Three dimensional plots with `pst-3dplot`

Herbert Voß

Abstract

The well-known `pstricks` package [7] offers excellent macros for creating more or less complex graphics which could be inserted into the document without having it exported to EPS or PDF. `pstricks` itself is the base for several other additional packages, which are typically named `pst-xxxx`, such as `pst-3dplot`.

There exist several packages for plotting three dimensional graphical objects. `pst-3dplot` handles three dimensional objects, mathematical functions, and data files similarly to `pst-plot` in two dimensions.

1 Introduction

The `pstricks` packages are available as usual from any possible CTAN server. The base parts are located at `CTAN:graphics/pstricks/generic/` and most of the additional packages at `CTAN:graphics/pstricks/contrib/` [7].

All `\psgrid` commands are only for a better view of the examples, they are not really necessary for the 3D-plots. They are always used with the globally defined options

```
\psset{subgriddiv=0,griddots=5,%
       gridlabels=7pt}
```

2 The parallel projection

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

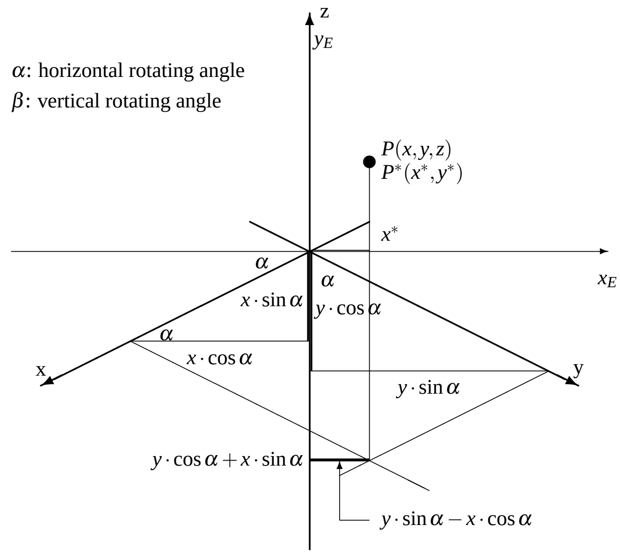

Figure 1: Lengths in a three dimensional system

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 perpendicularly 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.

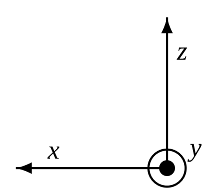

Figure 2: Coordinate system for $\alpha=\beta=0$ (y-axis comes out of the paper plane)

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

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

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 deg horizontally to the left and vertically by β also to the left.

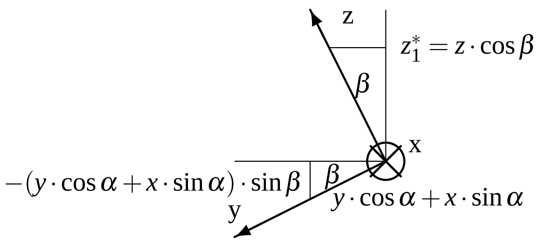

Figure 3: Coordinate system for $\alpha = 0$ and $\beta > 0$ (x-axis goes into the paper plane)

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 continously. 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 to multiply them 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 the same written in matrix form:

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

3 Coordinate axes

The syntax for drawing the coordinate axes is

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

Without any options, we get the default view seen in figure 4 with the predefined values:

```
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 (5) 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.

```
\begin{pspicture}(-2,-1)(1,2.25)
        \psgrid
```

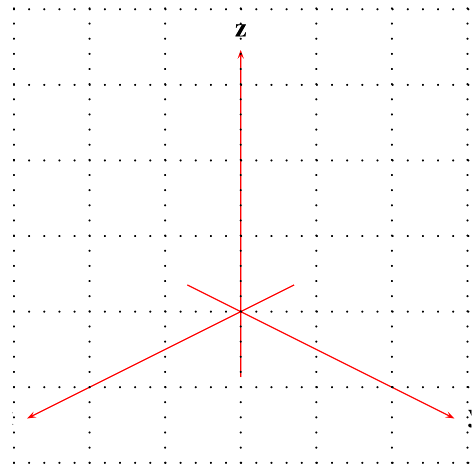

Figure 4: The default 3D coordinate system

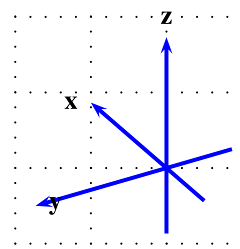

Figure 5: Axes with a different view and color

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

4 `put` command

The syntax is similar to the `\rput` macro from the package `pst-plot`:

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

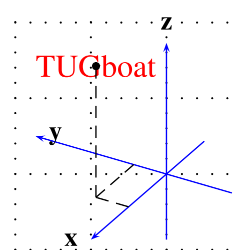

Figure 6: Example for the `\pstThreeDPut` macro

```
\begin{pspicture}(-2,-1)(1,2.25)
```

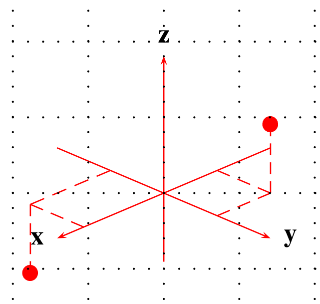

Figure 7: 3D dots with marked coordinates

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

Internally, the `\pstThreeDPut` macro defines a two dimensional node `temp@pstNode` and then uses the default `\rput` macro from `pstricks`. Because of the perspective from which the coordinate system is viewed, the 3D dot will not be seen as the center of the printed material when this is also a three dimensional one. This does not happen for figure 6, because the text is only a two dimensional object.

5 Nodes

The syntax is

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

This node is internally transformed into a two dimensional node, so it cannot be used as a replacement for the parameters `(x,y,z)` of the 3D dot which is possible with the macros from `pst-plot`. If `A` and `B` are two nodes, then `\psline{A}{B}` draws a line from `A` to `B`. Doing the same with `pst-3dplot` is not yet implemented. On the other hand, it is not a problem to define two 3D nodes `C` and `D` and then draw 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` (figure 7).

```
\begin{pspicture}(-2,-2)(2,2)
  \psset{xMin=-2,xMax=2,yMin=-2,%
         yMax=2,zMin=-1,zMax=2,Beta=25}
  \pstThreeDCoor
  \psset{dotstyle=*,dotscale=2,%
         linecolor=red,%
         drawCoor=true}
  \pstThreeDDot(-1,1,1)
  \pstThreeDDot(1.5,-1,-1)
  \psgrid
\end{pspicture}
```

In the figure 8 the coordinates of the dots are (a,a,a) where a is $-3,-2,-1,0,1,2,3$.

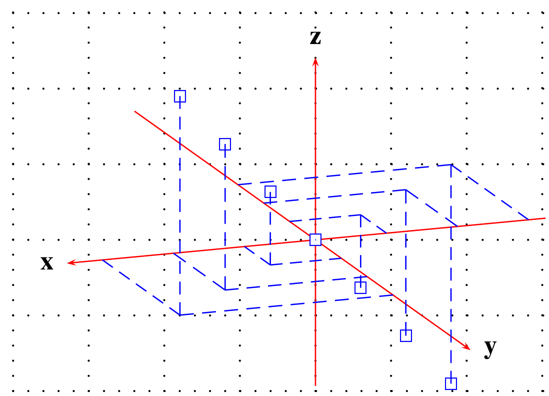

Figure 8: Another demonstration for drawing dots

```
\begin{pspicture}(-4,-2)(3,3.25)
  \psgrid
  \psset{xMin=-3.5,xMax=3.5,yMin=-7,yMax=6,zMin
      =-2,zMax=2.5,%
         Alpha=20,Beta=15}
  \pstThreeDCoor
  \psset{dotstyle=square,dotsize=5pt,%
         linecolor=blue,drawCoor=true}
  \multido{\n=-3+1}{7}{%
      \pstThreeDDot(\n,\n,\n)%
  }
\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. The only difference in drawing a line or a vector is that the first one has an arrow of type `-` and the second type `->` (figure 9).

```
\psset{xMin=-2,xMax=2,yMin=-2,yMax=2,%
       zMin=-2,zMax=2}
\begin{pspicture}(-2,-2.25)(2,2.25)
  \pstThreeDCoor
  \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)
```

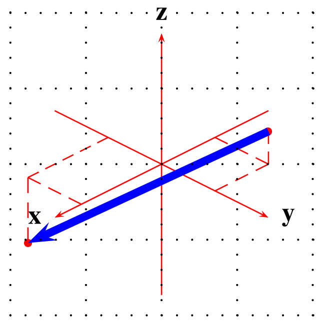

Figure 9: Drawing a 3D vector

```
    \psgrid
\end{pspicture}
```

8 Triangle

A triangle is given by its three points:

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

When the option `fillstyle` is set to value other than `none`, the triangle is filled with the active color or with the one which is set with the option `fillcolor` (figure 10).

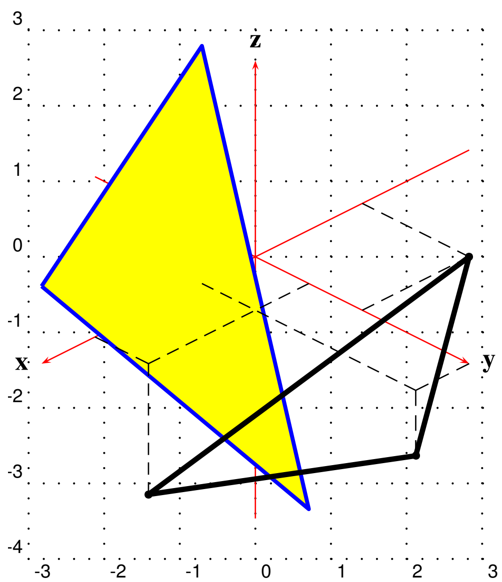

Figure 10: Triangles with fill option

```
\begin{pspicture}(-3,-4)(4,3.25)
  \psgrid
  \pstThreeDCoor[xMin=-4,xMax=5,yMin=-3,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)(-3,2,0)
\end{pspicture}
```

For triangles especially, the option `linejoin` is important. Its value is passed to the PostScript command `setlinejoin`. The default value is 1, which gives rounded edges (figure 11).

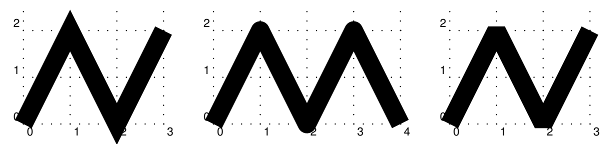

Figure 11: Meaning of the PostScript command `setlinejoin=0|1|2`

9 Squares

The syntax for a 3D square is:

```
\pstThreeDSquare%
      [<options>]
      (<vector o>)%
      (<vector u>)(<vector v>)
```

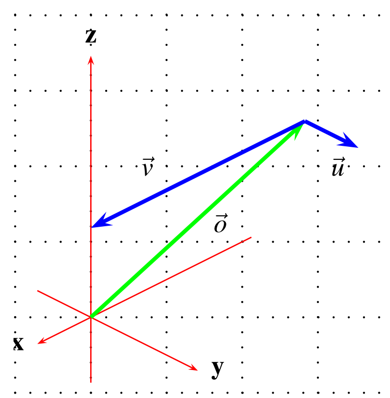

Figure 12: Drawing a square with three vectors

Squares are nothing more 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 as shown in figure 12. With the `fillstyle` option the square can be filled with the in `pst-plot` defined styles, for example `solid` like in figure 13. All the options of `pstricks` are allowed for this macro.

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

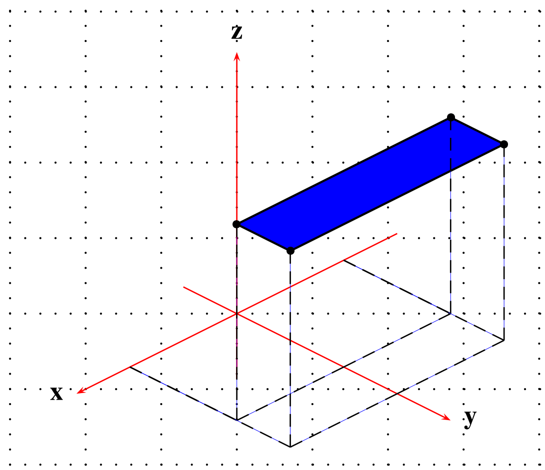

Figure 13: Drawing a filled square with the vectors from figure 12

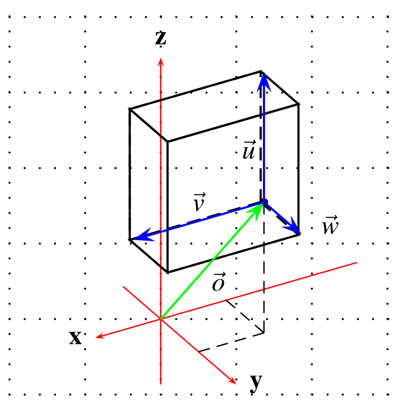

Figure 14: Drawing a box with three vectors

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>)
```

All options from `pstricks` are possible here. The other parameters are the origin vector $\vec{o}$ and the three direction vectors $\vec{u}$, $\vec{v}$ and $\vec{w}$. The figure 14 shows a box together with these four vectors. In this example the three direction vectors are perpendicular to each other.

```
\begin{pspicture}(-2,-1)(3,4.25)
   \psgrid
   \setkeys{psset}{Alpha=30,Beta=30}
   \pstThreeDCoor[xMin=-3,xMax=1,yMin=-1,yMax=2,
        zMin=-1,zMax=4]
   \pstThreeDPut(-1,1,2){\pstThreeDBox(0,0,2)
        (2,0,0)(0,1,0)}
   \pstThreeDDot[drawCoor=true](-1,1,2)
   \setkeys{psset}{arrows=->,arrowsize=0.2}
   \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=green](0,0,0)(-1,1,2)
   \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}
```

11 Ellipses and circles

The equation for a two dimensional ellipse (figure 15) is:

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

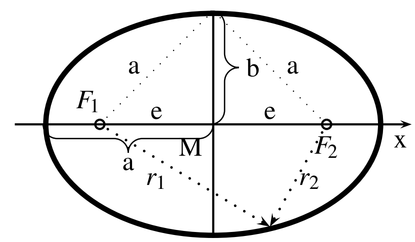

Figure 15: Definition of an ellipse

$(x_m; y_m)$ is the center, a and b the eccentricity. For $a = b = 1$ in equation 4 we get the "one" for the circle, which is nothing more than a special case of an ellipse. The equation written in parametric form 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:

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

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

11.1 Options

In addition to all possible options from the package `pst-plot`, we have two special ones for the drawing of an arc (with predefined values for a full ellipse or circle):

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

Using the `parametricplotThreeD` macro (described in section 13.2, ellipses and circles are drawn with a default setting of 50 points for the ellipse or circle.

11.2 Ellipse

In a 3D coordinate system, it is very difficult to see the difference between an ellipse and a circle. Depending on the point of view an ellipse may be seen as a circle and vice versa (figure 16). The syntax of the ellipse macro is:

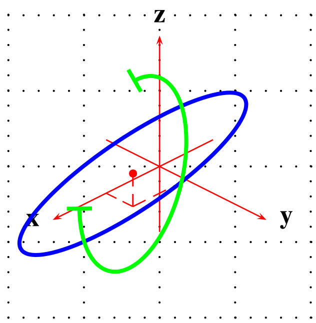

Figure 16: Drawing ellipses

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

where `c` is for center and `u` and `v` for the two direction vectors (eq. 6).

```
\psset{xMin=-1,xMax=2,yMin=-1,yMax=2,zMin=-1,zMax
    =2}
\begin{pspicture}(-2,-2)(2,2)
  \psgrid
  \pstThreeDCoor
  \pstThreeDDot[%
      linecolor=red,%
      drawCoor=true](1,0.5,0.5)% the center
  \pstThreeDEllipse[%
      linecolor=blue, linewidth=1.5pt]%
      (1,0.5,0.5)(-0.5,1,0.5)(1,-0.5,-1)
  % settings for an arc
  \pstThreeDEllipse[%
       beginAngle=0,endAngle=270,%
       linecolor=green]%
       (1,0.5,0.5)(-0.5,0.5,0.5)(0.5,0.5,-1)
\end{pspicture}
```

11.3 Circle

The circle is a special case of an ellipse (eq. 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 more 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)(2,2)
  \psgrid
  \pstThreeDCoor[%
      xMin=-1,xMax=2,yMin=-1,yMax=2,zMin=-1,zMax
          =2,%
      linecolor=black]
   \pstThreeDCircle[%
      linecolor=red,linewidth=2pt,%
      plotpoints=20,showpoints=true]%
     (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)
\end{pspicture}
```

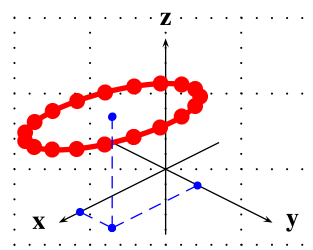

Figure 17: Drawing a circle with the option `showpoints`

12 Spheres

Internally, `pst-3dplot` uses the macro from the `pst-vue3d` package[1] to draw spheres, and places it with the `\rput` macro at the right place. The syntax for this macro is

```
\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, refer to the documentation.[3]

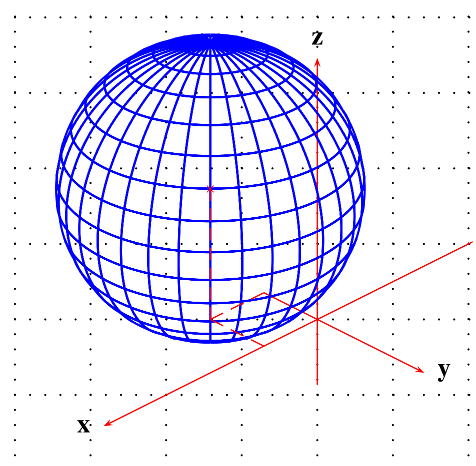

Figure 18: Drawing a sphere with package `pst-vue3d`

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

[1] CTAN:graphics/pstricks/contrib/pst-vue3d, and from Manuel Luque's homepage[3]. The documentation is in French, but it is mostly self-explanatory.

13 Mathematical functions

There exist two macros for plotting mathematical functions $f(x,y)$, which work similarly 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`[5], 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`. For example, `{x dup mul y dup mul add sqrt}` represents the math expression $\sqrt{x^2+y^2}$. The macro `\psplotThreeD` has the same plotstyle options as `\psplot`, except the `plotpoints`-option which is split into one for `x` and one for `y` (table 1).

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
`hiddenLine`	default is false

Equation 7 is plotted with the following parameters and seen in figure 19.

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

```
\begin{pspicture}(-6,-4)(6,5)
   \psgrid
   \psset{Alpha=45,Beta=15}
   \psplotThreeD[%
       plotstyle=line,%
       yPlotpoints=40,xPlotpoints=30,%
       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}
```

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);
```

Because of the inner loop it is only possible to get a closed curve in x direction. Therefore fewer `yPlotpoints` are not a real problem, but too few `xPlotpoints` results in a bad drawing of the mathematical function, especially for the plotstyle option `line`.

Drawing three dimensional mathematical 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 well when the y-interval is defined such that $y_2 > y_1$. Then, every new curve is plotted over the previous one and filled with the color white. Figure 20 is the same as figure 19, only with the option `hiddenLine=true`.

13.2 Parametric plots

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

```
\parametricplotThreeD[<options>]%
    (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 is by default `t`. In this case we do not need all parameters, so that we can write

```
\parametricplotThreeD[<options>]%
    (t1,t2)%
    {<three parametric functions x y z}
```

which is the same as `(0,0)` for the parameter `u`. Figure 21 shows a three dimensional curve.

```
\begin{pspicture}(-3,-2)(3,5)
   \psgrid
   \parametricplotThreeD[%
       xPlotpoints=200,%
       linecolor=blue,%
```

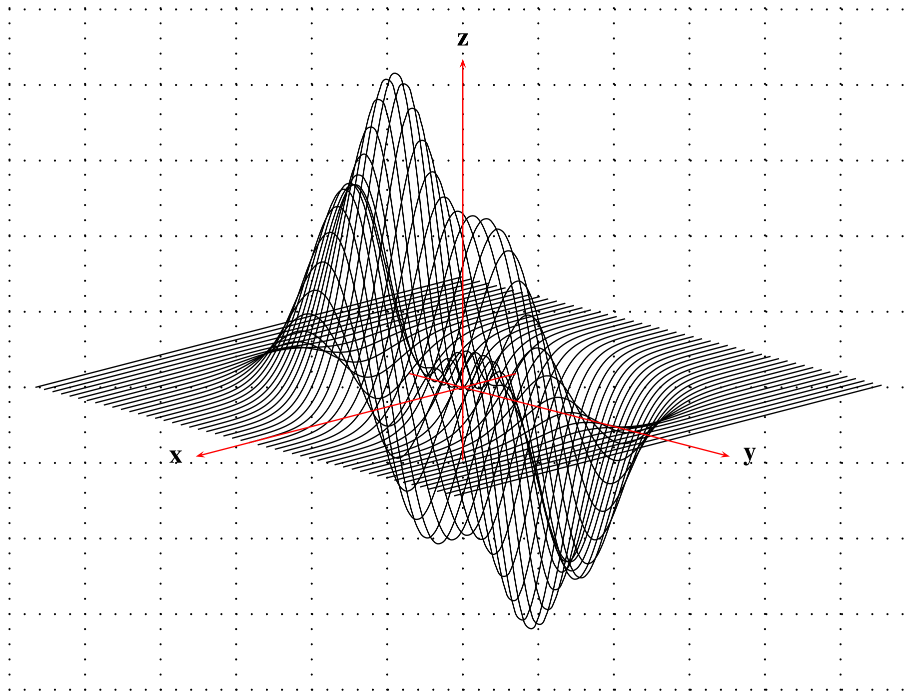

Figure 19: Plot of equation 7

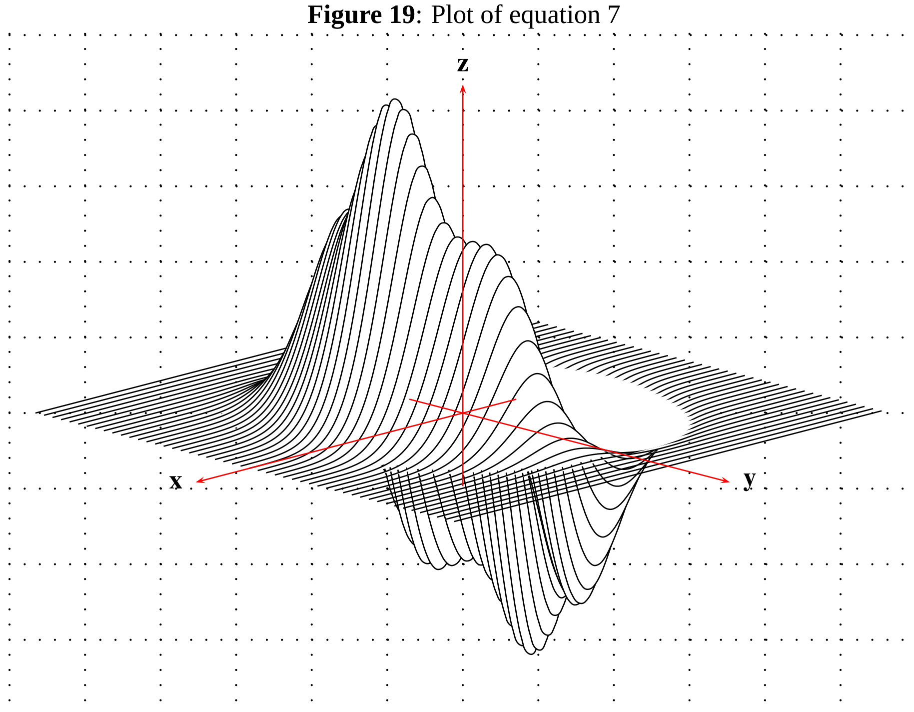

Figure 20: Plot of equation 7 with the `hiddenLine=true` option

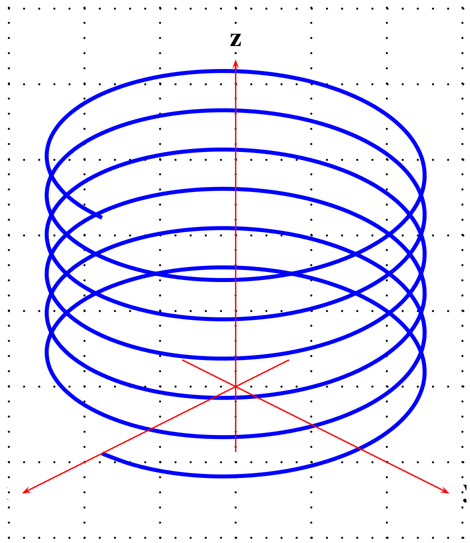

Figure 21: Drawing a 3D curve

```
      linewidth=1.5pt,
      plotstyle=curve](0,2160){%
          2.5 t cos mul
          2.5 t sin mul
          t 600 div%
  }
  \pstThreeDCoor[xMin=-1,xMax=4,yMin=-1,yMax=4,
       zMin=-1,zMax=5]
\end{pspicture}
```

Instead of using the `\pstThreeDSphere` macro (see section 12) 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 needs the parametric 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 lastly, we can have both of these parametric functions together in one `pspicture` environment (figure 22).

```
\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}
```

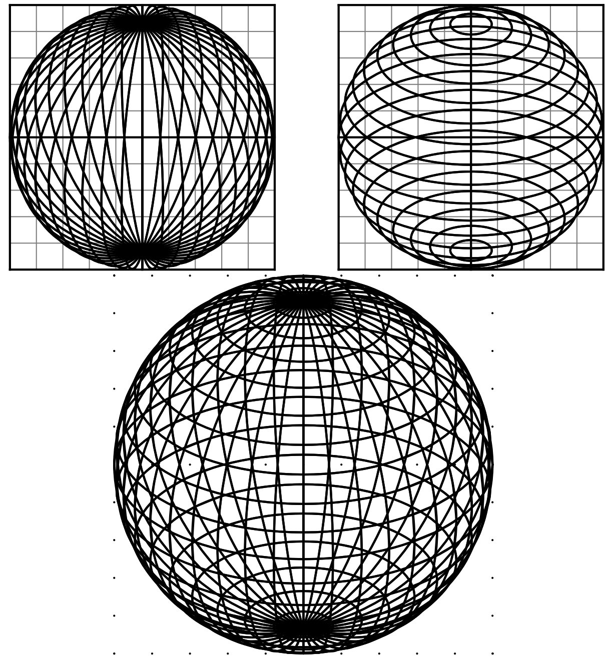

Figure 22: Different views of the same parametric functions

14 Plotting data files

We have the same conventions for data files which hold 3D coordinates as for 2D. 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 data file used in the following examples has 446 entries like

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

Using the `listplotThreeD` macro with many data entries may take considerable time on slow machines. The possible options for the lines are the same as earlier, given in table 1.

14.1 `\fileplotThreeD`

The syntax is straightforward:

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

If the data file is not in the same directory as the document, use the file name with the full path. Figure 23 shows a file plot with the option `linestyle=line`.

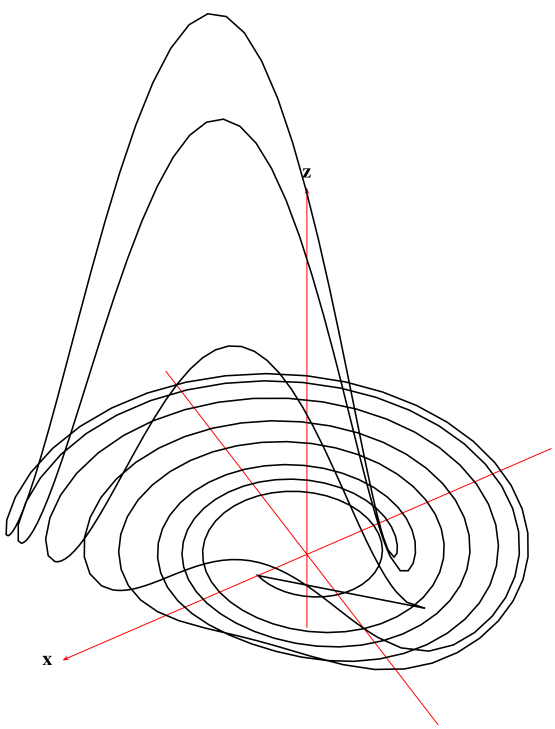

Figure 23: Demonstration of `\fileplotThreeD` with `Alpha=30` and `Beta=15`

```
\begin{pspicture}(-7.5,-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=polygon]{data3D.
        Roessler}
\end{pspicture}
```

14.2 `\dataplotThreeD`

The syntax is:

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

In contrast to `\fileplotThreeD`, the second macro `\dataplotThreeD` reads the data entries from another macro. Using `\readdata`, external data can be read from a file and saved in a macro, to be passed to `\dataThreeD` [1].

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

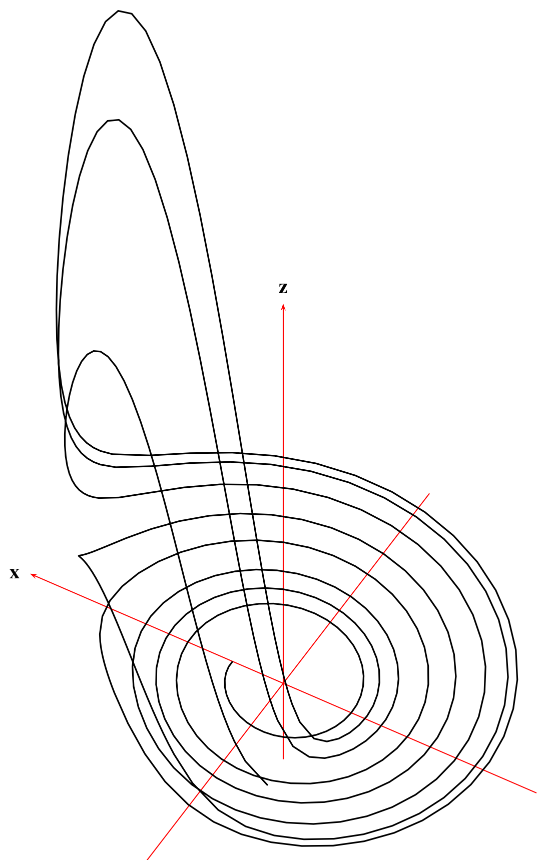

Figure 24: Demonstration of `\dataplotThreeD` with `Alpha=-30` and `Beta=30`

```
  \readdata{\dataThreeD}{data3D.Roessler} [...]
\begin{pspicture}(-6,-2.25)(6,11)
   \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}
```

14.3 `\listplotThreeD`

The syntax is:

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

There is no essential difference between the macros `\istplotThreeD` and `\dataplotThreeD`. With `\listplotThreeD`, one can pass additional PostScript code, which is appended to the data object. For example:

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

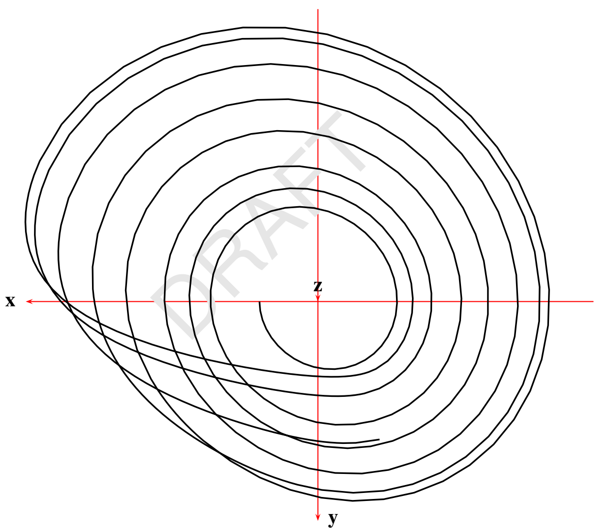

Figure 25: Demonstration of `\listplotThreeD` with a view from above (`Alpha=0` and `Beta=90`) and some additional PostScript code

Figure 25 shows what happens with this additional PostScript code. Another example can be found in [5], where `ScalePoints` is redefined. For `pst-3dplot`, the equivalent macro is named `ScalePointsThreeD`.

```
\begin{pspicture}(-5,-4)(5,4.5)
   \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}
```

15 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 pdfLaTeX when there are pstricks macros in the document. If you need PDF output, there are the following possibilities:

- the package `pdftricks.sty` [6]
- the free (for Linux only) program VTeX/Lnx (`http://www.micropress-inc.com/linux/`
- the `ps2pdf` (`dvi`→`ps`→`pdf`) or `dvipdfm` utilities
- the `ps4pdf` package [4].

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

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] Rolf Niepraschk. *ps4pdf*. `CTAN:/macros/latex/contrib/ps4pdf/`, 2003.

[5] Herbert Voß. Die mathematischen Funktionen von PostScript. *Die TeXnische Komödie*, 1/02:40–47, March 2002.

[6] Herbert Voß. *PSTricks Support for pdf*. `http://www.educat.hu-berlin.de/~voss/lyx/pdf/pdftricks.phtml`, 2002.

[7] Timothy van Zandt. *PSTricks - PostScript macros for Generic TeX*. `http://www.tug.org/application/PSTricks`, 1993.

◇ Herbert Voß
Wasgenstr. 21
14129 Berlin GERMANY
`voss@perce.de`
`http://www.perce.de`