pst-solides3d : guide de l'utilisateur

v. 4.10 (2008/07/24)

Jean-Paul VIGNAULT, Manuel LUQUE, Arnaud SCHMITTBUHL
<jpv@melusine.eu.org>, <manuel.luque27@gmail.com>, <aschmittbuhl@libertysurf.fr> [1]

15 juillet 2008

[1] Avec la collaboration de : Jürgen GILG<gilg@acrotex.net>, Jean-Michel SARLAT<jm.sarlat@gmail.com>, Herbert VOSS<Herbert.Voss@FU-Berlin.DE>.

Table des matières

0.1 Constitution du package – Distribution

- Fichiers `pst-solides3d.sty`, `pst-solides3d.tex` et `solides.pro`.
- Documentation et exemples : `pst-solides3d-doc.tex(pdf)`.

Ce package est disponible à l'url http://syracuse.eu.org/syracuse/pstricks/pst-solides3d/
De nombreux exemples sont publiés ici : http://syracuse.eu.org/lab/bpst/pst-solides3d
Et enfin, la version de développement est disponible sur le svn de mélusine : http://syracuse-dev.org/pst-solides3d

0.2 Avant-propos

Le package présenté dans ce document est issu d'un travail collaboratif initié sur la liste de diffusion du site syracuse (http://melusine.eu.org/syracuse).

L'idée est née de la confrontation des travaux de Jean-Paul Vignault sur le logiciel jps2ps[1] avec ceux de Manuel Luque sur PSTricks[2], en particulier dans le domaine de la représentation des solides dans une scène en 3d.

Les deux auteurs ont décidé d'unifier leurs efforts dans l'écriture d'un package PSTricks dédié à la représentation de scènes 3d. Le travail s'effectue sur la machine *melusine*, dans un environnement informatique préparé et maintenu par Jean-Michel Sarlat.

L'équipe s'est ensuite étoffée avec l'arrivée d'Arnaud Schmittbuhl et de Jürgen Gilg, ce dernier s'étant spécialisé dans le beta-test à base d'animations[3].

0.3 Présentation

Le package `pst-solides3d` permet d'obtenir, avec PSTricks, des vues en 3d de solides prédéfinis ou construits par l'utilisateur. On trouvera la plupart des solides usuels que l'on peut représenter avec ou sans les arêtes cachées et dont la couleur peut varier avec l'éclairage.

Ce package permet également de projeter des textes ou des dessins simples en 2d sur un plan quelconque ou sur une face d'un solide déjà construit.

Du point de vue utilisateur, la plupart des fonctionnalités sont accessibles par trois macros TEX : `\psSolid`, qui sert à manipuler les objets à 3 dimensions, `\psSurface`, cousine de la première et dédiée à la représentation de surfaces définies par une équation du type $f(x,y)=z$, et `\psProjection` qui permet de projeter un dessin en 2 dimensions sur un plan quelconque de la scène 3d représentée.

Dans l'utilisation, deux langages cohabitent : d'une part PSTricks et ses macros où l'utilisateur retrouvera la syntaxe usuelle, d'autre part Postscript que l'on voit apparaître dans les argument optionnels des précédentes.

Le parti pris a été de limiter strictement le champ d'action de PSTricks, pour le cantonner au rôle d'interface entre TEX et Postscript. Plus précisément, le rôle de PSTricks a strictement été circonscrit à celui de la transmission des paramètres vers Postscript, ce dernier s'occupant de la totalité des calculs nécessaires puis de l'affichage.

Pour l'ensemble de ces procédures de calculs et d'affichages, nous utilisons une librairie Postscript développée pour une autre application (le logiciel *jps2ps*). Le code postscript utilisant cette librairie est appelé *code jps*.

Le but de ce présent document est de décrire la syntaxe PSTricks pour chacune des opérations offertes par le package.

0.4 Changements par rapport aux versions précédentes

0.4.1 Changements par rapport à la version 3.0

- La macro de projection est complètement remise à plat, et on perd la compatibilité. Il faut maintenant utiliser un objet de *type plan* pour définir une projection.
- L'objet `courbe` utilise maintenant l'argument r. Pour retrouver le comportement précédent, il faut spécifier $r=0$.
- L'option `resolution` de l'objet `courbe` est remplacée par l'option `ngrid`
- Suppression des arguments `tracelignedeniveau` et associés.

[1] http://melusine.eu.org/syracuse/bbgraf/
[2] http://melusine.eu.org/syracuse/pstricks/pst-v3d/
[3] http://melusine.eu.org/syracuse/pstricks/pst-solides3d/animations/

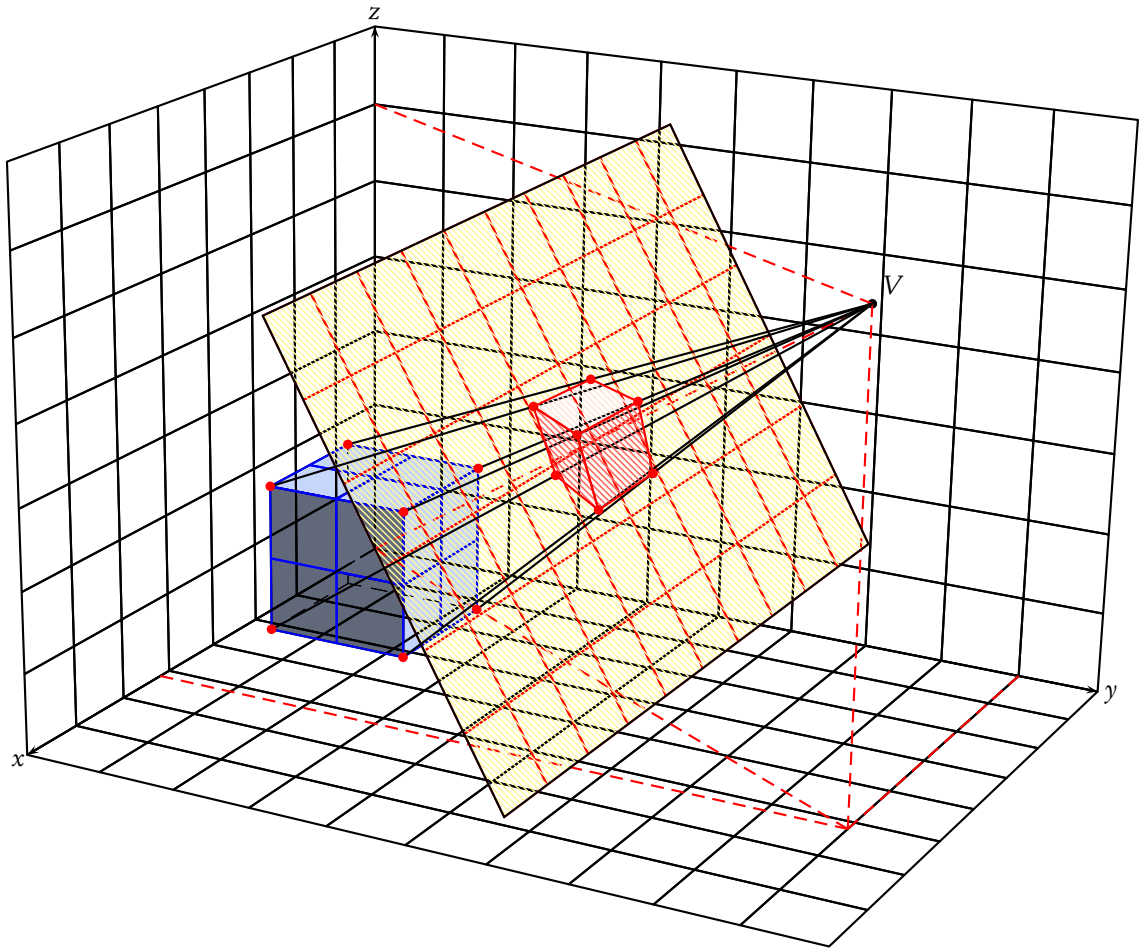
0.4.2 Changements par rapport à la version 2.0

- L'option `hue` n'est plus un booléen.
- La gestion de l'échelle en postscript se fait désormais avec le mécanisme jps. Pour se déplacer à l'échelle, les commandes `smoveto`, `srmoveto`, `slineto`, `srlineto` remplacent les commandes respectives `moveto`, `rmoveto`, `lineto`, `rlineto`.

Chapitre 1

Commandes de bases

1.1 Choix du point de vue



Les coordonnées de l'objet, ici le cube bleuté, sont données dans le repère $Oxyz$. Les coordonnées du point de vue (V), sont données dans ce même repère, soit en coordonnées cartésiennes qui est l'option par défaut, soit en coordonnées sphériques en ajoutant l'opérateur `[rtp2xyz]`,

Exemple : `[viewpoint=50 30 20 rtp2xyz]`

L'écran est placé perpendiculairement à la direction OV, à une distance de V : `[Decran=50]` (valeur par défaut), cette valeur peut être positive ou négative.

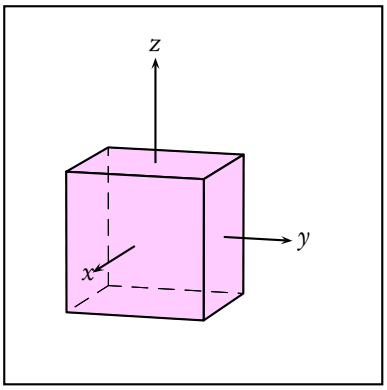
1.2 Les axes en 3D

la commande `\axesIIID[options](x1,y1,z1)(x2,y2,z2)` trace les axes Ox, Oy et Oz en pointillés de O respectivement, jusqu'au point de coordonnées $(x_1,0,0)$ pour l'axe des x, $(0,y_1,0)$ pour l'axe des y et $(0,0,z_1)$ pour l'axe des z et ensuite en trait continu jusqu'aux points $(x_2,0,0)$, $(0,y_2,0)$ et $(0,0,z_2)$.

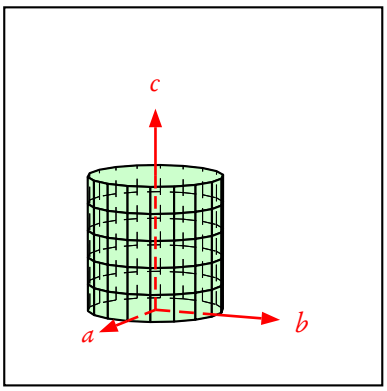
Les options sont les suivantes :

- toutes les options de couleur, d'épaisseur du trait, ainsi que des caractéristiques des flèches.
- `labelsep=valeur` qui permet de placer à la distance souhaitée de l'extrémité de la flèche, l'étiquette de l'axe, sa valeur par défaut est `labelsep=5pt`, il s'agit de la distance réelle en trois dimensions et non sur l'écran.
- Le choix des étiquettes(*labels*) de chaque axe avec l'option `axisnames=a,b,c`, avec par défaut `axisnames=x,y,z`.
- La possibilité de spécifier, le style de ces étiquettes avec l'option : `axisemph=`, par défaut il n'y a pas de style prédéfini, c'est-à-dire que si l'on ne précise rien on aura x,y,z.
- `showOrigin` est un booléen, `true` par défaut, s'il est positionné à `showOrigin=false` les pointillés ne seront plus tracés depuis l'origine.
- `mathLabel` est un booléen, `true` par défaut, qui dans ce cas écrit les étiquettes en mode mathématique, positionné à `mathLabel=false` on passe dans le mode usuel.

Les étiquettes sont placées aux extrémités des axes dans leur prolongement.




```
\begin{pspicture}(-2,-2)(3,3)
\psset{viewpoint=100 30 20,Decran=100}
\psframe(-2,-2)(3,3)
\psSolid[object=cube,a=2,
        action=draw*,
        fillcolor=magenta!20]
\axesIIID[showOrigin=false](1,1,1)(3,2,2.5)
\end{pspicture}
```



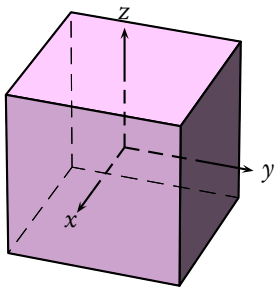
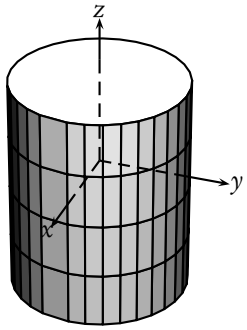
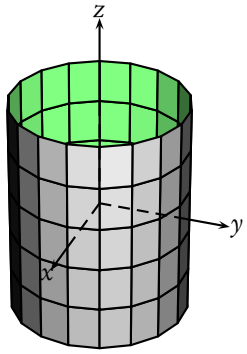

```
\begin{pspicture}(-2,-1)(3,4)
\psset{viewpoint=100 45 20,Decran=100}
\psframe(-2,-1)(3,4)
\psSolid[object=cylindre,h=2,r=1,
        action=draw*,mode=4,
        fillcolor=green!20]
\axesIIID[linewidth=1pt,linecolor=red,arrowsize=5pt,
          arrowinset=0,axisnames={a,b,c},
          axisemph={\boldmath\Large\color{red}},
          labelsep=10pt]
         (1,1,2)(2,2,3)
\end{pspicture}
```

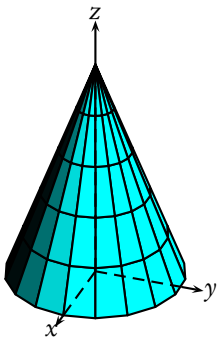
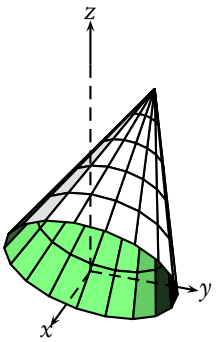
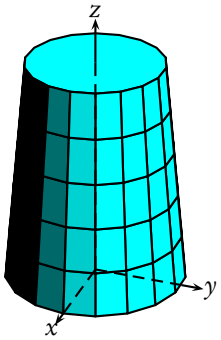
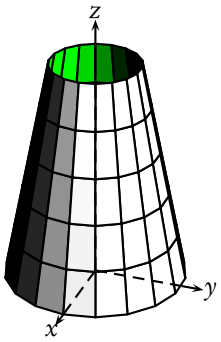
1.3 Les principaux solides prédéfinis et leurs paramètres

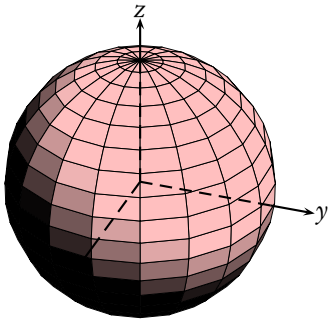
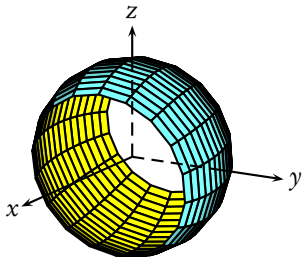
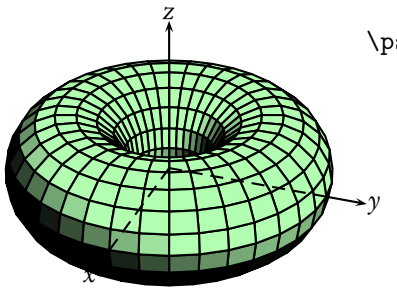
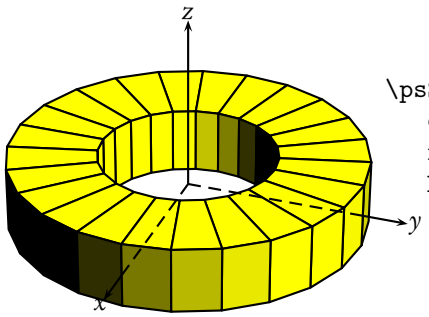
La commande de base est : `\psSolid[object=nom](x,y,z)` qui permet de tracer l'objet désigné par *nom* au point de coordonnées (x,y,z).

Les objets disponibles sont : `cube, cylindre, cylindrecreux, cone, conecreux, tronccone, troncconecreux, sphere, calottesphere, tore, anneau, tetrahedron, octahedron, dodecahedron, isocahedron, prisme, grille, parallelepiped, face, ruban, surface, plan, geode,vecteur.`

Le tableau ci-dessous donne un exemple de chacun des solides avec ses paramètres propres :

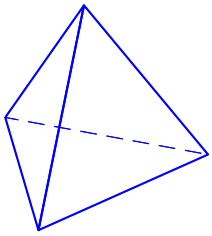
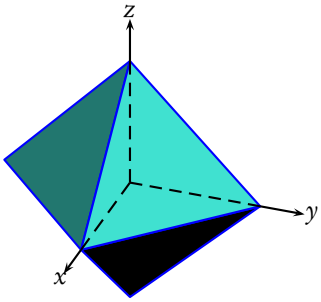
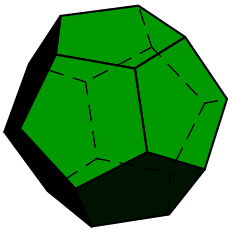
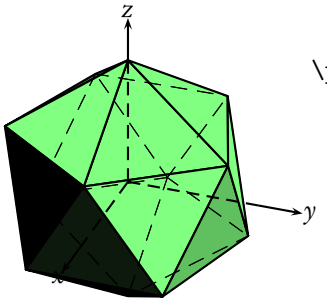
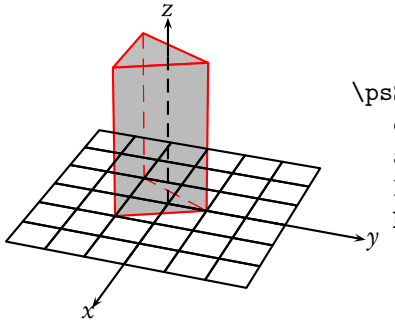
solide	paramètres par défaut	vue	code
cube	[a=4] arête		`\psSolid[ object=cube, a=2, action=draw*, fillcolor=magenta!20]`
cylindre	[h=6,r=2] hauteur et rayon le maillage : [ngrid=n1 n2]		`\psSolid[ object=cylindre, h=5,r=2, fillcolor=white, ngrid=4 32](0,0,-3)`
cylindre creux	[h=6,r=2] hauteur et rayon le maillage : [ngrid=n1 n2]		`\psSolid[ object=cylindrecreux, h=5,r=2, fillcolor=white, mode=4, incolor=green!50] (0,0,-3)`

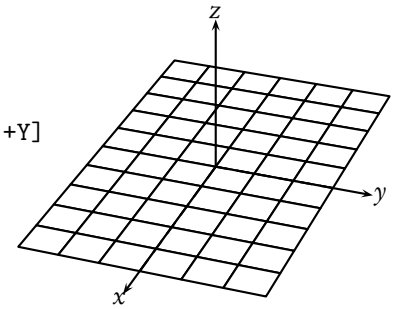
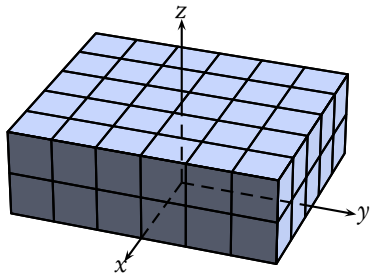
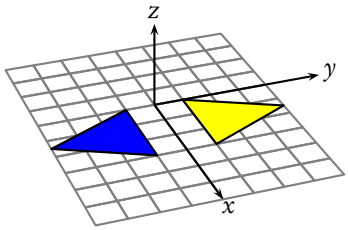
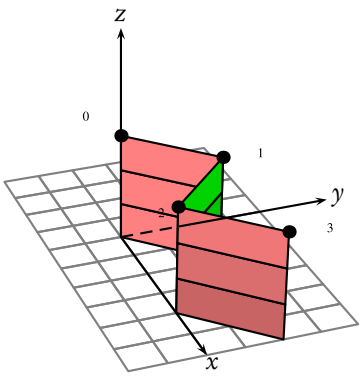
solide	paramètres par défaut	vue	code
cone	[h=6,r=2] hauteur et rayon le maillage : [ngrid=n1 n2]		\psSolid[object=cone, h=5,r=2, fillcolor=cyan, mode=4]%
cone creux	[h=6,r=2] hauteur et rayon le maillage : [ngrid=n1 n2]		\psSolid[object=conecreux, h=5,r=2, RotY=-60, fillcolor=white, incolor=green!50, mode=4]%
tronc de cone	[h=6,r0=4,r1=1.5] hauteur et rayons le maillage : [ngrid=n1 n2]		\psSolid[object=tronccone, r0=2,r1=1.5,h=5, fillcolor=cyan, mode=4]%
tronc de cone creux	[h=6,r0=4,r1=1.5] hauteur et rayons le maillage : [ngrid=n1 n2]		\psSolid[object=troncconecreux, r0=2,r1=1,h=5, fillcolor=white, mode=4]%

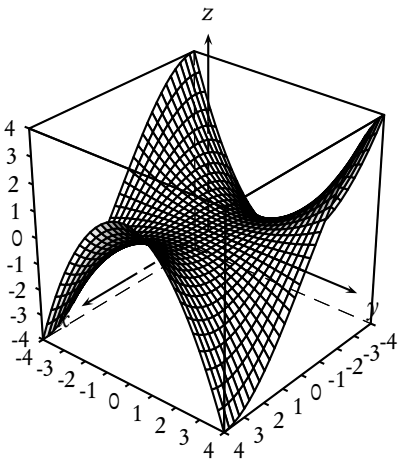
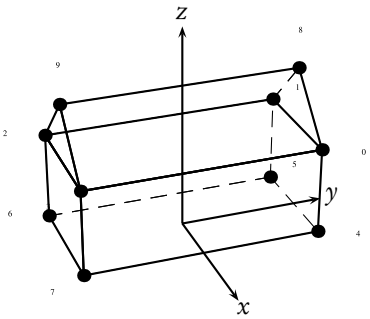
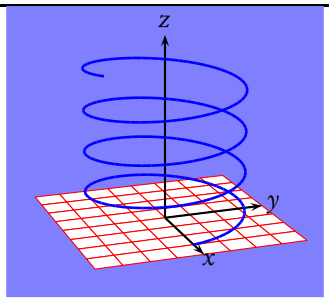
solide	paramètres par défaut	vue	code
sphère	[r=2] rayon le maillage : [ngrid=n1 n2]		\psSolid[object=sphere, r=2,fillcolor=red!25, ngrid=18 18]%
calotte sphérique	[r=2] rayon [phi=0,theta=90] latitudes pour découper la calotte respectivement vers le bas et le haut		\psSolid[object=calottesphere, r=3,ngrid=16 18, theta=45,phi=-30, hollow,RotY=-80]%
tore	[r0=4,r1=1.5] rayon intérieur rayon moyen le maillage : [ngrid=n1 n2]		\psSolid[r1=2.5,r0=1.5, object=tore, ngrid=18 36, fillcolor=green!30, action=draw*]%
anneau cylindrique	[r1=2.5,r0=1.5, h=6,section=...] rayon extérieur rayon intérieur hauteur section rectangulaire		\psSolid[object=anneau, fillcolor=yellow, h=1.5,r1=4,r0=3]%

Une documentation spécifique aux anneaux circulaires et aux parallélépipèdes est fournie dans la partie exemples :

- doc-grille-parallelepiped.tex(.pdf) ;
- doc-anneau.tex(.pdf).

solide	paramètres par défaut	vue	code
tétraèdre	[a=2] rayon de la sphère circonscrite		\psSolid[object=tetrahedron, r=3, linecolor=blue, action=draw]%
octaèdre	[a=2] rayon de la sphère circonscrite		\psSolid[object=octahedron, a=3, linecolor=blue, fillcolor=Turquoise]%
dodécaèdre	[a=2] rayon de la sphère circonscrite		\psSolid[object=dodecahedron, a=2.5,RotZ=90, action=draw*, fillcolor=OliveGreen]%
icosaèdre	[a=2] rayon de la sphère circonscrite		\psSolid[object=icosahedron, a=3, action=draw*, fillcolor=green!50]%
prisme	[axe=0 0 1] direction de l'axe [base= -1 -1 1 -1 0 1] coordonnées des sommets de la base [h=6] hauteur		\psSolid[object=prisme, action=draw*, linecolor=red, h=4]%

solide	paramètres par défaut	vue	code
grille	[base=-X +X -Y +Y]		\psSolid[object=grille, base=-5 5 -3 3]%
parallélépipède	[a=4,b=a,c=a] centre en O		\psSolid[object=parallelepiped,% a=5,b=6,c=2, fillcolor=yellow]% (0,0,c 2 div)
face	[base=x0 y0 x1 y1 x2 y2 etc.] les coordonnées des sommets dans le sens trigo		\psSolid[object=face, fillcolor=yellow, incolor=blue, base=0 0 3 0 1.5 3](0,1,0) \psSolid[object=face, fillcolor=yellow, incolor=blue, base=0 0 3 0 1.5 3, RotX=180](0,-1,0)
ruban	[base=x0 y0 x1 y1 x2 y2 etc.] [h=hauteur] [ngrid=valeur] nombre de mailles verticalement [axe=0 0 1] direction de l'inclinaison du ruban		\psSolid[object=ruban,h=3, fillcolor=red!50, base=0 0 2 2 4 0 6 2, num=0 1 2 3, show=0 1 2 3, ngrid=3])

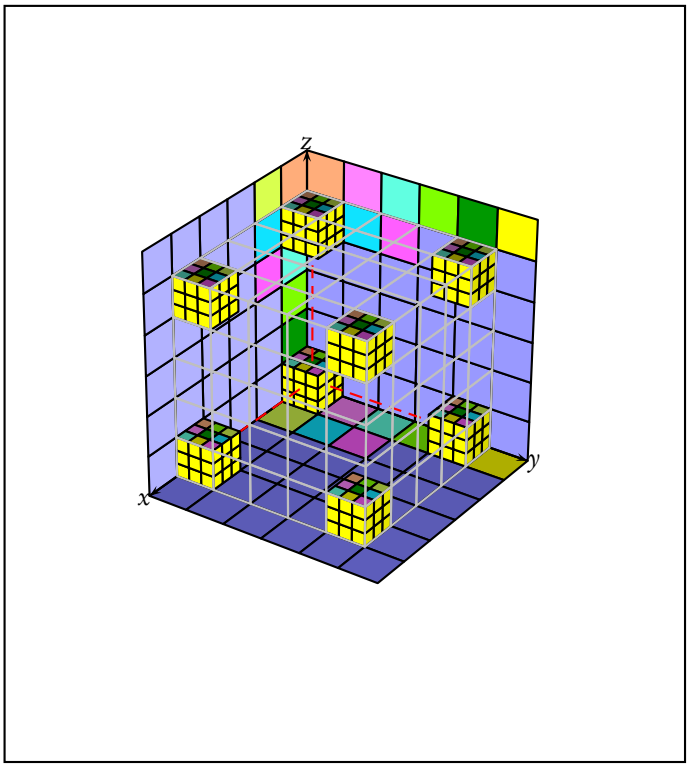
solide	paramètres par défaut	vue	code
surface	voir la documentation spécifique		`\psSurface[ngrid=.25 .25, incolor=Wwhite,axesboxed] (-4,-4)(4,4){% x dup mul y dup mul 3 mul sub x mul 32 div}`
new	solide défini par les coordonnées des sommets et les faces		`\psSolid[object=new, action=draw, sommets= 2 4 3 -2 4 3 -2 -4 3 2 -4 3 2 4 0 -2 4 0 -2 -4 0 2 -4 0 0 4 5 0 -4 5, faces={ [0 1 2 3] [7 6 5 4] [0 3 7 4] [3 9 2] [1 8 0] [8 9 3 0] [9 8 1 2] [6 7 3 2] [2 1 5 6]}]%`
courbe	tracé d'une fonction R -> R^3 définie par ses équations paramétriques		`\defFunction[algebraic]% {helice}(t) {3*cos(4*t)}{3*sin(4*t)}{t} \psSolid[object=courbe,r=0, range=0 6, linecolor=blue,linewidth=0.1, resolution=360, function=helice]%`

1.4 Positionner un solide

1.4.1 Translation

La commande suivante `\psSolid[object=cube,`*options*`](x,y,z)` déplace le centre du cube au point de coordonnées `(x,y,z)`.

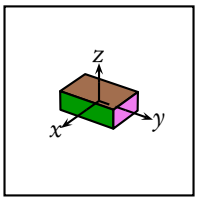
L'exemple suivant va recopier le cube d'arête 1  aux points de coordonnées `(0.5,0.5,0.5)`, `(4.5,0.5,0.5)` etc. afin que ces copies occupent les coins d'un cube d'arête 5.

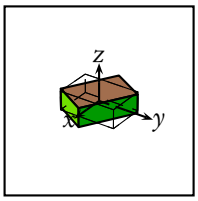



```
\psset{fillcolor=yellow,mode=3}
\psSolid[object=cube](0.5,0.5,0.5)
\psSolid[object=cube](4.5,0.5,0.5)
\psSolid[object=cube](0.5,4.5,0.5)
\psSolid[object=cube](0.5,0.5,4.5)
\psSolid[object=cube](4.5,4.5,4.5)
\psSolid[object=cube](4.5,0.5,4.5)
\psSolid[object=cube](4.5,4.5,0.5)
\psSolid[object=cube](0.5,4.5,4.5)
```

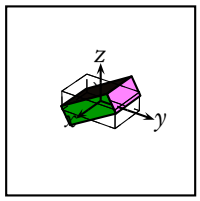
1.4.2 Rotation

La rotation s'effectue dans l'ordre autour des axes Ox, Oy et Oz. Prenons l'exemple d'un parallélépipède rectangle,  que l'on va faire tourner successivement autour des axes Ox, Oy et Oz.

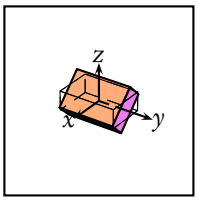




`[RotZ=60]`



`[RotX=30]`



`[RotY=-45]`

Chapitre 2

Les options de \psSolid

2.1 Commandes de tracé

La commande de tracé se fait avec le paramètre `action=` dans la commande `\psSolid`.
Quatres valeurs sont possibles :

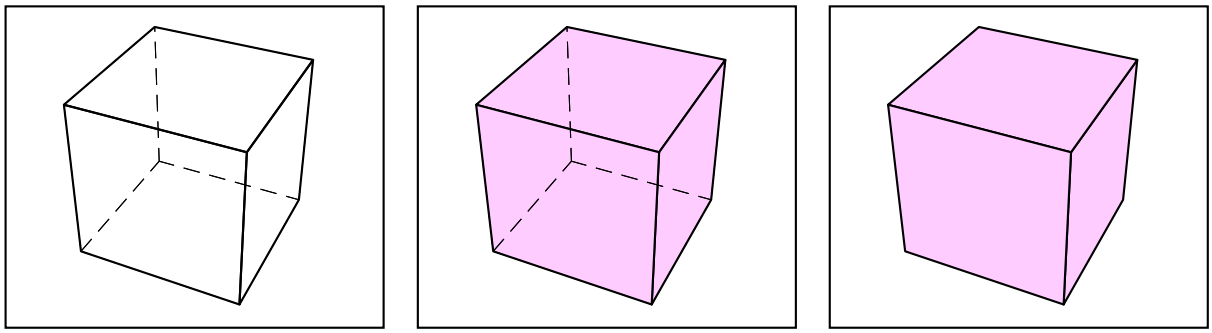
`[action=none]` : ne trace rien

`[action=draw]` : trace le solide en structure fil de fer avec tracé en pointillé des arêtes cachées

`[action=draw*]` : trace le solide avec tracé en pointillé des arêtes cachées et coloration des faces visibles

`[action=draw**]` : trace le solide avec l'algorithme du peintre, sans les arêtes cachées et avec coloration des faces visibles.

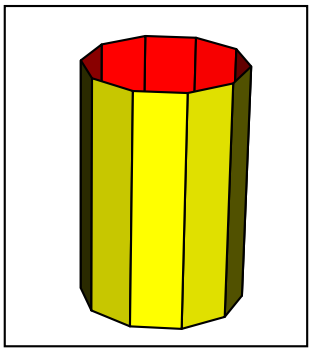
Les commandes `draw` et `draw*` ne sont pertinentes que pour les solides convexes.



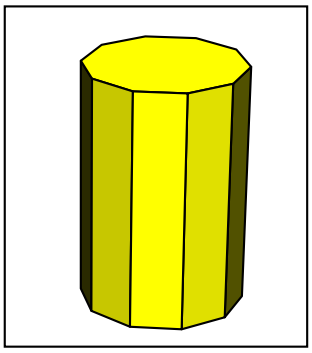
2.2 Évider un solide

Certains des solides prédéfinis ont un solide "*creux*" qui lui est naturellement associé (le cône, le tronc de cône, le cylindre, le prisme et la calotte sphérique). Pour ceux là, une option `[hollow=`*boolean*`]` est prévue. Positionné à *false*, on a le solide habituel ; positionné à *true* on a la version creuse.

Exemple 1 : cylindre et cylindre creux

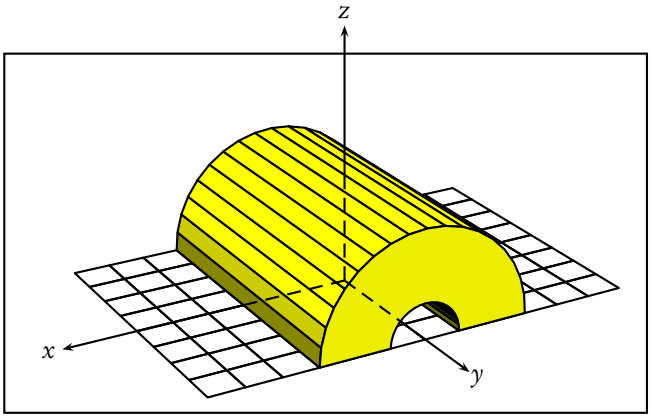


```
\psSolid[object=cylindre,
   h=6,r=2,
   fillcolor=yellow,
   incolor=red,
   hollow,
   ](0,4,0)
```

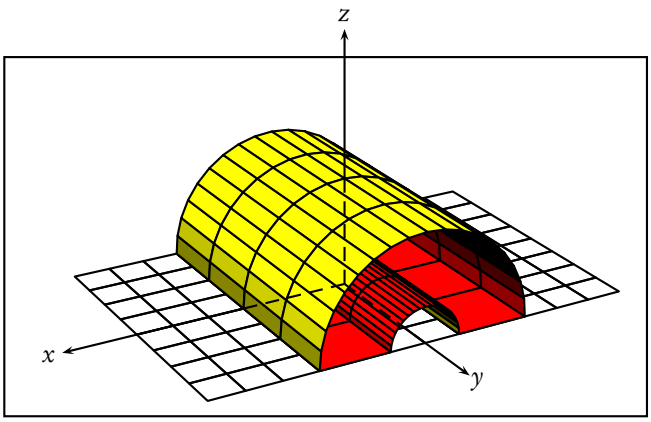


```
\psSolid[object=cylindre,
   h=6,r=2,
   fillcolor=yellow,
   ](0,4,0)
```

Exemple 2 : prisme et prisme creux

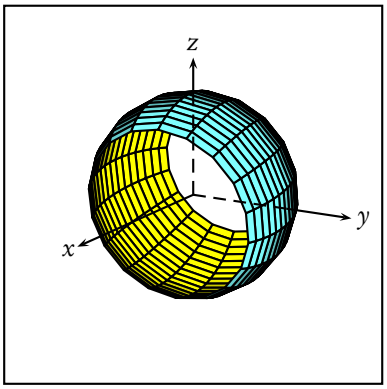



```
\defFunction{F}(t){t cos 3 mul}{t sin 3 mul}{}
\defFunction{G}(t){t cos}{t sin}{}
\psSolid[
   object=grille,base=-6 6 -4 4,action=draw]%
\psSolid[
   object=prisme,h=8,fillcolor=yellow,RotX=90,
   resolution=19,
   base=0 180 {F} CourbeR2+
        180 0 {G} CourbeR2+
   ](0,4,0)
\axesIIID(3,4,3)(8,6,7)
```

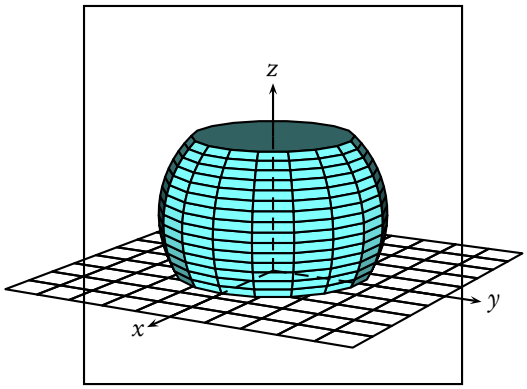



```
\defFunction{F}(t){t cos 3 mul}{t sin 3 mul}{}
\defFunction{G}(t){t cos}{t sin}{}
\psSolid[
   object=grille,base=-6 6 -4 4,action=draw]%
\psSolid[
   object=prisme,h=8,fillcolor=yellow,RotX=90,
   hollow,ngrid=4,incolor=red,
   resolution=19,
   base=0 180 {F} CourbeR2+
        180 0 {G} CourbeR2+
   ](0,4,0)
\axesIIID(3,4,3)(8,6,7)
```

Exemple 3 : calotte sphérique et calotte sphérique creuse




```
\psSolid[object=calottesphere,r=3,
        ngrid=16 18,
        fillcolor=cyan!50,incolor=yellow,
        theta=45,phi=-30,hollow,RotY=-60]%
```



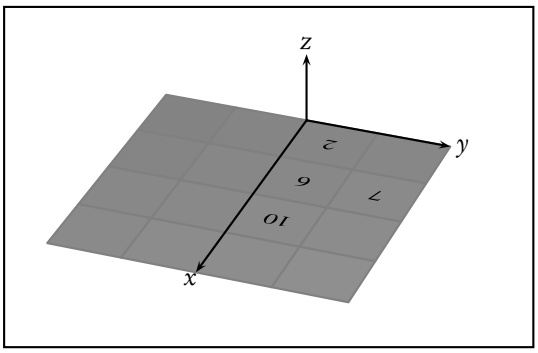

```
\psSolid[object=calottesphere,r=3,
        ngrid=16 18,fillcolor=cyan!50,
        incolor=yellow,theta=45,phi=-30]
        (0,0,1.5)
```

2.3 Numéroter les facettes

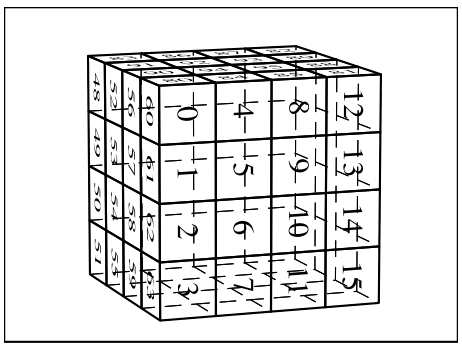
L'option `numfaces` permet d'afficher sur chaque face son indice rorrespondant.

- `[numfaces=all]` affiche tous les numéros de faces ;
- `[numefaces=0 1 2 3]` affiche tous les numéros de faces `[0,1,2 et 3]` ;

L'option `fontsize` permet de fixer la taille de la police utilisée. Enfin, le booléen `visibility` permet de spécifier si on doit ou non afficher le numéro de face si la face n'est pas visible. Par défaut, on a `visibility=true`, et on tient compte de la visibilité (ie. numéro pas affiché si la face n'est pas visible)



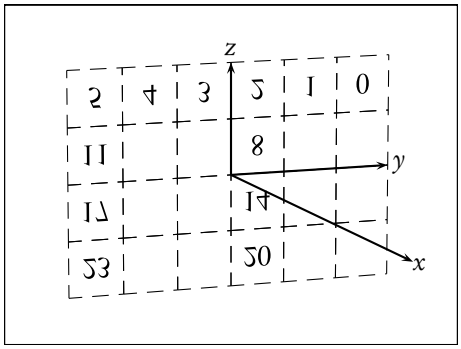

```
\psSolid[object=grille,
        base=0 4 -2 2,
        numfaces=2 6 7 10,
        linecolor=gray](0,0,0)
```



```
\psSolid[object=cube,
        RotY=90,
        ngrid=4,
        fontsize=15,
        action=draw,
        numfaces=all](0,0,0)
```

Les options de `\psSolid` acceptent des commandes postcript, et en particulier les boucles `for`.

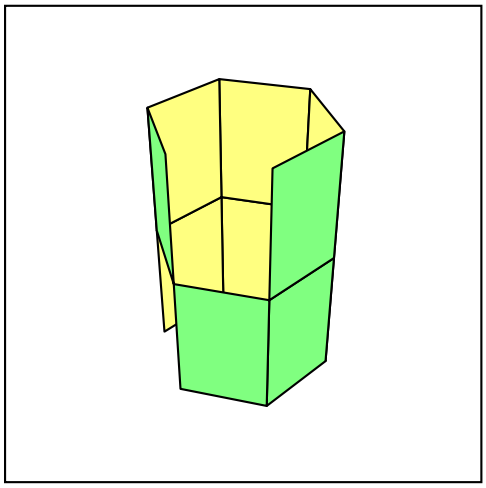
Ainsi l'instruction `[numfaces=0 1 5 {} for]` demande la numérotation de toutes les faces dont l'indice est compris entre 0 et 5. L'instruction `[numfaces=8 3 23 {} for]` demande la numérotation d'une face sur 3 entre les indices 8 et 23.



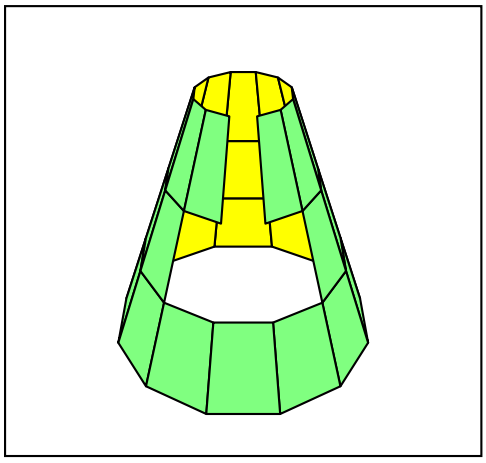

```
\axesIIID(0,0,0)(8,3,2)
\psSolid[object=grille,
         RotY=90,
         RotZ=180,
         ngrid=1.,
         fontsize=15,
         numfaces=
            0 1 5 {} for
            8 3 23 {} for,
         base=-2 2 -3 3,
         visibility=false,
         action=draw](0,0,0)
```

2.4 Enlever des facettes

L'argument `[rm=1 2 8]` permet de supprimer les facettes visibles 1, 2 et 8, afin de voir l'intérieur des solides creux.



```
\psSolid[rm=1 3 6,
         object=cylindrecreux,
         ngrid=2 6,
         h=6,r=2,fillcolor=green!50,
         incolor=yellow!50,RotZ=-60](0,0,-3)
```

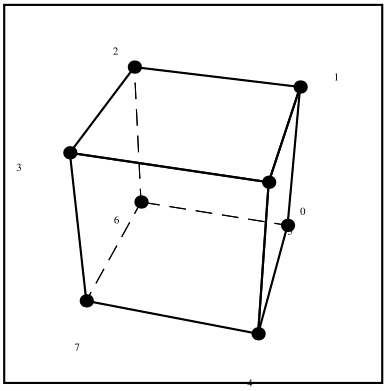


```
\psSolid[object=troncconecreux,
         rm=1 12 13 14,
         r0=3,r1=1,h=6,
         fillcolor=green!50,incolor=yellow,
         mode=3](0,0,-3)
```

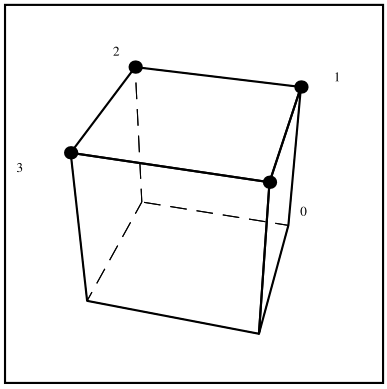
2.5 Pointage et numérotation des sommets

Une option permet de pointer les sommets et/ou de les numéroter soit globalement, soit individuellement.

- `[show=all]` pointe tous les sommets ;
- `[num=all]` numérote tous les sommets ;
- `[show=0 1 2 3]` pointe les sommets `[0,1,2 et 3]` ;
- `[num=0 1 2 3]` numérote les sommets `[0,1,2 et 3]`.




```
\psSolid[action=draw,
         object=cube,RotZ=30,
         show=all,num=all]%
```




```
\psSolid[object=cube,
         RotZ=30,action=draw,
         show=0 1 2 3,
         num=0 1 2 3]%
```

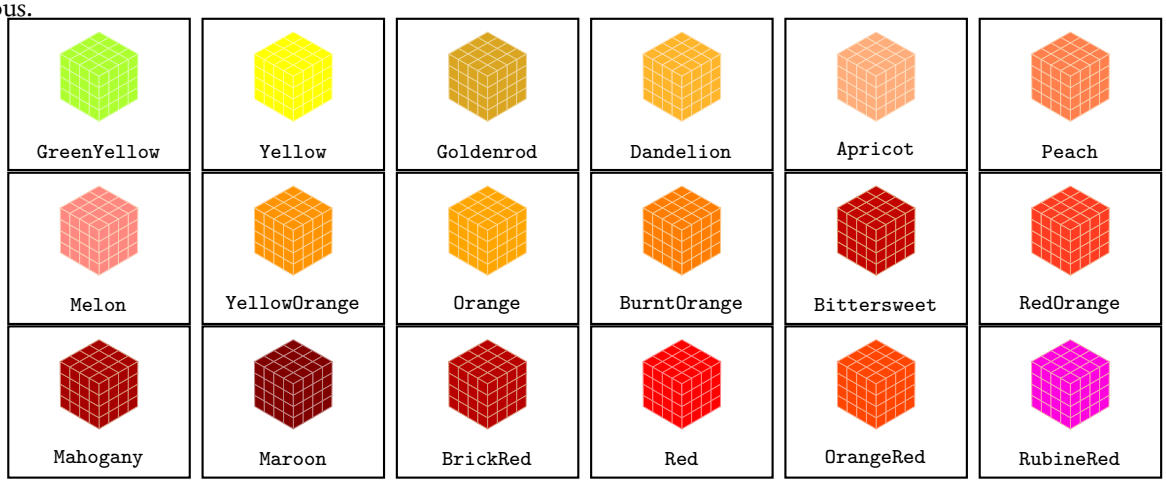
2.6 Les couleurs et les dégradés de couleur

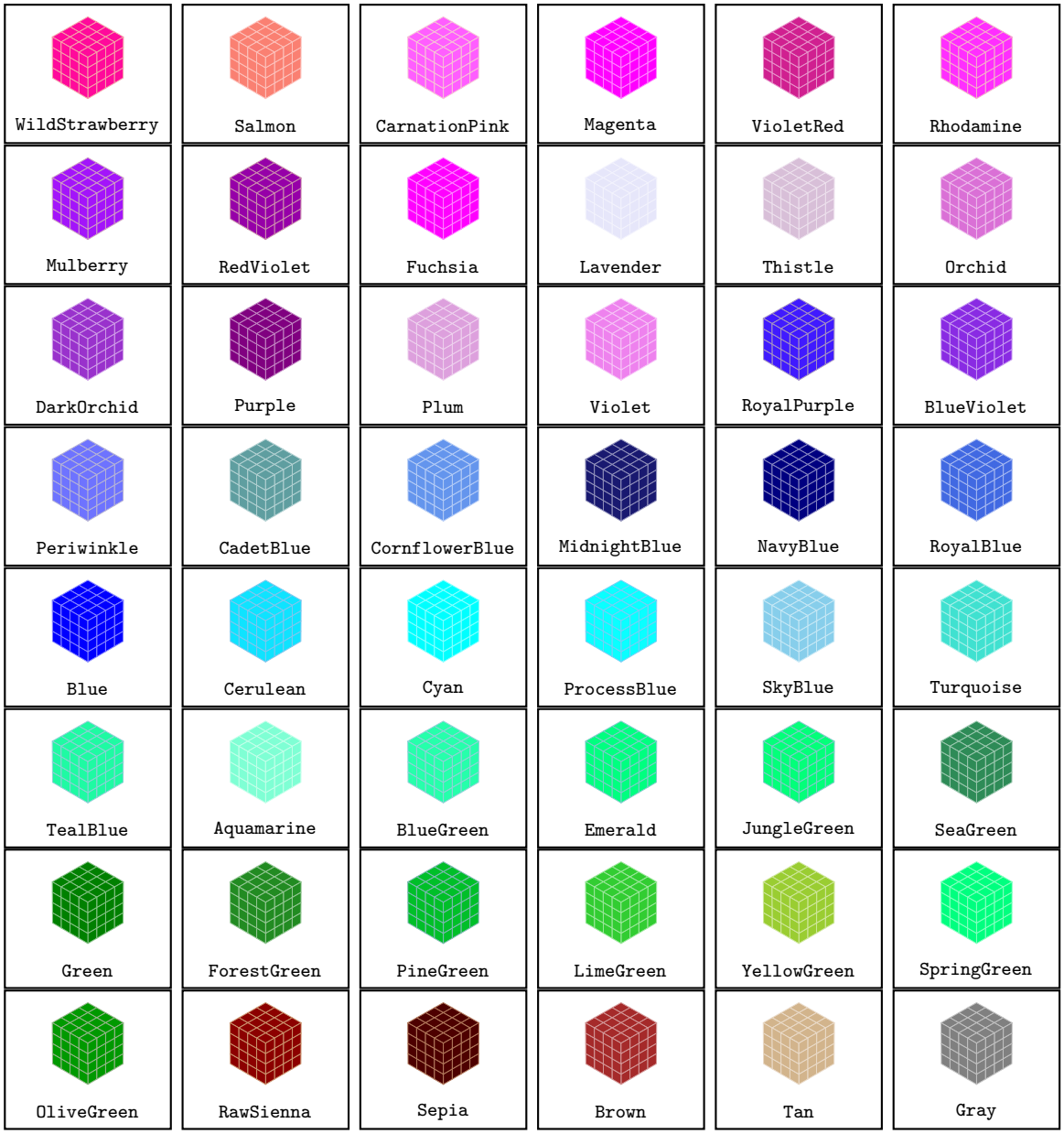
L'argument `[fillcolor=`*name*`]` permet de spécifier la couleur souhaitée pour les faces externes d'un solide. L'argument `[incolor=`*name*`]` permet de spécifier la couleur souhaitée pour les faces internes d'un solide.

Les valeurs possibles pour *name* sont toutes celles reconnues par PSTricks (et en particulier son package `xcolor`).

2.6.1 Couleurs prédéfinies par l'option `[dvipsnames]`

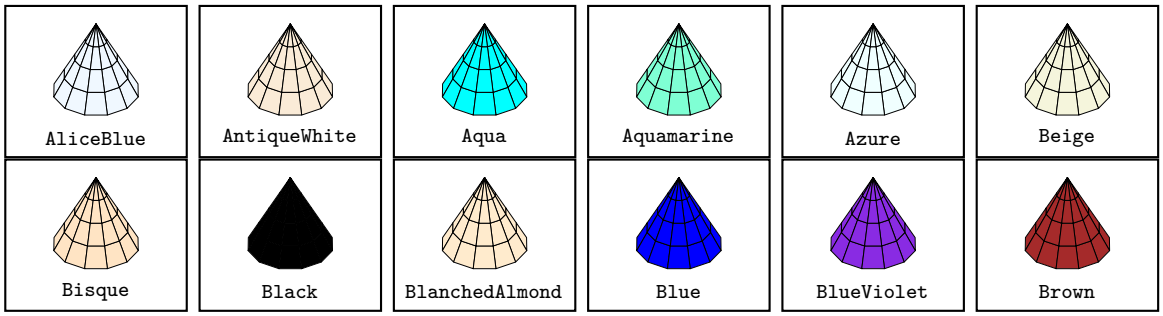
Il y a 68 couleurs prédéfinies, qui sont identifiées dans le fichier *solides.pro* : *Black*, *White*, et les 66 couleurs ci-dessous.

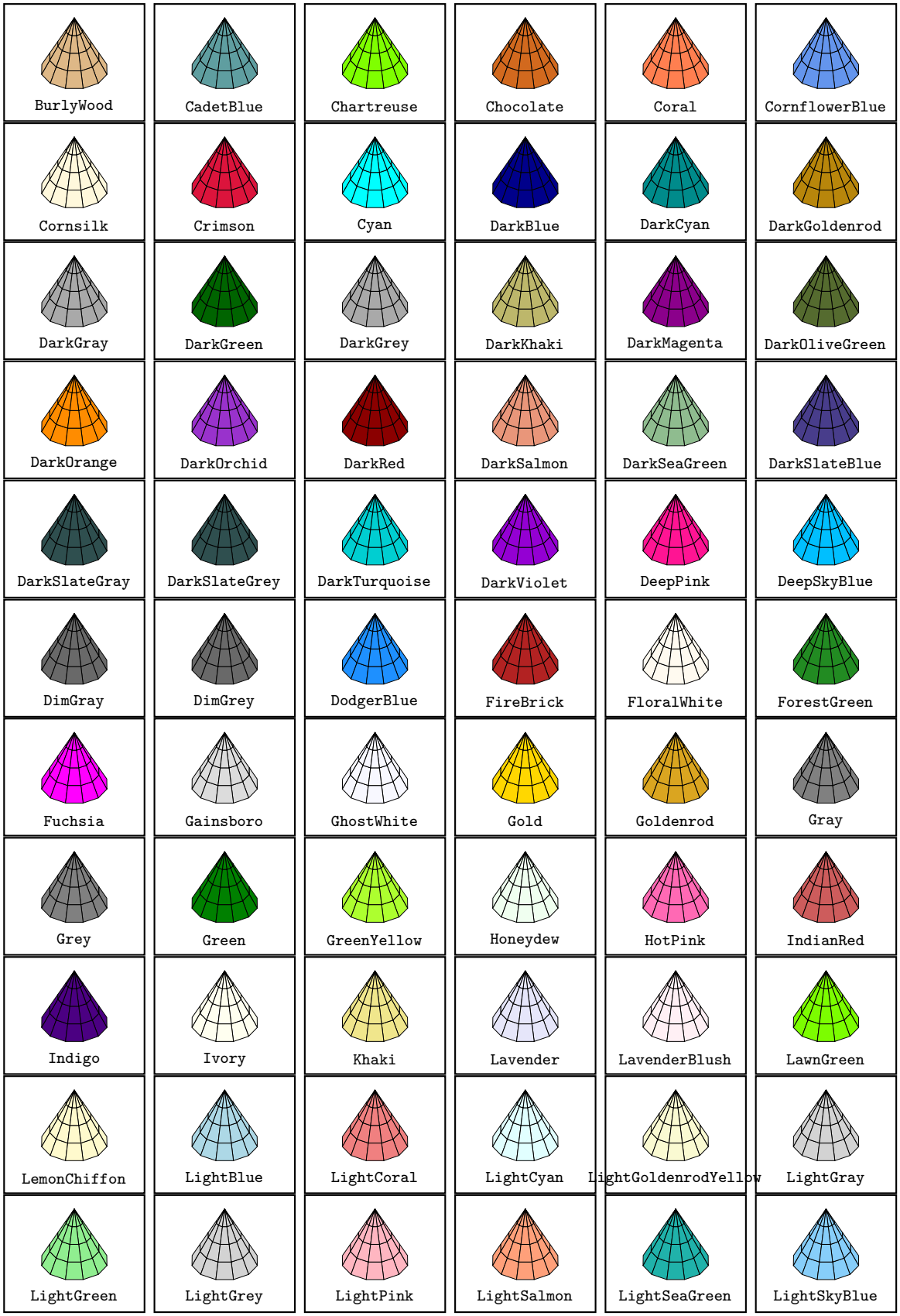


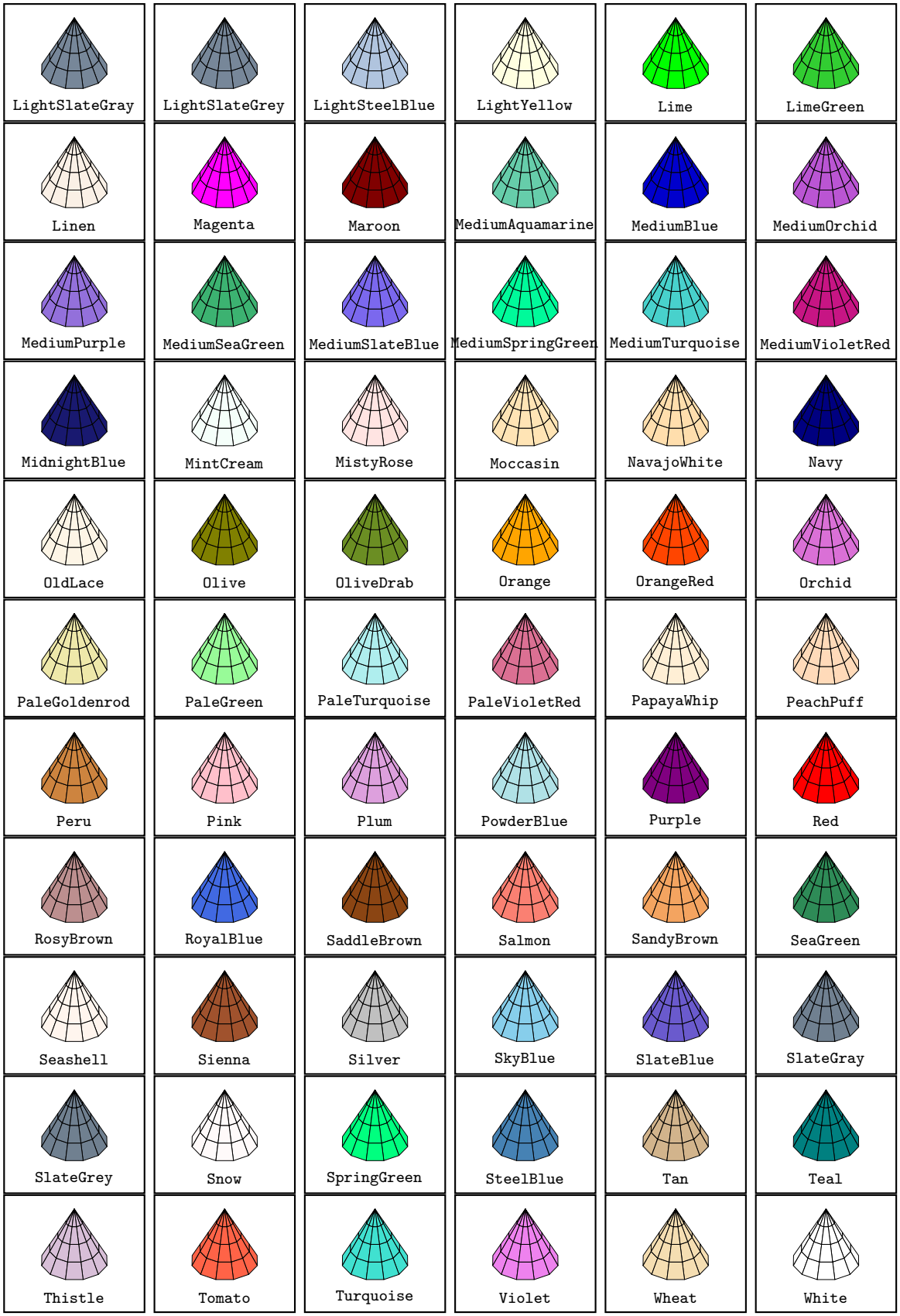


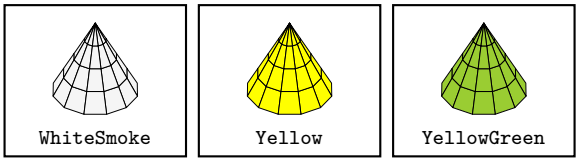
2.6.2 Couleurs prédéfinies par l'option [svgnames]

Les couleurs suivantes sont reconnues par pstricks si l'on utilise l'option [svgnames]. Par contre, elles ne sont pas identifiées dans le fichier *solides.pro* : on ne peut les utiliser directement dans l'option [fcol].
Ces couleurs sont proposées par le package xcolor.




BurlyWood
CadetBlue
Chartreuse
Chocolate
Coral
CornflowerBlue
Cornsilk
Crimson
Cyan
DarkBlue
DarkCyan
DarkGoldenrod
DarkGray
DarkGreen
DarkGrey
DarkKhaki
DarkMagenta
DarkOliveGreen
DarkOrange
DarkOrchid
DarkRed
DarkSalmon
DarkSeaGreen
DarkSlateBlue
DarkSlateGray
DarkSlateGrey
DarkTurquoise
DarkViolet
DeepPink
DeepSkyBlue
DimGray
DimGrey
DodgerBlue
FireBrick
FloralWhite
ForestGreen
Fuchsia
Gainsboro
GhostWhite
Gold
Goldenrod
Gray
Grey
Green
GreenYellow
Honeydew
HotPink
IndianRed
Indigo
Ivory
Khaki
Lavender
LavenderBlush
LawnGreen
LemonChiffon
LightBlue
LightCoral
LightCyan
LightGoldenrodYellow
LightGray
LightGreen
LightGrey
LightPink
LightSalmon
LightSeaGreen
LightSkyBlue


LightSlateGray
LightSlateGrey
LightSteelBlue
LightYellow
Lime
LimeGreen
Linen
Magenta
Maroon
MediumAquamarine
MediumBlue
MediumOrchid
MediumPurple
MediumSeaGreen
MediumSlateBlue
MediumSpringGreen
MediumTurquoise
MediumVioletRed
MidnightBlue
MintCream
MistyRose
Moccasin
NavajoWhite
Navy
OldLace
Olive
OliveDrab
Orange
OrangeRed
Orchid
PaleGoldenrod
PaleGreen
PaleTurquoise
PaleVioletRed
PapayaWhip
PeachPuff
Peru
Pink
Plum
PowderBlue
Purple
Red
RosyBrown
RoyalBlue
SaddleBrown
Salmon
SandyBrown
SeaGreen
Seashell
Sienna
Silver
SkyBlue
SlateBlue
SlateGray
SlateGrey
Snow
SpringGreen
SteelBlue
Tan
Teal
Thistle
Tomato
Turquoise
Violet
Wheat
White



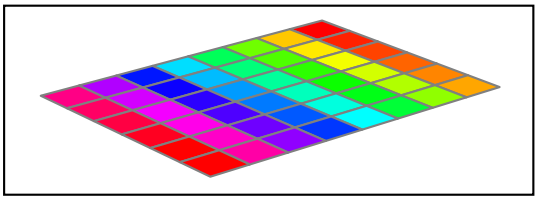
2.7 Les dégradés de couleur

Pour les dégradés de couleurs, on dispose des options `[hue]`, `[inhue]` et `[inouthue]` qui permettent respectivement de spécifier si le dégradé doit avoir lieu sur les faces externes, internes, ou toutes.

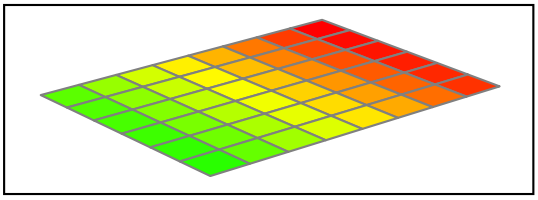
Ces dégradés peuvent être dans les espaces HSB, RGB ou CMYK. C'est le nombre d'arguments de `hue` (resp. `inhue`, `inouthue`) qui détermine le cas de figure

2.7.1 Dégradé dans l'espace HSB, saturation et brillance maximales

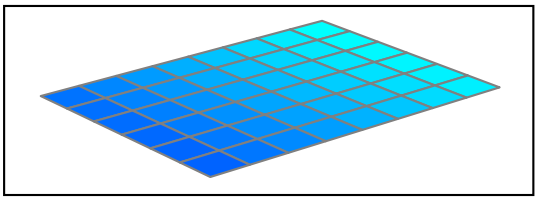
Il y a 2 arguments : `[hue=`h_0 h_1`]` où les nombres h_0 et h_1 vérifiant $0 \leq h_0 < h_1 \leq 1$ indiquent les bornes du premier paramètre dans l'espace HSB.



```
\psSolid[object=grille,
   base=-3 5 -3 3,
   linecolor=gray,
   hue=0 1]
```



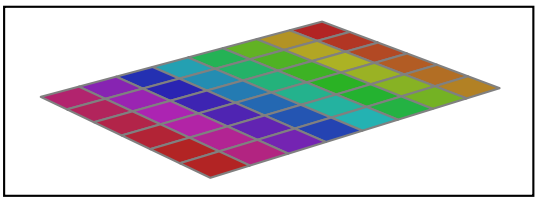
```
\psSolid[object=grille,
   base=-3 5 -3 3,
   linecolor=gray,
   hue=0 .3]
```



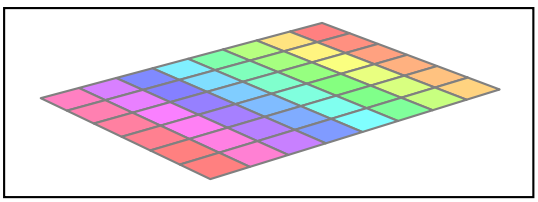
```
\psSolid[object=grille,
   base=-3 5 -3 3,
   linecolor=gray,
   hue=.5 .6]
```

2.7.2 Dégradé dans l'espace HSB, saturation et brillance fixes

Il y a 4 arguments : `[hue=`h_0 h_1 s b`]` où les nombres h_0 et h_1 vérifiant $0 \leq h_0 < h_1 \leq 1$ indiquent les bornes du premier paramètre dans l'espace HSB et où s et b sont les paramètres respectifs *saturastion* et *brillance*.



```
\psSolid[object=grille,
   base=-3 5 -3 3,
   linecolor=gray,
   hue=0 1 .8 .7]
```

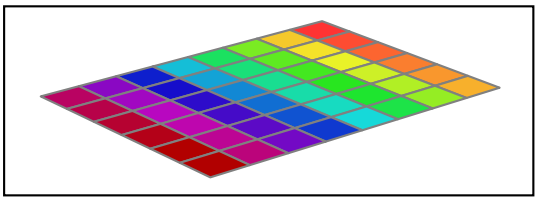


```
\psSolid[object=grille,
   base=-3 5 -3 3,
```

```
linecolor=gray,                    hue=0 1 .5 1]
```

2.7.3 Dégradé dans l'espace HSB, cas général

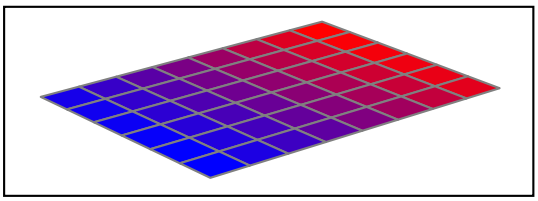
Il y a 7 arguments : `[hue=`h_0 s_0 b_0 h_1 s_1 b_1 `(hsb)]` où les nombres h_i, s_i et b_i indiquent les bornes des paramètre HSB.



```
\psSolid[object=grille,
   base=-3 5 -3 3,
   linecolor=gray,
   hue=0 .8 1 1 1 .7 (hsb)]
```

2.7.4 Dégradé dans l'espace RGB

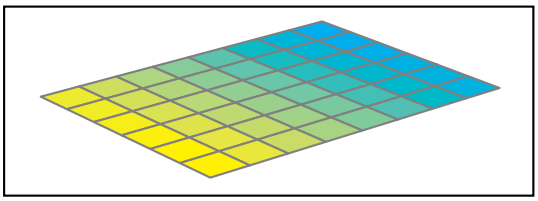
Il y a 6 arguments : `[hue=`r_0 g_0 b_0 r_1 g_1 b_1`]` où les nombres r_i, g_i et b_i indiquent les bornes respectives des 3 paramètres RGB.



```
\psSolid[object=grille,
   base=-3 5 -3 3,
   linecolor=gray,
   hue=1 0 0 0 0 1]
```

2.7.5 Dégradé dans l'espace CMYK

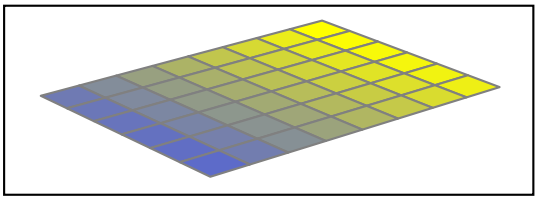
Il y a 8 arguments : `[hue=`c_0 m_0 y_0 k_0 c_1 m_1 y_1 k_1`]` où les nombres c_i, m_i, y_i et k_i indiquent les bornes respectives des 4 paramètres CMYK.



```
\psSolid[object=grille,
   base=-3 5 -3 3,
   linecolor=gray,
   hue=1 0 0 0 0 0 1 0]
```

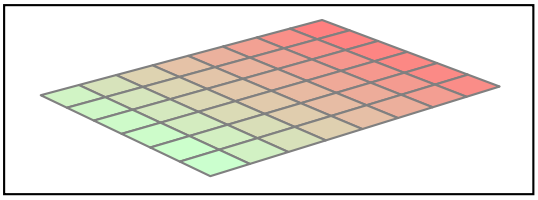
2.7.6 Dégradé entre 2 couleurs nommées

Il y a deux paramètres `[hue=`$str1$ $str2$`]` où $str1$ et $str2$ sont des chaînes de caractères désignant des noms de couleurs connues dans `solides.pro`.



```
\psSolid[object=grille,
   base=-3 5 -3 3,
   linecolor=gray,
   hue=(jaune) (CadetBlue)]
```

Si on veut utiliser des couleurs définies par `xcolor`, on utilise les paramètres `color1`, `color2`, etc... de `\psSolid`.



```
\psSolid[object=grille,
   base=-3 5 -3 3,
   linecolor=gray,
   color1=red!50,
   color2=green!20,
   hue=(color1) (color2)]
```

2.7.7 Désactiver la gestion des couleurs

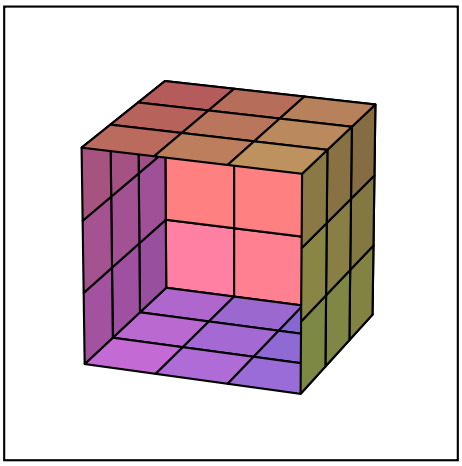
Pour certaines utilisations spécifiques, on peut avoir besoin de désactiver la gestion des couleurs. C'est en particulier le cas lorsque l'on utilise un objet déjà présent en mémoire ou défini dans des fichiers externes, Dans ces configurations, si on ne désactive pas la gestion des couleurs et si on n'en définit pas de nouvelles, ce sont les couleurs par défaut qui vont surcharger celles qui étaient sauvegardées.

Pour désactiver cette gestion, on utilise l'option `[deactivatecolor]`.

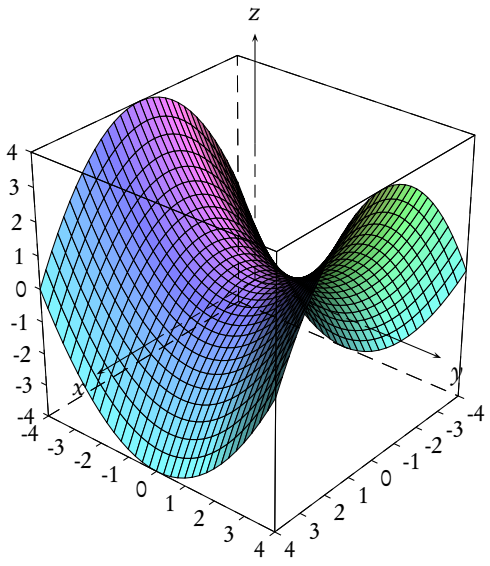
2.7.8 Exemples d'utilisation des options `[inhue]` et `[inouthue]`

On rappelle que l'option `[inhue]` permet de colorier les faces intérieures, `[hue]` permet de colorier les faces extérieures et `[inouthue]` peindra dans la continuité faces intérieurs et intérieures.

On rappelle que pour voir les faces intérieures il faut activer l'option `hollow`, comme dans les deux exemples suivants.

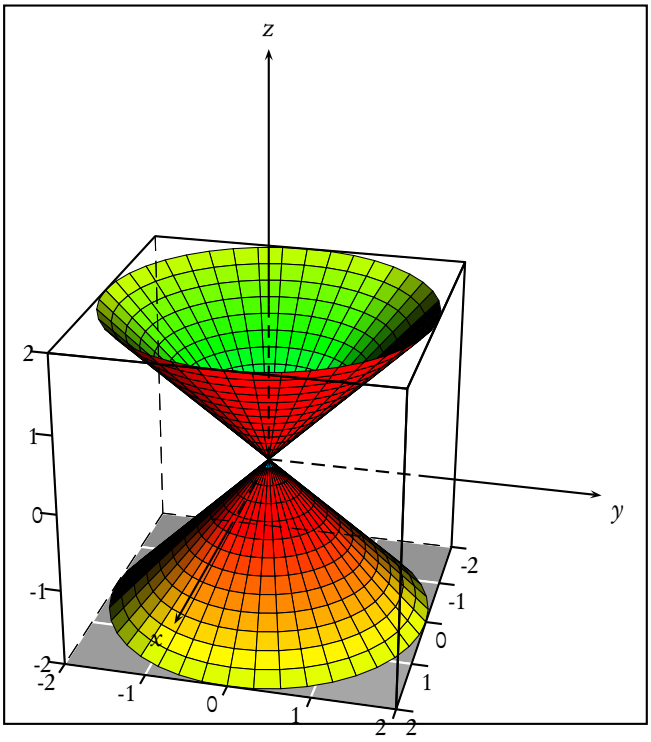


```
\psset{lightsrc=45 15 20,
       viewpoint=50 20 20 rtp2xyz,Decran=50}
\begin{pspicture}(-3,-3)(3,3)
\psframe(-3,-3)(3,3)
\psSolid[object=cube,
         a=3,ngrid=3,
         hollow,
         inouthue=0 1 0.5 1,
         rm=36 1 44 {} for]%
\end{pspicture}
```




```
\psset{unit=0.5}
\psset{lightsrc=30 30 25}
\psset{viewpoint=50 40 30 rtp2xyz,Decran=50}
\begin{pspicture}(-6,-8)(7,8)
\psSurface[ngrid=.25 .25,inouthue=1 0 0.5 1,
    linewidth=0.5\pslinewidth,axesboxed,
    algebraic](-4,-4)(4,4){%
    ((y^2)-(x^2))/4 }
\end{pspicture}
```

Pour colorier avec les paramètres de [hue] les faces intérieures et extérieures on utilisera l'option [hue] pour l'extérieur et l'option [inhue] pour l'intérieur, comme dans l'exemple suivant :




```
\psset{unit=0.5}
\begin{pspicture}(-7,-7)(10,12)
\psframe(-7,-7)(10,12)
\psset[pst-solides3d]{viewpoint=20 5 10,
      Decran=50,lightsrc=20 10 5}
\psSolid[object=grille,base=-2 2 -2 2,
      linecolor=white](0,0,-2)
 % Parametric Surfaces
\defFunction{cone}(u,v)
   {u v Cos mul}{u v Sin mul}{u}
\psSolid[object=surfaceparametree,
      base=-2 2 0 2 pi mul,
      inhue=0.8 0.2,hue=0.8 0.2,
      function=cone,linewidth=0.5\pslinewidth,
      ngrid=25 40]%
\gridIIID[Zmin=-2,Zmax=2](-2,2)(-2,2)
\end{pspicture}
```

2.8 Colorier les facettes une à une

L'argument [fcol=i_0 (c_0) i_1 (c_1) ...i_n (c_n)], où les i_k sont des entiers et les c_k des noms de couleurs, permet de spécifier la couleur de faces particulières. À la face d'incice i_k correspond la couleur c_k. L'entier n doit être inférieur à l'indice maximum des faces du solide considéré.

Pour les noms de couleurs c_k, il y a 68 valeurs prédéfinies (soit tous les noms définis dans le fichier color.pro au 12 octobre 2007). Ces valeurs sont : *GreenYellow*, *Yellow*, *Goldenrod*, *Dandelion*, *Apricot*, *Peach*, *Melon*, *YellowOrange*, *Orange*, *BurntOrange*, *Bittersweet*, *RedOrange*, *Mahogany*, *Maroon*, *BrickRed*, *Red*, *OrangeRed*, *RubineRed*, *WildStrawberry*, *Salmon*, *CarnationPink*, *Magenta*, *VioletRed*, *Rhodamine*, *Mulberry*, *RedViolet*, *Fuchsia*, *Lavender*, *Thistle*, *Orchid*, *DarkOrchid*, *Purple*, *Plum*, *Violet*, *RoyalPurple*, *BlueViolet*, *Periwinkle*, *CadetBlue*, *CornflowerBlue*, *MidnightBlue*, *NavyBlue*, *RoyalBlue*, *Blue*, *Cerulean*, *Cyan*, *ProcessBlue*, *SkyBlue*, *Turquoise*, *TealBlue*, *Aquamarine*, *BlueGreen*, *Emerald*, *JungleGreen*, *SeaGreen*, *Green*, *ForestGreen*, *PineGreen*, *LimeGreen*, *YellowGreen*, *SpringGreen*, *OliveGreen*, *RawSienna*, *Sepia*, *Brown*, *Tan*, *Gray*, *Black*, *White*. La liste de ces 68 couleurs est disponible dans la commande \colorfaces (voir exemple d'utilisation dans le paragraphe sur le maillage du cube).


Prévoir dans ce cas que le nombre de faces $n_1 \times n_2 + 2$(faces supérieure et inférieure) soit plus petit que 68 !

L'utilisateur peut également définir ses propres couleurs. Il dispose pour cela de deux méthodes :

- Il utilise l'un des 4 arguments optionnels [color1], [color2], [color3], [color4] de \psSolid, puis il transmet à fcol une paire du type i (color1) où i est l'indice de la face considérée. Les arguments [color1], etc...s'utilisent de la même façon que les arguments color et incolor.
  Par exemple, la commande suivante est une commande valide :
  ```
  \psSolid[a=1,object=cube,color1=red!60!yellow!20,fcol=0 (color1)]%
  ```

- Il définit ses propres noms de couleurs avec la commande \pstVerb puis utilise ces noms avec l'argument [fcol].
  Par exemple :
  ```
  \pstVerb{/hetre {0.764 0.6 0.204 setrgbcolor} def
           /chene {0.568 0.427 0.086 setrgbcolor} def
           /cheneclair {0.956 0.921 0.65 setrgbcolor} def
  ```

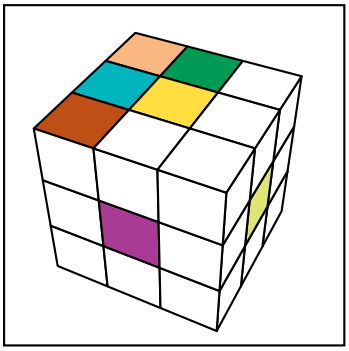
```
                }%
```

Puis ensuite :

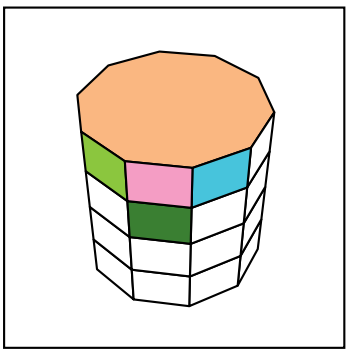
```
fcol=0 (hetre) 1 (chene)  2 (cheneclair)
```

Les 4 arguments `color1`, `color2`, `color3`, `color4` ont des valeurs par défaut :

- color1=cyan !50
- color2=magenta !60
- color3=blue !30
- color4=red !50



```
\psSolid[
   fcol=0 (Apricot)
        1 (Aquamarine)
        2 (Bittersweet)
        3 (ForestGreen)
        4 (Goldenrod)
        13 (GreenYellow)
        40 (Mulberry),
  object=cube,mode=3
    ]%
```

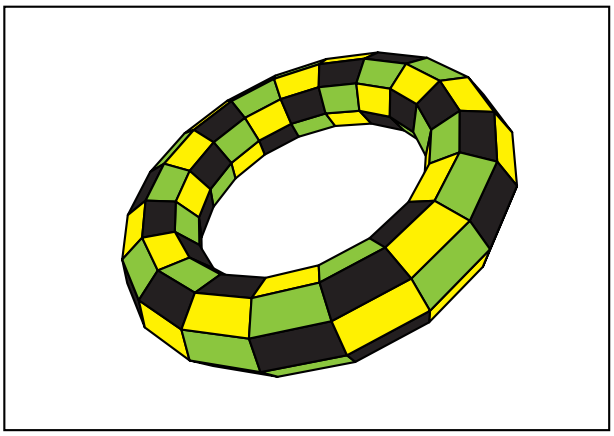


```
\psSolid[
  fcol= 0 (Apricot)
        2 (Lavender)
        3 (SkyBlue)
        10 (LimeGreen)
        12 (OliveGreen),
  object=cylindre,
   h=4,
   ngrid=4 10,
](0,0,-2)
```

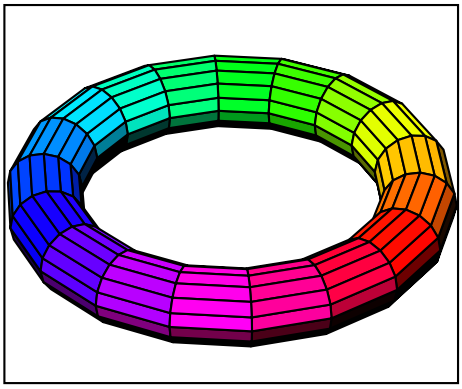
Le choix des faces à colorier peut se faire en utilisant un code `PostScript` :

```
fcol=48 {i (Black) i 1 add (LimeGreen) i 2 add (Yellow) /i i 3 add store} repeat
```

qui va colorier alternativement en noir, en vert et en jaune les facettes.



Si l'option **hue** est activée, les facettes du solide sont coloriées avec le dégradé de couleurs de l'arc-en-ciel.



```
\psset{viewpoint=50 50 50,Decran=86,
       lightsrc=50 20 1e2}
\psSolid[r1=5,r0=1,object=tore,
         ngrid=16 18,hue=0 1]%
```

2.9 Gestion de la transparence

Le paramètre `[opacity=k]` avec k réel vérifiant $0 \leq k \leq 1$, permet de définir le niveau d'opacité pour l'ensemble des tracés qui suivent. En code jps, on utilise l'instruction équivalente `k setfillopacity`. Cette dernière trouve notamment son application dans l'option `fcol`. Par exemple, l'instruction

```
fcol=0 (.5 setfillopacity yellow)
```

définit la face d'indice `0` comme étant jaune, avec une opacité de `0,5`.

2.10 Définition du maillage

L'utilisateur peut spécifier le maillage du solide avec l'option `[ngrid]` dans la commande `\psSolid`.

Pour les objets `cube`, `prisme`, `prismecreux`, la syntaxe est `[ngrid=`n_1`]` où n_1 représente le nombre de mailles sur l'axe vertical.

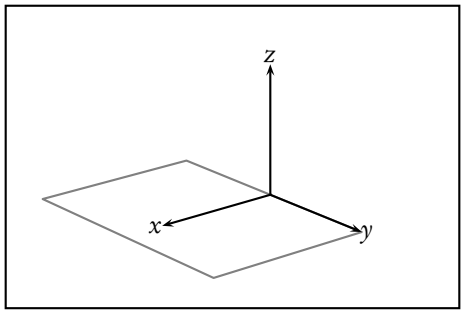
Pour les objets `sphere`, `cylindre`, `cylindrecreux`, `cone`, `conecreux`, `tronccone`, `troncconecreux`, `tore`, la syntaxe est
`[ngrid=`n_1 n_2`]` où n_1 est entier supérieur ou égal à 1 (à 2 pour `tore`) représentant le nombre de mailles sur l'axe vertical, et n_2 est un entier représentant le nombre de divisions sur le cercle.

Pour les objets `grille`, `surface`, `surface*`, `surfaceparametree`, la syntaxe est `[ngrid=`n_1 n_2`]` où n_1 et n_2 peuvent être réels ou entiers. Le nombre n_1 se rapporte à l'axe Ox et n_2 se rapporte à l'axe Oy. Si n_2 est absent, on considère que $n_2 = n_1$.

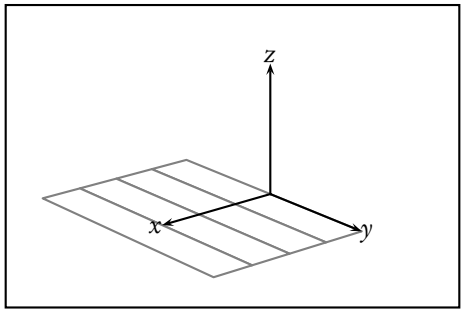
Si n_1 est entier, il représente le nombre de mailles sur l'axe Ox. S'il est réel, il représente le pas de maillage sur l'axe Ox. Par exemple, le nombre codé `1` est entier, alors que celui codé `1.` est réel (noter le point).

Voici quelques exemples :

2.10.1 La grille

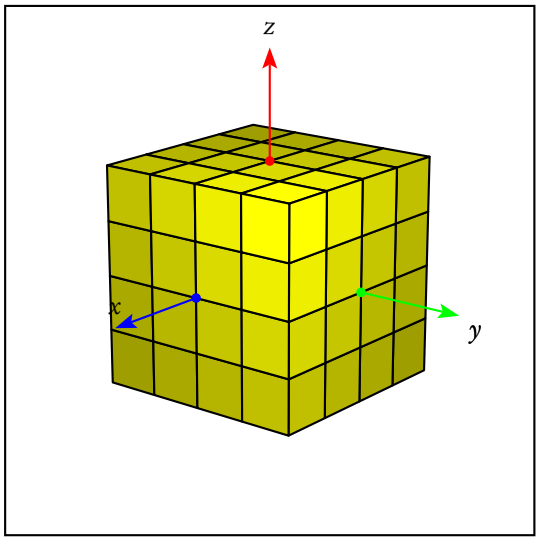


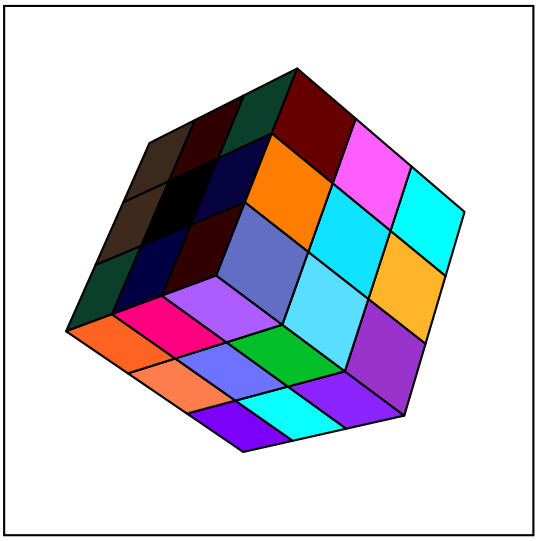

```
\psSolid[object=grille,
   ngrid=1,
   base=0 4 -3 3,
   linecolor=gray](0,0,0)
```




```
\psSolid[object=grille,
   ngrid=1. 1,
   base=0 4 -3 3,
   linecolor=gray](0,0,0)
```

2.10.2 Le cube





Pour le premier exemple, le maillage des faces est fixé à 4×4 facettes/face et la commande est la suivante :

```
\psSolid[a=8,object=cube,ngrid=4,
         fillcolor=yellow]%
```

Dans le deuxième exemple, le maillage des faces est fixé à 3×3 et les couleurs des facettes sont diverses. On utilise le package `arrayjob` pour stocker les couleurs :

```
\newarray\colors
\readarray{colors}{%
Apricot&Aquamarine%
etc.}
```

Puis la liste des couleurs à afficher est donnée par la commande :

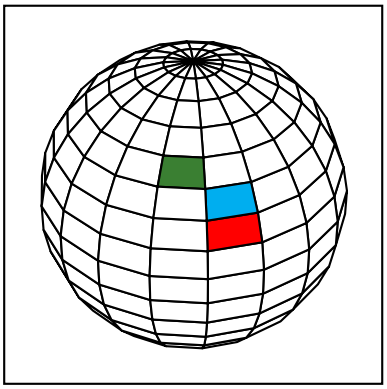
```
\edef\colorfaces{}%
\multido{\i=0+1}{67}{%
 \checkcolors(\i)
\xdef\colorfaces{%
     \colorfaces\i\space(\cachedata)\space}
     }
```

On place l'option : `fcol=\colorfaces`. Le cube maillé est appelé par :

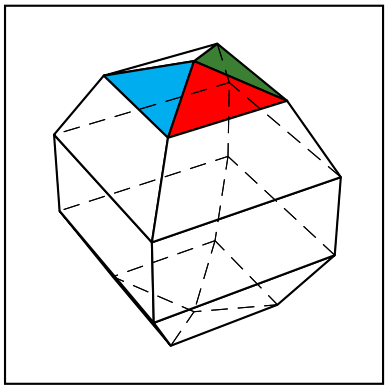
```
\psSolid[a=8,object=cube,ngrid=3,%
        fcol=\colorfaces,
        RotY=45,RotX=30,RotZ=20]%
```

L'option `[grid]` permet, éventuellement, de ne pas tracer les traits du quadrillage.

2.10.3 La sphère

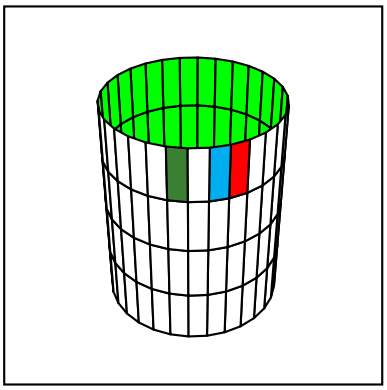


```
\psset{color1=cyan,color2=red}
   fcol=251 (OliveGreen) 232 (color1) 214 (color2),
   object=sphere,
   ngrid=16 18,
   RotX=180,RotZ=30
]%
```

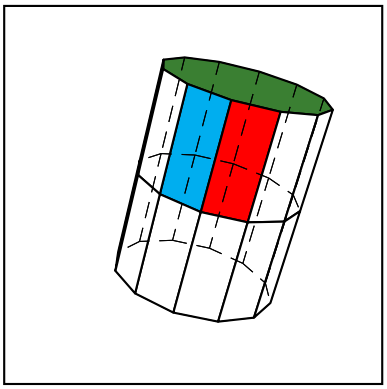


```
\psset{color1=cyan,color2=red}
\psSolid[
   action=draw*,
   fcol=0 (OliveGreen) 2 (color1) 3 (color2),
   object=sphere,
   ngrid=4 4,
   RotX=180,RotZ=30
]%
```

2.10.4 Cylindres

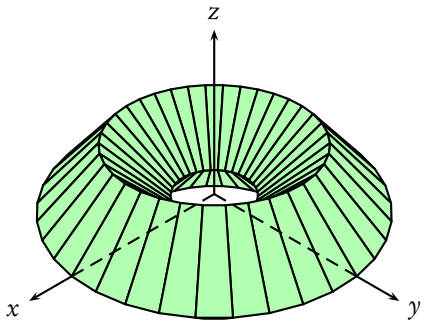


```
\psset{color1=cyan,color2=red}
\psSolid[
   fcol=0 (OliveGreen) 2 (color1) 3 (color2),
   h=5,r=2,
   object=cylindrecreux,
   ngrid=4 30,
   RotZ=30
    ](0,0,-2.5)
```

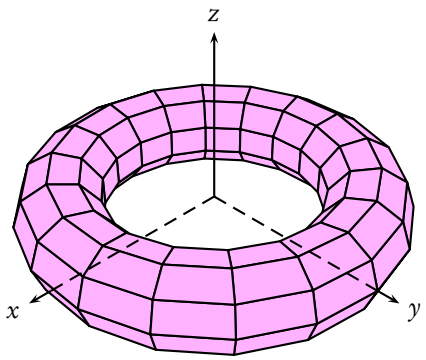


```
\psset{color1=cyan,color2=red}
\psSolid[
   action=draw*,
   fcol=0 (OliveGreen) 2 (color1) 3 (color2),
   object=cylindre,
   ngrid=2 12,
   RotY=-20
    ](0,0,-2.5)
```

2.10.5 Tore




```
\psSolid[r1=2.5,r0=1.5,
        object=tore,
        ngrid=4 36,
        fillcolor=green!30,
        action=draw**]%
```

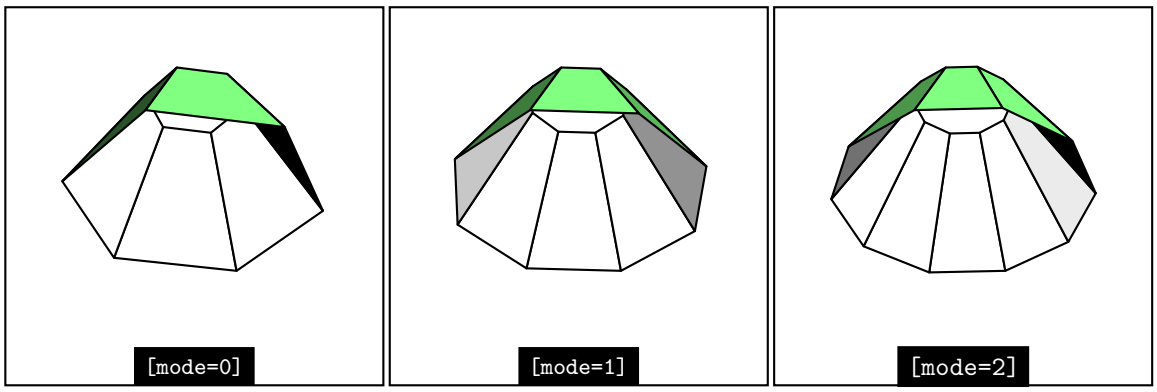


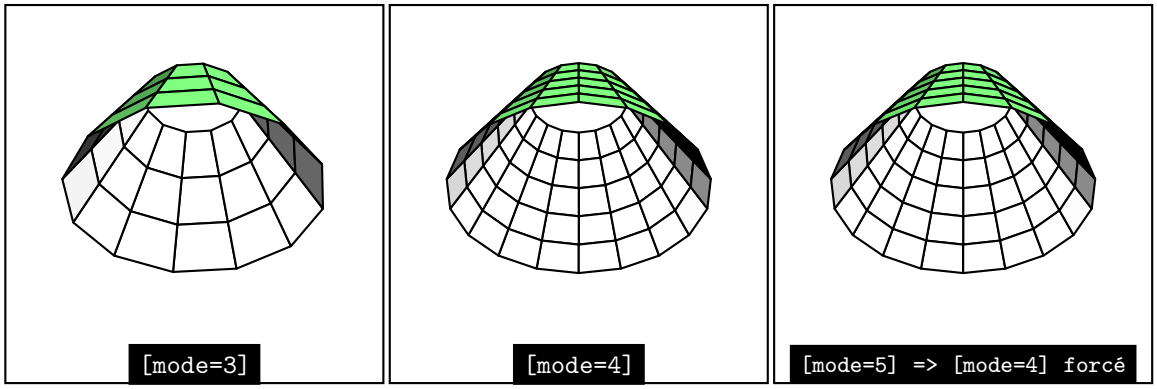

```
\psSolid[r1=3.5,r0=1,
         object=tore,
         ngrid=9 18,
         fillcolor=magenta!30,
         action=draw**]%
```

2.11 Les modes

Pour un certain nombre de solides, on a prédéfini certains maillages. Le positionnement du paramètre `[mode=0, 1, 2, 3 ou 4]` permet de passer du maillage prédéfini le plus grossier `[mode=0]` au maillage prédéfini le plus fin `[mode=4]`.

Ceci permet notamment de mettre au point une image avec tous les solides en `[mode=0]` afin d'accélérer les calculs, avant de passer au `[mode=4]` pour une image définitive.

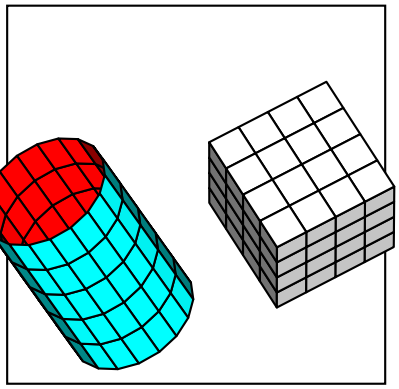




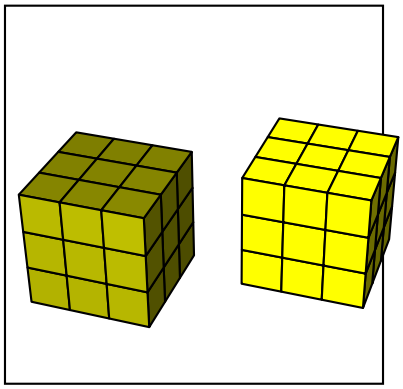
2.12 Éclairage par une source lumineuse ponctuelle

Deux paramètres, l'un positionne la source, l'autre fixe l'intensité lumineuse :

- `[lightsrc=20 30 50]` en coordonnées cartésiennes, ou `[lightsrc=viewpoint]` pour faire coïncider la source lumineuse avec l'observateur.
- `[lightintensity=2]` (valeur par défaut).

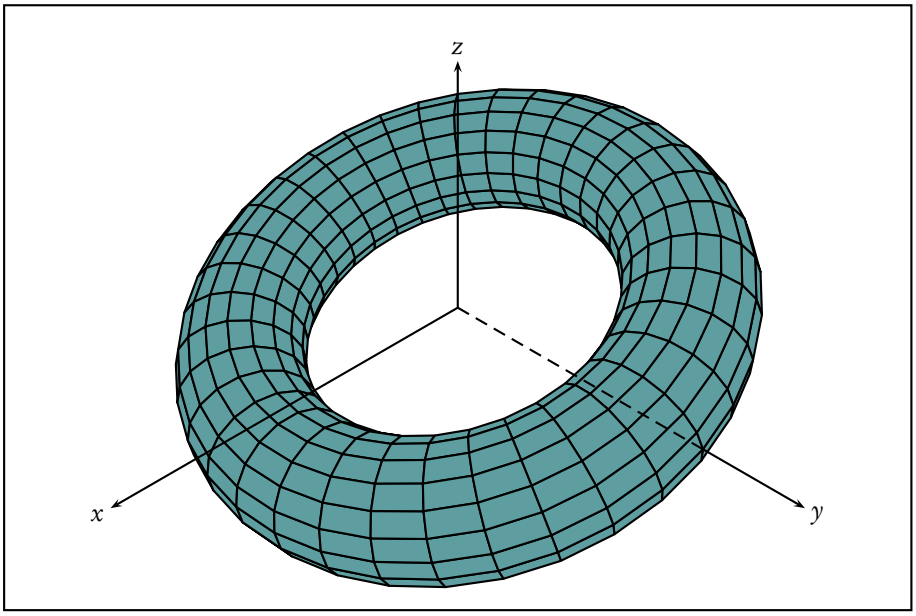


```
\psset{Decran=1e3,
       viewpoint=500 0 1000,
       lightsrc=viewpoint,
       mode=5}
\psSolid[object=cube,RotZ=30](0,2,0)
\psSolid[object=cylindrecreux,
         RotX=30,RotZ=-30,
         fillcolor=cyan,incolor=red](4,-3,0)
```



```
\psset{Decran=30,
       viewpoint=30 30 30,
       lightsrc=viewpoint,
       mode=3}
\psSolid[object=cube,
         lightintensity=3,
         RotX=90](0,3,0)
\psSolid[object=cube,
         lightintensity=1,
         RotX=90](3,-3,0)
```

Si l'option `[lightsrc=`*value1 value2 value3*`]` n'est pas spécifiée, l'objet est uniformément éclairé.

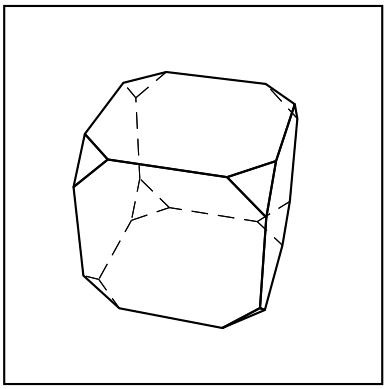



```
\psSolid[r1=3.5,r0=1,object=tore,ngrid=18 36,fillcolor={[rgb]{.372 .62 .628}}]%
```

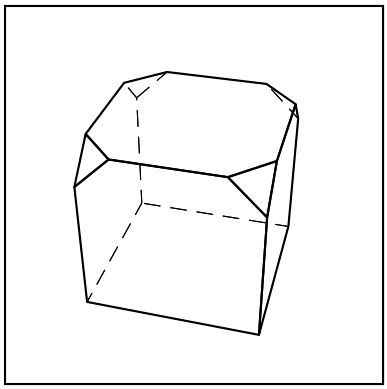
2.13 Tronquer les sommets d'un solide

L'option `[trunc]` permet de tronquer les sommets soit globalement, soit individuellement. Cette option utilise l'argument `[trunccoeff]` (valeur 0,25 par défaut) qui indique le rapport k utiliser pour la troncature ($0 < k \leq 0,5$).

- `[trunc=all]` tronque tous les sommets ;
- `[trunc=0 1 2 3]` tronque les sommets `[0,1,2 et 3]` ;



```
\psSolid[object=cube,
   action=draw,RotZ=30,
   trunccoeff=.2,trunc=all,
]%
```

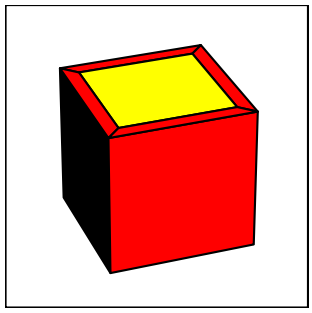


```
\psSolid[object=cube,
   RotZ=30,action=draw,
   trunccoeff=.2,
   trunc=0 1 2 3,
]%
```

2.14 Affiner un solide

Nous désignerons par *affinage de rapport k* l'opération qui, pour une face donnée de centre G, consiste à effectuer sur cette face une homothétie de rapport k et de centre G, puis à diviser la face originelle en utilisant cette nouvelle face.

Par exemple, voici un cube ayant subi un affinage de rapport 0,8 sur sa face supérieure :

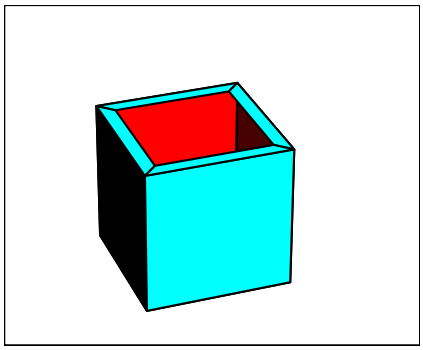


L'option `affinage` permet d'affiner les faces soit globalement, soit individuellement. Cette option utilise l'argument `affinagecoeff` (valeur 0,8 par défaut) qui indique le rapport k utiliser pour l'affinage ($0 < k < 1$).

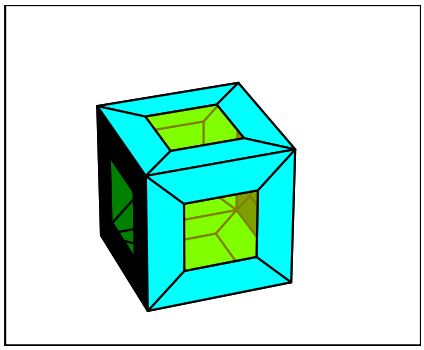
- `[affinage=all]` affine toutes les faces ;
- `[affinage=0 1 2 3]` affine les faces `[0,1,2 et 3]` ;

Lorsqu'un face se trouve affinée, le comportement par défaut supprime la face centrale obtenue. Toutefois, l'option `affinagerm` permet de conserver cette face centrale.

Si on conserve la face centrale, elle est par défaut de la couleur de la face originelle. L'option `fcolor` permet de spécifier une couleur de remplacement.

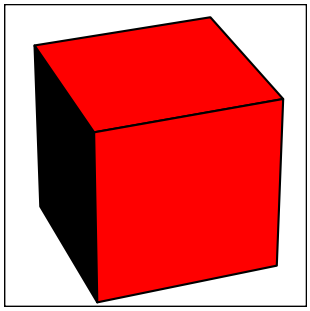


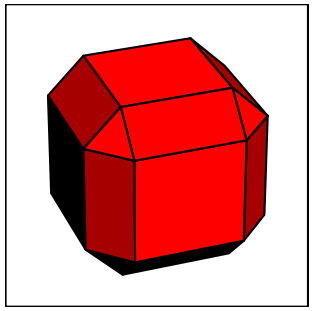
```
\psSolid[object=cube,
   fillcolor=cyan,
   incolor=red,
   hollow,
   affinage=0]
```



```
\psSolid[object=cube,
   fillcolor=cyan,
   affinagecoeff=.5,
   affinagerm,
   fcolor=.5 setfillopacity Yellow,
   hollow,
   affinage=all]
```

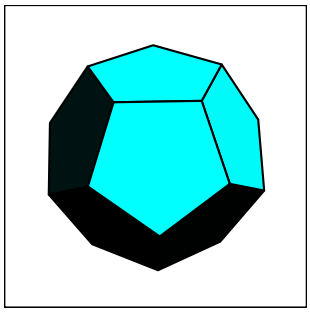
2.15 Chanfreiner un solide

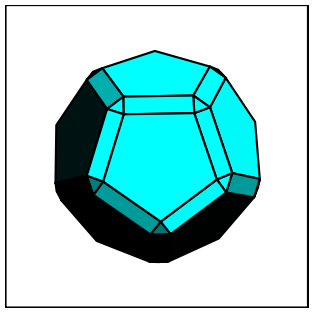




```
\psSolid[object=cube,
   a=5,
   fillcolor=red,
   chanfrein,
   chanfreincoeff=.6,
]
```

L'option `[chanfrein]` permet de chanfreiner un solide. Cette option utilise l'argument `[chanfreincoeff]` (valeur 0,8 par défaut) qui indique le rapport k à utiliser ($0 < k < 1$). Ce rapport est celui d'une homothétie de cetre le centre de la face considérée.





```
\psSolid[object=dodecahedron,
   a=5,
   fillcolor=cyan,
   chanfrein,
   chanfreincoeff=.8,
]
```

2.16 L'option `transform`

Avec l'option `transform=...`, c'est une formule de transformation de $\mathbf{R}^3$ vers $\mathbf{R}^3$ qui va être appliquée à chaque point du solide. Dans ce premier exemple, l'objet qui subira la transformation est un cube. Le cube de référence est en jaune, le cube transformé en vert et le cube en fil de fer représente le cube avant transformation.

2.16.1 Facteur d'échelle identique appliqué aux trois coordonnées

Le facteur d'échelle est pris égal à 0.5. On l'introduit soit en définissant la variable '`/Facteur`' :

```
\pstVerb{/Facteur {.5 mulv3d} def}%
```

puis en l'introduisant dans l'option '`transform`' :

```
\psSolid[object=cube,a=2,ngrid=3,
         transform=Facteur](2,0,1)%
```

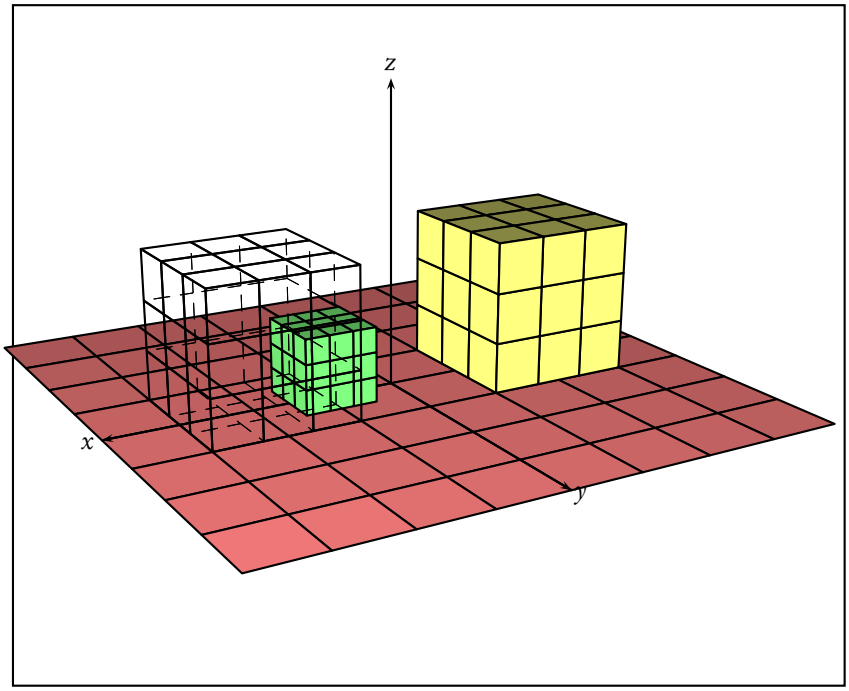
soit directement dans le code :

```
\psSolid[object=cube,a=2,ngrid=3,
         transform={.5 mulv3d}](2,0,1)%
```

Remarque : On vient d'utiliser ici un raccourci jps pour définir une fonction de de $\mathbf{R}^3$ vers $\mathbf{R}^3$. Une autre méthode aurait été d'utiliser le code

```
\defFunction[algebraic]{matransformation}(x,y,z)
   {.5*x}
   {.5*y}
   {.5*z}
```

puis de transmettre dans les options `[transform=matransformation]`.

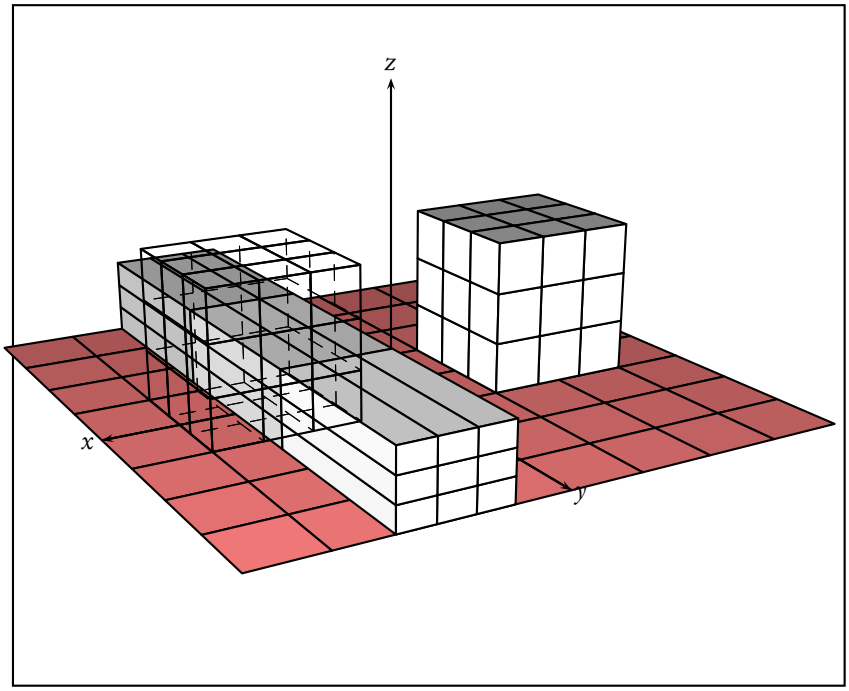



```
\psset{viewpoint=20 60 20 rtp2xyz,lightsrc=10 15 7,Decran=20}
\begin{pspicture}(-5,-5)(6,5)
\psframe(-5,-4)(6,5)
\psSolid[object=grille,base=-4 4 -4 4,fillcolor=red!50]%
\axesIIID(0,0,0)(4,4,4)%
\psSolid[object=cube,fillcolor=yellow!50,
      a=2,ngrid=3](-2,0,1)
\psSolid[object=cube,fillcolor=green!50,
      a=2,transform={.5 mulv3d},
      ngrid=3](2,0,1)
\psSolid[object=cube,
      action=draw,
      a=2,ngrid=3](2,0,1)
\end{pspicture}
```

Le facteur d'échelle s'applique aussi aux coordonnées de la position du centre du cube.

2.16.2 Facteur d'échelle différent pour les trois coordonnées

Prenons, par exemple, que l'on applique un facteur de 0.75 pour x, 4 pour y et 0.5 pour z, on transforme ainsi un cube en un parallélépipède en utilisant la fonction **scaleOpoint3d** de la librairie jps.

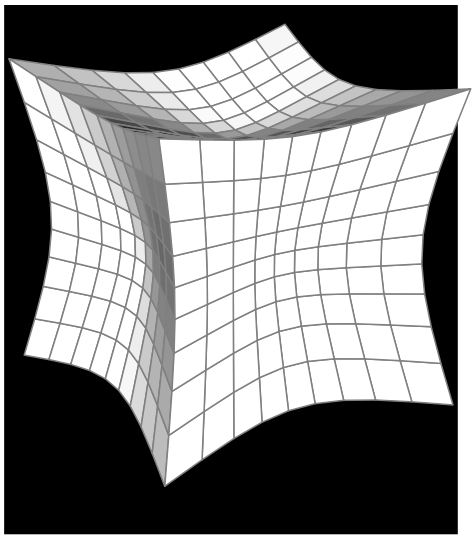



```
\psset{viewpoint=20 60 20 rtp2xyz,lightsrc=10 15 7,Decran=20}
\begin{pspicture}(-5,-5)(6,5)
\psframe(-5,-4)(6,5)
\psSolid[object=grille,base=-4 4 -4 4,fillcolor=red!50]%
\axesIIID(0,0,0)(4,4,4)%
\psSolid[object=cube,
      a=2,ngrid=3](-2,0,1)
\psSolid[object=cube,
      a=2,transform={.75 4 .5 scale0point3d},
      ngrid=3](2,0,1)
\psSolid[object=cube,
      action=draw,
      a=2,ngrid=3](2,0,1)
\end{pspicture}
```

2.16.3 Transformation liée à la distance du point à l'origine

Un exemple que l'on va appliquer à un cube :

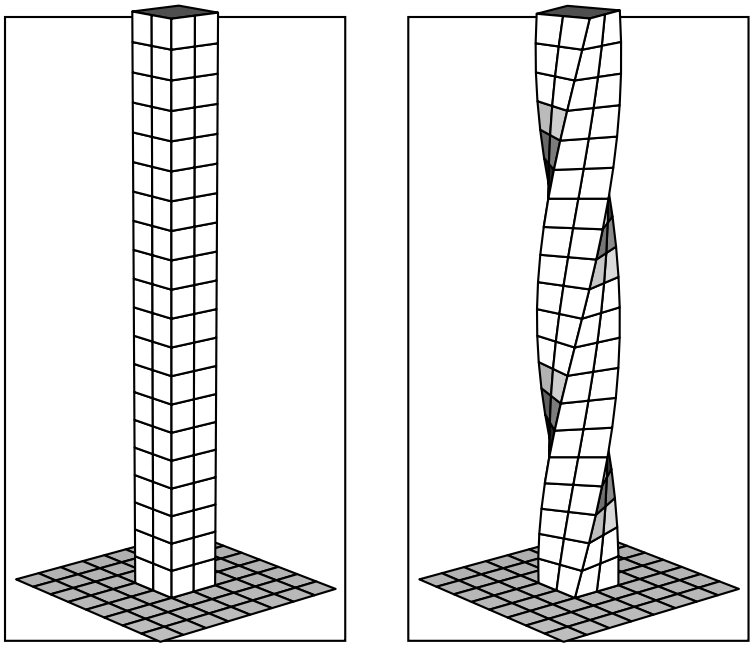
$$\begin{cases} x' = \left(0.5\sqrt{x^2+y^2+z^2}+1-0.5\sqrt{3}\right)x \\ y' = \left(0.5\sqrt{x^2+y^2+z^2}+1-0.5\sqrt{3}\right)y \\ z' = \left(0.5\sqrt{x^2+y^2+z^2}+1-0.5\sqrt{3}\right)z \end{cases}$$



```
\begin{pspicture}(-3,-4)(3,3)
\psset{viewpoint=20 60 20 rtp2xyz,lightsrc=10 15 7,Decran=20}
\pstVerb{
/gro {
4 dict begin
   /M defpoint3d
   /a .5 def
   /b 1 a 3 sqrt mul sub def
   /k M norme3d a mul b add def
   M k mulv3d
end
} def}%
\psframe*(-3,-4)(3,3)
\psset{linewidth=.02,linecolor=gray}
\psSolid[object=cube,a=3,ngrid=9,
     transform=gro]%
\end{pspicture}
```

2.16.4 Torsion d'une poutre

Le solide de départ est un prisme de hauteur 10 cm de 20 étages (`ngrid=20 2`). À chaque étage, on applique une rotation supplémentaire d'axe Oz et de valeur 10^o par exemple. Comme les niveaux sont espacés de $0,5$ cm, on multiplie $z \times 20$.



```
\psset{viewpoint=50 50 20 rtp2xyz,lightsrc=25 37 17,Decran=50,unit=0.75}
\begin{pspicture}(-3,-1)(3.5,10)
\psframe(-3,-1)(3,10)
\psSolid[object=grille,base=-2 2 -2 2,ngrid=8]%
\psSolid[object=prisme,h=10,ngrid=20 2,
       base=0.5 0 0.5 0.5 0 0.5 -0.5 0.5 -0.5 0 -0.5 -0.5 0 -0.5 0.5 -0.5]%
\end{pspicture}
\begin{pspicture}(-3.5,-1)(3,10)
\psframe(-3,-1)(3,10)
\psSolid[object=grille,base=-2 2 -2 2,ngrid=8]%
\pstVerb{
/torsion {
2 dict begin
   /M defpoint3d % on récupère les coordonnées
   M /z exch def pop pop
   % on tourne de 10 degrés à chaque niveau
    M 0 0 z 20 mul rotateOpoint3d
end} def}%
\psSolid[object=prisme,h=10,ngrid=20 2,
       base=0.5 0 0.5 0.5 0 0.5 -0.5 0.5 -0.5 0 -0.5 -0.5 0 -0.5 0.5 -0.5,
       transform=torsion]%
\end{pspicture}
```

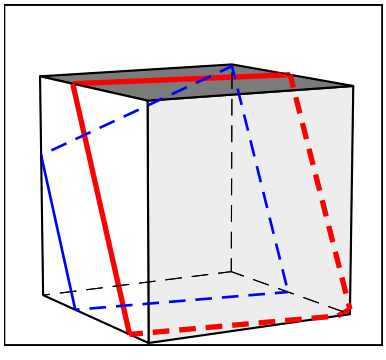
2.17 Tracés d'intersections planes

Pour chaque objet de type *solid*, il est possible de tracer l'intersection du solide considéré avec un ou plusieurs plans.

L'argument numérique `[intersectiontype=`k`]` (valeur -1 par défaut) détermine s'il y a ou non demande de tracé d'intersection. Positionné à 0, il y a tracé des intersections.

Restent 3 paramètres à régler :

- `[intersectionplan={`eq_1 ... eq_n`}]` définit la liste des équations e_i des plans de coupe. Les e_i peuvent être également des objets de type plan.
- `[intersectionlinewidth=`w_1 ... w_n`]` définit la liste des épaisseurs en picas w_i pour chacune des coupes.
- `[intersectioncolor=`str_1 ... str_n`]` définit la liste des couleurs des différents traits de coupe.



```
\psSolid[object=cube,
```

```
intersectiontype=0,
intersectionplan={[1 0 .5 2] [1 0 .5 -1]},
intersectionlinewidth=1 2,
intersectioncolor=(bleu) (rouge),
RotX=20,RotY=90,RotZ=30,
a=6,
action=draw*,
]
```

Chapitre 3

Utilisation de fichiers externes

Il peut être parfois utile d'utiliser des fichiers externes, que ce soit en lecture ou en écriture. Par exemple, lorsque l'on a construit des solides nécessitant de longs temps de calcul et que l'on veut tester différents points de vues ou différentes couleurs, il peut être intéressant de sauvegarder ces solides pour les relire ensuite, ce qui évitera de les recalculer. En particulier, cette technique est souvent utilisé pour construire des animations. On peut aussi avoir envie d'exporter un solide pour le réutiliser avec un autre logiciel.

Pour pst-solides3d, le choix a été fait de déléguer toutes les procédures de lecture/écriture à l'interpréteur postscript (et non pas à TEX ou LATEX). En conséquence, ce n'est pas la compilation LATEX qui provoquera l'exécution d'un ordre de lecture/écriture, mais la visualisation du fichier postscript produit.

En règle générale, la lecture de fichiers externes par un interpréteur postscript ne pose pas de problème (s'il n'y a pas franchissement de répertoire). Pour l'écriture en revanche, cela peut poser des problèmes de sécurité et il n'est pas rare que le visualisateur postscript interdise l'écriture par défaut. Il faut alors le configurer pour autoriser cette écriture.

Par défaut, sous Windows et Linux, la protection des fichiers du disque dur est activée et ne permet donc pas l'écriture sur le disque. Pour désactiver cette protection, tout au moins temporairement, voici les deux procédures correspondantes :

Linux : le plus simple est donc d'utiliser ghostscript directement, en console. Comme il n'y a rien à attendre comme image :

$> gs -dNOSAFER monfichier.ps quit.ps

Windows : dans le menu Options, l'option Protection des fichiers ne doit pas être cochée.

3.1 Fichiers .dat (spécifique à pst-solides3d)

Dans pst-solides3d, la structure de données utilisée pour un solide comporte 4 champs. Elle peut être stockée dans un ensemble de 4 fichiers .dat.

3.1.1 Écriture de fichiers .dat

On utilise l'action `[action=writesolid]` dans `\psSolid`, et on utilise l'option `file` pour spécifier le nom du fichier.

Par exemple, considérons le code ci-dessous :

```
\psSolid[object=tore,
   file=montore,
   action=writesolid]
```

La chaîne de commandes `LaTeX->dvips->GSview (Windows)ou gv (Linux)` permet de compiler, puis de transformer en postscript pour enfin visualiser.

Cette dernière opération va créer 4 fichiers :

- `montore-sommets.dat` -> la liste des sommets ;
- `montore-faces.dat` -> la liste des faces ;
- `montore-couleurs.dat` -> les couleurs des faces ;
- `montore-io.dat` -> les bornes des indices des faces externes et internes.

3.1.2 Lecture de fichiers .dat

On utilise l'objet `[object=datfile]` de `\psSolid`, avec l'argument `file` pour spécifier le nom du fichier. Ainsi le code

```
\psSolid[object=datfile, file=montore]
```

va permettre d'utiliser l'objet stocké dans les fichiers .dat créés au paragraphe précédent.

3.2 Fichiers .obj

Ce format n'utilise qu'un seul fichier, et permet de spécifier sommets et faces. Ce sont des fichiers très utilisés dans le domaine de la 3D et qu'on trouve en abondance sur internet. Par contre, nous n'utilisons qu'une forme simplifiée du format obj. On veillera à supprimer tous les commentaires `#` du fichier original et ne conserver que les sommets : ce sont les lignes qui commencent par `v` et les faces, lignes commençant par `f`.

Les fichiers trop volumineux ne seront pas pris en compte car le nombre maximal d'éléments pour un tableau `postscript` est 65535. Donc il faut que le nombre de sommets soit inférieur à 21845.

3.2.1 Écriture de fichiers .obj

On utilise l'action `[action=writeobj` dans `\psSolid`, et on utilise l'option `file` pour spécifier le nom du fichier.

Par exemple, le code ci-dessous :

```
\psSolid[object=tore,
   file=montore,
   action=writeobj]
```

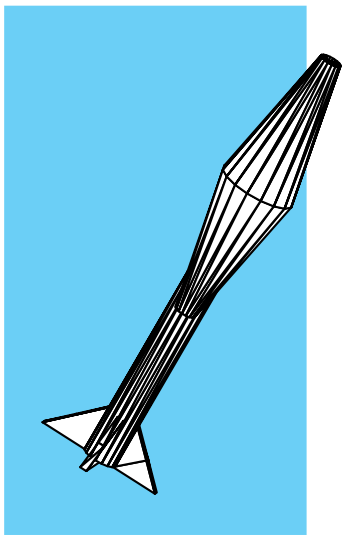
produira le fichier *montore.obj* (après compilation et visualisation du .ps produit).

3.2.2 Lecture de fichiers .obj

On utilise l'objet `[object=objfile` de `\psSolid`, avec l'argument `file` pour spécifier le nom du fichier. Ainsi le code

```
\psSolid[object=objfile, file=montore]
```

va permettre d'utiliser l'objet stocké dans les fichiers .obj créés au paragraphe précédent.



```
\psset{viewpoint=20 15 10 rtp2xyz,Decran=20}
\begin{pspicture}(-3,-4)(3,3)
\psframe*[linecolor=cyan!50](-3,-4)(1,3)
\psSolid[object=objfile,
        unit=20,RotX=60,
        file=rocket]%
\end{pspicture}
```

3.3 Fichiers .off

Nous n'utilisons qu'une forme simplifiée du format off. En particulier, les fichiers ne doivent comporter que des entrées `v` ou `f`.

Ce format n'utilise qu'un seul fichier, et permet de spécifier sommets et faces.

3.3.1 Écriture de fichiers .off

On utilise l'action `[action=writeoff]` dans `\psSolid`, et on utilise l'option `file` pour spécifier le nom du fichier.

Par exemple, le code ci-dessous :

```
\psSolid[object=tore,
   file=montore,
   action=writeoff]
```

produira le fichier *montore.off* (après compilation et visualisation du .ps produit).

3.3.2 Lecture de fichiers .off

On utilise l'objet `[object=offfile` de `\psSolid`, avec l'argument `file` pour spécifier le nom du fichier. Ainsi le code

```
\psSolid[object=offfile, file=montore]
```

va permettre d'utiliser l'objet stocké dans les fichiers .off créés au paragraphe précédent.

Chapitre 4

Quelques objets spécifiques

4.1 L'objet `plan`

4.1.1 Présentation : type *plan* et type *solid*

Le statut de l'objet `plan` est tout à fait particulier dans `pst-solides3d`. En effet, tous les objets vus jusqu'à présents ont une structure commune : ils sont de type *solid*. Autrement dit ils sont entièrement définis par une liste de sommets, de faces et de couleurs. Or pour de nombreuses applications, il est nécessaire d'avoir des renseignements complémentaires pour un plan : une origine, une orientation, une base de référence, etc...

Pour pouvoir répondre à ces exigences, il a été créé une autre structure de données, dite de type *plan*, qui permet de stocker toutes les informations nécessaires. Toutes les manipulations de plan vont transiter par un tel objet. Ce n'est qu'au moment de la représentation que l'objet de type *plan* sera converti en un objet de type *solid* représentable par la macro `\psSolid`.

Un objet de type *plan* permet donc de décrire une portion de plan affine orienté. Pour une définition complète d'un tel objet, il nous faut une origine I, une base vectorielle $(\vec{u},\vec{v})$ de ce plan, une étendue sur l'axe $(I,\vec{u})$ et une étendue sur l'axe $(I,\vec{v})$. De plus, on pourra spécifier le maillage souhaité, autrement dit le nombre de facettes utilisées pour représenter cette portion de plan affine lors de la transformation en objet de type *solid*.

Ce type d'objet peut être utilisé pour définir des sections planes, et il est obligatoire pour définir un plan de projection.

Son utilisation est transparente pour l'utilisateur PSTricks. La seule chose à savoir, c'est que lorsque l'on manipule un `[object=plan]` avec la macro `\psSolid`, on manipule en fait deux objets en même temps : l'un de type *plan* et l'autre de type *solid*. Et lorsque l'on demande une sauvegarde de cet objet (voir le chapitre «*Utilisation avancée*») sous le nom *monplan* par exemple avec l'option `[name=monplan]`, ce sont en fait 2 sauvegardes qui sont effectuées. La première, sous le nom *monplan*, est l'objet de type *plan*, et la deuxième, sous le nom *monplan_s*, est l'objet de type *solid*.

4.1.2 Définir un plan orienté

Pour créer un tel objet, on utilise `[object=plan]` qui utilise plusieurs arguments :

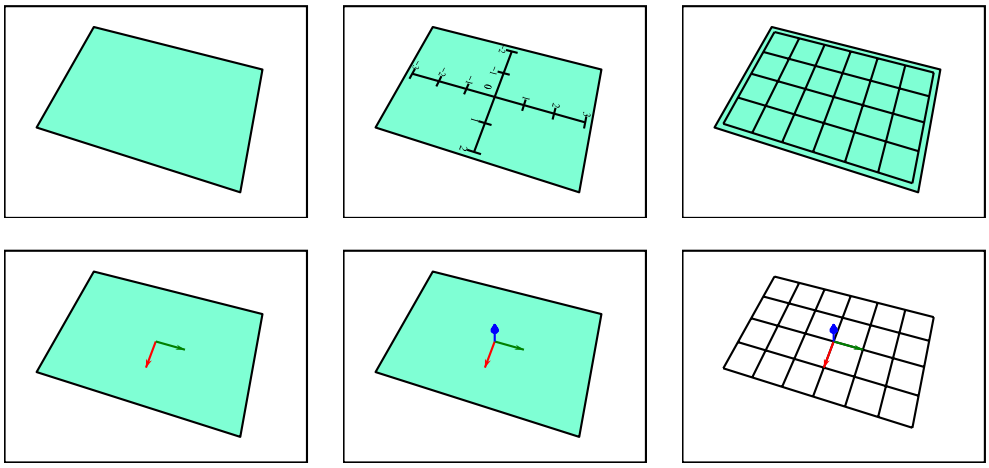
- `[definition=...]` qui permet de spécifier la méthode choisie pour définir le plan
- `[args=...]` qui permet de spécifier les arguments nécessaires à la méthode choisie précédemment
- `[bases=`*xmin xmax ymin ymax*`]` qui permet de spécifier l'étendue sur chacun des axes.
- `[phi]` (valeur `0` par défaut) qui spécifie l'angle de rotation (en degrés) du plan autour de sa normale après sa définition première.

4.1.3 Options spécifiques

L'objet `plan` possède plusieurs options spécifiques d'affichage : `[planmarks]` qui projette l'impression des axes et graduations, `[plangrid]` qui projette l'impression d'un quadrillage, `[showbase]` qui projette l'impression des

vecteurs de base du plan, et `[showBase]` (noter la majuscule) qui projette l'impression des vecteurs de base du plan et qui dessine le vecteur normal associé.

Ces options sont valables quelque soit le mode de définition choisi pour le plan.



Ces options peuvent être utilisées, même si le plan n'est pas représenté.

4.1.4 Définir un plan à partir d'une équation cartésienne

On appelle *équation cartésienne* d'un plan affine une équation du type

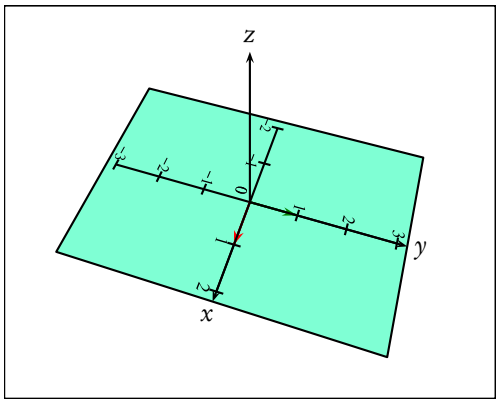
$$ax + by + cz + d = 0$$

La donnée des coefficients a, b, c et d permet de définir un plan affine.

Utilisation avec l'orientation et l'origine par défaut

Pour définir un plan affine, on peut utiliser `[definition=equation]`, et `[args={[a b c d]}]`. L'orientation et l'origine du plan affine sont alors choisis par le package.

Par exemple, le quadruplet $(a,b,c,d) = (0,0,1,0)$ désigne le plan d'équation $z = 0$:



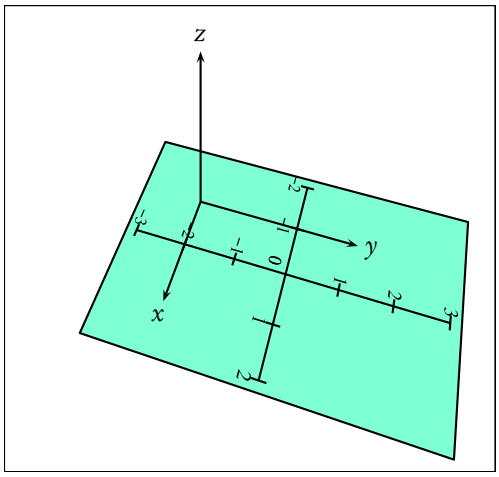

```
\psSolid[object=plan,
   definition=equation,
   args={[0 0 1 0]},
   fillcolor=Aquamarine,
   planmarks,
   base=-2.2 2.2 -3.2 3.2,
   showbase,
]
```

Le paramètre `[base=xmin xmax ymin ymax]` permet de spécifier l'étendue sur chacun des axes.

Spécification de l'origine

Le paramètre `[origine=`x_0 y_0 z_0`]` permet de spécifier l'origine du plan affine. Si le point (x_0, y_0, z_0) proposé ne vérifie pas l'équation du plan, alors on ne tient pas compte de cette origine.

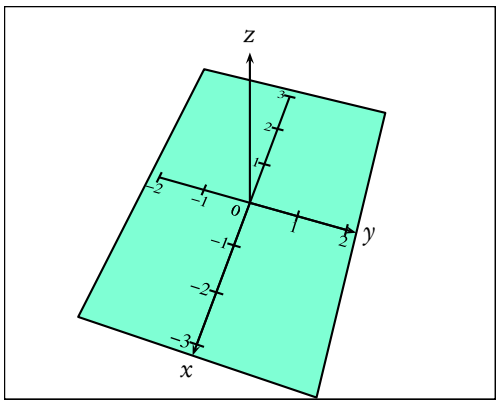
Par exemple, voici une représentation du plan d'équation $z = 0$ pour laquelle on a spécifié $(1,2,0)$ comme origine :




```
\psSolid[object=plan,
   definition=equation,
   args={[0 0 1 0]},
   origine=1 2 0,
   fillcolor=Aquamarine,
   base=-2.2 2.2 -3.2 3.2,
   planmarks,
]
```

Spécification de l'orientation

Si l'orientation proposée ne convient pas, on peut spécifier un angle de rotation α (en degrés) autour de la normale avec la syntaxe `[args={[a b c d] α}]`.




```
\psSolid[object=plan,
   definition=equation,
   args={[0 0 1 0] 90},
   fillcolor=Aquamarine,
   base=-2.2 2.2 -3.2 3.2,
   planmarks,
]
```

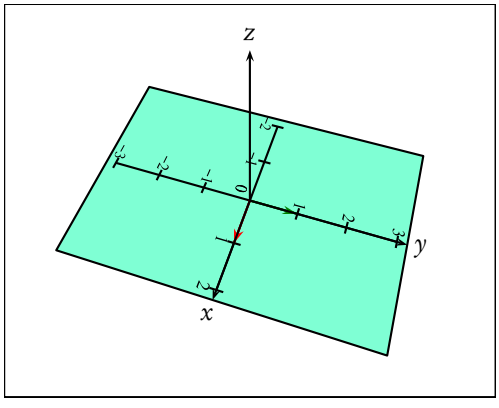
4.1.5 Définir un plan à partir d'un vecteur normal et d'un point

Il est possible de définir un plan affine à partir d'un point et d'un vecteur normal. On utilise pour cela le paramètre `[definition=normalpoint]`.

On peut ensuite préciser ou non l'orientation.

Méthode 1 : Sans préciser l'orientation

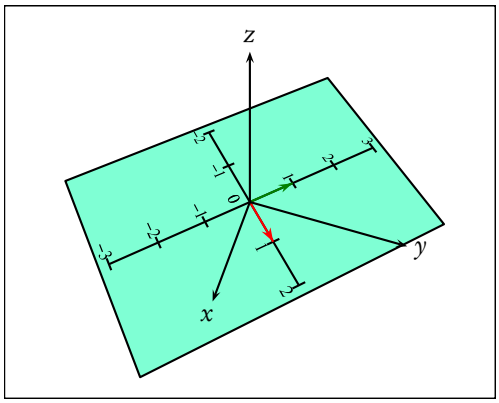
On utilise `[args={x0 y0 z0 [a b c]}]` où (x_0, y_0, z_0) est l'origine du plan affine, et (a, b, c) un vecteur normal à ce plan.




```
\psSolid[object=plan,
   definition=normalpoint,
   args={0 0 0 [0 0 1]},
   fillcolor=Aquamarine,
   planmarks,
   base=-2.2 2.2 -3.2 3.2,
   showbase,
]
```

Méthode 2 : préciser un angle de rotation

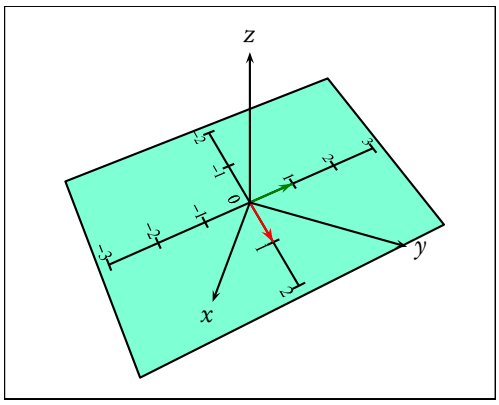
On utilise `[args={`x_0 y_0 z_0 `[`a b c α`]}]` où (x_0, y_0, z_0) est l'origine du plan affine, (a, b, c) un vecteur normal à ce plan, et α l'angle de rotation (en degrés) autour de l'axe normal.




```
\psSolid[object=plan,
   definition=normalpoint,
   args={0 0 0 [0 0 1 45]},
   fillcolor=Aquamarine,
   planmarks,
   base=-2.2 2.2 -3.2 3.2,
   showbase,
]
```

Méthode 3 : préciser le premier vecteur de la base

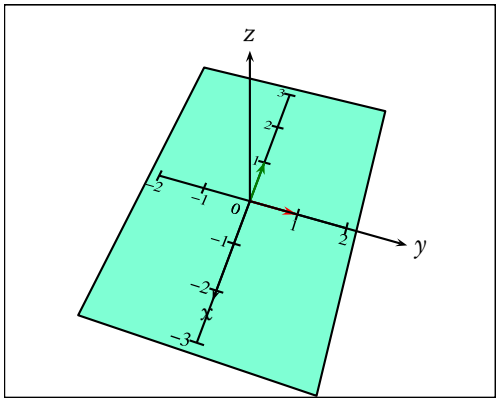
On utilise `[args={`x_0 y_0 z_0 `[`u_x u_y u_z a b c `]}]` où (x_0, y_0, z_0) est l'origine du plan affine, (a, b, c) un vecteur normal à ce plan, et (u_x, u_y, u_z) le premier vecteur d'une base de ce plan.




```
\psSolid[object=plan,
   definition=normalpoint,
   args={0 0 0 [1 1 0 0 0 1]},
   fillcolor=Aquamarine,
   planmarks,
   base=-2.2 2.2 -3.2 3.2,
   showbase,
]
```

Méthode 4 : préciser le premier vecteur de la base et un angle de rotation

On utilise `[args={`x_0 y_0 z_0 `[`u_x u_y u_z a b c α`]}]` où (x_0, y_0, z_0) est l'origine du plan affine, (a, b, c) un vecteur normal à ce plan, et (u_x, u_y, u_z) le premier vecteur d'une base de ce plan, auquel on fait subir une rotation de α (en degrés) autour de l'axe normal.

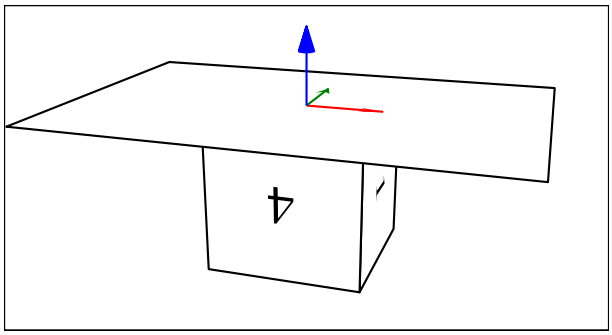



```
\psSolid[object=plan,
   definition=normalpoint,
   args={0 0 0 [1 1 0 0 0 1 45]},
   fillcolor=Aquamarine,
   planmarks,
   base=-2.2 2.2 -3.2 3.2,
   showbase,
]
```

4.1.6 Définition d'un plan à partir d'une face de solide

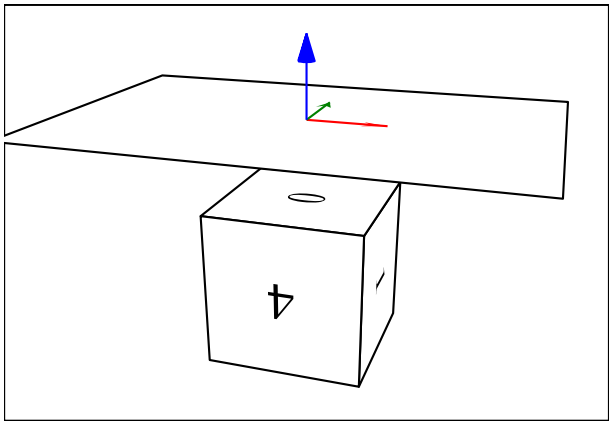
On utilise `[definition=solidface]` avec les arguments `[args=name i]` où *name* est un nom désignant le solide et i est l'indice de la face considérée. On prend pour origine le centre de la face considérée.

Dans l'exemple ci-dessous, on définit le plan par la face d'indice `0` du cube nommé A.



```
\psset{solidmemory}
\psSolid[object=cube,a=2,fontsize=20,numfaces=all,
   name=A]
\psSolid[object=plan,
   definition=solidface,
   args=A 0,
   showBase,
]
```

Si l'utilisateur précise les coordonnées (x,y,z) dans la macro `\psSolid[...](x,y,z)`, alors le plan construit est parallèle à la face d'indice i du solide *name*, et il passe par le point (x,y,z) qui est pris pour origine.

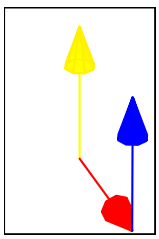


```
\psset{solidmemory}
\psSolid[object=cube,a=2,fontsize=20,numfaces=all,
   name=A]
\psSolid[object=plan,
   definition=solidface,
   args=A 0,
   showBase,
](0,0,2)
```

4.2 Vecteurs

4.2.1 Définition à partir des coordonnées

L'objet `vecteur` permet de définir un vecteur. Sous sa forme la plus simple, on utilise l'argument `[args=x y z]` pour en spécifier les coordonnées.



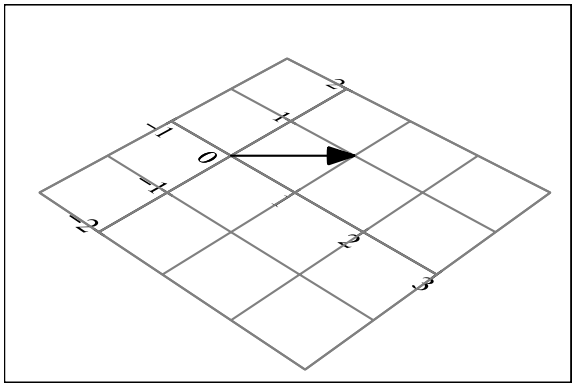
```
\psSolid[object=vecteur,
   args=0 0 1,
   linecolor=yellow]%
\psSolid[object=vecteur,
   args=1 0 0,
   linecolor=red]
\psSolid[object=vecteur,
   args=0 0 1,
   linecolor=blue](1,0,0)
```

4.2.2 Définition à partir de 2 points

On peut également définir un vecteur par la donnée de 2 points A et B de R^3. On utilise alors les arguments `[definition=vecteur3d]` et `[args=`x_A y_A z_A x_B y_B z_B`]` où (x_A,y_A,z_A) et (x_B,y_B,z_B) sont les coordonnées res-

pectives des points A et B

Si les points A et B ont été préalablement définis, alors on peut utiliser des variables nommées : `[args=A B]`.




```
\psSolid[object=vecteur,
   definition=vecteur3d,
   args=0 0 1 1 1 1]%
```

4.2.3 Autres modes de définition

Il existe d'autres possibilités pour définir un vecteur. Voici une liste des définitions possibles avec les arguments correspondant :

- `[definition=addv3d]` ; `args=` $\vec{u}\ \vec{v}$. addition de 2 vecteurs.
- `[definition=subv3d]` ; `args=` $\vec{u}\ \vec{v}$. différence de 2 vecteurs.
- `[definition=mulv3d]` ; `args=` $\vec{u}\ \lambda$. multiplication d'un vecteur par un réel.
- `[definition=vectprod3d]` ; `args=` $\vec{u}\ \vec{v}$. produit vectoriel de 2 vecteurs.
- `[definition=normalize3d]` ; `args=` $\vec{u}$. Renvoie le vecteur $||\vec{u}||^{-1}\vec{u}$.

4.3 Point

4.3.1 Définition à partir des coordonnées

L'objet `point` permet de définir un point. Sous sa forme la plus simple, on utilise l'argument `[args=x y z]` pour en spécifier les coordonnées. Si on a précédemment nommé M un point (x,y,z) (voir chapitre *Utilisation avancée*), on peut utiliser l'argument `[args=M]`.

4.3.2 Autres modes de définition

Il existe d'autres possibilités pour définir un point. Voici une liste des définitions possibles avec les arguments correspondant :

- `[definition=solidgetsommet]` ; `args=` $solik\ k$. Le sommet d'indice k du solid $solid$.
- `[definition=solidcentreface]` ; `args=` $solik\ k$. Le centre de la face d'indice k du solid $solid$.
- `[definition=isobarycentre3d]` `args=` $\{[\ A_0 \ldots A_n\ ]\}$ le barycentre du système $[(A_0,1);\ldots;(A_n,1)]$
- `[definition=barycentre3d]` `args=` $\{[\ A\ a\ B\ b\ ]\}$ le barycentre du système $[(A,a);(B,b)]$
- `[definition=hompoint3d]` `args=` $M\ A\ \alpha$ l'image de M par l'homothétie de centre A et de rapport α
- `[definition=sympoint3d]` `args=` $M\ A$ l'image de M par la symétrie de centre A
- `[definition=translatepoint3d]` `args=` $M\ u$ l'image de M par la translation de vecteur $\vec{u}$
- `[definition=scale0point3d]` `args=` $x\ y\ z\ k_1\ k_2\ k_3$ opère une « dilatation » des coordonnées du point $M(x,y,z)$ sur les axes Ox, Oy et Oz suivant les facteurs k_1, k_2 et k_3
- `[definition=rotate0point3d]` `args=` $M\ \alpha_x\ \alpha_y\ \alpha_z$ l'image de M par les rotations successives de centre O et d'angles respectifs $\alpha_x\ \alpha_y\ \alpha_z$ sur les axes Ox, Oy, Oz

- `[definition=orthoprojplane3d]` `args=` M A $\vec{v}$ Le projeté du point M sur le plan P défini par le point A et le vecteur $\vec{v}$, normal à P.
- `[definition=milieu3d]` `args=` A B Le milieu de $[AB]$
- `[definition=addv3d]` `args=` A u Le point B tel que $\overrightarrow{AB} = \vec{u}$

4.4 Les géodes et leurs duales

4.4.1 Présentation mathématique

D'excellentes études sur les géodes et leurs duales sont disponibles sur les sites suivants :

http://fr.wikipedia.org/wiki/G%C3%A9ode

Le paramétrage d'une géode est fidèle aux indications de la page :

http://hypo.ge-dip.etat-ge.ch/www/math/html/amch104.html

« *On peut définir une géode à partir de deux paramètres : un numéro N indiquant le type de polyèdre initial (*$N = 3$ *pour le tétraèdre,* $N = 4$ *pour l'octaèdre et* $N = 5$ *pour l'icosaèdre) et un nombre n indiquant le nombre de divisions le long de l'arête.* »

L'article *Indexing the Sphere with the Hierarchical Triangular Mesh* décrit une méthode permettant d'obtenir une représentation des géodes :

http://research.microsoft.com/research/pubs/view.aspx?msr_tr_id=MSR-TR-2005-123

4.4.2 Construction avec pst-solides3d

Deux approches sont possibles pour construire une géode ou sa duale : soit *via* `\codejps`, soit en utilisant les objets de `\psSolid`.

Pour une géode, les codes

```
\codejps{N n newgeode drawsolid**}
```

et

```
\psSolid[object=geode,ngrid=N n]
```

sont équivalents. Et pour sa duale, les codes Pour une géode, les codes

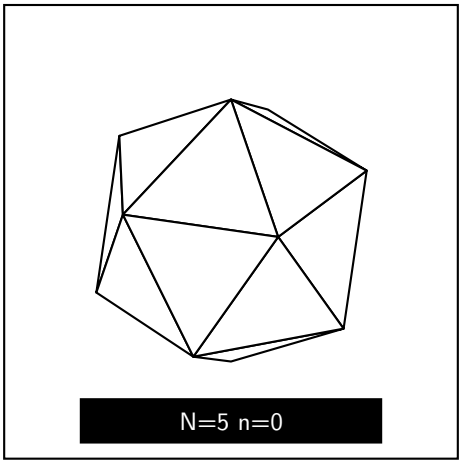
```
\codejps{N n newdualgeode drawsolid**}
```

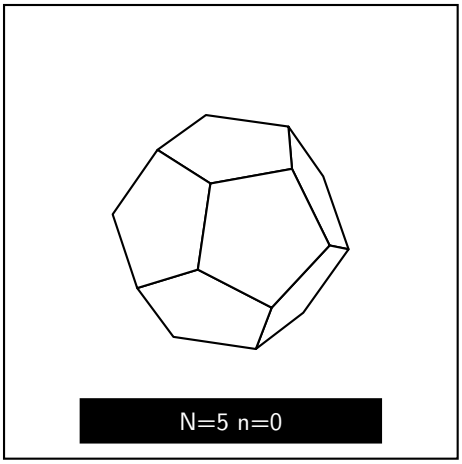
et

```
\psSolid[object=geode,dualreg,ngrid=N n]
```

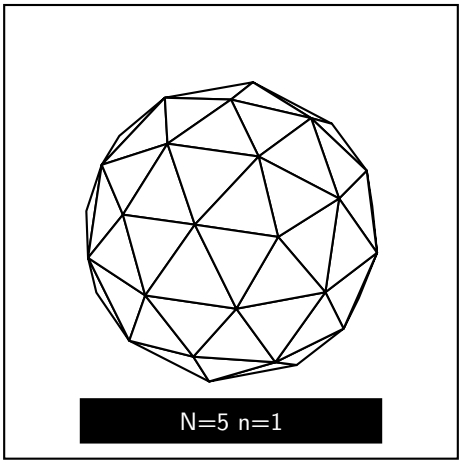
sont équivalents.

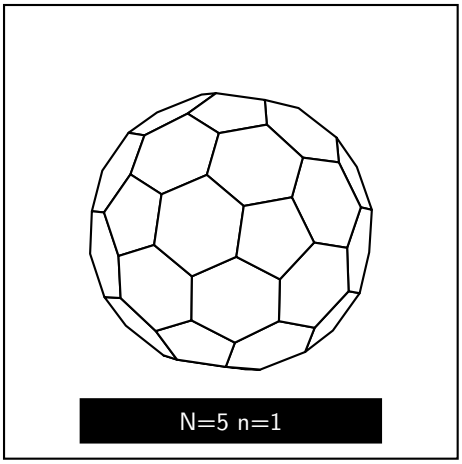
4.4.3 Quelques exemples de géodes et de duales






```
\psset{viewpoint=50 -20 30 rtp2xyz,Decran=100}
\begin{pspicture}(-3,-3)(3,3)
\psframe(-3,-3)(3,3)
\psSolid[object=geode,
   ngrid=5 0]
%\codejps{5 0 newgeode drawsolid**}
\psframe*(-2,-2.8)(2,-2.2)
\rput(0,-2.5){\textcolor{white}{\textsf{N=5 n=0}}}
\end{pspicture}
\hfill
\begin{pspicture}(-3,-3)(3,3)
\psframe(-3,-3)(3,3)
\psSolid[object=geode,
   dualreg,
   ngrid=5 0]
%\codejps{5 0 newdualgeode drawsolid**}
\psframe*(-2,-2.8)(2,-2.2)
\rput(0,-2.5){\textcolor{white}{\textsf{N=5 n=0}}}
\end{pspicture}
```






```
\psset{viewpoint=50 -20 30 rtp2xyz,Decran=100}
\begin{pspicture}(-3,-3)(3,3)
\psframe(-3,-3)(3,3)
\psSolid[object=geode,
   ngrid=5 1]
%\codejps{5 1 newgeode drawsolid**}
\psframe*(-2,-2.8)(2,-2.2)
\rput(0,-2.5){\textcolor{white}{\textsf{N=5 n=1}}}
\end{pspicture}
\hfill
\begin{pspicture}(-3,-3)(3,3)
\psframe(-3,-3)(3,3)
\psSolid[object=geode,
   dualreg,
   ngrid=5 1]
%\codejps{5 1 newdualgeode drawsolid**}
\psframe*(-2,-2.8)(2,-2.2)
\rput(0,-2.5){\textcolor{white}{\textsf{N=5 n=1}}}
\end{pspicture}
```

4.4.4 Les paramètres des géodes

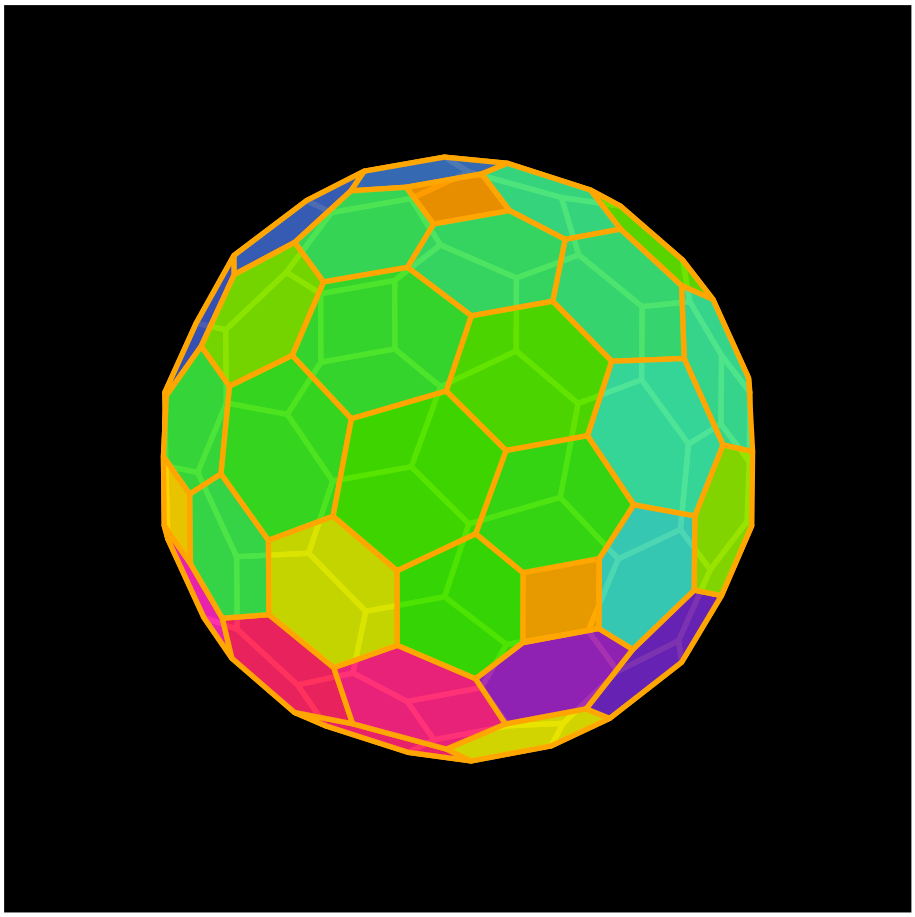
Le rayon de la sphère est fixé à 1, pour augmenter la taille des géodes on jouera sur l'un ou l'autre des deux paramètres suivants :

- l'unité : `\psset{unit=2}`
- la position de l'écran : `viewpoint=50 -20 30,Decran=100`, si la distance de l'écran est deux fois plus grande que la distance à laquelle se trouve l'observateur l'échelle de la scène est multipliée par 2.

En jps, le paramétrage s'effectue pour la géode dans le \codejps{**N n newgeode**} et pour sa duale dans \codejps{**N n newdualgeode**}.

Avec \psSolid, les paramètres N et n sont transmis via l'argument `ngrid`

Les options de couleurs et de transparence sont bien sûr possibles avec les géodes.



```
\psset{unit=2}
\psset{viewpoint=50 -20 30 rtp2xyz,Decran=100,linewidth=2pt}
\begin{pspicture}(-3,-3)(3,3)
\psframe*(-3,-3)(3,3)
\codejps{
/geode42{4 2 newdualgeode} def
.7 setfillopacity
orange
/geodetransparente{
geode42
dup videsolid
dup (orange) inputcolors
dup [.1 .9] solidputhuecolors} def
geodetransparente
drawsolid**}
\end{pspicture}
```

4.4.5 Conseils pour la construction 'rapide' des géodes

Le temps de calcul des géodes et de leurs duales dépend du nombre de divisions sur une arête (le deuxième paramètre n) et il devient rapidement très grand, ce qui est vraiment une gêne lorsqu'on est obligé d'attendre, plus ou moins patiemment, le résultat de la transformation `dvips->ps2pdf`.

Comme pour tous les autres solides, il est possible de sauvegarder la structure calculée dans des fichiers externes, ce qui permettra un gain de temps appréciable si on doit faire des essais de couleurs ou de point de vue.

Il faut opérer en deux étapes :

Sauvegarde en fichier `.dat` des paramètres de la géode

```
\documentclass{article}
\usepackage{pst-solides3d}
```

```
\begin{document}
\codejps{
4 4 newdualgeode
 dup {[.5 .6]} exec solidputhuecolors
(geodedual44) writesolidfile
}
\end{document}
```

`LaTeX->dvips->GSview (Windows)ou gv (Linux)`

Cette dernière opération va créer 4 fichiers :
- `geodedual44-couleurs.dat` -> les couleurs des faces ;
- `geodedual44-faces.dat` -> la liste des faces ;
- `geodedual44-sommets.dat` -> la liste des sommets ;
- `geodedual44-io.dat` -> le nombre de faces et de sommets.

Par défaut, sous Windows et Linux, la protection des fichiers du disque dur est activée et ne permet donc pas l'écriture sur le disque. Pour désactiver cette protection, tout au moins temporairement, voici les deux procédures correspondantes :
Linux : le conseil de Jean-Michel Sarlat : le plus simple est donc d'utiliser ghostscript directement, en console. Comme il n'y a rien à attendre comme image :
$> gs -dNOSAFER lissatest.ps quit.ps
Windows : dans le menu Options, l'option Protection des fichiers ne doit pas être cochée.

Lecture des données et dessin de la géode

L'avantage de cette méthode vous paraîtra plus évident en faisant la comparaison suivante : compilation de deux fichiers qui produisant le même résultat avec les deux méthodes en concurrence.

Le fichier `geode42_direct.tex` fait le calcul du solide et son affichage. Le fichier `geode42_precalcul.tex` utilise les fichiers `.dat` de données pré-calculées par `calc_geode42.tex`. Ces trois fichiers sont inclus dans la documentation.

4.4.6 D'autres exemples

Vous trouverez de nombreux autres exemples de géodes sur la page :
http://melusine.eu.org/lab/bpst/pst-solides3d/geodes

Chapitre 5

Fabriquer de nouveaux solides

5.1 Le code jps

Nous appelons *code jps* tout code postscript utilisant la bibliothèque développée pour le logiciel *jps2ps*.

Le fichier `solides.pro` du package `solides3d` est essentiellement constitué d'éléments en provenance de cette bibliothèque, qui contient environ 4 500 fonctions et procédures.

Son utilisation permet de disposer de commandes adaptées au dessin mathématique, sans qu'il soit besoin de tout reconstruire à partir des primitives `moveto`, `lineto`, `curveto`, etc...

Par exemple, on peut définir une fonction F telle que $F(t) = (3\cos^3 t, 3\sin^3 t)$, et demander le tracé de la courbe avec la code jps `0 360 F CourbeR2`.

Si on veut seulement le chemin de cette courbe, on utilise le code `0 360 {F} CourbeR2_`, et si on veut le dépot sur la pile des points de la courbe, on utilise `0 360 F CourbeR2+`.

Dans chacun des 3 exemples ci-dessus, le nombre de points est déterminé par la variable globale *resolution*.

Autrement dit, avec la fonction F précitée et une résolution fixée à 36, le code jps

```
0 350 {F} CourbeR2+
```

est équivalent au code postscript

```
0 10 350 {
   /angle exch def
   3 angle cos 3 exp mul
   3 angle sin 3 exp mul
} for
```

Nous n'avons pas encore développé la documentation sur la partie spécifique de cette bibliothèque embarquée dans le fichier `solides.pro`. Pour le moment, nous renvoyons le lecteur intéressé au *Guide de l'utilisateur de jps2ps* disponible sur le site `melusine.eu.org/syracuse/bbgraf`.

5.2 Définir une fonction

Il est possible de définir des fonctions utilisables dans l'environnement postscript. L'ensemble de départ peut être R, R^2 ou R^3, et l'ensemble d'arrivée peut être R, R^2 ou R^3.

La définition se fait avec la macro `\defFunction`. Cette macro nécessite six arguments, dont un seul est optionnel.

`\defFunction[<options>]{<nom>}(<var>){<x(var)>}{<y(var)>}{<z(var)>}`

Quand vous avez défini une fonction, cette fonction est toujours reprise avec son `<nom>` choisi.

Voilà quelques exemples :

- `\defFunction{moncercle}(t){t cos 3 mul}{0}{t sin 3 mul}`
  donne un cercle de rayon 3 dans le plan xOz (notation RPN).
- `\defFunction[algebraic]{helice}(t){cos(t)}{sin(t)}{t}`
  donne une hélice en notation algèbrique.
- `\defFunction[algebraic]{F}(t){t}{}{}` donne une fonction de **R** dans **R**
- `\defFunction[algebraic]{F}(t){t}{}{}` donne une fonction de **R** dans $\mathbf{R}^2$

`<options>`	On y insère les option typiques de PSTricks, comme `linewidth` etc., et en plus, quelques unes définies par `pst-solides3d`. Une très charmante option est `algebraic`, avec quelle on peut éviter la notation RPN (Reverse Polish Notation). Toutes options sont des paires (clé,valeur) et sont séparées avec des virgules.
`<nom>`	C'est un nom unique de votre choix – mais attention : évitez des noms avec des accents, PostScript ne les aime pas du tout.
`<var>`	On y insère au maximum trois variables arbitraires, séparées avec des virgules.
`<x(var)>` `<y(var)>` `<z(var)>`	On y met des fonctions dépendant des variables définies pour les directions euclidienness x, y, z. Si une de ces trois directions n'est pas voulue, insérez un `0` entre les parenthèses – ce qui vous donne la possibilité de définir aussi des projetés plans de courbes de fonctions.

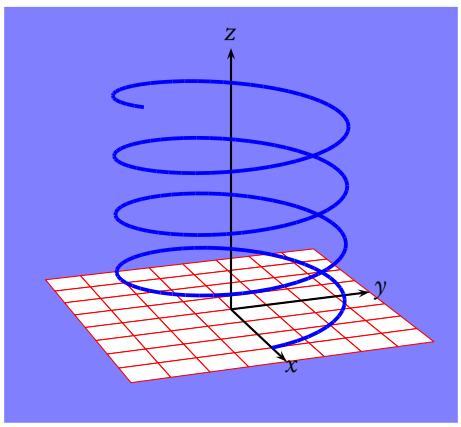
– `\defFunction[algebraic]{F}(t){t}{t}{t}` donne une fonction de $\mathbf{R}$ dans $\mathbf{R}^3$

Il nous reste encore du travail à faire sur cette macro, et elle ne permet pour le moment pas de choisir des noms de variables quelconques, car ils risquent d'entrer en conflit avec des noms déjà existant. Merci d'utiliser des noms analogues à ceux utilisés dans la documentation. Une bonne stratégie consiste à utiliser systématiquement un ou plusieurs caractères numériques à la fin de vos noms de variables.

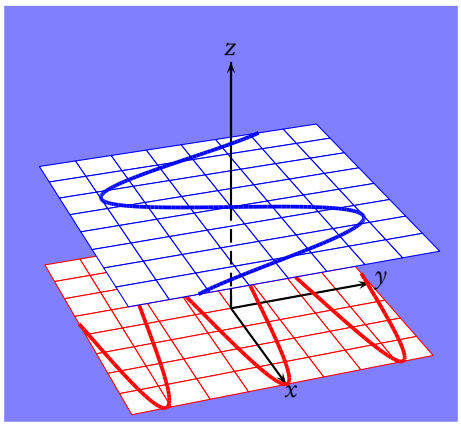
5.3 Courbes de fonctions de R vers R^3

Le tracé de la fonction ainsi définie fait appel à l'objet `courbe` et à l'option `function`.
On pourra réaliser le tracé de l'hélice :
en notation algébrique avec la fonction :

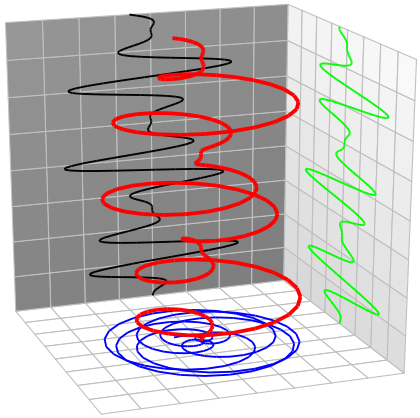
```
\defFunction[algebraic]{helice}(t){3*cos(4*t)}{3*sin(4*t)}{t}
```




```
\defFunction[algebraic]%
   {helice}(t){cos(3*t)}{sin(3*t)}{t}
\psSolid[object=courbe,
        range=0 6,
        r=0,
        linecolor=blue,
        resolution=360,
        function=helice]%
```




```
\psset{range=-4 4}
\psSolid[object=grille,base=-4 4 -4 4]%
\defFunction{CosRad}(t){t 2 mul Cos 4 mul}{t}{0}
\psSolid[object=courbe,linewidth=0.1,
        r=0,
        linecolor=red,
        resolution=360,
        function=CosRad]%
\defFunction{sinRad}(t){t}{t Sin 3 mul}{3}
\psSolid[object=grille,base=-4 4 -4 4](0,0,3)
\psSolid[object=courbe,
        r=0,
        linecolor=blue,
        resolution=360,
        function=sinRad]
```



```
\psSolid[object=grille,base=-4 4 -4 4]%
\psSolid[object=grille,base=-4 4 0 8](0,4,0)
\psSolid[object=grille,base=-4 4 -4 4](-4,0,4)
\defFunction[algebraic]{helice}%
   (t){1.3*(1-cos(2.5*t))*cos(6*t)}
   {1.3*(1-cos(2.5*t))*sin(6*t)}{t}
\defFunction[algebraic]{helice_xy}%
   (t){1.3*(1-cos(2.5*t))*cos(6*t)}
   {1.3*(1-cos(2.5*t))*sin(6*t)}{0}
\defFunction[algebraic]{helice_xz}%
   (t){1.3*(1-cos(2.5*t))*cos(6*t)}{4}{t}
\defFunction[algebraic]{helice_yz}%
   (t){-4}{1.3*(1-cos(2.5*t))*sin(6*t)}{t}
\psset{range=0 8}
\psSolid[object=courbe,
        r=0,
        linecolor=blue,
        linewidth=0.05,
        resolution=360,
        normal=0 0 1,
        function=helice_xy]%
\psSolid[object=courbe,
        r=0,
        linecolor=green,
        linewidth=0.05,
        resolution=360,
        normal=0 0 1,
        function=helice_xz]%
\psSolid[object=courbe,
        r=0,
        linewidth=0.05,
        resolution=360,
        normal=0 0 1,
        function=helice_yz]%
\psSolid[object=courbe,
        r=0,
        linecolor=red,
        linewidth=0.1,
        resolution=360,
        function=helice]%
```

Ces dernières courbes se trouvent sous forme d'animations sur la page :

http://melusine.eu.org/syracuse/pstricks/pst-solides3d/animations/

5.4 Tubes

Il s'agit de remplacer une courbe en deux ou trois dimensions (2D ou 3D) définie paramétriquement, par un tube dont la courbe initiale constituera l'axe et dont on pourra choisir le rayon et le quadrillage. On trouve des éléments mathématiques concernant ces objets sur les deux sites suivants :

http://fr.wikipedia.org/wiki/Tube_(math%C3%A9matiques)
http://www.mathcurve.com/surfaces/tube/tube.shtml

Comme à l'habitude, le package `pst-solides3d` offre deux possibilités pour dessiner les tubes :

- via `'PSTricks'` et l'argument `object` de `\psSolid`
- directement avec `\codejps`

Il est souvent préférable de calculer préalablement, à la main ou bien avec un logiciel de calcul formel, la dérivée première des fonctions paramétriques définissant les coordonnées.

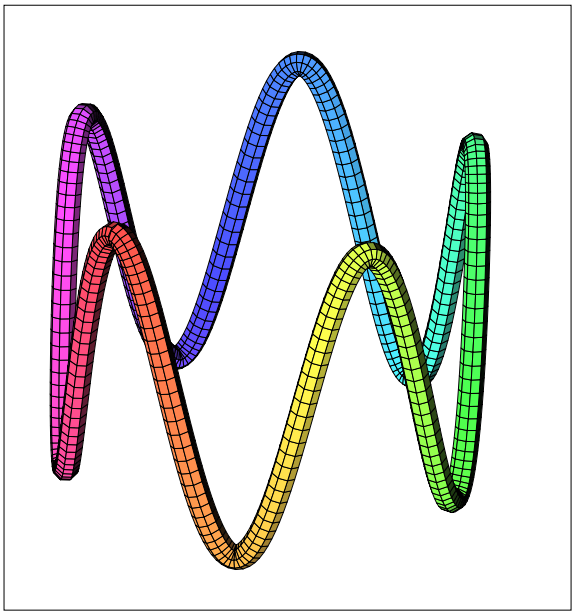
En effet, si cette dérivée n'est pas explicitement définie par l'utilisateur, le package fait des calculs approchés, mais le résultat n'est pas toujours satisfaisant.

5.4.1 Utilisation avec PSTricks

Donnez du relief à vos courbes

«*Donnez du relief à vos courbes*», c'est l'intitulé de l'article de Robert FERRÉOL sur :

http://mapage.noos.fr/r.ferreol/atelecharger/textes/relief/courbes%20en%20relief.html
à qui j'emprunte la courbe suivante qui est l'analogue d'une courbe de Lissajous enroulée autour d'un cylindre.



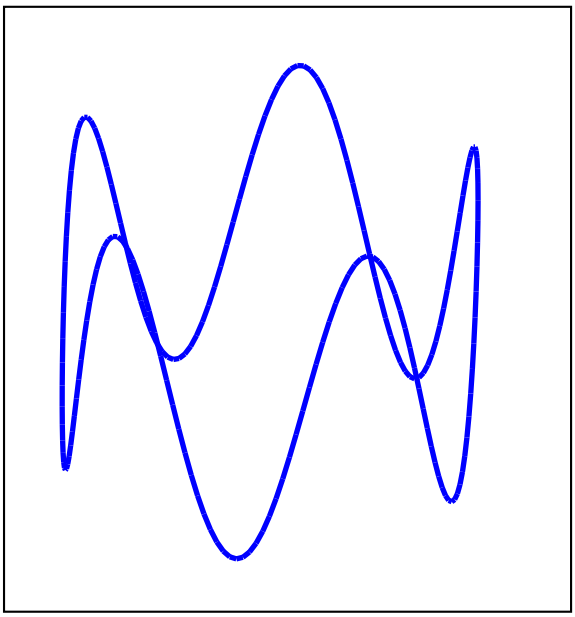
```
\begin{pspicture}(-3.5,-4)(4,4)
\psset{lightsrc=80 30 30,viewpoint=100 45 30 rtp2xyz,
  Decran=110,linewidth=0.2pt}
\psframe(-3.5,-4)(4,4)
\defFunction[algebraic]{F}(t)
  {2.5*cos(t)}
  {2.5*sin(t)}
  {2*cos(5*t)}
\defFunction[algebraic]{F'}(t)
  {-2.5*sin(t)}
  {2.5*cos(t)}
  {-10*sin(5*t)}
\psSolid[object=courbe,
  range=0 6.28,
  hue=0 1 0.7 1,
  ngrid=360 8,
  function=F,
  r=0.15]
\end{pspicture}
```

On a utilisé l'argument `[objet=courbe]` avec les paramètres `[r=]`, `[function=]` et `[range=]` pour spécifier respectivement le rayon du tube, le nom de la fonction à utiliser et l'intervalle de définition de la fonction.

On peut également préciser le maillage avec l'argument optionnel `[ngrid=`n_1 n_2`]` où n_2 représente le nombre de sommets sur une section du tube (si $n_2 = 6$, on a un tube à section hexagonale) et n_1 représente le nombre de divisions sur la longueur.

La courbe filaire s'obtient avec un rayon nul `[r=0]`

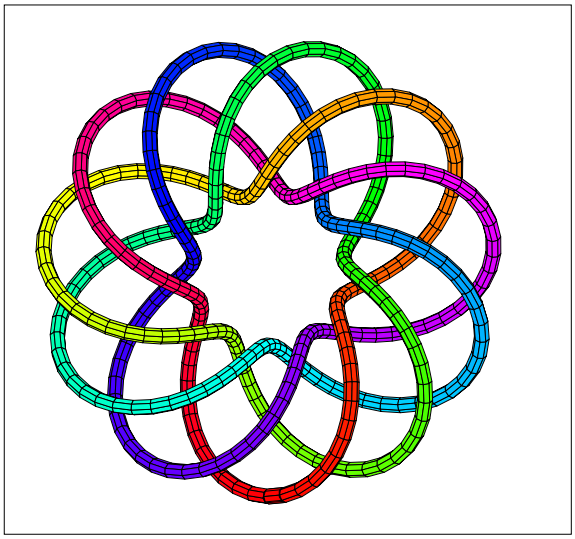
Et du coup, pas la peine de spécifier la fonction dérivée.



```
\begin{pspicture}(-3.5,-4)(4,4)
\psset{lightsrc=80 30 30,
  viewpoint=100 45 30 rtp2xyz,Decran=110}
\psframe(-3.5,-4)(4,4)
\defFunction[algebraic]{F}(t)
  {2.5*cos(t)}
  {2.5*sin(t)}
  {2*cos(5*t)}
\psSolid[object=courbe,
  range=0 6.28,
  linewidth=2pt,linecolor=blue,
  function=F,
  r=0]
\end{pspicture}
```

5.4.2 Utilisation avec le \codejps

La syntaxe est `\codejps{t1 t2 (nom_fonction) rayon [n1 n2] newtube}.`



```
\begin{pspicture}(-3.5,-3.5)(4,3.5)
\psset{lightsrc=80 30 30,viewpoint=100 45 90 rtp2xyz,Decran=100,linewidth=0.2pt}
\psframe(-3.5,-3.5)(4,3.5)
\codejps{
/rpn {tx@AlgToPs begin AlgToPs end cvx exec} def
/xc {((2+1*cos(2.75*t))*cos(t)) rpn } def
/yc {((2+1*cos(2.75*t))*sin(t)) rpn } def
/zc {(1*sin(2.75*t)) rpn } def
/xc' {(-2.75*sin(2.75*t)*cos(t)-(2+cos(2.75*t))*sin(t)) rpn } def
/yc' {(-2.75*sin(2.75*t)*sin(t)+(2+cos(2.75*t))*cos(t)) rpn } def
/zc' {(2.75*cos(2.75*t)) rpn } def
/g {
3 dict begin
  /t exch def
xc yc zc
end } def
/g' { % dérivée première
3 dict begin
  /t exch def
xc' yc' zc'
end } def
/solenoide{
% t_min t_max rayon_tube [resolution]
  0 25.2 (g) 0.1 [360 8] newtube
dup [0 1] solidputhuecolors} def
solenoide
drawsolid**
}%
\end{pspicture}
```

5.4.3 Améliorer la rapidité d'affichage

La courbe étudiée appelée « *horoptère* » est issue de la page :

http://www.mathcurve.com/courbes3d/horoptere/horoptere.shtml

L'obtention directe de la courbe

Les lignes suivantes permettent de faire le calcul des points et le tracé de la courbe. La résolution de la courbe étant élevée, le calcul prend un temps que certains jugeront trop long.

```
\begin{pspicture}(-7,-2)(7,4)
\psset{lightsrc=80 30 30}
\psset{viewpoint=1000 60 20 rtp2xyz,Decran=1000}
\psframe(-7,-2)(7,4)
\psset{solidmemory}
```

```
\codejps{/a 2 def /b 2 def}%
\defFunction[algebraic]{F}(t)
   {a*(1+cos(t))}
   {b*tan(t/2)}
   {a*sin(t)}
\defFunction[algebraic]{F'}(t)
   {-a*sin(t)}
   {b*(1+tan(1/2*t)^2)}
   {a*cos(t)}
\psSolid[object=courbe,
   range=-2.7468 2.7468,
   ngrid=72 12,
   function=F,hue=0 1 0.7 1,
   action=none,name=H1,
   r=1]%
\psSolid[object=cylindrecreux,
         h=20,r=1,RotX=90,
         incolor=green!30,action=none,
         name=C1,
         ngrid=36 36](2,10,0)
\psSolid[object=fusion,
   base=H1 C1]
\composeSolid
\end{pspicture}
```

La mise en mémoire des paramètres de la courbe

Si cete courbe doit être utilisée plusieurs fois il est alors préférable d'enregistrer toutes les caractéristiques de cette courbe : coordonnées de sommets, liste et couleurs des facettes en remplaçant la dernière commande par :

```
\psSolid[object=fusion,
         base=H1 C1,
         file=horoptere,
         action=writesolid]
```

La séquence suivante `LaTeX fichier.tex->dvips->GSview (Windows)ou gv (Linux)` va créer 4 fichiers :

- `horoptere-couleurs.dat` -> les couleurs des faces ;
- `horoptere-faces.dat` -> la liste des faces ;
- `horoptere-sommets.dat` -> la liste des sommets ;
- `horoptere-io.dat` -> le nombre de faces et de sommets.

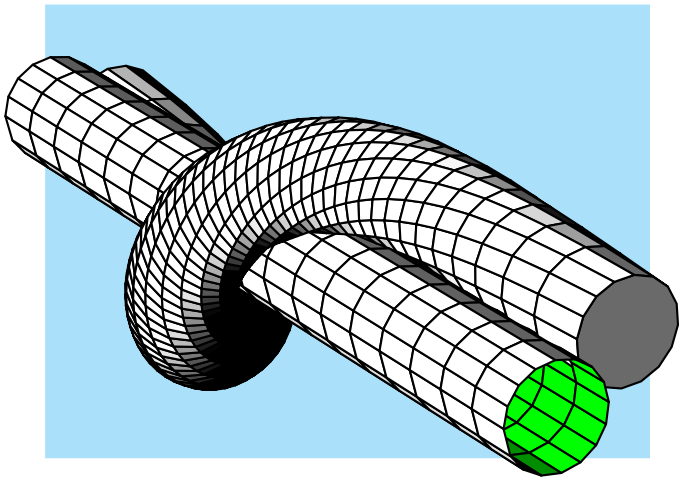
Par défaut, sous Windows et Linux, la protection des fichiers du disque dur est activée et ne permet donc pas l'écriture sur le disque. Pour désactiver cette protection, tout au moins temporairement, voici les deux procédures correspondantes :
Linux : le conseil de Jean-Michel Sarlat : le plus simple est donc d'utiliser ghostscript directement, en console. Comme il n'y a rien à attendre comme image :
$> gs -dNOSAFER fichier.ps quit.ps
Windows : dans le menu Options, l'option Protection des fichiers ne doit pas être cochée.

puis de les faire lire et exécuter avec la commande : `\psSolid[object=datfile,file=horoptere]`, le gain de temps est très appréciable !

```
\begin{pspicture}(-7,-2)(7,4)
\psset{lightsrc=80 30 30}
\psset{viewpoint=1000 60 20 rtp2xyz,Decran=1000}
\psframe(-7,-2)(7,4)
```

```
\psset{solidmemory}
\codejps{/a 2 def /b 2 def}%
\defFunction[algebraic]{F}(t)
   {a*(1+cos(t))}
   {b*tan(t/2)}
   {a*sin(t)}
\defFunction[algebraic]{F'}(t)
   {-a*sin(t)}
   {b*(1+tan(1/2*t)^2)}
   {a*cos(t)}
\psSolid[object=courbe,
   range=-2.7468 2.7468,
   ngrid=72 16,
   function=F,hue=0 1 0.7 1,
   action=none,name=H1,
   r=1]%
\psSolid[object=cylindrecreux,
         h=18,r=1,RotX=90,
         incolor=yellow!50,action=none,
         name=C1,
         ngrid=18 16](2,9,0)
\psSolid[object=fusion,
         base=H1 C1,
         file=horoptere,
         action=writesolid]
\composeSolid
\end{pspicture}
```

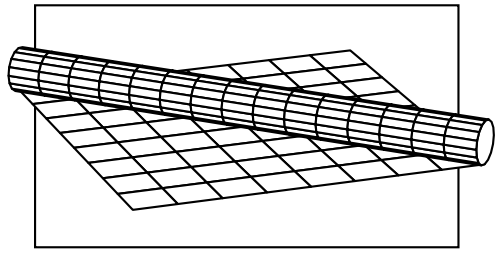
Le tracé de la courbe



```
\begin{pspicture}(-5,-3.5)(4,3)
\psset{lightsrc=80 30 30}
\psset{viewpoint=100 60 20 rtp2xyz,
       Decran=75}
\psframe*[linecolor=cyan!30](-4.5,-3)(3.5,3)
\psSolid[object=datfile,file=horoptere]
\end{pspicture}
```

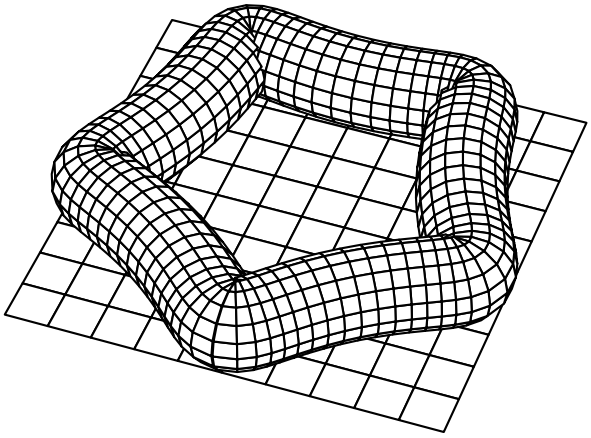
5.4.4 Autres exemples

Une droite



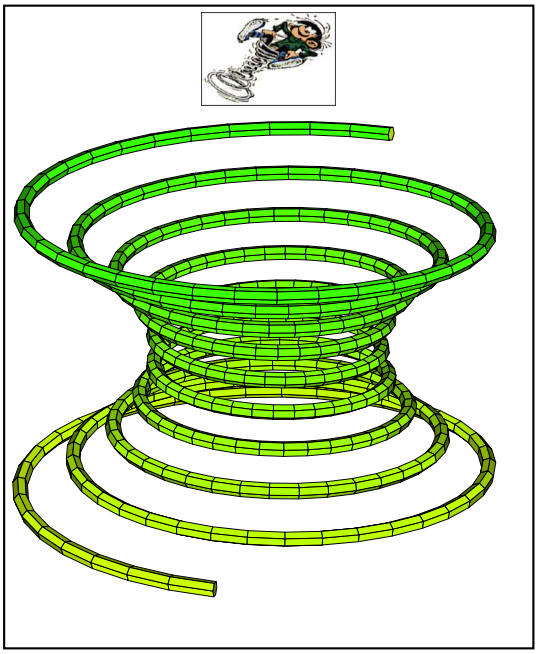
```
\begin{pspicture}(-3.5,-2)(3.5,2)
\psset{viewpoint=100 -20 20 rtp2xyz,
       Decran=75,unit=0.8}
\psframe(-3.5,-2)(3.5,2)
\psSolid[object=grille,base=-4 4 -4 4]%
\defFunction[algebraic]{F}(t){t}{t}{0.5}
\defFunction[algebraic]{F'}(t){1}{1}{0}
\psSolid[object=courbe,
   range=-4 4, ngrid=16 16,
   function=F, r=0.5]
\end{pspicture}
```

Hypocycloïde



```
\begin{pspicture}(-3.5,-3)(3.5,3)
\psset{viewpoint=100 20 45 rtp2xyz,
       Decran=75,unit=0.8}
\psSolid[object=grille,base=-5 5 -5 5]%
\defFunction[algebraic]{F}(t)
   {4*cos(t) + cos(4*t)/2}
   {4*sin(t) - sin(4*t)/2}
   {1}
\defFunction[algebraic]{F'}(t)
   {-4*sin(t)-2*sin(4*t)}
   {4*cos(t)-2*cos(4*t)}
   {0}
\psSolid[object=courbe,
   range=0 6.28, ngrid=90 16,
   function=F, r=1]
\end{pspicture}
```

Siège ressort de Gaston

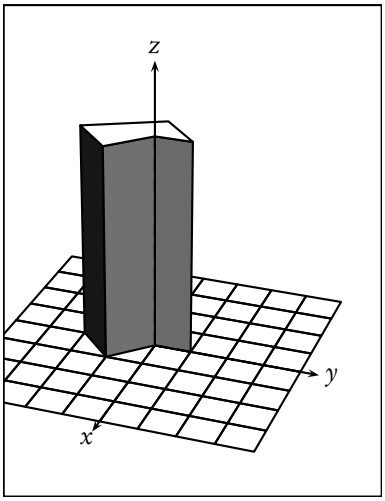


```
\begin{pspicture}(-3.5,-4)(3.5,4.5)
\psset{lightsrc=80 30 30}
\psset{viewpoint=100 20 20 rtp2xyz,Decran=50}
\psframe(-3.5,-4)(3.5,4.5)
\uput[u](0,3){\includegraphics[scale=0.25]{gaston.eps}}
\defFunction[algebraic]{F}(t)
        {(t^2+3)*sin(15*t)}
        {(t^2+3)*cos(15*t)}
        {2*t}
\defFunction[algebraic]{F'}(t)
        {2*t*sin(15*t)+15*(t^2+3)*cos(15*t)}
        {2*t*cos(15*t)-15*(t^2+3)*sin(15*t)}
        {2}
\psSolid[object=courbe,
   range=-2 2,
   ngrid=360 6,
   function=F,
   hue=0.2 0.3,
   linewidth=0.1pt,
   r=0.2]
\end{pspicture}
```

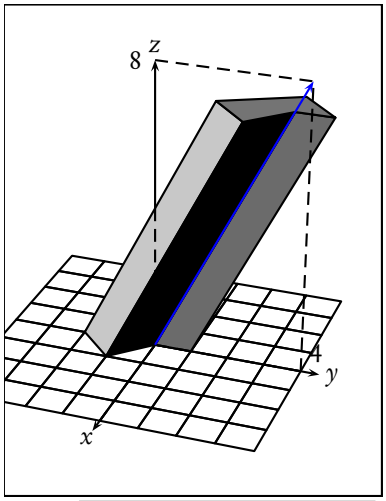
5.5 Le prisme

Deux paramètres sont propres au prisme :
- La base du prisme peut-être définie librement par les coordonnées des sommets dans le plan Oxy. Attention, il est nécessaire que les quatres premiers sommets soient rangés dans le sens trigonométrique par rapport à l'isobarycentre des sommets de cette base ;
- la direction de l'axe du prime par les coordonnées du vecteur directeur.

Exemple 1 : prisme droit et prisme oblique à section polygonale

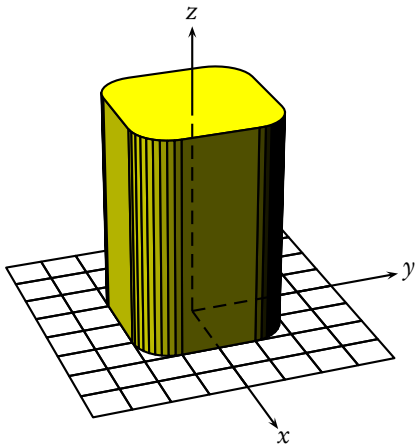



```
[base= 0 1 -1 0 0 -2 1 -1 0 0 ,h=6]
```



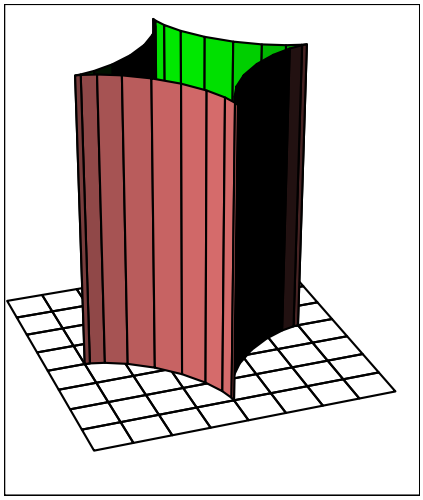

```
[base= 0 -2 1 -1 0 0 0 1 -1 0 ,
axe= 0 4 8 ,h=8]
```

Exemple 2 : prisme droit à section carrée arrondie



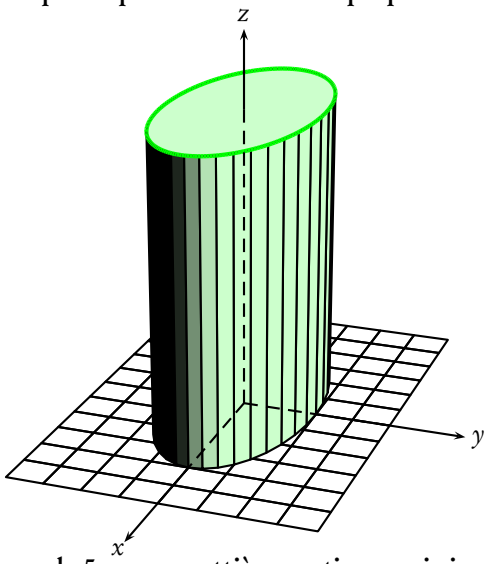

```
\psSolid[object=grille,base=-4 4 -4 4,action=draw]%
\psSolid[
   object=prisme,h=6,fillcolor=yellow,%
   base=%
     0 10 90 {/i exch def i cos 1 add i sin 1 add } for
     %
     90 10 180 {/i exch def i cos 1 sub i sin 1 add} for
     %
     180 10 270 {/i exch def i cos 1 sub i sin 1 sub} for
     %
     270 10 360 {/i exch def i cos 1 add i sin 1 sub} for
   ]%
 \axesIIID(4,4,6)(6,6,8)
```

Exemple 3 : prisme droit creux à section astroïdale



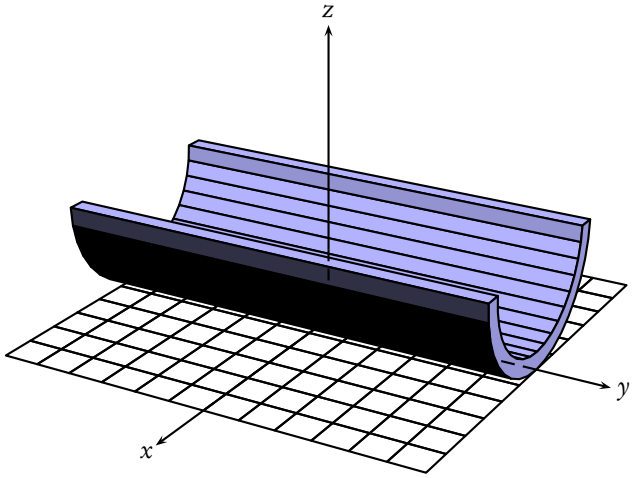
```
\defFunction{F}(t)
   {3 t cos 3 exp mul}{3 t sin 3 exp mul}{}
\psSolid[object=grille,base=-4 4 -4 4,action=draw]%
\psSolid[object=prismecreux,h=8,fillcolor=red!50,
       resolution=36,
       base=0 350 {F} CourbeR2+
      ]%
```

Exemple 4 : prisme à section elliptique




```
\defFunction{F}(t){t cos 4 mul}{t sin 2 mul}{}
\psSolid[object=grille,base=-6 6 -4 4,action=draw]%
\psSolid[object=prisme,h=8,fillcolor=green!20,%
       base=0 350 {F} CourbeR2+]%
\defFunction{F}(t){t cos 4 mul}{t sin 2 mul}{8}
\psSolid[object=courbe,
   r=0,
   function=F,range=0 360,
   linewidth=2\pslinewidth,
   linecolor=green]
```

Exemple 5 : une gouttière, section semi-circulaire à plat




```
\defFunction[algebraic]{F}(t)
   {3*cos(t)}{3*sin(t)}{}
\defFunction[algebraic]{G}(t)
   {2.5*cos(t)}{2.5*sin(t)}{}
\psSolid[object=grille,
  base=-6 6 -6 6,action=draw]%
\psSolid[object=prisme,h=12,
   fillcolor=blue!30,RotX=-90,
   resolution=19,
   base=0 pi {F} CourbeR2+
        pi 0 {G} CourbeR2+
      ](0,-6,3)
\axesIIID(6,6,2)(8,8,10)
```

On dessine d'abord la face extérieure (demi-cercle de rayon 3 cm), en tournant dans le sens trigonométrique : `0 pi F CourbeR2+`

Puis la face intérieure (demi-cercle de rayon 2,5 cm), en tournant cette fois dans le sens inverse du sens trigonométrique : `pi 0 G CourbeR2+`

On fait tourner le solide de -90^o en le plaçant au point $(0,-6,3)$.

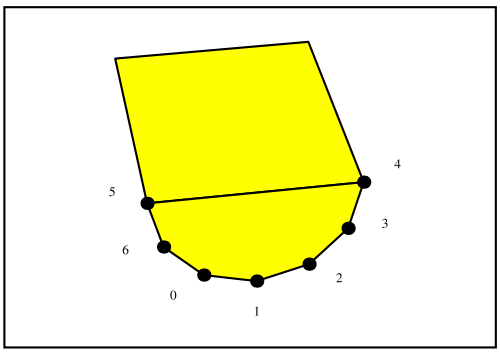

Comme on a utilisé l'option `algebraic` pour la définition des fontions F et G, les fonctions sin et cos utilisées fonctionnent en radian.

Le paramètre `decal`

Nous avons écrit plus haut qu'il était nécessaire que les quatres premiers sommets soient rangés dans le sens trigonométrique par rapport à l'isobarycentre des sommets de cette base. En fait, c'est la règle du comportement par défaut car la règle véritable est celle-ci : Si la base comporte $n+1$ sommets $(s_0, s_1, s_2, \ldots, s_{n-1}, s_n)$, et si G est l'isobarycentre des sommets, alors (s_0, s_1) d'une part, et (s_{n-1}, s_n) d'autre part, doivent être rangés dans le sens trigonométrique par rapport à G.

Cette règle induit des contraintes sur le codage de la base du prisme, rendant parfois ce dernier inesthétique. C'est pourquoi nous avons introduit l'argument `[decal]` (valeur par défaut$=-2$) qui permet de considérer la liste des sommets de la base comme une file circulaire que l'on décalera au besoin.

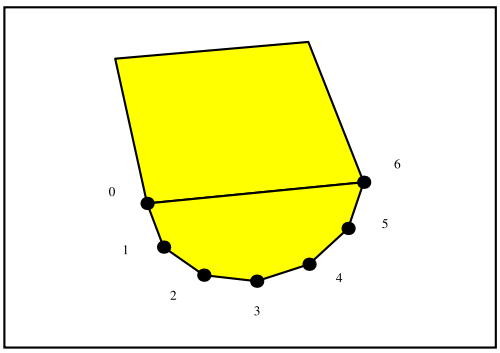
Un exemple : comportement par défaut avec $decal = -2$:




```
\defFunction{F}(t){t cos 3 mul}{t sin 3 mul}{}
\psSolid[object=prisme,h=8,
      fillcolor=yellow,RotX=-90,
      num=0 1 2 3 4 5 6,
      show=0 1 2 3 4 5 6,
      resolution=7,
      base=0 180 {F} CourbeR2+
      ](0,-10,0)
```

On voit que le sommet d'indice `0` n'est pas là où on s'attendrait à le trouver.

Recommençons, mais cette fois-ci en supprimant le décalage :

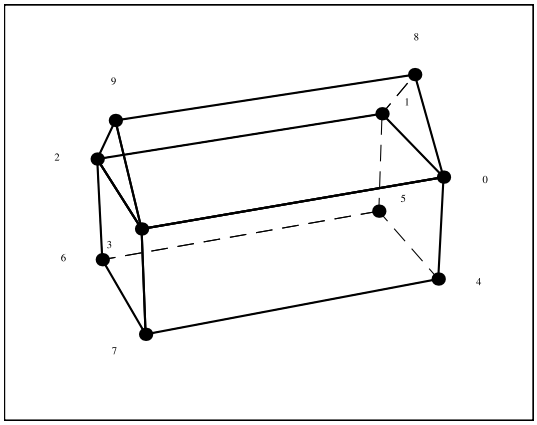



```
\defFunction{F}(t){t cos 3 mul}{t sin 3 mul}{}
\psSolid[object=prisme,h=8,
      fillcolor=yellow,RotX=-90,
      decal=0,
      num=0 1 2 3 4 5 6,
      show=0 1 2 3 4 5 6,
      resolution=7,
      base=0 180 {F} CourbeR2+
      ](0,-10,0)
```

5.6 Construire à partir du scratch

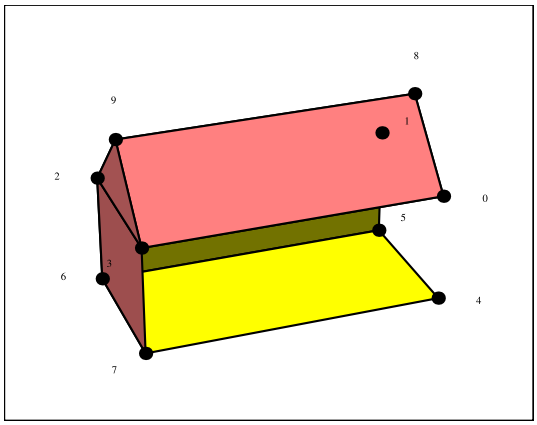
L'objet `new` permet de construire son propre solide. Deux paramètres sont utilisés : `sommets` qui indique la liste des coordonnées des différents sommets, et `faces` qui donne la liste de toutes les faces du solide, une face de solide étant caractérisée par la liste des indices des sommets la constituant, ceux-ci étant **rangés dans le sens trigonométrique lorsque l'on regarde la face du coté extérieur**.

5.6.1 Exemple 1 : une maison




```
\psSolid[object=new,
    sommets=
       2  4  3    -2  4  3
      -2 -4  3     2 -4  3
       2  4  0    -2  4  0
      -2 -4  0     2 -4  0
       0  4  5     0 -4  5,
    faces={
      [0 1 2 3]    [7 6 5 4]
      [0 3 7 4]    [3 9 2]
      [1 8 0]      [8 9 3 0]
      [9 8 1 2]    [6 7 3 2]
      [2 1 5 6]},
    num=all,show=all,action=draw]
```

Il est à remarquer que le solide `new` accepte les mêmes options que les autres solides. Par exemple, on a représenté ci-dessous le solide précédent en utilisant les paramètres `hollow`, `incolor`, `fillcolor` et `rm`.




```
\psSolid[object=new,fillcolor=red!50,
    incolor=yellow,
    action=draw**,
    hollow,
    rm=2,
    ...
```

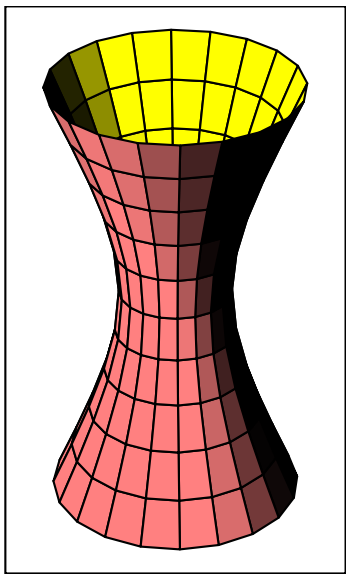
5.6.2 Exemple 2 : Hyperboloïde de rayon fixe

Comme à chaque fois, les options de la macro `\psSolid` peuvent embarquer du code postscript, voire du code jps.

Ci-contre un exemple en pur postscript, où on utilise les variables a, b et h qui sont transmises par les options de PSTricks. On obtient ainsi un solide variable construit à partir du scratch.

Remarque : le code utilisé provient d'un source jps pratiquement utilisé tel que :

http://melusine.eu.org/lab/bjps/solide/tour.jps



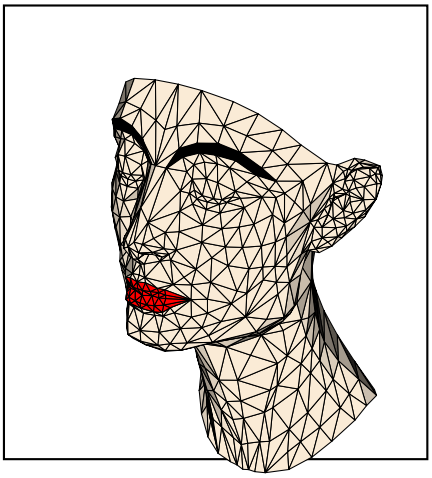
Le code utilisé est le suivant :

```
\psSolid[object=new,fillcolor=red!50,incolor=yellow,
    hollow,
    a=10, %% nb d'etages
    b=20, %% diviseur de 360, nb de meridiens
    h=8,  %% hauteur
    action=draw**,
    sommets=
      /z0 h neg 2 div def
      a -1 0 {
         /k exch def
         0 1 b 1 sub {
             /i exch def
             /r z0 h a div k mul add dup mul 4 div 1 add sqrt def
             360 b idiv i mul cos r mul
             360 b idiv i mul sin r mul
             z0 h a div k mul add
         } for
      } for,
    faces={
      0 1 a 1 sub {
      /k exch def
         k b mul 1 add 1 k 1 add b mul 1 sub {
             /i exch def
             [i i 1 sub b i add 1 sub b i add]
         } for
         [k b mul k 1 add b mul 1 sub k 2 add b mul 1 sub k 1 add b mul]
      } for
    }]
```

5.6.3 Exemple 3 : Import de fichiers externes

À partir d'un fichier de description de solide dans un format particulier (notamment autre que obj ou off), on peut fabriquer soit-même un fichier .dat contenant les coordonnées des sommets, et un autre fichier .dat contenant les tableaux des indices des sommets de chaque facette. Ces fichiers peuvent alors être utilisés en entrée pour les paramètres `sommets` et `faces` en utilisant l'instruction postscript `run`.

Dans l'exemple ci-dessous, les fichiers `sommets_nefer.dat` et `faces_nefer.dat` ont été placés dans le répertoire de compilation.



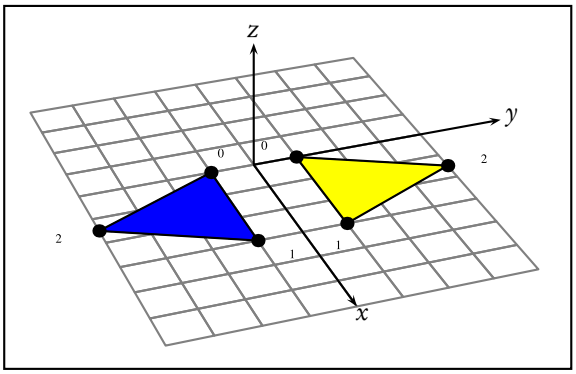
```
\definecolor{AntiqueWhite}{rgb}{0.98,0.92,0.84}
\psset{lightsrc=30 -40 10}
\psset{viewpoint=50 -50 20 rtp2xyz,Decran=50}
\psframe(-7,-7)(7,9)
\psSolid[object=new,RotX=90,fillcolor=AntiqueWhite,linewidth=0.5\pslinewidth,
    sommets= (sommets_nefer.dat) run,
    faces={(faces_nefer.dat) run}]%
```

5.7 Solide monoface – Solide biface

Le contour de `face` est défini sur le plan Oxy par les coordonnées des sommets placés dans le sens trigonométrique par le paramètre `base` :

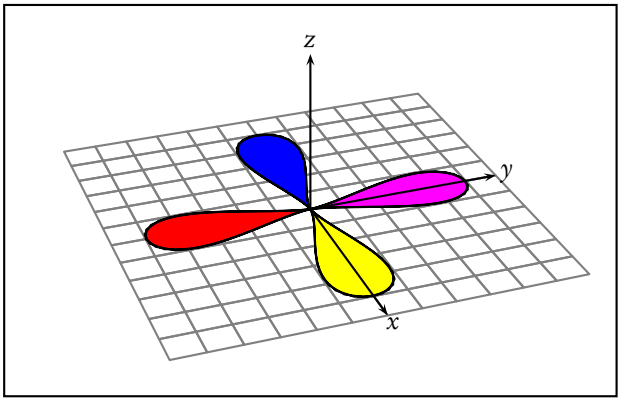
```
\psSolid[object=face,base=x1 y1 x2 y2 x3 y3 ...xn yn](0,0,0)%
```

5.7.1 'face' triangulaire




```
\psset{unit=0.6}
\psset{
  viewpoint=50 -20 30 rtp2xyz,Decran=50}
\begin{pspicture}(-6,-4.5)(7,3.5)
\psframe(-6,-4.5)(7,3.5)
\psSolid[
   object=face,fillcolor=yellow,action=draw*,
   incolor=blue, biface, base=0 0 3 0 1.5 3
   ](0,1,0)
\psSolid[
   object=face,fillcolor=yellow,action=draw*,
   incolor=blue, base=0 0 3 0 1.5 3,
   biface, RotX=180](0,-1,0)
\end{pspicture
```

5.7.2 'face' définie par une fonction



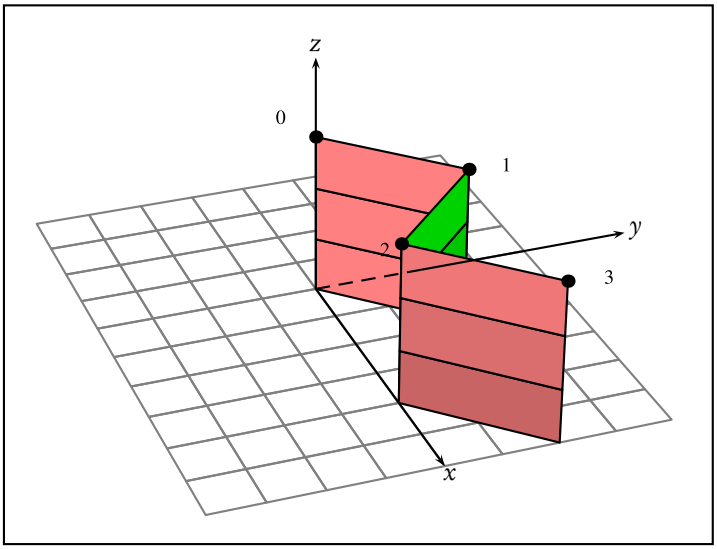

```
\defFunction[algebraic]{F}(t)
   {5*(cos(t))^2}
   {3*(sin(t))*(cos(t))^3}{}
base=0 2 pi mul {F} CourbeR2
```

5.8 Solide ruban

Le ruban est un paravent posé sur le sol horizontal. La base du paravent est définie sur le plan Oxy par les coordonnées des sommets placés dans le sens trigonométrique par le paramètre `base` :

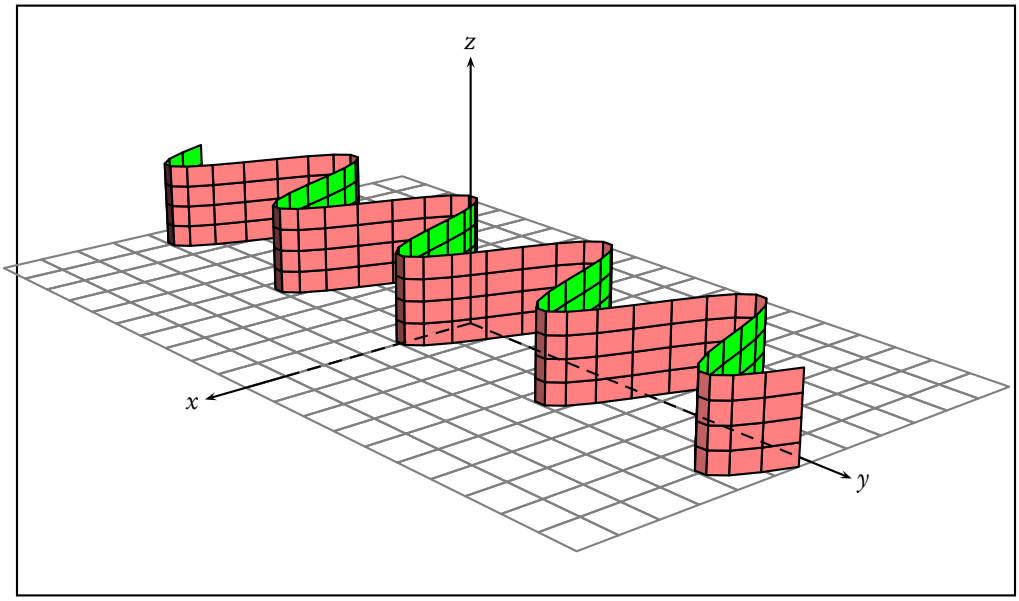
```
\psSolid[object=ruban,h=3,base=x1 y1 x2 y2 x3 y3 ...xn yn,ngrid=n](0,0,0)%
```

5.8.1 Un simple paravent




```
\begin{pspicture}(-5,-4)(6,7)
\psframe(-5,-4)(6,7)
\psSolid[
   object=grille,base=-4 6 -4 4,
   action=draw](0,0,0)
\psSolid[
   object=ruban,h=3,
   fillcolor=red!50,
   base=0 0 2 2 4 0 6 2,
   num=0 1 2 3,
   show=0 1 2 3,
   ngrid=3
   ](0,0,0)
\axesIIID(0,2,0)(6,6,6)
\end{pspicture}
```

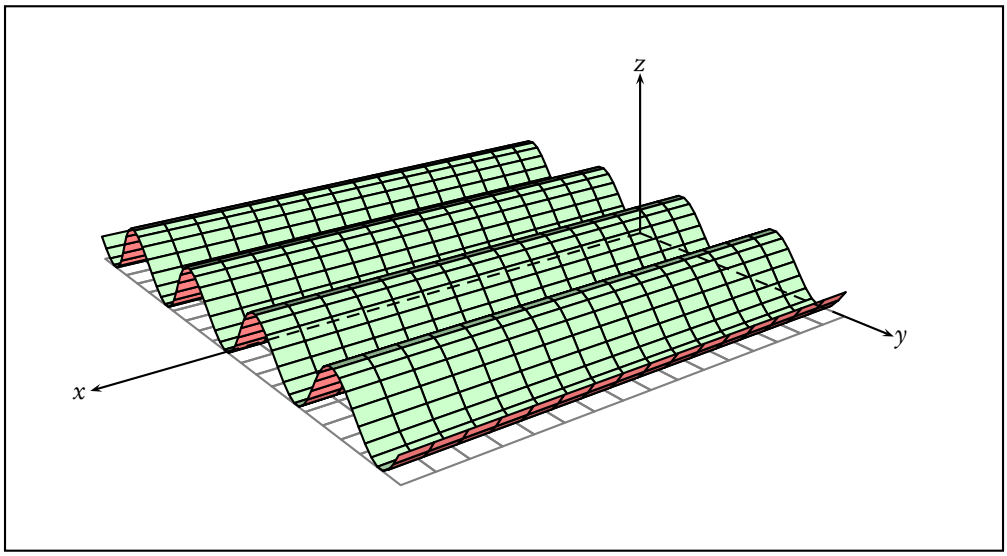
5.8.2 Un paravent sinusoïdal




```
\psset{unit=0.6}
\psset{lightsrc=10 30 10,viewpoint=50 50 20 rtp2xyz,Decran=50}
\begin{pspicture}(-10,-5)(12,7)
\psframe(-10,-5)(12,7)
\defFunction{F}(t){2 t 4 mul cos mul}{t 20 div}{}
\psSolid[object=grille,base=-6 6 -10 10,action=draw,linecolor=gray](0,0,0)
\psSolid[object=ruban,h=2,fillcolor=red!50,
      resolution=72,
      base=-200 200 {F} CourbeR2+,
      ngrid=4](0,0,0)
\axesIIID(5,10,0)(7,11,6)
\end{pspicture}
```

5.8.3 Une surface ondulée

C'est le même objet que précédemment en lui faisant subir une rotation de 90° autour de Oy.




```
\psset{unit=0.6}
\psset{lightsrc=10 30 10,viewpoint=50 50 20 rtp2xyz,Decran=30}
\begin{pspicture}(-14,-7)(8,7)
\defFunction{F}(t){t 4 mul cos}{t 20 div}{}
\psSolid[object=grille,base=0 16 -10 10,action=draw,linecolor=gray](0,0,0)
\psSolid[object=ruban,h=16,fillcolor=red!50,RotY=90,incolor=green!20,
      resolution=72,
      base=-200 200 {F} CourbeR2+,
      ngrid=16](0,0,1)
\psframe(-14,-7)(8,7)
\axesIIID(16,10,0)(20,12,6)
\end{pspicture}
```

On peut ensuite l'imaginer comme toit en tôle ondulée d'un abri quelconque.

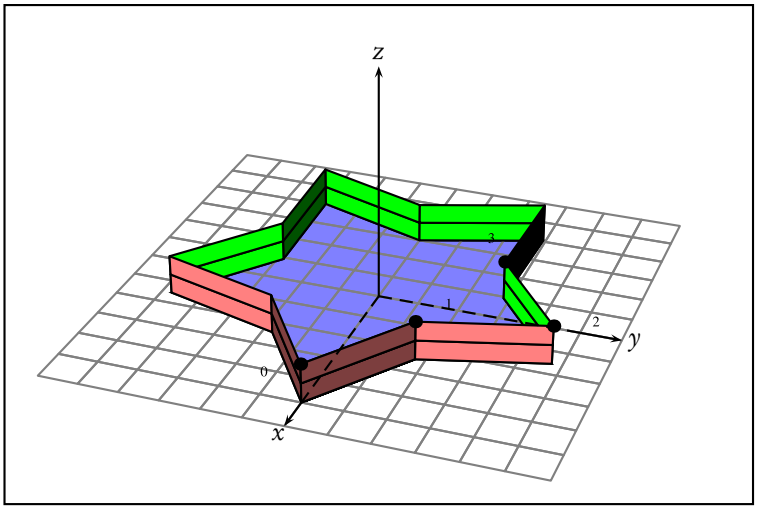
5.8.4 Un paravent étoilé : version 1

Le contour du paravent est défini dans une boucle :

```
base=0 72 360 {/Angle ED 5 Angle cos mul 5 Angle sin mul
     3 Angle 36 add cos mul 3 Angle 36 add sin mul} for
```

la surface bleutée du fond est définie à l'aide d'un polygone dont les sommets sont calculés par la commande `\psPoint(x,y,z){P}`

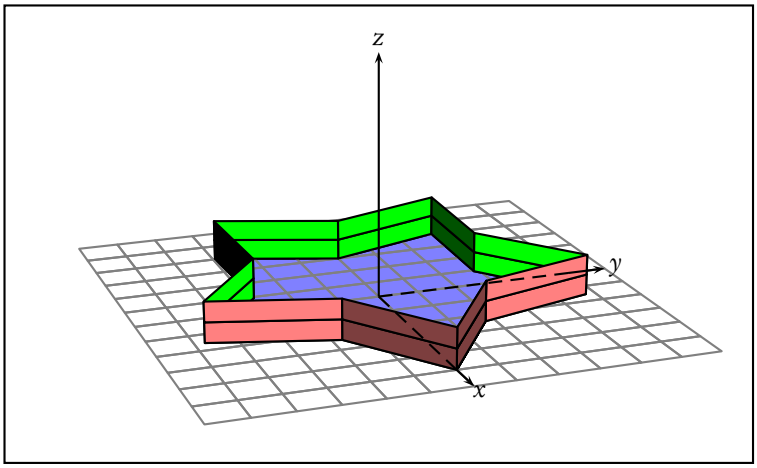
```
\multido{\iA=0+72,\iB=36+72,\i=0+1}{6}{%
    \psPoint(\iA\space cos 5 mul,\iA\space sin 5 mul,0){P\i}
    \psPoint(\iB\space cos 3 mul,\iB\space sin 3 mul,0){p\i}
    }%
\pspolygon[fillstyle=solid,fillcolor=blue!50](P0)(p0)(P1)(p1)(P2)(p2)
                                             (P3)(p3)(P4)(p4)(P5)(p5)
```




```
\begin{pspicture}(-9,-5)(9,7)
\psframe(-9,-5)(9,7)
\multido{\iA=0+72,\iB=36+72,\i=0+1}{6}{%
    \psPoint(\iA\space cos 5 mul,\iA\space sin 5 mul,0){P\i}
    \psPoint(\iB\space cos 3 mul,\iB\space sin 3 mul,0){p\i}
    }%
\pspolygon[fillstyle=solid,fillcolor=blue!50]
          (P0)(p0)(P1)(p1)(P2)(p2)(P3)(p3)(P4)(p4)(P5)(p5)
\defFunction{F}(t){t cos 5 mul}{t sin 5 mul}{}
\defFunction{G}(t){t 36 add cos 3 mul}{t 36 add sin 3 mul}{}
\psSolid[object=grille,base=-6 6 -6 6,action=draw,linecolor=gray](0,0,0)
\psSolid[object=ruban,h=1,fillcolor=red!50,
      base=0 72 360 {/Angle exch def Angle F Angle G} for,
      num=0 1 2 3, show=0 1 2 3, ngrid=2](0,0,0)
\axesIIID(5,5,0)(6,6,6)
\end{pspicture}
```

5.8.5 Un paravent étoilé : version 2

Le fond du récipient est défini par l'objet `face` avec l'option `biface` :




```
\psset{lightsrc=10 0 10,viewpoint=50 -20 20 rtp2xyz,Decran=50}
\begin{pspicture}(-9,-4)(9,7)
\psframe(-9,-4)(9,7)
\defFunction{F}(t){t cos 5 mul}{t sin 5 mul}{}
```

```
\defFunction{G}(t){t 36 add cos 3 mul}{t 36 add sin 3 mul}{}
\psSolid[object=face,fillcolor=blue!50,
      biface,
      base=0 72 360 {/Angle exch def Angle F Angle G} for,
      ](0,0,0)
\psSolid[object=grille,base=-6 6 -6 6,action=draw,linecolor=gray](0,0,0)
\psSolid[object=ruban,h=1,fillcolor=red!50,
      base=0 72 360 {/Angle exch def Angle F Angle G} for,
      ngrid=2](0,0,0)
\axesIIID(5,5,0)(6,6,6)
\end{pspicture}
```

5.9 Solide anneau

Cette partie traite des anneaux cylindriques. Dans la commande `\psSolid`, cet objet est tracé avec l'option : `[object=anneau]`, dont il faut fixer 3 paramètres :

- le rayon intérieur `[r=1.5]` (valeur par défaut) ;
- le rayon extérieur `[R=4]` (valeur par défaut)
- la hauteur `[h=6]` (valeur par défaut).

En option, l'argument `ngrid` permet de définir le nombre de sections utilisées pour faire une rotation complète de 360 degrés. Sa valeur par défaut est 24.

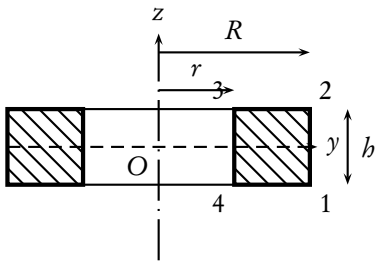
La section de l'anneau dont la forme `rectangulaire` a été choisie par défaut, peut être redessinée par l'utilisateur. Nous allons détailler différents exemples de section.

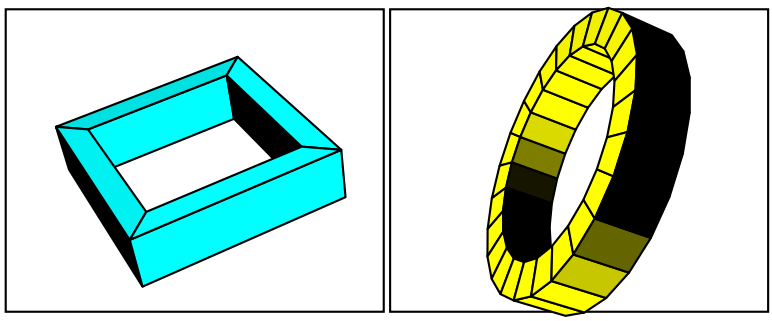
5.9.1 Commande pré-définie : l'anneau à section rectangulaire.

Cette section est définie dans le plan Oyz, elle est paramétrée par le triplet (r,R,h). Les valeurs du rayon extérieur R, du rayon intérieur r et de la hauteur h sont passées dans les options de `\psSolid`. Par défaut, on a donc un anneau à section rectangulaire variable, et la définition se fait au moment de la transmission des valeurs pour (r,R,h) dans les options de `\psSolid`.

Si l'utilisateur redéfinit la macro TEX `\Section` avec des valeurs numériques plutôt que les paramètres r, R et h, alors l'anneau n'est plus variable et il n'est plus besoin de transmettre des valeurs pour r, R, et h dans les options de `\psSolid`.

```
\newcommand\Section{%
% y z
  R h 2 div neg % sommet 1
  % S1 (R,-h/2)
  R h 2 div     % sommet 2
  % S2 (r,h/2)
  r h 2 div     % sommet 3
  % S3 (r,h/2)
  r h 2 div neg % sommet 4
  % S4 (r,-h/2)
            }
```

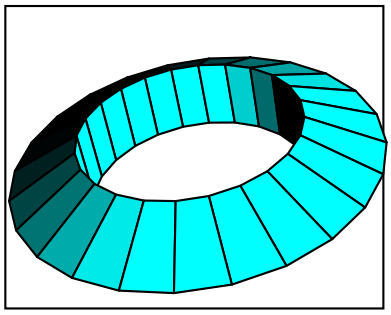




```
\psSolid[object=anneau,fillcolor=cyan,h=3,R=8,r=6,ngrid=4,RotX=10](0,0,0)
\psSolid[object=anneau,fillcolor=yellow,h=3,R=8,r=6,RotX=90,RotZ=10](0,0,0)
```

5.9.2 Un simple anneau à section triangulaire

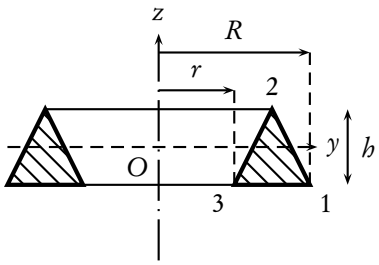
Ci-dessous un anneau très simple, à section triangulaire fixe. La section est définie par la donnée des 3 points $(6,-2)$, $(10,0)$ et $(6,2)$ dans l'option `section` de `\psSolid`.

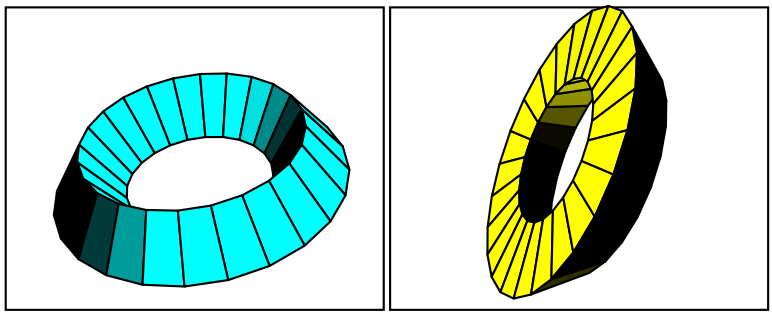


```
\psset{unit=0.5}
\begin{pspicture}(-5,-6)(5,6)
\psframe(-5,-4)(5,4)
\psset[pst-solides3d]{
   viewpoint=50 20 40 rtp2xyz,Decran=25,
   lightsrc=10 20 20}
\psSolid[object=anneau,
   section=6 -2 10 0 6 2,
   fillcolor=cyan,RotX=10]%
\end{pspicture}
```

5.9.3 Un anneau variable à section triangulaire

```
\newcommand\SectionTriangulaire{
% y <----z---->
  R h 2 div neg
  % S1 (R,-h/2)
  R r add 2 div h 2 div
  % S2 ((R+r)/2,h/2)
  r h 2 div neg
  % S3 (r,-h/2)
}
```

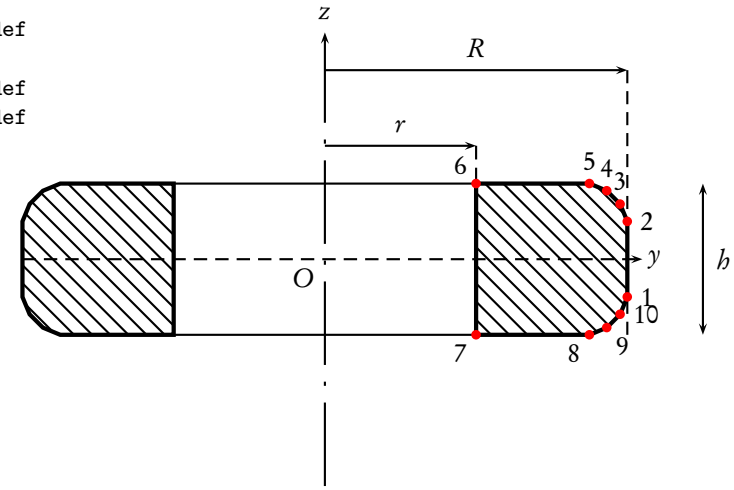


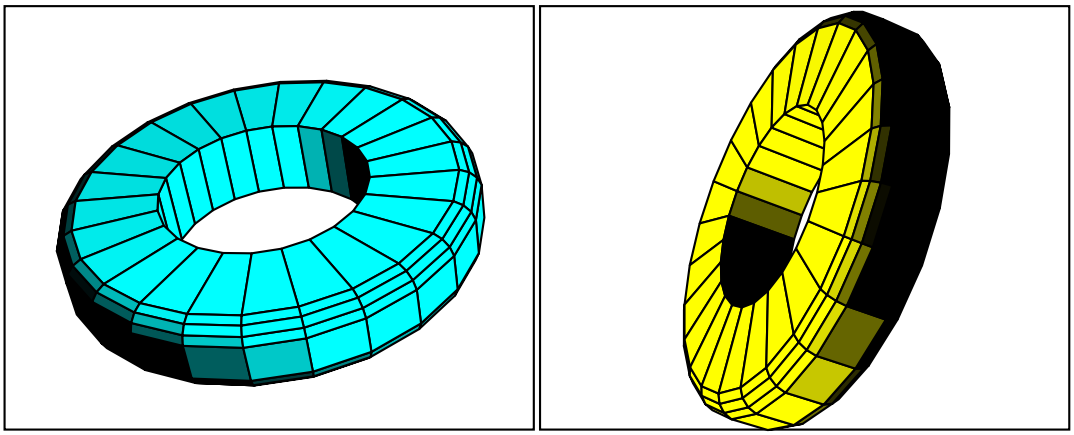


```
\psSolid[object=anneau,section=\SectionTriangulaire,%
        fillcolor=cyan,h=3,R=8,r=4,RotX=10](0,0,0)
%%
\psSolid[object=anneau,section=\SectionTriangulaire,%
        fillcolor=yellow,h=3,R=8,r=4,RotX=-90,RotZ=10](0,0,0)
```

5.9.4 L'anneau à section "pneu" : anneau cylindrique à arêtes chanfreinées.

```
\renewcommand\SectionPneu{
   /m {90 4 div} bind def
   /Scos {m cos 2 m mul cos add 3 m mul cos add} bind def
   /Z0 {h 4 div} bind def
   /c {Z0 Scos div} bind def
   /Z1 {Z0 c m cos mul add} bind def
   /Z2 {Z1 c m 2 mul cos mul add} bind def
   /R1 {R c m sin mul sub} bind def
   /R2 {R1 c m 2 mul sin mul sub} bind def
   /R3 {R2 c m 3 mul sin mul sub} bind def
   R h 4 div neg % 1
   R h 4 div % 2
   R1 Z1 % 3
   R2 Z2 % 4
   R3 h 2 div % 5
   r h 2 div  % 6
   r h 2 div neg  % 7
   R3 h 2 div neg % 8
   R2 Z2 neg % 9
   R1 Z1 neg % 10
   }
```

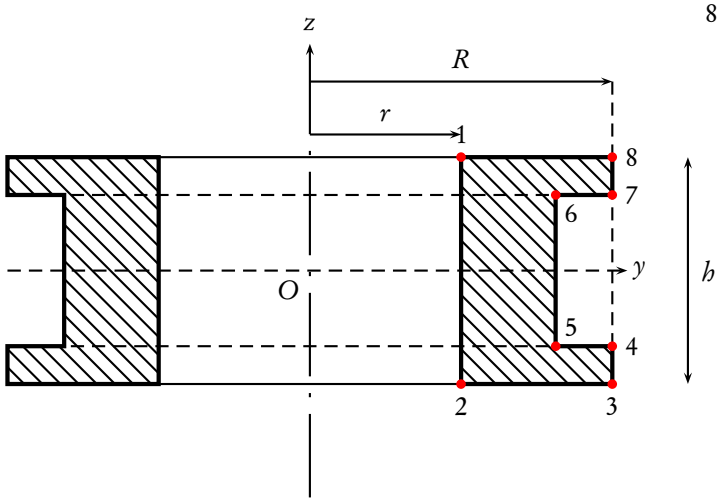


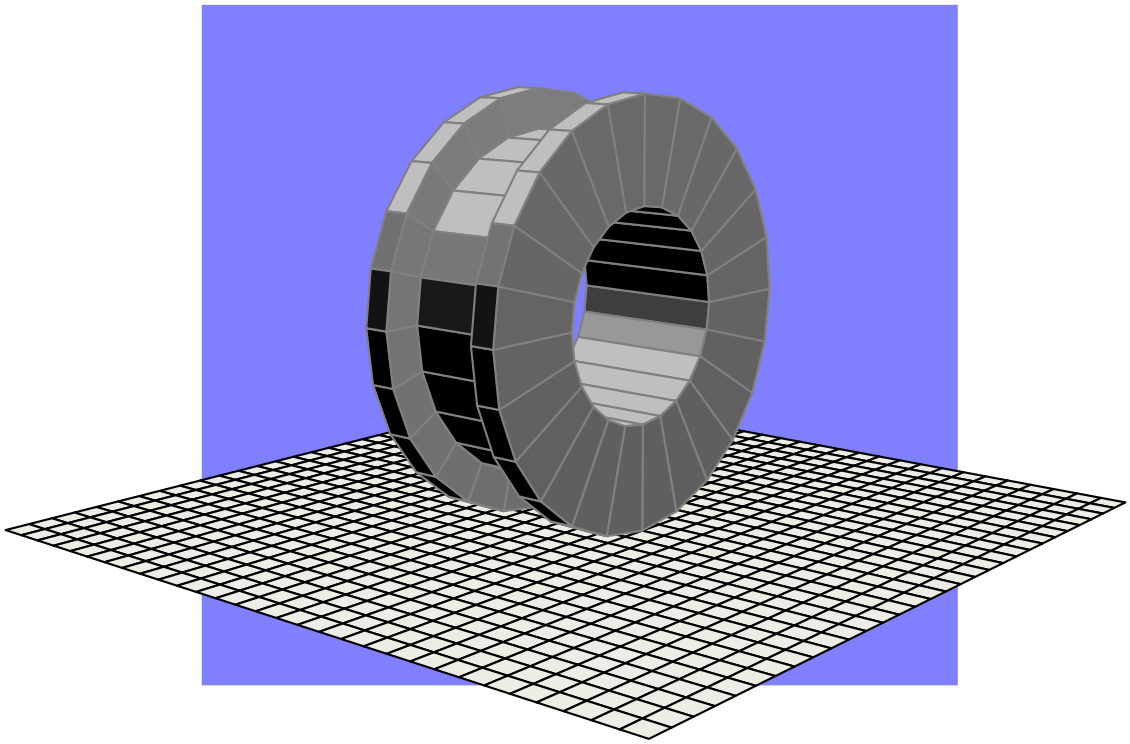


```
\psSolid[object=anneau,section=\SectionPneu,%
         fillcolor=cyan,h=3,R=8,r=4,RotX=10](0,0,0)
\psSolid[object=anneau,section=\SectionPneu,%
         fillcolor=yellow,h=3,R=8,r=4,RotX=-90,RotZ=10]%
```

5.9.5 Bobine vide

```
\newcommand\SectionBobine{
   r h 2 div % 1
   r h 2 div neg % 2
   R h 2 div neg % 3
   R h 3 div neg % 4
   R h 4 div sub h 3 div neg % 5
   R h 4 div sub h 3 div % 6
   R h 3 div % 7
   R h 2 div % 8
   }
```





```
\psSolid[object=grille,base=-15 15 -15 15,fillcolor=yellow!30](0,0,-8)
\psSolid[object=anneau,section=\SectionBobine,%
        fillcolor=gray!50,h=6,R=8,r=4,RotX=90,linecolor=gray]%
```

5.9.6 D'autres anneaux

Trois autres exemples sont disponibles sur la page :

http://syracuse.eu.org/lab/bpst/pst-solides3d/anneaux

5.10 Généralisation de la notion de cylindre et de cône

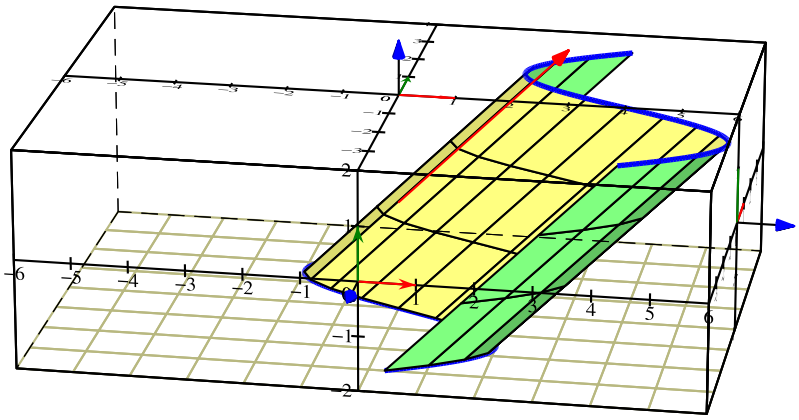
5.10.1 Cylindre ou nappe cylindrique quelconque

Cette partie généralise la notion de cylindre, ou de nappe cylindrique[1]. Il faut définir une courbe *directrice* par une fonction et la direction de l'axe du *cylindre*. Dans l'exemple ci-dessous la directrice est une sinusoïde, située dans le plan $z = -2$:

```
\defFunction[algebraic]{G1}(t){t}{2*sin(t)}{-2}
```

La direction du cylindre est définie par les coordonnées d'un vecteur dans le paramètre `axe=0 1 1`. Le dessin fait appel à `object=cylindre` qui en plus de ses paramètres usuels dont la hauteur `h=4` – il s'agit de la **longueur de la génératrice** et non de la distance entre les deux plans passant par les bases, est affecté de ceux définissant la directrice `function=G1` et la plage de la variable t `range=-3 3`.

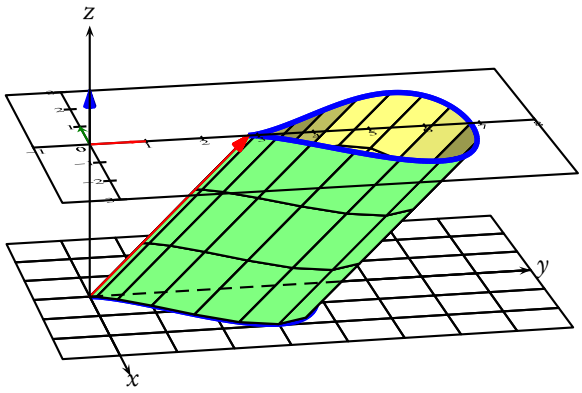
```
\psSolid[object=cylindre,
   h=4,function=G1,
   range=-3 3,
   ngrid=3 16,
   axe=0 1 1,
   incolor=green!50,
   fillcolor=yellow!50]
```



Dans cet exemple suivant, afin de représenter les deux plans horizontaux passant par les bases, on fait le calcul de la distance entre ces deux plans.

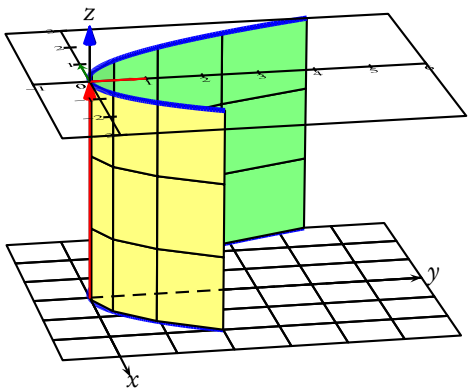
```
\pstVerb{/ladistance 2 sqrt 2 mul def}
```

[1] Cette partie a été écrite à l'initiative de Maxime Chupin, suite à une question sur la liste de diffusion de http://melusine.eu.org/cgi-bin/mailman/listinfo/syracuse

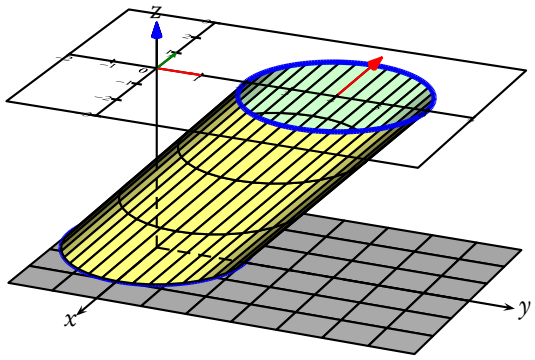


axe=0 1 1

```
\begin{pspicture}(-1.5,-3)(6.5,6)
\psSolid[object=grille,base=-3 3 -1 8,action=draw]
\pstVerb{/ladistance 2 sqrt 2 mul def}
\defFunction[algebraic]{G3}(t)
   {6*(cos(t))^3*sin(t)}
   {4*(cos(t))^2}
   {0}
\defFunction[algebraic]{G4}(t)
   {6*(cos(t))^3*sin(t)}
   {4*(cos(t))^2+ladistance}
   {ladistance}
\psSolid[object=courbe,function=G3,
   range=0 6.28,r=0,
   linecolor=blue,
   linewidth=2pt]
\psSolid[object=cylindre,
   range=0 -6.28,
   h=4,
   function=G3,
   axe=0 1 1,
   ngrid=3 36,
   fillcolor=green!50,
   incolor=yellow!50]
\psSolid[object=courbe,function=G4,
   range=0 6.28,r=0,
   linecolor=blue,
   linewidth=2pt]
\psSolid[object=vecteur,
         linecolor=red,
         args=0 ladistance dup]
\psSolid[object=plan,action=draw,
   definition=equation,
   args={[0 0 1 ladistance neg] 90},
   base=-1 8 -3 3,planmarks,showBase]
\axesIIID(0,4.5,0)(4,8,5)
\rput(0,-3){\texttt{axe=0 1 1}}
\end{pspicture}
```

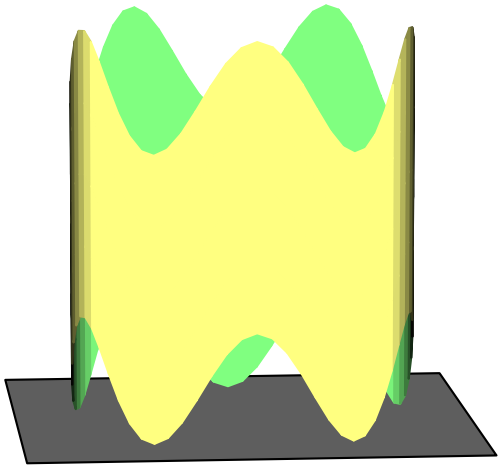



```
\begin{pspicture}(-1.5,-3)(6.5,6)
\psSolid[object=grille,base=-3 3 -1 6,action=draw]
\defFunction[algebraic]{G5}(t)
   {t}{0.5*t^2}{0}
\defFunction[algebraic]{G6}(t)
    {t}{0.5*t^2}{4}
\psSolid[object=courbe,function=G5,
   range=-3 2,r=0,
   linecolor=blue,
   linewidth=2pt]
\psSolid[object=cylindre,
   range=-3 2,
   h=4,
   function=G5,
   axe=0 0 1, %% valeur par défaut
   incolor=green!50,
   fillcolor=yellow!50,
   ngrid=3 8]
\psSolid[object=courbe,function=G6,
   range=-3 2,r=0,
   linecolor=blue,
   linewidth=2pt]
\axesIIID(0,4.5,0)(4,6,5)
\psSolid[object=vecteur,
         linecolor=red,
         args=0 0 4]
\psSolid[object=plan,action=draw,
   definition=equation,
   args={[0 0 1 -4] 90},
   base=-1 6 -3 3,planmarks,showBase]
\end{pspicture}
```




```
\begin{pspicture}(-1.5,-3)(6.5,6)
\psset{lightsrc=viewpoint,viewpoint=100 45 45,Decran
  =100}
\psSolid[object=grille,base=-3 3 -2 7,fillcolor=gray
  !30]
\defFunction[algebraic]{G7}(t)
  {2*cos(t)}{2*sin(t)}{0}
\defFunction[algebraic]{G8}(t)
   {2*cos(t)}{2*sin(t)+4}{4}
\psSolid[object=courbe,function=G7,
  range=0 6.28,r=0,
  linecolor=blue,
  linewidth=2pt]
\psSolid[object=cylindre,
  range=0 6.28,
  h=5.65685,
  function=G7,
  axe=0 1 1,
  incolor=green!20,
  fillcolor=yellow!50,
  ngrid=3 36]
\psSolid[object=courbe,function=G8,
  range=0 6.28,r=0,
  linecolor=blue,
  linewidth=2pt]
\axesIIID(2,4.5,2)(4,8,5)
\psSolid[object=vecteur,
        linecolor=red,
        args=0 1 1](0,4,4)
\psSolid[object=plan,action=draw,
  definition=equation,
  args={[0 0 1 -4] 90},
  base=-2 7 -3 3,planmarks,showBase]
\end{pspicture}
```

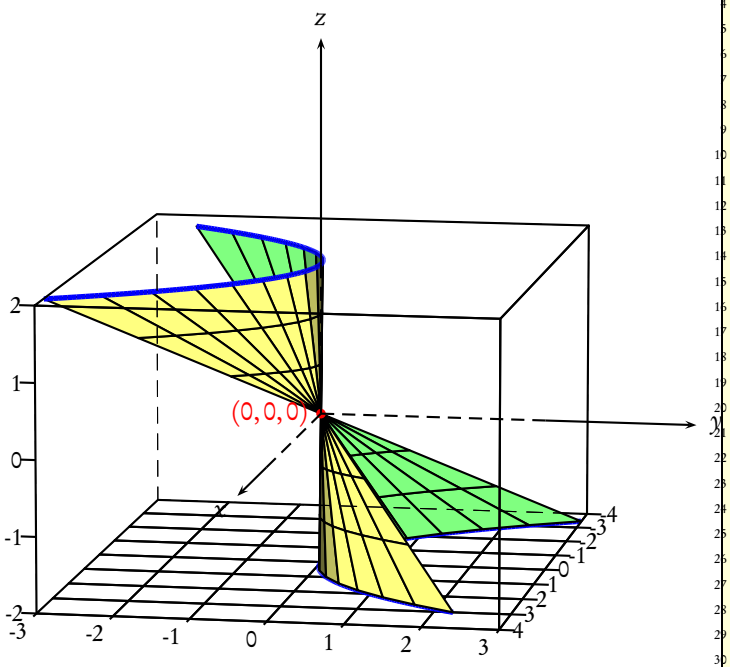
La directrice peut être une courbe quelconque et n'est pas obligatoirement une courbe plane et horizontale.



```
\begin{pspicture}(-1.5,-4)(4,6)
\psset{unit=0.75,lightsrc=viewpoint,viewpoint=100 -5
  10 rtp2xyz,Decran=100}
\psSolid[object=grille,base=-4 4 -4 4,ngrid=8.
  8.](0,0,-1)
\defFunction[algebraic]{G9}(t)
   {3*cos(t)}
   {3*sin(t)}
   {1*cos(5*t)}
\psSolid[object=cylindre,
  range=0 6.28,
  h=5,
  function=G9,
  axe=0 0 1,
  incolor=green!50,
  fillcolor=yellow!50,
  ngrid=4 72,grid]
\end{pspicture}
```

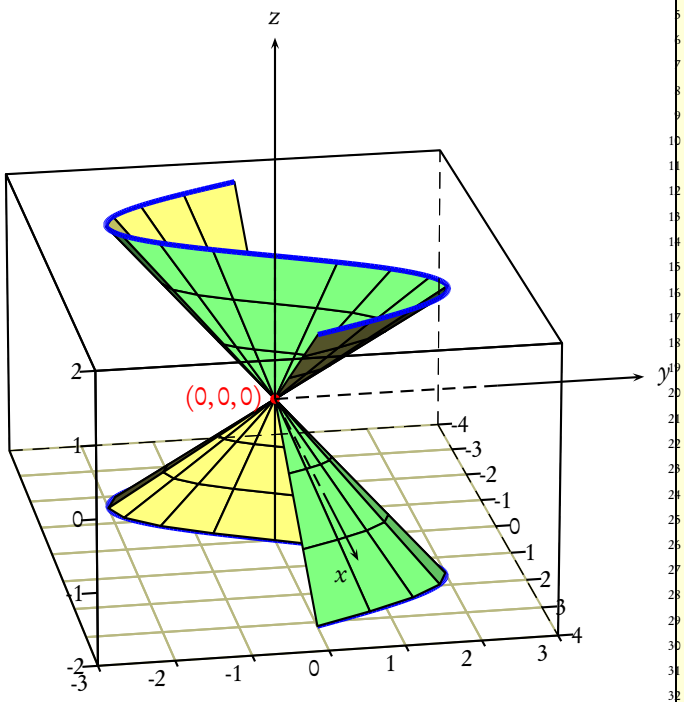
5.10.2 Cône ou nappe conique quelconque

Cette partie généralise la notion de cône et de nappe conique[2]. Il faut définir une courbe *directrice* par une fonction qui dessinera la base du cône, puis le sommet du *cône* qui par défaut est `origine=0 0 0`. Les parties supérieure et inférieure du cône sont symétriques par rapport au sommet. Dans l'exemple ci-dessous la directrice est un arc de parabole, situé dans le plan $z = -2$.

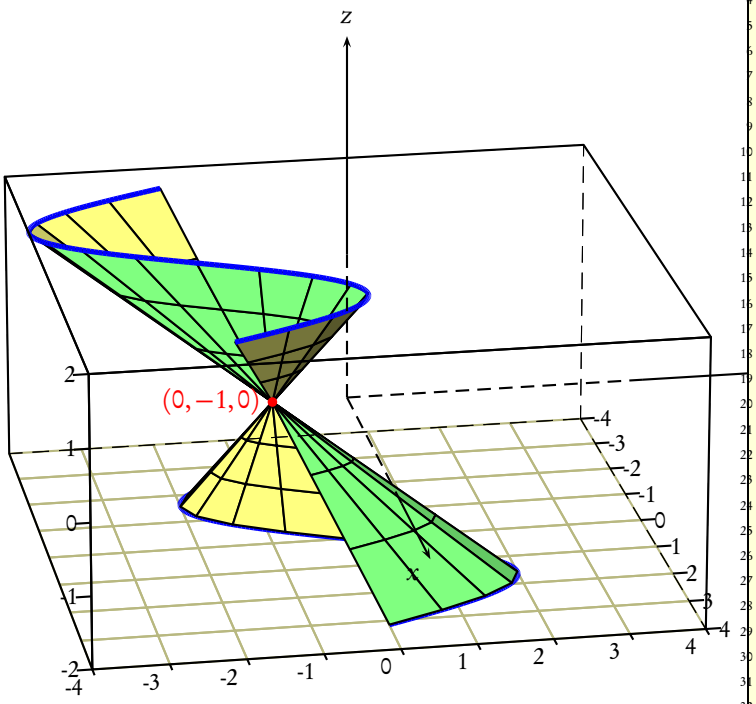



```
\begin{pspicture}(-3,-3)(4.5,5)
\psset{lightsrc=viewpoint,viewpoint=100 10 10 rtp2xyz,
  Decran=100}
\psSolid[object=grille,base=-4 4 -3 3,action=draw
  ](0,0,-2)
\defFunction[algebraic]{G1}(t)
    {t}
    {0.25*t^2}
    {-2}
\defFunction[algebraic]{G2}(t)
    {-t}
    {-0.25*t^2}
    {2}
\psSolid[object=courbe,
  function=G1,
  range=-3.46 3,
  r=0,
  linecolor=blue,
  linewidth=2pt]
\psSolid[object=cone,
  function=G1,
  range=-3.46 3,
  ngrid=3 16,
  incolor=green!50,
  fillcolor=yellow!50,
  origine=0 0 0]
\psSolid[object=courbe,
  function=G2,
  range=-3.46 3,
  r=0,
  linecolor=blue,
  linewidth=2pt]
\psPoint(0,0,0){I}
\uput[l](I){\red$(0,0,0)$}
\psdot[linecolor=red](I)
\gridIIID[Zmin=-2,Zmax=2,spotX=r](-4,4)(-3,3)
\end{pspicture}
```

[2]Cette partie, comme la précédente, a été écrite à l'initiative de Maxime Chupin, suite à une question sur la liste de diffusion de http://melusine.eu.org/cgi-bin/mailman/listinfo/syracuse




```
\begin{pspicture}(-3,-3)(4.5,5)
\psset{lightsrc=viewpoint,viewpoint=100 -10 20 rtp2xyz,
  Decran=100}
\psSolid[object=grille,base=-4 4 -3 3,linecolor={[rgb
  ]{0.72 0.72 0.5}},action=draw](0,0,-2)
\defFunction[algebraic]{G1}(t)
    {t}
    {2*sin(t)}
    {-2}
\defFunction[algebraic]{G2}(t)
    {-t}
    {-2*sin(t)}
    {2}
\psSolid[object=courbe,
   function=G1,
   range=-3.14 3.14,
   r=0,
   linecolor=blue,
   linewidth=2pt]
\psSolid[object=cone,
   function=G1,
   range=-3.14 3.14,
   ngrid=3 16,
   incolor=green!50,
   fillcolor=yellow!50,
   origine=0 0 0]
\psSolid[object=courbe,
   function=G2,
   range=-3.14 3.14,
   r=0,
   linecolor=blue,
   linewidth=2pt]
\psPoint(0,0,0){I}
\uput[l](I){\red$(0,0,0)$}
\psdot[linecolor=red](I)
\gridIIID[Zmin=-2,Zmax=2,spotX=r](-4,4)(-3,3)
\end{pspicture}
```




```
\begin{pspicture}(-3,-3)(4.5,5)
\psset{lightsrc=viewpoint,viewpoint=100 -10 20 rtp2xyz,
  Decran=100}
\psSolid[object=grille,base=-4 4 -4 4,linecolor={[rgb
  ]{0.72 0.72 0.5}},action=draw](0,0,-2)
\defFunction[algebraic]{G1}(t)
    {t}
    {2*sin(t)}
    {-2}
\defFunction[algebraic]{G2}(t)
    {-t}{-2*sin(t)-2}
    {2}
\psSolid[object=courbe,
  function=G1,
  range=-3.14 3.14,
  r=0,
  linecolor=blue,
  linewidth=2pt]
\psSolid[object=cone,
  function=G1,
  range=-3.14 3.14,
  ngrid=3 16,
  incolor=green!50,
  fillcolor=yellow!50,
  origine=0 -1 0]
\psSolid[object=courbe,
  function=G2,
  range=-3.14 3.14,
  r=0,
  linecolor=blue,
  linewidth=2pt]
\psPoint(0,-1,0){I}
\uput[l](I){\red$(0,-1,0)$}
\psdot[linecolor=red](I)
\gridIIID[Zmin=-2,Zmax=2,spotX=r](-4,4)(-4,4)
\end{pspicture}
```

Pour les cônes aussi, la directrice peut être une courbe quelconque et n'est pas obligatoirement une courbe plane et horizontale, comme dans l'exemple suivant écrit par Maxime Chupin.
http://melusine.eu.org/lab/bpst/pst-solides3d/cone/cone-dir_02.pst

5.11 Les surfaces paramétrées

5.11.1 Méthode

Les surfaces paramétrées écrites sous la forme $[x(u,v), y(u,v), z(u,v)]$ seront gérées grâce à la commande `\psSolid` par l'option `object=surfaceparametree` et définies soit en *notation polonaise inverse* (RPN, *Reverse Polish Notation*) :

```
\defFunction{shell}(u,v){1.2 v exp u Sin dup mul v Cos mul mul}% x(u,v)
                        {1.2 v exp u Sin dup mul v Sin mul mul}% y(u,v)
                        {1.2 v exp u Sin u Cos mul mul}        %z(u,v)
```

soit en *notation algébrique* :

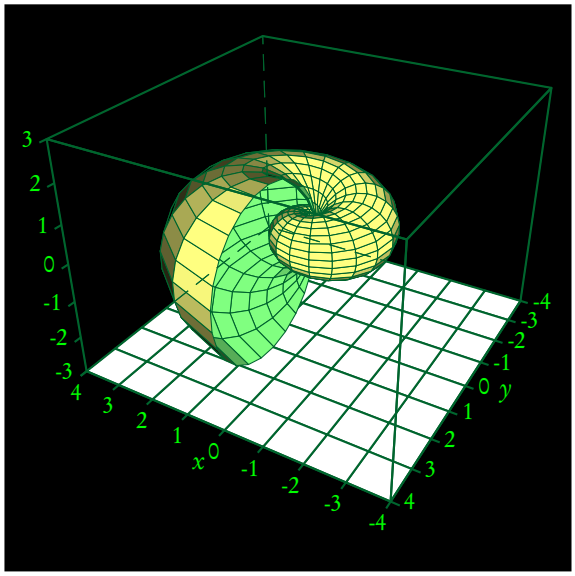
```
\defFunction[algebraic]{shell}(u,v){1.2^v*(sin(u)^2*cos(v))}% x(u,v)
                                   {1.2^v*(sin(u)^2*sin(v))}% y(u,v)
                                   {1.2^v*(sin(u)*cos(u))}  %z(u,v)
```

Les plages de valeurs pour u et v sont définies dans l'option `range=`u_{min} u_{max} v_{min} v_{max}.

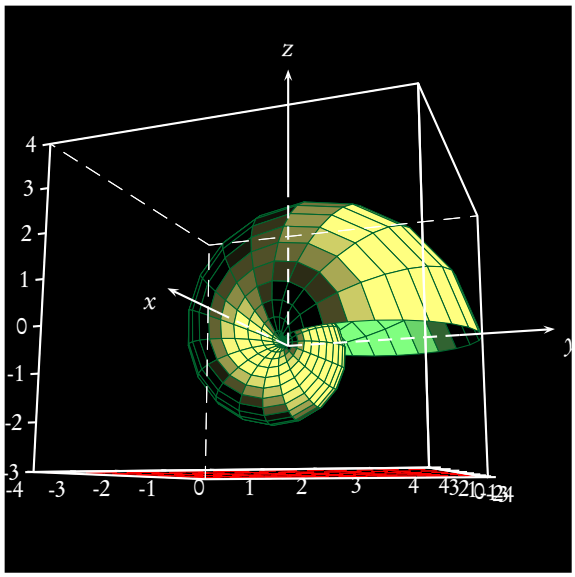
Le tracé de la fonction est activé par `function=nom_de_la_fonction`, ce nom a été précisé lorsque les équations paramétriques ont été écrites : `\defFunction{nom_de_la_fonction}...`

Tout autre choix que u et v est acceptable. Rappelons que l'argument de `Sin` et `Cos` doit être en radians et celui de `sin` et `cos` en degrés si vous utilisez la *RPN*. En notation algébrique, l'argument est en radians.

5.11.2 Exemple 1 : dessin d'un coquillage

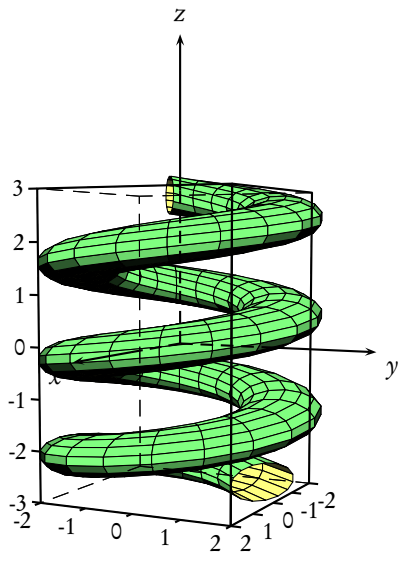



```
\psset{unit=0.75}
\begin{pspicture}(-5.5,-6)(4.5,4)
\psframe*(-5.5,-6)(4.5,4)
\psset[pst-solides3d]{viewpoint=20 120 30 rtp2xyz,
  Decran=15,lightsrc=-10 15 10}
% Parametric Surfaces
\psSolid[object=grille,base=-4 4 -4 4,
  action=draw*,linecolor={[cmyk]{1,0,1,0.5}}]
  (0,0,-3)
\defFunction{shell}(u,v)
  {1.2 v exp u Sin dup mul v Cos mul mul}
  {1.2 v exp u Sin dup mul v Sin mul mul}
  {1.2 v exp u Sin u Cos mul mul}
\psSolid[object=surfaceparametree,
  linecolor={[cmyk]{1,0,1,0.5}},
  base=0 pi pi 4 div neg 5 pi mul 2 div,
  fillcolor=yellow!50,incolor=green!50,
  function=shell,linewidth=0.5\pslinewidth,ngrid=25]%
\psSolid[object=parallelepiped,a=8,b=8,c=6,
  action=draw,linecolor={[cmyk]{1,0,1,0.5}}]%
\quadrillage
\end{pspicture}
```



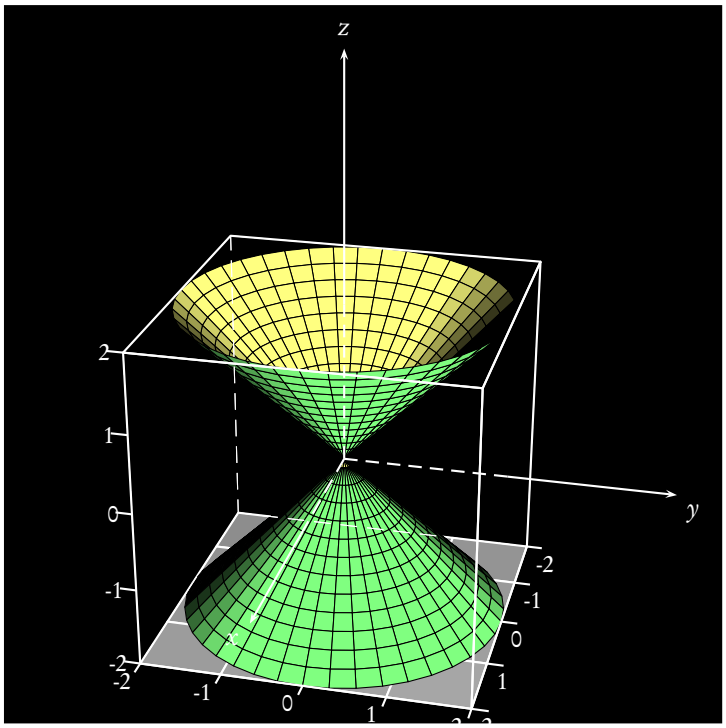

```
\psset{unit=0.75}
\begin{pspicture}(-5,-4)(5,6)
\psframe*(-5,-4)(5,6)
\psset[pst-solides3d]{viewpoint=20 20 -10 rtp2xyz,
  Decran=15,lightsrc=5 10 2}
% Parametric Surfaces
\psSolid[object=grille,base=-4 4 -4 4,
  action=draw*,linecolor=red](0,0,-3)
\defFunction[algebraic]{shell}(u,v)
  {1.21^v*(sin(u)*cos(u))}
  {1.21^v*(sin(u)^2*sin(v))}
  {1.21^v*(sin(u)^2*cos(v))}
%% \defFunction{shell}(u,v)
%%    {1.2 v exp u Sin u Cos mul mul}
%%    {1.2 v exp u Sin dup mul v Sin mul mul}
%%    {1.2 v exp u Sin dup mul v Cos mul mul}
\psSolid[object=surfaceparametree,
   linecolor={[cmyk]{1,0,1,0.5}},
   base=0 pi pi 4 div neg 5 pi mul 2 div,
   fillcolor=green!50,incolor=yellow!50,
   function=shell,linewidth=0.5\pslinewidth,
   ngrid=25]%
\white%
\gridIIID[Zmin=-3,Zmax=4,linecolor=white,
  QZ=0.5](-4,4)(-4,4)
\end{pspicture}
```

5.11.3 Exemple 2 : une hélice tubulaire




```
\psset{unit=0.75}
\begin{pspicture}(-3,-4)(3,6)
\psset[pst-solides3d]{viewpoint=20 10 2,Decran=20,
  lightsrc=20 10 10}
% Parametric Surfaces
\defFunction{helix}(u,v)
  {1 .4 v Cos mul sub u Cos mul 2 mul}
  {1 .4 v Cos mul sub u Sin mul 2 mul}
  {.4 v Sin mul u .3 mul add}
\psSolid[object=surfaceparametree,linewidth=0.5\pslinewidth,
  base=-10 10 0 6.28,fillcolor=yellow!50,incolor=green!50,
  function=helix,
  ngrid=60 0.4]%
\gridIIID[Zmin=-3,Zmax=3](-2,2)(-2,2)
\end{pspicture}
```

5.11.4 Exemple 3 : un cône




```
\psset{unit=0.5}
\begin{pspicture}(-9,-7)(10,12)
\psframe*(-9,-7)(10,12)
\psset[pst-solides3d]{
  viewpoint=20 5 10,
  Decran=50,lightsrc=20 10 5}
\psSolid[
  object=grille,base=-2 2 -2 2,
  linecolor=white](0,0,-2)
% Parametric Surfaces
\defFunction{cone}(u,v)
  {u v Cos mul}{u v Sin mul}{u}
\psSolid[object=surfaceparametree,
   base=-2 2 0 2 pi mul,
   fillcolor=yellow!50,
   incolor=green!50,function=cone,
   linewidth=0.5\pslinewidth,
   ngrid=25 40]%
\psset{linecolor=white}\white
\gridIIID[Zmin=-2,Zmax=2]
  (-2,2)(-2,2)
\end{pspicture}
```

5.11.5 Un site

Vous trouverez sur le site :

http://k3dsurf.sourceforge.net/

un excellent logiciel pour représenter les surfaces avec de nombreux exemples de surfaces paramétrées et autres.

Chapitre 6

Surfaces définies par une fonction $z = f(x, y)$

6.1 Présentation

Cette commande prend la forme suivante :

```
\psSurface[options](xmin,ymin)(xmax,ymax){equation de la surface z=f(x,y)}
```

avec comme options possibles les mêmes que dans le cas des solides avec quelques options spécifiques :

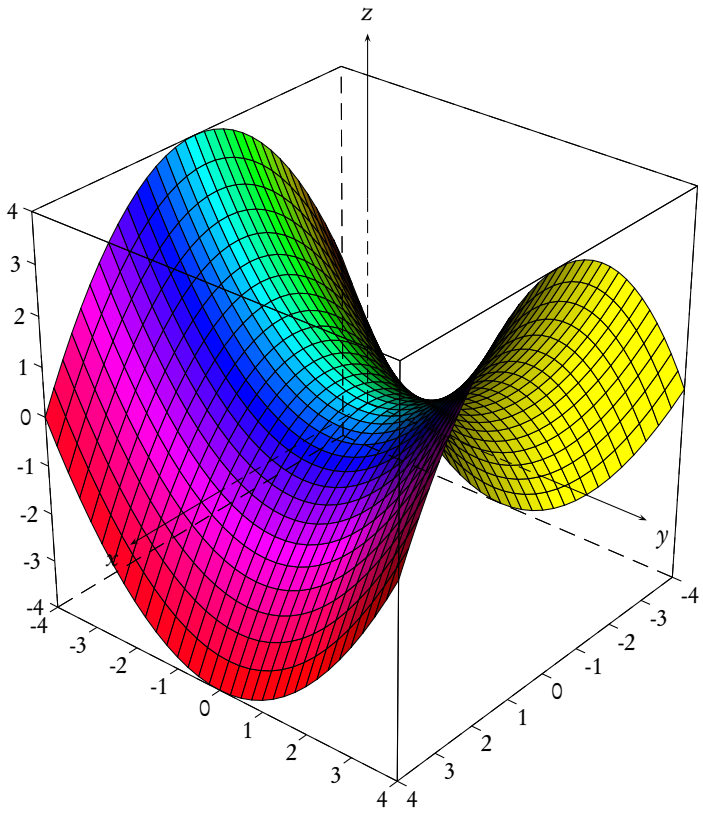
- Le maillage de la surface est défini par le paramètre `[ngrid=n1 n2]`, qui possède quelques particularités :
  - Si `n1` et/ou `n2` sont entiers, ce(s) nombre(s) représente(nt) le nombre de mailles suivant Ox et/ou Oy.
  - Si `n1` et/ou `n2` sont décimaux, ce(s) nombre(s) représente(nt) le pas d'incrémentation suivant Ox et/ou Oy.
  - Si `[ngrid=n]` ne possède qu'un seul paramètre, alors le nombre de mailles ou, suivant le cas, le pas d'incrémentation sera identique sur les deux axes.
-  `[algebraic]` : **cette option permet d'écrire la fonction en notation algébrique, `pstricks.pro` contient maintenant le code `AlgToPs` de Dominique Rodriguez qui le permet et qui auparavant était inclus dans `pstricks-add.pro`. Cette version de `pstricks` est fournie avec `pst-solides3d`. Le cas échéant, il faudra inclure le package `pstricks-add` dans le préambule de votre document.**
- `[grid]` : par défaut le maillage est activé, si l'option `[grid]` est écrite, alors le maillage est désactivé !
- `[axesboxed]` : cette option permet de tracer un quadrillage 3D de façon semi-automatique, car il convient de placer à la main les bornes de z, par défaut cette option est désactivée :
  - `[Zmin]` ;
  - `[Zmax]` ;
  - `[QZ]` : permet de décaler verticalement le repère de la valeur `[QZ=valeur]` ;
  - `[spotX]` : permet de placer, si le choix fait par défaut n'est pas satisfaisant, les valeurs des graduations sur l'axe des x autour de l'extrémité de la graduation. Cette valeur est celle que l'on indique à la commande `\uput[angle](x,y){donnée}` ;
  - `[spotY]` : idem ;
  - `[spotZ]` : idem.

Si l'option `[axesboxed]` ne vous donne pas satisfaction il est possible d'adapter la commande suivante, qui convient au premier exemple :

```
\psSolid[object=parallelepiped,a=8,b=8,c=8,action=draw](0,0,0)
\multido{\ix=-4+1}{9}{%
    \psPoint(\ix\space,4,-4){X1}
    \psPoint(\ix\space,4.2,-4){X2}
    \psline(X1)(X2)\uput[dr](X1){\ix}}
\multido{\iy=-4+1}{9}{%
    \psPoint(4,\iy\space,-4){Y1}
    \psPoint(4.2,\iy\space,-4){Y2}
    \psline(Y1)(Y2)\uput[dl](Y1){\iy}}
\multido{\iz=-4+1}{9}{%
```

```
    \psPoint(4,-4,\iz\space){Z1}
    \psPoint(4,-4.2,\iz\space){Z2}
    \psline(Z1)(Z2)\uput[l](Z1){\iz}}
```

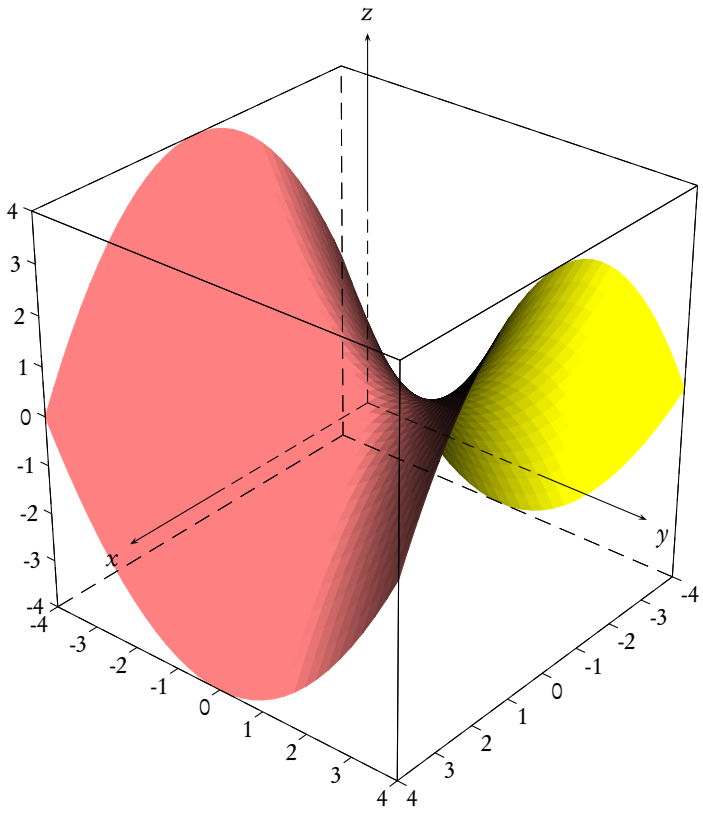
6.2 Exemple 1 : selle de cheval




```
\psset{unit=0.75}
\psset{viewpoint=50 40 30 rtp2xyz,Decran=50}
\psset{lightsrc=viewpoint}
\begin{pspicture}(-6,-7)(7,7)
\psSurface[ngrid=.25 .25,incolor=yellow,
   linewidth=0.5\pslinewidth,axesboxed,
   algebraic,hue=0 1](-4,-4)(4,4){%
   ((y^2)-(x^2))/4 }
\end{pspicture}
```

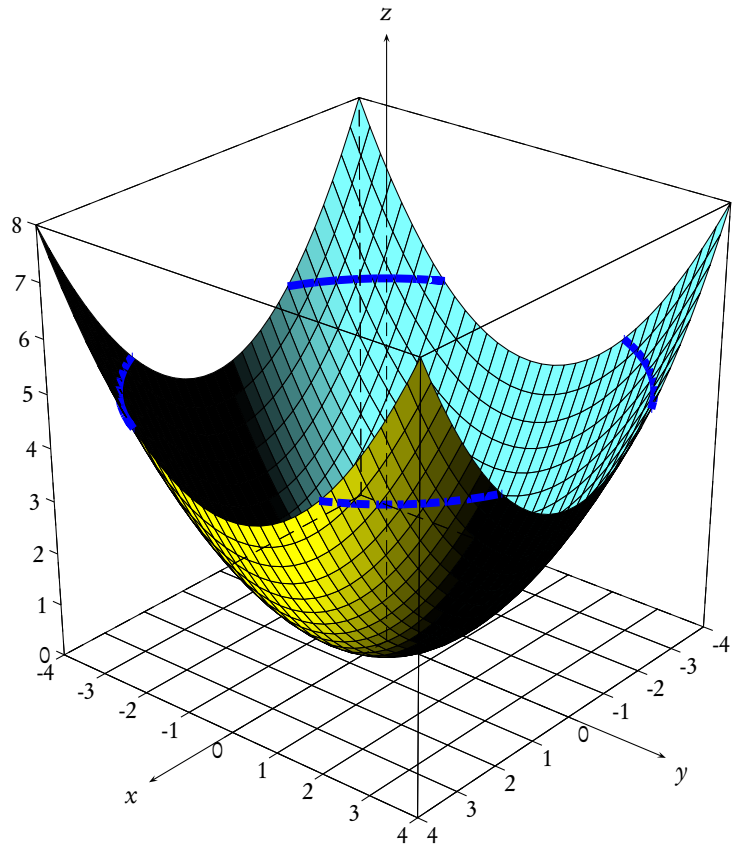
6.3 Exemple 2 : selle de cheval sans maillage

Les lignes du maillage sont supprimées en écrivant dans les options : `grid`.



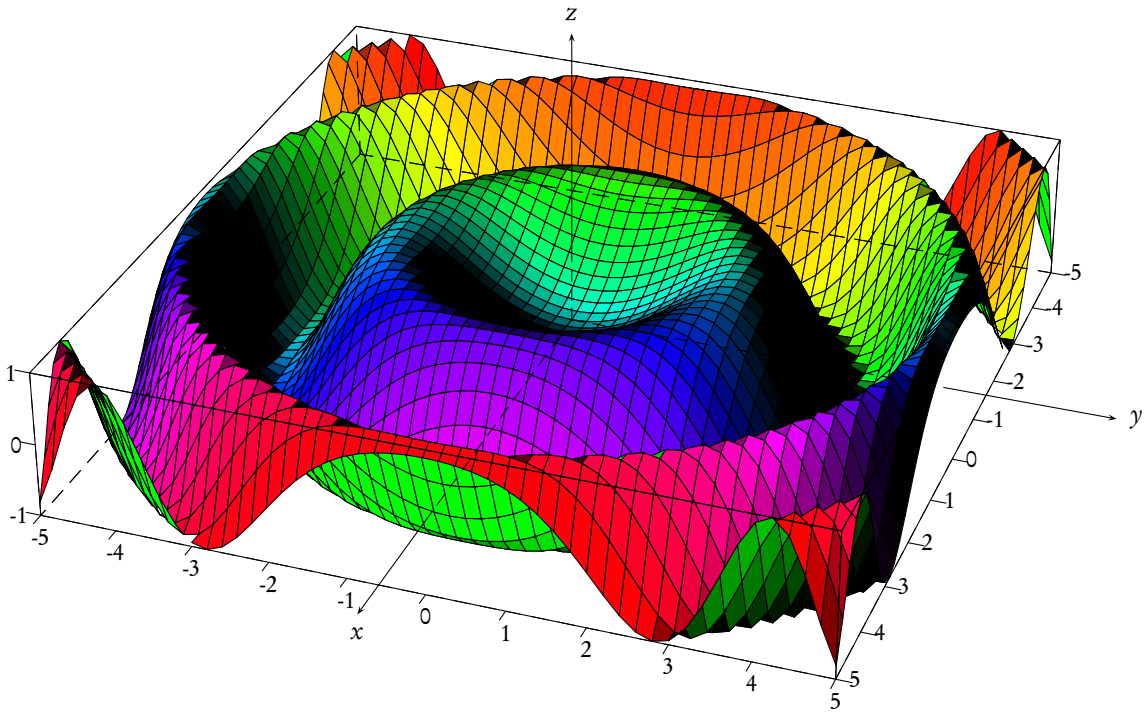

```
\psset{unit=0.75}
\psset{lightsrc=30 30 25}
\psset{viewpoint=50 40 30 rtp2xyz,Decran=50}
\begin{pspicture}(-6,-8)(7,8)
\psSurface[fillcolor=red!50,ngrid=.25 .25,
   incolor=yellow,linewidth=0.5\pslinewidth,
   grid,axesboxed](-4,-4)(4,4){%
   y dup mul x dup mul sub 4 div }
\end{pspicture}
```

6.4 Exemple 3 : paraboloïde




```
\psset{unit=0.75}
\psset{lightsrc=30 -10 10,linewidth=0.5\pslinewidth}
\psset{viewpoint=50 40 30 rtp2xyz,Decran=50}
\begin{pspicture}(-6,-4)(7,12)
\psSolid[object=grille,base=-4 4 -4 4,action=draw]%
\psSurface[
   fillcolor=cyan!50,
   intersectionplan={[0 0 1 -5]},
   intersectioncolor=(bleu),
   intersectionlinewidth=3,
    intersectiontype=0,
     ngrid=.25 .25,incolor=yellow,
   axesboxed,Zmin=0,Zmax=8,QZ=4](-4,-4)(4,4){%
   y dup mul x dup mul add 4 div }
\end{pspicture}
```

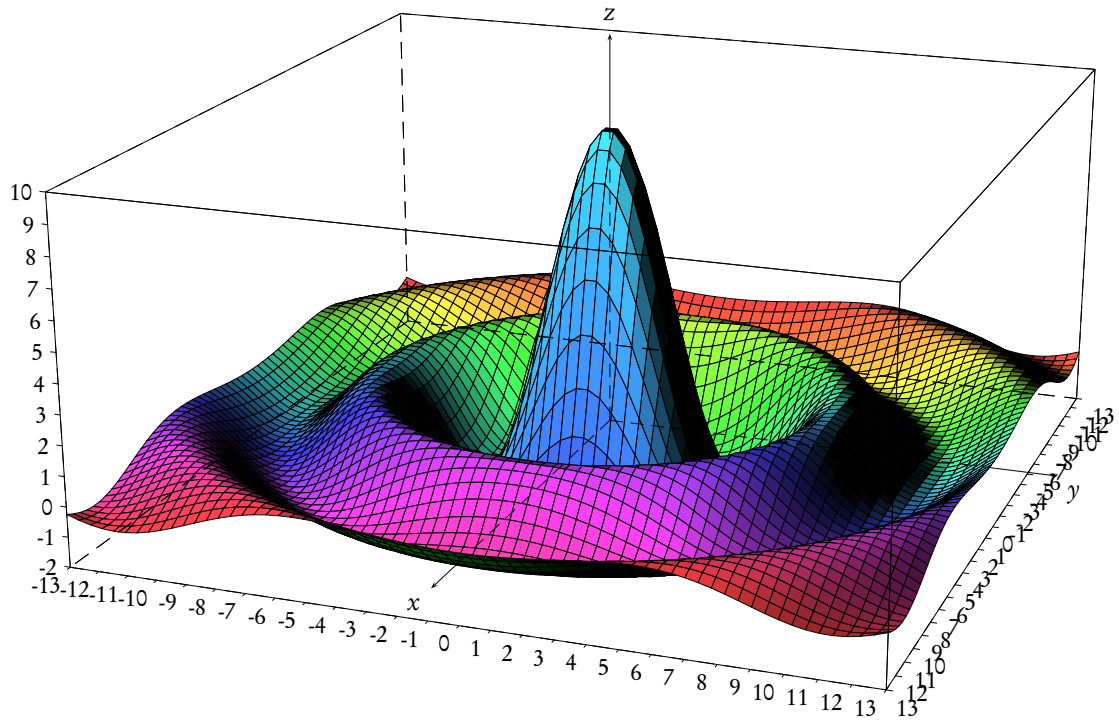
6.5 Exemple 4




```
\psset{unit=0.75}
\psset{lightsrc=30 -10 10}
\psset{viewpoint=50 20 30 rtp2xyz,Decran=70}
\begin{pspicture}(-7,-8)(7,8)
\psSurface[ngrid=.2 .2,algebraic,axesboxed,Zmin=-1,Zmax=1,
          linewidth=0.5\pslinewidth,spotX=r,spotY=d,spotZ=1,
          hue=0 1](-5,-5)(5,5){%
   sin((x^2+y^2)/3) }
\end{pspicture}
```

6.6 Exemple 5

Dans cet exemple, on montre comment colorier les facettes chacune avec une teinte différente en utilisant directement le code `postscript`.



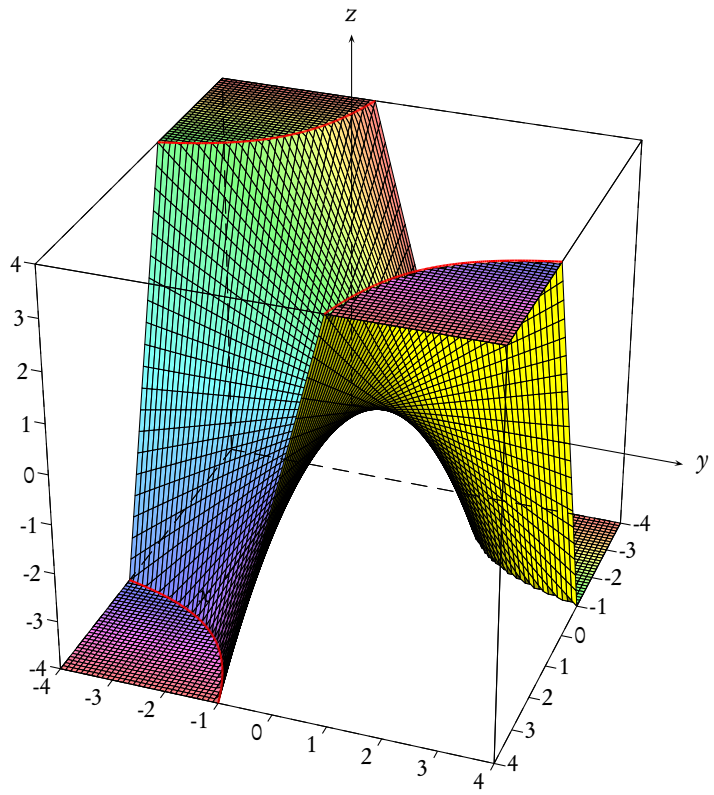

```
\psset{unit=0.5}
\psset{lightsrc=30 -10 10}
\psset{viewpoint=100 20 20 rtp2xyz,Decran=80}
\begin{pspicture}(-6,-12)(7,14)
\psSurface[ngrid=0.4 0.4,algebraic,axesboxed,Zmin=-2,Zmax=10,QZ=4,
          linewidth=0.25\pslinewidth,
          fcol=0 1 4225
          {/iF ED iF [iF 4225 div 0.75 1] (sethsbcolor) astr2str} for
         ](-13,-13)(13,13){%
   10*sin(sqrt((x^2+y^2)))/(sqrt(x^2+y^2)) }
\end{pspicture}
```

6.7 Exemple 6 : paraboloïde hyperbolique d'équation $z = xy$

Dans cet exemple, on combine le tracé de la surface et celui des contours de l'intersection du paraboloïde avec les plans $z = 4$ et $z = -4$. Pour cela on utilise `\psSolid[object=courbe]`.

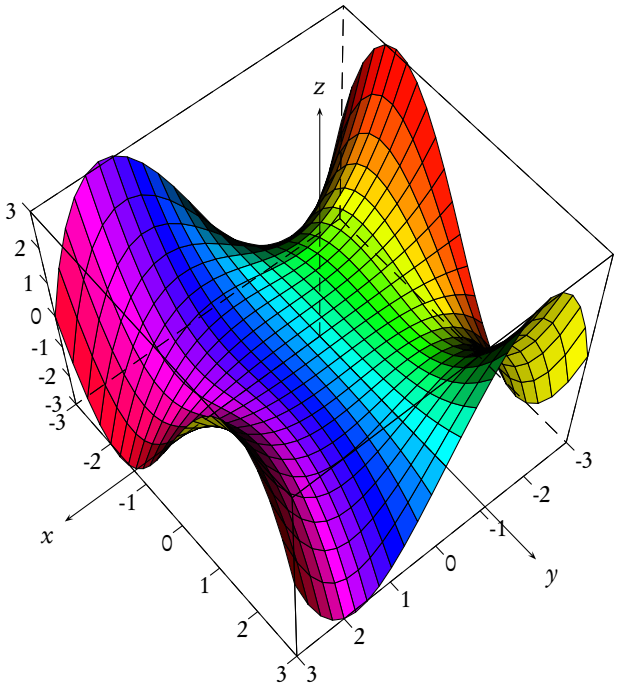
```
\defFunction{F}(t){t}{4 t div 4 min}{4}
\psSolid[object=courbe,range=1 4,
   linecolor=red,linewidth=2\pslinewidth,
   function=F]
```

On notera l'utilisation de deux fonctions `min` et `max`, qui permettent à partir d'un couple de valeurs, d'extraire la plus petite ou la plus grande.




```
\psSurface[hue=0 1,ngrid=.2 .5,incolor=yellow,axesboxed,
           Zmin=-4,Zmax=4,spotX=r](-4,-4)(4,4){x y mul 4 min -4 max}
```

6.8 Exemple 8 : surface d'équation $z = xy(x^2 + y^2)$




```
\psset{unit=0.5}
\psset{lightsrc=10 12 20,linewidth=0.5\pslinewidth}
\psset{viewpoint=30 50 60 rtp2xyz,Decran=50}
\begin{pspicture}(-10,-10)(10,10)
\psSurface[
   fillcolor=cyan!50,algebraic,axesboxed,
   ngrid=.25 .25,incolor=yellow,hue=0 1,
   Zmin=-3,Zmax=3](-3,-3)(3,3){%
   x*y*(x^2-y^2)*0.1}
\end{pspicture}
```

Chapitre 7

Utilisation avancée

7.1 Nommer un solide

Pour certaines utilisations, on a besoin de stocker un solide en mémoire afin de pouvoir y faire référence par la suite. Pour ce faire on dispose du booléen `solidmemory`, qui permet la transmission d'une variable tout au long de la scène.

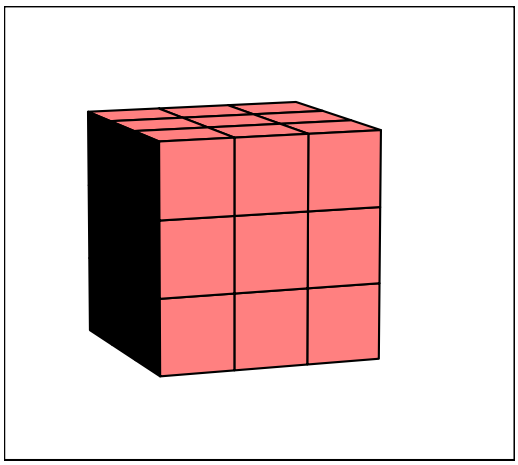
En revanche, l'activation de ce booléen désactive le dessin immédiat des macros `\psSolid`, `\psSurface` et `\psProjection`. Pour obtenir ce dessin, on utilise la macro `\composeSolid` à la fin de la scène.

Lorsque l'activation `\psset{solidmemory}` est faite, on peut alors utiliser l'option `[name=...]` de la macro `\psSolid`.

Dans l'exemple ci-dessous, on construit un solide coloré, que l'on sauvagarde sous le nom $A1$. On le dessine ensuite, après coup, en utilisant l'objet `[object=load]` avec le paramètre `[load=`*name*`]`.

À noter que l'instruction `linecolor=blue` utilisée lors de la construction de notre cube n'a pas d'impact sur le dessin : seule la structure du solide a été sauvegardé (sommets, faces, couleurs des faces), pas l'épaisseur de la ligne de tracé ou sa couleur ou la position de la source lumineuse. C'est au moment du dessin du solide considéré qu'il faut régler ces paramètres.

Enfin, on remarquera l'utilisation de l'option `[deactivatecolor]` qui permet au cube de garder sa couleur rouge d'origine (sinon les couleurs par défaut auraient repris le dessus dans l'objet `load`).



```
\psset{solidmemory}
\psSolid[object=cube,
      linecolor=blue,
      a=4,fillcolor=red!50,
      ngrid=3,
      action=none,
      name=A,
      ](0,0,0)
\psSolid[object=load,
   deactivatecolor,
   load=A]
\composeSolid
```

Avec l'option `solidmemory`, les noms de variables sont relativement bien encapsulés, et il n'y a pas de conflit avec les variables de dvips par exemple. Il reste par contre le risque de surcharge des noms utilisés par `solides.pro`. On peut utiliser tous les noms de variables à un seul carctère alphabétique, mais il faut éviter d'utiliser des noms comme `vecteur`, `distance`, `droite`, etc...qui sont déjà définis par le package.

7.2 Sectionner un solide par un plan

7.2.1 Tracer l'intersection d'un plan et d'un solide

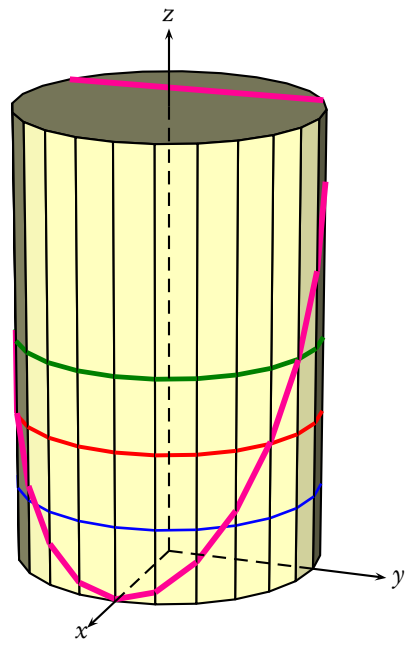
Les paramètres

C'est l'option `intersectionplan={[a b c d]}` qui permet de tracer l'intersection d'un plan et d'un solide. L'argument entre les crochets contient les paramètres du plan affine ayant pour équation : $ax + by + cz + d = 0$. Il est possible de dessiner l'intersection du solide avec plusieurs plans en plaçant à la suite les paramètres de ces plans comme dans l'exemple suivant.

Le tracé est activé avec `intersectiontype=0` ou toute autre valeur ≥ 0.

La couleur du tracé est choisie dans l'option `intersectioncolor=(bleu) (rouge) etc.`. On donne successivement dans l'ordre, l'épaisseur de chaque tracé par `intersectionlinewidth=1 2 etc.` (dimensions en picas).

Le tracé en traits discontinus des parties cachées sera activé avec `action=draw`.




```
\begin{pspicture}(-3,-2)(3,7)
\psset{lightsrc=viewpoint,viewpoint=50 20 20 rtp2xyz,Decran=50}
\psset{lightsrc=viewpoint}
\psSolid[object=cylindre,
   ngrid=1 24,
   r=2,
   fillcolor=yellow!25,
   intersectiontype=0,
   intersectionplan={
      [0 0 1 -1]
      [0 0 1 -2]
      [0 0 1 -3]
      [0.894 0 0.447 -1.8]},
   intersectioncolor=(bleu) (rouge) (vert) (rose),
   intersectionlinewidth=1 1.5 1.8 2.2]
\axesIIID(2,2,6)(3,3,7)
\end{pspicture}
```

7.2.2 Coupes d'un solide

Coupe du solide plein

L'objet étudié est un cylindre. Le plan qui coupe l'objet sera défini par :

```
plansepare={[a b c d]}
```

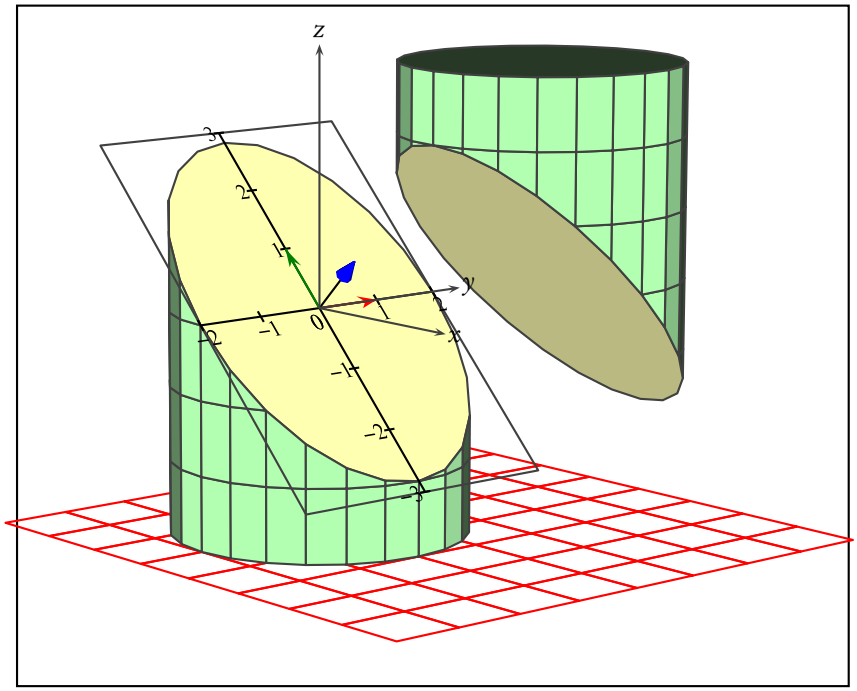
Les deux parties ne seront pas tracées mais mises en mémoire avec le nom commun `name=partiescylindre` :

```
\psset{solidmemory}
\psSolid[object=cylindre,
   r=2,h=6
   ngrid=6 24,
   plansepare={[0.707 0 0.707 0]},
```

```
   name=partiescylindre,
   action=none](0,0,-3)
```

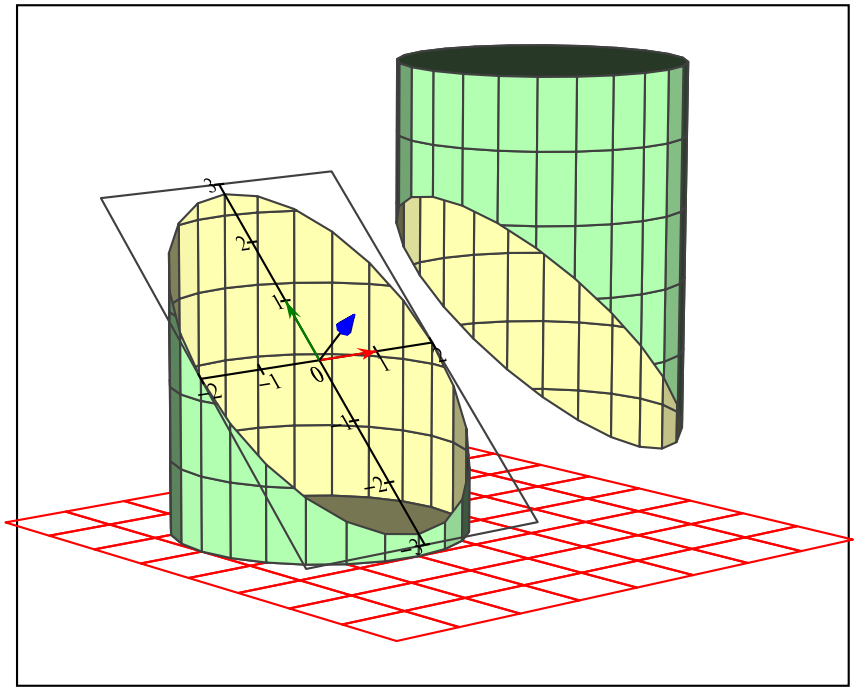
Puis affichées séparément avec leur indice respectif. C'est le sens de la normale du plan de séparation qui détermine le numérotage des deux parties : `0` celle qui est *au-dessus* de la normale et `1` celle qui est *au-dessous*. Pour les deux parties, la face de découpe porte le numéro `0`. S'il y a plusieurs faces de découpe, comme dans le cas du tore elles sont numérotées `0, 1 etc.`

```
\psSolid[object=load,
   load=partiescylindre1,
   fillcolor={[rgb]{0.7 1 0.7 }},
   fcol=0 (1 1 0.7 setrgbcolor)]
\psSolid[object=load,
   load=partiescylindre0,RotZ=60,
   fillcolor={[rgb]{0.7 1 0.7 }},
   fcol=0 (1 1 0.7 setrgbcolor)](0,4,0)
```



Coupe du solide creux

Les options `rm=0,hollow` permettent, l'une d'enlever la face de découpe `rm=0` et l'autre, `hollow` de voir l'intérieur.



7.2.3 Tranche d'une pyramide

Marquage des lignes de niveau et première découpe

Cette pyramide est créée comme un `object=new` en donnant le liste des coordonnées des sommets et des faces.

```
 sommets=
    0 -2 0 %% 0
    -2 0 0 %% 1
    0 4 0  %% 2
    4 0 0  %% 3
    0 0 5, %% 4
 faces={
  [3 2 1 0]
  [4 0 3]
  [4 3 2]
  [4 2 1]
}
```

Dans une première étape, facultative, on marque les lignes de découpe.

```
 intersectiontype=0,
 intersectionplan={[0 0 1 -1] [0 0 1 -2]},
 intersectionlinewidth=1 2,
 intersectioncolor=(bleu) (rouge)
```

Et on coupe la pointe supérieure, en dessinant aussi le plan de coupe.

```
\psSolid[object=new,
   sommets=
      0 -2 0 %% 0
      -2 0 0 %% 1
      0 4 0  %% 2
      4 0 0  %% 3
      0 0 5, %% 4
  faces={
     [3 2 1 0]
     [4 0 3]
     [4 3 2]
     [4 2 1]
     [4 1 0]},
   plansepare={[0 0 1 -2]},
   name=firstSlice,
   action=none]
\psSolid[object=load,action=draw*,
        load=firstSlice1]
\psSolid[object=plan,
       definition=equation,
       args={[0 0 1 -2]},
       base=-3 5 -3 5,action=draw]
```

Pour ne pas avoir à chaque fois à réécrire sommets et faces de la pyramide, on enregistre ses données dans les fichiers :

- Pyramid-couleurs.dat
- Pyramid-faces.dat
- Pyramid-sommets.dat
- Pyramid-io.dat

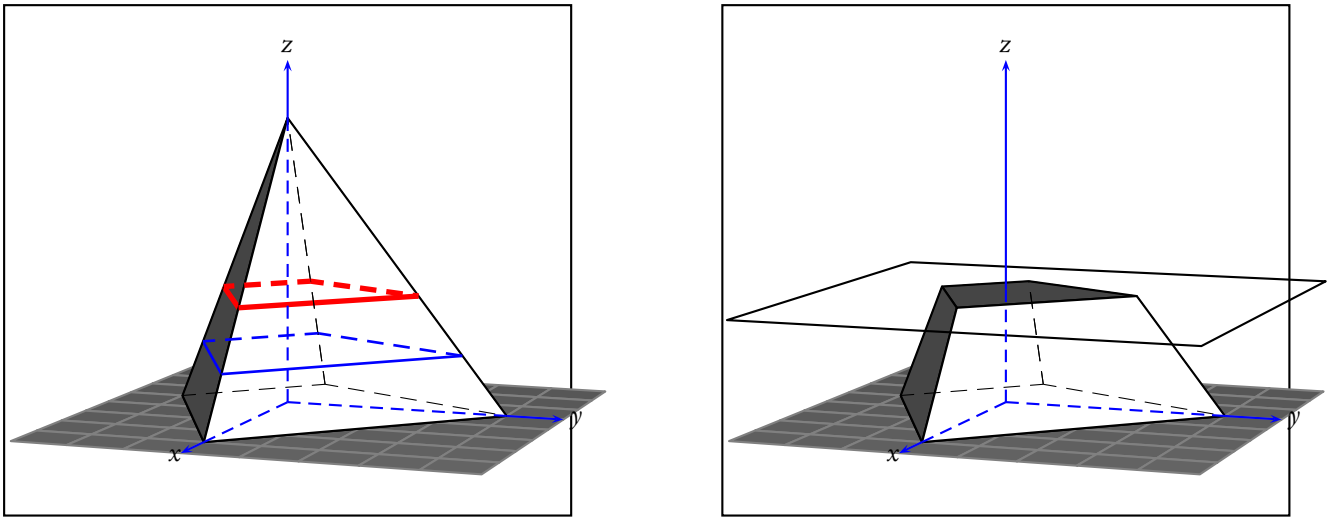
grâce à la commande `action=writesolid` :

```
\psSolid[object=new,
   sommets=
      0 -2 0 %% 0
      -2 0 0 %% 1
      0 4 0  %% 2
      4 0 0  %% 3
      0 0 5, %% 4
  faces={
     [3 2 1 0]
     [4 0 3]
     [4 3 2]
     [4 2 1]
     [4 1 0]
},file=Pyramid,fillcolor=yellow!50,
  action=writesolid]
```

Toutes ces lignes pourront alors être supprimées et par la suite, on appellera ces données avec la commande :

```
\psSolid[object=datfile,
```

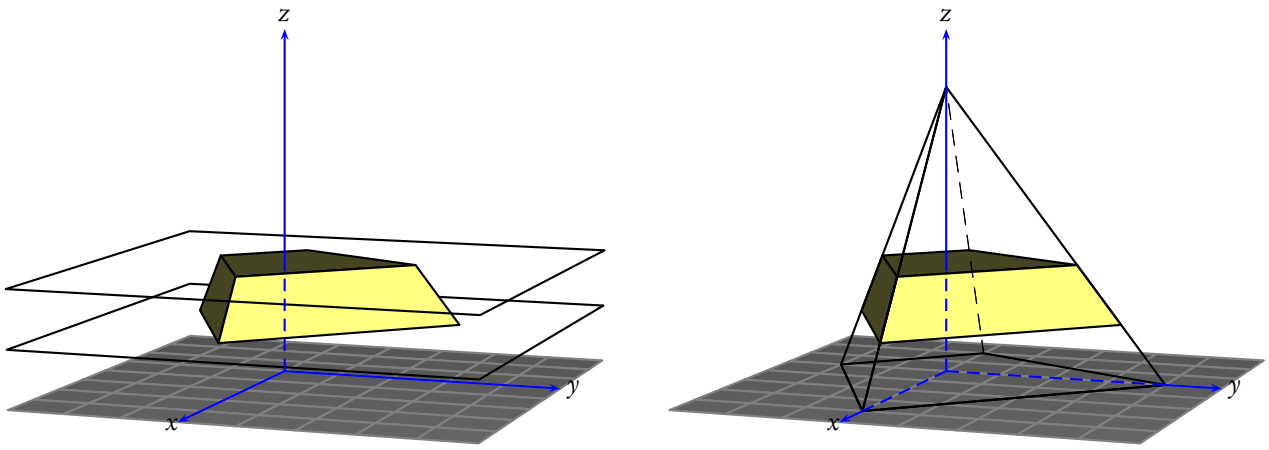
```
        file=Pyramid]
```



Deuxième découpe et son insertion dans la pyramide

Après avoir ôté la partie supérieure `firstSlice0` (elle n'apparaît plus), on découpe dans la partie restante, nommée `firstSlice1`, la base de la pyramide en gardant le haut `secondSlice0`, puis on enregistre la tranche de pyramide restante afin de l'insérer dans la pyramide en fil de fer :

```
\psset{solidmemory}
\psSolid[object=datfile,
         file=Pyramid,
   plansepare={[0 0 1 -2]},
   name=firstSlice,
   action=none]
\psSolid[object=load,
        load=firstSlice1,
        action=none,
        plansepare={[0 0 1 -1]},
        name=secondSlice]
\psSolid[object=load,action=draw*,
        load=secondSlice0]
\psSolid[object=load,
       load=secondSlice0,
       file=slicePyramid,
      action=writesolid]
\psSolid[object=datfile,fillcolor=yellow!50,
         file=slicePyramid]
```



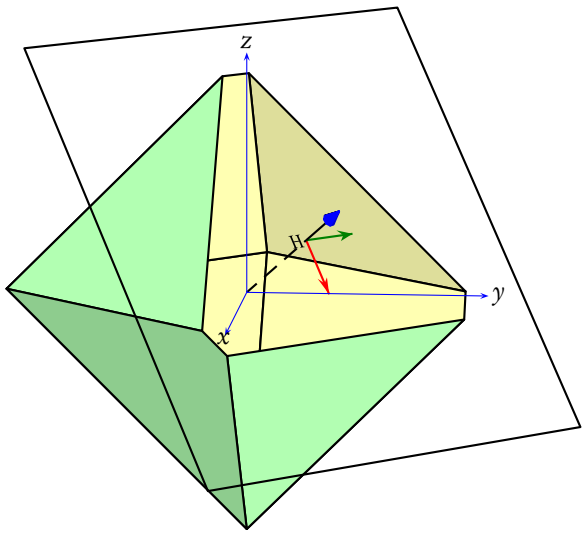
7.2.4 Coupe d'un octaèdre par un plan parallèle à l'une des faces

Voir l'intérieur

Rappelons que ce sont les options `rm=0,hollow` qui permettent, l'une d'enlever la face de découpe `rm=0` et l'autre `hollow` de voir l'intérieur.

Dans l'exemple ci-dessous, on commence par construire les objets nécessaires, sans les dessiner (`action=none`).

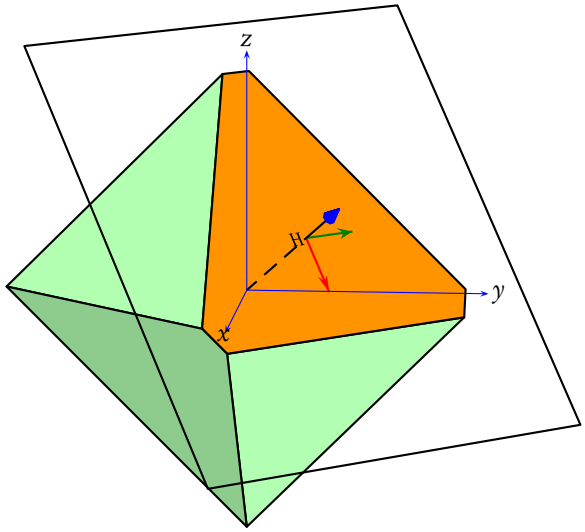
On construit l'octaèdre, on nomme G le centre de la face d'indice 1, puis on définit le point H vérifiant $\overrightarrow{OH} = 0,8\overrightarrow{OG}$. On définit ensuite le plan P passant par H et parallèle à la face d'indice 1 de l'octaèdre. On pratique ensuite la séparation de l'octaèdre par le plan P.




```
\begin{pspicture}(-3.5,-3)(4.5,5)
\psset{viewpoint=100 5 10 rtp2xyz,Decran=80,
   lightsrc=viewpoint,solidmemory,action=none}
\psSolid[object=octahedron,
   a=4,name=my_octahedron,]
\psSolid[object=point,
   definition=solidcentreface,
   args=my_octahedron 1,
   name=G,]
\psSolid[object=point,
   definition=mulv3d,
   args=G .8,
   name=H,]
\psSolid[object=plan,
   definition=solidface,
   args=my_octahedron 1,
   base=-4 4 -4 4,
   name=P,](H,,)
\psSolid[object=load,
   load=my_octahedron,
   plansepare=P,
   name=part]
\psSolid[object=load,load=part1,
        rm=0,hollow,action=draw**,
        fillcolor={[rgb]{0.7 1 0.7}},
        incolor={[rgb]{1 1 0.7}},]
\psSolid[object=plan,args=P,
      action=draw,showBase]
\psSolid[object=line,
   args=0 0 0 H,
   linestyle=dashed,]
\psProjection[object=point,plan=P,args=0 0,
    fontsize=20,pos=cl,text=H,phi=90,]
\axesIIID[linecolor=blue,linewidth=0.4pt](0,0,0)(4,4,4)
\end{pspicture}
```

On considère le solide comme plein

L'option `fcol=0 (YellowOrange)` permet de colorier la face de découpe qui est la face `0`.




```
\begin{pspicture}(-3.5,-3)(4.5,5)
\psset{viewpoint=100 5 10 rtp2xyz,Decran=80,
   lightsrc=viewpoint,solidmemory,action=none}
\psSolid[object=octahedron,
   a=4,name=my_octahedron,]
\psSolid[object=point,
   definition=solidcentreface,
   args=my_octahedron 1,
   name=G,]
\psSolid[object=point,
   definition=mulv3d,
   args=G .8,
   name=H,]
\psSolid[object=plan,
   definition=solidface,
   args=my_octahedron 1,
   base=-4 4 -4 4,
   name=P,](H,,)
\psSolid[object=load,
   load=my_octahedron,
   plansepare=P,
   name=part]
\psSolid[object=load,
   load=part1,
   fcol=0 (YellowOrange),
   action=draw**,
   fillcolor={[rgb]{0.7 1 0.7}},]
\psSolid[object=plan,args=P,
      action=draw,showBase]
\psSolid[object=line,
   args=0 0 0 H,
   linestyle=dashed,]
\psProjection[object=point,plan=P,args=0 0,
    fontsize=20,pos=cl,text=H,phi=90,]
\axesIIID[linecolor=blue,linewidth=0.4pt](0,0,0)(4,4,4)
\end{pspicture}
```

Les deux parties du solide découpé

On rappelle que c'est le sens de la normale du plan de séparation qui détermine le numérotage des deux parties : `0` celle qui est *au-dessus* de la normale et `1` celle qui est *au-dessous*. Pour les deux parties, la face de découpe porte le numéro `0`. S'il y a plusieurs faces de découpe, comme dans le cas du tore elles sont numérotées `0, 1 etc.`

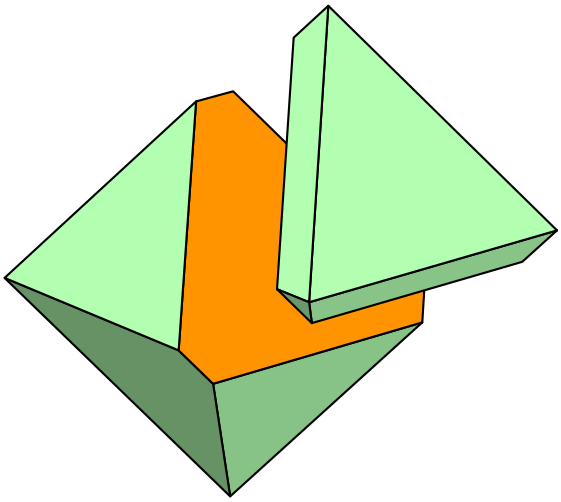
On opère en deux étapes, mise en mémoire des deux parties du solide découpé :

```
\psSolid[object=load,
   load=my_octahedron,
   plansepare=P,
   name=part]
```

Puis placement et traitement de chacune des parties :

```
\psSolid[object=load,
         fcol=0 (YellowOrange),
         fillcolor={[rgb]{0.7 1 0.7}},
        load=part1]
\psSolid[object=load,
```

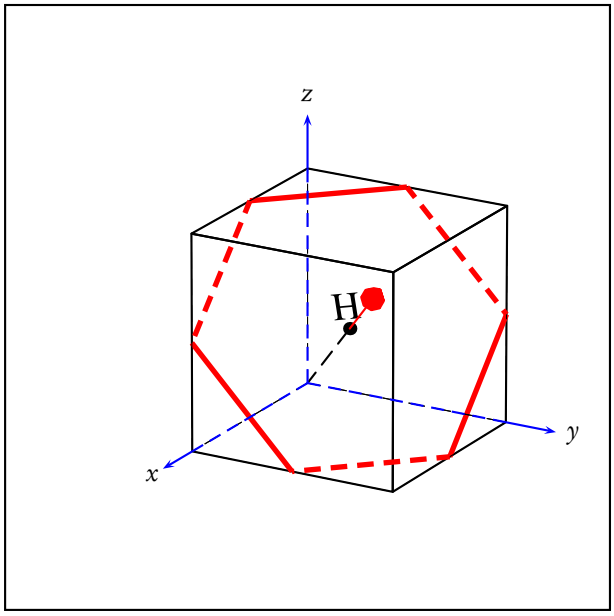
```
        fillcolor={[rgb]{0.7 1 0.7}},
       load=part0](H 2 mulv3d,,)
\composeSolid
```



```
\begin{pspicture}(-3.5,-3)(4.5,5)
\psset{viewpoint=100 5 20 rtp2xyz,Decran=150,
   lightsrc=viewpoint,solidmemory,action=none}
\psSolid[object=octahedron,
   a=2,name=my_octahedron,]
\psSolid[object=point,
   definition=solidcentreface,
   args=my_octahedron 1,
   name=G,]
\psSolid[object=point,
   definition=mulv3d,
   args=G .7,
   name=H,]
\psSolid[object=plan,
   definition=solidface,
   args=my_octahedron 1,
   base=-4 4 -4 4,
   name=P,](H,,)
\psSolid[object=load,
   load=my_octahedron,
   plansepare=P,
   name=part]
\psset{action=draw**}
\psSolid[object=load,
   load=part1,
   fcol=0 (YellowOrange),
   fillcolor={[rgb]{0.7 1 0.7}},]
\psSolid[object=load,
        fillcolor={[rgb]{0.7 1 0.7}},
       load=part0](H 2 mulv3d,,)
\composeSolid
\end{pspicture}
```

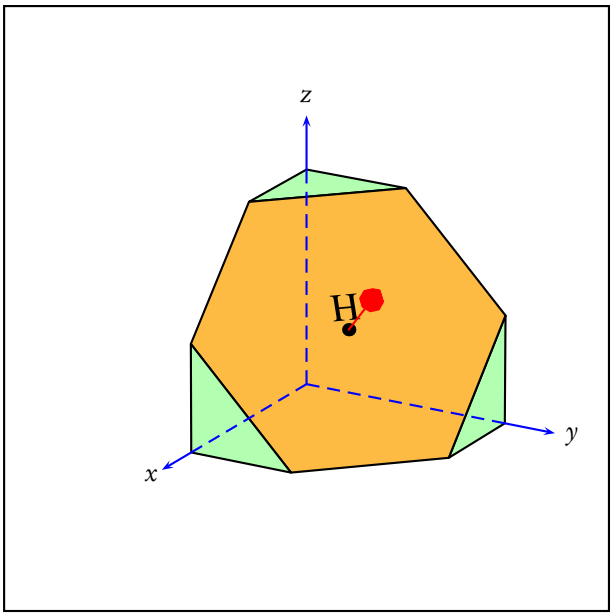
7.2.5 Coupes d'un cube

Marquage de la ligne de découpage




```
\psset{viewpoint=100 30 20 rtp2xyz,Decran=150}
\begin{pspicture}(-4,-3)(4,5)
\psframe(-4,-3)(4,5)
\psset{solidmemory}
\psSolid[object=plan,definition=normalpoint,
   args={1 1 1 [1 1 1]},action=none,name=P]
\psSolid[object=cube,a=2,action=draw,
   intersectiontype=0,
   intersectionplan=P,
   intersectionlinewidth=2,
   intersectioncolor=(rouge),
](1,1,1)
\psProjection[object=point,
   args=0 0,fontsize=10,pos=dc,
   text=H,phi=-30,plan=P,
]
\psSolid[object=line,
   linestyle=dashed,
   args=0 0 0 1 1 1]
\psSolid[object=vecteur,
   linecolor=red,
   args=1 1 1 .7 mulv3d](1,1,1)
\axesIIID[linecolor=blue](2,2,2)(2.5,2.5,2.5)
\end{pspicture}
```

Représentation du cube découpé avec une face de découpe hexagonale

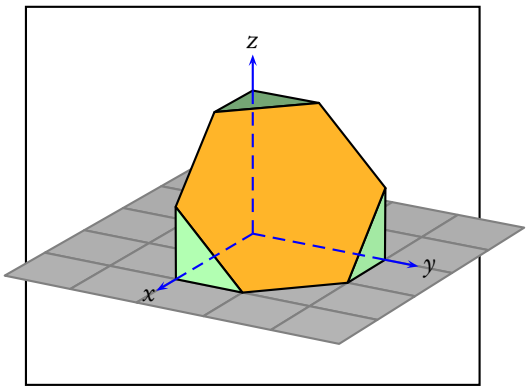


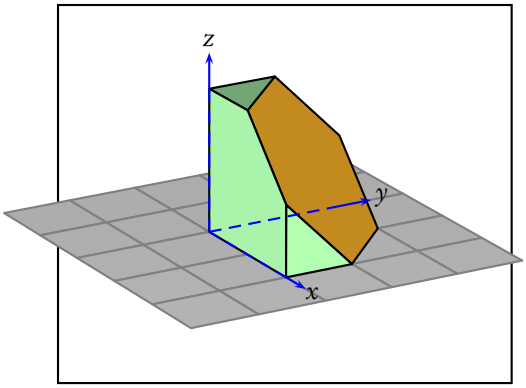

```
\psset{viewpoint=100 30 20 rtp2xyz,Decran=150}
\begin{pspicture}(-4,-3)(4,5)
\psframe(-4,-3)(4,5)
\psset{solidmemory}
\psSolid[object=plan,action=none,definition=
  normalpoint,
   args={1 1 1 [1 1 1]},name=P]
\psSolid[object=cube,a=2,
   plansepare=P,
   action=none,
   name=parts_cube,
](1,1,1)
\psSolid[object=load,
        load=parts_cube1,
        fcol=0 (Dandelion),
        fillcolor={[rgb]{0.7 1 0.7}},
]
\psProjection[object=point,
   args=0 0,fontsize=10,pos=dc,
   text=H,phi=-30,plan=P,
]
\psSolid[object=vecteur,
   linecolor=red,
   args=1 1 1 .7 mulv3d](1,1,1)
\axesIIID[linecolor=blue](2,2,2)(2.5,2.5,2.5)
\end{pspicture}
```

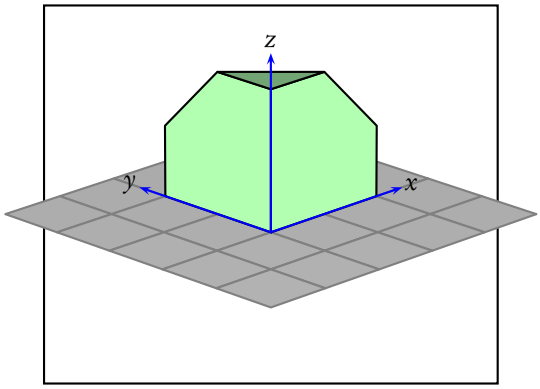
Cube découpé dans des différentes positions

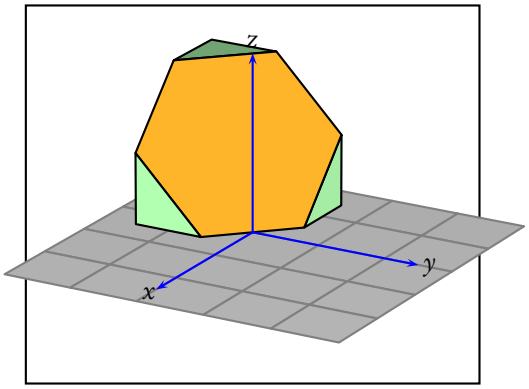
Où on utilise l'option permettant de mémoriser un solide pour, après diverses transformations, faire reposer le cube tronqué sur sa face de découpe.

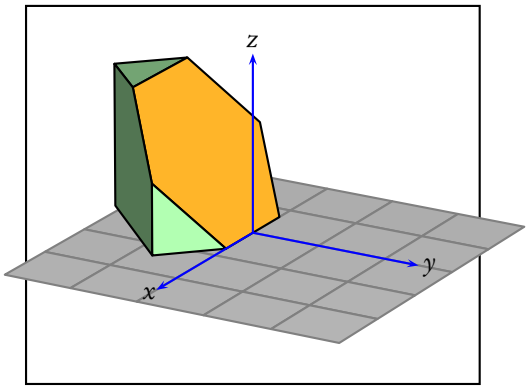
```
\psset{solidmemory}
\psSolid[object=datfile,
         fcol=0 (Dandelion),
         fillcolor={[rgb]{0.7 1 0.7}},
         name=C1,
         action=none,
         file=cubeHexagone]
```

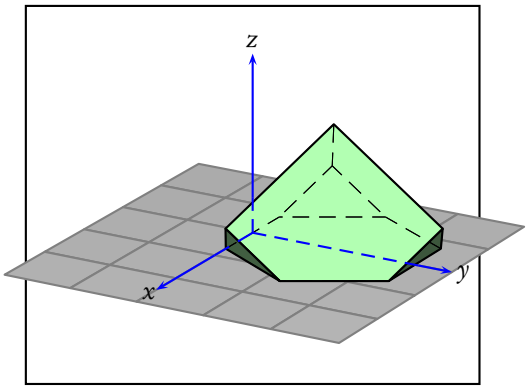






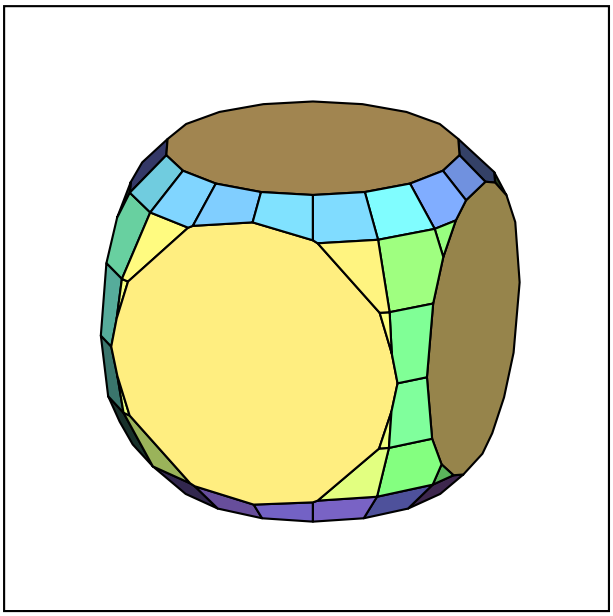






7.2.6 Sections multiples

Coupes dans une sphère avec `PStricks`

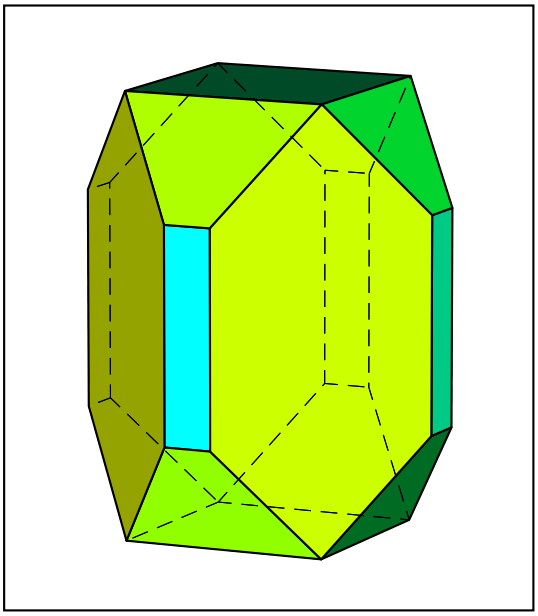


```
\begin{pspicture}(-4,-4)(4,4)
\psframe(-4,-4)(4,4)
\psset{viewpoint=100 20 20 rtp2xyz,Decran=75}
\psset{solidmemory,lightsrc=viewpoint}
\codejps{
   /coeff 0.75 def /r0 4 def /OH coeff r0 mul neg def}
      %
\psSolid[object=sphere,
   r=r0,ngrid=9 18,
   plansepare={[1 0 0 OH]},
   name=part,
   action=none]
\psSolid[object=load,
   load=part1,plansepare={[-1 0 0 OH]},action=none,
      name=part]
\psSolid[object=load,
   load=part1,plansepare={[0 1 0 OH]},action=none,name
      =part]
\psSolid[object=load,
   load=part1,plansepare={[0 -1 0 OH]},action=none,
      name=part]
\psSolid[object=load,
   load=part1,plansepare={[0 0 1 OH]},action=none,name
      =part]
\psSolid[object=load,
   load=part1,plansepare={[0 0 -1 OH]},action=none,
      name=part]
\psSolid[object=load,hue=.1 .8 0.5 1,
   load=part1](0,0,0)
\composeSolid
\end{pspicture}
```

Sections multiples d'un parallélépipède

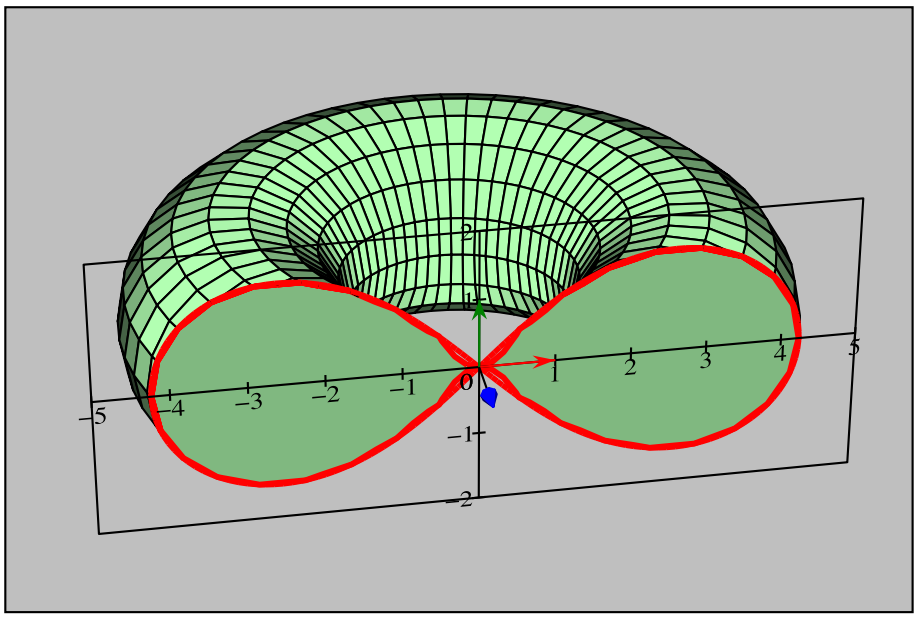
Les sections multiples gagneront à être exécutées dans une boucle `postscript`, dans `\codejps`, c'est plus simple et plus rapide !

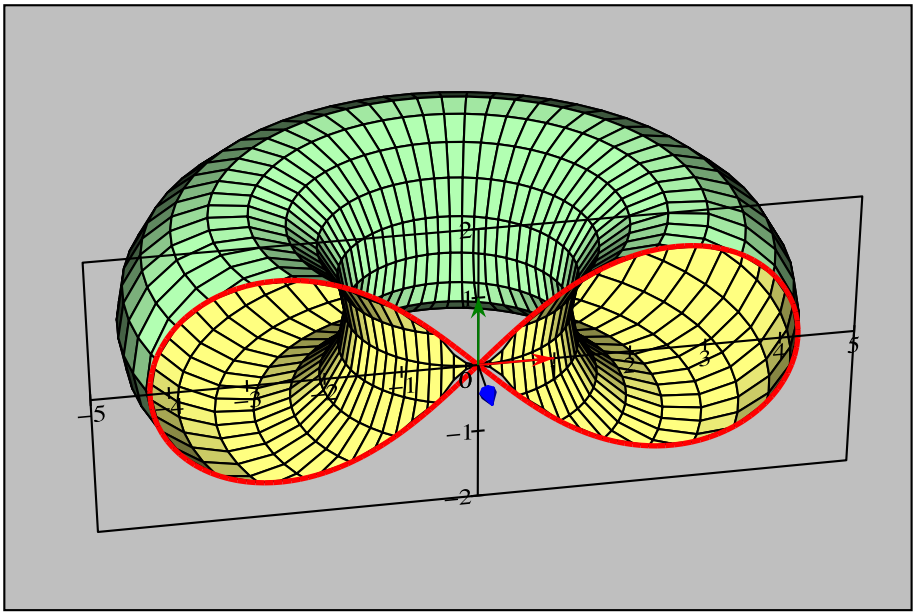
Dans cet exemple, le solide de départ est un parallélépipède. Troncatures des sommets et chanfreinages des arêtes sont réalisées avec des plans de coupe successifs, d'abord les sommets puis les arêtes.



```
\begin{pspicture}(-3.5,-4)(3.5,4)
\psset{viewpoint=100 -20 10 rtp2xyz,Decran=100}
%\lightsource
\psset{lightsrc=viewpoint}
\psframe(-3.5,-4)(3.5,4)
\codejps{
4 4 6 newparallelepiped
45 90 360 {
/iAngle exch def
  /n_x iAngle cos 35.2644 cos mul def
  /n_y iAngle sin 35.2644 cos mul def
  /n_z 35.2644 sin def
  /distance 2 3 add 3 sqrt div neg def
[ n_x n_y n_z distance]
solidplansepare
} for
45 90 360 {
/iAngle exch def
  /n_x iAngle cos 35.2644 cos mul def
  /n_y iAngle sin 35.2644 cos mul def
  /n_z 35.2644 sin neg def
  /distance 2 3 add 3 sqrt div neg def
[ n_x n_y n_z distance]
solidplansepare
} for
45 90 360 {
/iAngle exch def
% plan : ax+by+cz-d=0
[ iAngle cos % a
  iAngle sin % b
  0 % c
  -2.5 % -d
] solidplansepare
} for
dup [.5 .2] solidputhuecolors
solidlightOn
drawsolid*}
\end{pspicture}
```

7.2.7 Sections d'un tore





7.2.8 Autres exemples

1. Vous trouverez une version codée de ce document en `jps` dans la commande `\codejps` dans le document suivant :
   http://melusine.eu.org/syracuse/mluque/solides3d2007/sections
2. Une étude des sections coniques sur :
   http://melusine.eu.org/syracuse/mluque/solides3d2007/sections/sections-cone
3. Une étude des sections cylindriques sur :
   http://melusine.eu.org/syracuse/mluque/solides3d2007/sections/section-cylindre
4. Une étude sur les sections du tore :
   http://melusine.eu.org/syracuse/mluque/solides3d2007/sections/section-tore

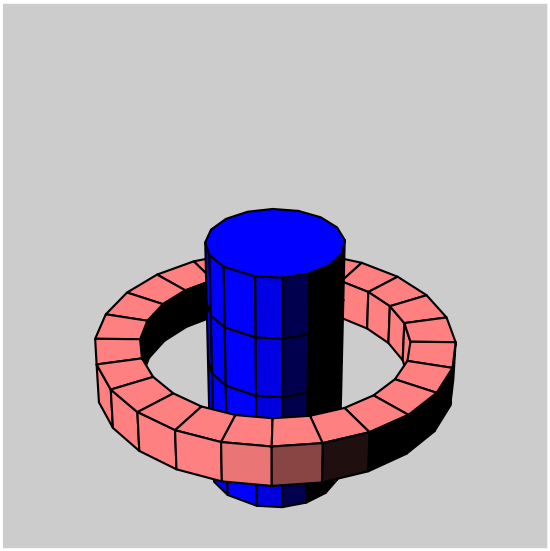
7.3 Fusionner des solides

Il est possible d'assembler plusieurs solides en une seule structure : c'est l'opération de *fusion* des solides. Cette technique permet d'appliquer l'algorithme du peintre à toute une scéne.

Pour ce faire, il faut activer la possibilité de mettre en mémoire par `\psset{solidmemory}`, puis construire ses différents solides avec `\psSolid`, sans omettre de leur donner un nom chacun.

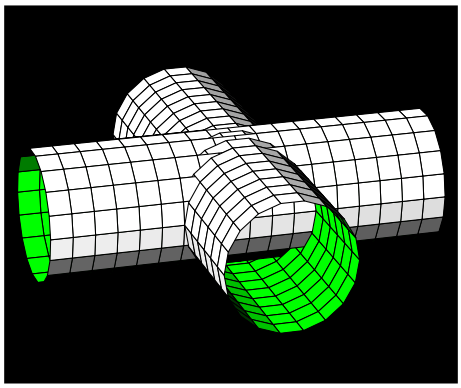
On utilise ensuite l'objet `fusion` de `\psSolid`, en indiquent dans le paramètre `base` la liste des noms des solides à fusionner.

Pour visualiser la scène, ne pas oublier de conclure par `\composeSolid`.



```
\psset{solidmemory}
\psSolid[object=cylindre,h=6,fillcolor=blue,
      r=1.5,
      ngrid=4 16,
      action=none,
      name=A1,
      ](0,0,-4)
\psSolid[object=anneau,h=6,fillcolor=red!50,
      R=4,r=3,h=1,
      action=none,
```

```
      name=B1,
      ](0,0,-1)
\psSolid[object=fusion,
      action=draw**,
      base=A1 B1,
      ](0,0,0)
\composeSolid
```



```
\psset{solidmemory}
\psset{lightsrc=50 -50 50,
   viewpoint=100 -30 40,
   Decran=100,linewidth=0.5\pslinewidth,
   ngrid=18 18,fillcolor=white,
   h=12,r=2,RotX=90}
\psframe*[linecolor=black](-6,-5)(6,5)
\psSolid[object=cylindrecreux,
   action=none,
   name=cylindre1](0, 6, 0)
\psSolid[object=cylindrecreux,
   RotZ=90,
   action=none,
   name=cylindre2](-6, 0, 0)
\psSolid[object=fusion,
   base=cylindre1 cylindre2,RotX=0]
\composeSolid
```

7.4 Fusion avec le code jps

On peut également opérer la fusion de solides en passant directement par le code jps. Le calcul des parties cachées est effectué par les routines du code `PostScript` du fichier `solides.pro`, mais les lignes de code sont “encapsulées” dans un environnement `pspicture` grâce à la commande `\codejps{code ps}`.

7.4.1 Le code `jps`

Le choix de l'objet

- [section] n newanneau : choix de l'anneau cylindrique défini par sa section, coordonnées des sommets dans le plan Oyz.
- 2␣1.5␣6␣[4␣16]␣newcylindre : choix du cylindre vertical avec comme caractéristiques :
  - `rayon=1.5`;
  - `z0=2` est la position du centre de la base inférieure sur l'axe Oz;
  - `z1=6` est la position du centre de la base supérieure sur l'axe Oz;
  - `[4 16]` : le cylindre est découpé verticalement en 4 morceaux et horizontalement en 16 secteurs.

Les transformations

- `{-1␣2␣5␣translatepoint3d} solidtransform` : l'objet préalablement sélectionné subit une translation au point de coordonnées ($x=-1, y=2, z=5$).
- `{90␣0␣45␣rotateOpoint3d} solidtransform` : l'objet préalablement sélectionné subit une rotation autour des axes (Ox, Oy, Oz), dans cet ordre, de 90° autour de (Ox) suivie d'une rotation de 45° autour de (Oz).

Le choix de la couleur de l'objet

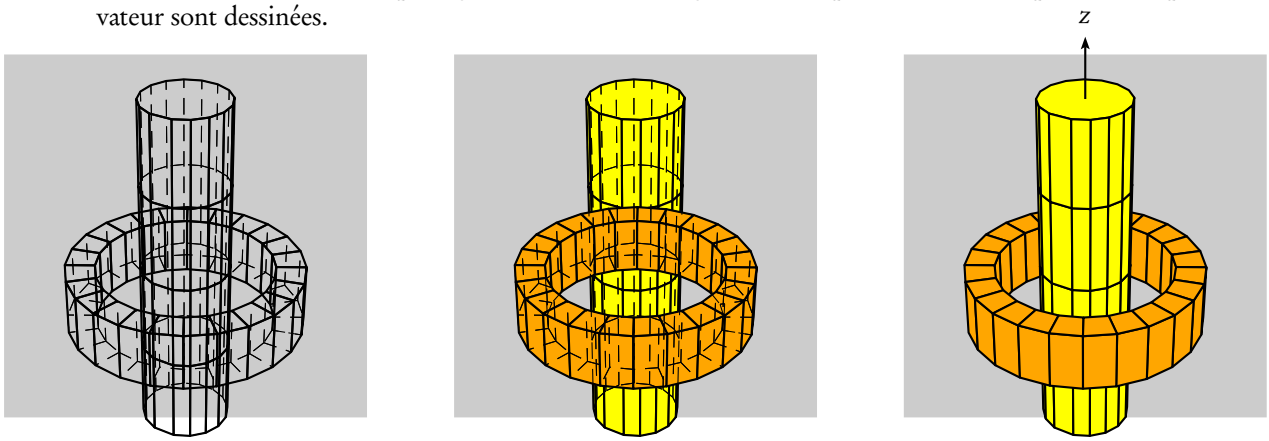
- dup (jaune) outputcolors : l'objet de couleur jaune éclairé en lumière blanche.

La fusion des objets

- Elle se fait avec l'instruction `solidfuz`.

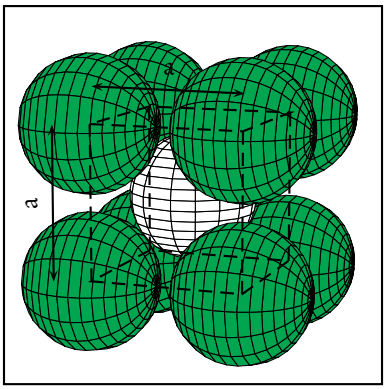
Le dessin des objets

- Le tracé a trois options :
  - `drawsolid` : tracé des arêtes uniquement, les arêtes cachées sont en pointillés ;
  - `drawsolid*` : tracé et remplissage des solides dans l'ordre de leur programmation (option peu intéressante à priori), avec le dessin des arêtes cachées en pointillés ;
  - `drawsolid**` : tracé et remplissage des solides avec l'algorithme du peintre : seules les parties vues par l'observateur sont dessinées.




```
\psset{lightsrc=50 -50 50,viewpoint=50 20 50 rtp2xyz,Decran=50}
\begin{pspicture}(-6,-2)(6,8)
\psframe(-6,-2)(6,8)
\codejps{
% solide 1
 /tour {
    -6 1.5 6 [4 16] newcylindre
    dup (jaune) outputcolors
    } def
% solide 2
 /anneau {
    [4 -1 4 1 3 1 3 -1] 24 newanneau
   {0 0 -1 translatepoint3d} solidtransform
    dup (orange) outputcolors
    } def
% fusion
    tour anneau solidfuz
    drawsolid**}
\end{pspicture}
```

7.4.2 Un ion chlorure




```
\codejps{
/Cl {9.02  [12 8] newsphere
 {-90 0 0 rotateOpoint3d} solidtransform
 dup (Green) outputcolors} def
/Cl1 { Cl {10.25 10.25 10.25 translatepoint3d} solidtransform } def
/Cl2 { Cl {10.25 -10.25 10.25 translatepoint3d} solidtransform } def
/Cl3 { Cl {-10.25 -10.25 10.25 translatepoint3d} solidtransform } def
/Cl4 { Cl {-10.25 10.25 10.25 translatepoint3d} solidtransform } def
/Cl5 { Cl {10.25 10.25 -10.25 translatepoint3d} solidtransform } def
/Cl6 { Cl {10.25 -10.25 -10.25 translatepoint3d} solidtransform } def
/Cl7 { Cl {-10.25 -10.25 -10.25 translatepoint3d} solidtransform } def
/Cl8 { Cl {-10.25 10.25 -10.25 translatepoint3d} solidtransform } def
/Cs {8.38  [12 8] newsphere
 dup (White) outputcolors} def
/Cl12{ Cl1 Cl2 solidfuz} def
/Cl123{ Cl12 Cl3 solidfuz} def
/Cl1234{ Cl123 Cl4 solidfuz} def
/Cl12345{ Cl1234 Cl5 solidfuz} def
/Cl123456{ Cl12345 Cl6 solidfuz} def
/Cl1234567{ Cl123456 Cl7 solidfuz} def
/Cl12345678{ Cl1234567 Cl8 solidfuz} def
/C_Cs { Cl12345678 Cs solidfuz} def
C_Cs  drawsolid**}
```

On définit l'ion chlorure Cl^- :

```
/Cl {9.02  [12 8] newsphere
 {-90 0 0 rotateOpoint3d} solidtransform
 dup (Green) outputcolors} def
```

que l'on recopie aux sommets du cube :

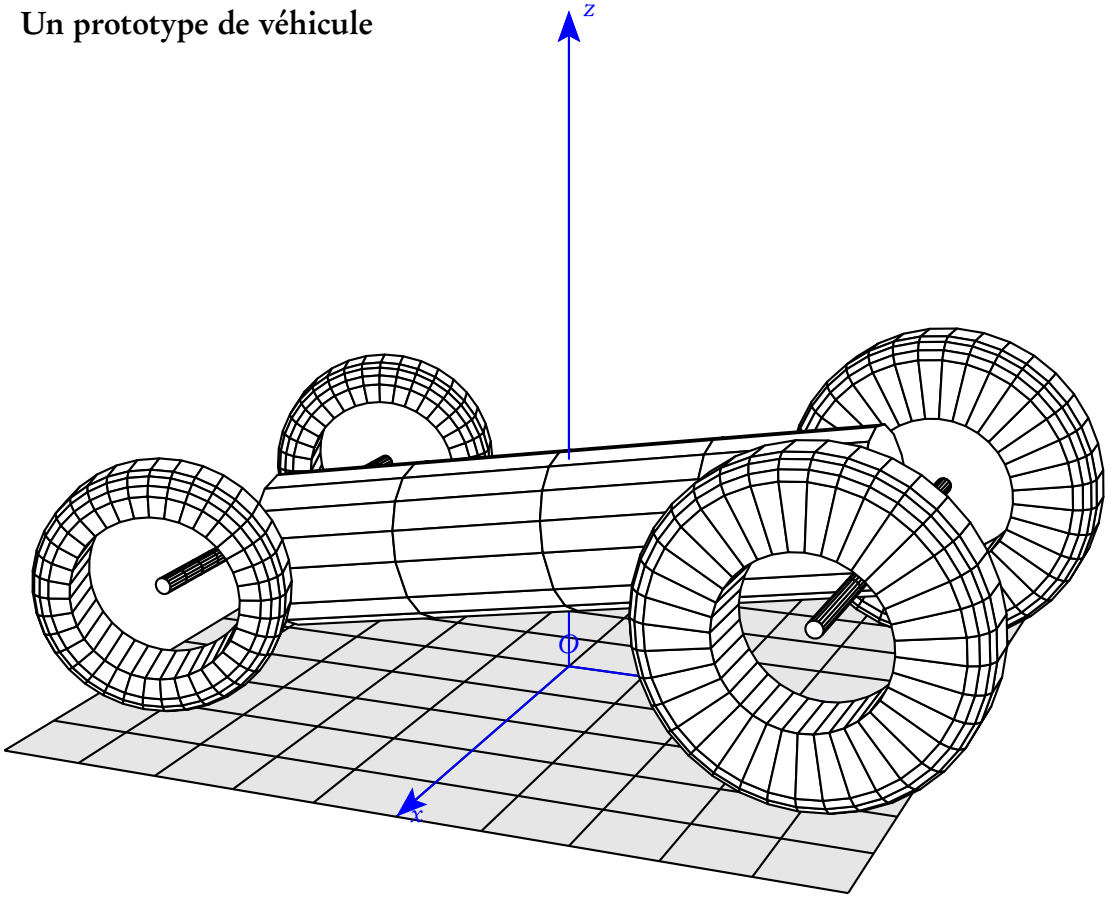
```
/Cl1 { Cl {10.25 10.25 10.25 translatepoint3d} solidtransform } def
/Cl2 { Cl {10.25 -10.25 10.25 translatepoint3d} solidtransform } def
/Cl3 { Cl {-10.25 -10.25 10.25 translatepoint3d} solidtransform } def
/Cl4 { Cl {-10.25 10.25 10.25 translatepoint3d} solidtransform } def
/Cl5 { Cl {10.25 10.25 -10.25 translatepoint3d} solidtransform } def
/Cl6 { Cl {10.25 -10.25 -10.25 translatepoint3d} solidtransform } def
/Cl7 { Cl {-10.25 -10.25 -10.25 translatepoint3d} solidtransform } def
/Cl8 { Cl {-10.25 10.25 -10.25 translatepoint3d} solidtransform } def
```

Puis l'ion césium Cs^+, placé au centre :

```
/Cs {8.38  [12 8] newsphere
 dup (White) outputcolors} def
```

Ensuite on fusionne deux par deux les différentes sphères.

7.4.3 Un prototype de véhicule



Il faut opérer en plusieurs étapes en fusionnant les solides deux par deux.

- On fusionne d'abord les deux roues avant `roue12` :

```
/roue12 {
% solide 1
    /R 2 def /r 1 def /h 1 def
    [Pneu] 36 newanneau
     {90 0 90 rotateOpoint3d} solidtransform
     {3 4 2 translatepoint3d} solidtransform
     dup (White) outputcolors
% solide 2
    [Pneu] 36 newanneau
    {90 0 90 rotateOpoint3d} solidtransform
   {-3 4 2 translatepoint3d} solidtransform
    dup (White) outputcolors
% fusion
    solidfuz } def
```

- Puis ces deux roues et leur axe :

```
/axe12{
0 0.1 6  [4 16] newcylindre
{90 0 90 rotateOpoint3d} solidtransform
{-3 4 2  translatepoint3d} solidtransform
dup (White) outputcolors
} def
/roue12axes {
```

```
roue12 axe12 solidfuz } def
```

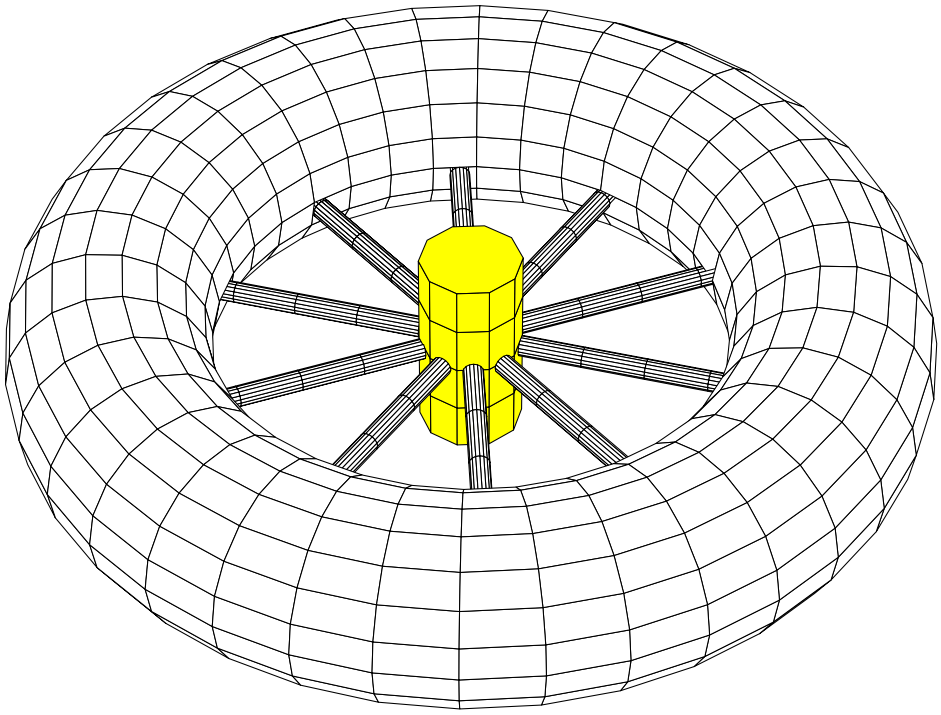
- On opère de même pour les roues arrière et leur axe :

```
/roue34 {
% solide 3
   /R 1.5 def /r 1 def /h 1 def
    [Pneu] 36 newanneau
    {90 0 110 rotateOpoint3d} solidtransform
   {3 -4 1.5 translatepoint3d} solidtransform
    dup (White) outputcolors
% solide 4
    [Pneu] 36 newanneau
   {90 0 110 rotateOpoint3d} solidtransform
   {-3 -4 1.5 translatepoint3d} solidtransform
    dup (White) outputcolors
% fusion
    solidfuz } def
/axe34{
0 0.1 6 [16 16] newcylindre
{90 0 90 rotateOpoint3d} solidtransform
{-3 -4 1.5  translatepoint3d} solidtransform
dup (White) outputcolors
} def
/roue34axes34 {
roue34 axe34 solidfuz } def
/roues {roue34axes34 roue12axes solidfuz} def
```

- La dernière étape consiste à fusionner les deux solides ainsi obtenus avec le semblant de chassis :

```
/chassis {
0 1 8  [4 16] newcylindre
{100 0 0 rotateOpoint3d} solidtransform
{0 4 2.5  translatepoint3d} solidtransform
dup (White) outputcolors
} def
roues chassis solidfuz
    drawsolid**}
```

7.4.4 Une roue ou bien une station spatiale !



On définit d'abord le premier rayon :

```
/rayon0 {
    1 0.2 6 [4 16] newcylindre
    {90 0 0 rotateOpoint3d} solidtransform
     dup (White) outputcolors
  } def
```

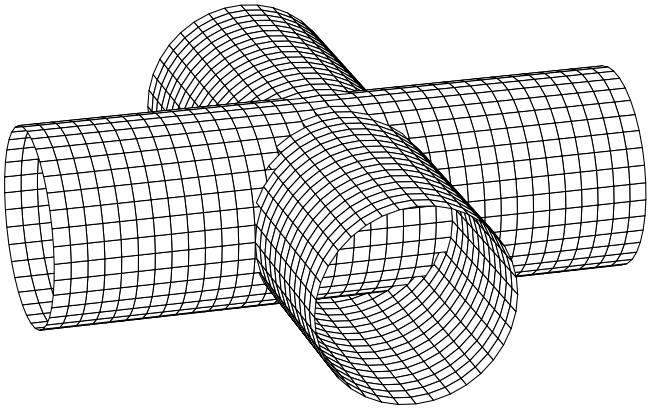
Puis dans une boucle on fusionne tous les rayons de la roue :

```
36 36 360 {
    /angle exch def
  /rayon1 {
     1 0.2 6  [4 16] newcylindre
     {90 0 angle rotateOpoint3d} solidtransform
      dup (White) outputcolors
   } def
  /rayons {rayon0 rayon1 solidfuz} def
  /rayon0 rayons def
  } for
```

Ensuite, on dessine le moyeu et la circonférence (pneu) de la roue pour enfin fusionner l'ensemble :

```
/moyeu { -0.5 1 0.5  [4 10] newcylindre dup (White) outputcolors} def
/rayonsmoyeu {rayons  moyeu solidfuz} def
/pneu {2 7 [18 36] newtore dup (jaune) outputcolors} def
/ROUE {pneu rayonsmoyeu solidfuz} def
 ROUE  drawsolid**
```

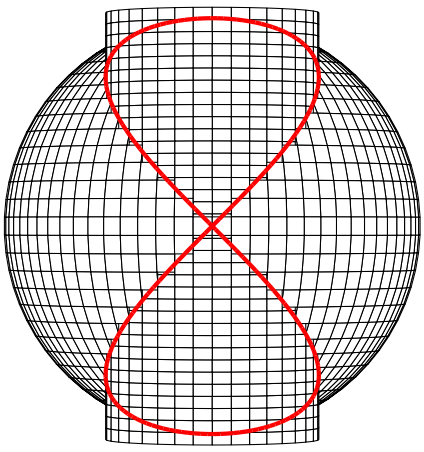
7.4.5 Intersection de deux cylindres



```
\codejps{
 /cylindre1 {
     -6 2 6 [36 36] newcylindrecreux %newcylindre
     {90 0 0 rotateOpoint3d} solidtransform
      dup (White) (White) inoutputcolors
   } def
 /cylindre2 {
     -6 2 6 [36 36] newcylindrecreux% newcylindre
     {90 0 90 rotateOpoint3d} solidtransform
      dup (White) (White) inoutputcolors
   } def
 /UnionCylindres {cylindre1 cylindre2 solidfuz} def
  UnionCylindres  drawsolid**}
```

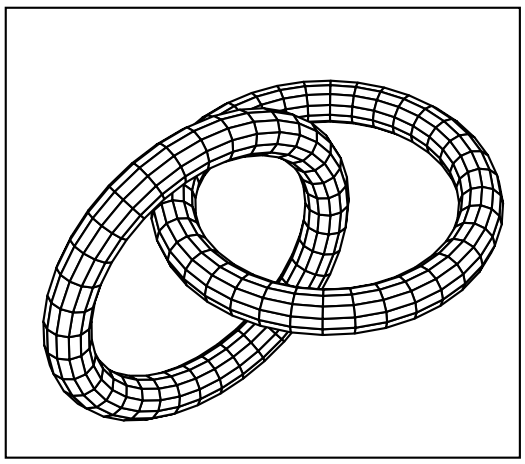
7.4.6 Intersection d'une sphère et d'un cylindre

Dans cette partie on dessine en utilisant `\psSolid[object=courbe]` le contour de l'intersection.



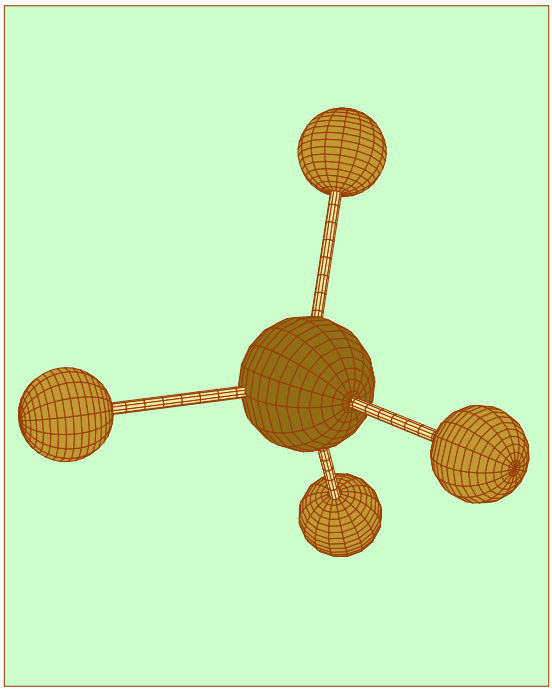
```
\codejps{%
   /cylindre1 {
       -5 2.5 5 [36 36] newcylindre
       {2.5 0 0 translatepoint3d} solidtransform
        dup (White) outputcolors
   } def
   /sphere1 {
        5 [36 72] newsphere
        dup (White) outputcolors
   } def
   /CS {cylindre1 sphere1 solidfuz} def
   CS drawsolid**}
\psPoint(0,0,0){O}
\psSolid[object=courbe,r=0,
   function=F,
   range=0 360,
   linecolor=red,linewidth=4\pslinewidth]
```

7.4.7 Réunion de deux anneaux



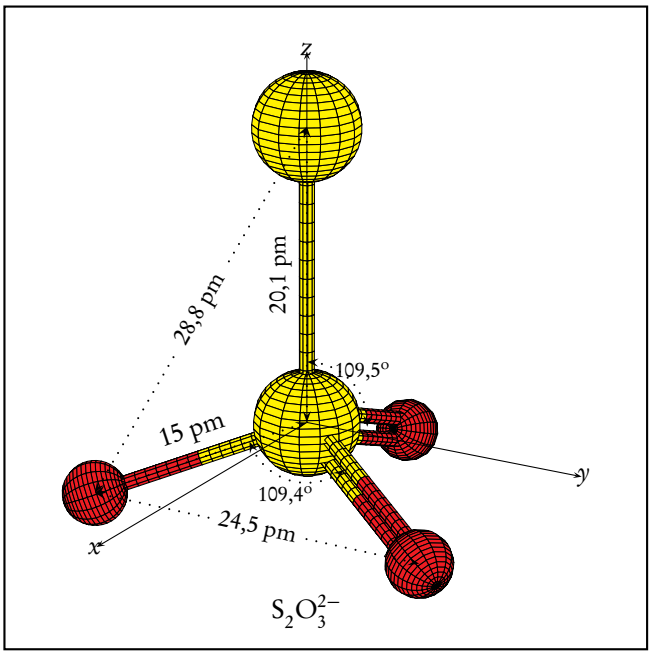
```
\codejps{
 /anneau1 {1 7 [9 18] newtore
 {0 0 0 translatepoint3d} solidtransform
 dup (White) outputcolors} def
 /anneau2 {1 7 [9 18] newtore
 {90 0 0 rotateOpoint3d} solidtransform
 {7 0 0 translatepoint3d} solidtransform
 dup (White) outputcolors} def
 /collier {anneau1 anneau2 solidfuz} def
  collier  drawsolid**}
```

7.4.8 La molécule de méthane : modèle en bois



```
\pstVerb{/hetre {0.764 0.6 0.204 setrgbcolor} def
         /chene {0.568 0.427 0.086 setrgbcolor} def
         /bois {0.956 0.921 0.65 setrgbcolor} def
         }%
\codejps{
 /H1 {
 2  [18 16] newsphere
 {-90 0 0 rotateOpoint3d} solidtransform
 {0 10.93 0 translatepoint3d} solidtransform
 dup (hetre) outputcolors} def
  /L1 {
     0 0.25 10  [12 10] newcylindre
     {-90 0 0 rotateOpoint3d} solidtransform
      dup (bois) outputcolors
   } def
 /HL1{ H1 L1  solidfuz} def
 /HL2 {
   HL1 {0 0 -109.5 rotateOpoint3d} solidtransform
   } def
 /HL3 {
   HL2 {0 -120 0 rotateOpoint3d} solidtransform } def
 /HL4 {
   HL2 {0 120 0 rotateOpoint3d} solidtransform } def
 /C {3  [18 16] newsphere
   {90 0 0 rotateOpoint3d} solidtransform
    dup (chene) outputcolors} def
 /HL12 { HL1 HL2 solidfuz} def
 /HL123 { HL12 HL3 solidfuz} def
 /HL1234 { HL123 HL4 solidfuz} def
 /methane { HL1234 C solidfuz} def
 methane drawsolid**}
```

7.4.9 L'ion thiosulfate



On définit d'abord les deux atomes de soufre placés sur l'axe Oz. S_1 est placé en O.

```
\codejps{
/Soufre1 {3.56  [20 16] newsphere
 dup (Yellow) outputcolors} def
/Soufre2 {3.56  [20 16] newsphere
 {0 0.000 20.10 translatepoint3d} solidtransform
 dup (Yellow) outputcolors} def
```

Puis la liaison simple S-O avec la convention suivante : c'est un cylindre avec une moitié rouge -celle qui est liée à O, et l'autre jaune -celle du côté de S.

```
/LiaisonR {
     7.5 0.5 15  [10 10] newcylindre
      dup (Red) outputcolors
   } def
/LiaisonY {
     0 0.5 7.5  [10 10] newcylindre
      dup (Yellow) outputcolors
   } def
/Liaison{LiaisonR LiaisonY solidfuz} def
```

L'atome d'oxygène, sa liaison, puis la mise en position de l'ensemble :

```
/Ox {2.17  [20 16] newsphere
     {0 0 15 translatepoint3d} solidtransform
     dup (Red) outputcolors} def
/LO { Liaison Ox solidfuz} def
/LO1 { LO  {0 -109.5 0 rotateOpoint3d} solidtransform } def
/LOx1 { LO1  {0 0 120 rotateOpoint3d} solidtransform } def
% fin liaison simple S-O
```

La liaison double double S=O, on se sert de la liaison simple définie précédemment et on la duplique en la décalant suivant l'axe Ox de 0,75 cm.

```
% Liaison double S=O
/LiaisonD1 {Liaison {-0.75 0 0 translatepoint3d} solidtransform} def
/LiaisonD2 {Liaison {0.75 0 0 translatepoint3d} solidtransform} def
/LiaisonDD { LiaisonD1 LiaisonD2 solidfuz} def
```

On lie cette liaison double avec l'atome d'O :

```
/LiaisonDOx {LiaisonDD Ox solidfuz} def
```

et par deux rotations successives on positionne les deux liaisons =O :

```
/LiaisonDOx1 {LiaisonDOx {0 -109.5 0 rotateOpoint3d} solidtransform } def
/LiaisonDOx2 {LiaisonDOx1 {0  0 -120 rotateOpoint3d} solidtransform } def
```

L'étape suivante consiste à fusionner ces deux liaisons :

```
/LO12 { LiaisonDOx1 LiaisonDOx2 solidfuz} def
/LO123 {LO12 LOx1 solidfuz} def
```

On passe ensuite à la liaison simple S-S :

```
% liaison simple S-S
/L4 { 0 0.5 20.10  [16 10] newcylindre
      dup (Yellow) outputcolors
    } def
```

Que l'on fusionne avec les deux atomes S-S :

```
/S1L4{ Soufre1 L4 solidfuz} def
/S1S2L4{ S1L4 Soufre2 solidfuz} def
```

La dernière étape consiste à fusionner S-S et les trois O déjà munis de leur liaisons :

```
/S2O3 { S1S2L4 LO123 solidfuz} def
S2O3  drawsolid**}
```

Chapitre 8

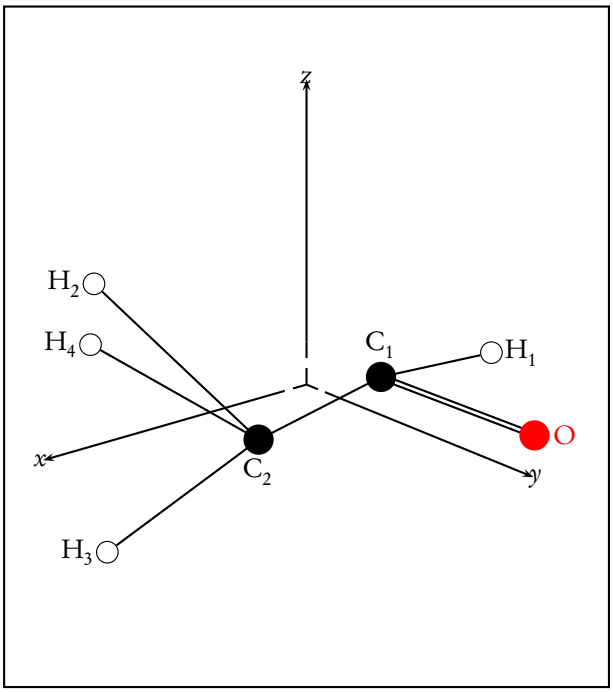
Interaction avec PSTricks

8.1 Positionner un point connu

`\psPoint(x,y,z){name}`

C'est une commande analogue à `\pnode(! x y){name}`. Elle place dans le nœud `(name)` le point de coordonnées (x,y,z), vu avec le point de vue choisi `viewpoint=vx vy vz`. On peut donc ensuite s'en servir pour marquer des points, tracer des lignes, des polygones etc.

Plaçons les positions des centres des atomes de la molécule d'éthanal CH_3COH.




```
\psPoint(-4.79,2.06,0){C1}
\psPoint(-4.79,15.76,0){Ox}
\psPoint(8.43,5.57,0){C2}
\psPoint(-14.14,3.34,0){H3}
\psPoint(14.14,-2.94,8.90){H6}
\psPoint(14.14,-2.94,-8.90){H7}
\psPoint(6.43,-16.29,0){H8}
\psline(C1)(H3)
\psline(C2)(H7)
\psline(C2)(H8)
\psline(C1)(C2)
\psline[doubleline=true](C1)(Ox)
\psline(C2)(H6)
\uput[r](H3){$\mathrm{H_1}$}
\uput[l](H6){$\mathrm{H_2}$}
\uput[l](H7){$\mathrm{H_3}$}
\uput[l](H8){$\mathrm{H_4}$}
\uput{0.25}[u](C1){$\mathrm{C_1}$}
\uput{0.25}[d](C2){$\mathrm{C_2}$}
\uput{0.25}[r](Ox){$\red\mathrm{O}$}
\psdots[dotstyle=o,dotsize=0.3]
       (H3)(H6)(H7)(H8)
\psdots[dotsize=0.4](C1)(C2)
\psdot[linecolor=red,dotsize=0.4](Ox)
```

8.2 Tracer une ligne brisée

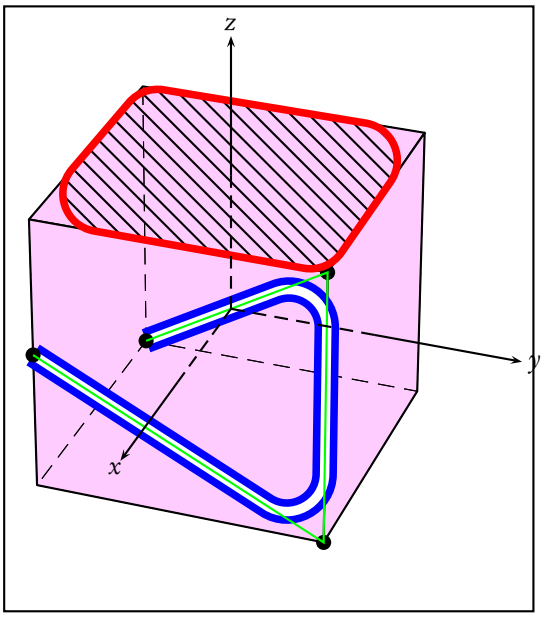
Cette commande est adaptée de la macro `\pstThreeDLine` du package `pst-3dplot` de H.Voss[1]

On l'utilise : `\psLineIIID[options](x0,y0,z0)(x1,y1,z1)...(xn,yn,zn)`, avec les options suivantes possibles :

[1] http://tug.ctan.org/tex-archive/graphics/pstricks/contrib/pst-3dplot.

- `linecolor=couleur`;
- `doubleline=true`;
- `linearc=valeur`.

On ne peut pas flécher les extrémités d'une ligne.




```
\begin{pspicture}(-3,-4)(4,4)
\psframe(-3,-4)(4,4)
\psSolid[object=cube,a=4,action=draw*,fillcolor=magenta!20]%
\psLineIIID[linecolor=blue,linewidth=0.1,linearc=0.5,doubleline=
  true](-2,-2,-2)(2,2,2)(2,2,-2)(2,-2,0)
\psPoint(2,-2,0){A}\psPoint(-2,-2,-2){B}
\psPoint(2,2,2){C}\psPoint(2,2,-2){D}
\psdot[dotsize=0.2](A)\psdot[dotsize=0.2](B)
\psdot[dotsize=0.2](C)\psdot[dotsize=0.2](D)
\psLineIIID[linecolor=green](-2,-2,-2)(2,2,2)(2,2,-2)(2,-2,0)
\psPolygonIIID[linecolor=red,fillstyle=vlines,linearc=0.5,
  linewidth=0.1](-2,-2,2)(-2,2,2)(2,2,2)(2,-2,2)
\axesIIID(2,2,2)(4,4,4)
\end{pspicture}
```

8.3 Tracer un polygone

On utilise : `\psPolygonIIID[options](x0,y0,z0)(x1,y1,z1)...(xn,yn,zn)`, avec les options suivantes possibles :

- `linecolor=couleur`;
- `doubleline=true`;
- `linearc=valeur`;
- `fillstyle=solid`;
- `fillstyle=vlines` ou `fillstyle=hlines` ou `fillstyle=crosshatch`.

8.4 Transformer un point et le mémoriser

Soit un point initial $A(x,y,z)$. On fait subir à ce point des rotations autour des axes Ox, Oy et Oz d'angles respectifs : `[RotX=valeurX,RotX=valeurY,RotX=valeurZ]`, dans cet ordre, puis on opère une translation de vecteur (v_x,v_y,v_z). Le problème a été de récupérer les coordonnées du point final $A'(x',y',z')$.

Le code `\psTransformPoint[RotX=valeurX,RotX=valeurY, RotX=valeurZ](x y z)(vx,vy,vz){A'}` permet de stocker dans le nœud A', les coordonnées du point transformé.

Dans l'exemple suivant $A(2,2,2)$ est l'un des sommets du cube initial, dont le centre est placé à l'origine du repère.

```
\psSolid[object=cube,a=4,action=draw*,linecolor=red]%
```

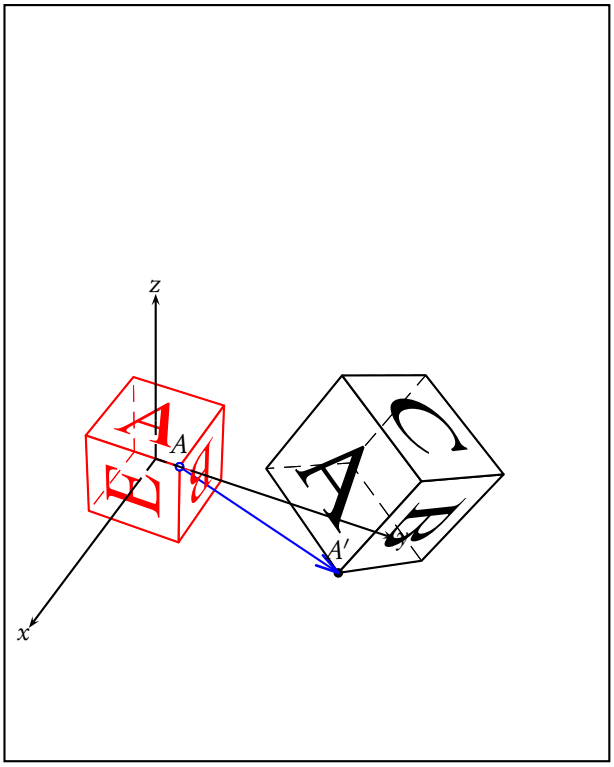
Ce cube subit différentes transformations :

```
\psSolid[object=cube,a=4,action=draw*,RotX=-30,RotY=60,RotZ=-60](7.5,11.25,10)%
```

Pour obtenir l'image de A, on applique la commande suivante :

```
\psTransformPoint[RotX=-30,RotY=60,RotZ=-60](2 2 2)(7.5,11.25,10){A'}
```

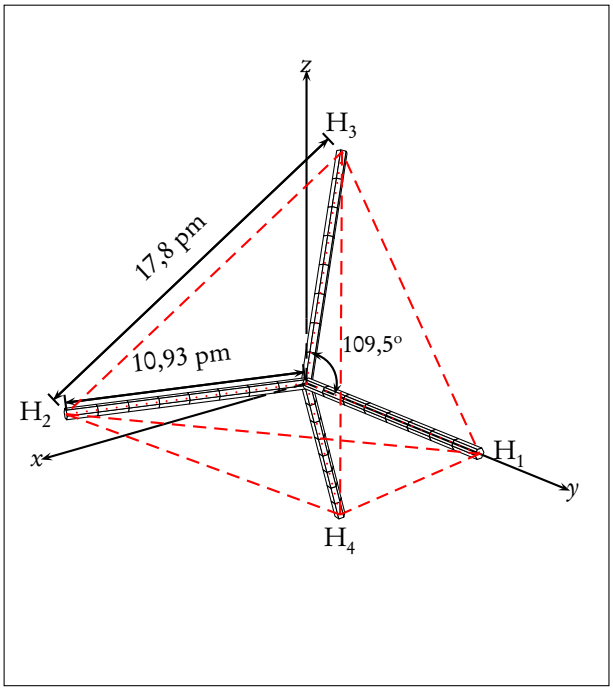
Ce qui permet, par exemple, de nommer ces points et de dessiner le vecteur $\overrightarrow{AA'}$.

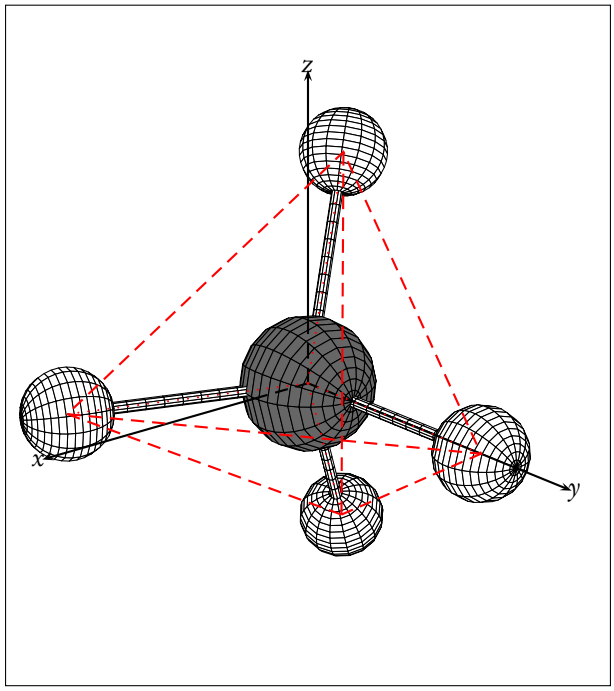


8.5 Annoter un schéma

Il est évidemment intéressant de pouvoir annoter un schéma, prenons l'exemple de la molécule de méthane dont nous voulons indiquer les distances et les angles.

Une première étape consiste à représenter la molécule avec uniquement les liaisons et les grandeurs caractéristiques, puis la molécule dans une représentation plus esthétique.





La construction de la molécule est détaillée dans le document `molecules.tex`. Pour annoter le schéma il suffit de repérer les sommets du tétraèdre :

```
\psPoint(0,10.93,0){H1}
\psPoint(10.3,-3.64,0){H2}
\psPoint(-5.15,-3.64,8.924){H3}
\psPoint(-5.15,-3.64,-8.924){H4}
```

et d'utiliser toute la puissance du package `pst-node`. D'abord pour les distances :

```
\pcline[offset=0.25]{<->}(H2)(H3)
\aput{:U}{17,8 pm}
\pcline[offset=0.15]{<->}(H2)(O)
\aput{:U}{10,93 pm}
\psPoint(-5.15,-3.64,-8.924){H4}
```

Puis, pour les angles, en s'aidant du package `pst-eucl`

```
\pstMarkAngle[arrows=<->]{H1}{O}{H3}{\small 109,5$^{\mathrm{o}}$}
```

Chapitre 9

Projections

9.1 Présentation

Le package permet la manipulation et la représentation de quelques objets simples à deux dimensions. Ces opérations se font toutes par le biais de la macro `\psProjection` qui est destinée à la manipulation d'objets 2d au regard d'un plan donné.

Sa syntaxe est analogue à celle de `\psSolid`, avec en particulier la présence obligatoire de l'argument `[object=]` permettant de spécifier le type d'objet manipulé.

Sa syntaxe générale est `\psSolid[object=`*objectname*`, plan=`*plantype*`<,`*options*`>](`*x*`,`*y*`)`

9.2 Le paramètre `visibility`

Pour toutes les projections, le booléen `[visibility]` (valeur `true` par défaut) permet de spécifier si l'on souhaite tenir compte ou non de la visibilité du plan de projection.

Positionné à `false`, la projection est toujours effectuée. Positionné à `true`, la projection n'a lieu que si le plan de projection est visible du point de vue de l'observateur.

9.3 Définition du plan de projection

La définition du plan de projection se fait avec l'argument `[plan=`*plantype*`]` qui attend un argument de *type plan*. La création d'un tel argument passe obligatoirement par la commande `\psSolid[object=plan]` (voir le paragraphe dédié au chapitre 4 et l'exemple d'utilisation ci dessous au sous-paragraphe *Labels* du paragraphe *Points*).

9.4 Points

9.4.1 Définition directe

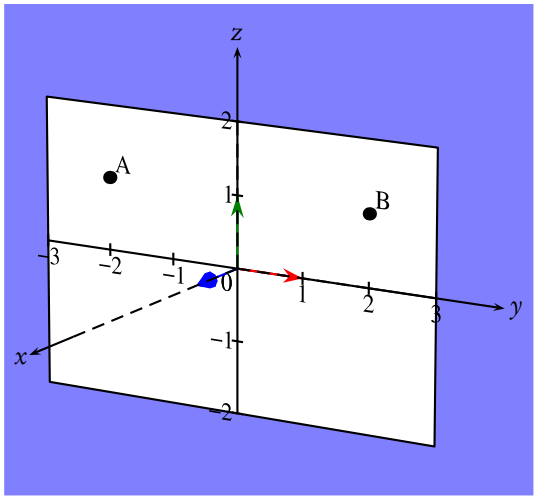
L'objet `point` permet de définir un point. Sous sa forme la plus simple, on peut utiliser les valeurs (x,y) de ses coordonnées directement dans la commande `\psProjection` ou par le biais de l'argument `args`.

Ainsi les 2 commandes `\psProjection[object=point](1,2)` et `\psProjection[object=point,arg=1 2]` sont équivalentes et aboutissent au tracé du point de coordonnées $(1,2)$ sur le plan considéré.

9.4.2 Labels

L'option `[text=`*str*`]` permet de spécifier une chaîne de caractère à projeter sur le plan de référence au voisinage du point considéré. La position d'affichage par rapport au point se fait avec l'argument `[pos=`*value*`]` où *value* est un élément de {ul, cl, bl, dl, ub, cb, bb, db, uc, cc, bc, dc, ur, cr, br, dr}.

L'utilisation du paramètre pos est détaillée dans un paragraphe ultérieur.




```
\psset{solidmemory}
%% definition et dessin du plan de projection
\psSolid[object=plan,
   definition=equation,
   args={[1 0 0 0] 90},
   name=monplan,
   planmarks,
   showBase,]
%% affectation du plan de projection
\psset{plan=monplan}
\psProjection[object=point,args=-2 1,
   text=A,pos=ur,]
\psProjection[object=point,text=B,pos=ur,
](2,1)
\composeSolid
```

9.4.3 Nommage et sauvegarde d'un point

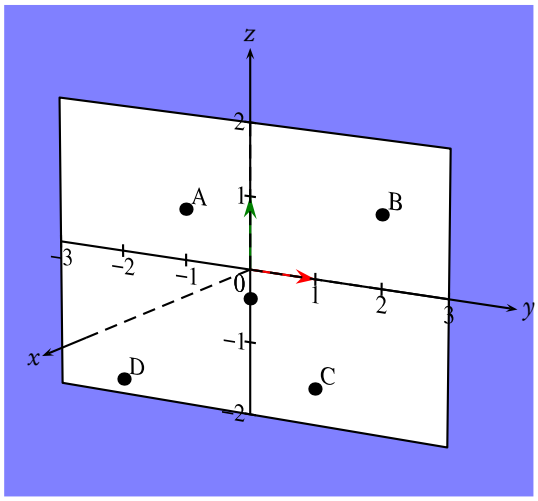
Si l'option `[name=str]` est présente, les coordonnées (x,y) du point considéré seront sauvegardées sous le nom désigné par str et pourront être réutilisées.

9.4.4 Autres définitions

Il existe d'autres méthodes pour définir un point 2d. L'argument `definition`, couplé à l'argument `args` permet d'utiliser les différentes méthodes supportées :

- `[definition=milieu]` ; `args`=$A\ B$. Le milieu du segment $[AB]$
- `[definition=parallelopoint]` ; `args`=$A\ B\ C$. Le point D tel que $(ABCD)$ soit un parallélogramme.
- `[definition=translatepoint]` ; `args`=$M\ u$. L'image du point M par la translation de vecteur $\vec{u}$
- `[definition=rotatepoint]` ; `args`=$M\ I\ r$. Le point image de M par la rotation de centre I et d'angle r (en degrés)
- `[definition=hompoint]` ; `args`=$M\ A\ k$. Le point M' vérifiant $\overrightarrow{AM'} = k\overrightarrow{AM}$
- `[definition=orthoproj]` ; `args`=$M\ d$. Le projeté orthogonal du point M sur la droite d.
- `[definition=projx]` ; `args`=M. Le projeté du point M sur l'axe Ox .
- `[definition=projy]` ; `args`=M. Le projeté du point M sur l'axe Oy .
- `[definition=sympoint]` ; `args`=$M\ I$. Le symétrique du point M par rapport au point I.
- `[definition=axesympoint]` ; `args`=$M\ d$. Le symétrique du point M par rapport à la droite d.
- `[definition=cpoint]` ; `args`=$\alpha\ C$. Le point correspondant à l'angle α du cercle C
- `[definition=xdpoint]` ; `args`=$x\ d$. Le point d"abscisse x de la droite d.
- `[definition=ydpoint]` ; `args`=$y\ d$. Le point d'ordonnée y de la droite d.
- `[definition=interdroite]` ; `args`= $d_1\ d_2$. Le point d'intersection des droites d_1 et d_2.
- `[definition=interdroitecercle]` ; `args`= $d\ I\ r$. Les points d'intersection de la droite d avec le cercle de centre I de rayon r.

Dans l'exemple ci-dessous, on définit et on nomme 3 points A, B et C, puis on calcule le point D tel que $(ABCD)$ parallélogramme ainsi que le centre de ce parallélogramme.



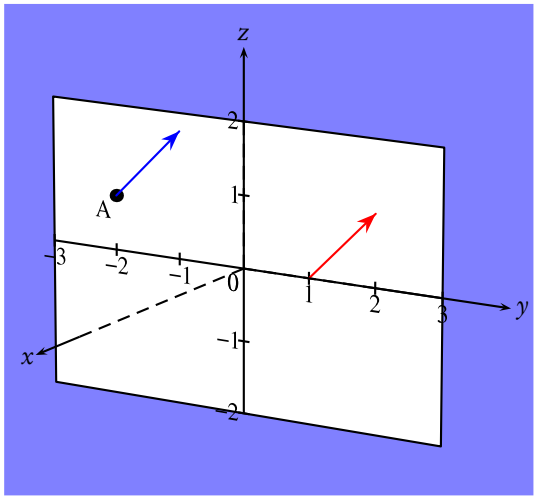

```
\psProjection[object=point,
   text=A,pos=ur,name=A,](-1,.7)
\psProjection[object=point,
   text=B,pos=ur,name=B,](2,1)
\psProjection[object=point,
   text=C,pos=ur,name=C,](1,-1.5)
\psProjection[object=point,
   definition=parallelopoint,
   args=A B C,
   text=D,pos=ur,name=D,]
\psProjection[object=point,
   definition=milieu,
   args=D B,]
```

9.5 Vecteurs

9.5.1 Définition directe

L'objet `vecteur` permet de définir et tracer un vecteur. Sous sa forme la plus simple, on utilise l'argument l'argument `args` pour définir ses coordonnées (X,Y) et on spécifie le point d'origine où tracer le vecteur en utilisant les valeurs (x,y) dans la commande `\psProjection` (ou éventuellement un point nommé).

Comme pour les points, on peut sauvegarder les coordonnées d'un vecteur en utilisant l'option `name`.




```
\psProjection[object=point,
   args=-2 0.75,name=A,
   text=A,pos=dl,]
\psProjection[object=vecteur,
   linecolor=red,
   args=1 1,name=U,
](1,0)
\psProjection[object=vecteur,
   args=U,
   linecolor=blue,
](A)
```

9.5.2 Autres définitions

Il existe d'autres méthodes pour définir un vecteur 2d. L'argument `definition`, couplé à l'argument `args` permet d'utiliser les différentes méthodes supportées :

- `[definition=vecteur]` ; `args=` $A\ B$. Le vecteur $\overrightarrow{AB}$
- `[definition=orthovecteur]` ; `args=` u. Un vecteur orthogonal à $\vec{u}$ et de même norme.
- `[definition=normalize]` ; `args=` u. Le vecteur $\|\vec{u}\|^{-1}\vec{u}$ si $\vec{u}\neq\vec{0}$, et $\vec{0}$ sinon.
- `[definition=addv]` ; `args=` $u\ v$. Le vecteur $\vec{u}+\vec{v}$
- `[definition=subv]` ; `args=` $u\ v$. Le vecteur $\vec{u}-\vec{v}$
- `[definition=mulv]` ; `args=` $u\ \alpha$. Le vecteur $\alpha\vec{u}$

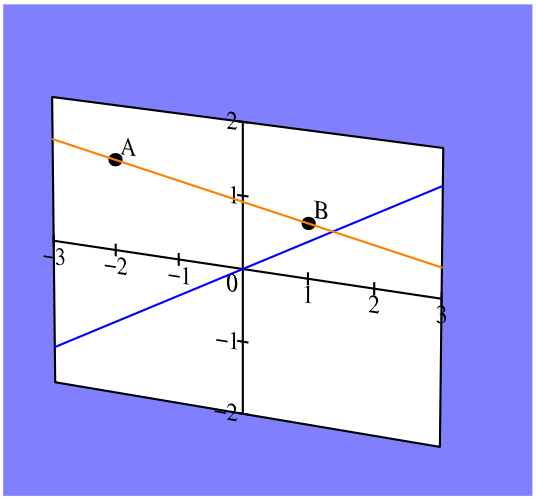
9.6 Droites

9.6.1 Définition directe

L'objet `droite` permet de définir et tracer une droite. Dans `pst-solides3d`, une droite en 2d est définie par la donnée de 2 de ses points.

Sous la forme la plus simple, on utilise l'argument l'argument `args` pour spécifier 2 points de la droite considérée. On peut utiliser les coordonnées ou des points nommés.

Comme pour les points et les vecteurs, on peut sauvegarder la donnée d'une droite en utilisant l'option `name`.




```
\psProjection[object=point,
   name=A,text=A,pos=ur,](-2,1.25)
\psProjection[object=point,
   name=B,text=B,pos=ur,](1,.75)
\psProjection[object=droite,
   linecolor=blue,args=0 0 1 .5,]
\psProjection[object=droite,
   linecolor=orange,args=A B,]
```

9.6.2 Autres définitions

Il existe d'autres méthodes pour définir une droite 2d. L'argument `definition`, couplé à l'argument `args` permet d'utiliser les différentes méthodes supportées :

- `[definition=horizontale]` ; `args=` b. La droite d'équation $y = b$.
- `[definition=verticale]` ; `args=` a. La droite d'équation $x = a$.
- `[definition=paral]` ; `args=` $d\ A$. La droite parallèle à la droite d passant par le point A.
- `[definition=perp]` ; `args=` $d\ A$. La droite perpendiculaire à la droite d passant par le point A.
- `[definition=mediatrice]` ; `args=` $A\ B$. La droite médiatrice du segment $[AB]$.
- `[definition=bissectrice]` ; `args=` $A\ B\ C$. La droite bissectrice de l'angle $\widehat{ABC}$.
- `[definition=axesymdroite]` ; `args=`$d\ D$. Symétrique de la droite d par rapport à la droite D.
- `[definition=rotatedroite]` ; `args=`$d\ I\ r$. Image de la droite d par la rotation de centre I et d'angle r (en degrés)
- `[definition=translatedroite]` ; `args=`$d\ u$. Image de la droite d par la translation de vecteur $\vec{u}$.

9.7 Cercles

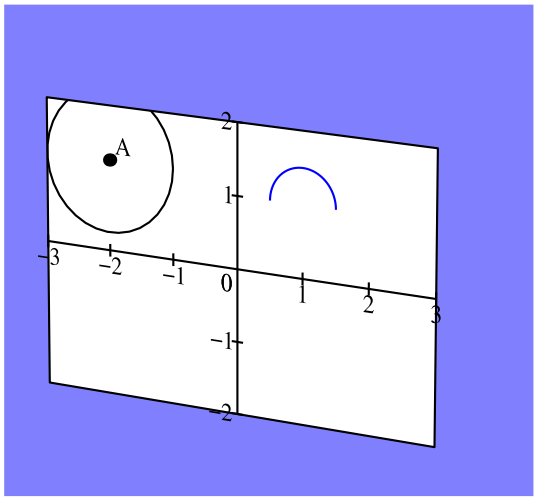
9.7.1 Définition directe

L'objet `cercle` permet de définir et tracer un cercle. Dans `pst-solides3d`, un cercle en 2d est définie par la donnée de son centre et de son rayon.

Sous la forme la plus simple, on utilise l'argument l'argument `args` pour spécifier le centre et le rayon de la droite considérée. On peut utiliser les coordonnées ou des variables nommées.

L'argument `[range=`t_{min} t_{max}`]` permet de spécifier l'intervalle de tracé du cercle considéré.

Comme pour les autres objets, on peut sauvegarder la donnée d'un cercle en utilisant l'option `name`.




```
\psset{solidmemory}
...
\psProjection[object=point,
   name=A,text=A,pos=ur,
](-2,1.25)
\psProjection[object=cercle,
   args=A 1,range=0 360,]
\psProjection[object=cercle,
   args=1 1 .5,linecolor=blue,
   range=0 180,]
\composeSolid
```

9.7.2 Autres définitions

Il existe d'autres méthodes pour définir un cercle 2d. L'argument `definition`, couplé à l'argument `args` permet d'utiliser les différentes méthodes supportées :

- `[definition=ABcercle]` ; `args=`A B C. Le cercle passant par les points non alignés A, B et C.
- `[definition=diamcercle]` ; `args=`A B. Le cercle de diamètre $[AB]$.

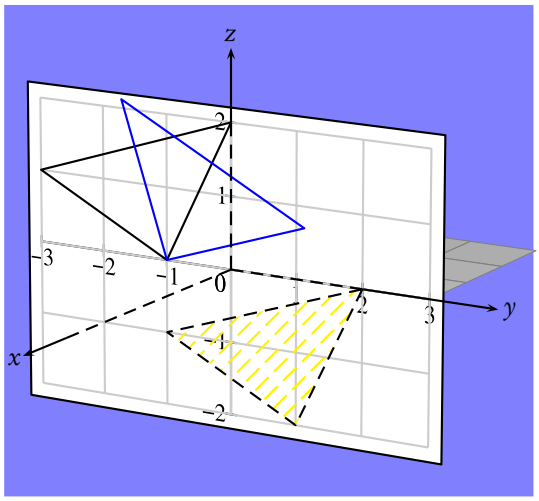
9.8 Polygones

L'objet `polygone` permet de définir un polygone. Sous sa forme usuelle, on utilise l'argument `args` pour spécifier la liste des points : `[object=polygone,args=`A_0 A_1 $\ldots A_n$`]`

Il existe d'autres méthodes pour définir un polygone 2d. L'argument `definition`, couplé à l'argument `args` permet d'utiliser les différentes méthodes supportées :

- `[definition=translatepol]` ; `args=`pol u. Translaté du polygone pol par le vecteur $\vec{u}$
- `[definition=rotatepol]` ; `args=`pol I α. Image du polygone pol par la rotation de centre I et d'angle α
- `[definition=hompol]` ; `args=`pol I α. Image du polygone pol par l'homothétie de centre I et de rapport α.
- `[definition=sympol]` ; `args=`pol I. Image du polygone pol par la symétrie de centre I.
- `[definition=axesympol]` ; `args=`pol d. Image du polygone pol par la symétrie axiale de droite d.

Dans l'exemple ci-dessous, on définit, on nomme et on trace le polygone de sommets $(1,0)$, $(-3,1)$, $(0,2)$ puis on trace en bleu son image par la rotation de centre $(-1,0)$ et d'angle -45. Enfin, on représente le translaté du polygone d'origine par le vecteur $(2,-2)$, et ce en incorporant directement du code jps dans l'argument `[definition=]`.

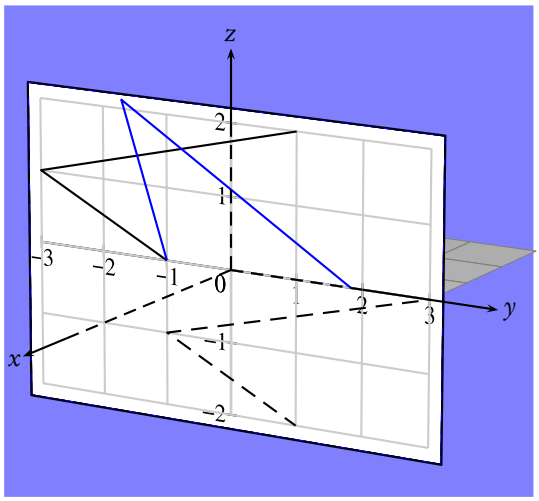



```
\psProjection[object=polygone,
   args=-1 0 -3 1 0 2,
   name=P,]
\psProjection[object=polygone,
   definition=rotatepol,
   linecolor=blue,
   args=P -1 0 -45,]
%% du code jps dans la definition
\psProjection[object=polygone,
   definition={2 -2 addv} papply,
   fillstyle=hlines,hatchcolor=yellow,
   linestyle=dashed,
   args=P,]
```

9.9 Lignes

L'objet `line` permet de définir une ligne brisée. Sous sa forme usuelle, on utilise l'argument `args` pour spécifier la liste des points : `[object=line,args=`$A_0\ A_1\ \ldots A_n$`]`

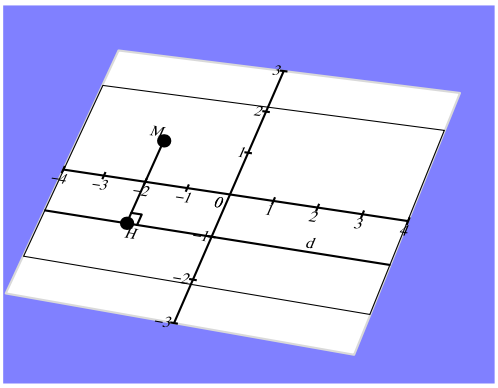
On peut également définir les transformées d'une ligne brisée par une translation, une rotation, une homothétie, etc...en reprenant les opérations disponibles sur les polygones.




```
\psProjection[object=line,
   args=-1 0 -3 1 0 2,
   name=P,]
\psProjection[object=line,
   definition=rotatepol,
   linecolor=blue,
   args=P -1 0 -45,]
%% du code jps dans la definition
\psProjection[object=line,
   definition={2 -2 addv} papply,
   linestyle=dashed,
   args=P,]
```

9.10 Angle droit

L'objet `rightangle` permet de définir un angle droit. Sa syntaxe est `[object=rightangle,args=A B C]`




```
\psProjection[object=droite,
   definition=horizontale,args=-1,name=d,]
\psProjection[object=point,args=-2 1,
   name=M,text=M,pos=ul,]
\psProjection[object=point,
   definition=orthoproj,
   args=M d,name=H,text=H,pos=dr,]
%% definition du point H' pour orienter l'angle droit
\psProjection[object=point,
   definition=xdpoint,args=2 d,
   name=H',action=none,text=d,pos=ur,]
\psProjection[object=line,args=M H,]
\psProjection[object=rightangle,args=M H H',]
```

9.11 Courbes de fonction numériques et courbes paramétrées

9.11.1 Courbe de fonction numérique

L'objet `courbe` permet d'obtenir le tracé de la courbe d'une fonction numérique dont le nom est passée *via* l'argument `function`. Cette fonction à valeurs dans **R** ayant été préalablement définie avec la macro `\defFunction` vue plus avant dans ce guide.

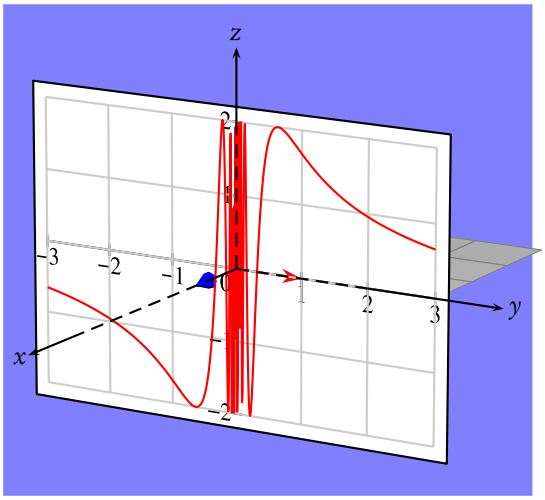
On pourra donc définir cette fonction, soit en notation algébrique avec l'option `[algebraic]`, soit en notation polonaise inversée (langage postscript), avec une variable quelconque ($x, u, t \ldots$), par une expression de la forme suivant le cas :

```
\defFunction[algebraic]{nom_fonction}(x){x*sin(x)}{}{}
```

```
\defFunction{nom_fonction}(x){x dup sin mul}{}{}
```

Cette expression dans doit être incluse dans l'environnement `pspicture`.

Les limites de la variable sont définies dans l'option `range=`*xmin xmax*, et l'option `resolution=`*n* permet de préciser le nombre de points calculés pour le dessin de la courbe.




```
\defFunction[algebraic]{1_sin}(x)
   {2*sin(1/x)}{}{}
\psset{plan=monplan}
...
\psProjection[object=courbe,linecolor=red,
   range=-3 3,resolution=720,function=1_sin]
```

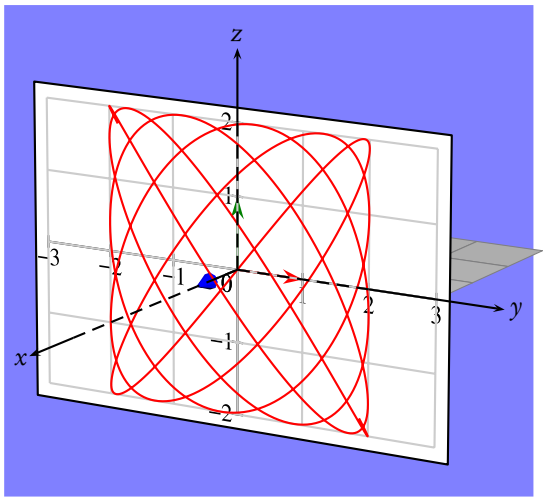
9.11.2 Courbes paramétrées

La technique est analogue, à la différence près que la fonction évoquée est à valeurs dans R^2, et que l'objet passé en paramètre à `\psProjection` est `courbeR2`.

Pour dessiner un cercle de rayon 3, on écrira :

```
\defFunction[algebraic]{cercle}(t){3*cos(t)}{3*sin(t)}{}
```

Autre exemple : les courbes de Lissajous.




```
\defFunction[algebraic]{F}(t)
   {2*sin(0.57735*t)}
   {2*sin(0.707*t)}
   {}
\psset{plan=monplan}
...
\psProjection[object=courbeR2,
   range=-25.12 25.12,resolution=720,
   normal=1 1 2,linecolor=red,
   function=F,
]
```

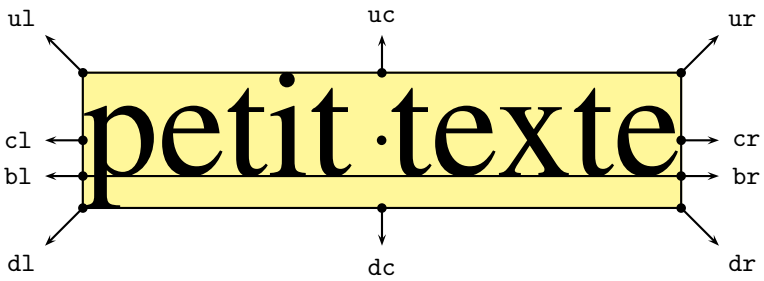
9.12 Texte

L'objet `texte` de la macro `\psProjection` permet de projeter des chaînes de caractères sur des plans quelconques.

9.12.1 Les paramètres et les options

Il y a 3 paramètres : `[text]` qui définit la chaîne à afficher, `[fontsize]`, qui donne la taille de la fonte en points (rappelons qu'une taille de 28,45 pts correspond à 1 cm), et enfin `[pos]`, qui définit la position du texte par rapport au point visé. Par défaut le texte est centré sur l'origine du plan.

Ce dernier paramètre nécessite un peu d'explications. Considérons la chaîne de caractères `petit texte` représentée ci-dessous.



Nous avons 4 lignes horizontales de références : la ligne inférieure `(d)own`, la ligne de base `(b)aseline`, la ligne médiane, ou ligne centrale `(c)enter`, et la ligne supérieure `(u)p`.

Il y a également 4 lignes de référence sur l'axe vertical : la ligne de gauche `(l)eft`, la ligne de base `(b)aseline`, la ligne centrale `(c)enter` et la ligne de droite `(r)ight`. Dans le cas d'une chaîne de caractère, les 2 lignes verticales `l` et `b` sont confondues.

L'intersection de ces 4 lignes horizontales avec ces 4 lignes verticales nous donne 16 points appelés `dl`, `bl`, `cl`, `ul`, `db`, `bb`, `cb`, `ub`, `dc`, `bc`, `cc`, `uc`, `dr`, `br`, `cr`, `ur`.

Parmi ceux-ci, 4 sont considérés comme *points intérieurs* : `bb`, `bc`, `cb` et `cc`.

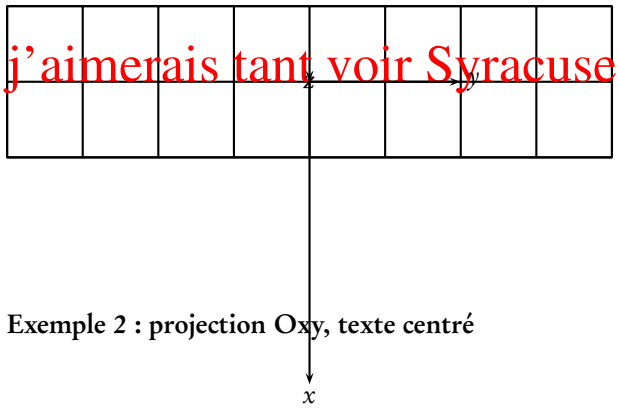
Lorsque le paramètre `[pos]` de `\psProjection` est affecté à l'un de ces 4 points, cela signifie que ce dernier doit être positionné au point d'origine du plan de projection.

Lorsque le paramètre `[pos]` de `\psProjection` est affecté à l'un des 12 points restant, cela indique une direction dans laquelle doit être positionné le texte par rapport au point d'origine du plan de projection.

Par exemple, `\psProjection[...,pos=uc](0,0)` indique que le texte doit être centré par rapport au point (0,0) et affiché au-dessus.

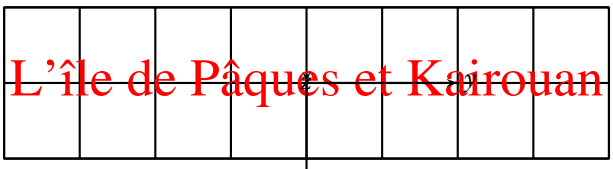
9.12.2 Exemples de projetés sur un plan quelconque

Exemple 1 : projection sur Oxy, avec l'option `pos=bc`



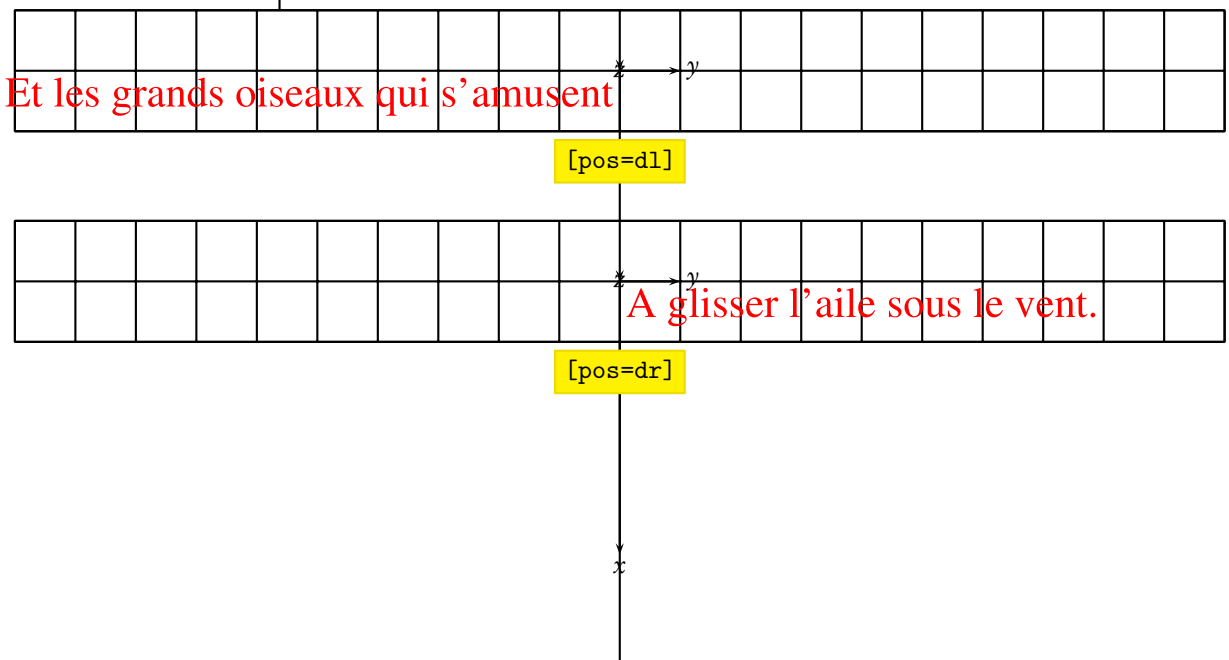

```
\begin{pspicture}(-4,-1.5)(4,1.5)
\psset{solidmemory}
\psset{lightsrc=10 0 10,
   viewpoint=50 -90 90 rtp2xyz,Decran=50}
\psSolid[object=plan,definition=normalpoint,plangrid,
   base=-4 4 -1 1,args={0 0 0 [0 0 1 90]},name=monplan
      ,]
\psProjection[object=texte,
   fontsize=20,linecolor=red,
   pos=bc,plan=monplan,
   text=j'aimerais tant voir Syracuse,
](0,0)%
\axesIIID(0,0,0)(4,2,1)
\composeSolid
\end{pspicture}
```

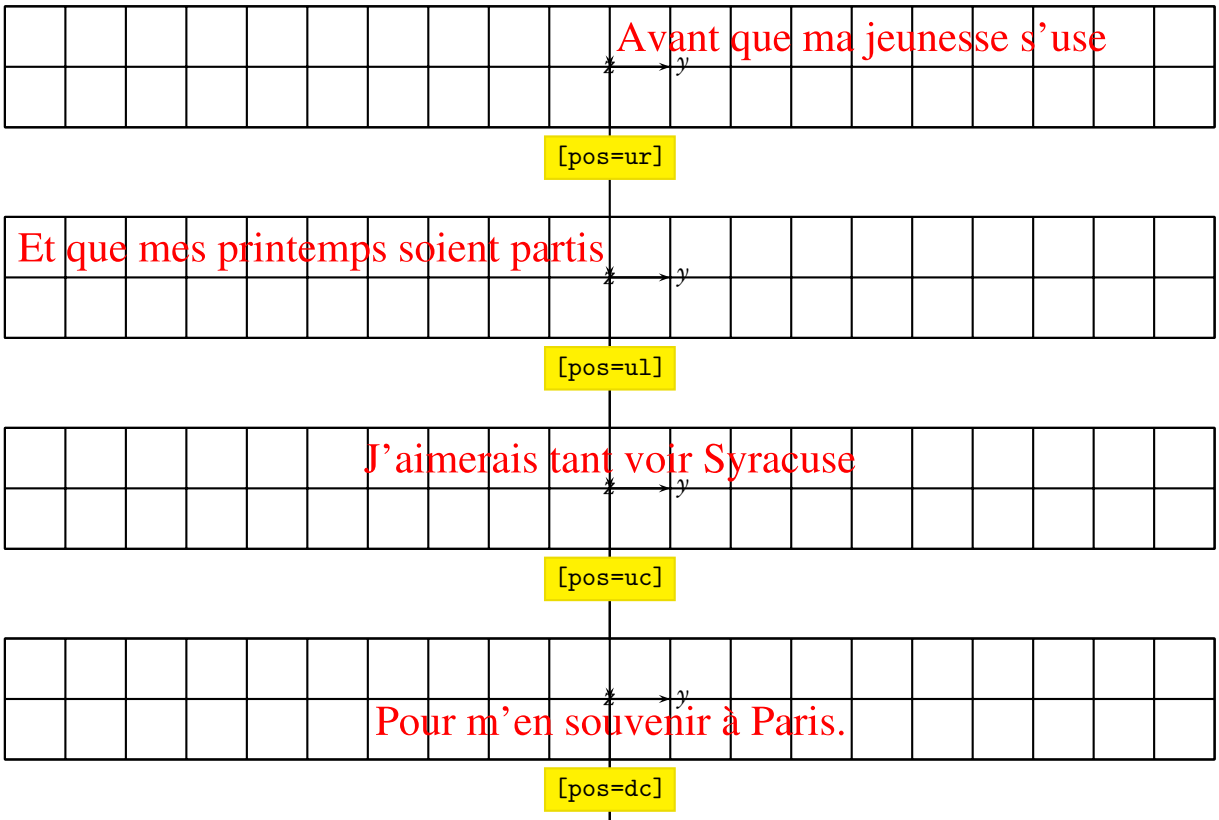
Exemple 2 : projection Oxy, texte centré



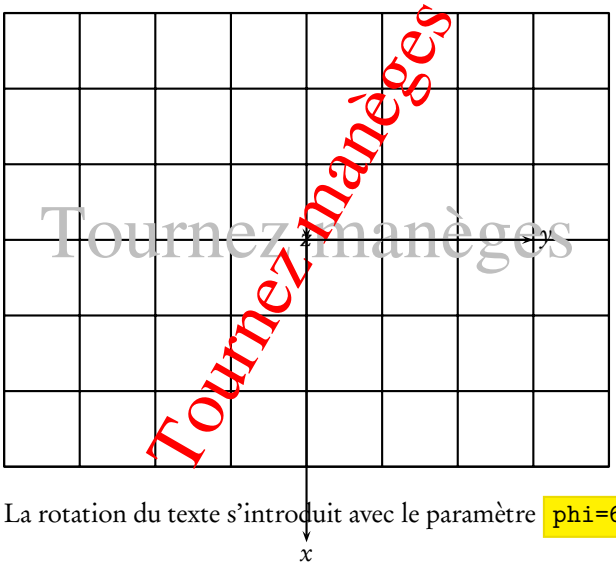

```
\begin{pspicture}(-4,-1.5)(4,1.5)
\psset{solidmemory}
\psset{lightsrc=10 0 10,
   viewpoint=50 -90 90 rtp2xyz,Decran=50}
\psSolid[object=plan,definition=normalpoint,plangrid,
   base=-4 4 -1 1,args={0 0 0 [0 0 1 90]},name=monplan
      ,]
\psProjection[object=texte,
   fontsize=20,linecolor=red,
   text= L'île de Pâques et Kairouan,
   plan=monplan]%
\axesIIID(0,0,0)(4,2,1)
\end{pspicture}
```

Exemple 3 : projection Oxy, avec différentes options `pos=dl, etc.`





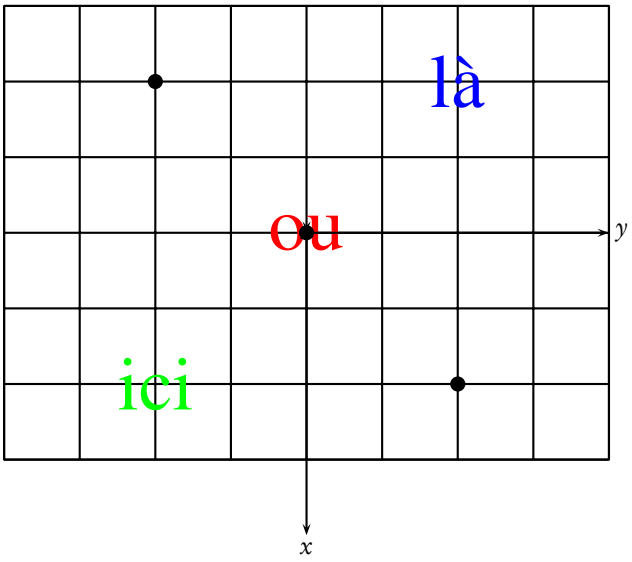
Exemple 4 : projection Oxy avec rotation du texte




```
\begin{pspicture}(-4,-3)(4,3)
\psset{solidmemory}
\psset{lightsrc=10 0 10,
   viewpoint=50 -90 90 rtp2xyz,Decran=50}
\psSolid[object=plan,definition=normalpoint,plangrid,
   base=-4 4 -3 3,args={0 0 0 [0 0 1 90]},name=monplan
      ,]
\psset{plan=monplan}
\psProjection[object=texte,
   fontsize=28.45,linecolor=gray!50,
   text=Tournez manèges]%
\psProjection[object=texte,
   fontsize=28.45,linecolor=red,
   text=Tournez manèges,
   phi=60]%
\axesIIID(0,0,0)(4,3,1)
\end{pspicture}
```

La rotation du texte s'introduit avec le paramètre `phi=60`.

Exemple 5 : positionnement du texte en un point quelconque

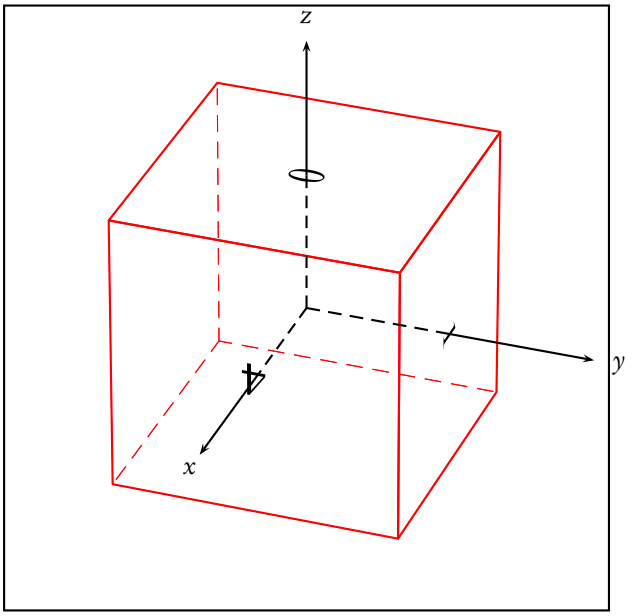



```
\begin{pspicture}(-4,-3)(4,3)
\psset{solidmemory}
\psset{viewpoint=50 -90 90 rtp2xyz,
   Decran=50}
\psSolid[object=plan,definition=normalpoint,plangrid,
   base=-4 4 -3 3,args={0 0 0 [0 0 1 90]},name=monplan
      ,]
\psset{fontsize=28.45,plan=monplan}
\psProjection[object=texte,
   linecolor=green,
   text=ici](-2,-2)
\psProjection[object=texte,
   linecolor=red,
   text=ou]%
\psProjection[object=texte,
   linecolor=blue,
   text=là](2,2)
\psPoint(0,0,0){O}
\psPoint(-2,-2,0){O1}
\psPoint(2,2,0){O2}
\psdots[dotsize=0.2](O)(O1)(O2)
\axesIIID(0,0,0)(4,4,1)
\end{pspicture}
```

9.12.3 Exemples de projections sur une face d'un solide

Méthode

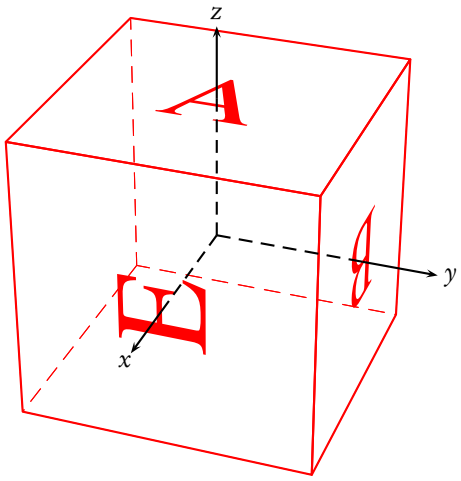
Le solide doit être mémorisé avec l'option générale `\psset{solidmemory}`. La première chose à faire est de repérer les numéros des faces du solide avec l'option `[numfaces=all]`.




```
\psset{viewpoint=50 20 30 rtp2xyz,Decran=100}
\begin{pspicture}(-4,-4)(4,4)
\psframe(-4,-4)(4,4)
\psSolid[object=cube,a=2,action=draw,
   linecolor=red,numfaces=all]%
\axesIIID(1,1,1)(2,2,2)
\end{pspicture}
```

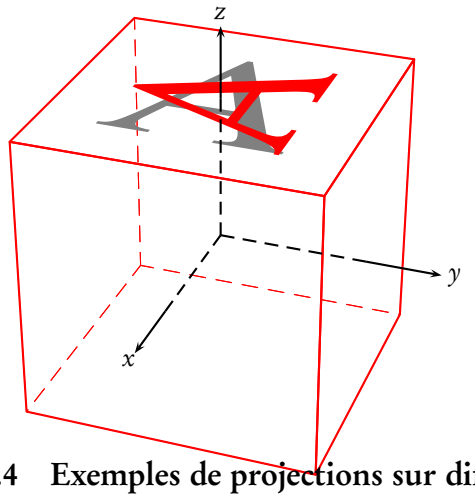
Puis on définit le plan de projection par la face choisie, ici on affiche `A` sur la face de numéro `0` :

```
\psSolid[object=plan,definition=solidface,args=A 0,name=P0]
\psProjection[object=texte,linecolor=red,text=A,plan=P0]%
```




```
\psset{viewpoint=50 20 30 rtp2xyz,Decran=50}
\begin{pspicture}(-3,-4)(4,5)
\psset{unit=0.5}
\psset{solidmemory}
\psSolid[object=cube,a=8,action=draw,name=A,linecolor=
  red]%
\psset{fontsize=100}
\psSolid[object=plan,action=none,
   definition=solidface,args=A 0,name=P0]
\psProjection[object=texte,linecolor=red,text=A,plan=P
  0]%
\psSolid[object=plan,action=none,
   definition=solidface,args=A 1,name=P1]
\psProjection[object=texte,linecolor=red,text=B,plan=P
  1]%
\psSolid[object=plan,action=none,
   definition=solidface,args=A 4,name=P4]
\psProjection[object=texte,linecolor=red,text=E,plan=P
  4]%
\axesIIID(4,4,4)(6,6,6)
\end{pspicture}
```

Rotation du texte avec l'option `phi`




```
\psset{viewpoint=50 20 30 rtp2xyz,Decran=50}
\psset{unit=0.5}
\begin{pspicture}(-3,-4)(4,5)
\psset{solidmemory}
\psSolid[object=cube,a=8,action=draw,linecolor=red,
  name=A]%
\psset{fontsize=200}
\psSolid[object=plan,action=none,
   definition=solidface,args=A 0,name=P0]
\psProjection[object=texte,linecolor=gray,text=A,plan=
  P0]%
\psset{phi=90}
\psProjection[object=texte,linecolor=red,text=A,plan=P
  0]%
\axesIIID(4,4,4)(6,6,6)
\end{pspicture}
```

9.12.4 Exemples de projections sur différentes faces d'un solide

Nous projetons un poème, vers après vers, sur 4 faces d'un cube. Il est nécessaire de passer l'option `solidmemory` au début des commandes. On définit ensuite le cube, que l'on mémorise à l'aide de la commande `name=A` :

```
\psset{solidmemory}
\psSolid[object=cube,a=8,name=A1](0,0,4.2)%
```

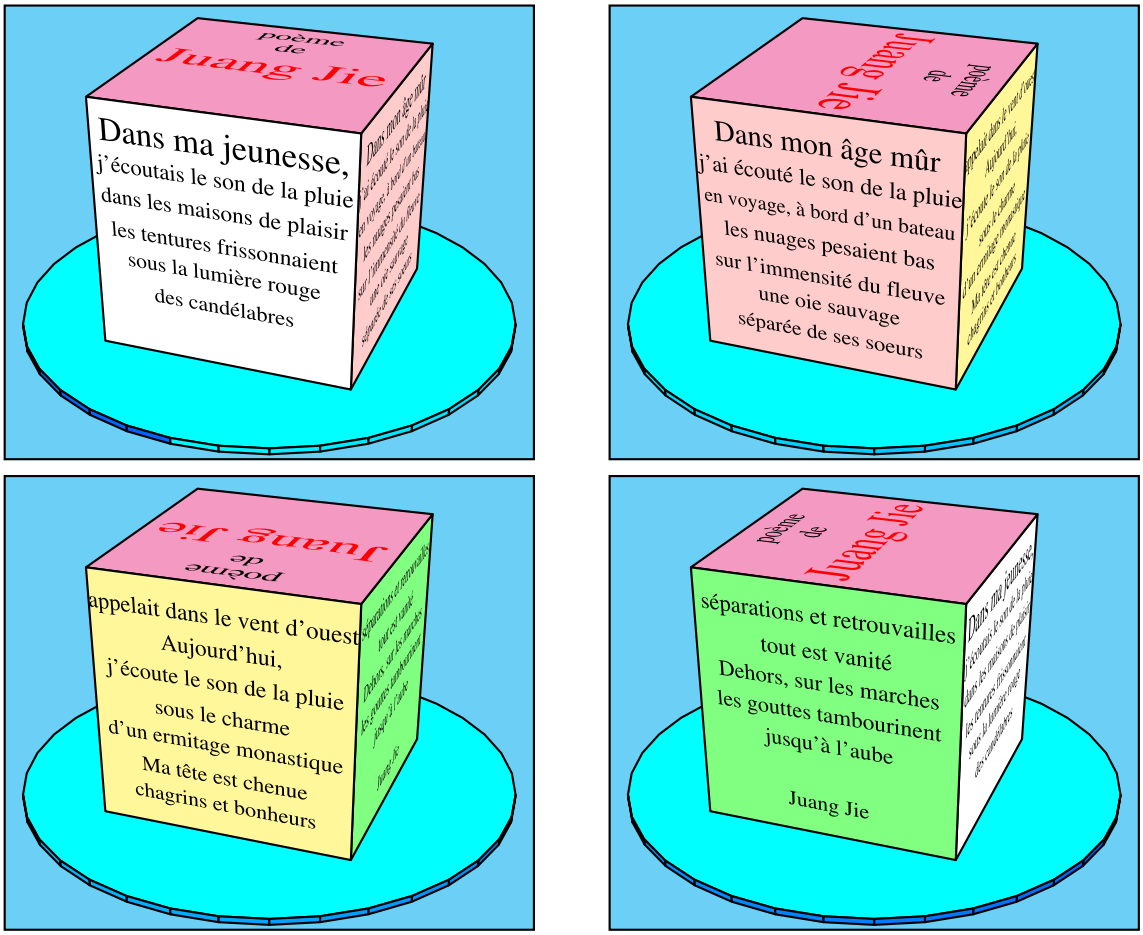
Le numéro de chaque face ayant été repéré dans un essai préalable avec l'option `numfaces=all`, les commandes suivantes :

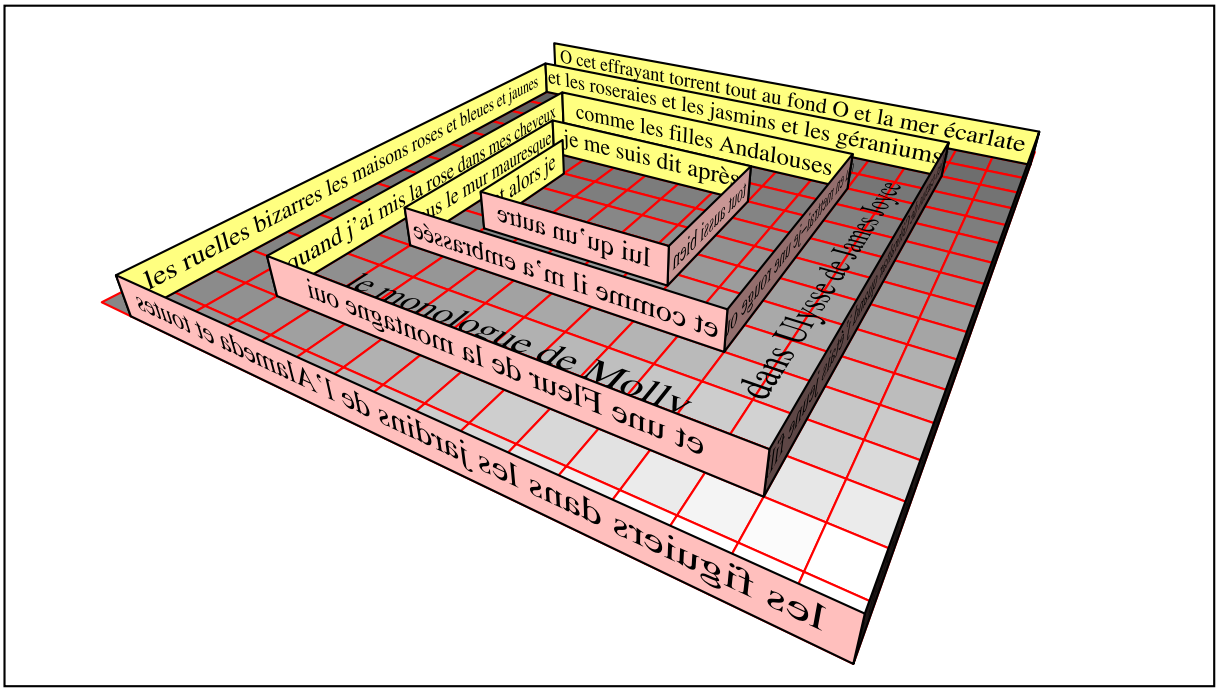
```
\psSolid[object=plan,action=none,definition=solidface,args=A 0,name=P0]%
\psProjection[object=texte,text=poème,fontsize=30,plan=P0](0,3)%
```

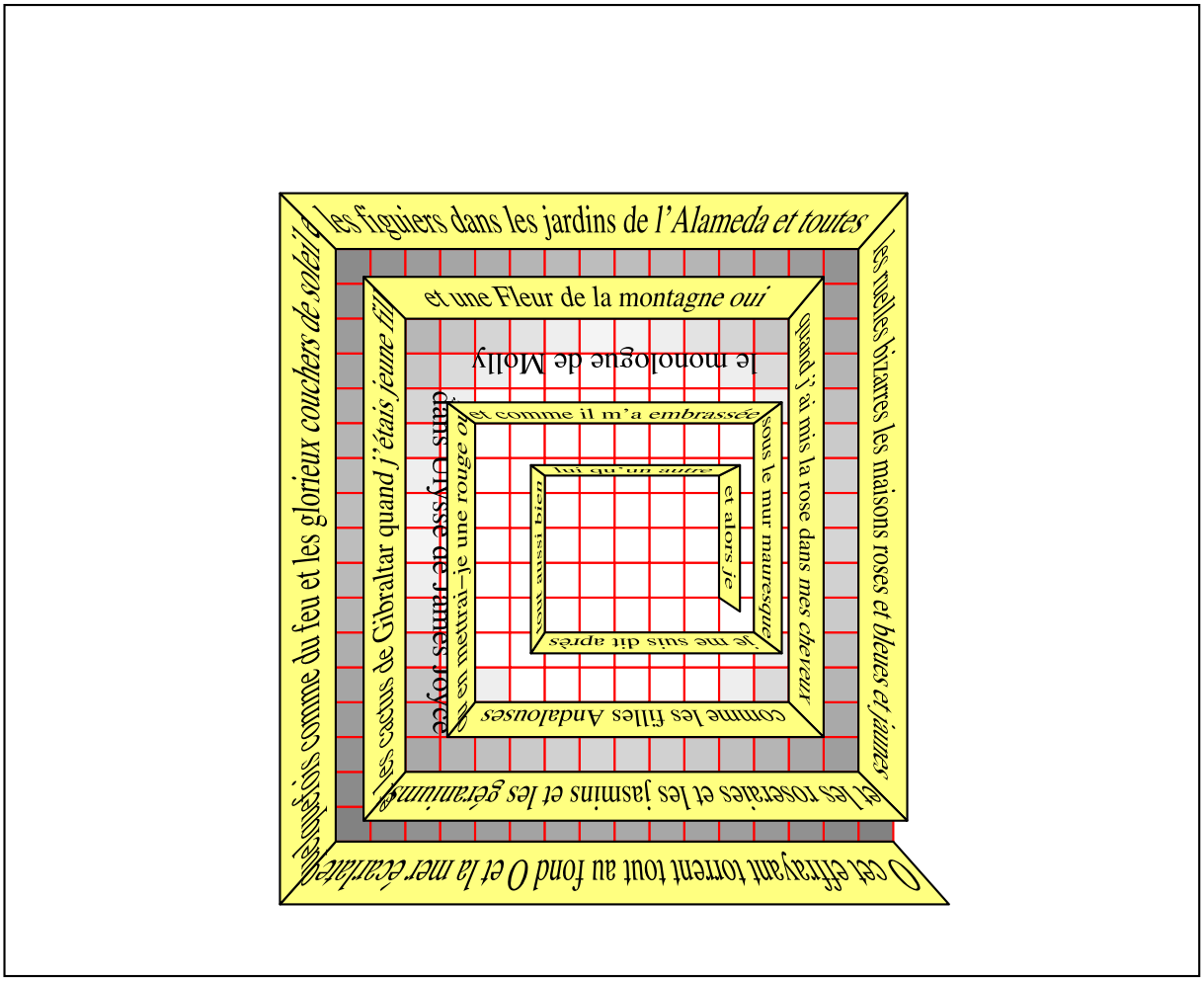
définissent tout d'abord le plan *P*0 comme étant le plan orienté défini par la face d'indice 0 du solide *A*, avant de demander l'impression du mot poème sur le plan *P*0, avec une fonte de taille `30 pts`, au point de coordonnées `(0,3)` (dans le repère lié à ce plan). On aurait pu changer l'orientation du texte avec `phi=-90` par exemple, dans l'une ou l'autre de ces commandes.

Par défaut, si la face n'est pas visible le texte reste caché. En écrivant `visibility` dans les options, le texte apparaîtra même s'il est caché comme dans l'exemple suivant.

On n'oubliera pas d'écrire à la fin des commandes d'écriture des textes `\composeSolid`, pour que tous ces tracés soient pris en compte. Toute autre commande de `PStricks` aura le même effet et `composeSolid` sera superflu.






les figuiers dans les jardins de l'Alameda et toutes
les nuelles bizarres les maisons roses et bleues et jaunes
et les roseraies et les jasmins et les géraniums
O cet effrayant torrent tout au fond O et la mer éclatée
les cactus de Gibraltar quand j'étais jeune fille
dans Ulysse de James Joyce
et une Fleur de la montagne oui
le monologue de Molly
quand j'ai mis la rose dans mes cheveux
comme les filles Andalouses
sous le mur mauresque
et comme il m'a embrassée
et alors je
je me suis dit après

Chapitre 10

Annexe

10.1 Les paramètres de `pst-solid`

Paramètre	Défaut	Signification
`viewpoint`	10 10 10	coordonnées du point de vue
`a`	2	arête du cube, rayon de la sphère circonscrite aux polyèdres réguliers, l'une des arêtes du parallélépipède
`r`	2	rayon du cylindre et de la sphère
`h`	6	hauteur du cylindre, cône, tronc de cône, du prisme
`r0`	1.5	rayon intérieur du tore
`r1`	4	rayon moyen du tore
`phi`	0	latitude pour découper la calotte sphérique vers la bas
`theta`	90	latitude pour découper la calotte sphérique vers le haut
`a,b et c`	4	les 3 arêtes du parallélépipède
`base`	-1 -1 1 -1 0 1	coordonnées d'un triangle, base triangulaire par défaut
`axe`	0 0 1	axe du prisme
`action`	draw**	utilise l'algorithme du peintre pour tracer le solide sans les arêtes cachées et avec coloration des faces
`lightsrc`	20 30 50	coordonnées cartésiennes de la source lumineuse
`lightintensity`	2	intensité de la source de lumière
`ngrid`	n1 n2	permet à l'utilisateur de spécifier la finesse du maillage pour le solide considéré
`mode`	0	permet de passer du maillage prédéfini le plus grossier `[mode=0]` au maillage prédéfini le plus fin `mode=4`
`grid`	true	si l'option `[grid]` est écrite, alors les lignes du maillage ne sont pas tracées
`biface`	true	dessine la face intérieure, si on ne souhaite représenter que l'extérieur, écrire alors : `[biface]`
`algebraic`	false	permet si `algebraic=true` (écrire alors : `[algebraic]`) de donner l'expression algébrique de l'équation de la surface, le package `pstricks-add` doit être chargé dans le préambule.
`fillcolor`	white	permet de spécifier la couleur souhaitée pour les faces externes du solide.
`incolor`	green	permet de spécifier la couleur souhaitée pour les faces intérieures du solide.
`hue`		permet de spécifier le dégradé de couleurs pour les faces extérieures du solide.

Suite à la page suivante

Paramètre	Défaut	Signification
fcol		permet de spécifier dans l'ordre : le numéro de la facette de 0 à n-1, pour n facettes ; la couleur de la facette : fcol=0 (Apricot) 1 (Aquamarine) etc.
rm		permet de supprimer des facettes visibles : rm=1 2 8 supprime les facettes 1, 2 et 8
show		permet de pointer les sommets : show=0 1 2 3 pointe les sommets 0,1,2 et 3, show=all pointe tous les sommets.
num		permet de numéroter les sommets : num=0 1 2 3 numérote les sommets 0,1,2 et 3, num=all numérote tous les sommets.
name		nom attribué à un solide
solidname		nom du solide activé
RotX	0	angle de rotation du solide autour de Ox en degrés
RotY	0	angle de rotation du solide autour de Oy en degrés
RotZ	0	angle de rotation du solide autour de Oz en degrés
hollow	false	dessine l'intérieur des solides creux : cylindre, cône, tronc de cône et prisme
decal	-2	permet de faire subir un décalage des sommets dans le paramètre base
axesboxed	false	cette option permet de tracer un quadrillage 3D et les axes de façon semi-automatique, car il convient de placer à la main les bornes de z, écrire [axesboxed]
Zmin	−4	valeur minimale de z
Zmax	4	valeur maximale de z
QZ	0	permet de décaler verticalement le repère de la valeur choisie
spotX	dr	permet de placer, si le choix fait par défaut n'est pas satisfaisant, les valeurs des graduations sur l'axe des x autour de l'extrémité de la graduation.
spotY	dl	"idem"
spotZ	l	"idem"
resolution	36	nombre de points lors du tracé d'une courbe
range	-4 4	limites pour les fonctions numériques
function	f	nom attribué à une fonction
path	newpath 0 0 moveto	le chemin à projeter
normal	0 0 1	normale à la surface à définir
text		texte à projeter
visibility	false	si "false" le texte appliqué une face cachée n'est pas affiché
chanfreincoeff	0.2	coefficient de chanfreinage
trunccoeff	0.25	coefficient de troncature
affinagecoeff	0.8	coefficient d'affinage
dualregcoeff	1	coefficient du dual
intersectiontype	-1	type d'intersection plan/solide, une valeur positive trace l'intersection
plansection		équations des plans de section, s'utilise que pour la trace de l'intersection
plansepare		équation du plan de séparation du solide
intersectionlinewidth	1	épaisseur de la ligne d'intersection en points, s'il y a plusieurs plans avec des épaisseurs différentes, les valeurs sont données à la suite, par exemple : intersectionlinewidth=1 1.5 1.8 etc.
intersectioncolor	(rouge)	couleur de la ligne d'intersection du plan de section, s'il y a plusieurs sections, il faut placer les couleurs à la suite (rouge) (vert) etc.
intersectionplan	[0 0 1 0]	équations des plans d'intersection
definition		pour définir un point, un vecteur, un plan, un arc sphérique etc.
args		ce sont les arguments associés aux définitions précédentes

Suite à la page suivante

Paramètre	Défaut	Signification
transform		transformation d'un solide par application de formules
section	\Section	coordonnées des sommets de la section de l'objet de révolution
planmarks	false	graduation des axes du plan
plangrid	false	quadrillage du plan
showbase	false	dessine les vecteurs unitaire du plan
showBase	false	dessine les vecteurs unitaire et la normale au plan
deactivatecolor	false	désactive la gestion des couleurs par PSTricks
transform		formule(s) à appliquer aux sommets d'un solide pour le transformer
axisnames	{x,y,z}	étiquettes des axes en 3D
axisemph		style des étiquettes des axes en 3D
showOrigin	true	trace les axes à partir de l'origine ou non si false
mathLabel	true	écriture des étiquettes des axes en mode mathématique ou non
		Fin de la table

10.2 Les poèmes

Dans ma jeunesse, j'écoutais le son de la pluie dans les maisons de plaisir ;
les tentures frissonnaient sous la lumière rouge des candélabres.
Dans mon âge mûr, j'ai écouté le son de la pluie en voyage, à bord d'un bateau ;
les nuages pesaient bas sur l'immensité du fleuve ;
une oie sauvage séparée de ses soeurs appelait dans le vent d'ouest.
Aujourd'hui, j'écoute le son de la pluie sous le charme d'un ermitage monastique.
Ma tête est chenue, chagrins et bonheurs, séparations et retrouvailles - tout est vanité.
Dehors, sur les marches, les gouttes tambourinent jusqu'à l'aube.

Juang Jie dans *Les idées de autres* de Simon Leys

O cet effrayant torrent tout au fond
O et la mer la mer écarlate quelquefois comme du feu
Et les glorieux couchers de soleil
Et les figuiers dans les jardins de l'Alameda
Et toutes les ruelles bizarres
Et les maisons roses et bleues et jaunes
Et les roseraies et les jasmins et les géraniums
Et les cactus de Gibraltar quand j'étais jeune fille
Et une Fleur de la montagne oui
Quand j'ai mis la rose dans mes cheveux comme les filles Andalouses
Ou en mettrai-je une rouge oui
Et comme il m'a embrassée sous le mur mauresque
Je me suis dit après tout aussi bien lui qu'un autre
Et alors je lui ai demandé avec les yeux de demander encore oui
Et alors il m'a demandé si je voulais oui
Dire oui ma fleur de la montagne
Et d'abord je lui ai mis mes bras autour de lui oui
Et je l'ai attiré sur moi pour qu'il sente mes seins tout parfumés oui
Et son coeur battait comme un fou
Et oui j'ai dit oui
Je veux bien Oui.

Monologue de *Molly Bloom* dans *Ulysse* de James Joyce

Bibliographie

[1] Hendri Adriaens. xkeyval package. CTAN:/macros/latex/contrib/xkeyval, 2004.

[2] Bill Casselman. *Mathematical Illustrations – a manual of geometry and PostScript*. Cambridge University Press, Cambridge, first edition, 2005.

[3] Denis Girou. Présentation de PSTricks. *Cahier GUTenberg*, 16 :21–70, April 1994.

[4] Michel Goosens, Frank Mittelbach, Sebastian Rahtz, Denis Roegel, and Herbert Voß. *The LaTeX Graphics Companion*. Addison-Wesley Publishing Company, Reading, Mass., 2007.

[5] Alan Hoenig. *TeX Unbound : LaTeX & TeX Strategies, Fonts, Graphics, and More*. Oxford University Press, London, 1998.

[6] Frank Mittelbach and Michel Goosens et al. *The LaTeX Companion*. Addison-Wesley Publishing Company, Boston, second edition, 2004.

[7] Sebastian Rahtz. An introduction to PSTricks, part I. *Baskerville*, 6(1) :22–34, February 1996.

[8] Sebastian Rahtz. An introduction to PSTricks, part II. *Baskerville*, 6(2) :23–33, April 1996.

[9] Herbert Voß. *PSTricks, Grafik für TeX und LaTeX*. DANTE – Lehmanns, Heidelberg/Hamburg, forth edition, 2007.

[10] Timothy Van Zandt and Denis Girou. Inside PSTricks. *TUGboat*, 15 :239–246, September 1994.