

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 10, 2005 (hv)

Abstract

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

Contents

1	introduction	2
2	Shadow	3
2.1	Parameters	3
2.1.1	Tshadowangle	3
2.1.2	Tshadowcolor	4
2.1.3	Tshadowsize	4
3	Tilting	4
3.1	\pstilt	5
3.2	\psTilt	6
4	Three dimensional representations	7
4.1	\ThreeDput	7
4.2	3D parameters	10
4.2.1	viewpoint	10
4.2.2	viewangle	12
4.2.3	normal	13
4.2.4	embedangle	15
5	Driver file	16
6	‘pst-3d’ L^AT_EX wrapper	16
7	‘pst-3d’ code	16
7.1	Basic 3D transformations	17
7.2	Parameter	20
7.3	PostScript code	22
7.4	Three dimensional operations	22
7.5	Arithmetic	23
7.6	Tilting	25
7.7	Shadow	26
7.8	Closing	27

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 T_EX 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

<i>package</i>	<i>content</i>
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.

Shadow

$f(x) = x^2$



```

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

```

2.1 Parameters

Table 2 shows a compilation of the used parameters.

Table 2: Summary of all **shadow** parameters

<i>name</i>	<i>values</i>	<i>default</i>
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 90° therewith corresponds to the text itself. Negative angles cause the shadow to arise from the paper plane.

shadow
shadow
shadow
shadow

```

1 \newgray{gray75}{.75}
2 \psset{Tshadowcolor=gray75}
3 \psshadow{\huge shadow}\[5pt]
4 \psshadow[Tshadowangle=30]{\huge shadow}\[5pt]
5 \psshadow[Tshadowangle=70]{\huge shadow}\[5pt]
6 \psshadow[Tshadowangle=-30]{\huge shadow}

```

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

2.1.2 Tshadowcolor

Tshadowcolor defines the shadow colour.

shadow
shadow
shadow
shadow

```

1 \psshadow{\huge shadow}\[5pt]
2 \psshadow[Tshadowcolor=red]{\huge shadow}\[5pt]
3 \psshadow[Tshadowcolor=green]{\huge shadow}\[5pt]
4 \psshadow[Tshadowcolor=blue]{\huge shadow}

```

2.1.3 Tshadowsize

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

shadow
shadow
shadow
shadow
shadow

```

1 \psshadow{\Huge shadow}\[5pt]
2 \psshadow[Tshadowsize=0.5]{\Huge shadow}\[10pt]
3 \psshadow[Tshadowsize=1.5]{\Huge shadow}\[20pt]
4 \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),

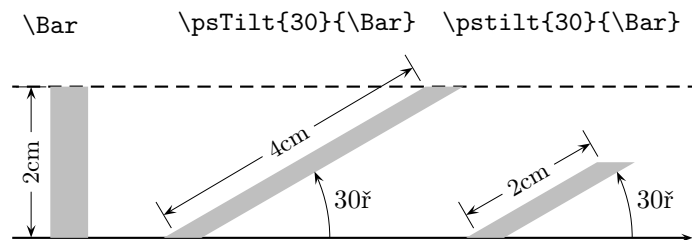
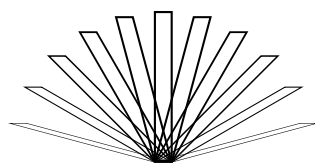


Figure 1: Demonstration of the difference between `\pstilt` and `\psTilt`

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

3.1 `\pstilt`

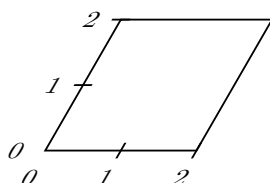
`\pstilt` tilts objects that their original height appears as new length of the tilted object, wherewith the object becomes smaller. The hypotenuse of the triangle from nadir, height and perpendicular now corresponds to the old height (see figure 1). At this the length is calculated from the middle of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



```

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

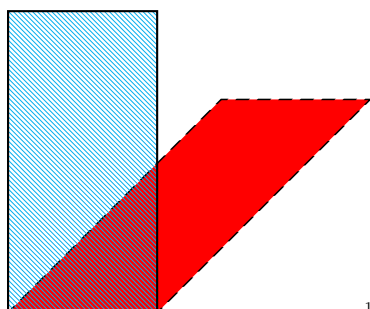
```



```

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

```

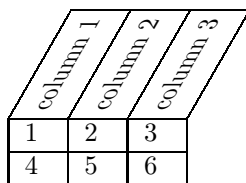


```

1 \newpsstyle{TCyan}{%
2   fillstyle=vlines,hatchcolor=cyan,
3   hatchwidth=0.1\pslinewidth,%
4   hatchsep=1.5\pslinewidth}
5 \begin{pspicture}(2,4)
6   \rput[1b](0,0){\pstilt{45}{%
7     \psframe[linestyle=dashed,%
8       fillstyle=solid,fillcolor=red](2,4)}}
9   \psframe[style=TCyan](0,0)(2,4)
10 \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.



1	2	3
4	5	6

```

1 \begin{tabular}{l}
2 \pstilt{60}{%
3 \begin{tabular}{|p{1em}|p{1em}|p{1em}|}\hline
4 \psrotateleft{column 1\ }
5 & \psrotateleft{column 2\ }
6 & \psrotateleft{column 3\ }
7 \end{tabular}}\
8 \begin{tabular}{|p{1em}|p{1em}|p{1em}|}\hline
9 1 & 2 & 3 \\\hline
10 4 & 5 & 6 \\\hline
11 \end{tabular}
12 \end{tabular}

```

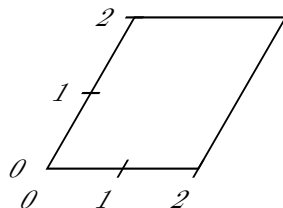
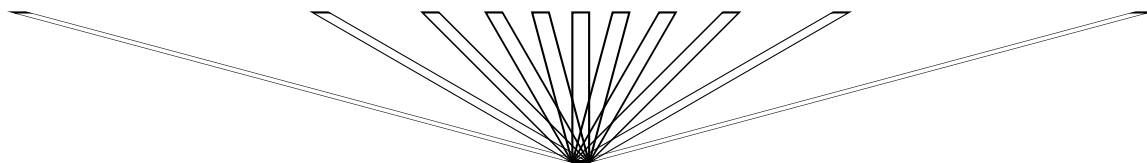
3.2 `\psTilt`

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

```

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

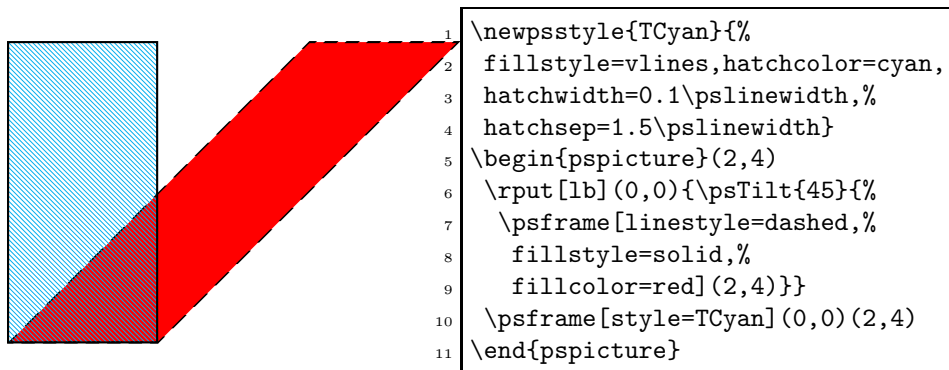
```



```

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

```



4 Three dimensional representations¹

`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 $\text{T}_{\text{E}}\text{X}$ sense, it has to be put in a `\parbox` or `minipage` before.

To simplify the specified source code, the macro `\IIIDK0System` is used in the following, which draws the coordinate axes with the grid and is not specified in the following anymore.

```
newgraygray850.85
```

```

1 \makeatletter
2 \def\xyPlain#1{%
3   \ThreeDput[normal=0 0 1](0,0,0){% xy-plane
4     \psgrid[subgriddiv=0,gridcolor=lightgray](0,0)(#1,#1)

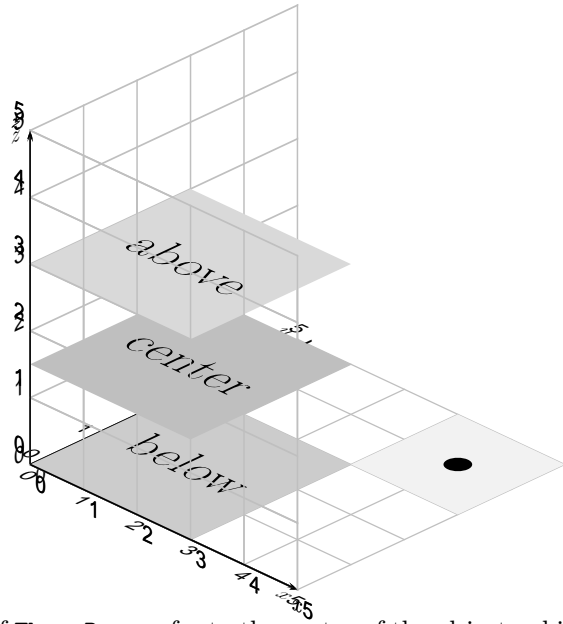
```

¹Some of the examples were created by Manuel Luque.

```

5 \psline{->}(0,0)(0,#1) \psline{->}(0,0)(#1,0)
6 \ifdim\psk@gridlabels pt>\z@
7 \uput[180]{0.2}(0,#1){$y$}\uput[-90]{0.2}(#1,0){$x$}\fi }}
8 \def\xzPlain#1{%
9 \ThreeDput[normal=0 -1 0](0,0,0){% xz-plane
10 \psgrid[subgriddiv=0,gridcolor=lightgray](0,0)(#1,#1)
11 \psline{->}(0,0)(0,5) \psline{->}(0,0)(#1,0)
12 \ifdim\psk@gridlabels pt>\z@
13 \uput[180]{0.2}(0,#1){$z$}\uput[-90]{0.2}(#1,0){$x$}%
14 \fi }}
15 \def\yzPlain#1{%
16 \ThreeDput[normal=1 0 0](0,0,0){% yz-plane
17 \psgrid[subgriddiv=0,gridcolor=lightgray](0,0)(#1,#1)
18 \psline{->}(0,0)(0,#1) \psline{->}(0,0)(#1,0)
19 \ifdim\psk@gridlabels pt>\z@
20 \uput[180]{0.2}(0,#1){$z$}\uput[-90]{0.2}(#1,0){$y$}%
21 \fi }}
22 \def\IIIDKOSystem{\@ifnextchar[{\IIIDKOSystem@i}{\IIIDKOSystem@i[]}}
23 \def\IIIDKOSystem@i[#1]#2{%
24 \psset{#1}%
25 \xyPlain{#2}\xzPlain{#2}\yzPlain{#2}}
26 \makeatother
27 \newgray{gray75}{0.75}
28 \newgray{gray80}{0.8}
29 \newgray{gray85}{0.85}
30 \newgray{gray95}{0.95}
31 \begin{pspicture}(0,-1.25)(5,6)
32 \psset{viewpoint=1 -1 0.75}
33 \IIIDKOSystem{5}
34 \ThreeDput{\psframe*[linecolor=gray80](3,3)}
35 \ThreeDput(1.5,1.5,0){\Huge below}
36 \ThreeDput(0,0,1.5){\psframe*[linecolor=gray75](3,3)}
37 \ThreeDput(1.5,1.5,1.5){\Huge center}
38 \ThreeDput(0,0,3){\psframe*[linecolor=gray85](3,3)}
39 \ThreeDput(1.5,1.5,3){\Huge above}
40 \xzPlain{5}
41 \ThreeDput(4,4,0){\psframe*[linecolor=gray95](-1,-1)(1,1)}
42 \ThreeDput(4,4,0){\psdot[dotsscale=3]}
43 \end{pspicture}

```

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

<code>\psframe(2,2)%</code>	centre bottom left (0,0)
<code>\psframe(-1,-1(1,1)%</code>	centre in the middle (0,0)
<code>arbitrary text%</code>	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.

```

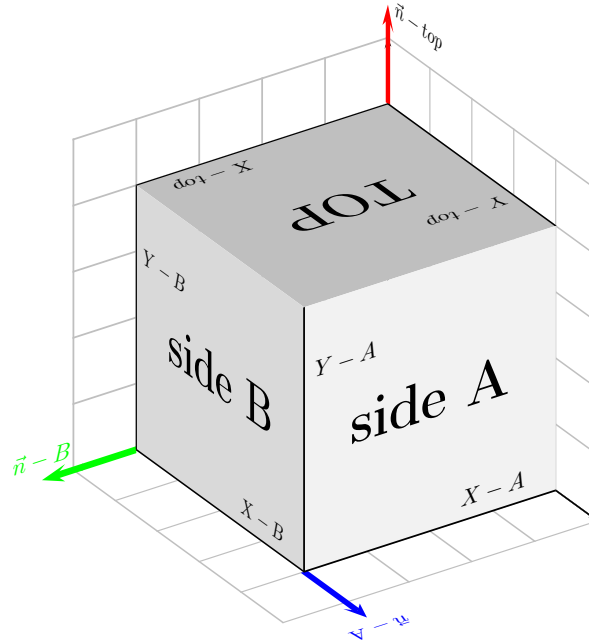
1 \newgray{gray75}{0.75}\newgray{gray85}{0.85}\newgray{gray95}{0.95}
2 \begin{pspicture}(-4.5,-3.5)(3,4.75)
3 \psset{viewpoint=1 1.5 1}
4 \IIIDKOSystem[gridlabels=0pt,gridcolor=lightgray,subgriddiv=0]{5}%
5 \ThreeDput[normal=0 0 1]{% xy-plane
6   \psline[linewidth=3pt,linecolor=blue]{->}(4,4)(4,5.5)%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

```

```

12 \ThreeDput[normal=1 0 0]{% yz-plane
13 \psline[linewidth=3pt,linecolor=red]{->}(0,4)(0,5.5)%
14 \uput[0](0,5.5){$\vec{n}-\{top\}$}}% cube and axes
15 \ThreeDput[normal=0 0 1](0,0,4){%
16 \psframe*[linecolor=gray75](4,4)\rput(2,2){\Huge\textbf{TOP}}}%
17 \ThreeDput[normal=0 1 0](4,4,0){%
18 \psframe*[linecolor=gray95](4,4)\rput(2,2){\Huge\textbf{side A}}}%
19 \ThreeDput[normal=1 0 0](4,0,0){%
20 \psframe*[linecolor=gray85](4,4)\rput(2,2){\Huge\textbf{side B}}}%
21 \ThreeDput[normal=0 0 1](0,0,4){%
22 \psline(4,0)\uput[90](3,0){$X-top$}\psline(0,4)\uput[0](0,3){$Y-top$}}%
23 \ThreeDput[normal=0 1 0](4,4,0){%
24 \psline(4,0)\uput[90](3,0){$X-A$}\psline(0,4)\uput[0](0,3){$Y-A$}}%
25 \ThreeDput[normal=1 0 0](4,0,0){%
26 \psline(4,0)\uput[90](3,0){$X-B$}\psline(0,4)\uput[0](0,3){$Y-B$}}%
27 \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
<code>viewpoint</code>	<code><valuex valuey valuez></code>	1 -1 1
<code>viewangle</code>	<code><angle></code>	0
<code>normal</code>	<code><valuex valuey valuez></code>	0 0 1
<code>embedangle</code>	<code><angle></code>	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.

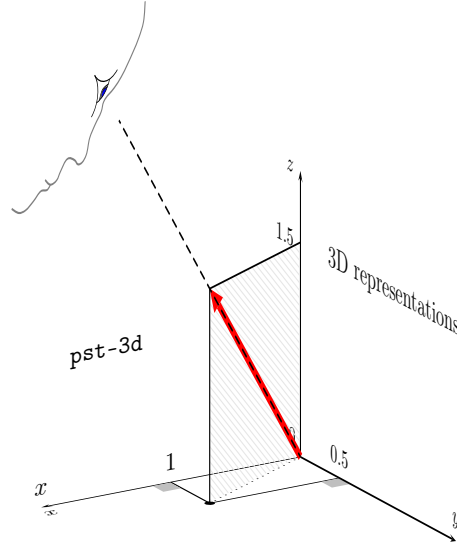


Figure 2: Definition of the viewpoints

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.

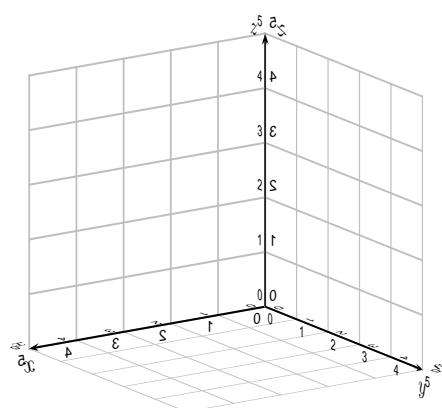
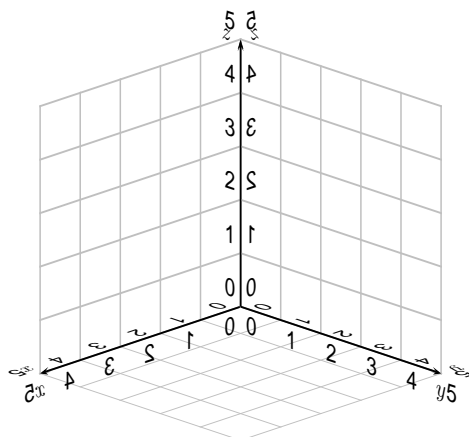
A good value for the viewpoint would be `viewpoint=1 1 0.5` for instance, which corresponds to a horizontal rotation by 45° and a vertical by ca. 20°.

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

```

1 \begin{pspicture}(-3,-2.5)(-3,4)
2   \psset{unit=0.75}
3   \psset{viewpoint=1 1 0.5}
4   \IIIDKOSystem{5}
5 \end{pspicture}\hfill
6 \begin{pspicture}(-3,-2.5)(2.2,4)
7   \psset{unit=0.75}
8   \psset{viewpoint=1 1.5 0.5}
9   \psset{gridlabels=6pt}
10  \IIIDKOSystem{5}
11 \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 .

```

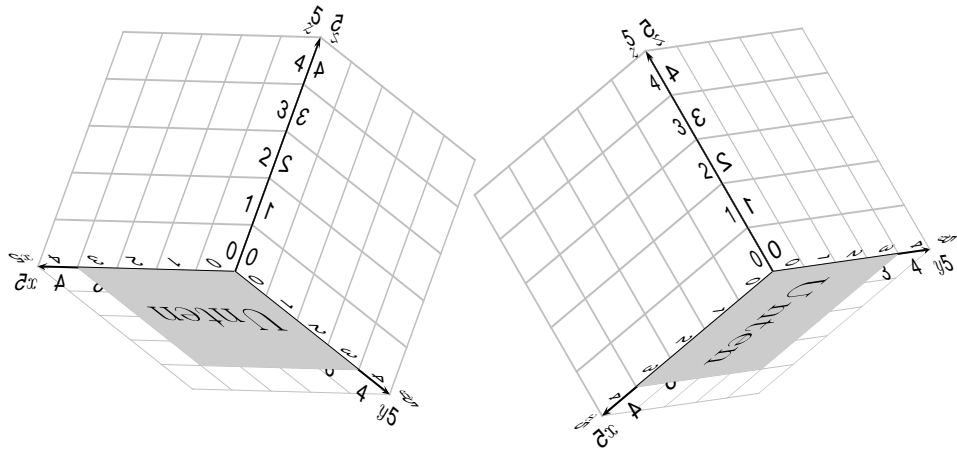
1 \begin{pspicture}(-1,-2.5)(4,4)
2   \psset{unit=0.7,viewpoint=1 1 0.5,viewangle=20}
3   \IIIDKOSystem{5}

```

```

4 \ThreeDput(0,0,0){\psframe*[linecolor=gray80](4,4)}
5 \ThreeDput(2,2,0){\Huge Unten}
6 \end{pspicture}
7 \begin{pspicture}(-3,-2.5)(1,4)
8 \psset{unit=0.7,viewpoint=1 1.5 0.5,viewangle=-30}
9 \IIIDKOSystem{5}
10 \ThreeDput(0,0,0){\psframe*[linecolor=gray80](4,4)}
11 \ThreeDput(2,2,0){\Huge Unten}
12 \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.

```

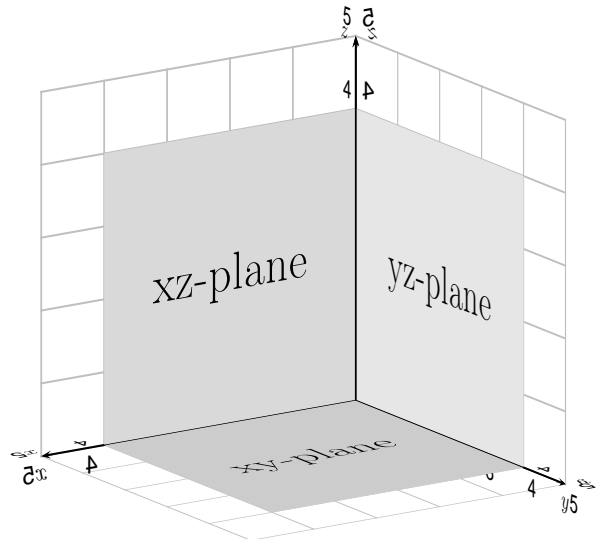
1 \newgray{gray75}{0.75}\newgray{gray85}{0.85}\newgray{gray95}{0.95}
2 \begin{pspicture}(-3.5,-2.5)(-3,5)
3 \psset{viewpoint=1 1.5 0.5}
4 \IIIDKOSystem{5}
5 \ThreeDput(0,0,0){\psframe*[linecolor=gray80](4,4)}
6 \ThreeDput(2,2,0){\huge\psrotatedown{xy-plane}}
7 \ThreeDput[normal=0 -1 0](0,0,0){\psframe*[linecolor=gray85](4,4)}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

```

```

12 \psline{->}(0,0)(0,5)\psline{->}(0,0)(5,0)}
13 \ThreeDput[normal=0 1 0](0,0,0){\psline{->}(0,0)(0,5)}
14 \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 180° 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.

```

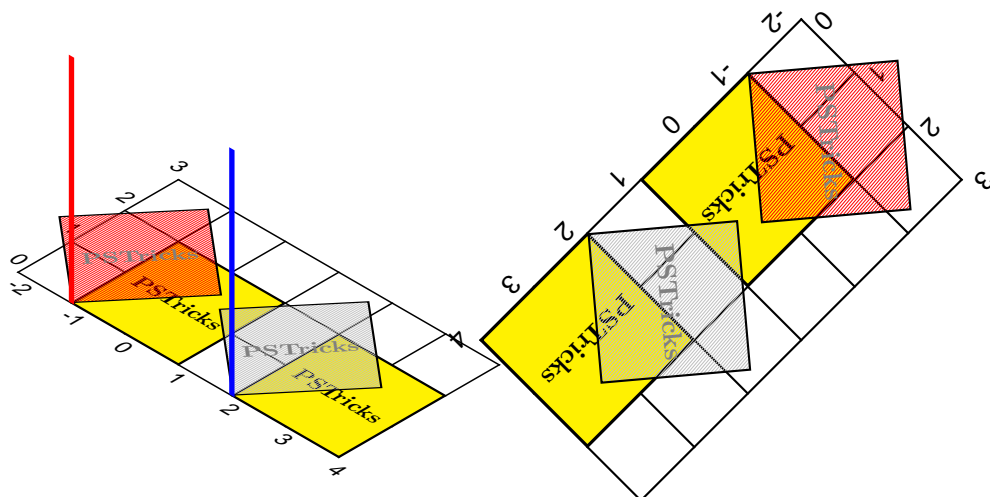
1 \newgray{gray75}{0.75}\newgray{gray85}{0.85}\newgray{gray95}{0.95}
2 \def\tBlack#1#2{%
3   \psframe[style=#2](2,2)
4   \rput(1,1){\textcolor{#1}{\textbf{PSTricks}}}}
5 \newpsstyle{SolidYellow}{fillstyle=solid,fillcolor=yellow}
6 \newpsstyle{TransparencyRed}{fillstyle=vlines,hatchcolor=red,
7   hatchwidth=0.1\pslinewidth,hatchsep=1\pslinewidth}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

```

```

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

```



5 Driver file

The next bit of code contains the documentation driver file for $\text{T}_{\text{E}}\text{X}$, 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’ $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ wrapper

```

1 <*latex – wrapper>
2 %%
3 \RequirePackage{pstricks}
4 \ProvidesPackage{pst-3d}[2005/09/02 package wrapper for
5   pst-3d.tex (hv)]
6 \input{pst-3d.tex}
7 \ProvidesFile{pst-3d.tex}
8   [\filedate\space v\fileversion\space ‘PST-3d’ (tvz)]
9 </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.

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

Catcodes changes.

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

Add the key-family name to the `xkeyval` package

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

Mark the package as loaded

```
15 \csname PSTthreeDLoaded\endcsname
16 \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 of 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,

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

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

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

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

```
17 \pst@def{SetMatrixThreeD}<%
18   dup sin /e ED cos /f ED
19   /p3 ED /p2 ED /p1 ED
20   p1 0 eq
21   { /a 0 def /b p2 0 le { 1 } { -1 } ifelse def
22     p3 p2 abs
23   }
24   { p2 0 eq
25     { /a p1 0 lt { -1 } { 1 } ifelse def /b 0 def
26       p3 p1 abs
27     }
28     { p1 dup mul p2 dup mul add sqrt dup
29       p1 exch div /a ED
```

```

30      p2 exch div neg /b ED
31      p3 p1 a div
32    }
33    ifelse
34  }
35  ifelse
36  atan dup sin /c ED cos /d ED
37  /Matrix3D
38  [
39    b f mul c a mul e mul sub
40    a f mul c b mul e mul add
41    d e mul
42    b e mul neg c a mul f mul sub
43    a e mul neg c b mul f mul add
44    d f mul
45  ] def>

```

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

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

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

```

46 \pst@def{ProjThreeD}<%
47   /z ED /y ED /x ED
48   Matrix3D aload pop
49   z mul exch y mul add exch x mul add
50   4 1 roll
51   z mul exch y mul add exch x mul add
52   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

$$R_z(\hat{\alpha})R_x(\hat{\beta})R_y(\hat{\gamma}) \begin{bmatrix} x \\ 0 \\ y \end{bmatrix} = \begin{bmatrix} \hat{x}_1x + \hat{z}_1y \\ \hat{x}_2x + \hat{z}_2y \\ \hat{x}_3x + \hat{z}_3y \end{bmatrix}$$

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:

$$\begin{bmatrix} \hat{x}_1x + \hat{z}_1y + x_0 \\ \hat{x}_2x + \hat{z}_2y + y_0 \\ \hat{x}_3x + \hat{z}_3y + z_0 \end{bmatrix}$$

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

$$\begin{aligned}
x' &= \tilde{x}_1(\hat{x}_1x + \hat{z}_1y + x_0) + \tilde{x}_2(\hat{x}_2x + \hat{z}_2y + y_0) + \tilde{x}_3(\hat{x}_3x + \hat{z}_3y + z_0) \\
&= (\tilde{x}_1\hat{x}_1 + \tilde{x}_2\hat{x}_2 + \tilde{x}_3\hat{x}_3)x + (\tilde{x}_1\hat{z}_1 + \tilde{x}_2\hat{z}_2 + \tilde{x}_3\hat{z}_3)y + \tilde{x}_1x_0 + \tilde{x}_2y_0 + \tilde{x}_3z_0 \\
y' &= \tilde{z}_1(\hat{x}_1x + \hat{z}_1y + x_0) + \tilde{z}_2(\hat{x}_2x + \hat{z}_2y + y_0) + \tilde{z}_3(\hat{x}_3x + \hat{z}_3y + z_0) \\
&= (\tilde{z}_1\hat{x}_1 + \tilde{z}_2\hat{x}_2 + \tilde{z}_3\hat{x}_3)x + (\tilde{z}_1\hat{z}_1 + \tilde{z}_2\hat{z}_2 + \tilde{z}_3\hat{z}_3)y + \tilde{z}_1x_0 + \tilde{z}_2y_0 + \tilde{z}_3z_0
\end{aligned}$$

Hence, the transformation matrix is:

$$\begin{bmatrix}
\tilde{x}_1\hat{x}_1 + \tilde{x}_2\hat{x}_2 + \tilde{x}_3\hat{x}_3 \\
\tilde{z}_1\hat{x}_1 + \tilde{z}_2\hat{x}_2 + \tilde{z}_3\hat{x}_3 \\
\tilde{x}_1\hat{z}_1 + \tilde{x}_2\hat{z}_2 + \tilde{x}_3\hat{z}_3 \\
\tilde{z}_1\hat{z}_1 + \tilde{z}_2\hat{z}_2 + \tilde{z}_3\hat{z}_3 \\
\tilde{x}_1x_0 + \tilde{x}_2y_0 + \tilde{x}_3z_0 \\
\tilde{z}_1x_0 + \tilde{z}_2y_0 + \tilde{z}_3z_0
\end{bmatrix}$$

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

$$\begin{array}{ccccccc}
x_0 & y_0 & z_0 & \hat{v}_1 & \hat{v}_2 & \hat{v}_3 & \hat{\gamma} \\
v_1 & v_2 & v_3 & \gamma & \text{setMatrixEmbed}
\end{array}$$

`SetMatrixEmbed` first sets `<x1 x2 x3 y1 y2 y3>` to the basis vectors for the view-point 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.

```

53 \pst@def{SetMatrixEmbed}<%
54 \tx@SetMatrixThreeD
55 Matrix3D aload pop
56 /z3 ED /z2 ED /z1 ED /x3 ED /x2 ED /x1 ED
57 \tx@SetMatrixThreeD
58 [
59 Matrix3D aload pop
60 z3 mul exch z2 mul add exch z1 mul add 4 1 roll
61 z3 mul exch z2 mul add exch z1 mul add
62 Matrix3D aload pop
63 x3 mul exch x2 mul add exch x1 mul add 4 1 roll
64 x3 mul exch x2 mul add exch x1 mul add
65 3 -1 roll 3 -1 roll 4 -1 roll 8 -3 roll 3 copy
66 x3 mul exch x2 mul add exch x1 mul add 4 1 roll
67 z3 mul exch z2 mul add exch z1 mul add
68 ]
69 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:

```

70 \let\pssetzlength\pssetylength

```

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

```

71 \define@key[psset]{pst-3d}{viewpoint}{%
72   \pst@expandafter\psset@@viewpoint#1 {} {} {} \@nil
73   \let\psk@viewpoint\pst@tempg}
74 \def\psset@@viewpoint#1 #2 #3 #4\@nil{%
75   \begingroup
76     \pssetxlength\pst@dima{#1}%
77     \pssetylength\pst@dimb{#2}%
78     \pssetzlength\pst@dimc{#3}%
79     \xdef\pst@tempg{%
80       \pst@number\pst@dima \pst@number\pst@dimb \pst@number\pst@dimc}%
81   \endgroup}
82 \psset[pst-3d]{viewpoint=1 -1 1}

```

\psk@viewangle

```

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

```

\psk@normal

```

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

```

\psk@embedangle

```

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

```

\psTshadowsize

```

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

```

\psk@Tshadowangle

```

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

```

\psTshadowcolor

```

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

```

7.3 PostScript code

```

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

\tx@TMRestore
105 \pst@def{TMRestore}<%
106   CP /TMatrix [ TMatrix setmatrix ] cvx def moveto>
107 %

\tx@TMChange The syntax:
               {<Proc for modifying tm>} TMChange

108 \pst@def{TMChange}<%
109   \tx@TMSave
110   /cp [ currentpoint ] cvx def % ??? Check this later.
111   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.
112   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').
113   CM matrix invertmatrix      % Inv(M')
114   matrix concatmatrix         % M Inv(M')

   Now modify transformation matrix:
115   exch exec

   Now apply M Inv(M')
116   concat cp moveto>

```

7.4 Three dimensional operations

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

```

\ThreeDput
117 \def\ThreeDput{\def\pst@par{}\pst@object{ThreeDput}}
118 \def\ThreeDput@i{\@ifnextchar(\ThreeDput@ii){\ThreeDput@ii(\z@,\z@,\z@)}}
119 \def\ThreeDput@ii(#1,#2,#3){%
120   \pst@killglue\pst@makebox{\ThreeDput@iii(#1,#2,#3)}}
121 \def\ThreeDput@iii(#1,#2,#3){%
122   \begingroup
123   \use@par

```

```

124 \if@star\pst@starbox\fi
125 \pst@makesmall\pst@hbox
126 \pssetxlength\pst@dima{#1}%
127 \pssetylength\pst@dimb{#2}%
128 \pssetzlength\pst@dimc{#3}%
129 \leavevmode
130 \hbox{%
131   \pst@Verb{%
132     { \pst@number\pst@dima
133       \pst@number\pst@dimb
134       \pst@number\pst@dimc
135       \psk@normal
136       \psk@embedangle
137       \psk@viewpoint
138       \psk@viewangle
139       \tx@SetMatrixEmbed
140     } \tx@TMChange}%
141   \box\pst@hbox
142   \pst@Verb{\tx@TMRestore}}}%
143 \endgroup
144 \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.

```

145 \def\pst@sinandcos#1{%
146   \begingroup
147     \pst@dima=#1\relax
148     \pst@dima=.366022\pst@dima      %Now 1pt=1/32rad
149     \pst@dimb=\pst@dima % dimb->32sin(angle) in pts
150     \pst@dimc=32\p@ % dimc->32cos(angle) in pts
151     \pst@dimtonum\pst@dima\pst@tempa
152     \pst@cntb=\tw@
153     \pst@cntc=-\@ne
154     \pst@cntg=32
155     \loop
156     \ifnum\pst@dima>\@cclvi % 256
157       \pst@dima=\pst@tempa\pst@dima
158       \divide\pst@dima\pst@cntg
159       \divide\pst@dima\pst@cntb
160       \ifodd\pst@cntb
161         \advance\pst@dimb \pst@cntc\pst@dima
162         \pst@cntc=-\pst@cntc

```

```

163     \else
164     \advance\pst@dimc by \pst@cntc\pst@dima
165     \fi
166     \advance\pst@cntb\@ne
167     \repeat
168     \divide\pst@dimb\pst@cntg
169     \divide\pst@dimc\pst@cntg
170     \global\pst@ding\pst@dimb
171     \global\pst@dimh\pst@dimc
172 \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.

```

173 \def\pst@getsinandcos#1{%
174   \pst@ding=100000sp
175   \pst@ding=#1\pst@ding
176   \pst@dimh=36000000sp
177   \pst@cntg=0
178   \loop
179   \ifnum\pst@ding<\z@
180     \advance\pst@ding\pst@dimh
181   \repeat
182   \loop
183   \ifnum\pst@ding>\pst@dimh
184     \advance\pst@ding-\pst@dimh
185   \repeat
186   \pst@dimh=9000000sp
187   \def\pst@tempg{%
188     \ifnum\pst@ding<\pst@dimh\else
189       \advance\pst@ding-\pst@dimh
190       \advance\pst@cntg\@ne
191       \ifnum\pst@cntg>\thr@@ \advance\pst@cntg-4 \fi
192       \expandafter\pst@tempg
193     \fi}%
194   \pst@tempg
195   \chardef\pst@quadrant\pst@cntg
196   \ifdim\pst@ding=\z@
197     \def\pst@sin{0}%
198     \def\pst@cos{1}%
199   \else
200     \pst@sinandcos\pst@ding
201     \pst@dimtonum\pst@ding\pst@sin
202     \pst@dimtonum\pst@dimh\pst@cos
203   \fi%
204 }

```


7.6 Tilting

`\pstilt`

```

205 \def\pstilt#1{\pst@makebox{\pstilt@{#1}}}
206 \def\pstilt@#1{%
207   \begingroup
208     \leavevmode
209     \pst@getsinandcos{#1}%
210     \hbox{%
211       \ifcase\pst@quadrant
212         \kern\pst@cos\dp\pst@hbox
213         \pst@dima=\pst@cos\ht\pst@hbox
214         \ht\pst@hbox=\pst@sin\ht\pst@hbox
215         \dp\pst@hbox=\pst@sin\dp\pst@hbox
216       \or
217         \kern\pst@sin\ht\pst@hbox
218         \pst@dima=\pst@sin\dp\pst@hbox
219         \ht\pst@hbox=\pst@cos\ht\pst@hbox
220         \dp\pst@hbox=\pst@cos\dp\pst@hbox
221       \or
222         \kern\pst@cos\ht\pst@hbox
223         \pst@dima=\pst@sin\dp\pst@hbox
224         \pst@ding=\pst@sin\ht\pst@hbox
225         \ht\pst@hbox=\pst@sin\dp\pst@hbox
226         \dp\pst@hbox=\pst@ding
227       \or
228         \kern\pst@sin\dp\pst@hbox
229         \pst@dima=\pst@sin\ht\pst@hbox
230         \pst@ding=\pst@cos\ht\pst@hbox
231         \ht\pst@hbox=\pst@cos\dp\pst@hbox
232         \dp\pst@hbox=\pst@ding
233       \fi
234       \pst@Verb{%
235         { [ 1 0
236           \pst@cos\space \ifnum\pst@quadrant>\@ne neg \fi
237           \pst@sin\space
238           \ifnum\pst@quadrant>z@\ifnum\pst@quadrant<\thr@@ neg \fi\fi
239           \ifodd\pst@quadrant exch \fi
240           0 0
241         ] concat
242         } \tx@TMChange}%
243       \box\pst@hbox
244       \pst@Verb{\tx@TMRestore}%
245       \kern\pst@dima}%
246     \endgroup}

```

`\psTilt`

```

247 \def\psTilt#1{\pst@makebox{\psTilt@{#1}}}
248 \def\psTilt@#1{%
249   \begingroup

```

```

250 \leavevmode
251 \pst@getsinandcos{#1}%
252 \hbox{%
253   \ifodd\pst@quadrant
254     \pst@@divide{\dp\pst@hbox}{\pst@cos\p}%
255     \ifnum\pst@quadrant=\thr@@\kern\else\pst@dima=\fi\pst@sin\pst@dimg
256     \pst@@divide{\ht\pst@hbox}{\pst@cos\p}%
257     \ifnum\pst@quadrant=\@ne\kern\else\pst@dima=\fi\pst@sin\pst@dimg
258   \else
259     \ifdim\pst@sin\p=\z@
260       \@pstrickserr{\string\psTilt\space angle cannot be 0 or 180}\@ehpa
261       \def\pst@sin{.7071}%
262       \def\pst@cos{.7071}%
263     \fi
264     \pst@@divide{\dp\pst@hbox}{\pst@sin\p}%
265     \ifnum\pst@quadrant=\z@\kern\else\pst@dima=\fi\pst@cos\pst@dimg
266     \pst@@divide{\ht\pst@hbox}{\pst@sin\p}%
267     \ifnum\pst@quadrant=\tw@\kern\else\pst@dima=\fi\pst@cos\pst@dimg
268   \fi
269   \ifnum\pst@quadrant>\@ne
270     \pst@dimg=\ht\pst@hbox
271     \ht\pst@hbox=\dp\pst@hbox
272     \dp\pst@hbox=\pst@dimg
273   \fi
274   \pst@Verb{%
275     { [ 1 0
276       \pst@cos\space \pst@sin\space
277       \ifodd\pst@quadrant exch \fi
278       \tx@Div
279       \ifnum\pst@quadrant>\z@\ifnum\pst@quadrant<\thr@@ neg \fi\fi
280       \ifnum\pst@quadrant>\@ne -1 \else 1 \fi
281       0 0
282     ] concat
283     } \tx@TMChange}%
284   \box\pst@hbox
285   \pst@Verb{\tx@TMRestore}%
286   \kern\pst@dima}%
287 \endgroup}

```

7.7 Shadow

\psshadow

```

288 \def\psshadow{\pst@object{psshadow}}
289 \def\psshadow@i{\pst@makebox{\psshadow@ii}}
290 \def\psshadow@ii{%
291   \begingroup
292     \use@par
293     \leavevmode
294     \pst@getsinandcos{\psk@Tshadowangle}%

```

```

295 \hbox{%
296 \lower\dp\pst@hbox\hbox{%
297 \pst@Verb{%
298 { [ 1 0
299 \pst@cos\space \psTshadowsize mul
300 \ifnum\pst@quadrant>\@ne neg \fi
301 \pst@sin\space \psTshadowsize mul
302 \ifnum\pst@quadrant>\z@\ifnum\pst@quadrant<\thr@@ neg \fi\fi
303 \ifodd\pst@quadrant exch \fi
304 0 0
305 ] concat
306 } \tx@TMChange}}%
307 \hbox to\z@{% patch 2 (hv), to get it run with xcolor _and_ TeX
308 \pst@Verb{ gsave \pst@usecolor\psTshadowcolor}%
309 \copy\pst@hbox
310 \pst@Verb{ grestore}\hss}%
311 % \hbox to\z@{\@nameuse{\psTshadowcolor}\copy\pst@hbox\hss}}%
312 \pst@Verb{\tx@TMRestore}%
313 \box\pst@hbox}%
314 \endgroup}

```

7.8 Closing

Catcodes restoration.

```

315 \catcode'\@=\PstAtCode\relax
</pst-3d>

```

Change History

v0.90		xkey instead of the old pst-key
General: First public release. (tvz)	1	package; creating a dtx file; new
v1.00		L ^A T _E X wrapper file (hv) 1
General: using the extended pst-		

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.

Symbols	\ignorespaces 144
\@pstrickserr 260	
I	P
\if@star 124	\psk@embedangle <u>90</u> , 136
	\psk@normal <u>86</u> , 135

<code>\psk@Tshadowangle</code>	<u>96</u> , 294	<code>\pst@makesmall</code>	125
<code>\psk@viewangle</code>	<u>83</u> , 138	<code>\pst@number</code>	80, 132–134
<code>\psk@viewpoint</code>	<u>70</u> , 137	<code>\pst@object</code>	117, 288
<code>\psset@viewpoint</code>	72, 74, 87	<code>\pst@par</code>	117
<code>\pssetxlength</code>	76, 126	<code>\pst@quadrant</code> ..	195, 211, 236, 238, 239, 253, 255, 257, 265, 267, 269, 277, 279, 280, 300, 302, 303
<code>\pssetylength</code>	70, 77, 127	<code>\pst@sin</code> 197, 201, 214, 215, 217, 218, 223–225, 228, 229, 237, 255, 257, 259, 261, 264, 266, 276, 301	
<code>\pssetzlength</code>	70, 78, 128	<code>\pst@sinandcos</code>	<u>145</u> , 200
<code>\psshadow</code>	<u>288</u>	<code>\pst@starbox</code>	124
<code>\psshadow@i</code>	289	<code>\pst@tempa</code>	151, 157
<code>\psshadow@ii</code>	289, 290	<code>\pst@tempg</code> ..	73, 79, 88, 187, 192, 194
<code>\pst@@divide</code>	254, 256, 264, 266	<code>\pst@usecolor</code>	308
<code>\pst@checknum</code>	94	<code>\pst@Verb</code>	131, 142, 234, 244, 274, 285, 297, 308, 310, 312
<code>\pst@cntb</code>	152, 159, 160, 166	<code>\psTilt</code>	<u>247</u>
<code>\pst@cntc</code>	153, 161, 162, 164	<code>\pstilt</code>	<u>205</u>
<code>\pst@cntg</code>	154, 158, 168, 169, 177, 190, 191, 195	<code>\psTilt@</code>	247, 248
<code>\pst@cos</code>	198, 202, 212, 213, 219, 220, 222, 230, 231, 236, 254, 256, 262, 265, 267, 276, 299	<code>\pstilt@</code>	205, 206
<code>\pst@def</code>	17, 46, 53, 102, 105, 108	<code>\psTshadowcolor</code>	<u>99</u> , 308, 311
<code>\pst@dima</code>	76, 80, 126, 132, 147–149, 151, 156– 159, 161, 164, 213, 218, 223, 229, 245, 255, 257, 265, 267, 286	<code>\psTshadowsize</code>	<u>93</u> , 299, 301
<code>\pst@dimb</code>	77, 80, 127, 133, 149, 161, 168, 170	<code>\PSTthreeDLoaded</code>	16
<code>\pst@dimc</code>	78, 80, 128, 134, 150, 164, 169, 171	<code>\PSTXKeyLoaded</code>	11
<code>\pst@dimg</code>	170, 174, 175, 179, 180, 183, 184, 188, 189, 196, 200, 201, 224, 226, 230, 232, 255, 257, 265, 267, 270, 272	S	
<code>\pst@dimh</code>	171, 176, 180, 183, 184, 186, 188, 189, 202	scaling factor	4
<code>\pst@dimtonum</code>	151, 201, 202	shadowsize	4
<code>\pst@expandafter</code>	72, 87	T	
<code>\pst@getangle</code>	84, 91, 97	<code>\ThreeDput</code>	<u>117</u>
<code>\pst@getcolor</code>	100	<code>\ThreeDput@i</code>	118
<code>\pst@getsinandcos</code> ..	<u>173</u> , 209, 251, 294	<code>\ThreeDput@ii</code>	118, 119
<code>\pst@hbox</code>	125, 141, 212– 215, 217–220, 222–226, 228– 232, 243, 254, 256, 264, 266, 270–272, 284, 296, 309, 311, 313	<code>\ThreeDput@iii</code>	120, 121
<code>\pst@killglue</code>	120	<code>\tx@Div</code>	278
<code>\pst@makebox</code>	120, 205, 247, 289	<code>\tx@ProjThreeD</code>	<u>46</u>
		<code>\tx@SetMatrixEmbed</code>	<u>53</u> , 139
		<code>\tx@SetMatrixThreeD</code>	<u>17</u> , 54, 57
		<code>\tx@STV</code>	112
		<code>\tx@TMChange</code> ..	<u>108</u> , 140, 242, 283, 306
		<code>\tx@TMRestore</code> ..	<u>105</u> , 142, 244, 285, 312
		<code>\tx@TMSave</code>	<u>102</u> , 109
		U	
		<code>\use@par</code>	123, 292