

3D plots: PST-3dplot v1.2b

Documentation

Herbert Voß*

November 2, 2003

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 similar to the `pst-plot` package for two dimensional objects and amthematical functions.

Contents

1 The Parallel Projection	2	13 Mathematical Functions	10
		13.1 Function $f(x, y)$	10
		13.2 Parametric Plots	11
2 Options	2	14 Plotting Data Files	16
3 Coordinate Axes	2	14.1 <code>\fileplotThreeD</code>	16
4 Put	4	14.2 <code>\dataplotThreeD</code>	17
4.1 <code>pstThreeDPut</code>	4	14.3 <code>\listplotThreeD</code>	17
4.2 <code>pstPlanePut</code>	4	15 Utility macros	18
5 Nodes	6	15.1 Transformation of coordinates . . .	18
6 Dots	6	15.2 Adding two vectors	18
7 Lines	6	15.3 Subtract two vectors	18
8 Triangles	7	16 PDF Output	18
9 Squares	8	17 FAQ	18
10 Boxes	8	18 Change Log	19
11 Ellipses and Circles	9	19 Credits	19
11.1 Options	9		
11.2 Ellipse	9		
11.3 Circle	10		
12 Spheres	10		

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) .

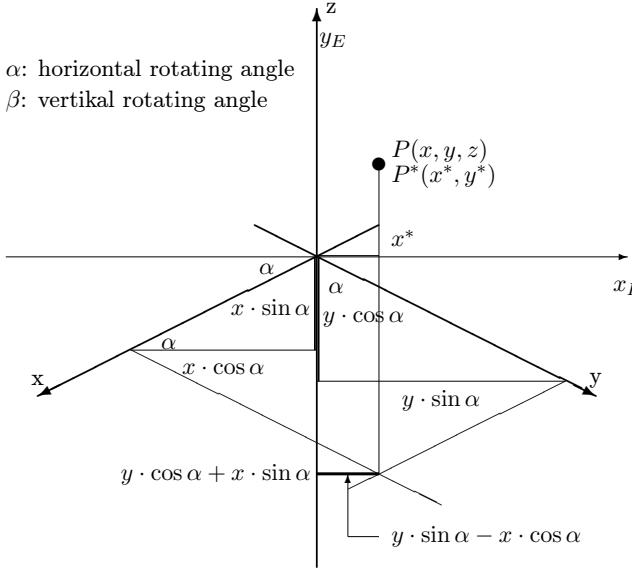


Figure 1: Lengths in a three dimensional System

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.

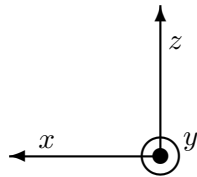


Figure 2: Coordinate System for $\alpha = \beta = 0$ (y-axis comes out of the paper plane)

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.

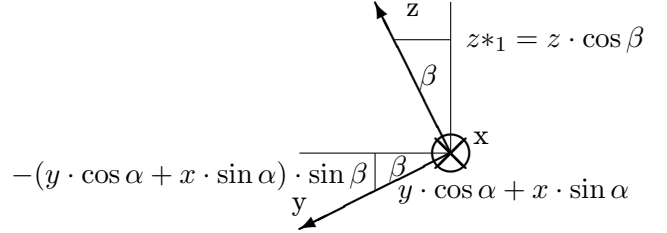


Figure 3: Coordinate System for $\alpha = 0$ and $\beta > 0$ (x-axis goes into the paper plane)

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 is the difference 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:

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

or written in matrix form:

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

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 Coordinate Axes

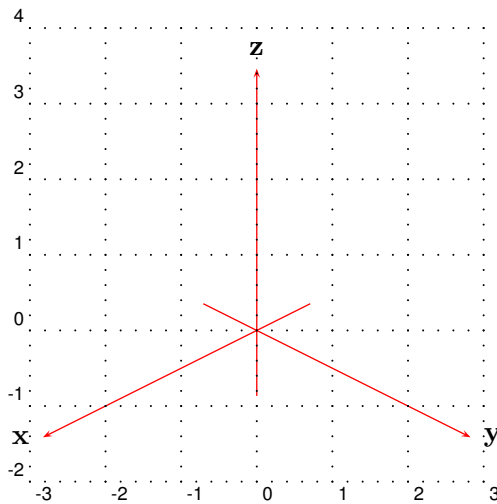
The syntax for drawing the coordinate axes is

`\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.

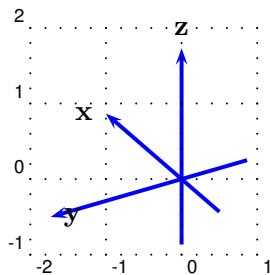
Without any options we get the default view with:

```
xMin=-1,xMax=4,
yMin=-1,yMax=4,
zMin=-1,zMax=4,
Alpha=45,Beta=30
```

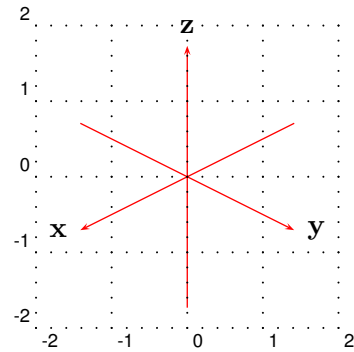


There are no restrictions for the angles and the max and min values for the axes; all `pstricks` options are possible as well. The following example changes the color and the width of the axes.

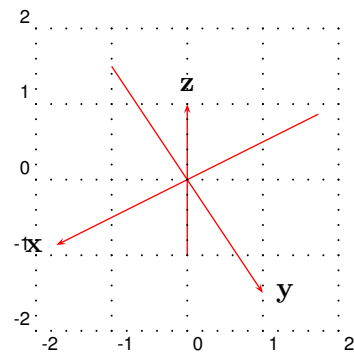
The angles `Alpha` and `Beta` are important to all macros and should always be set with `psset` to make them global to all other macros. Otherwise they are only local inside the macro to which they are passed.



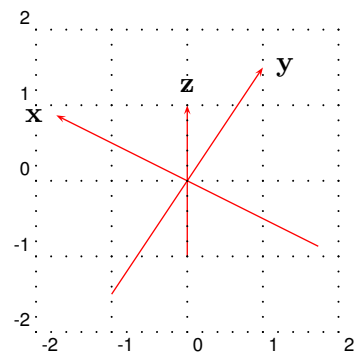
```
1 \begin{pspicture}(-2,-1.25)(1,2.25)
2 \pstThreeDCoor[
3 linewidth=1.5pt,linecolor=blue,%
4 xMin=-1,xMax=2,yMin=-1,yMax=2,%
5 zMin=-1,zMax=2,Alpha=-60,Beta=30]
6 \psgrid
7 \end{pspicture}
```



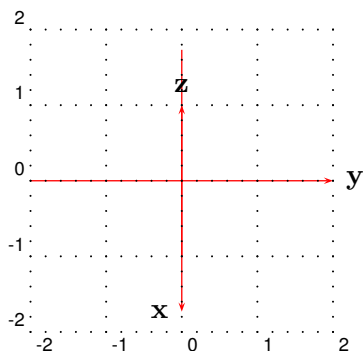
```
1 \begin{pspicture}(-2,-2)(2,2)
2 \pstThreeDCoor[
3 xMin=-2,xMax=2,%
4 yMin=-2,yMax=2,%
5 zMin=-2,zMax=2]
6 \psgrid
7 \end{pspicture}
```



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



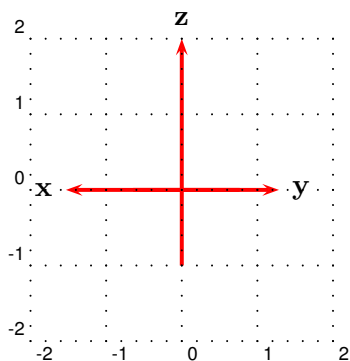
```
1 \begin{pspicture}(-2,-2)(2,2)
2 \pstThreeDCoor[
3 xMin=-2,xMax=2,%
4 yMin=-2,yMax=2,%
5 zMin=-2,zMax=2,%
6 Alpha=30,Beta=-60]
7 \psgrid
8 \end{pspicture}
```



```

1 \begin{pspicture}(-2,-2)(2,2)
2 \pstThreeDCoor[%
3   xMin=-2,xMax=2,%
4   yMin=-2,yMax=2,%
5   zMin=-2,zMax=2,%
6   Alpha=90,Beta=60]
7 \psgrid
8 \end{pspicture}

```



```

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

```

4 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.

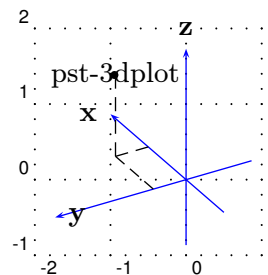


This works only well for the `\pstThreeDPut` macro. The default is `c` and for the `\pstPlanePut` the left baseline `lB`.

4.1 pstThreeDPut

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

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



```

1 \begin{pspicture}(-2,-1.25)(1,2.25)
2 \psgrid
3 \psset{Alpha=-60,Beta=30}
4 \pstThreeDCoor[%
5   linecolor=blue,%
6   xMin=-1,xMax=2,%
7   yMin=-1,yMax=2,%
8   zMin=-1,zMax=2]
9 \pstThreeDPut(1,0.5,1.25){pst-3dplot}
10 \pstThreeDDot[drawCoord=true](1,0.5,1.25)
11 \end{pspicture}

```

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

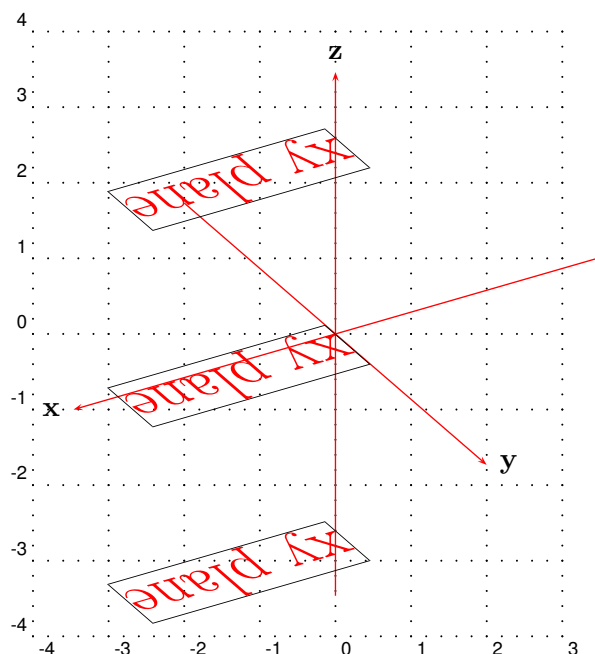
4.2 pstPlanePut

The syntax of the `\pstPlanePut` is

`\pstPlanePut[plane=<2D plane>]%`
`(x,y,z){Object}`

Possible values for the two dimensional plane are `xy` `xz` `yz`. If this optional parameter is missing then `plane=xy` is set. The first letter marks the positive direction for the width and the second for the height.

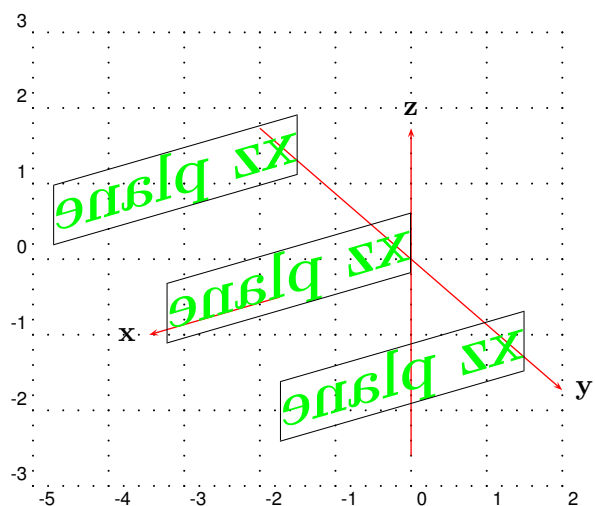
The object can be of any type, in most cases it will be some kind of text. The reference point for the object is the left side and vertically centered, often abbreviated as `lB`. The following examples show for all three planes the same textbox.



```

1 \begin{pspicture} (-4,-4) (3,4)
2 \psgrid
3 \pstThreeDCoor
4 \pstPlanePut[plane=xy] (0,0,-3) {\fbox{\Huge\
   red\textbf{xy plane}}}
5 \pstPlanePut[plane=xy] (0,0,0) {\fbox{\Huge\
   red\textbf{xy plane}}}
6 \pstPlanePut[plane=xy] (0,0,3) {\fbox{\Huge\
   red\textbf{xy plane}}}
7 \end{pspicture}

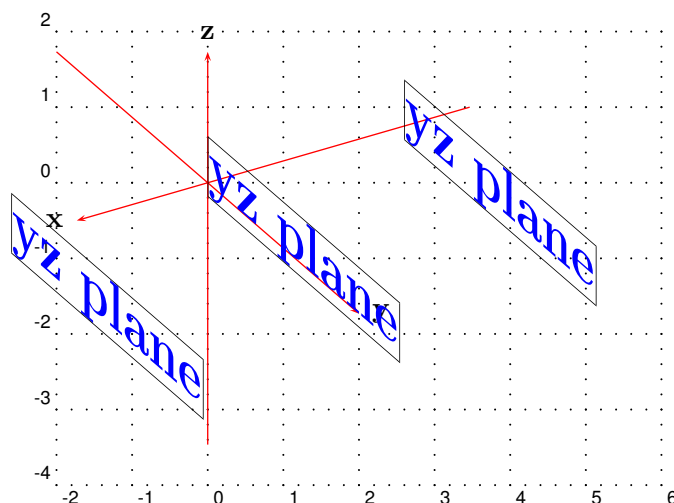
```



```

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

```

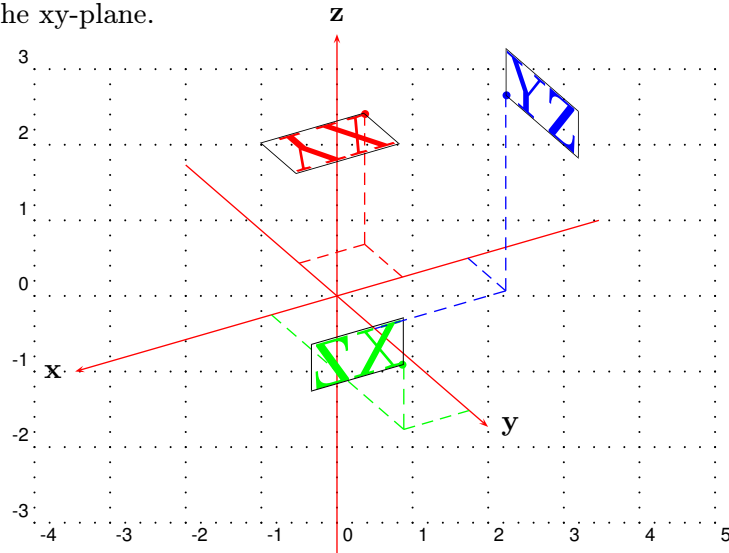


```

1 \begin{pspicture} (-2,-4) (6,2)
2 \psgrid
3 \pstThreeDCoor[xMax=2,zMax=2]
4 \pstPlanePut[plane=yz] (-3,0,0) {\fbox{\Huge\
   blue\textbf{yz plane}}}
5 \pstPlanePut[plane=yz] (0,0,0) {\fbox{\Huge\
   blue\textbf{yz plane}}}
6 \pstPlanePut[plane=yz] (3,0,0) {\fbox{\Huge\
   blue\textbf{yz plane}}}
7 \end{pspicture}
8
9 }

```

The following examples use the `origin` option to show that there are still some problems with the `xy`-plane.



```

1 \psset{origin=lb}
2 \begin{pspicture} (-4,-3) (5,3)
3 \psgrid
4 \pstThreeDCoor
5 \pstThreeDDot[drawCoord=true,linecolor=red]
   (-1,-1,2)
6 \pstPlanePut[plane=xy] (-1,-1,2) {\fbox{\Huge\
   red\textbf{XY}}}
7 \pstThreeDDot[drawCoord=true,linecolor=green]
   (1,3.5,1)
8 \pstPlanePut[plane=xz] (1,3.5,1) {\fbox{\Huge\
   green\textbf{XZ}}}
9 \pstThreeDDot[drawCoord=true,linecolor=blue]
   (-2,1,3)

```

```

10 \pstPlanePut[plane=yz](-2,1,3){\fbox{\Huge\
    blue\textbf{YZ}}}
11 \end{pspicture}

```

5 Nodes

The syntax is

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

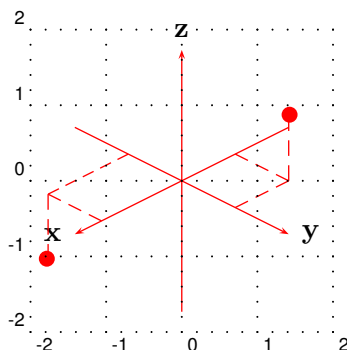
This node is internally a two dimensional node, so it cannot be used as a replacement for the parameters (x,y,z) of a 3D dot, which is possible with the `\psline` macro from `pst-plot`: `\psline{A}{B}`, where A and B are two nodes. It is still on the to do list, that it may also be possible with `pst-3dplot`. 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.

6 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`.

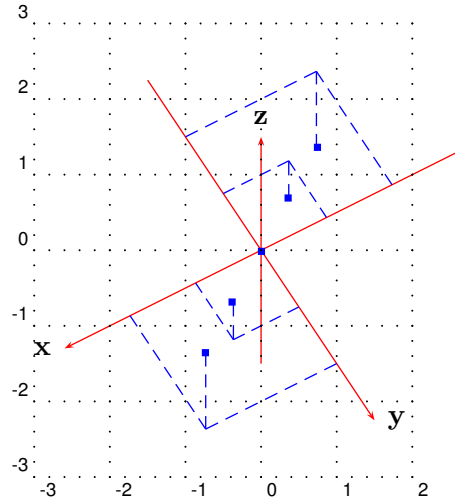


```

1 \begin{pspicture}(-2,-2)(2,2)
2 \pstThreeDCoor
3 \psset{dotstyle=*,dotscale=2,%
4   linecolor=red,%
5   drawCoor=true}
6 \pstThreeDDot(-1,1,1)
7 \pstThreeDDot(1.5,-1,-1)
8 \psgrid
9 \end{pspicture}

```

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



```

1 \psset{xMin=-3,xMax=3,%
2   yMin=-3,yMax=3,zMin=-3,zMax=3}
3 \begin{pspicture}(-3,-3.25)(2.5,3.25)
4 \pstThreeDCoor[Alpha=30,Beta=60]
5 \psset{dotstyle=pentagon*,dotsize=5pt,%
6   linecolor=blue,%
7   drawCoor=true}
8 \multido{\n=-2+1}{5}{
9 \pstThreeDDot(\n,\n,\n)
10 }
11 \psgrid
12 \end{pspicture}

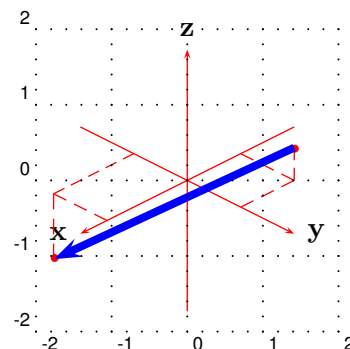
```

7 Lines

The syntax for a three dimensional line is

```
\pstThreeDLine[<options>]%
(x1,y1,z1)(x2,y2,z2)
```

All options for lines from `pst-plot` 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 `"->"`.



```

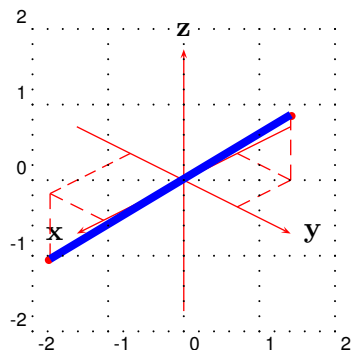
1 \psset{xMin=-2,xMax=2,yMin=-2,yMax=2,%
2   zMin=-2,zMax=2}
3 \begin{pspicture}(-2,-2.25)(2,2.25)
4 \pstThreeDCoor
5 \psset{dotstyle=*,linecolor=red,%

```

```

6 drawCoor=true}
7 \pstThreeDDot (-1,1,0.5)
8 \pstThreeDDot (1.5,-1,-1)
9 \pstThreeDLine[
10 linewidth=3pt,%
11 linecolor=blue,
12 arrows=->%
13 ] (-1,1,0.5) (1.5,-1,-1)
14 \psgrid
15 \end{pspicture}

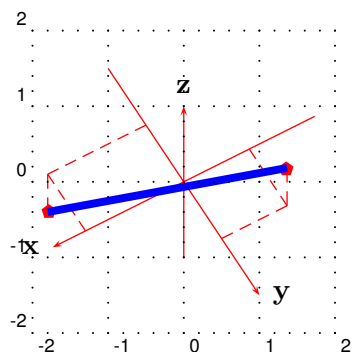
```



```

1 \psset{xMin=-2,xMax=2,yMin=-2,yMax=2,%
2 zMin=-2,zMax=2}
3 \begin{pspicture} (-2,-2) (2,2)
4 \pstThreeDCoor
5 \psset{dotstyle=*,linecolor=red,%
6 drawCoor=true}
7 \pstThreeDDot (-1,1,1)
8 \pstThreeDDot (1.5,-1,-1)
9 \psgrid
10 \end{pspicture}

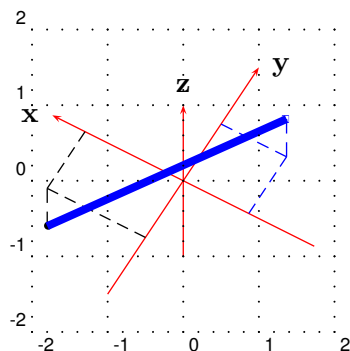
```



```

1 \psset{xMin=-2,xMax=2,yMin=-2,yMax=2,%
2 zMin=-2,zMax=2}
3 \begin{pspicture} (-2,-2) (2,2)
4 \psset{Alpha=30,Beta=60}
5 \pstThreeDCoor
6 \psset{dotstyle=pentagon*,dotsize=5pt,%
7 linecolor=red,%
8 drawCoor=true}
9 \pstThreeDDot (-1,1,1)
10 \pstThreeDDot (1.5,-1,-1)
11 \psgrid
12 \end{pspicture}

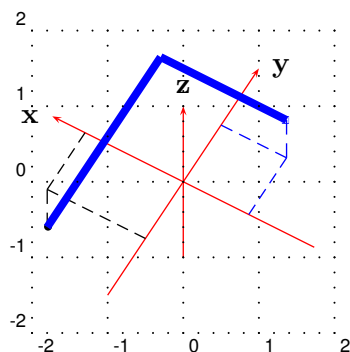
```



```

1 \psset{xMin=-2,xMax=2,%
2 yMin=-2,yMax=2,%
3 zMin=-2,zMax=2}
4 \begin{pspicture} (-2,-2) (2,2)
5 \psset{Alpha=30,Beta=-60}
6 \pstThreeDCoor
7 \pstThreeDDot[dotstyle=square,%
8 linecolor=blue,%
9 drawCoor=true] (-1,1,1)
10 \pstThreeDDot[
11 drawCoor=true] (1.5,-1,-1)
12 \psgrid
13 \end{pspicture}

```



```

1 \begin{pspicture} (-2,-2) (2,2)
2 \psset{Alpha=30,Beta=-60}
3 \pstThreeDCoor
4 \pstThreeDDot[dotstyle=square,linecolor=blue
5 ,%
6 drawCoor=true] (-1,1,1)
7 \pstThreeDDot[drawCoor=true] (1.5,-1,-1)
8 \pstThreeDLine[
9 linewidth=3pt,%
10 linecolor=blue%
11 ] (-1,1,1) (1.5,2,-1)
12 \pstThreeDLine[
13 linewidth=3pt,%
14 linecolor=blue%
15 ] (1.5,2,-1) (1.5,-1,-1)
16 \psgrid

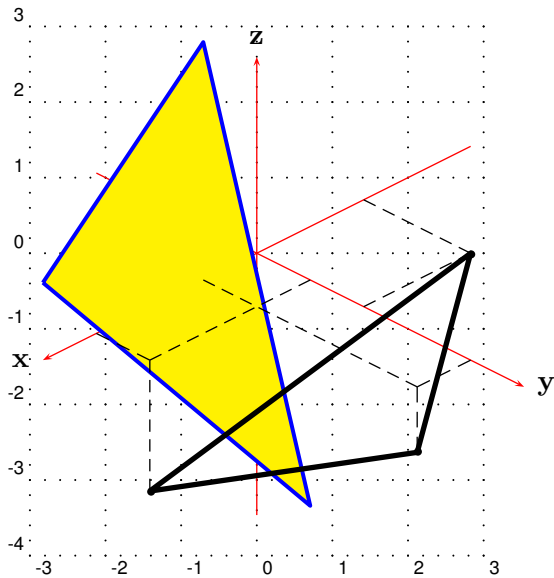
```

8 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`.



```

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

```

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

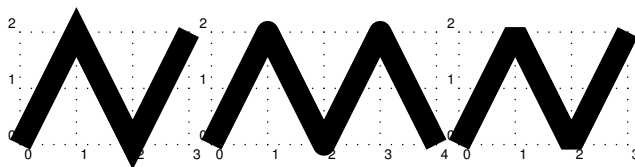


Figure 4: The meaning of the option `linejoin=0|1|2` for drawing lines

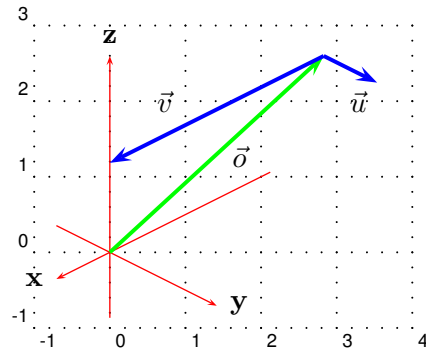
9 Squares

The syntax for a 3D square is:

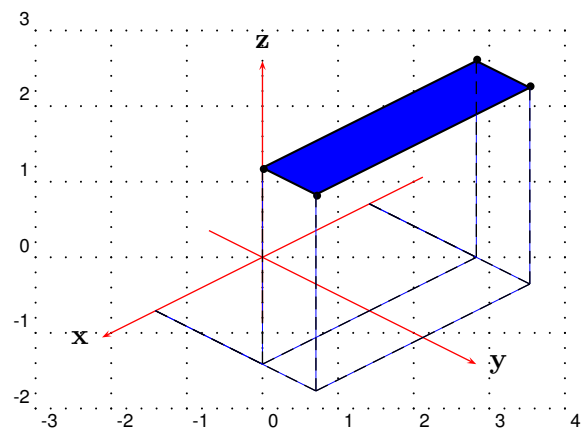
```

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

```



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.



```

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

```

10 Boxes

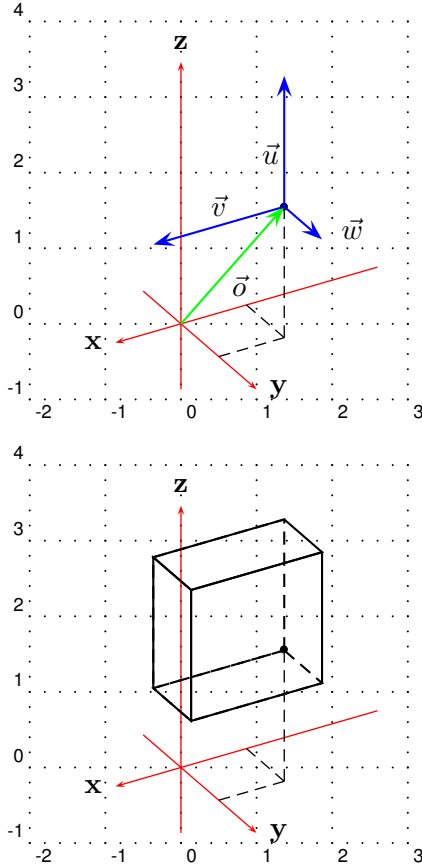
A box is a special case of a square and has the syntax

```

\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.



```

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

```

11 Ellipses and Circles

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

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

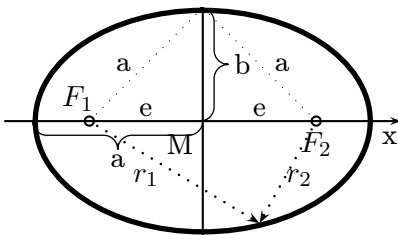


Figure 5: Definition of an Ellipse

$(x_m; y_m)$ is the center, a and b 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

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

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

$$e: \vec{x} = \vec{m} + \cos \alpha \cdot \vec{u} + \sin \alpha \cdot \vec{v} \quad 0 \leq \alpha \leq 360 \quad (6)$$

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

11.1 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 13.2 described `parametricplotThreeD` macro with a default setting of 50 points for a full ellipse/circle.

11.2 Ellipse

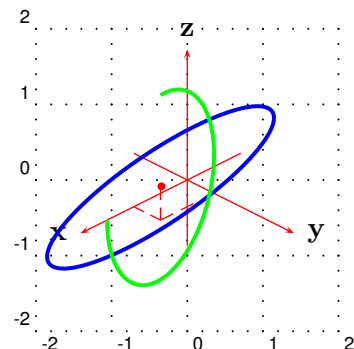
It is very difficult to see in a 3D coordinate system the difference of an ellipse and a circle. Depending to the view point an ellipse maybe seen as a circle and vice versa. The syntax of the ellipse macro is:

```

\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.



```

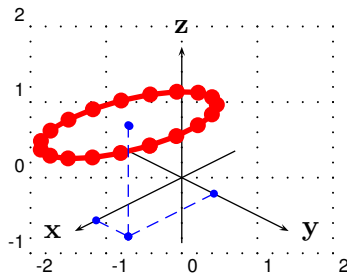
1 \psset{xMin=-1,xMax=2,yMin=-1,yMax=2,zMin=-1,
  zMax=2}
2 \begin{pspicture}(-2,-2.25)(2,2.25)
3 \psgrid
4 \pstThreeDCoor
5 \pstThreeDDot[%
6   linecolor=red,%
7   drawCoor=true](1,0.5,0.5)% the center
8 \psset{linecolor=blue,linewidth=1.5pt}
9 \pstThreeDEllipse(1,0.5,0.5)(-0.5,1,0.5)
10  (1,-0.5,-1)
11 % settings for an arc
12 \psset{beginAngle=0,endAngle=270,linecolor=
  green}
13 \pstThreeDEllipse(1,0.5,0.5)(-0.5,0.5,0.5)
  (0.5,0.5,-1)
14 \end{pspicture}

```

11.3 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`.



```

1 \begin{pspicture}(-2,-1.25)(2,2.25)
2 \psgrid
3 \pstThreeDCoor[%
4   xMin=-1,xMax=2,yMin=-1,yMax=2,zMin=-1,zMax
  =2,%
5   linecolor=black]
6 \psset{linecolor=red,linewidth=2pt,%
7   plotpoints=20,showpoints=true}
8 \pstThreeDCircle(1.6,+0.6,1.7)(0.8,0.4,0.8)
  (0.8,-0.8,-0.4)
9 \pstThreeDDot[drawCoor=true,linecolor=blue
  ](1.6,+0.6,1.7)
10 \end{pspicture}

```

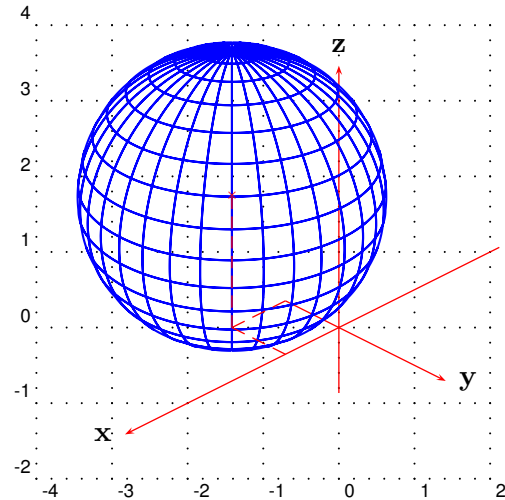
12 Spheres

To draw spheres `pst-3dplot` uses the macros from the `pst-vue3d` package and places it with internally the `\rput` macro at the right place.¹ The syntax for this macro is

¹This package is not on CTAN, but you can download it from Manuel Luques homepage <http://members.aol.com/Mluques5130/vue3d16112002.zip>. The

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

(x, y, z) is the center of the sphere. For all the other possible options or the possibility to draw demispheres, have a look at the documentation.[3]



```

1 \begin{pspicture}(-4,-2.25)(2,4.25)
2 \psgrid
3 \pstThreeDCoor[xMin=-3,xMax=4,yMin=-1,yMax
  =2,zMin=-1,zMax=4]
4 \pstThreeDSphere[linecolor=blue](1,-1,2){2}
5 \pstThreeDDot[dotstyle=x,linecolor=red,
  drawCoor=true](1,-1,2)
6 \end{pspicture}

```

13 Mathematical Functions

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

13.1 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 1).

documentation is in french, but it is mostly self explanatory

Table 1: 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.

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

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.

```
1 \begin{pspicture}(-6,-4)(6,5)
2 \psgrid
3 \psset{Alpha=45,Beta=15}
4 \psplotThreeD[%
5 plotstyle=line,%
6 drawStyle=xLines,% is the default anyway
7 yPlotpoints=50,xPlotpoints=50,%
8 linewidth=1pt](-4,4)(-4,4){%
9 x 3 exp x y 4 exp mul add x 5 div sub 10
10 mul
11 2.729 x dup mul y dup mul add neg
12 exp mul
13 2.729 x 1.225 sub dup mul y dup
14 mul add neg exp add}
15 \pstThreeDCoor[xMin=-1,xMax=5,yMin=-1,yMax=5,zMin=-1,zMax=5]
16 \end{pspicture}
```

13.2 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.

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

To draw a spiral we have the parametric functions:

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

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 $0^\circ \dots 2160^\circ$. 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}
```

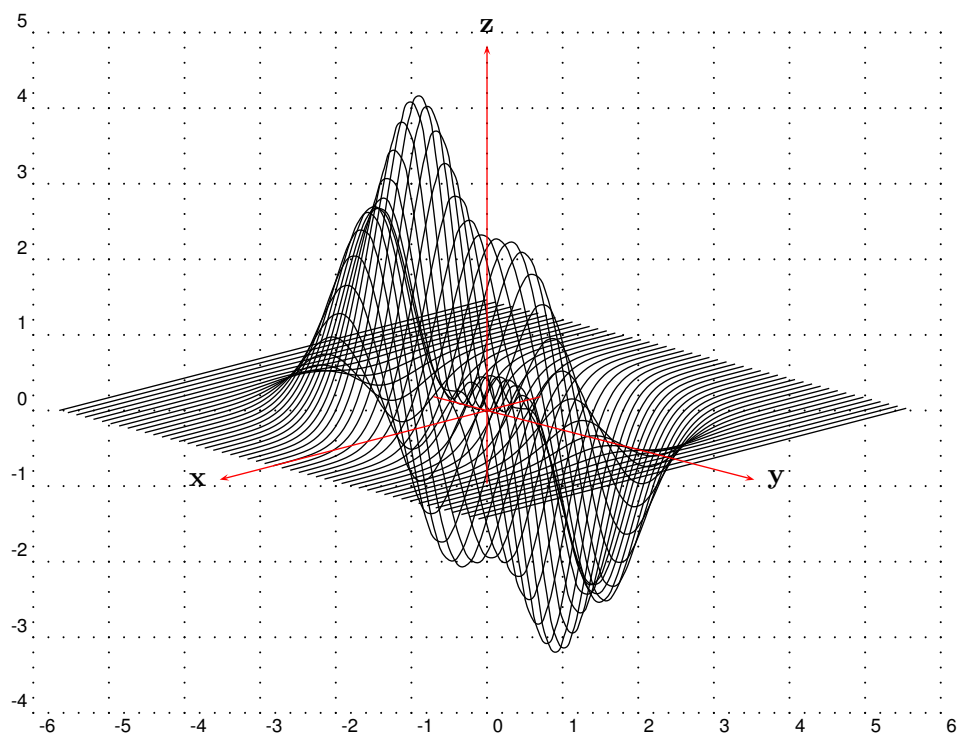
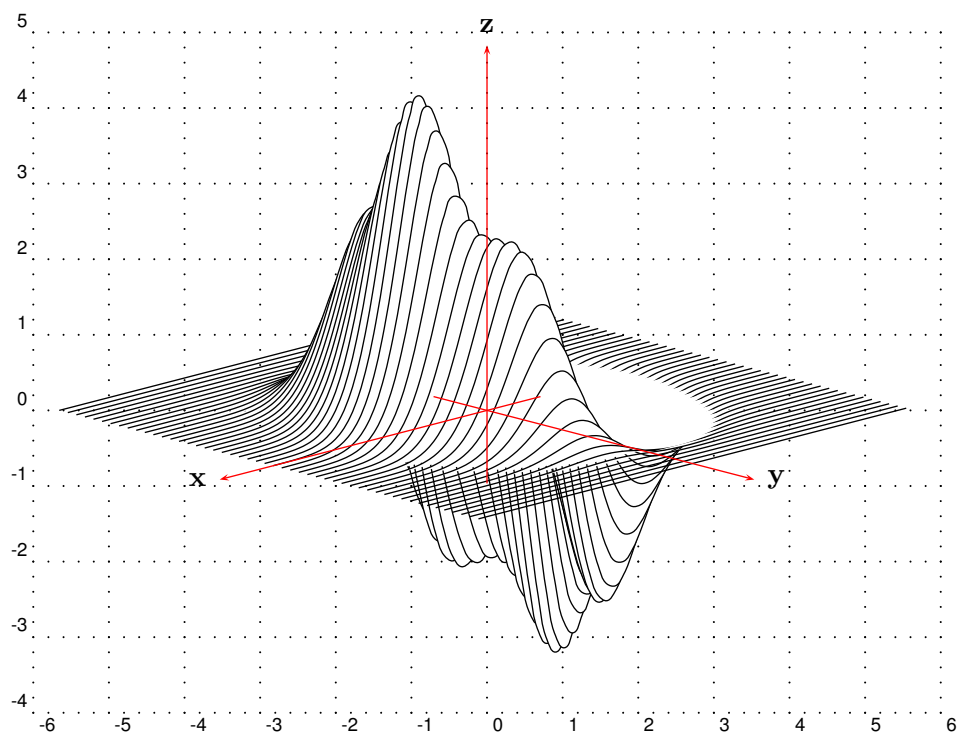


Figure 6: Plot of the equation 7

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

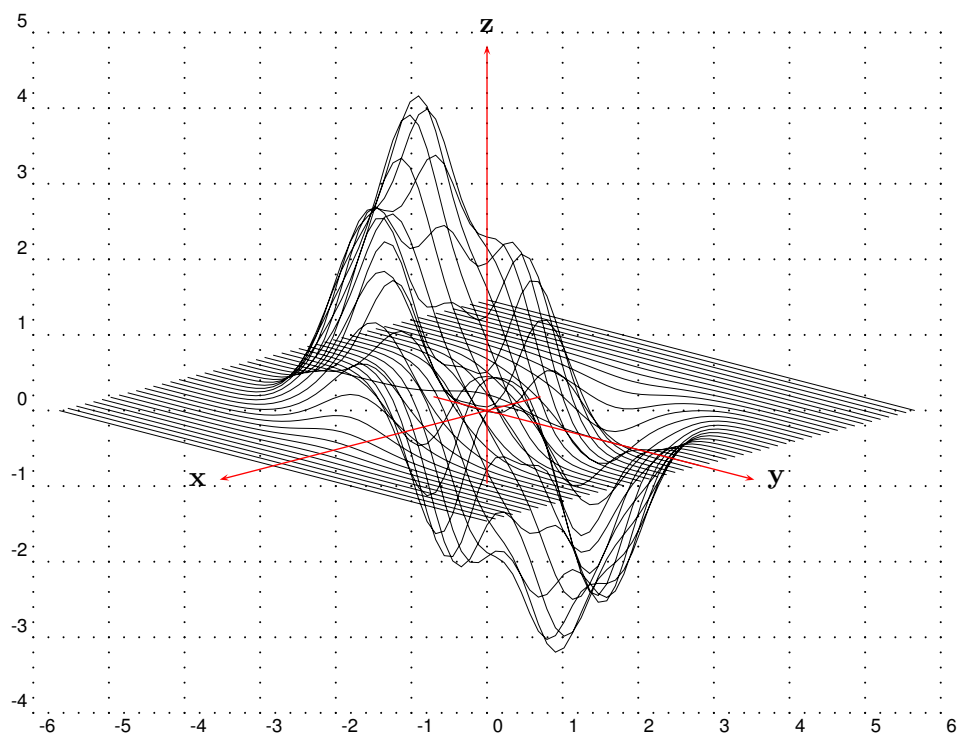


Figure 8: Plot of the equation 7 with the drawStyle=yLines option

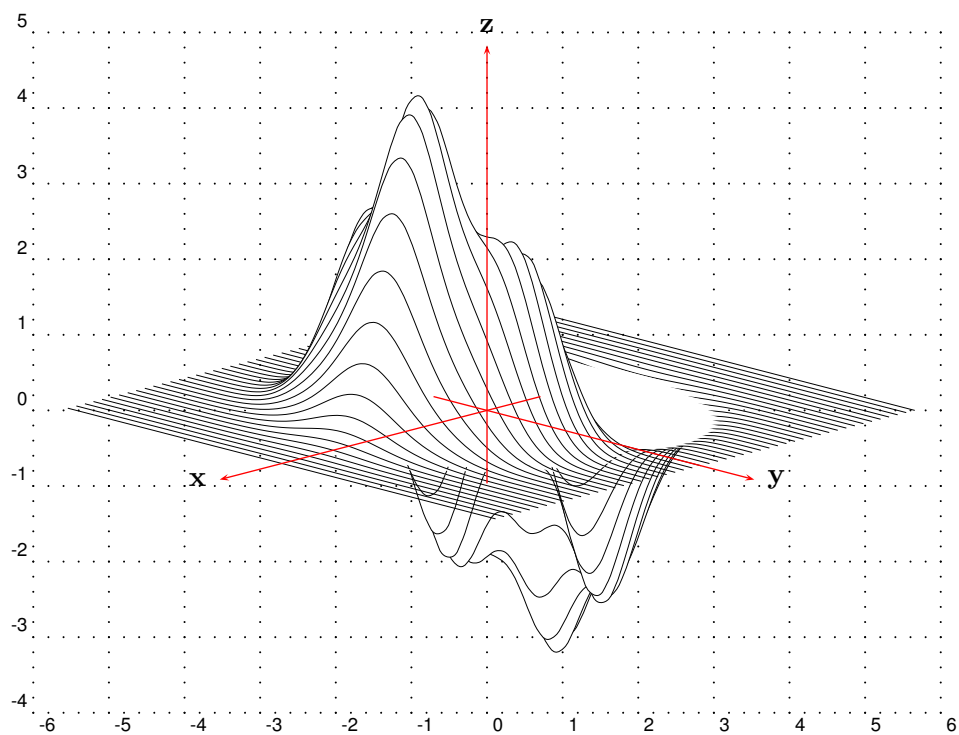
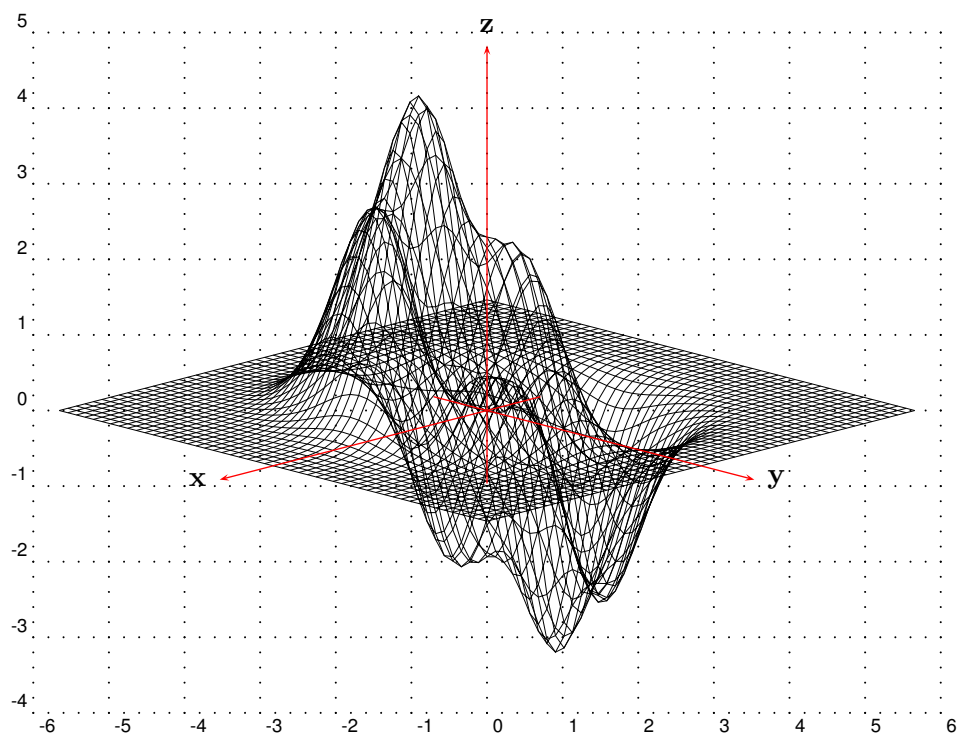
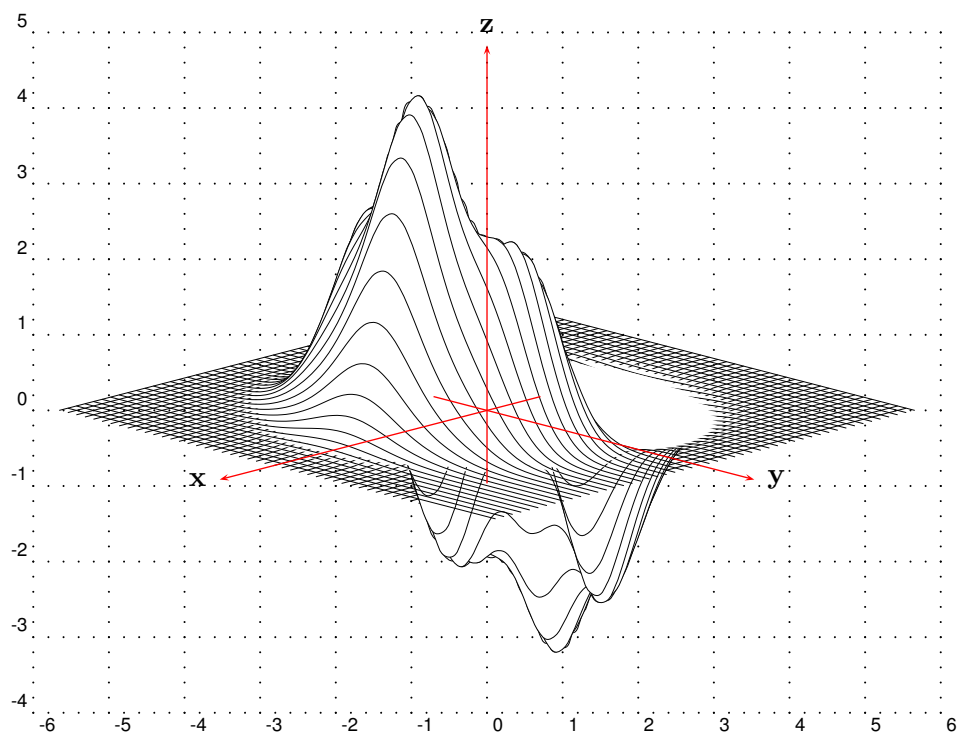


Figure 9: Plot of the equation 7 with the drawStyle=yLines and hiddenLine=true option

Figure 10: Plot of the equation 7 with the `drawStyle=xyLines` optionFigure 11: Plot of the equation 7 with the `drawStyle=xyLines` and `hiddenLine=true` option

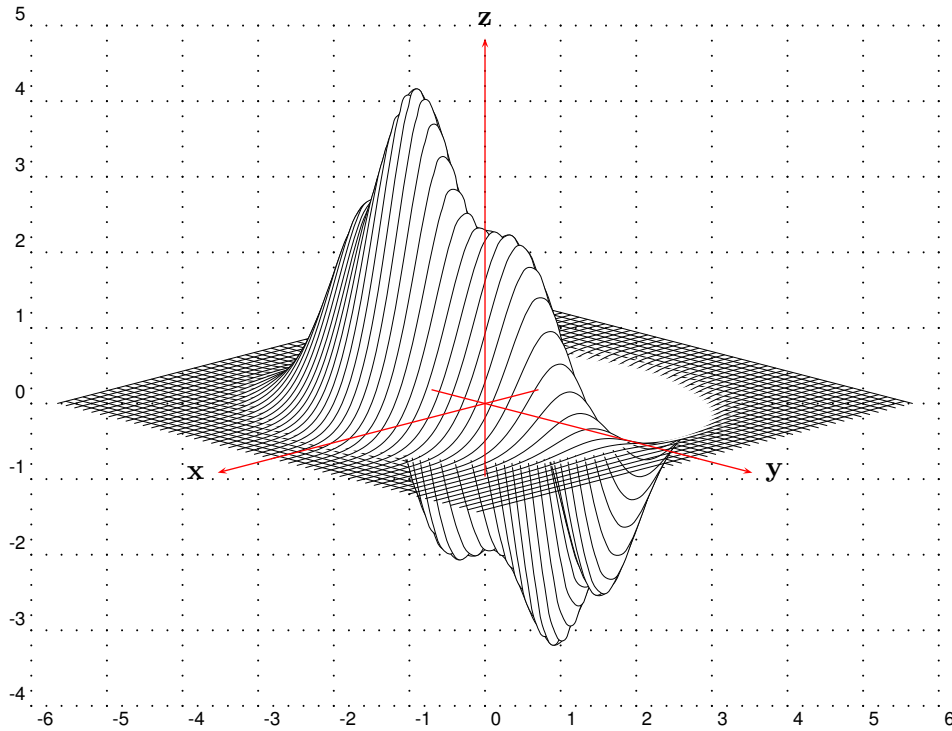
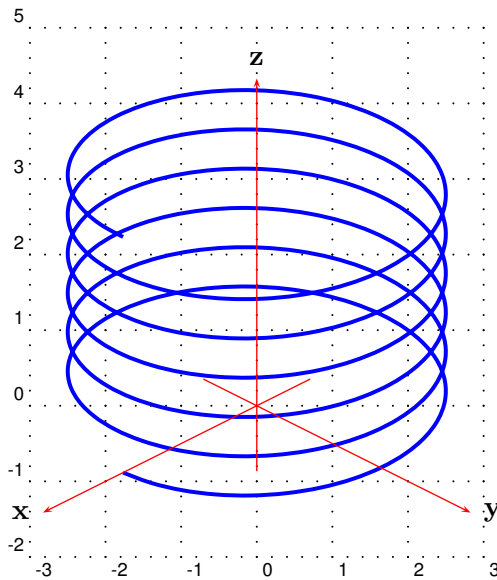


Figure 12: Plot of the equation 7 with the drawStyle=yxLines and hiddenLine=true option

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



```

1 \begin{pspicture} (-3.25,-2.25) (3.25,5.25)
2 \psgrid
3 \parametricplotThreeD[
4   xPlotpoints=200,%
5   linecolor=blue,%
6   linewidth=1.5pt,
7   plotstyle=curve](0,2160){
8   2.5 t cos mul
9   2.5 t sin mul
10  t 600 div%
11 }
12 \pstThreeDCoor[xMin=-1,xMax=4,yMin=-1,yMax
13   =4,zMin=-1,zMax=5]
14 \end{pspicture}

```

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

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

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

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

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

```

1 \begin{pspicture} (-1,-1) (1,1)
2 \psgrid
3 \parametricplotThreeD[plotstyle=curve,
4   yPlotpoints=40](0,360)(0,360){
5   t cos u sin mul
6   t cos u cos mul
7   t sin
8 }
9 \parametricplotThreeD[plotstyle=curve,
10  yPlotpoints=40](0,360)(0,360){
11  u cos t sin mul
12  u cos t cos mul
13  u sin

```



```

12 }
13 \end{pspicture}

```

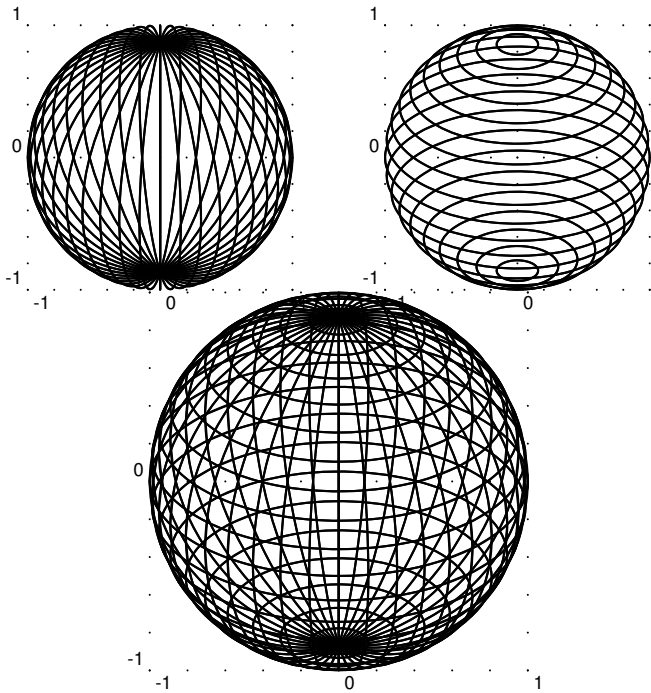


Figure 13: Different Views of the same Parametric Functions

14 Plotting Data Files

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

```

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 1.

14.1 \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.

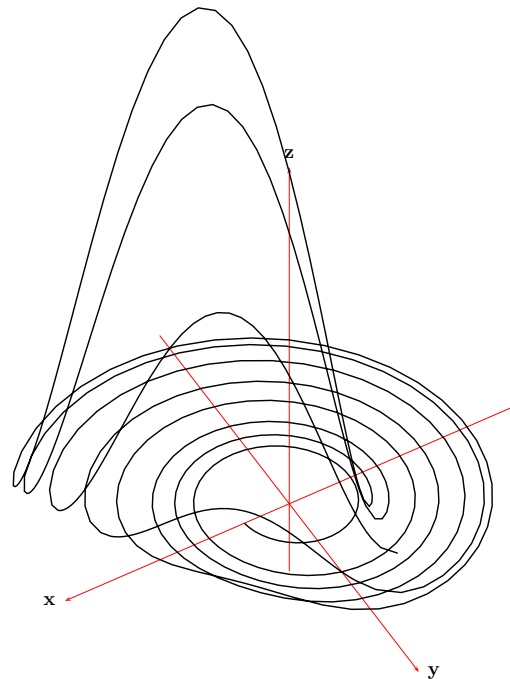


Figure 14: Demonstration of \fileplotThreeD with Alpha=30 and Beta=15

```

1 \begin{pspicture}(-7.5,-3)(6,10)
2   \psset{xunit=0.5cm,yunit=0.75cm,%
3     Alpha=30,Beta=30}% the global parameters
4   \pstThreeDCoord[%
5     xmin=-10,xmax=10,%
6     ymin=-10,ymax=10,%
7     zmin=-2,zmax=10%
8   ]
9   \fileplotThreeD[plotstyle=line]{data3
10     D.Roessler}
11 \end{pspicture}

```


14.2 \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>}`

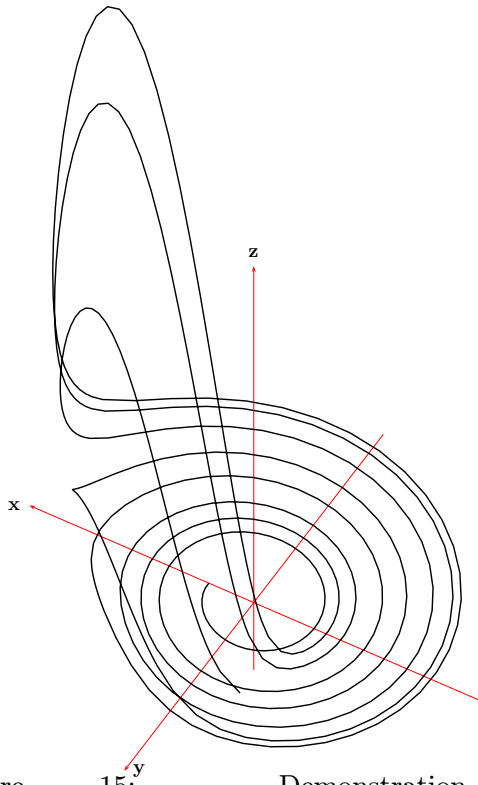


Figure 15: Demonstration of `\dataplotThreeD` with $\text{Alpha}=-30$ and $\text{Beta}=30$

```
1 \readdata{\dataThreeD}{data3D.Roessler}
2 [...]
3 \begin{pspicture}(-6,-2.5)(6,11)
4   \psset{xunit=0.5cm,yunit=0.75cm,%
5     Alpha=-30}
6   \pstThreeDCoor[%
7     xmin=-10,xmax=10,%
8     ymin=-10,ymax=10,%
9     zmin=-2,zmax=10%
10  ]
11     \dataplotThreeD[plotstyle=line]{\
12       dataThreeD}
13 \end{pspicture}
```

14.3 \listplotThreeD

The syntax is

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

`\listplotThreeD` is similar 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.:

```
1 \dataread{\data}{data3D.Roessler}
2 \newcommand{\dataThreeDDraft}{%
3   \data\space
4   gsave           % save graphic status
5   /Helvetica findfont 40 scalefont setfont
6   45 rotate       % rotate 45 degrees
7   0.9 setgray     % list white
8   -60 30 moveto (DRAFT) show
9   grestore
10 }
```

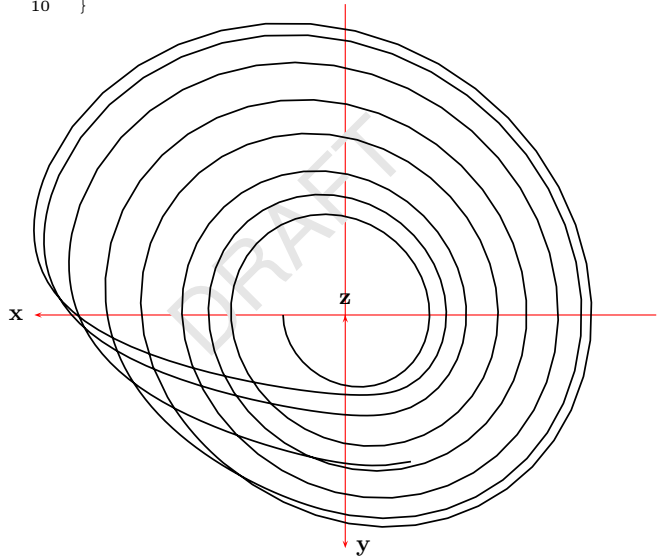


Figure 16: Demonstration of `\listplotThreeD` with a view from above ($\text{Alpha}=0$ and $\text{Beta}=90$) and some additional PostScript code

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`.

```
1 \begin{pspicture}(-5,-4)(5,4)
2   \psset{xunit=0.5cm,yunit=0.5cm,%
3     Alpha=0,Beta=90}
4   \pstThreeDCoor[%
5     xmin=-10,xmax=10,%
6     ymin=-10,ymax=7.5,%
7     zmin=-2,zmax=10%
8   ]
9   \listplotThreeD[plotstyle=line]{\
10     dataThreeDDraft}
11 \end{pspicture}
```

15 Utility macros

15.1 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, divided by a space. The syntax is:

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

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

15.2 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 $\vec{C} = \vec{A} + \vec{B}$.

15.3 Subtract 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 $\vec{C} = \vec{A} - \vec{B}$.

16 PDF Output

`pst-3dplot` is based on the popular `pstricks` package and writes pure PostScript code^[2], so it is not possible to run \TeX files with `pdf \LaTeX` 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`²
- build the PDF with `ps2pdf` (`dvi` \rightarrow `ps` \rightarrow `pdf`)

- use the `ps4pdf` package.³

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.

17 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}
```

References

- [1] Laura E. Jackson and Herbert Voß. Die Plot-Funktionen von `pst-plot`. *Die \TeX nische 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 \TeX nische Komödie*, 1/02:40–47, March 2002.
- [5] Herbert Voss. *PSTricks Support for pdf*. <http://www.educat.hu-berlin.de/~voss/lyx/pdf/pdftricks.phtml>, 2002.

²<http://www.micropress-inc.com/linux/>

³<http://www.perce.de/LaTeX/ps4pdf/>

18 Change Log

1.2b bugfix for `\def\pst@calcThreeDVec`

1.2a Disable loading of package `pstcol.sty`

1.2 Fixes small bugs.

Added new options for drawing 3d functions: `xLines` | `yLines` | `xyLines` | `yxLines`.

1.1 Fixes a bug in macro `\pstPlanePut`, added an undocumented key `origin`, which is not fully supported yet.

1.0 The macros `\pstThreeDEllipse` and `\pstThreeDBox` are now handled in the same way than the other ones: `\macroname[<options>] (...)`. An additional setting of the parameters with `\psset{...}` is no more needed.

This is important for `\pstThreeDBox`, which has now 5 parameters and do not need the setting with the `\rput` macro anymore:

old: `\pstThreeDPut(<origin>){\pstThreeDBox(<u>)(<v>)(<w>)}`

new: `\pstThreeDBox[<option>](<origin>)(<u>)(<v>)(<w>)`

0.9 Added macro `pstPlanePut`

19 Credits

Chris Kuklewicz (chrisk@MIT.EDU)