

PSTricks

pst-3d
basic three dimension functions
v.1.11

September 8, 2021

Documentation by
Herbert Voß

Package author(s):
Timothy Van Zandt
Herbert Voß

This version of pst-3d uses the extended keyval handling of pst-xkey.

Thanks to:

Contents

1	PostScript	4
2	Keywords	6
2.1	viewpoint	6
2.2	viewangle	7
2.3	normal	7
2.4	embedangle	7
3	Transformation matrix	7
4	Macros	8
4.1	\ThreeDput	8
5	Arithmetic	8
6	Tilting	11
7	Affin Transformations	13
8	List of all optional arguments for pst-3d	15
	References	15

1 PostScript functions SetMatrixThreeD, ProjThreeD, and SetMatrixEmbed

The 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 basis 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 $v_1 \ v_2 \ v_3 \ \gamma$ 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 $[\tilde{x} \tilde{z}]$.

```

/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
} def

```

The syntax of ProjThreeD is $x \ y \ z \ \text{ProjThreeD} \ x' \ y'$ where $x' = \langle x, y, z \rangle \cdot \tilde{x}$ and $y' = \langle x, y, z \rangle \cdot \tilde{z}$.

```

/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
} def

```

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:

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

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

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

Hence, the transformation matrix is:

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

The syntax of SetMatrixEmbed is $x_0 \ y_0 \ z_0 \ \hat{v}_1 \ \hat{v}_2 \ \hat{v}_3 \ \hat{v}_1 \ \hat{v}_2 \ \hat{v}_3 \ \gamma$ SetMatrixEmbed

SetMatrixEmbed first sets $\langle x_1 \ x_2 \ x_3 \ y_1 \ y_2 \ y_3 \rangle$ 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.

```
/SetMatrixEmbed {
  SetMatrixThreeD
  Matrix3D aload pop
  /z3 ED /z2 ED /z1 ED /x3 ED /x2 ED /x1 ED
  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
} def
```

2 Keywords

2.1 viewpoint

```
\let\pssetlength\pssetlength
\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{%
\beginpgroup
\pssetxlength\pst@dima{#1}%
\pssetylength\pst@dimb{#2}%
\pssetzlength\pst@dimc{#3}%
\edef\pst@tempg{%
\pst@number\pst@dima \pst@number\pst@dimb \pst@number\pst@dimc}%
\endpgroup}
\psset[pst-3d]{viewpoint=1 -1 1}

```

2.2 viewangle

```

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

```

2.3 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}

```

2.4 embedangle

```

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

```

3 Transformation matrix

```

/TMSave {
tx@Dict /TMatrix known not { /TMatrix { } def /RAngle { 0 } def } if end
/TMatrix [ TMatrix CM ] cvx def
} def
/TMRestore { CP /TMatrix [ TMatrix setmatrix ] cvx def moveto } def
/TMChange {
TMSave
/cp [ currentpoint ] cvx def % Check this later.CM def

```

Set standard coord. system, with pt units and origin at currentpoint. This lets 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 STV
```

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

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

Now modify transformation matrix:

```
exch exec
```

Now apply $M \text{ Inv}(M')$

```
concat cp moveto
```

4 Macros

4.1 \ThreeDput

```
\def\ThreeDput{\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}
```

5 Arithmetic

\pst@divide This is adapted from Donald Arseneau's `shapepar.sty`. Syntax:

```
\pst@divide{<numerator>}{<denominator>}{<command>}
\pst@@divide{<numerator>}{<denominator>}
```

<numerator> and <denominator> should be dimensions. \pst@divide sets <command> to <num>/<den> (in points). \pst@@divide sets \pst@dimg to <num>/<den>.

```
\def\pst@divide#1#2#3{%
  \pst@@divide{#1}{#2}%
  \pst@dimtonum\pst@dimg{#3}}
\def\pst@@divide#1#2{%
  \pst@dimg=#1\relax
```



```
\pst@dimh=#2\relax
\pst@cntg=\pst@dimh
\pst@cnth=67108863
\pst@@@divide\pst@@@divide\pst@@@divide\pst@@@divide
\divide\pst@dimg\pst@cntg}
```

The number 16 is the level of uncertainty. Use a lower power of 2 for more accuracy (2 is most precise). But if you change it, you must change the repetitions of \pst@@@divide in \pst@@@divide above:

$$\text{precision}^{\text{repetitions}} = 65536$$

(E.g., $16^4 = 65536$).

```
\def\pst@@@divide{%
\ifnum
\ifnum\pst@dimg<\z@-\fi\pst@dimg<\pst@cnth
\multiply\pst@dimg\sixt@@n
\else
\divide\pst@cntg\sixt@@n
\fi}
```

\pst@pyth Syntax:

\pst@pyth{<dim1>}{<dim2>}{<dimen register>}

<dimen register> is set to $((\text{dim1})^2 + (\text{dim2})^2)^{1/2}$.

The algorithm is copied from PiCTeX, by Michael Wichura (with permission). Here is his description:

Suppose $x > 0$, $y > 0$. Put $s = x + y$. Let $z = (x^2 + y^2)^{1/2}$. Then $z = s \times f$, where

$$f = (t^2 + (1 - t)^2)^{1/2} = ((1 + \tau^2)/2)^{1/2}$$

and $t = x/s$ and $\tau = 2(t - 1/2)$.

```
\def\pst@pyth#1#2#3{%
\beginpgroup
\pst@dima=#1\relax
\ifnum\pst@dima<\z@\pst@dima=-\pst@dima\fi % dima=abs(x)
\pst@dimb=#2\relax
\ifnum\pst@dimb<\z@\pst@dimb=-\pst@dimb\fi % dimb=abs(y)
\advance\pst@dimb\pst@dima % dimb=s=abs(x)+abs(y)
\ifnum\pst@dimb=\z@
\global\pst@dimg=\z@ % dimg=z=sqrt(x^2+y^2)
\else
\multiply\pst@dima 8\relax % dima= 8abs(x)
\pst@@divide\pst@dima\pst@dimb % dimg =8t=8abs(x)/s
\advance\pst@dimg -4pt % dimg = 4tau = (8t-4)
\multiply\pst@dimg 2
\pst@dimtonum\pst@dimg\pst@tempa
\pst@dima=\pst@tempa\pst@dimg % dima=(8tau)^2
\advance\pst@dima 64pt % dima=u=[64+(8tau)^2]/2
\divide\pst@dima 2\relax % =(8f)^2
\pst@dimd=7pt % initial guess at sqrt(u)
\pst@@pyth\pst@@pyth\pst@@pyth % dimd=sqrt(u)
\pst@dimtonum\pst@dimd\pst@tempa
\pst@dimg=\pst@tempa\pst@dimb
```

```

\global\divide\pst@dimg 8 % dimg=z=(8f)*s/8
\fi
\endgroup
#3=\pst@dimg}
\def\pst@pyth{%
\pst@divide\pst@dima\pst@dimd
\advance\pst@dimd\pst@dimg
\divide\pst@dimd 2\relax}

```

\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}

```

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@dimaj}%
\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@dimaj=\fi\pst@sin\pst@dimg
\pst@@divide{\ht\pst@hbox}{\pst@cos\p@}%
\ifnum\pst@quadrant=\@ne\kern\else\pst@dimaj=\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@dimaj=\fi\pst@cos\pst@dimg
\pst@@divide{\ht\pst@hbox}{\pst@sin\p@}%
\ifnum\pst@quadrant=\tw@\kern\else\pst@dimaj=\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@dimaj}%

```

```
\endgroup}
```

```
\psset@Tshadowsize,\psTshadowsize
```

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

```
\psset@Tshadowangle,\psk@Tshadowangle
```

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

```
\psset@Tshadowcolor,\psTshadowcolor
```

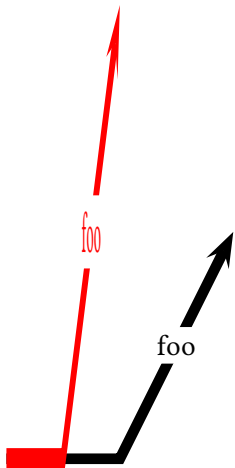
```
\define@key[psset]{pst-3d}{Tshadowcolor}{%
  \pst@getcolor{#1}\psTshadowcolor}
\psset[pst-3d]{Tshadowcolor=lightgray}
```

```
\psshadow
```

```
\def\psshadow{\def\pst@par{}\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@{\@nameuse{\psTshadowcolor}\copy\pst@hbox\hss}}%
      \pst@Verb{\tx@TMRestore}%
      \box\pst@hbox}%
    \endgroup}
```

7 Affin Transformations

```
\psAffinTransform [Options] {transformation matrix}{object}
```

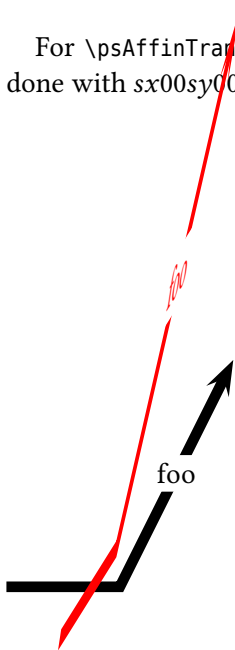


```
\pspicture(3,6)\psset{linewidth=4pt,arrows=->}
\psline(0,0)(1.5,0)(3,3)\rput*(2.25,1.5){foo}
\psAffinTransform{0.5 0 0 2 0 0}{\color{red}%
  \psline[linecolor=red](0,0)(1.5,0)(3,3)\rput*(2.25,1.5){foo}}}%
\endpspicture
```

The transformation matrix must be a list of 6 values divided by a space. For a translation modify the last two values of $1001dx dy$. The values for dx and dy must be of the unit pt! For a rotation we have the transformation matrix

$$\begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (1)$$

For `\psAffinTransform` the four values have to be modified as $\cos \alpha \sin \alpha -\sin \alpha \cos \alpha 0 0$. Tilting can be done with $sx 0 0 sy 0 0$. All effects can be combined.



```
\pspicture(3,6)\psset{linewidth=4pt,arrows=->}
\psline(0,0)(1.5,0)(3,3)\rput*(2.25,1.5){foo}
\psAffinTransform{0.5 0.8 0.3 2 20 -20}{\color{red}%
  \psline[linecolor=red](0,0)(1.5,0)(3,3)\rput*(2.25,1.5){foo}}}%
\endpspicture
```

8 List of all optional arguments for pst-3d

Key	Type	Default
viewpoint	ordinary	1 -1 1
viewangle	ordinary	0
normal	ordinary	0 0 1
embedangle	ordinary	0
Tshadowsize	ordinary	1
Tshadowangle	ordinary	60
Tshadowcolor	ordinary	lightgray

References

- [1] Michel Goossens et al. *The L^AT_EX Graphics Companion*. 2nd ed. Reading, Mass.: Addison-Wesley Publishing Company, 2007.
- [2] Laura E. Jackson and Herbert Voß. “Die Plot-Funktionen von pst-plot”. In: *Die T_EXnische Komödie* 2/02 (June 2002), pp. 27–34.
- [3] Nikolai G. Kollock. *PostScript richtig eingesetzt: vom Konzept zum praktischen Einsatz*. Vaterstetten: IWT, 1989.
- [4] Manuel Luque. *Vue en 3D*. <http://members.aol.com/Mluque5130/vue3d16112002.zip>, 2002.
- [5] Herbert Voß. “Die mathematischen Funktionen von Postscript”. In: *Die T_EXnische Komödie* 1/02 (Mar. 2002), pp. 40–47.
- [6] Herbert Voß. *PSTricks – Grafik für T_EX und L^AT_EX*. 4. Heidelberg/Hamburg: DANTE – Lehmanns, 2007.
- [7] Herbert Voß. *L^AT_EX Referenz*. 1. Heidelberg/Hamburg: DANTE – Lehmanns, 2007.
- [8] Herbert Voss. *PSTricks Support for pdf*. <http://PSTricks.de/pdf/pdfoutput.phtml>, 2002.
- [9] Michael Wiedmann and Peter Karp. *References for T_EX and Friends*. <http://www.miwie.org/tex-refs/>, 2003.
- [10] Timothy Van Zandt. *PSTricks - PostScript macros for Generic TeX*. <http://www.tug.org/application/PSTricks>, 1993.

Index

C

currentpoint, 7

M

Macro

- \psAffinTransform, 13f

Matrix3D, 5

P

Package

- pst-3d, 2

- pst-xkey, 2

PostScript

- Matrix3D, 5

- ProjThreeD, 4f

- SetMatrixEmbed, 4, 6

- SetMatrixThreeD, 4

ProjThreeD, 4f

\psAffinTransform, 13f

pst-3d, 2

pst-xkey, 2

S

SetMatrixEmbed, 4, 6

SetMatrixThreeD, 4

V

viewpoint, 4