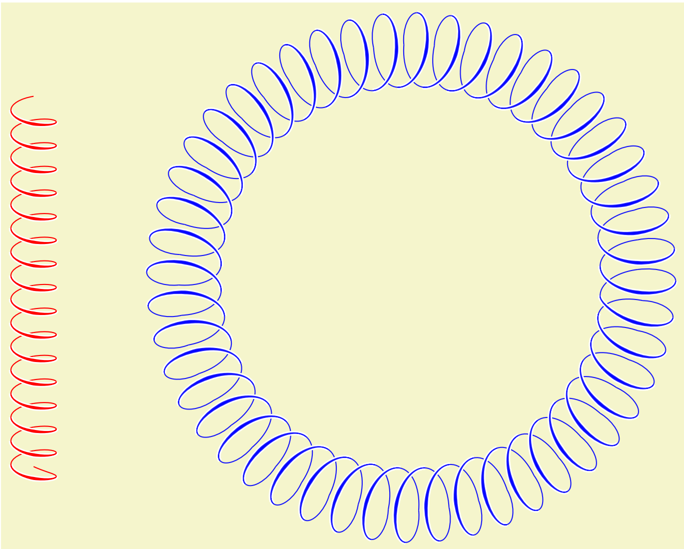
1.4 3D-like decorations

`/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}
```