3D plots: PST-3dplot v1.72 Documentation

Herbert Voß*

February 7, 2006

Abstract

The well known `pstricks` package offers excellent macros to insert more or less complex graphics into a document. `pstricks` itself is the base for several other additional packages, which are mostly named `pst-xxxx`, like `pst-3dplot`.

There exist several packages for plotting three dimensional graphical objects. `pst-3dplot` is similiar to the `pst-plot` package for two dimensional objects and mathematical functions.

This version uses the extended keyval package `xkeyval`, so be sure that you have installed this package together with the spcecial one `pst-xkey` for PSTricks. The `xkeyval` package is available at CTAN:/macros/latex/contrib/xkeyval/. It is also important that after `pst-3dplot` no package is loaded, which uses the old keyval interface.

Contents

*voss@perce.de

1 The Parallel projection

Figure 1 shows a point $P(x, y, z)$ in a three dimensional coordinate system (x, y, z) with a transformation into $P^*(x^*, y^*)$, the Point in the two dimensional system (x_E, y_E).

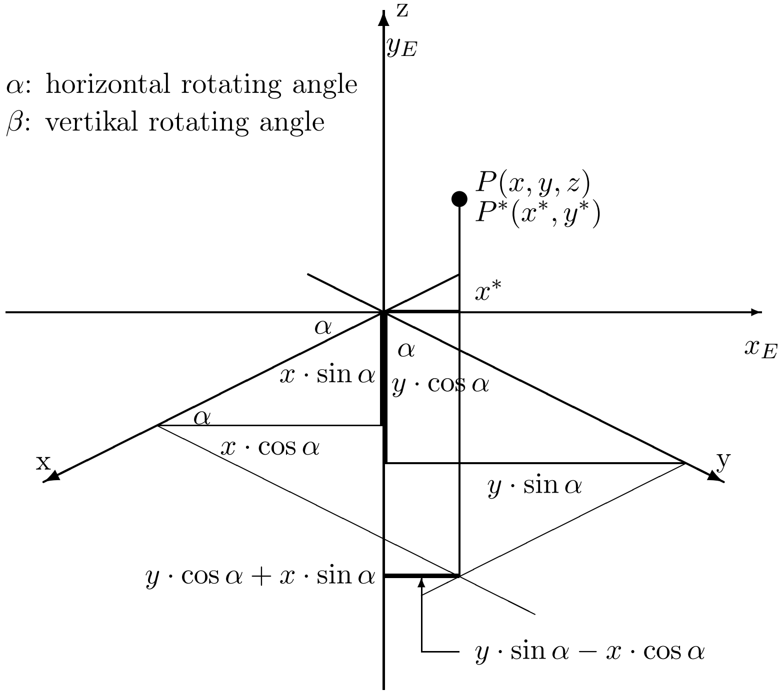


Figure 1: Lengths in a three dimensional System

The angle α is the horizontal rotation with positive values for anti clockwise rotations of the 3D coordinates. The angle β is the vertical rotation (orthogonal to the paper plane). In figure 2 we have $\alpha = \beta = 0$. The y-axis comes perpendicular out of the paper plane. Figure 3 shows the same for another angle with a view from the side, where the x-axis shows into the paper plane and the angle β is greater than 0 degrees.

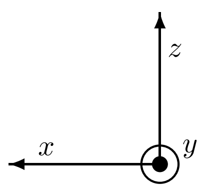


Figure 2: Coordinate System for $\alpha = \beta = 0$ (y-axis comes out of the paper plane)

The two dimensional x coordinate x^* is the difference of the two horizontal lengths $y \cdot \sin\alpha$ und $x \cdot \cos\alpha$ (figure 1):

$$x^* = -x \cdot \cos\alpha + y \cdot \sin\alpha \quad (1)$$

The z-coordinate is unimportant, because the rotation comes out of the paper plane, so we have only a different y^* value for the two dimensional coordinate but no other x^* value. The β angle is well seen in figure 3 which derives from figure 2, if the coordinate system is rotated by 90° horizontally to the left and vertically by β also to the left.

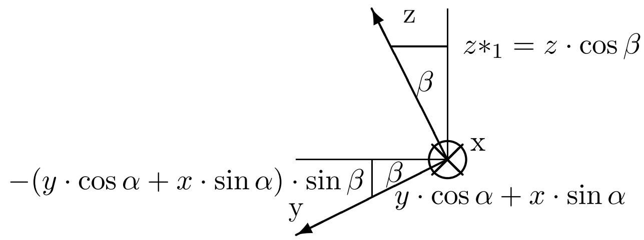


Figure 3: Coordinate System for $\alpha = 0$ and $\beta > 0$ (x-axis goes into the paper plane)

The value of the perpendicular projected z coordinate is $z^* = z{\cdot}cos\beta$. With figure 3 we see, that the point $P(x, y, z)$ runs on an elliptical curve when β is constant and α changes continues. The vertical alteration of P id the difefrence of the two "perpendicular" lines $y \cdot \cos\alpha$ and $x \cdot \sin\alpha$. These lines are rotated by the angle β, so we have them to multiply with $\sin\beta$ to get the vertical part. We get the following transformation equations:

$$\begin{aligned} x_E &= -x\cos\alpha + y\sin\alpha \\ y_E &= -(x\sin\alpha + y\cos\alpha)\cdot\sin\beta + z\cos\beta \end{aligned} \tag{2}$$

or written in matrix form:

$$\begin{pmatrix} x_E \\ y_E \end{pmatrix} = \begin{pmatrix} -\cos\alpha & \sin\alpha & 0 \\ -\sin\alpha\sin\beta & -\cos\alpha\sin\beta & \cos\beta \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{3}$$

All following figures show a grid, which has only the sense to make things clearer.

2 Options

All options which are set with `psset` are global and all which are passed with the optional argument of a macro are local for this macro. This is an important fact for setting the angles `Alpha` and `Beta`. Mostly all macro need these values, this is the reason why they should be set with `psset` and not part of an optional argument.

3 Coordinates

`pst-3dplot` accepts cartesian or spherical coordinates. In both cases there must be three parameters: `(x,y,z)` or alternatively (r,ϕ,θ), where r is the radius, *phi* the longitude angle and θ the lattitude angle. For the spherical coordinates set the option `SphericalCoor=true`. Spherical coordinates are possible for all macros where three dimensional coordinates are expected, except for the plotting functions (math functions and data records). Maybe that this is also interesting for someone, then let me know.

4 Coordinate axes

The syntax for drawing the coordinate axes is

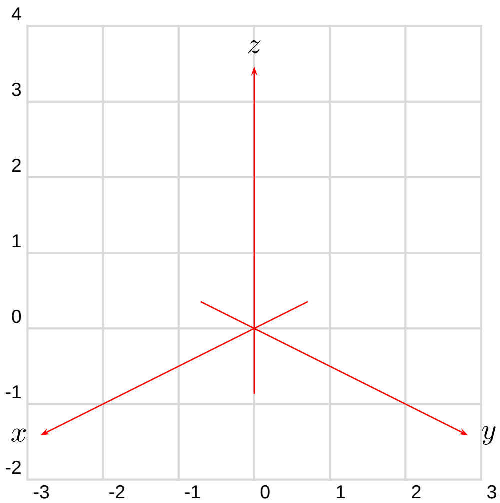
```
\pstThreeDCoor[<options>]
```

The only special option is `drawing=true|false`, which enables the drawing of the coordinate axes. The default is true. In nearly all cases the `\pstThreeDCoor` macro must be part of any drawing to initialize the 3d-system. If `drawing` is set to `false`, then all ticklines options are also disabled.

Without any options we get the default view with the in table 1 listed options with the predefined values.

Table 1: All new parameters for pst-plot

Name	Type	Default
Alpha	<angle>	45
Beta	<angle>	30
xMin	<value>	-1
xMax	<value>	4
yMin	<value>	-1
yMax	<value>	4
zMin	<value>	-1
zMax	<value>	4
nameX	<string>	\$x\$
spotX	<angle>	180
nameY	<string>	\$y\$
spotY	<angle>	0
nameZ	<string>	\$z\$
spotZ	<angle>	90
IIIDticks	false|true	false
Dx	<value>	1
Dy	<value>	1
Dz	<value>	1
IIIDxTicksPlane	xy|xz|yz	xy
IIIDyTicksPlane	xy|xz|yz	yz
IIIDzTicksPlane	xy|xz|yz	yz
IIIDticksize	<value>	0.1
IIIDxticksep	<value>	-0.4
IIIDyticksep	<value>	-0.2
IIIDzticksep	<value>	0.2
RotX	<angle>	0
RotY	<angle>	0
RotZ	<angle>	0
RotSequence	xyz|xzy|yxz|yzx|zxy|zyx	xyz

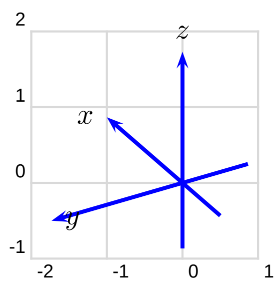



```
\begin{pspicture}(-3,-2.5)(3,4.25)\psgrid
  \pstThreeDCoor
\end{pspicture}
```

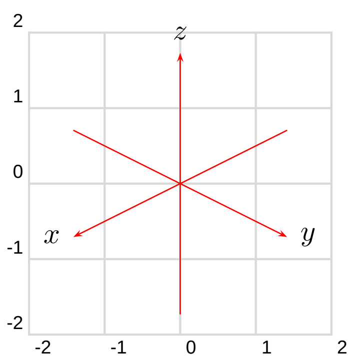
There are no restrictions for the angles and the max and min values for the axes; all `pstricks` options are possible as well. The following example changes the color and the width of the axes.

> The angles `Alpha` and `Beta` are important to all macros and should always be set with `psset` to make them global to all other macros. Otherwise they are only local inside the macro to which they are passed.

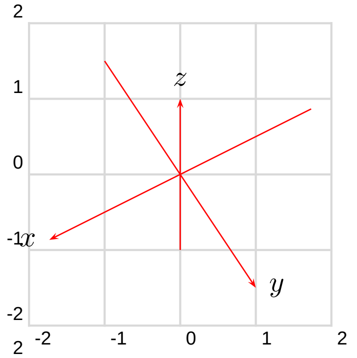
`Alpha` ist the horizontal and `Beta` the vertical rotation angle of the Cartesian coordinate system.



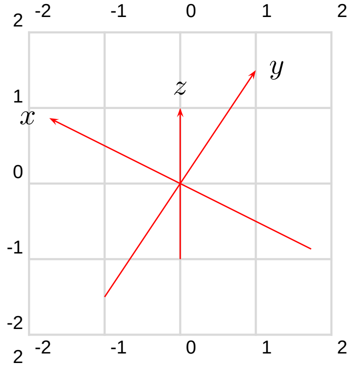

```
\begin{pspicture}(-2,-1.25)(1,2.25)\psgrid
  \pstThreeDCoor[%
    linewidth=1.5pt,linecolor=blue,%
    xMin=-1,xMax=2,
    yMin=-1,yMax=2,%
    zMin=-1,zMax=2,%
    Alpha=-60,Beta=30]
\end{pspicture}
```



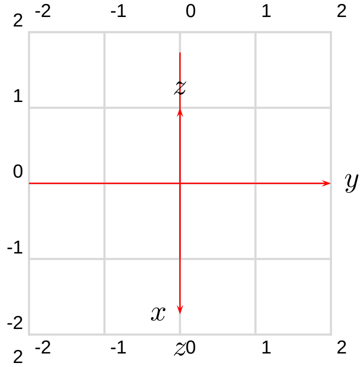

```
\begin{pspicture}(-2,-2)(2,2)\psgrid
  \pstThreeDCoor[xMin=-2,xMax=2,yMin=-2,yMax=2,%
    zMin=-2,zMax=2]
\end{pspicture}
```



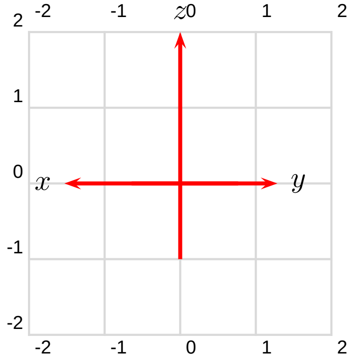

```
\begin{pspicture}(-2,-2)(2,2)\psgrid
 \pstThreeDCoor[xMin=-2,xMax=2,yMin=-2,yMax=2,zMin=-2,zMax=2,%
   Alpha=30,Beta=60]
\end{pspicture}
```




```
\begin{pspicture}(-2,-2)(2,2)\psgrid
 \pstThreeDCoor[xMin=-2,xMax=2,yMin=-2,yMax=2,zMin=-2,zMax=2,%
  Alpha=30,Beta=-60]
\end{pspicture}
```




```
\begin{pspicture}(-2,-2)(2,2)\psgrid
 \pstThreeDCoor[
  xMin=-2,xMax=2,yMin=-2,yMax=2,%
  zMin=-2,zMax=2,Alpha=90,Beta=60]
\end{pspicture}
```

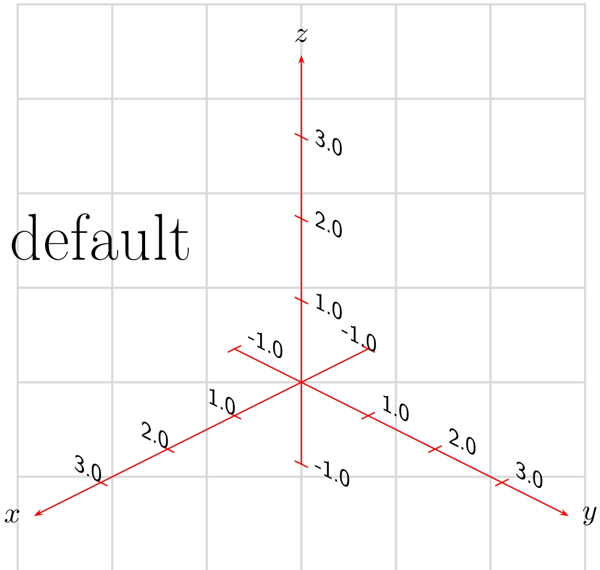



```
\begin{pspicture}(-2,-2)(2,2)\psgrid
 \pstThreeDCoor[linewidth=1.5pt,%
   xMin=-1,xMax=2,yMin=-1,yMax=2,%
   zMin=-1,zMax=2,Alpha=40,Beta=0]
\end{pspicture}
```

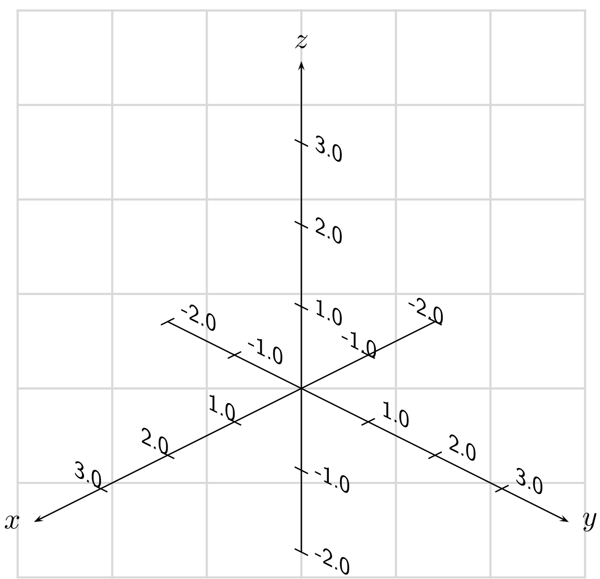
4.1 Ticks

With the option `IIIDticks` the axes get ticks and labels. There are several options to place the labels in right plane to get an optimal view. The view of the ticklabels can be changed by redefining the macro

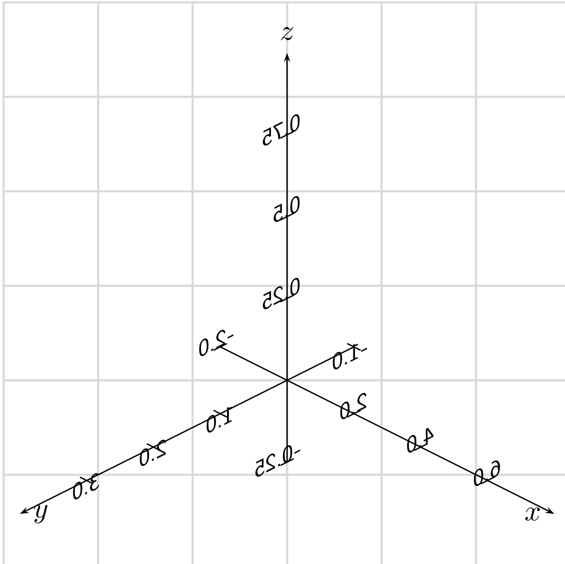
```
\def\psxyzlabel#1{\bgroup\footnotesize\textsf{#1}\egroup}
```




```
\begin{pspicture}(-3,-2.5)(3,4)
  \psgrid
  \pstThreeDCoor[IIIDticks]%
  \pstThreeDPut(3,0,3){\Huge default}
\end{pspicture}
```

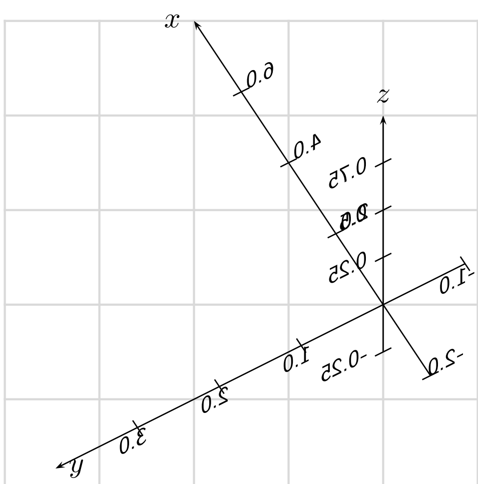



```
\begin{pspicture}(-3,-2.5)(3,4)\psgrid
  \pstThreeDCoor[linecolor=black,%
    IIIDticks,xMin=-2,yMin=-2,zMin=-2]%
\end{pspicture}
```

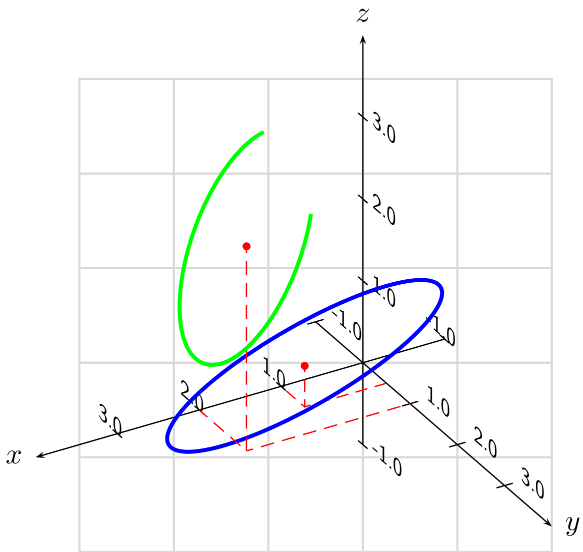



```
\begin{pspicture}(-3,-2.5)(3,4)\psgrid
 \pstThreeDCoor[linecolor=black,%
   IIIDticks,IIIDzTicksPlane=xz,IIIDzticksep=-0.2,%
   IIIDxTicksPlane=xz,,IIIDxticksep=-0.2,%
   IIIDyTicksPlane=xy,,IIIDyticksep=0.2,%
   Dx=2,Dy=1,Dz=0.25,Alpha=-135,Beta=-30]%
\end{pspicture}
```

The following example shows a wrong placing of the labels, the planes should be changed.



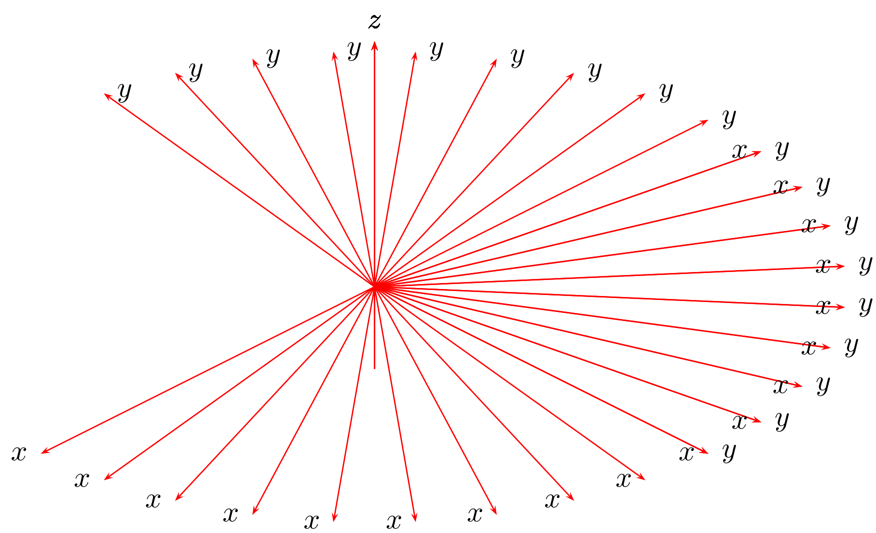

```
\psset{Alpha=-60,Beta=60}
\begin{pspicture}(-4,-2.25)(1,3)
 \psgrid
 \pstThreeDCoor[linecolor=black,%
    IIIDticks,Dx=2,Dy=1,Dz=0.25]%
\end{pspicture}
```



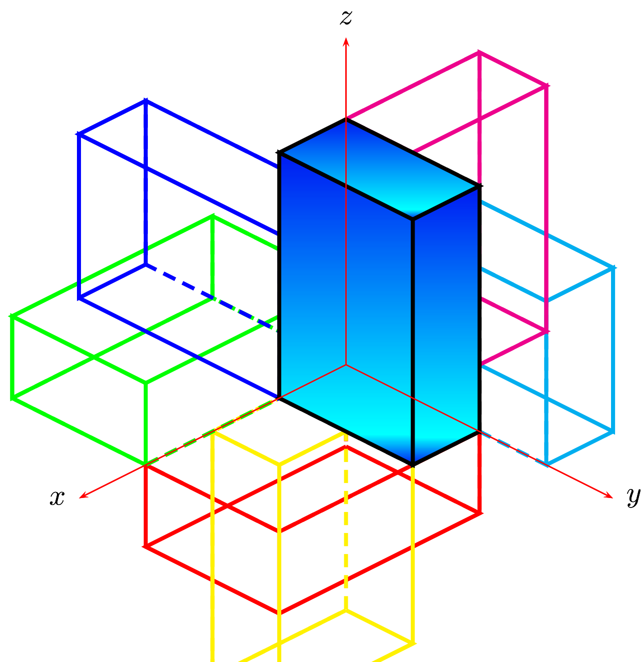

```
\begin{pspicture}(-3,-2.25)(2,3)
 \psgrid
 \psset{Alpha=30,Beta=30}
 \pstThreeDCoor[linecolor=black,IIIDticks]
 \pstThreeDDot[linecolor=red,drawCoor=true](1,0.5,0.5)%
    the center
 \psset{linecolor=blue, linewidth=1.5pt}
 \pstThreeDEllipse(1,0.5,0.5)(-0.5,1,0.5)(1,-0.5,-1)
 \psset{beginAngle=0,endAngle=270,linecolor=green}
 \pstThreeDDot[linecolor=red,drawCoor=true](2,1,2.5)% the
    center
 \pstThreeDEllipse(2,1,2.5)(-0.5,0.5,0.5)(0.5,0.5,-1)
\end{pspicture}
```

5 Rotation

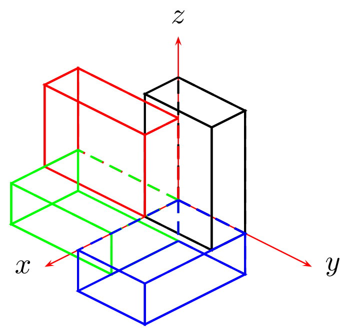
The coordinate system can be rotated independent from the given Alpha and Beta values. This makes it possible to place the axes in any direction and any order. There are the three options `RotX`, `RotY`, `RotZ` and an additional one for the rotating sequence, which can be any combination of the three letters `xyz`.



```
\begin{pspicture}(-6,-3)(6,3)
 \multido{\iA=0+10}{18}{%
   \pstThreeDCoor[RotZ=\iA,xMin=0,xMax=5,yMin=0,yMax=5,zMin=-1,zMax=3]%
 }
\end{pspicture}
```




```
\psset{unit=2,linewidth=1.5pt}
\begin{pspicture}(-2,-1.5)(2,2.5)%
  \pstThreeDCoor[xMin=0,xMax=2,yMin=0,yMax=2,zMin=0,zMax=2]%
  \pstThreeDBox[RotX=90,RotY=90,RotZ=90,%
    linecolor=red](0,0,0)(.5,0,0)(0,1,0)(0,0,1.5)
  \pstThreeDBox[RotSequence=xzy,RotX=90,RotY=90,RotZ=90,%
    linecolor=yellow](0,0,0)(.5,0,0)(0,1,0)(0,0,1.5)
  \pstThreeDBox[RotSequence=zyx,RotX=90,RotY=90,RotZ=90,%
    linecolor=green](0,0,0)(.5,0,0)(0,1,0)(0,0,1.5)
  \pstThreeDBox[RotSequence=zxy,RotX=90,RotY=90,RotZ=90,%
    linecolor=blue](0,0,0)(.5,0,0)(0,1,0)(0,0,1.5)
  \pstThreeDBox[RotSequence=yxz,RotX=90,RotY=90,RotZ=90,%
    linecolor=cyan](0,0,0)(.5,0,0)(0,1,0)(0,0,1.5)
  \pstThreeDBox[RotSequence=yzx,RotX=90,RotY=90,RotZ=90,%
    linecolor=magenta](0,0,0)(.5,0,0)(0,1,0)(0,0,1.5)
  \pstThreeDBox[fillstyle=gradient,RotX=0](0,0,0)(.5,0,0)(0,1,0)(0,0,1.5)
  \pstThreeDCoor[xMin=0,xMax=2,yMin=0,yMax=2,zMin=0,zMax=2]%
\end{pspicture}%
```




```
\begin{pspicture}(-2,-1.5)(2,2.5)%
  \pstThreeDCoor[xMin=0,xMax=2,yMin=0,yMax=2,zMin=0,zMax=2]%
  \pstThreeDBox(0,0,0)(.5,0,0)(0,1,0)(0,0,1.5)
  \pstThreeDBox[RotX=90,linecolor=red](0,0,0)(.5,0,0)(0,1,0)(0,0,1.5)
  \pstThreeDBox[RotX=90,RotY=90,linecolor=green](0,0,0)(.5,0,0)(0,1,0)(0,0,1.5)
  \pstThreeDBox[RotX=90,RotY=90,RotZ=90,linecolor=blue](0,0,0)(.5,0,0)(0,1,0)(0,0,1.5)
\end{pspicture}%
```

6 Plane Grids

`\pstThreeDPlaneGrid[<options>](xMin,yMin)(xMax,yMax)`

There are three additional options

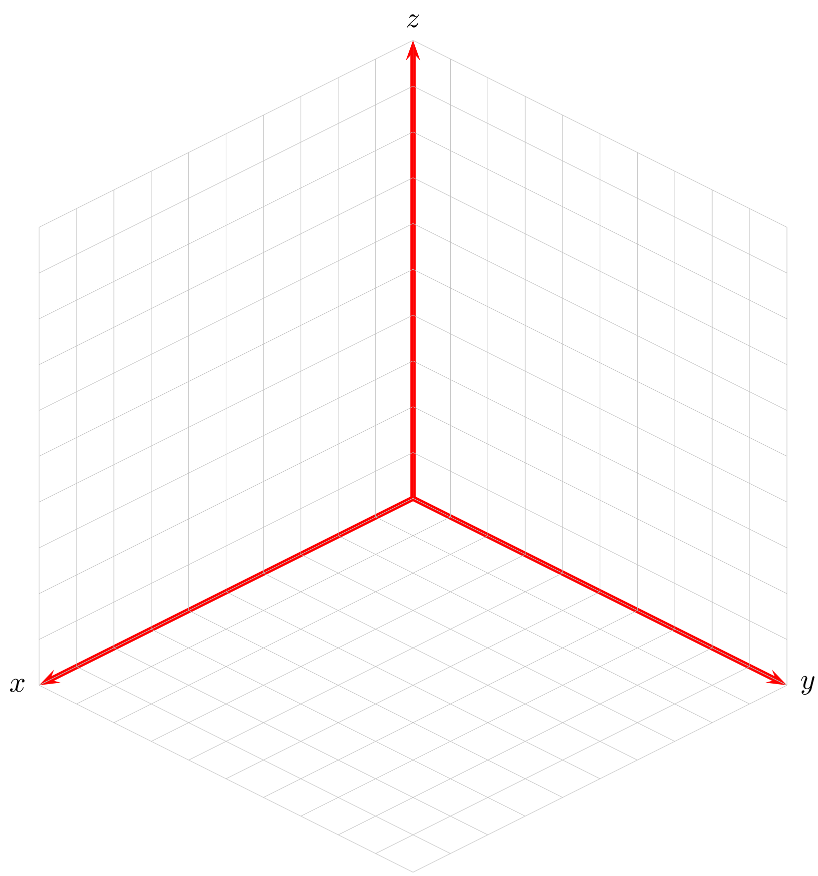
planeGrid can be one of the following values: `xy`, `xz`, `yz`. Default is `xy`.

subticks Number of ticks. Default is `10`.[1]

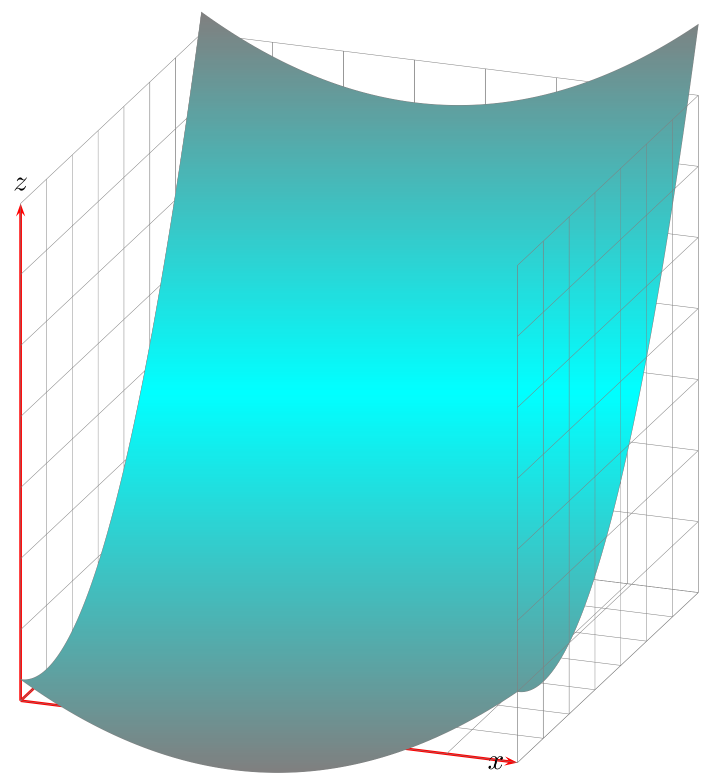
planeGridOffset a length for the shift of the grid. Default is `0`.

This macro is a special one for the coordinate system to show the units, but can be used in any way. `subticks` defines the number of ticklines for both axes and `xsubticks` and `ysubticks` for each one.

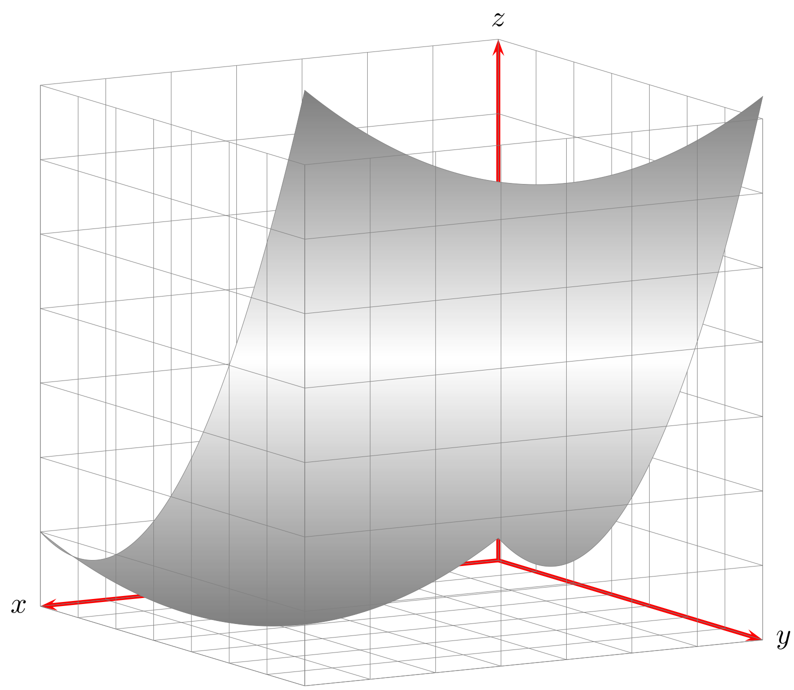
[1] This options is also defined in the package `pstricks-add`, so it is nessecary to to set this option locally or with the family option of `pst-xkey`




```
\begin{pspicture}(-5,-5)(5,6.5)
  \pstThreeDCoor[xMin=0,yMin=0,zMin=0,xMax=7,yMax=7,zMax=7,linewidth=2pt]
  \psset{linewidth=0.1pt,linecolor=lightgray}
  \pstThreeDPlaneGrid(0,0)(7,7)
  \pstThreeDPlaneGrid[planeGrid=xz](0,0)(7,7)
  \pstThreeDPlaneGrid[planeGrid=yz](0,0)(7,7)
\end{pspicture}
```




```
\begin{pspicture}(-1,-2)(10,10)
  \psset{Beta=20,Alpha=160,subticks=7}
  \pstThreeDCoor[xMin=0,yMin=0,zMin=0,xMax=7,yMax=7,zMax=7,linewidth=1pt]
  \psset{linewidth=0.1pt,linecolor=gray}
  \pstThreeDPlaneGrid(0,0)(7,7)
  \pstThreeDPlaneGrid[planeGrid=xz,planeGridOffset=7](0,0)(7,7)
  \pstThreeDPlaneGrid[planeGrid=yz](0,0)(7,7)
  \pscustom[linewidth=0.1pt,fillstyle=gradient,gradbegin=gray,gradmidpoint=0.5,plotstyle=curve]{
    \psset{xPlotpoints=200,yPlotpoints=1}
    \psplotThreeD(0,7)(0,0){%
       x dup mul y dup mul 2 mul add x 6 mul sub y 4 mul sub 3 add 10 div }
    \psset{xPlotpoints=1,yPlotpoints=200,drawStyle=yLines}
    \psplotThreeD(7,7)(0,7){%
       x dup mul y dup mul 2 mul add x 6 mul sub y 4 mul sub 3 add 10 div }
    \psset{xPlotpoints=200,yPlotpoints=1,drawStyle=xLines}
    \psplotThreeD(7,0)(7,7){%
       x dup mul y dup mul 2 mul add x 6 mul sub y 4 mul sub 3 add 10 div }
    \psset{xPlotpoints=1,yPlotpoints=200,drawStyle=yLines}
    \psplotThreeD(0,0)(7,0){%
       x dup mul y dup mul 2 mul add x 6 mul sub y 4 mul sub 3 add 10 div }
  }
  \pstThreeDPlaneGrid[planeGrid=yz,planeGridOffset=7](0,0)(7,7)
\end{pspicture}
```




```
\begin{pspicture}(-6,-2)(4,7)
 \psset{Beta=10,Alpha=30,subticks=7}
 \pstThreeDCoor[xMin=0,yMin=0,zMin=0,xMax=7,yMax=7,zMax=7,linewidth=1.5pt]
 \psset{linewidth=0.1pt,linecolor=gray}
 \pstThreeDPlaneGrid(0,0)(7,7)
 \pstThreeDPlaneGrid[planeGrid=xz](0,0)(7,7)
 \pstThreeDPlaneGrid[planeGrid=yz](0,0)(7,7)
 \pscustom[linewidth=0.1pt,fillstyle=gradient,gradbegin=gray,gradend=white,gradmidpoint=0.5,plotstyle=curve
   ]{
   \psset{xPlotpoints=200,yPlotpoints=1}
   \psplotThreeD(0,7)(0,0){%
      x dup mul y dup mul 2 mul add x 6 mul sub y 4 mul sub 3 add 10 div }
   \psset{xPlotpoints=1,yPlotpoints=200,drawStyle=yLines}
   \psplotThreeD(7,7)(0,7){%
      x dup mul y dup mul 2 mul add x 6 mul sub y 4 mul sub 3 add 10 div }
   \psset{xPlotpoints=200,yPlotpoints=1,drawStyle=xLines}
   \psplotThreeD(7,0)(7,7){%
      x dup mul y dup mul 2 mul add x 6 mul sub y 4 mul sub 3 add 10 div }
   \psset{xPlotpoints=1,yPlotpoints=200,drawStyle=yLines}
   \psplotThreeD(0,0)(7,0){%
      x dup mul y dup mul 2 mul add x 6 mul sub y 4 mul sub 3 add 10 div }
 }
 \pstThreeDPlaneGrid[planeGrid=xz,planeGridOffset=7](0,0)(7,7)
 \pstThreeDPlaneGrid[planeGrid=yz,planeGridOffset=7](0,0)(7,7)
\end{pspicture}
```

The equation for the examples is

$$f(x,y) = \frac{x^2 + 2y^2 - 6x - 4y + 3}{10}$$

7 Put

There exists a special option for the put macros:

`origin=lt|lB|lb|t|c|B|b|rt|rB|rb`

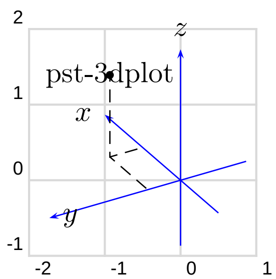
for the placing of the text or other objects.

This works only well for the `\pstThreeDPut` macro. The default is `c` and for the `pstPlanePut` the left baseline `lB`.

7.1 pstThreeDPut

The syntax is similiar to the `\rput` macro:

`\pstThreeDPut[options](x,y,z){<any stuff>}`




```
\begin{pspicture}(-2,-1.25)(1,2.25)
  \psgrid
  \psset{Alpha=-60,Beta=30}
  \pstThreeDCoor[linecolor=blue,%
    xMin=-1,xMax=2,yMin=-1,yMax=2,zMin=-1,zMax=2]
  \pstThreeDPut(1,0.5,1.25){pst-3dplot}
  \pstThreeDDot[drawCoor=true](1,0.5,1.25)
\end{pspicture}
```

Internally the `\pstThreeDPut` macro defines the two dimensional node `temp@pstNode` and then uses the default `\rput` macro from `pstricks`. In fact of the perspective view od the coordinate system, the 3D dot must not be seen as the center of the printed stuff.

7.2 pstPlanePut[2]

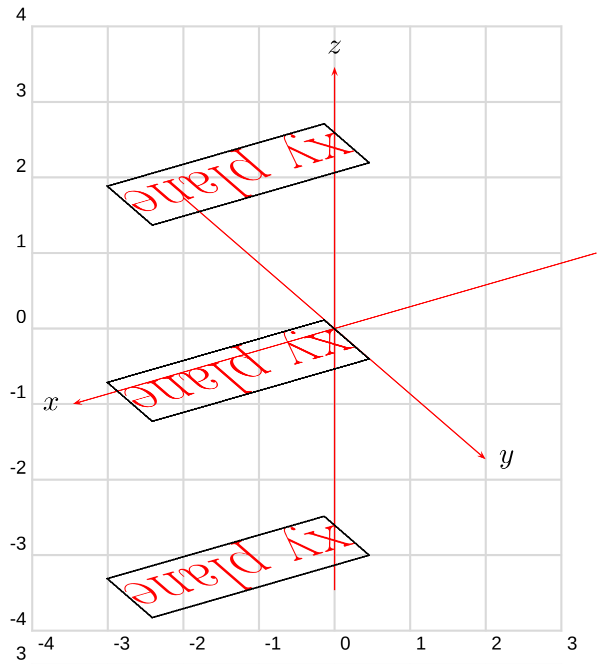
The syntax of the `pstPlanePut` is

`\pstPlanePut[plane=<2D plane>,planecorr=<Correction of plane's alignment>](x,y,z){Object}`

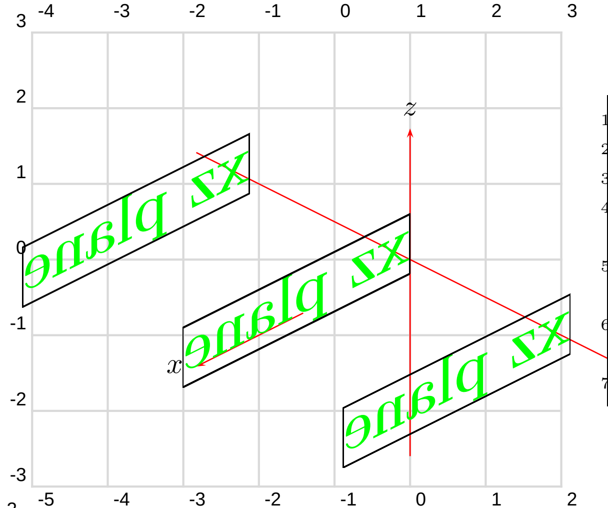
We have two parameters, `plane` and `planecorr`; both are optional. Let's start with the first parameter, `plane`. Possible values for the two dimensional plane are `xy xz yz` . If this parameter is missing then `plane=xy` is set. The first letter marks the positive direction for the width and the second for the height.

The object can be of any type, in most cases it will be some kind of text. The reference point for the object is the left side and vertically centered, often abbreviated as `lB`. The following examples show for all three planes the same textbox.

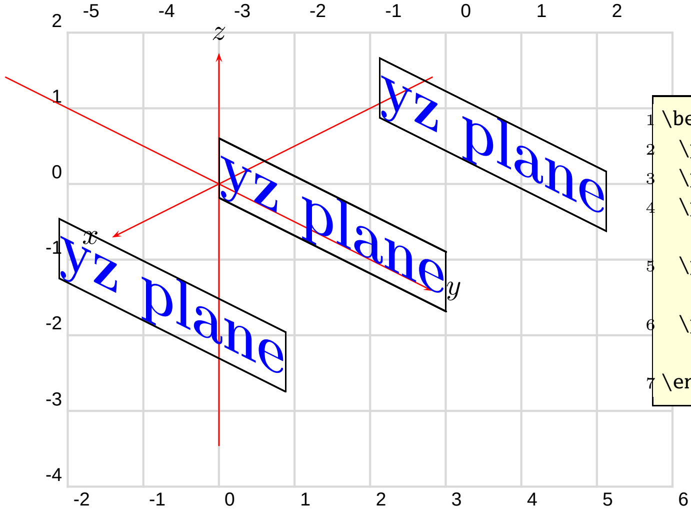
[2] Thanks to Torsten Suhling




```
\begin{pspicture}(-4,-4)(3,4)
  \psgrid
  \psset{Alpha=30}
  \pstThreeDCoor[xMin=-4,yMin=-4,zMin=-4]
  \pstPlanePut[plane=xy](0,0,-3){\fbox{\Huge\red xy plane}}
  \pstPlanePut[plane=xy](0,0,0){\fbox{\Huge\red xy plane}}
  \pstPlanePut[plane=xy](0,0,3){\fbox{\Huge\red xy plane}}
\end{pspicture}
```




```
\begin{pspicture}(-5,-3)(2,3)
  \psgrid
  \pstThreeDCoor[xMin=2,yMin=-4,zMin=-3,zMax=2]
  \pstPlanePut[plane=xz](0,-3,0){\fbox{\Huge\green\textbf{
     xz plane}}}
  \pstPlanePut[plane=xz](0,0,0){\fbox{\Huge\green\textbf{xz
      plane}}}
  \pstPlanePut[plane=xz](0,3,0){\fbox{\Huge\green\textbf{xz
      plane}}}
\end{pspicture}
```




```
\begin{pspicture}(-2,-4)(6,2)
  \psgrid
  \pstThreeDCoor[xMin=-4,yMin=-4,zMin=-4,xMax=2,zMax=2]
  \pstPlanePut[plane=yz](-3,0,0){\fbox{\Huge\blue\textbf{yz
      plane}}}
  \pstPlanePut[plane=yz](0,0,0){\fbox{\Huge\blue\textbf{yz
     plane}}}
  \pstPlanePut[plane=yz](3,0,0){\fbox{\Huge\blue\textbf{yz
     plane}}}
\end{pspicture}
```

The following examples use the `origin` option to show that there are still some problems with the xy-plane. The second parameter is `planecorr`. As first the values:

off Former and default behaviour; nothing will be changed. This value is set, when parameter is missing.

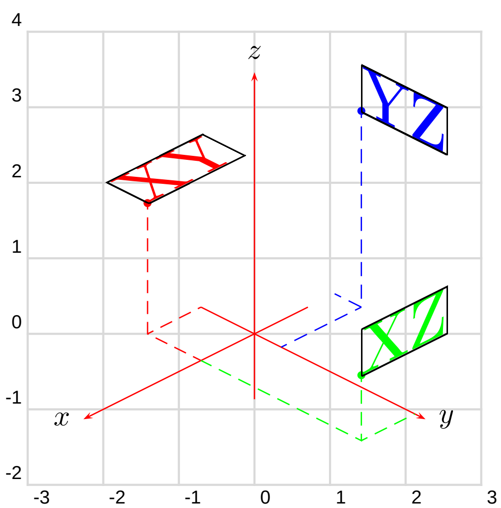
normal Default correction, planes will be rotated to be readable.

xyrot Additionaly correction for xy plane; bottom line of letters will be set parallel to the y-axis.

What kind off correction is ment? In the plots above labels for the xy plane and the xz plane are mirrored. This is not a bug, it's ... mathematics.

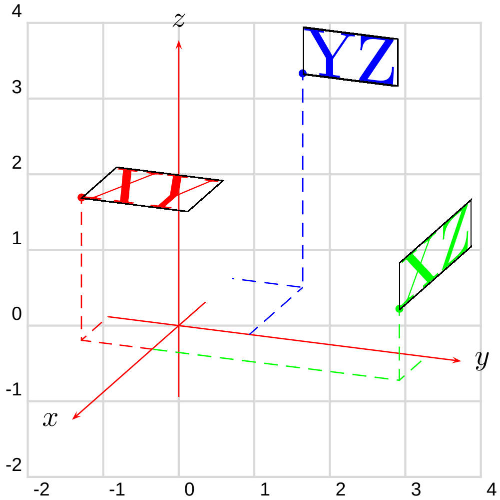
`\pstPlanePut` puts the labels on the plane of it's value. That means, `plane=xy` puts the label on the xy plane, so that the x marks the positive direction for the width, the y for the height and the label XY plane on the top side of plane. If you see the label mirrored, you just look from the bottom side of plane. ...

If you want to keep the labels readable for every view, i. e. for every value of `Alpha` and `Beta`, you should set the value of the parameter `planecorr` to `normal`; just like in next example:




```
\begin{pspicture}(-3,-2)(3,4)\psgrid
 \psset{origin=lb}
 \pstThreeDCoor[xMax=3.2,yMax=3.2,zMax=4]
 \pstThreeDDot[drawCoor=true,linecolor=red](1,-1,2)
 \pstPlanePut[plane=xy,planecorr=normal](1,-1,2)
   {\fbox{\Huge\red\textbf{XY}}}
 \pstThreeDDot[drawCoor=true,linecolor=green](1,3,1)
 \pstPlanePut[plane=xz,planecorr=normal](1,3,1)
   {\fbox{\Huge\green\textbf{XZ}}}
 \pstThreeDDot[drawCoor=true,linecolor=blue](-1.5,0.5,3)
 \pstPlanePut[plane=yz,planecorr=normal](-1.5,0.5,3)
   {\fbox{\Huge\blue\textbf{YZ}}}
\end{pspicture}
```

But, why we have a third value `xyrot` of `planecorr`? If there isn't an symmetrical view, – just like in this example – it could be usefull to rotate the label for xy-plane, so that body line of letters is parallel to the y axis. It's done by setting `planecorr=xyrot`:




```
\begin{pspicture}(-2,-2)(4,4)\psgrid
  \psset{origin=lb}
  \psset{Alpha=69.3,Beta=19.43}
  \pstThreeDCoor[xMax=4,yMax=4,zMax=4]
  \pstThreeDDot[drawCoor=true,linecolor=red](1,-1,2)
  \pstPlanePut[plane=xy,planecorr=xyrot](1,-1,2)
    {\fbox{\Huge\red\textbf{XY}}}
  \pstThreeDDot[drawCoor=true,linecolor=green](1,3.5,1)
  \pstPlanePut[plane=xz,planecorr=xyrot](1,3.5,1)
    {\fbox{\Huge\green\textbf{XZ}}}
  \pstThreeDDot[drawCoor=true,linecolor=blue](-2,1,3)
  \pstPlanePut[plane=yz,planecorr=xyrot](-2,1,3)
    {\fbox{\Huge\blue\textbf{YZ}}}
\end{pspicture}
```

8 Nodes

The syntax is

`\pstThreeDNode(x,y,z){<node name>}`

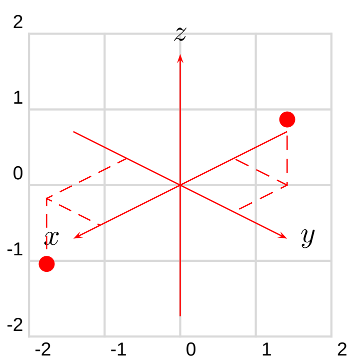
This node is internally a two dimensional node, so it cannot be used as a replacement for the parameters `(x,y,z)` of a 3D dot, which is possible with the `\psline` macro from `pst-plot`: `\psline{A}{B}`, where `A` and `B` are two nodes. It is still on the to do list, that it may also be possible with `pst-3dplot`. On the other hand it is no problem to define two 3D nodes `C` and `D` and then drawing a two dimensional line from `C` to `D`.

9 Dots

The syntax for a dot is

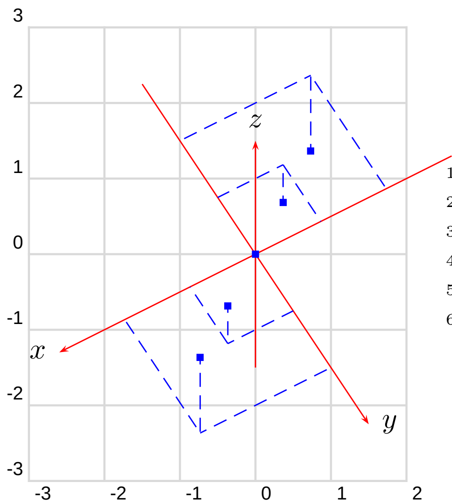
`\pstThreeDDot[<options>](x,y,z)`

Dots can be drawn with dashed lines for the three coordinates, when the option `drawCoor` is set to `true`. It is also possible to draw an unseen dot with the option `dotstyle=none`. In this case the macro draws only the coordinates when the `drawCoor` option is set to true.




```
\begin{pspicture}(-2,-2)(2,2)\psgrid
  \pstThreeDCoor[xMin=-2,xMax=2,yMin=-2,yMax=2,zMin=-2,zMax=2]
  \psset{dotstyle=*,dotscale=2,linecolor=red,drawCoor=true}
  \pstThreeDDot(-1,1,1)
  \pstThreeDDot(1.5,-1,-1)
\end{pspicture}
```

In the following figure the coordinates of the dots are (a, a, a) where a is $-2, -1, 0, 1, 2$.




```
\begin{pspicture}(-3,-3.25)(2,3.25)\psgrid
  \psset{Alpha=30,Beta=60,dotstyle=square*,dotsize=3pt,%
    linecolor=blue,drawCoor=true}
  \pstThreeDCoor[xMin=-3,xMax=3,yMin=-3,yMax=3,zMin=-3,zMax=3]
  \multido{\n=-2+1}{5}{\pstThreeDDot(\n,\n,\n)}
\end{pspicture}
```

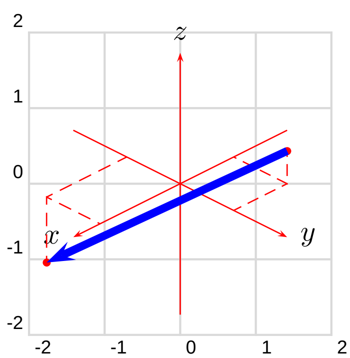
10 Lines

The syntax for a three dimensional line is just like the same from `\psline`

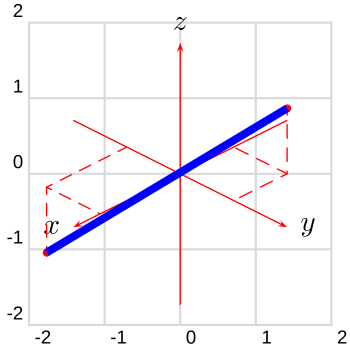
`\pstThreeDLine[<options>]{<arrow>}(x1,y1,z1)(...)(xn,yn,zn)`

The option and arrow part are both optional and the number of points is only limited to the memory. All options for lines from `pstricks` are possible, there are no special ones for a 3D line. There is no difference in drawing a line or a vector; the first one has an arrow of type "’-"‘ and the second of "’->"‘.

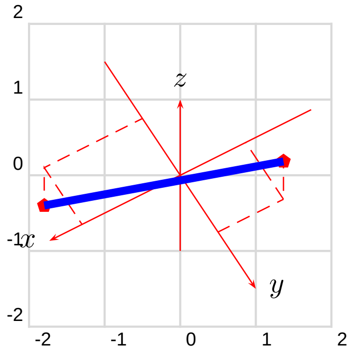
There is no special `polygon` macro, because you can get nearlx the same with `\pstThreeDLine`.



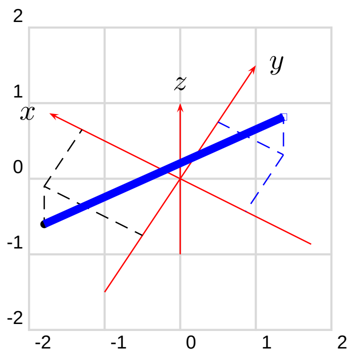

```
\begin{pspicture}(-2,-2.25)(2,2.25)\psgrid
  \pstThreeDCoor[xMin=-2,xMax=2,yMin=-2,yMax=2,zMin=-2,zMax=2]
  \psset{dotstyle=*,linecolor=red,drawCoor=true}
  \pstThreeDDot(-1,1,0.5)
  \pstThreeDDot(1.5,-1,-1)
  \pstThreeDLine[linewidth=3pt,linecolor=blue,arrows=->]%
    (-1,1,0.5)(1.5,-1,-1)
\end{pspicture}
```



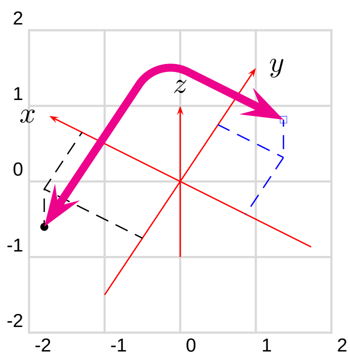

```
\begin{pspicture}(-2,-2.25)(2,2.25)\psgrid
  \pstThreeDCoor[xMin=-2,xMax=2,yMin=-2,yMax=2,zMin=-2,zMax=2]
  \psset{dotstyle=*,linecolor=red,drawCoor=true}
  \pstThreeDDot(-1,1,1)
  \pstThreeDDot(1.5,-1,-1)
  \pstThreeDLine[linewidth=3pt,linecolor=blue](-1,1,1)(1.5,-1,-1)
\end{pspicture}
```



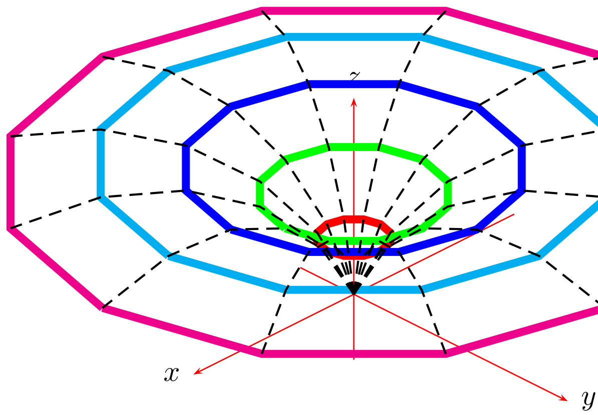

```
\begin{pspicture}(-2,-2.25)(2,2.25)\psgrid
  \psset{Alpha=30,Beta=60,dotstyle=pentagon*,dotsize=5pt,%
    linecolor=red,drawCoor=true}
  \pstThreeDCoor[xMin=-2,xMax=2,yMin=-2,yMax=2,zMin=-2,zMax=2]
  \pstThreeDDot(-1,1,1)
  \pstThreeDDot(1.5,-1,-1)
  \pstThreeDLine[linewidth=3pt,linecolor=blue](-1,1,1)(1.5,-1,-1)
\end{pspicture}
```




```
\begin{pspicture}(-2,-2.25)(2,2.25)\psgrid
  \psset{Alpha=30,Beta=-60}
  \pstThreeDCoor[xMin=-2,xMax=2,yMin=-2,yMax=2,zMin=-2,zMax=2]
  \pstThreeDDot[dotstyle=square,linecolor=blue,drawCoor=true](-1,1,1)
  \pstThreeDDot[drawCoor=true](1.5,-1,-1)
  \pstThreeDLine[linewidth=3pt,linecolor=blue](-1,1,1)(1.5,-1,-1)
\end{pspicture}
```




```
\begin{pspicture}(-2,-2.25)(2,2.25)\psgrid
  \psset{Alpha=30,Beta=-60}
  \pstThreeDCoor[xMin=-2,xMax=2,yMin=-2,yMax=2,zMin=-2,zMax=2]
  \pstThreeDDot[dotstyle=square,linecolor=blue,drawCoor=true](-1,1,1)
  \pstThreeDDot[drawCoor=true](1.5,-1,-1)
  \pstThreeDLine[linewidth=3pt,arrowscale=1.5,%
    linecolor=magenta,linearc=0.5]{<->}(-1,1,1)(1.5,2,-1)(1.5,-1,-1)
\end{pspicture}
```



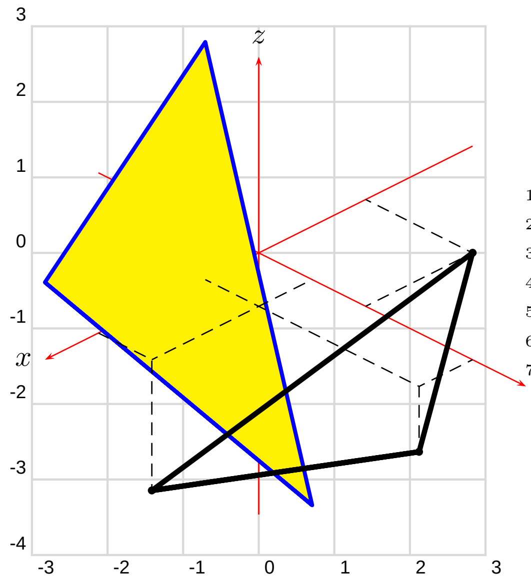

```
\begin{pspicture}(-3,-2)(4,5)\label{lines}
 \pstThreeDCoor[xMin=-3,xMax=3,yMin=-1,yMax=4,zMin=-1,zMax=3]
 \multido{\iA=1+1,\iB=60+-10}{5}{%
   \ifcase\iA\or\psset{linecolor=red}\or\psset{linecolor=green}
     \or\psset{linecolor=blue}\or\psset{linecolor=cyan}
     \or\psset{linecolor=magenta}
   \fi
   \pstThreeDLine[SphericalCoor=true,linewidth=3pt]%
     (\iA,0,\iB)(\iA,30,\iB)(\iA,60,\iB)(\iA,90,\iB)(\iA,120,\iB)(\iA,150,\iB)%
     (\iA,180,\iB)(\iA,210,\iB)(\iA,240,\iB)(\iA,270,\iB)(\iA,300,\iB)%
     (\iA,330,\iB)(\iA,360,\iB)%
 }
 \multido{\iA=0+30}{12}{%
   \pstThreeDLine[SphericalCoor=true,linestyle=dashed]%
      (0,0,0)(1,\iA,60)(2,\iA,50)(3,\iA,40)(4,\iA,30)(5,\iA,20)}
\end{pspicture}
```

11 Triangles

A triangle is given with its three points:

```
\pstThreeDTriangle[<options>](P1)(P2)(P3)
```

When the option `fillstyle` is set to another value than `none` the triangle is filled with the active color or with the one which is set with the option `fillcolor`.




```
\begin{pspicture}(-3,-4.25)(3,3.25)\psgrid
  \pstThreeDCoor[xMin=-4,xMax=4,yMin=-3,yMax=5,zMin=-4,zMax=3]
  \pstThreeDTriangle[fillcolor=yellow,fillstyle=solid,%
    linecolor=blue,linewidth=1.5pt](5,1,2)(3,4,-1)(-1,-2,2)
  \pstThreeDTriangle[drawCoor=true,linecolor=black,%
    linewidth=2pt](3,1,-2)(1,4,-1)(-2,2,0)
\end{pspicture}
```

Especially for triangles the option `linejoin` is important. The default value is 1, which gives rounded edges.

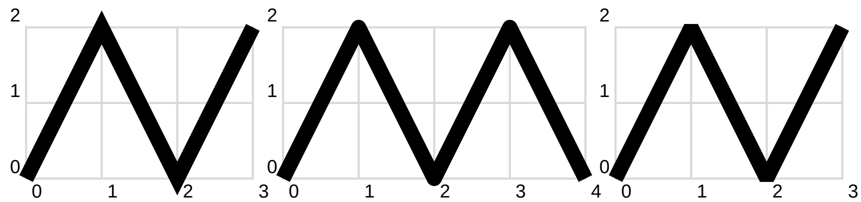
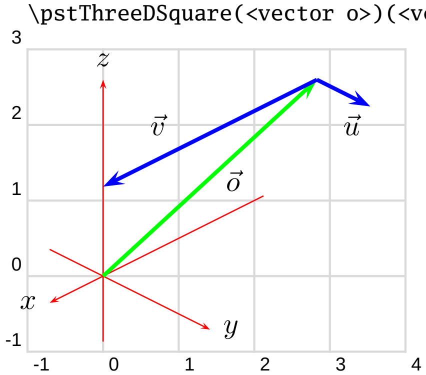


Figure 4: The meaning of the option `linejoin=0|1|2` for drawing lines

12 Squares

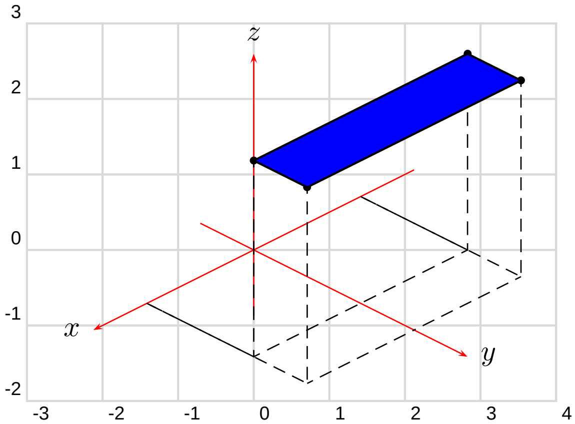
The syntax for a 3D square is:

`\pstThreeDSquare(<vector o>)(<vector u>)(<vector v>)`




```
\begin{pspicture}(-1,-1)(4,3)\psgrid
  \pstThreeDCoor[xMin=-3,xMax=1,yMin=-1,yMax=2,zMin=-1,zMax=3]
  \psset{arrows=->,arrowsize=0.2,linecolor=blue,linewidth=1.5pt}
  \pstThreeDLine[linecolor=green](0,0,0)(-2,2,3)\uput[45](1.5,1){$\vec{o}$
    }
  \pstThreeDLine(-2,2,3)(2,2,3)\uput[0](3,2){$\vec{u}$}
  \pstThreeDLine(-2,2,3)(-2,3,3)\uput[180](1,2){$\vec{v}$}
\end{pspicture}
```

Squares are nothing else than a polygon with the starting point P_o given with the origin vector $\vec{o}$ and the two direction vectors $\vec{u}$ and $\vec{v}$, which build the sides of the square.



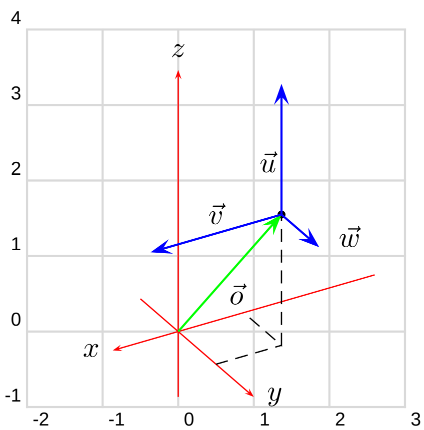

```
\begin{pspicture}(-3,-2)(4,3)\psgrid
 \pstThreeDCoor[xMin=-3,xMax=3,yMin=-1,yMax=4,zMin=-1,zMax
   =3]
 {\psset{fillcolor=blue,fillstyle=solid,drawCoor=true,
   dotstyle=*}
  \pstThreeDSquare(-2,2,3)(4,0,0)(0,1,0)}
\end{pspicture}
```

13 Boxes

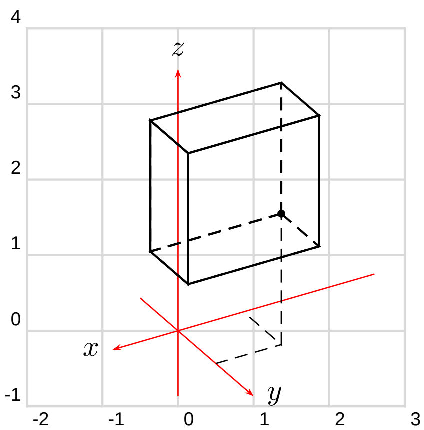
A box is a special case of a square and has the syntax

```
\pstThreeDBox[<options>](<vector o>(<vector u>)(<vector v>)(<vector w>)
```

These are the origin vector $\vec{o}$ and three direction vectors $\vec{u}$, $\vec{v}$ and $\vec{w}$, which are for example shown in the following figure.




```
\begin{pspicture}(-2,-1.25)(3,4.25)\psgrid
  \psset{Alpha=30,Beta=30}
  \pstThreeDCoor[xMin=-3,xMax=1,yMin=-1,yMax=2,zMin=-1,zMax=4]
  \pstThreeDDot[drawCoor=true](-1,1,2)
  \psset{arrows=->,arrowsize=0.2}
  \pstThreeDLine[linecolor=green](0,0,0)(-1,1,2)
  \uput[0](0.5,0.5){$\vec{o}$}
  \uput[0](0.9,2.25){$\vec{u}$}
  \uput[90](0.5,1.25){$\vec{v}$}
  \uput[45](2,1.){$\vec{w}$}
  \pstThreeDLine[linecolor=blue](-1,1,2)(-1,1,4)
  \pstThreeDLine[linecolor=blue](-1,1,2)(1,1,2)
  \pstThreeDLine[linecolor=blue](-1,1,2)(-1,2,2)
\end{pspicture}
```




```
\begin{pspicture}(-2,-1.25)(3,4.25)\psgrid
  \psset{Alpha=30,Beta=30}
  \pstThreeDCoor[xMin=-3,xMax=1,yMin=-1,yMax=2,zMin=-1,zMax=4]
  \pstThreeDBox(-1,1,2)(0,0,2)(2,0,0)(0,1,0)
  \pstThreeDDot[drawCoor=true](-1,1,2)
\end{pspicture}
```

14 Ellipses and circles

The equation for a two dimensional ellipse (figure 5)is:

$$e : \frac{(x - x_M)^2}{a^2} + \frac{(y - y_M)^2}{b^2} = 1 \tag{4}$$

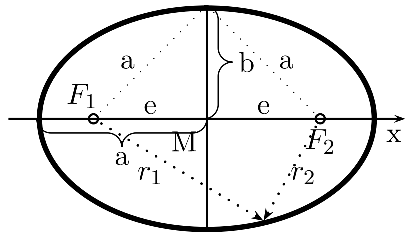


Figure 5: Definition of an Ellipse

$(x_m; y_m)$ is the center, a and b the semi major and semi minor axes respectively and e the excentricity. For $a = b = 1$ in equation 4 we get the one for the circle, which is nothing else than a special ellipse. The equation written in the parameterform is

$$\begin{aligned} x &= a \cdot \cos\alpha \\ y &= b \cdot \sin\alpha \end{aligned} \tag{5}$$

or the same with vectors to get an ellipse in a 3D system:

$$e : \vec{x} = \vec{m} + \cos\alpha \cdot \vec{u} + \sin\alpha \cdot \vec{v} \qquad 0 \le \alpha \le 360 \tag{6}$$

where $\vec{m}$ is the center, $\vec{u}$ and $\vec{v}$ the directions vectors which are perpendicular to each other.

14.1 Options

In addition to all possible options from `pst-plot` there are two special options to allow drawing of an arc (with predefined values for a full ellipse/circle):

```
beginAngle=0
endAngle=360
```

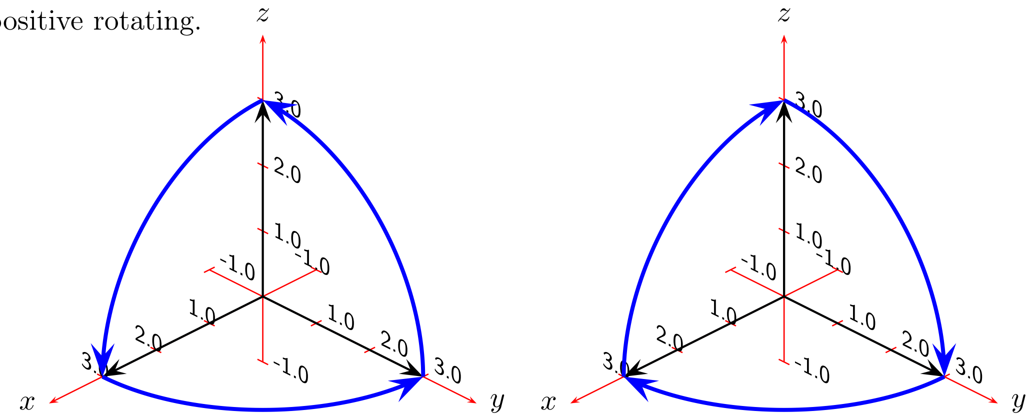
Ellipses and circles are drawn with the in section 16.2 described `parametricplotThreeD` macro with a default setting of 50 points for a full ellipse/circle.

14.2 Ellipse

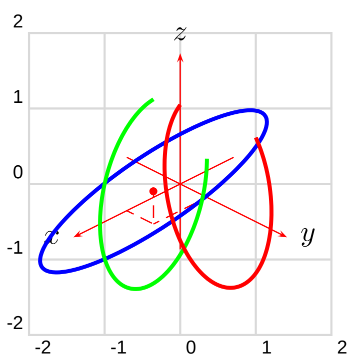
It is very difficult to see in a 3D coordinate system the difference of an ellipse and a circle. Depending to the view point an ellipse maybe seen as a circle and vice versa. The syntax of the ellipse macro is:

```
\pstThreeDEllipse[<option>](cx,cy,cz)(ux,uy,uz)(vx,vy,vz)
```

where `c` is for center and `u` and `v` for the two direction vectors. The order of these two vectors is important for the drawing if it is a left or right turn. It follows the right hand rule: flap the first vector $\vec{u}$ on the shortest way into the second one $\vec{u}$, then you'll get the positive rotating.




```
\begin{pspicture}(-3,-2)(3,3)
  \pstThreeDCoor[IIIDticks]
  \psset{arrowscale=2,arrows=->}
  \pstThreeDLine(0,0,0)(3,0,0)\pstThreeDLine(0,0,0)(0,3,0)\pstThreeDLine(0,0,0)(0,0,3)
  \psset{linecolor=blue,linewidth=1.5pt,beginAngle=0,endAngle=90}
  \pstThreeDEllipse(0,0,0)(3,0,0)(0,3,0) \pstThreeDEllipse(0,0,0)(0,0,3)(3,0,0)
  \pstThreeDEllipse(0,0,0)(0,3,0)(0,0,3)
\end{pspicture}\hspace{2em}
\begin{pspicture}(-3,-2)(3,3)
  \pstThreeDCoor[IIIDticks]
  \psset{arrowscale=2,arrows=->}
  \pstThreeDLine(0,0,0)(3,0,0)\pstThreeDLine(0,0,0)(0,3,0)\pstThreeDLine(0,0,0)(0,0,3)
  \psset{linecolor=blue,linewidth=1.5pt,beginAngle=0,endAngle=90}
  \pstThreeDEllipse(0,0,0)(0,3,0)(3,0,0) \pstThreeDEllipse(0,0,0)(3,0,0)(0,0,3)
  \pstThreeDEllipse(0,0,0)(0,0,3)(0,3,0)
\end{pspicture}
```

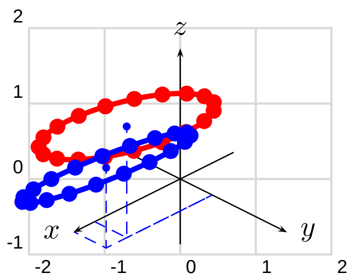



```
\begin{pspicture}(-2,-2.25)(2,2.25)\psgrid
  \pstThreeDCoor[xMax=2,yMax=2,zMax=2]
  \pstThreeDDot[linecolor=red,drawCoor=true](1,0.5,0.5)
  \psset{linecolor=blue, linewidth=1.5pt}
  \pstThreeDEllipse(1,0.5,0.5)(-0.5,1,0.5)(1,-0.5,-1)
  \psset{beginAngle=0,endAngle=270,linecolor=green}
  \pstThreeDEllipse(1,0.5,0.5)(-0.5,0.5,0.5)(0.5,0.5,-1)
  \pstThreeDEllipse[RotZ=45,linecolor=red](1,0.5,0.5)(-0.5,0.5,0.5)(0.5,0.5,-1)
\end{pspicture}
```

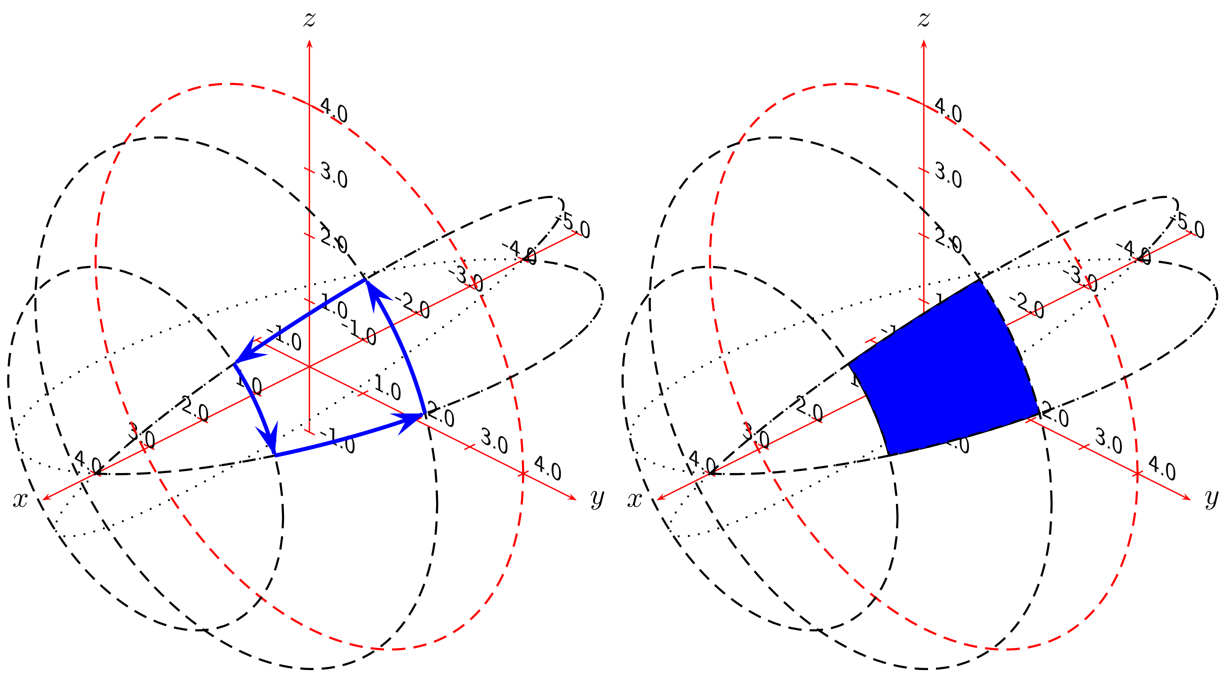
14.3 Circle

The circle is a special case of an ellipse (equ. 6) with the vectors $\vec{u}$ and $\vec{v}$ which are perpendicular to each other: $|\vec{u}| = |\vec{v}| = r$. with $\vec{u} \cdot \vec{v} = \vec{0}$

The macro `\pstThreeDCircle` is nothing else than a synonym for `\pstThreeDEllipse`. In the following example the circle is drawn with only 20 plotpoints and the option `showpoints=true`.




```
\begin{pspicture}(-2,-1.25)(2,2.25)\psgrid
  \pstThreeDCoor[xMax=2,yMax=2,zMax=2,linecolor=black]
  \psset{linecolor=red,linewidth=2pt,plotpoints=20,showpoints=true}
  \pstThreeDCircle(1.6,+0.6,1.7)(0.8,0.4,0.8)(0.8,-0.8,-0.4)
  \pstThreeDDot[drawCoor=true,linecolor=blue](1.6,+0.6,1.7)
  \pstThreeDCircle[RotY=15,linecolor=blue](1.6,+0.6,1.7)(0.8,0.4,0.8)
     (0.8,-0.8,-0.4)
  \pstThreeDDot[RotY=15,drawCoor=true,linecolor=blue](1.6,+0.6,1.7)
\end{pspicture}
```




```
\def\radius{4 }\def\PhiI{20 }\def\PhiII{50 }
%
\def\RadIs{\radius \PhiI sin mul}
\def\RadIc{\radius \PhiI cos mul}
\def\RadIIs{\radius \PhiII sin mul}
\def\RadIIc{\radius \PhiII cos mul}
\begin{pspicture}(-4,-4)(4,5)
  \psset{Alpha=45,Beta=30,linestyle=dashed}
  \pstThreeDCoor[linestyle=solid,xMin=-5,xMax=5,yMax=5,zMax=5,IIIDticks]
```

```
  \pstThreeDEllipse[linecolor=red](0,0,0)(0,\radius,0)(0,0,\radius)
  \pstThreeDEllipse(\RadIs,0,0)(0,\RadIc,0)(0,0,\RadIc)
  \pstThreeDEllipse(\RadIIs,0,0)(0,\RadIIc,0)(0,0,\RadIIc)
%
  \pstThreeDEllipse[linestyle=dotted,SphericalCoor](0,0,0)(\radius,90,\PhiI)(\radius,0,0)
  \pstThreeDEllipse[SphericalCoor,
    beginAngle=-90,endAngle=90](0,0,0)(\radius,90,\PhiI)(\radius,0,0)
  \pstThreeDEllipse[linestyle=dotted,SphericalCoor](0,0,0)(\radius,90,\PhiII)(\radius,0,0)
  \pstThreeDEllipse[SphericalCoor,
    beginAngle=-90,endAngle=90](0,0,0)(\radius,90,\PhiII)(\radius,0,0)
%
  \psset{linecolor=blue,arrows=->,arrowscale=2,linewidth=1.5pt,linestyle=solid}
    \pstThreeDEllipse[SphericalCoor,beginAngle=\PhiI,endAngle=\PhiII]%
    (0,0,0)(\radius,90,\PhiII)(\radius,0,0)
    \pstThreeDEllipse[beginAngle=\PhiII,endAngle=\PhiI](\RadIIs,0,0)(0,\RadIIc,0)(0,0,\RadIIc)
    \pstThreeDEllipse[SphericalCoor,beginAngle=\PhiII,endAngle=\PhiI]%
    (0,0,0)(\radius,90,\PhiI)(\radius,0,0)
    \pstThreeDEllipse[beginAngle=\PhiI,endAngle=\PhiII](\RadIs,0,0)(0,\RadIc,0)(0,0,\RadIc)
\end{pspicture}
\begin{pspicture}(-4,-4)(4,5)

[ ... ]

  \pstThreeDEllipse[linestyle=dotted,SphericalCoor](0,0,0)(\radius,90,\PhiI)(\radius,0,0)
  \pstThreeDEllipse[SphericalCoor,
    beginAngle=-90,endAngle=90](0,0,0)(\radius,90,\PhiI)(\radius,0,0)
  \pstThreeDEllipse[linestyle=dotted,SphericalCoor](0,0,0)(\radius,90,\PhiII)(\radius,0,0)
  \pstThreeDEllipse[SphericalCoor,
    beginAngle=-90,endAngle=90](0,0,0)(\radius,90,\PhiII)(\radius,0,0)
%
  \pscustom[fillstyle=solid,fillcolor=blue]{
    \pstThreeDEllipse[SphericalCoor,beginAngle=\PhiI,endAngle=\PhiII]%
    (0,0,0)(\radius,90,\PhiII)(\radius,0,0)
    \pstThreeDEllipse[beginAngle=\PhiII,endAngle=\PhiI](\RadIIs,0,0)(0,\RadIIc,0)(0,0,\RadIIc)
    \pstThreeDEllipse[SphericalCoor,beginAngle=\PhiII,endAngle=\PhiI]%
    (0,0,0)(\radius,90,\PhiI)(\radius,0,0)
    \pstThreeDEllipse[beginAngle=\PhiI,endAngle=\PhiII](\RadIs,0,0)(0,\RadIc,0)(0,0,\RadIc)
  }
\end{pspicture}
```

14.4 \pstParaboloid

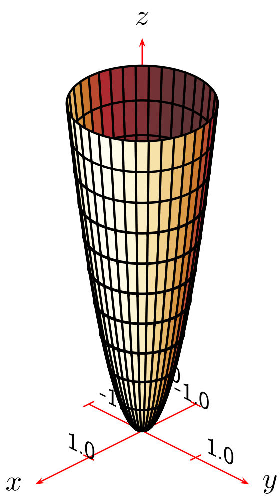
The syntax is

```
\pstParaboloid[Parameter]{height}{radius}
```

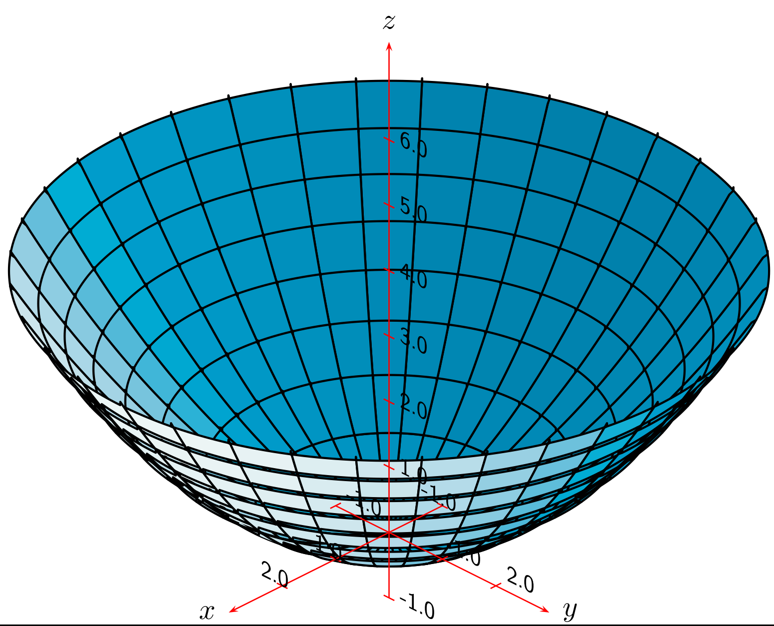
`height` and `radius` depend to each other, it is the radius of the circle at the height. By default the paraboloid is placed in the origin of coordinate system, but with `\pstThreeDput` it can be placed anywhere. The possible options are listed in table 2. The segment color must be set as a cmyk color `SegmentColor={[cmyk]{c,m,y,k}}` in parenthesis, otherwise `xcolor` cannot read the values. A white color is given by `SegmentColor={[cmyk]{0,0,0,0}}`.

Table 2: Options for the `\pstParaboloid` macro

Option name	**value**
`SegmentColor`	cmyk color for the segments (0.2,0.6,1,0)
`showInside`	show inside (true)
`increment`	number for the segments (10)

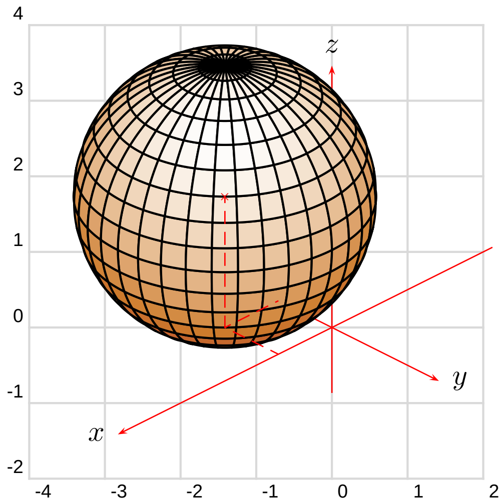



```
\begin{pspicture}(-2,-1)(2,5)
\pstThreeDCoor[xMax=2,yMax=2,zMin=0,zMax=6,IIIDticks]%
\pstParaboloid{5}{1}% Höhe 5 und Radius 1
\end{pspicture}
```




```
\begin{pspicture}(-.5\linewidth,-1)(.5\linewidth,7.5)
\pstParaboloid[showInside=false,SegmentColor={[cmyk]{0.8,0.1,.11,0}}]{4}{5}%
\pstThreeDCoor[xMax=3,yMax=3,zMax=7.5,IIIDticks]
\end{pspicture}
```

15 Spheres



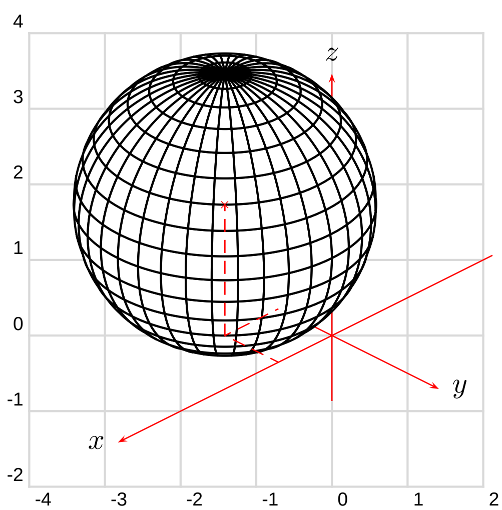

```
\begin{pspicture}(-4,-2.25)(2,4.25)\psgrid
  \pstThreeDCoor[xMin=-3,yMax=2]
  \pstThreeDSphere(1,-1,2){2}
  \pstThreeDDot[dotstyle=x,linecolor=red,drawCoor=true](1,-1,2)
\end{pspicture}
```

`\pstThreeDSphere[<options>](x,y,z){Radius}`

`(x,y,z)` is the center of the sphere and possible options are listed in table 3. The segment color must be set as a cmyk color `SegmentColor={[cmyk]{c,m,y,k}}` in parenthesis, otherwise `xcolor` cannot read the values. A white color is given by `SegmentColor={[cmyk]{0,0,0,0}}`.

Table 3: Options for the sphere macro

Option name	**value**
`SegmentColor`	cmyk color for the segments (0.2,0.6,1,0)
`increment`	number for the segments (10)




```
\begin{pspicture}(-4,-2.25)(2,4.25)\psgrid
  \pstThreeDCoor[xMin=-3,yMax=2]
  \pstThreeDSphere[SegmentColor={[cmyk]{0,0,0,0}}](1,-1,2){2}
  \pstThreeDDot[dotstyle=x,linecolor=red,drawCoor=true](1,-1,2)
\end{pspicture}
```

16 Mathematical functions

There are two macros for plotting mathematical functions, which work similiar to the one from `pst-plot`.

16.1 Function $f(x, y)$

The macro for plotting functions does not have the same syntax as the one from `pst-plot`[4], but it is used in the same way:

```
\psplotThreeD[<options>](xMin,xMax)(yMin,yMax){<the function>}
```

The function has to be written in PostScript code and the only valid variable names are `x` and `y`, f.ex: `{x dup mul y dup mul add sqrt}` for the math expression $\sqrt{x^2 + y^2}$. The macro has the same plotstyle options as `psplot`, except the `plotpoints`-option which is split into one for `x` and one for `y` (table 4).

Table 4: Options for the plot Macros

Option name	value
`plotstyle`	`dots`
	`line`
	`polygon`
	`curve`
	`ecurve`
	`ccurve`
	`none` (default)
`showpoints`	default is false
`xPlotpoints`	default is 25
`yPlotpoints`	default is 25
`drawStyle`	default is `xLines`
	`yLines`
	`xyLines`
	`yxLines`
`hiddenLine`	default is false

The equation 7 is plotted with the following parameters and seen in figure 6.

$$z = 10\left(x^3 + xy^4 - \frac{x}{5}\right)e^{-\left(x^2+y^2\right)} + e^{-\left((x-1.225)^2+y^2\right)} \tag{7}$$

The function is calculated within two loops:

```
for (float y=yMin; y<yMax; y+=dy)
    for (float x=xMin; x<xMax; x+=dx)
         z=f(x,y);
```

It depends to the inner loop in which direction the curves are drawn. There are four possible values for the option `drawStyle` :

- `xLines` (default) Curves are drawn in x direction
- `yLines` Curves are drawn in y direction
- `xyLines` Curves are first drawn in x and then in y direction
- `yxLines` Curves are first drawn in y and then in x direction

In fact of the inner loop it is only possible to get a closed curve in the defined direction. For lines in x direction less `yPlotpoints` are no problem, in difference to `xPlotpoints`, especially for the plotstyle options `line` and `dots`.

Drawing three dimensional functions with curves which are transparent makes it difficult to see if a point is before or behind another one. `\psplotThreeD` has an option `hiddenLine` for a primitive hidden line mode, which only works when the y-intervall is defined in a way that $y_2 > y_1$. Then every new curve is plotted over the forgoing one and filled with the color white. Figure 7 is the same as figure 6, only with the option `hiddenLine=true`.

```
\begin{pspicture}(-6,-4)(6,5)\psgrid
  \psset{Beta=15}
  \psplotThreeD[plotstyle=line,drawStyle=xLines,% is the default anyway
    yPlotpoints=50,xPlotpoints=50,linewidth=1pt](-4,4)(-4,4){%
    x 3 exp x y 4 exp mul add x 5 div sub 10 mul
    2.729 x dup mul y dup mul add neg exp mul
    2.729 x 1.225 sub dup mul y dup mul add neg exp add}
  \pstThreeDCoor[xMin=-1,xMax=5,yMin=-1,yMax=5,zMin=-1,zMax=5]
\end{pspicture}
```

16.2 Parametric Plots

Parametric plots are only possible for drawing curves or areas. The syntax for this plot macro is:

`\parametricplotThreeD(t1,t2)(u1,u2){<three parametric functions x y z}`

The only possible variables are `t` and `u` with `t1,t2` and `u1,u2` as the range for the parameters. The order for the functions is not important and `u` may be optional when having only a three dimensional curve and not an area.

$$\begin{aligned} x &= f(t,u) \\ y &= f(t,u) \\ z &= f(t,u) \end{aligned} \tag{8}$$

To draw a spiral we have the parametric functions:

$$\begin{aligned} x &= r\cos t \\ y &= r\sin t \\ z &= t/600 \end{aligned} \tag{9}$$

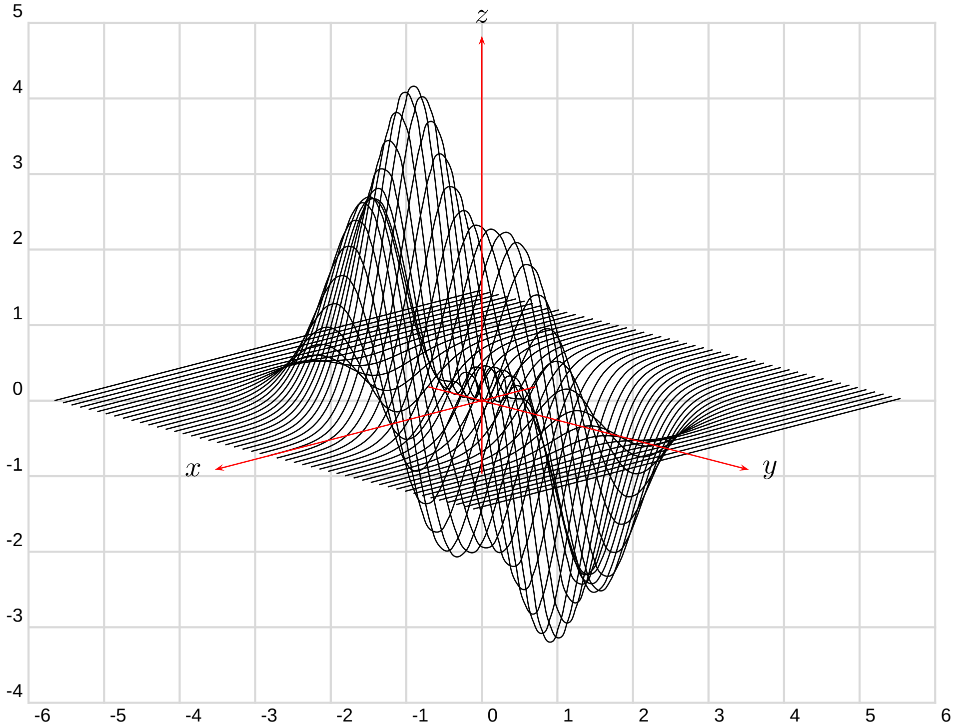


Figure 6: Plot of the equation 7

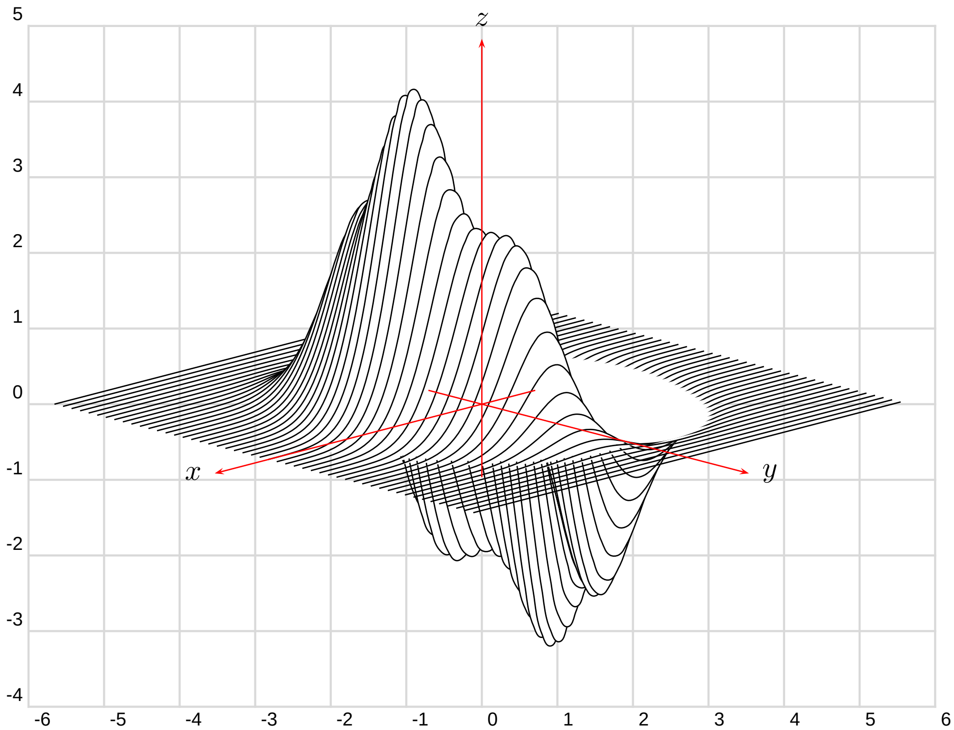


Figure 7: Plot of the equation 7 with the `hiddenLine=true` option

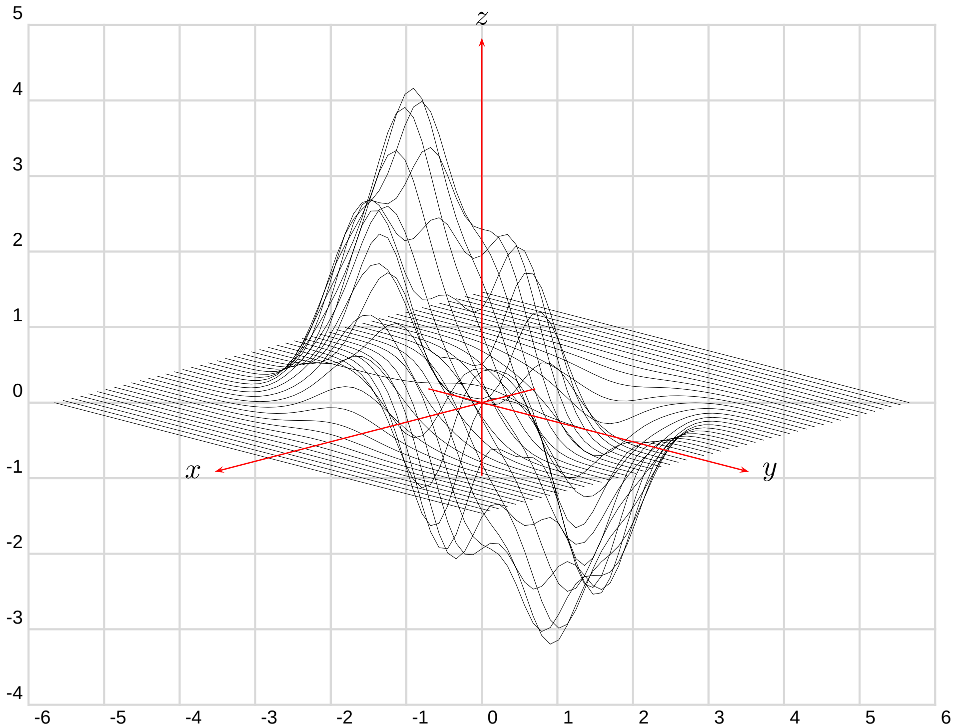


Figure 8: Plot of the equation 7 with the `drawStyle=yLines` option

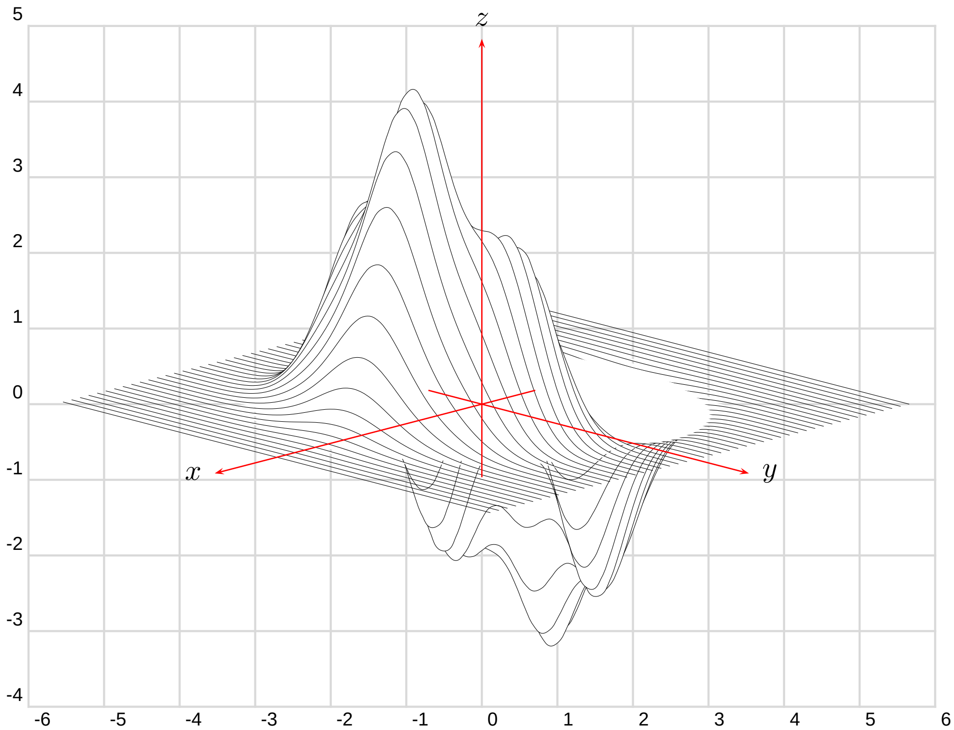


Figure 9: Plot of the equation 7 with the `drawStyle=yLines` and `hiddenLine=true` option

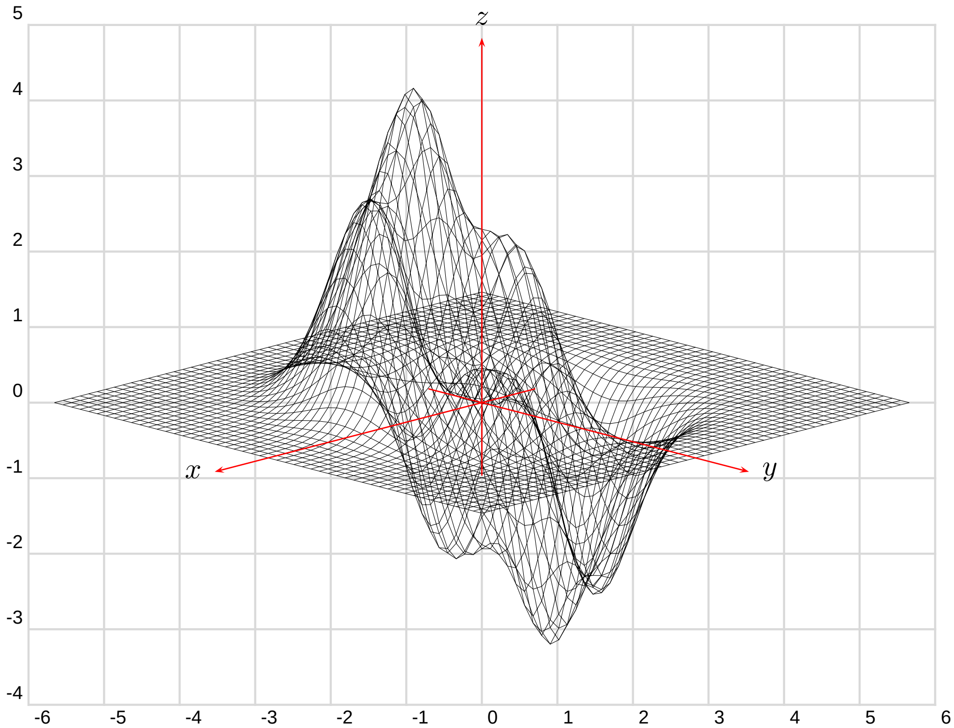


Figure 10: Plot of the equation 7 with the `drawStyle=xyLines` option

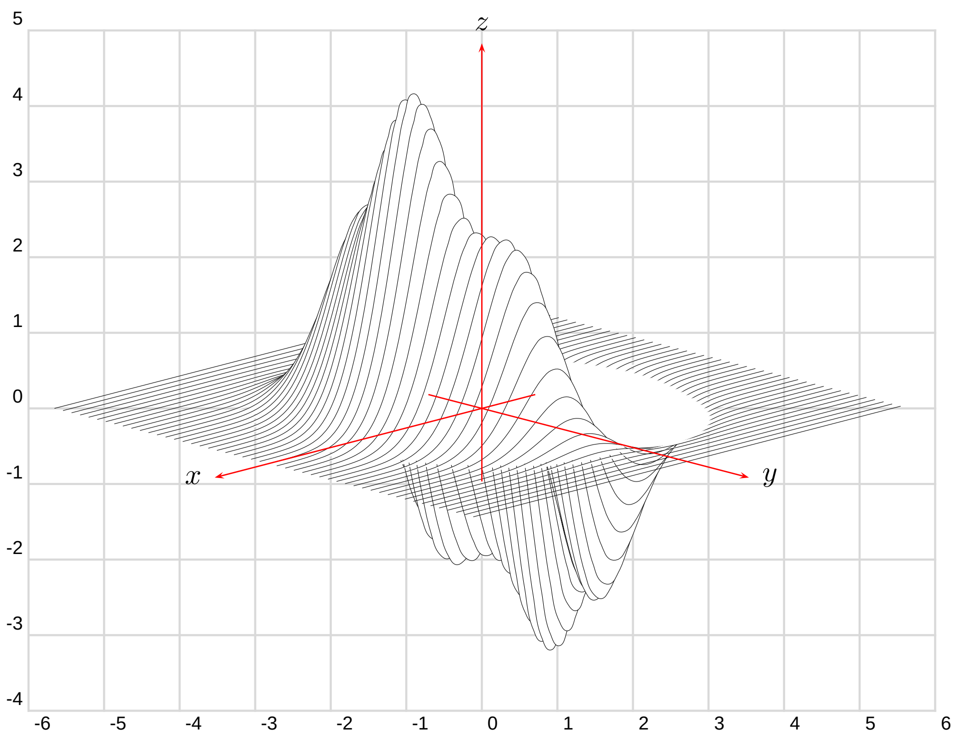


Figure 11: Plot of the equation 7 with the `drawStyle=xLines` and `hiddenLine=true` option

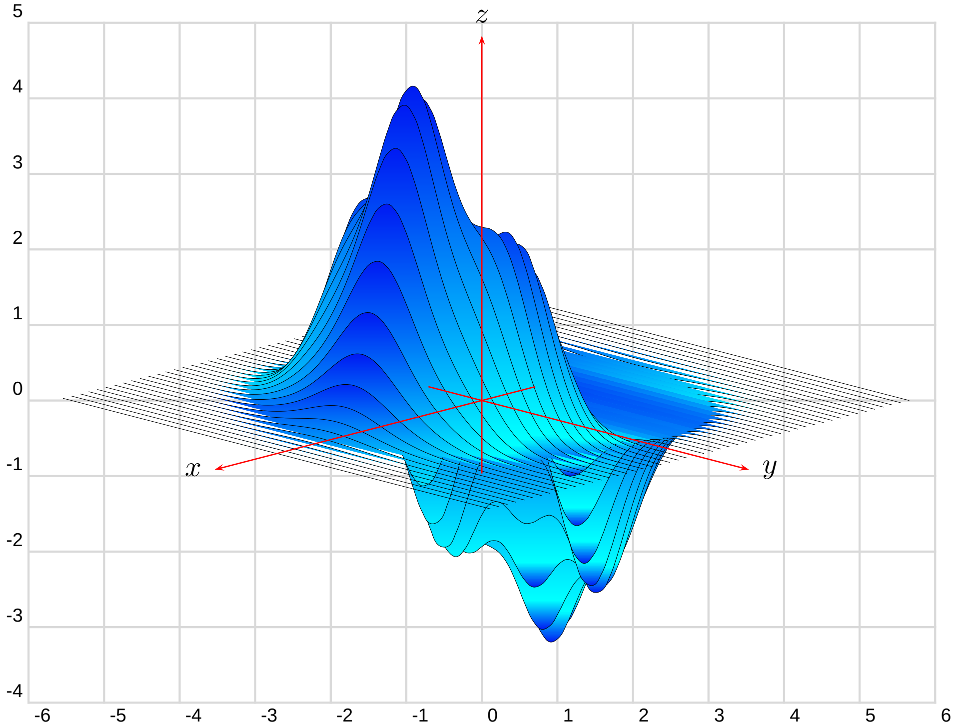
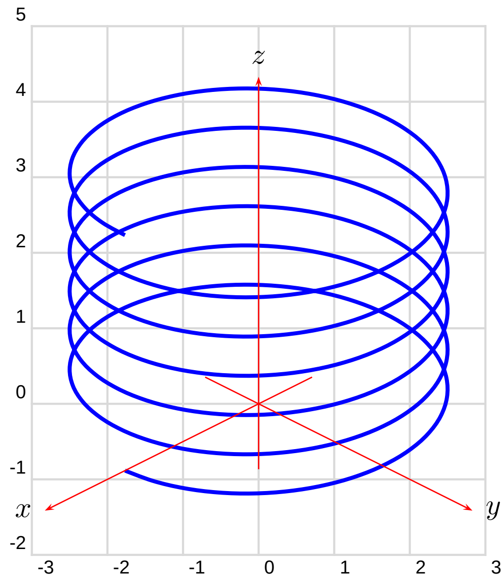


Figure 12: Plot of the equation 7 with the `drawStyle=yLines` and `hiddenLine=true` option

In the example the t value is divided by 600 for the `z` coordinate, because we have the values for t in degrees, here with a range of $0^{\circ} \ldots 2160^{\circ}$. Drawing a curve in a three dimensional coordinate system does only require one parameter, which has to be by default `t`. In this case we do not need all parameters, so that one can write

```
\parametricplotThreeD(t1,t2){<three parametric functions x y z}
```

which is the same as `(0,0)` for the parameter `u`.




```
\begin{pspicture}(-3.25,-2.25)(3.25,5.25)\psgrid
 \parametricplotThreeD[xPlotpoints=200,linecolor=blue,%
  linewidth=1.5pt,plotstyle=curve](0,2160){%
  2.5 t cos mul 2.5 t sin mul t 600 div}
 \pstThreeDCoor[zMax=5]
\end{pspicture}
```

Instead of using the `\pstThreeDSphere` macro (see section 15) it is also possible to use parametric functions for a sphere. The macro plots continous lines only for the `t` parameter, so a sphere plotted with the longitudes need the parameter equations as

$$\begin{aligned} x &= \cos t \cdot \sin u \\ y &= \cos t \cdot \cos u \\ z &= \sin t \end{aligned} \tag{10}$$

The same is possible for a sphere drawn with the latitudes:

$$\begin{aligned} x &= \cos u \cdot \sin t \\ y &= \cos u \cdot \cos t \\ z &= \sin u \end{aligned} \tag{11}$$

and at last both together is also not a problem when having these parametric functions together in one `pspicture` environment (see figure 13).

```
\begin{pspicture}(-1,-1)(1,1)\psgrid
\parametricplotThreeD[plotstyle=curve,yPlotpoints=40](0,360)(0,360){%
  t cos u sin mul t cos u cos mul t sin
}
\parametricplotThreeD[plotstyle=curve,yPlotpoints=40](0,360)(0,360){%
  u cos t sin mul  u cos t cos mul u sin
}
\end{pspicture}
```

17 Plotting data files

There are the same conventions for data files which holds 3D coordinates, than for the 2D one. For example:

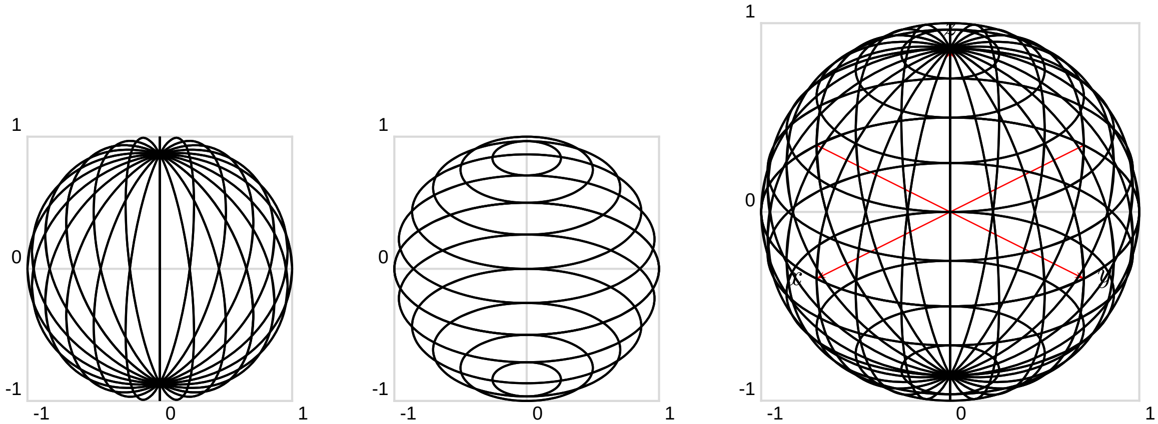


Figure 13: Different Views of the same Parametric Functions

```
 0.0000  1.0000  0.0000
-0.4207  0.9972  0.0191
....

0.0000, 1.0000, 0.0000
-0.4207, 0.9972, 0.0191
....

(0.0000,1.0000,0.0000)
(-0.4207,0.9972,0.0191)
....

{0.0000,1.0000,0.0000}
{-0.4207,0.9972,0.0191}
....
```

There are the same three plot functions:

```
\fileplotThreeD[<options>]{<datafile>}
\dataplotThreeD[<options>]{<data object>}
\listplotThreeD[<options>]{<data object>}
```

The in the following examples used data file has 446 entries like

```
6.26093349..., 2.55876582..., 8.131984...
```

This may take some time on slow machines when using the `\listplotThreeD` macro. The possible options for the lines are the ones from table 4.

17.1 \fileplotThreeD

The syntax is very easy

```
\fileplotThreeD[<options>]{<datafile>}
```

If the data file is not in the same directory than the document, insert the file name with the full path. Figure 15 shows a file plot with the option `linestyle=line`.

17.2 \dataplotThreeD

The syntax is

```
\dataplotThreeD[<options>]{<data object>}
```

In difference to the macro `\fileplotThreeD` the `\dataplotThreeD` cannot plot any external data without reading this with the macro `\readdata` which reads external data and save it in a macro, f.ex.: `\dataThreeD`.[1]

```
\readdata{<data object>}{<datafile>}
```

17.3 \listplotThreeD

The syntax is

```
\listplotThreeD[<options>]{<data object>}
```

`\listplotThreeD` ist similiar to `\dataplotThreeD`, so it cannot plot any external data in a direct way, too. But `\readdata` reads external data and saves it in a macro, f.ex.: `\dataThreeD`.[1] `\listplot` can handle some additional PostScript code, which can be appended to the data object, f.ex.:

```
\dataread{\data}{data3D.Roessler}
\newcommand{\dataThreeDDraft}{%
  \data\space
  gsave          % save grafic status
  /Helvetica findfont 40 scalefont setfont
  45 rotate      % rotate 45 degrees
  0.9 setgray    % 1 ist white
  -60 30 moveto (DRAFT) show
  grestore
}
```

Figure 16 shows what happens with this code. For another example see [4], where the macro `ScalePoints` is modified. This macro is in `pst-3dplot` called `ScalePointsThreeD`.

18 Utility macros

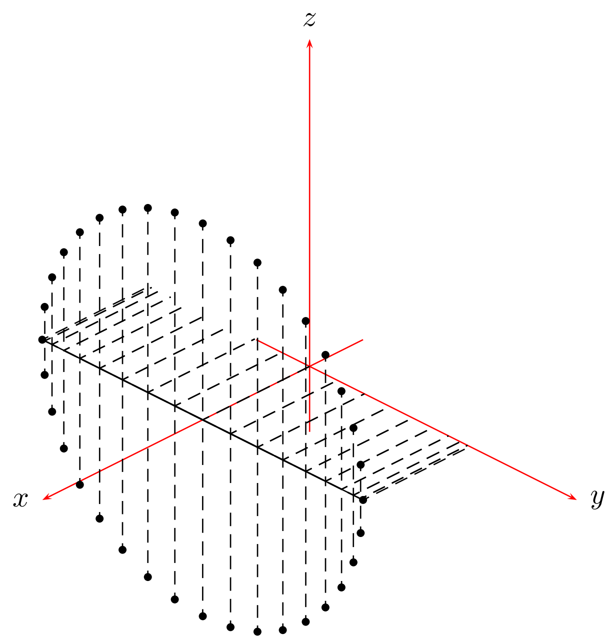
18.1 Rotation of three dimensional coordinates

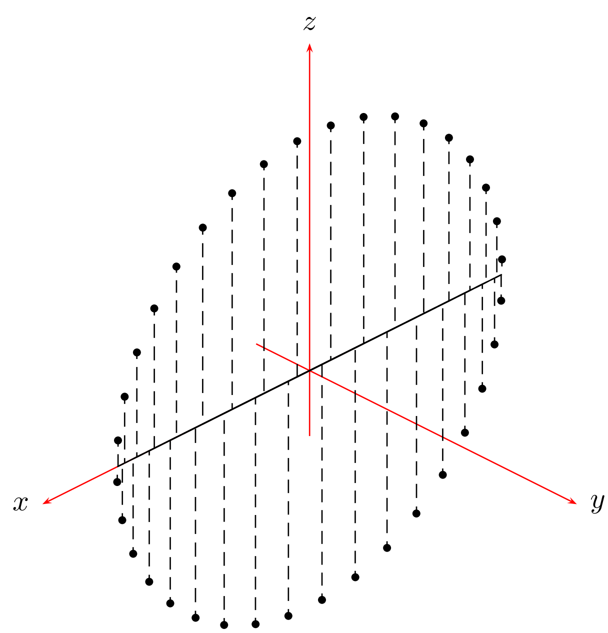
With the three optional arguments `RotX` `RotY` `RotZ` one can rotate a three dimensional point. This makes only sense when one wants to save the coordinates. In general it is more powerful to use directly the optional parameters `RotX`, `RotY`, `RotZ` for the plot macros. However, the macro syntax is

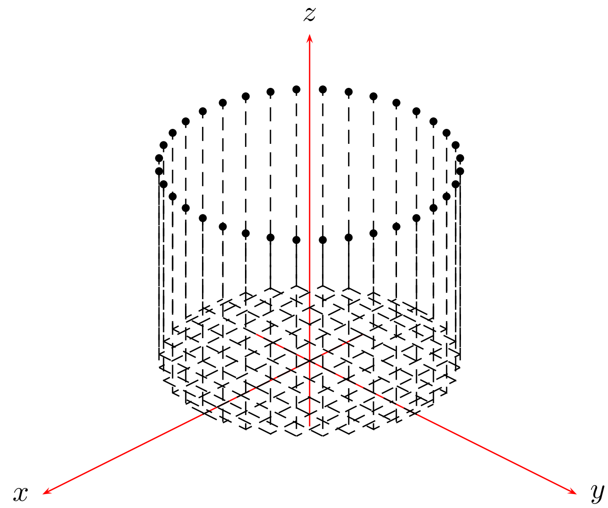
```
\pstRotPOintIIID[RotX=...,RotY=...,RotZ=...](<x,y,z>)<\xVal><\yVal><\zVal>
```

the `\xVal \yVal \zVal` hold the new rotated coordinates and must be defined by the user like `\def\xVal{}`, where the name of the macro is not important.

The rotation angles are all predefined to 0 degrees.








```
\def\xVal{}\def\yVal{}\def\zVal{}
\begin{pspicture}(-6,-4)(6,5)
  \pstThreeDCoor[xMin=-1,xMax=5,yMin=-1,yMax=5,zMin=-1,zMax=5]
  \multido{\iA=0+10}{36}{\pstRotPointIIID[RotX=\iA](2,0,3){\xVal}{\yVal}{\zVal}
    \pstThreeDDot[drawCoor=true](\xVal,\yVal,\zVal)
  }
\end{pspicture}

\begin{pspicture}(-6,-4)(6,5)
  \pstThreeDCoor[xMin=-1,xMax=5,yMin=-1,yMax=5,zMin=-1,zMax=5]
  \multido{\iA=0+10}{36}{\pstRotPointIIID[RotY=\iA](2,0,3){\xVal}{\yVal}{\zVal}
    \pstThreeDDot[drawCoor=true](\xVal,\yVal,\zVal)
  }
\end{pspicture}

\begin{pspicture}(-6,-4)(6,5)
  \pstThreeDCoor[xMin=-1,xMax=5,yMin=-1,yMax=5,zMin=-1,zMax=5]
  \multido{\iA=0+10}{36}{\pstRotPointIIID[RotZ=\iA](2,0,3){\xVal}{\yVal}{\zVal}
    \pstThreeDDot[drawCoor=true](\xVal,\yVal,\zVal)
  }
\end{pspicture}
```

18.2 Transformation of coordinates

To run the macros with more than 9 parameters `pst-3dplot` uses the syntax `(#1)` for a collection of three coordinates `(#1,#2,#3)`. To handle these triple in PostScript the following macro is used, which converts the parameter `#1` into a sequence of the three coordinates, dived by a space. The syntax is:

```
\getThreeDCoor(<vector>)<\macro>
```

`\macro` holds the sequence of the three coordinates `x y z`, divided by a space.

18.3 Adding two vectors

The syntax is

```
\pstaddThreeDVec(<vector A>)(<vector B>)\tempa\tempb\tempc
```

`\tempa\tempb\tempc` must be user or system defined macros, which holds the three coordinates of the vector $\vec{C} = \vec{A} + \vec{B}$.

18.4 Substract two vectors

The syntax is

```
\pstsubThreeDVec(<vector A>)(<vector B>)\tempa\tempb\tempc
```

`\tempa\tempb\tempc` must be user or system defined macros, which holds the three coordinates of the vector $\vec{C} = \vec{A} - \vec{B}$.

19 PDF output

`pst-3dplot` is based on the popular `pstricks` package and writes pure PostScriptcode[2], so it is not possible to run TEX files with pdfLATEX when there are pstricks macros in the document. If you still need a PDF output use one of the following possibilities:

- package `pdftricks.sty`[5]
- the for Linux free available program VTeX/Lnx[3]
- build the PDF with `ps2pdf` (`dvi`→`ps`→`pdf`)
- use the `ps4pdf` package.[4]

If you need package `graphicx.sty` load it before any `pstricks` package. You do not need to load `pstricks.sty`, it will be done by `pst-3dplot` by default.

[3] http://www.micropress-inc.com/linux/
[4] http://www.perce.de/LaTeX/ps4pdf/

20 FAQ

- The labels for the axis are not right placed in the preview.

  Be sure that you view your output with a dvi viewer which can show PostScript code, like kdvi but not xdvi. It is better to run `dvips` and then view the ps-file with `gv`.

- The three axes have a wrong intersection point.

  Be sure that you have the "newest" `pst-node.tex` file

  ```
  \def\fileversion{97 patch 11}
  \def\filedate{2000/11/09}
  ```

  and the "newest" `pst-plot.tex`

  ```
  \def\fileversion{97 patch 2}
  \def\filedate{1999/12/12}
  ```

- Using `amsmath` and `\hat` or other accents as label for the axes gives an error. In this case save prevent expanding with e.g.: `\psset{nameX=$\noexpand\hat{x}$}`.

21 Credits

Bruce Burlton | Christophe Jorssen | Chris Kuklewicz | Thorsten Suhling

References

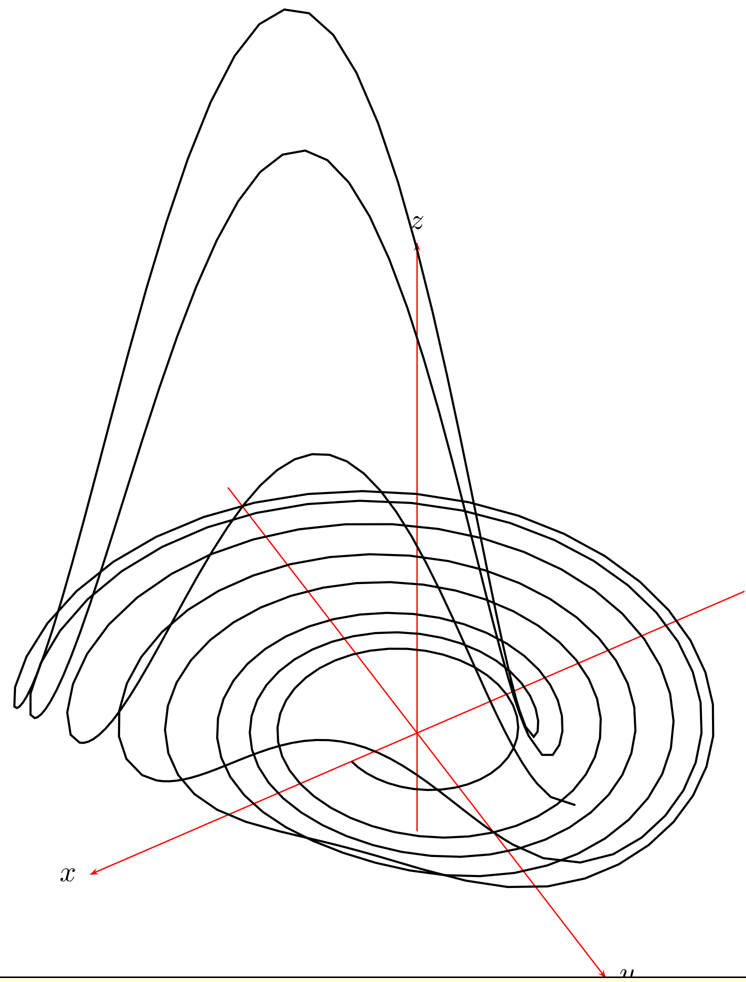
[1] Laura E. Jackson and Herbert Voß. Die Plot-Funktionen von `pst-plot`. *Die TEXnische Komödie*, 2/02:27–34, June 2002.

[2] Nikolai G. Kollock. *PostScript richtig eingesetzt: vom Konzept zum praktischen Einsatz*. IWT, Vaterstetten, 1989.

[3] Manuel Luque. *Vue en 3D*. http://members.aol.com/Mluque5130/vue3d16112002.zip, 2002.

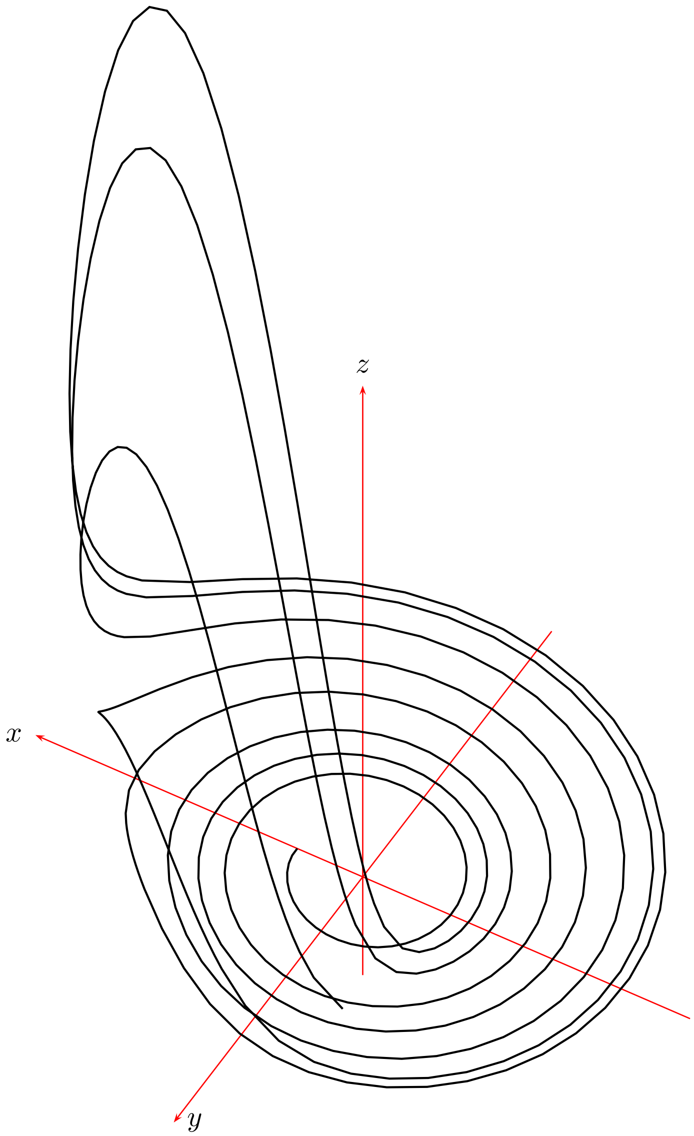
[4] Herbert Voß. Die mathematischen Funktionen von Postscript. *Die TEXnische Komödie*, 1/02:40–47, March 2002.

[5] Herbert Voss. *PSTricks Support for pdf*. http://PSTricks.de/pdf/pdfoutput.phtml, 2002.



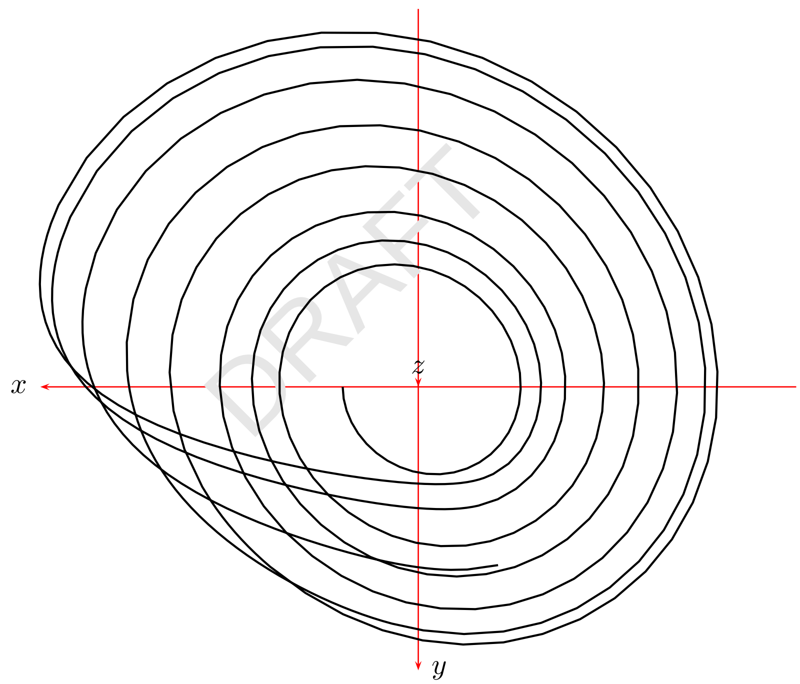

```
\begin{pspicture}(-6,-3)(6,10)
  \psset{xunit=0.5cm,yunit=0.75cm,Alpha=30,Beta=30}% the global parameters
  \pstThreeDCoor[xMin=-10,xMax=10,yMin=-10,yMax=10,zMin=-2,zMax=10]
  \fileplotThreeD[plotstyle=line]{data3D.Roessler}
\end{pspicture}%
```

Figure 14: Demonstration of `\fileplotThreeD` with `Alpha=30` and `Beta=15`




```
\begin{pspicture}(-4.5,-3.5)(4,11)
  \psset{xunit=0.5cm,yunit=0.75cm,Alpha=-30}
  \pstThreeDCoor[xMin=-10,xMax=10,yMin=-10,yMax=10,
     zMin=-2,zMax=10]
  \dataplotThreeD[plotstyle=line]{\dataThreeD}
\end{pspicture}%
```

Figure 15: Demonstration of `\dataplotThreeD` with `Alpha=-30` and `Beta=30`




```
\begin{pspicture}(-5,-4)(5,4)
  \psset{xunit=0.5cm,yunit=0.5cm,Alpha=0,Beta=90}
  \pstThreeDCoor[xMin=-10,xMax=10,yMin=-10,yMax=7.5,zMin=-2,zMax=10]
  \listplotThreeD[plotstyle=line]{\dataThreeDDraft}
\end{pspicture}%
```

Figure 16: Demonstration of `\listplotThreeD` with a view from above (`Alpha=0` and `Beta=90`) and some additional PostScript code