The 'pst-3d' package
Tilting and other pseudo-3D tricks with PSTricks

Timothy Van Zandt
Herbert Voß

Version 1.00 2005/09/03
Documentation revised September 6, 2005 (hv)

Abstract

`pst-3d` provides basic macros for shadows, tilting and three dimensional representations of text or graphical objects.

Contents

1 introduction

The base package `pstricks` already disposes of some macros with which three dimensional effects can be obtained. There are several packages though which support the creation of three dimensional objects or functions. A compilation is shown in table 1. Here already several of the packages overlap, for parallel developments are nothing unusual in the TeX world. Although `pst-3d` is one of the older packages, it shall be dealt with nevertheless, for it also contains the preliminary stage of the 3D representations, that is shadow creation and tilting.

Table 1: Summary of all 3D packages

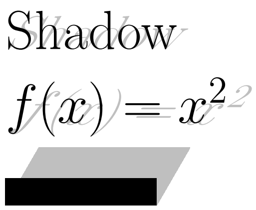
package	*content*
`pst-3d`	basic 3D operations
`pst-3dplot`	Three dimensional plots
`pst-fr3d`	Three dimensional framed Boxes
`pst-gr3d`	3D grids
`pst-map3dII`	3D Geographical Projection
`pst-ob3d`	Three dimensional basic objects
`pst-vue3d`	Three dimensional views

2 Shadow

`pst-3d` defines the macro `\psshadow` with the following syntax:

```
\psshadow[<parameters>]{<material>]
```

As parameters the ones given in table 2 are available next to all previously defined, if they have a meaning for the material to be shadowed. This can be anything text-like, text, rules and mathematical expressions in inline mode.




```
\newgray{gray75}{.75}
\psset{Tshadowcolor=gray75}
\psshadow{\huge Shadow}\\[10pt]
\psshadow{\huge $f(x)=x^2$}\\[15pt]
\psshadow[Tshadowsize=2.5]{%
  \rule{2cm}{10pt}}
```

2.1 Parameters

Table 2 shows a compilation of the used parameters.

Table 2: Summary of all `shadow` parameters

name	*values*	*default*
`Tshadowangle`	`<angle>`	`60`
`Tshadowcolor`	`<colour>`	`lightgray`
`Tshadowsize`	`<value>`	`1`

2.1.1 `Tshadowangle`

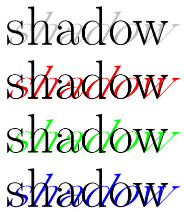
`Tshadowangle` denotes the angle of the shadow, referring to the perpendicular of the paper plane. The angle of 90ř therewith corresponds to the text itself. Negative angles cause the shadow to arise from the paper plane.


```
\newgray{gray75}{.75}
\psset{Tshadowcolor=gray75}
\psshadow{\huge shadow}\\[5pt]
\psshadow[Tshadowangle=30]{\huge shadow}\\[5pt]
\psshadow[Tshadowangle=70]{\huge shadow}\\[5pt]
\psshadow[Tshadowangle=-30]{\huge shadow}
```

- Angular values of 0ř and 180ř are not allowed.

2.1.2 `Tshadowcolor`

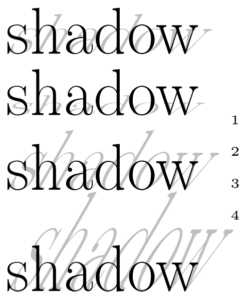
`Tshadowcolor` deontes the shadow colour.




```
\psshadow{\huge shadow}\\[5pt]
\psshadow[Tshadowcolor=red]{\huge shadow}\\[5pt]
\psshadow[Tshadowcolor=green]{\huge shadow}\\[5pt]
\psshadow[Tshadowcolor=blue]{\huge shadow}
```

2.1.3 `Tshadowsize`

`Tshadowsize` determines the size of the shadow as a scaling factor.




```
\psshadow{\Huge shadow}\\[5pt]
\psshadow[Tshadowsize=0.5]{\Huge shadow}\\[10pt]
\psshadow[Tshadowsize=1.5]{\Huge shadow}\\[20pt]
\psshadow[Tshadowsize=2.5]{\Huge shadow}
```

3 Tilting

With the tilting of objects the perspective views of three dimensional objects can be simulated. `pst-3d` defines two macros for this.

```
\pstilt[<parameters>]{<angle>}{<material>}
\psTilt[<parameters>]{<angle>}{<material>}
```

Figure 1 shows the difference between these two macros. Principally everything can be given as argument to those macros and therewith tilted. With vertical material, as distinguished formulae, eventually the argument has to be put into a `\parbox` before (see example),

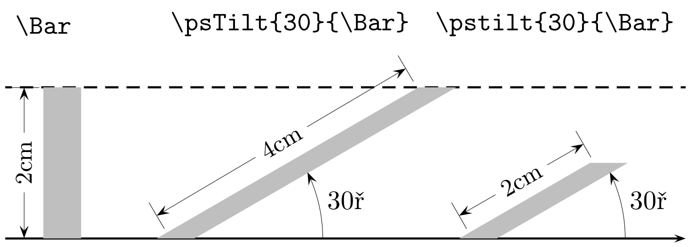
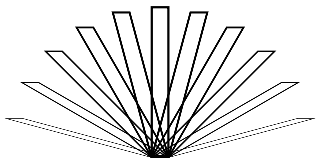


Figure 1: Demonstration of the difference between `\pstilt` and `\psTilt`

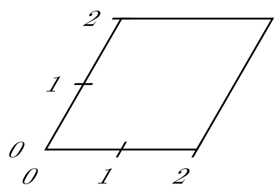
- Angular values of 0° and 180° are not allowed.

3.1 `\pstilt`

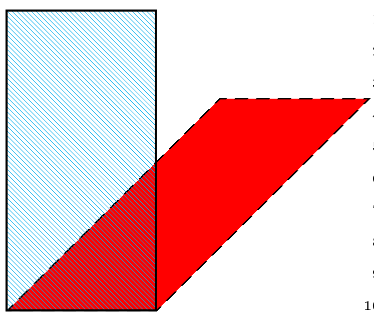
`\pstilt` tilts objects that their original height appears as new length of the tilted object, wherewith the object becomes smaller. The hynotenuse of the triangle from nadir, height and perpendicular now corresponds to the old height (see figure 1). At this the length is calculated from the middle of the base side.



```
\def\Bar{\psframe(0,0)(0.25,2)}
\begin{pspicture}(5,2)
  \multido{\nA=15+15}{11}{\rput(2.5,0){%
    \pstilt{\nA}{\Bar}}}
\end{pspicture}
```

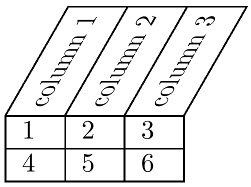



```
\pstilt{60}{%
  \begin{pspicture}(-0.5,-0.5)(2,2)
    \psaxes[axesstyle=frame](2,2)
  \end{pspicture}}
```



```
\newpsstyle{TCyan}{%
  fillstyle=vlines,hatchcolor=cyan,
  hatchwidth=0.1\pslinewidth,%
  hatchsep=1.5\pslinewidth}
\begin{pspicture}(2,4)
  \rput[lb](0,0){\pstilt{45}{%
    \psframe[linestyle=dashed,%
      fillstyle=solid,fillcolor=red](2,4)}}
  \psframe[style=TCyan](0,0)(2,4)
\end{pspicture}
```

With the package `rotating` macros to rotate text are provided, to achieve slant table headings for example. It is more difficult when they are provided with a frame. With `\pstilt` or `\psTilt` this is no problem. The program listing given below only shows the application of `\pstilt` for the macro only has to be replaced by `\psTilt` to obtain the other example.

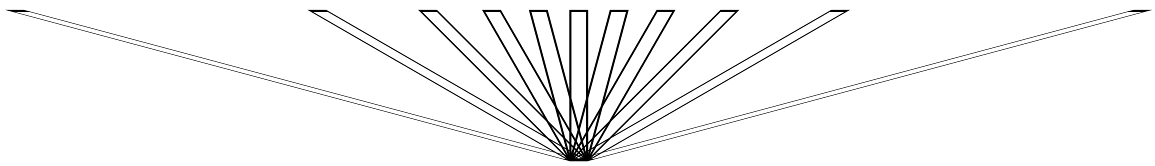


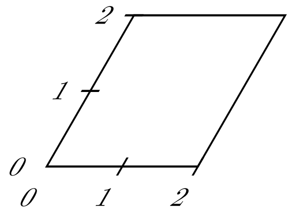

```
\begin{tabular}{l}
\pstilt{60}{%
\begin{tabular}{|p{1em}|p{1em}|p{1em}|}\hline
\psrotateleft{column 1\ }
  &  \psrotateleft{column 2\ }
  &  \psrotateleft{column 3\ }
\end{tabular}}\\
\begin{tabular}{|p{1em}|p{1em}|p{1em}|}\hline
 1 & 2 & 3 \\\hline
 4 & 5 & 6 \\\hline
\end{tabular}
\end{tabular}
```

3.2 \psTilt

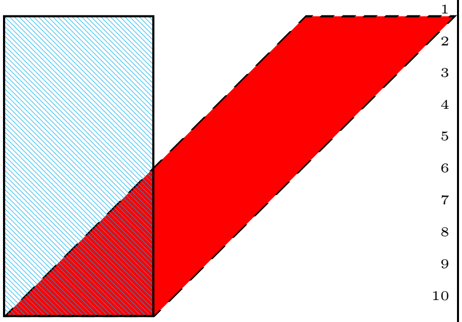
`\psTilt` tilts objects that their original height is preserved, so that the object could become infinitely long in theory (see figure 1).

```
\begin{pspicture}(5,2)
\def\Bar{\psframe(0,0)(0.25,2)}
\multido{\nA=15+15}{11}{\rput(2.5,0){%
  \psTilt{\nA}{\Bar}}}
\end{pspicture}
```






```
\psTilt{60}{%
  \begin{pspicture}(-0.5,-0.5)(2,2)
    \psaxes[axesstyle=frame](2,2)
  \end{pspicture}}
```



```
\newpsstyle{TCyan}{%
 fillstyle=vlines,hatchcolor=cyan,
 hatchwidth=0.1\pslinewidth,%
 hatchsep=1.5\pslinewidth}
\begin{pspicture}(2,4)
 \rput[lb](0,0){\psTilt{45}{%
  \psframe[linestyle=dashed,%
   fillstyle=solid,%
   fillcolor=red](2,4)}}
 \psframe[style=TCyan](0,0)(2,4)
\end{pspicture}
```

4 Three dimensional representations

`pst-3d` only supports parallel projections, so that geometrical objects such as spheres or cylinders can only be displayed restricted. Although `pst-3d` principally only defines one single macro for the 3D projection, the package is very efficient in its application and is also used as a base for other packages.[?][?]

4.1 `\ThreeDput`

`pst-3d` only defines this single macro, which can be used to arbitrarily display line or area shaped objects in the three dimensional space in the end though.

```
\ThreeDput[<parameters>]{<material>}
\ThreeDput[<parameters>](<x,y,z>){<material>}
```

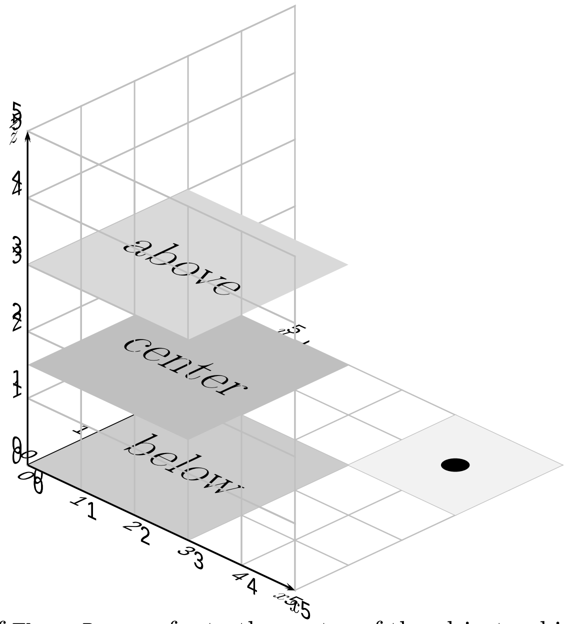
Without a specification of coordinates, $(0, 0, 0)$ is taken as origin of ordinates as a rule. As "material" anything is understood that can be put into a box. If it is vertical material in the TEX sense, it has to be put in a `\parbox` or `minipage` before.

To simplify the specified source code, the macro `\IIIDKOSystem` is used in the following, which draws the coordinate axes with the grid and is not specified in the following anymore.

newgraygray850.85

```
\makeatletter
\def\xyPlain#1{%
 \ThreeDput[normal=0 0 1](0,0,0){% xy-plane
  \psgrid[subgriddiv=0,gridcolor=lightgray](0,0)(#1,#1)
  \psline{->}(0,0)(0,#1) \psline{->}(0,0)(#1,0)
  \ifdim\psk@gridlabels pt>\z@
```

```
     \uput[180]{0.2}(0,#1){$y$}\uput[-90]{0.2}(#1,0){$x$}\fi }}
\def\xzPlain#1{%
 \ThreeDput[normal=0 -1 0](0,0,0){% xz-plane
  \psgrid[subgriddiv=0,gridcolor=lightgray](0,0)(#1,#1)
  \psline{->}(0,0)(0,5) \psline{->}(0,0)(#1,0)
  \ifdim\psk@gridlabels pt>\z@
    \uput[180]{0.2}(0,#1){$z$}\uput[-90]{0.2}(#1,0){$x$}%
  \fi }}
\def\yzPlain#1{%
 \ThreeDput[normal=1 0 0](0,0,0){% yz-plane
  \psgrid[subgriddiv=0,gridcolor=lightgray](0,0)(#1,#1)
  \psline{->}(0,0)(0,#1) \psline{->}(0,0)(#1,0)
  \ifdim\psk@gridlabels pt>\z@
    \uput[180]{0.2}(0,#1){$z$}\uput[-90]{0.2}(#1,0){$y$}%
  \fi }}
\def\IIIDKOSystem{\@ifnextchar[{\IIIDKOSystem@i}{\IIIDKOSystem@i[]}}
\def\IIIDKOSystem@i[#1]#2{%
 \psset{#1}%
 \xyPlain{#2}\xzPlain{#2}\yzPlain{#2}}
\makeatother
\newgray{gray75}{0.75}
\newgray{gray80}{0.8}
\newgray{gray85}{0.85}
\newgray{gray95}{0.95}
\begin{pspicture}(0,-1.25)(5,6)
 \psset{viewpoint=1 -1 0.75}
 \IIIDKOSystem{5}
 \ThreeDput{\psframe*[linecolor=gray80](3,3)}
 \ThreeDput(1.5,1.5,0){\Huge below}
 \ThreeDput(0,0,1.5){\psframe*[linecolor=gray75](3,3)}
 \ThreeDput(1.5,1.5,1.5){\Huge center}
 \ThreeDput(0,0,3){\psframe*[linecolor=gray85](3,3)}
 \ThreeDput(1.5,1.5,3){\Huge above}
 \xzPlain{5}
 \ThreeDput(4,4,0){\psframe*[linecolor=gray95](-1,-1)(1,1)}
 \ThreeDput(4,4,0){\psdot[dotscale=3]}
\end{pspicture}
```



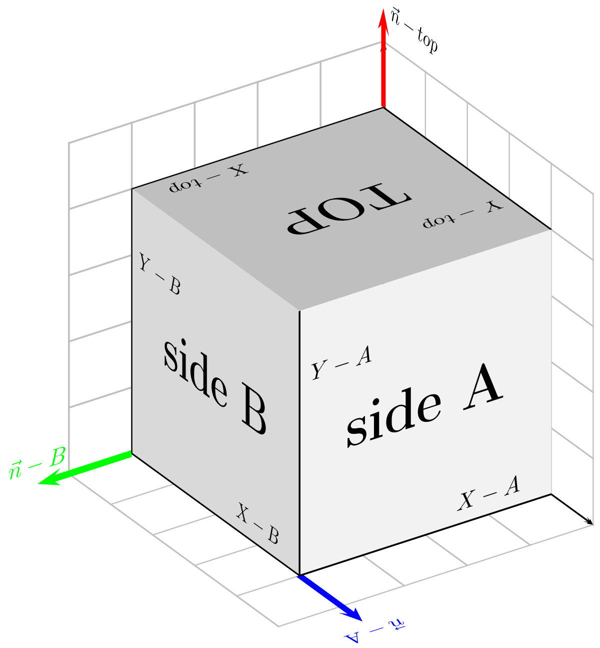
The coordinates of `ThreeDput` refer to the centre of the object, which does not necessarily need to be the geometrical centre.

```
\psframe(2,2)%          centre bottom left (0,0)
\psframe(-1,-1(1,1)%    centre in the middle (0,0)
arbitrary text%         centre in the middle of the base line
```

In the above example the smaller square with its centre $(0,0)$ has been set exactly to the coordinated $(4,4,0)$. The macro `ThreeDput` can be manifoldly applied, which is performed especially by the package `pst-vue3d`[?]. By specifying the normal vector $\vec{n}$ and a point $P(x,y,z)$ of the stright line and/or the plane the posture in space can be determined definitely. Areas can be provided with different levels of brightness to increase the spatial impression.

```
\newgray{gray75}{0.75}\newgray{gray85}{0.85}\newgray{gray95}{0.95}
\begin{pspicture}(-4.5,-3.5)(3,4.75)
\psset{viewpoint=1 1.5 1}
\IIIDKOSystem[gridlabels=0pt,gridcolor=lightgray,subgriddiv=0]{5}%
\ThreeDput[normal=0 0 1]{% xy-plane
  \psline[linewidth=3pt,linecolor=blue]{->}(4,4)(4,5.5)%
  \uput[90](4,5.5){\color{blue}$\vec{n}-{A}$}}%
\ThreeDput[normal=0 -1 0]{%  xz-plane
  \psline[linewidth=3pt,linecolor=green]{->}(4,0)(5.5,0)%
  \uput[90](5.5,0){\psscalebox{-1 1}{%
    \textcolor{green}{$\vec{n}-B$}}}}%
```

```
\ThreeDput[normal=1 0 0]{%  yz-plane
  \psline[linewidth=3pt,linecolor=red]{->}(0,4)(0,5.5)%
  \uput[0](0,5.5){$\vec{n}-{top}$}}% cube and axes
\ThreeDput[normal=0 0 1](0,0,4){%
  \psframe*[linecolor=gray75](4,4)\rput(2,2){\Huge\textbf{TOP}}}%
\ThreeDput[normal=0 1 0](4,4,0){%
  \psframe*[linecolor=gray95](4,4)\rput(2,2){\Huge\textbf{side A}}}%
\ThreeDput[normal=1 0 0](4,0,0){%
  \psframe*[linecolor=gray85](4,4)\rput(2,2){\Huge\textbf{side B}}}%
\ThreeDput[normal=0 0 1](0,0,4){%
  \psline(4,0)\uput[90](3,0){$X-top$}\psline(0,4)\uput[0](0,3){$Y-top$}}%
\ThreeDput[normal=0 1 0](4,4,0){%
  \psline(4,0)\uput[90](3,0){$X-A$}\psline(0,4)\uput[0](0,3){$Y-A$}}%
\ThreeDput[normal=1 0 0](4,0,0){%
  \psline(4,0)\uput[90](3,0){$X-B$}\psline(0,4)\uput[0](0,3){$Y-B$}}%
\end{pspicture}
```



4.2 3D parameters

Table 3 shows a compilation of the parameters which can be used to influence 3D representations.

4.2.1 `viewpoint`

The viewing direction to the 3D object influences the representation essentially. With `viewpoint` the (x, y, z) coordinates which denote the vector of the viewing

Table 3: Summary of all 3D parameters

name	values	default
`viewpoint`	`<valuex valuey valuez>`	`1 -1 1`
`viewangle`	`<angle>`	`0`
`normal`	`<valuex valuey valuez>`	`0 0 1`
`embedangle`	`<angle>`	`0`

direction are specified. Because of the parallel projection the length of this vector is unimportant, so that `(10.5 1.5)` and `(2 1 3)` yield the same representations. Figure 2 shows who somebody would regard this representation, whereat the representation itself is of course regarded from another point in this case, otherwise one had to look directly onto the vector.

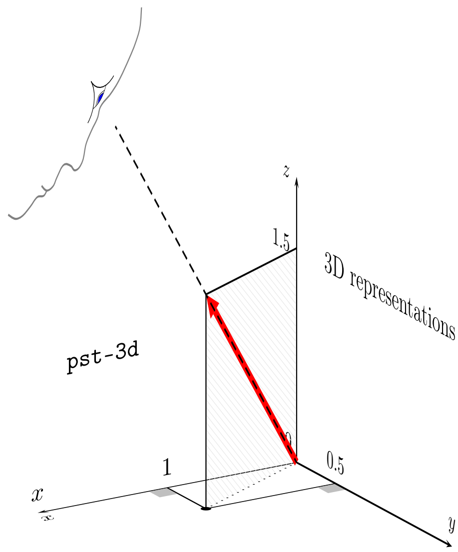


Figure 2: Definition of the `viewpoints`

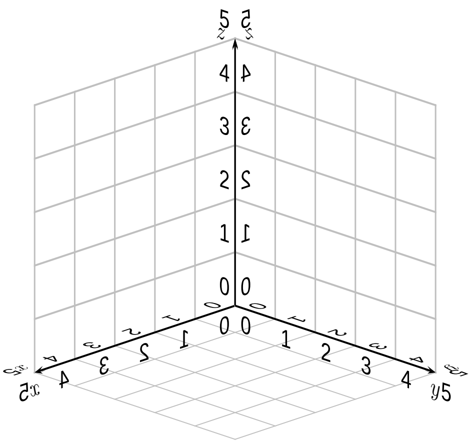
For figure 2 a viewpoint of `viewpoint=3 5 2` was defined. If one desires to regard it for instance from the y axis from a larger height, `viewpoint=0 1 3` could be chosen. The viewer has moved one unit in y direction and four units in z direction from the centre (origin) and regards everything from there.

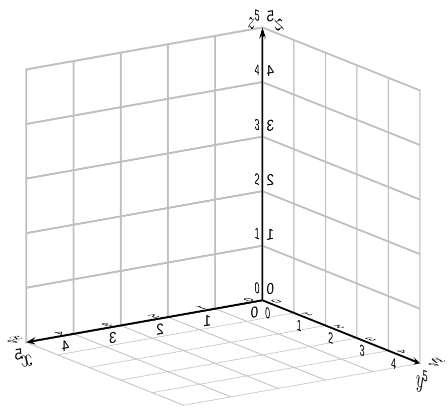
- The `viewpoint` principally **has** to be defined with values not equal to zero, for this would lead to a division by zero. Specifications of 0.001 for a coordinate are already sufficing to escape the division by zero and blind out the coordinate.

A good value for the viewpoint would be `viewpoint=1 1 0.5` for instance, which corresponds to a horizontal rotation by 45ř and a vertical by ca. 20ř.

Another meaningful point is also `viewpoint=1.5 1 0.5`, which now corresponds to a horizontal rotation by 33ř and the same vertical rotation. Both can be seen in the examples below.

```
\begin{pspicture}(-3,-2.5)(-3,4)
 \psset{unit=0.75}
 \psset{viewpoint=1 1 0.5}
 \IIIDKOSystem{5}
\end{pspicture}\hfill
\begin{pspicture}(-3,-2.5)(2.2,4)
 \psset{unit=0.75}
 \psset{viewpoint=1 1.5 0.5}
 \psset{gridlabels=6pt}
 \IIIDKOSystem{5}
\end{pspicture}
```



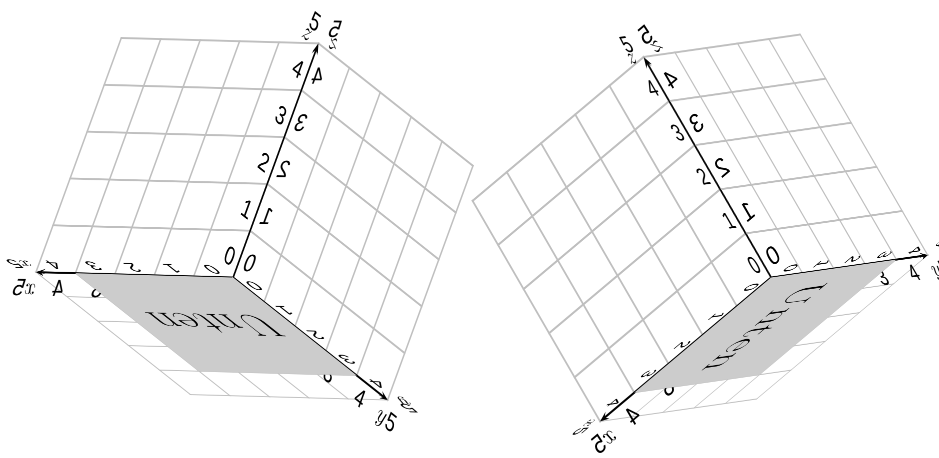


4.2.2 `viewangle`

Additional to the `viewpoint` option one can rotate the object by another option called `viewangle`. This could also be done by the macro `\rotatebox`, but `viewangle` has some advantages .

```
\begin{pspicture}(-1,-2.5)(4,4)
  \psset{unit=0.7,viewpoint=1 1 0.5,viewangle=20}
  \IIIDKOSystem{5}
```

```
  \ThreeDput(0,0,0){\psframe*[linecolor=gray80](4,4)}
  \ThreeDput(2,2,0){\Huge Unten}
\end{pspicture}
\begin{pspicture}(-3,-2.5)(1,4)
  \psset{unit=0.7,viewpoint=1 1.5 0.5,viewangle=-30}
  \IIIDKOSystem{5}
  \ThreeDput(0,0,0){\psframe*[linecolor=gray80](4,4)}
  \ThreeDput(2,2,0){\Huge Unten}
\end{pspicture}
```

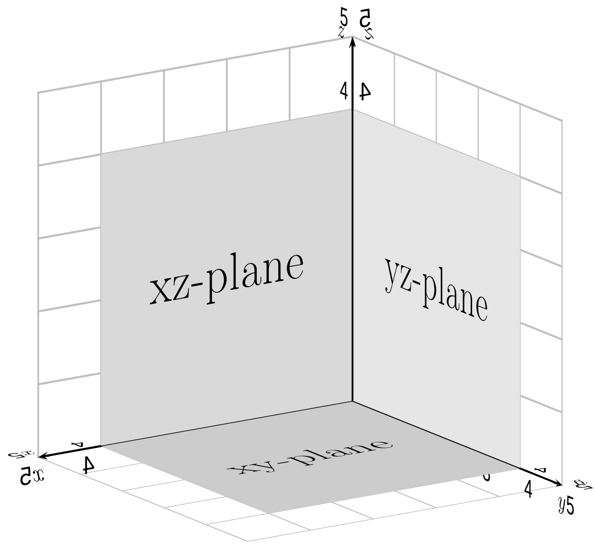


4.2.3 normal

`normal` denotes the direction of the normal vector which is perpendicular to a corresponding area. Therewith the posture of an object in three dimensional space is definitely determined by the normal vector.

```
\newgray{gray75}{0.75}\newgray{gray85}{0.85}\newgray{gray95}{0.95}
\begin{pspicture}(-3.5,-2.5)(-3,5)
 \psset{viewpoint=1 1.5 0.5}
 \IIIDKOSystem{5}
 \ThreeDput(0,0,0){\psframe*[linecolor=gray80](4,4)}
 \ThreeDput(2,2,0){\huge\psrotatedown{xy-plane}}
 \ThreeDput[normal=0 -1 0](0,0,0){\psframe*[linecolor=gray85](4,4)}
 \ThreeDput[normal=0 1 0](2,0,2){\huge xz-plane}
 \ThreeDput[normal=1 0 0](0,0,0){\psframe*[linecolor=gray90](4,4)}
 \ThreeDput[normal=1 0 0](0,2,2){\huge yz-plane}
 \ThreeDput[normal=0 0 1](0,0,0){% xy-plane
```

```
  \psline{->}(0,0)(0,5)\psline{->}(0,0)(5,0)}
 \ThreeDput[normal=0 1 0](0,0,0){\psline{->}(0,0)(0,5)}
\end{pspicture}
```



Without a assignment through the normal vector the above example could not have been created that easily. Let us step through the code for a better understanding.

`\psset{viewpoint=1 1.5 0.5}`: the `viewpoint` is set to the point $P(1, 1.5, 0.5)$.

`\IIIDKOSystem{5}`: first the coordinate system with the grid is drawn, so that axes and grid remain visible on the areas, which makes a better optical allocation possible.

`\ThreeDput(0,0,0){\psframe*[linecolor=gray80](4,4)}`: puts a square with a side length of four into the origin of ordinates with the lower left edge. Since no normal vector is specified here, the default value $\vec{n} = (0, 0, 1)$ is taken, wherewith the area is positioned in the first quadrant of the xy plane.

`\ThreeDput(2,2,0){\huge\psrotatedown{xy-plane}}`: puts the text rotated by 180ř centric to the point $(2, 2, 0)$ in the `xy-plane`.

`\ThreeDput[normal=0 -1 0](0,0,0){\psframe*[linecolor=gray85](4,4)}`: puts a square with a side length of four in the origin of ordinates with the lower left edge. Since the normal vector is the “negative” y axis, the square is positioned in the first quadrant of the xz plane. With `normal=0 1 0` it would have been the second quadrant.

`\ThreeDput[normal=0 1 0](2,0,2){\huge xz-plane}`: puts the text in the `xy-plane` centric to the point $(2, 0, 2)$. Because the xz plane is regarded from the back from the viewpoint, the normal vector of the area has to be reversed, otherwise the text would be read from the "back".

`\ThreeDput[normal=1 0 0](0,0,0){\psframe*[linecolor=gray90](4,4)}`: puts a square with a side length of four in the origin of ordinates with the lower left edge. The unit vector is the "positive" x axis, therefore the square is positioned in the first quadrant of the yz plane.

`\ThreeDput[normal=1 0 0](0,2,2){\huge yz-plane}`: puts the text in the `yz-plane` centric to the point $(0, 2, 2)$. Since the text is written at the "positive" side of the area, the normal vector stays the same.

`\ThreeDput[normal=0 0 1](0,0,0)`: the coordinate axes have been overwritten by the three areas and are redrawn now, first the xy axes.

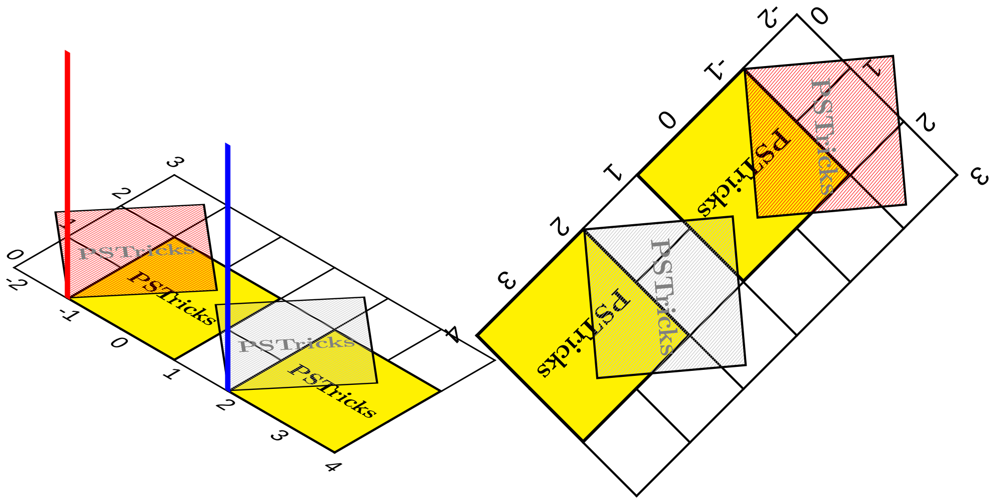
`\ThreeDput[normal=0 1 0](0,0,0)`: and now the z axis is drawn.

4.2.4 embedangle

With `viewangle` a rotation perpendicular to the plane of the viewer could be made. With `embedangle` a rotation perpendicular to the normal vector can be made. The counting of the angles is made in the mathematical sense, counterclockwise.

```
\newgray{gray75}{0.75}\newgray{gray85}{0.85}\newgray{gray95}{0.95}
\def\tBlack#1#2{%
 \psframe[style=#2](2,2)
 \rput(1,1){\textcolor{#1}{\textbf{PSTricks}}}}
\newpsstyle{SolidYellow}{fillstyle=solid,fillcolor=yellow}
\newpsstyle{TransparencyRed}{fillstyle=vlines,hatchcolor=red,
  hatchwidth=0.1\pslinewidth,hatchsep=1\pslinewidth}
\newpsstyle{TransparencyBlue}{fillstyle=vlines,hatchcolor=gray75,%
  hatchwidth=0.1\pslinewidth,hatchsep=1\pslinewidth}
\begin{pspicture}(-1.2,-1.75)(4.8,3.7)
  \ThreeDput{\psgrid[subgriddiv=0](-2,0)(4,3)}
  \ThreeDput(-1,0,0){\tBlack{black}{SolidYellow}}
  \ThreeDput(2,0,0){\tBlack{black}{SolidYellow}}
  \ThreeDput[embedangle=50](-1,0,0){\tBlack{gray}{TransparencyRed}}
  \ThreeDput[embedangle=50](2,0,0){\tBlack{gray}{TransparencyBlue}}
  \ThreeDput[normal=0 1 0](-1,0,0){\psline[linewidth=0.1,linecolor=red](0,4)}
  \ThreeDput[normal=0 1 0](2,0,0){\psline[linewidth=0.1,linecolor=blue](0,4)}
\end{pspicture}
\psset{viewpoint=1 1 100}
\begin{pspicture}(-2.5,-4.5)(2.8,1.7)
 \ThreeDput{\psgrid[subgriddiv=0](-2,0)(4,3)}
```

```
 \ThreeDput(-1,0,0){\tBlack{black}{SolidYellow}}
 \ThreeDput(2,0,0){\tBlack{black}{SolidYellow}}
 \ThreeDput[embedangle=50](-1,0,0){\tBlack{gray}{TransparencyRed}}
 \ThreeDput[embedangle=50](2,0,0){\tBlack{gray}{TransparencyBlue}}
 \ThreeDput[normal=0 1 0](-1,0,0){\psline[linewidth=0.1,linecolor=red](0,4)}
 \ThreeDput[normal=0 1 0](2,0,0){\psline[linewidth=0.1,linecolor=blue](0,4)}
\end{pspicture}
```



5 Driver file

The next bit of code contains the documentation driver file for TEX, i.e., the file that will produce the documentation you are currently reading. It will be extracted from this file by the `docstrip` program.

6 'pst-3d' LATEX wrapper

```
⟨*latex − wrapper⟩
%%
\RequirePackage{pstricks}
\ProvidesPackage{pst-3d}[2005/09/02 package wrapper for
  pst-3d.tex (hv)]
\input{pst-3d.tex}
\ProvidesFile{pst-3d.tex}
  [\filedate\space v\fileversion\space 'PST-3d' (hv)]
⟨/latex − wrapper⟩
```

7 'pst-3d' code

<*pst-3d>

`pst-3d` Require the basic `pstricks` package and for the key value operations the `pst-xkey` package.

```
\ifx\PSTricksLoaded\endinput\else\input pstricks.tex\fi
\ifx\PSTXKeyLoaded\endinput\else\input pst-xkey \fi % (hv 2005-09-03)
```

Catcodes changes.

```
\edef\PstAtCode{\the\catcode`\@}
\catcode`\@=11\relax
```

Add the key-family name to the xkeyval package

```
\pst@addfams{pst-3d}
```

Mark the package as loaded

```
\csname PSTthreeDLoaded\endcsname
\let\PSTthreeDLoaded\endinput
```

7.1 Basic 3D transformations

`\tx@SetMatrixThreeD`

Viewpoint for 3D coordinates is given by three angles: α, β and γ. α and β determine the direction from which one is looking. γ then determines the orientation of the observing. When α, β and γ are all zero, the observer is looking from the negative part of the y-axis, and sees the xz-plane the way in 2D one sees the xy plan. Hence, to convert the 3D coordinates to their 2D project, $\langle x, y, z\rangle$ map to $\langle x, z\rangle$. When the orientation is different, we rotate the coordinates, and then perform the same projection. We move up to latitude β, over to longitude α, and then rotate by γ. This means that we first rotate around y-axis by γ, then around x-axis by β, and the around z-axis by α.

Here are the matrices:

$$
\begin{aligned}
R_z(\alpha) &= \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
R_x(\beta) &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\beta & \sin\beta \\ 0 & -\sin\beta & \cos\beta \end{bmatrix} \\
R_y(\gamma) &= \begin{bmatrix} \cos\gamma & 0 & -\sin\gamma \\ 0 & 1 & 0 \\ \sin\gamma & 0 & \cos\gamma \end{bmatrix}
\end{aligned}
$$

The rotation of a coordinate is then performed by the matrix $R_z(\alpha)R_x(\beta)R_y(\gamma)$. The first and third columns of the matrix are the basis vectors of the plan upon which the 3D coordinates are project (the old basis vectors were $\langle 1, 0, 0\rangle$ and $\langle 0, 0, 1\rangle$; rotating these gives the first and third columns of the matrix).

These new base vectors are:

$$\begin{aligned}
\tilde{x} &= \begin{bmatrix} \cos\alpha\cos\gamma - \sin\beta\sin\alpha\sin\gamma \\ \sin\alpha\cos\gamma + \sin\beta\cos\alpha\sin\gamma \\ \cos\beta\sin\gamma \end{bmatrix} \\
\tilde{z} &= \begin{bmatrix} -\cos\alpha\sin\gamma - \sin\beta\sin\alpha\cos\gamma \\ -\sin\alpha\sin\gamma + \sin\beta\cos\alpha\cos\gamma \\ \cos\beta\cos\gamma \end{bmatrix}
\end{aligned}$$

Rather than specifying the angles α and β, the user gives a vector indicating where the viewpoint is. This new viewpoint is the rotation o the old viewpoint. The old viewpoint is $\langle 0, -1, 0\rangle$, and so the new viewpoint is

$$R_z(\alpha)R_x(\beta)\begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix} = \begin{bmatrix} \cos\beta\sin\alpha \\ -\cos\beta\cos\alpha \\ \sin\beta \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

Therefore,

$$\begin{aligned}
\alpha &= \texttt{arc}\tan(v_1/-v_2) \\
\beta &= \texttt{arc}\tan(v_3\sin\alpha/v_1)
\end{aligned}$$

Unless $p_1 = p_2 = 0$, in which case $\alpha = 0$ and $\beta = \texttt{sign}(p_3)90$, or $p_1 = p_3 = 0$, in which case $\beta = 0$.

The syntax of `SetMatrixThreeD` is

$$v_1\ v_2\ v_3\ \gamma\ \text{SetMatrixThreeD}$$

`SetMatrixThreeD` first computes

$$\begin{array}{ll} a = \sin\alpha & b = \cos\alpha \\ c = \sin\beta & d = \cos\beta \\ e = \sin\gamma & f = \cos\gamma \end{array}$$

and then sets `Matrix3D` to `[`$\tilde{x}\ \tilde{z}$`]`.

```
\pst@def{SetMatrixThreeD}<%
  dup sin /e ED cos /f ED
  /p3 ED /p2 ED /p1 ED
  p1 0 eq
  { /a 0 def /b p2 0 le { 1 } { -1 } ifelse def
    p3 p2 abs
  }
  { p2 0 eq
    { /a p1 0 lt { -1 } { 1 } ifelse def /b 0 def
      p3 p1 abs
    }
    { p1 dup mul p2 dup mul add sqrt dup
      p1 exch div /a ED
```

```
      p2 exch div neg /b ED
      p3 p1 a div
    }
    ifelse
  }
  ifelse
  atan dup sin /c ED cos /d ED
  /Matrix3D
  [
    b f mul c a mul e mul sub
    a f mul c b mul e mul add
    d e mul
    b e mul neg c a mul f mul sub
    a e mul neg c b mul f mul add
    d f mul
  ] def>
```

\tx@ProjThreeD The syntax of the macro `tx@ProjThreeD` is

$$x\ y\ z\ \mathrm{ProjThreeD}\ x'\ y'$$

where $x' = \langle x, y, z\rangle \cdot \tilde{x}$ and $y' = \langle x, y, z\rangle \cdot \tilde{z}$.

```
\pst@def{ProjThreeD}<%
  /z ED /y ED /x ED
  Matrix3D aload pop
  z mul exch y mul add exch x mul add
  4 1 roll
  z mul exch y mul add exch x mul add
  exch>
```

To embed 2D $\langle x, y\rangle$ coordinates in 3D, the user specifies the normal vector and an angle. If we decompose this normal vector into an angle, as when converting 3D coordinates to 2D coordinates, and let $\hat{\alpha}$, $\hat{\beta}$ and $\hat{\gamma}$ be the three angles, then when these angles are all zero the coordinate $\langle x, y\rangle$ gets mapped to $\langle x, 0, y\rangle$, and otherwise $\langle x, y\rangle$ gets mapped to

$$R_z(\hat{\alpha})R_x(\hat{\beta})R_y(\hat{\gamma}) \begin{bmatrix} x \\ 0 \\ y \end{bmatrix} = \begin{bmatrix} \hat{x}_1 x + \hat{z}_1 y \\ \hat{x}_2 x + \hat{z}_2 y \\ \hat{x}_3 x + \hat{z}_3 y \end{bmatrix}$$

where $\hat{x}$ and $\hat{z}$ are the first and third columns of $R_z(\hat{\alpha})R_x(\hat{\beta})R_y(\hat{\gamma})$.

Now add on a 3D-origin:

$$\begin{bmatrix} \hat{x}_1 x + \hat{z}_1 y + x_0 \\ \hat{x}_2 x + \hat{z}_2 y + y_0 \\ \hat{x}_3 x + \hat{z}_3 y + z_0 \end{bmatrix}$$

Now when we project back onto 2D coordinates, we get

$$
\begin{aligned}
x' &= \tilde{x}_1(\hat{x}_1 x + \hat{z}_1 y + x_0) + \tilde{x}_2(\hat{x}_2 x + \hat{z}_2 y + y_0) + \tilde{x}_3(\hat{x}_3 x + \hat{z}_3 y + z_0) \\
&= (\tilde{x}_1\hat{x}_1 + \tilde{x}_2\hat{x}_2 + \tilde{x}_3\hat{x}_3)x + (\tilde{x}_1\hat{z}_1 + \tilde{x}_2\hat{z}_2 + \tilde{x}_3\hat{z}_3)y + \tilde{x}_1 x_0 + \tilde{x}_2 y_0 + \tilde{z}_3 z_0 \\
y' &= \tilde{z}_1(\hat{x}_1 x + \hat{z}_1 y + x_0) + \tilde{z}_2(\hat{x}_2 x + \hat{z}_2 y + y_0) + \tilde{z}_3(\hat{x}_3 x + \hat{z}_3 y + z_0) \\
&= (\tilde{z}_1\hat{x}_1 + \tilde{z}_2\hat{x}_2 + \tilde{z}_3\hat{x}_3)x + (\tilde{z}_1\hat{z}_1 + \tilde{z}_2\hat{z}_2 + \tilde{z}_3\hat{z}_3)y + \tilde{z}_1 x_0 + \tilde{z}_2 y_0 + \tilde{z}_3 z_0
\end{aligned}
$$

Hence, the transformation matrix is:

$$
\begin{bmatrix}
\tilde{x}_1\hat{x}_1 + \tilde{x}_2\hat{x}_2 + \tilde{x}_3\hat{x}_3) \\
\tilde{z}_1\hat{x}_1 + \tilde{z}_2\hat{x}_2 + \tilde{z}_3\hat{x}_3) \\
\tilde{x}_1\hat{z}_1 + \tilde{x}_2\hat{z}_2 + \tilde{x}_3\hat{z}_3) \\
\tilde{z}_1\hat{z}_1 + \tilde{z}_2\hat{z}_2 + \tilde{z}_3\hat{z}_3) \\
\tilde{x}_1 x_0 + \tilde{x}_2 y_0 + \tilde{z}_3 z_0 \\
\tilde{z}_1 x_0 + \tilde{z}_2 y_0 + \tilde{z}_3 z_0
\end{bmatrix}
$$

`\tx@SetMatrixEmbed` The syntax of `SetMatrixEmbed` is

$$
x_0\ y_0\ z_0\ \hat{v}_1\ \hat{v}_2\ \hat{v}_3\ \hat{\gamma}
$$
$$
v_1\ v_2\ v_3\ \gamma\ \text{setMatrixEmbed}
$$

`SetMatrixEmbed` first sets `<x1 x2 x3 y1 y2 y3>` to the basis vectors for the viewpoint projection (the tilde stuff above). Then it sets `Matrix3D` to the basis vectors for the embedded plane. Finally, it sets the transformation matrix to the matrix given above.

```
\pst@def{SetMatrixEmbed}<%
  \tx@SetMatrixThreeD
  Matrix3D aload pop
  /z3 ED /z2 ED /z1 ED /x3 ED /x2 ED /x1 ED
  \tx@SetMatrixThreeD
  [
  Matrix3D aload pop
  z3 mul exch z2 mul add exch z1 mul add 4 1 roll
  z3 mul exch z2 mul add exch z1 mul add
  Matrix3D aload pop
  x3 mul exch x2 mul add exch x1 mul add 4 1 roll
  x3 mul exch x2 mul add exch x1 mul add
  3 -1 roll 3 -1 roll 4 -1 roll 8 -3 roll 3 copy
  x3 mul exch x2 mul add exch x1 mul add 4 1 roll
  z3 mul exch z2 mul add exch z1 mul add
  ]
  concat>
```

7.2 Parameter

`\psk@viewpoint` First we need a macro `\pssetzlength` for the third coordinate. It is adopted from the definition of the y-axes:

```
\let\pssetzlength\pssetylength
```

The viewpoint is set by its three coordinates (x y z). It is preset to $x = 1$, $y = -1$ and $z = 1$.

```
\define@key[psset]{pst-3d}{viewpoint}{%
  \pst@expandafter\psset@@viewpoint#1 {} {} {} \@nil
  \let\psk@viewpoint\pst@tempg}
\def\psset@@viewpoint#1 #2 #3 #4\@nil{%
  \begingroup
    \pssetxlength\pst@dima{#1}%
    \pssetylength\pst@dimb{#2}%
    \pssetzlength\pst@dimc{#3}%
    \xdef\pst@tempg{%
      \pst@number\pst@dima \pst@number\pst@dimb \pst@number\pst@dimc}%
  \endgroup}
\psset[pst-3d]{viewpoint=1 -1 1}
```

`\psk@viewangle`

```
\define@key[psset]{pst-3d}{viewangle}{%
  \pst@getangle{#1}\psk@viewangle}
\psset[pst-3d]{viewangle=0}
```

`\psk@normal`

```
\define@key[psset]{pst-3d}{normal}{%
  \pst@expandafter\psset@@viewpoint#1 {} {} {} \@nil
  \let\psk@normal\pst@tempg}
\psset[pst-3d]{normal=0 0 1}
```

`\psk@embedangle`

```
\define@key[psset]{pst-3d}{embedangle}{%
  \pst@getangle{#1}\psk@embedangle}
\psset[pst-3d]{embedangle=0}
```

`\psTshadowsize`

```
\define@key[psset]{pst-3d}{Tshadowsize}{%
  \pst@checknum{#1}\psTshadowsize}
\psset[pst-3d]{Tshadowsize=1}
```

`\psk@Tshadowangle`

```
\define@key[psset]{pst-3d}{Tshadowangle}{%
  \pst@getangle{#1}\psk@Tshadowangle}
\psset[pst-3d]{Tshadowangle=60}
```

`\psTshadowcolor`

```
\define@key[psset]{pst-3d}{Tshadowcolor}{%
  \pst@getcolor{#1}\psTshadowcolor}
\psset[pst-3d]{Tshadowcolor=lightgray}
```

7.3 PostScript code

\tx@TMSave

```
\pst@def{TMSave}<%
  tx@Dict /TMatrix known not { /TMatrix { } def /RAngle { 0 } def } if
  /TMatrix [ TMatrix CM ] cvx def>
```

\tx@TMRestore

```
\pst@def{TMRestore}<%
  CP /TMatrix [ TMatrix setmatrix ] cvx def moveto>
%
```

\tx@TMChange The syntax:

```
{<Proc for modifying tm>} TMChange
```

```
\pst@def{TMChange}<%
  \tx@TMSave
  /cp [ currentpoint ] cvx def % ??? Check this later.
  CM
```

Set "standard" coordinate system , with pt units and origin at currentpoint. This let's us rotate, or whatever, around TEX's current point, without having to worry about strange coordinate systems that the dvi-to-ps driver might be using.

```
  CP T \tx@STV
```

Let M = old matrix (on stack), and M' equal current matrix. Then go from M' to M by applying M Inv(M').

```
  CM matrix invertmatrix    % Inv(M')
  matrix concatmatrix       % M Inv(M')
```

Now modify transformation matrix:

```
  exch exec
```

Now apply M Inv(M')

```
  concat cp moveto>
```

7.4 Three dimensional operations

There is only one macro which collects all the basic operations for three dimansional representation of a text or graphic object.

\ThreeDput

```
\def\ThreeDput{\def\pst@par{}\pst@object{ThreeDput}}
\def\ThreeDput@i{\@ifnextchar({\ThreeDput@ii}{\ThreeDput@ii(\z@,\z@,\z@)}}
\def\ThreeDput@ii(#1,#2,#3){%
  \pst@killglue\pst@makebox{\ThreeDput@iii(#1,#2,#3)}}
\def\ThreeDput@iii(#1,#2,#3){%
  \begingroup
    \use@par
```

```
    \if@star\pst@starbox\fi
    \pst@makesmall\pst@hbox
    \pssetxlength\pst@dima{#1}%
    \pssetylength\pst@dimb{#2}%
    \pssetzlength\pst@dimc{#3}%
    \leavevmode
    \hbox{%
      \pst@Verb{%
        { \pst@number\pst@dima
          \pst@number\pst@dimb
          \pst@number\pst@dimc
          \psk@normal
          \psk@embedangle
          \psk@viewpoint
          \psk@viewangle
          \tx@SetMatrixEmbed
        } \tx@TMChange}%
    \box\pst@hbox
    \pst@Verb{\tx@TMRestore}}%
  \endgroup
  \ignorespaces}
```

7.5 Arithmetic

`\pst@sinandcos` Syntax:

```
%    \pst@sinandcos{<dim>}{<int>}
```

<dim>, in "sp" units, should equal 100,000 times the angle, in degrees between 0 and 90. <int> should equal the angle's quadrant (0, 1, 2 or 3). `\pst@dimg` is set to $\sin(\theta)$ and `\pst@dimh` is set to $\cos(\theta)$ (in pt's).

The algorithms uses the usual McLaurin expansion.

```
\def\pst@sinandcos#1{%
  \begingroup
    \pst@dima=#1\relax
    \pst@dima=.366022\pst@dima      %Now 1pt=1/32rad
    \pst@dimb=\pst@dima   % dimb->32sin(angle) in pts
    \pst@dimc=32\p@       % dimc->32cos(angle) in pts
    \pst@dimtonum\pst@dima\pst@tempa
    \pst@cntb=\tw@
    \pst@cntc=-\@ne
    \pst@cntg=32
    \loop
    \ifnum\pst@dima>\@cclvi % 256
      \pst@dima=\pst@tempa\pst@dima
      \divide\pst@dima\pst@cntg
      \divide\pst@dima\pst@cntb
      \ifodd\pst@cntb
        \advance\pst@dimb \pst@cntc\pst@dima
        \pst@cntc=-\pst@cntc
```

```
      \else
        \advance\pst@dimc by \pst@cntc\pst@dima
      \fi
      \advance\pst@cntb\@ne
    \repeat
    \divide\pst@dimb\pst@cntg
    \divide\pst@dimc\pst@cntg
    \global\pst@dimg\pst@dimb
    \global\pst@dimh\pst@dimc
  \endgroup}
```

`\pst@getsinandcos`

`\pst@getsinandcos` normalizes the angle to be in the first quadrant, sets `\pst@quadrant` to 0 for the first quadrant, 1 for the second, 2 for the third, and 3 for the fourth, invokes `\pst@sinandcos`, and sets `\pst@sin` to the sine and `\pst@cos` to the cosine.

```
\def\pst@getsinandcos#1{%
  \pst@dimg=100000sp
  \pst@dimg=#1\pst@dimg
  \pst@dimh=36000000sp
  \pst@cntg=0
  \loop
  \ifnum\pst@dimg<\z@
    \advance\pst@dimg\pst@dimh
  \repeat
  \loop
  \ifnum\pst@dimg>\pst@dimh
    \advance\pst@dimg-\pst@dimh
  \repeat
  \pst@dimh=9000000sp
  \def\pst@tempg{%
    \ifnum\pst@dimg<\pst@dimh\else
      \advance\pst@dimg-\pst@dimh
      \advance\pst@cntg\@ne
      \ifnum\pst@cntg>\thr@@ \advance\pst@cntg-4 \fi
      \expandafter\pst@tempg
    \fi}%
  \pst@tempg
  \chardef\pst@quadrant\pst@cntg
  \ifdim\pst@dimg=\z@
    \def\pst@sin{0}%
    \def\pst@cos{1}%
  \else
    \pst@sinandcos\pst@dimg
    \pst@dimtonum\pst@dimg\pst@sin
    \pst@dimtonum\pst@dimh\pst@cos
  \fi%
}
```

7.6 Tilting

\pstilt

```
\def\pstilt#1{\pst@makebox{\pstilt@{#1}}}
\def\pstilt@#1{%
  \begingroup
    \leavevmode
    \pst@getsinandcos{#1}%
    \hbox{%
      \ifcase\pst@quadrant
        \kern\pst@cos\dp\pst@hbox
        \pst@dima=\pst@cos\ht\pst@hbox
        \ht\pst@hbox=\pst@sin\ht\pst@hbox
        \dp\pst@hbox=\pst@sin\dp\pst@hbox
      \or
        \kern\pst@sin\ht\pst@hbox
        \pst@dima=\pst@sin\dp\pst@hbox
        \ht\pst@hbox=\pst@cos\ht\pst@hbox
        \dp\pst@hbox=\pst@cos\dp\pst@hbox
      \or
        \kern\pst@cos\ht\pst@hbox
        \pst@dima=\pst@sin\dp\pst@hbox
        \pst@dimg=\pst@sin\ht\pst@hbox
        \ht\pst@hbox=\pst@sin\dp\pst@hbox
        \dp\pst@hbox=\pst@dimg
      \or
        \kern\pst@sin\dp\pst@hbox
        \pst@dima=\pst@sin\ht\pst@hbox
        \pst@dimg=\pst@cos\ht\pst@hbox
        \ht\pst@hbox=\pst@cos\dp\pst@hbox
        \dp\pst@hbox=\pst@dimg
      \fi
      \pst@Verb{%
        { [ 1 0
            \pst@cos\space \ifnum\pst@quadrant>\@ne neg \fi
            \pst@sin\space
            \ifnum\pst@quadrant>\z@\ifnum\pst@quadrant<\thr@@ neg \fi\fi
            \ifodd\pst@quadrant exch \fi
            0 0
          ] concat
        } \tx@TMChange}%
      \box\pst@hbox
      \pst@Verb{\tx@TMRestore}%
      \kern\pst@dima}%
  \endgroup}
```

\psTilt

```
\def\psTilt#1{\pst@makebox{\psTilt@{#1}}}
\def\psTilt@#1{%
  \begingroup
```

```
    \leavevmode
    \pst@getsinandcos{#1}%
    \hbox{%
      \ifodd\pst@quadrant
        \pst@@divide{\dp\pst@hbox}{\pst@cos\p@}%
        \ifnum\pst@quadrant=\thr@@\kern\else\pst@dima=\fi\pst@sin\pst@dimg
        \pst@@divide{\ht\pst@hbox}{\pst@cos\p@}%
        \ifnum\pst@quadrant=\@ne\kern\else\pst@dima=\fi\pst@sin\pst@dimg
      \else
        \ifdim\pst@sin\p@=\z@
          \@pstrickserr{\string\psTilt\space angle cannot be 0 or 180}\@ehpa
          \def\pst@sin{.7071}%
          \def\pst@cos{.7071}%
        \fi
        \pst@@divide{\dp\pst@hbox}{\pst@sin\p@}%
        \ifnum\pst@quadrant=\z@\kern\else\pst@dima=\fi\pst@cos\pst@dimg
        \pst@@divide{\ht\pst@hbox}{\pst@sin\p@}%
        \ifnum\pst@quadrant=\tw@\kern\else\pst@dima=\fi\pst@cos\pst@dimg
      \fi
      \ifnum\pst@quadrant>\@ne
        \pst@dimg=\ht\pst@hbox
        \ht\pst@hbox=\dp\pst@hbox
        \dp\pst@hbox=\pst@dimg
      \fi
      \pst@Verb{%
        { [ 1 0
            \pst@cos\space \pst@sin\space
            \ifodd\pst@quadrant exch \fi
            \tx@Div
            \ifnum\pst@quadrant>\z@\ifnum\pst@quadrant<\thr@@ neg \fi\fi
            \ifnum\pst@quadrant>\@ne -1 \else 1 \fi
            0 0
          ] concat
        } \tx@TMChange}%
      \box\pst@hbox
      \pst@Verb{\tx@TMRestore}%
      \kern\pst@dima}%
  \endgroup}
```

7.7 Shadow

\psshadow

```
\def\psshadow{\pst@object{psshadow}}
\def\psshadow@i{\pst@makebox{\psshadow@ii}}
\def\psshadow@ii{%
  \begingroup
    \use@par
    \leavevmode
    \pst@getsinandcos{\psk@Tshadowangle}%
```

```
    \hbox{%
      \lower\dp\pst@hbox\hbox{%
        \pst@Verb{%
          { [ 1 0
              \pst@cos\space \psTshadowsize mul
              \ifnum\pst@quadrant>\@ne neg \fi
              \pst@sin\space \psTshadowsize mul
              \ifnum\pst@quadrant>\z@\ifnum\pst@quadrant<\thr@@ neg \fi\fi
              \ifodd\pst@quadrant exch \fi
              0 0
            ] concat
          } \tx@TMChange}}%
      \hbox to\z@{%   patch 2 (hv), to get it run with xcolor _and_ TeX
       \pst@Verb{ gsave \pst@usecolor\psTshadowcolor}%
       \copy\pst@hbox
       \pst@Verb{ grestore}\hss}%
%      \hbox to\z@{{\@nameuse{\psTshadowcolor}\copy\pst@hbox\hss}}%
      \pst@Verb{\tx@TMRestore}%
      \box\pst@hbox}%
  \endgroup}
```

7.8 Closing

Catcodes restoration.

```
\catcode`\@=\PstAtCode\relax
```

</pst-3d>

Change History

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.