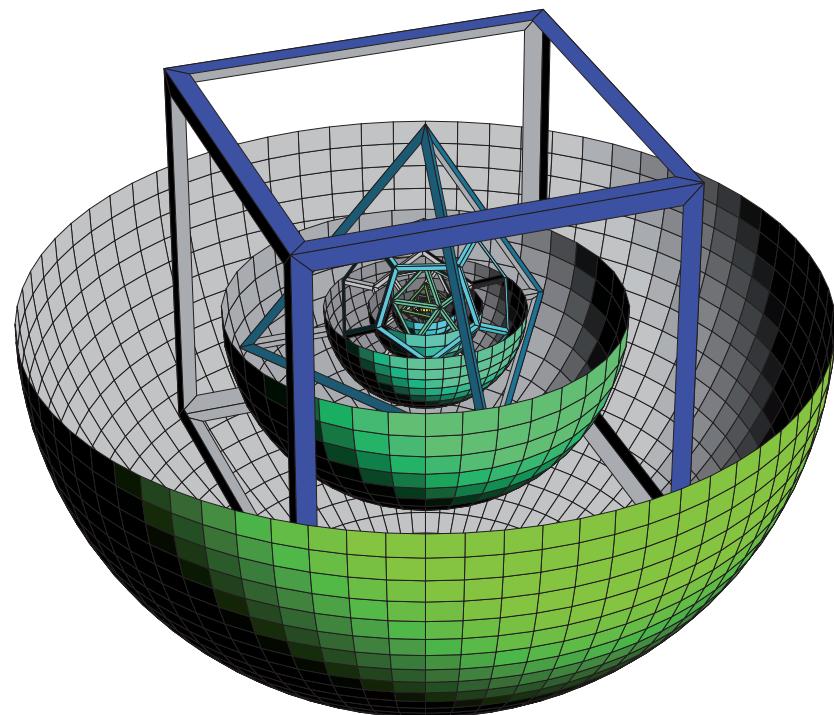


pst-solides3d: **The Documentation – The Basics**

v. 4.28 (2014/09/05)



Jean-Paul Vignault* Manuel Luque† Arnaud Schmittbuhl‡

Jürgen Gilg§ Jean-Michel Sarlat¶ Herbert Voß||

September 5, 2014

*jpv@melusine.eu.org
†manuel.luque27@gmail.com
‡aschmittbuhl@libertysurf.fr
§gilg@acrotex.net
¶jm.sarlat@gmail.com
||herbert.voss@fu-berlin.de

Contents

1. Basics for the package	9
1.1. Constitution of the package – Distribution	9
1.2. Installation hints	9
1.3. Preface	10
1.4. Presentation	10
1.5. Changes by comparison with previous versions	11
1.5.1. Changes compared to version 3.0	11
1.5.2. Changes compared to version 2.0	11
2. Setting the layout of the scenery	13
2.1. Choice of the view point	13
2.2. The definition of the option <code>Decran</code>	14
2.3. Lighting by a point light source	16
2.4. The axes in 3d	18
3. Predefined solids and their positioning	19
3.1. The predefined solids and their parameters	19
3.2. Positioning a solid	27
3.2.1. Translation	27
3.2.2. Rotation	28
3.2.3. Default sequence xyz	28
3.2.4. Rotations Sequence	28
4. More options of \psSolid	31
4.1. Commands for drawing	31
4.2. Emptying a solid	31
4.3. Numbering of the faces	34
4.4. Removing faces	35
4.5. Numbering of the vertices	36
4.6. Colours and the nuances of a colour	37
4.6.1. Predefined colours by the option <code>dvipsnames</code>	37
4.6.2. Predefined colours by the option <code>svgnames</code>	38
4.6.3. Nuances in the colour scheme of HSB, saturation and maximum brilliance	41
4.6.4. Nuances in the colour scheme of HSB, saturation and fixed brilliance	42
4.6.5. Nuances in the colour scheme of HSB, gneral case	42
4.6.6. Nuances in the colour scheme of RGB	43
4.6.7. Nuances in the colour scheme of CMYK	43
4.6.8. Nuances between 2 named colours	43

Contents

4.6.9. Deactivation of the colour application	44
4.7. Colouring some single faces	44
4.8. Nuances of transparency	46
4.9. Definition of grating	46
4.9.1. The cube	47
4.9.2. Sphere	47
4.9.3. Cylinders	48
4.9.4. Torus	49
4.10. The modes	49
4.11. Truncate a solid's vertices	50
4.12. Hollowing out a solid's faces	51
4.13. Chamfering a solid	52
4.14. The option transform	53
4.14.1. Identical scaling factor in the three coordinates	53
4.14.2. Different scaling factors for the three coordinates	54
4.14.3. Transformation associated with the distance to the origin	55
4.14.4. Bending and torsion of beams	56
4.15. Lines of intersecting planes	58
5. Usage of external files	59
5.1. Using the data file types .obj and .off	59
5.1.1. .dat files (specific to pst-solides3d)	60
5.1.2. .obj files	60
5.1.3. .off files	61
6. Some special objects	63
6.1. The grid	63
6.2. The object point	64
6.2.1. Definition via coordinates	64
6.2.2. Some other definitions	64
6.3. The object vecteur	65
6.3.1. Definition with components	65
6.3.2. Definition with 2 points	66
6.3.3. Some other definitions of a vector	66
6.4. The object plan	67
6.4.1. Presentation: type plan and type solid	67
6.4.2. Defining an oriented plane	67
6.4.3. Special options	68
6.4.4. Defining a plane with a cartesian equation	68
6.4.5. Defining a plane using a normal vector and a point	70
6.4.6. Defining a plane from a face of a solid	72
6.5. The object geode	72
6.5.1. Mathematical presentation	72
6.5.2. Construction with pst-solides3d	73
6.5.3. Some examples of geodes and their duals	73
6.5.4. The parameters of the geodes	74
6.5.5. Advice for a 'fast' construction of a geode	75
6.5.6. Some other examples	76
7. Generating some new solids	77

7.1.	The jps code	77
7.2.	Defining a function	78
7.3.	Curves of functions from R in R ³	79
7.4.	Tubes	80
7.4.1.	Usage with PSTricks	81
7.4.2.	Usage with \code{jps}	82
7.4.3.	Improving the speed of readout	83
7.4.4.	Some other examples	86
7.5.	The prism	87
7.6.	Construction from scratch	91
7.6.1.	Example 1: a house	92
7.6.2.	Example 2: a hyperboloid with a fixed radius	92
7.6.3.	Example 3: importing external files	93
7.7.	One- and two-sided solids	94
7.7.1.	Triangular ‘faces’	95
7.7.2.	‘face’ defined by a function	95
7.8.	Solid strip	95
7.8.1.	A simple folding screen	96
7.8.2.	A sinusoidal folding screen	96
7.8.3.	A corrugated surface	96
7.8.4.	An asteroidal folding screen: version 1	97
7.8.5.	An asteroidal folding screen: version 2	98
7.9.	Solid rings	98
7.9.1.	Predefined command: the ring with a rectangular section	98
7.9.2.	Example 1: a simple ring with a triangular section	100
7.9.3.	Example 2: a ring with a variable triangular section	101
7.9.4.	Example 3: a ring with a “tyre”-like section: cylindric ring with chamfered edges	102
7.9.5.	Example 4: an empty bobbin	103
7.9.6.	Some other rings	103
7.10.	Generalization of the notion of a cylinder and a cone	104
7.10.1.	Cylinder or cylindric area	104
7.10.2.	Cone or conic area	107
7.11.	Parameterised surfaces	109
7.11.1.	The method	109
7.11.2.	Example 1: a sea shell	110
7.11.3.	Example 2: a helix	111
7.11.4.	Example 3: a cone	111
7.11.5.	An advised website	112
8.	Surfaces defined by a function of the form $z = f(x, y)$	113
8.1.	Presentation	113
8.2.	Example 1: a saddle	114
8.3.	Example 2: a saddle without a grid	115
8.4.	Example 3: a paraboloid	115
8.5.	Example 4: a sinusoidal wave	116
8.6.	Example 5: another sinusoidal wave	116
8.7.	Example 6: a hyperbolic paraboloid with the equation $z = xy$	116
8.8.	Example 7: a surface with the equation $z = xy(x^2 + y^2)$	117
8.9.	Example 8: a surface with the equation $z = \left(1 - \frac{x^2+y^2}{2}\right)^2$	117

Contents

9. Advanced usage	119
9.1. Naming a solid	119
9.2. Sectioning a solid with a plane	120
9.2.1. Drawing the intersection between a plane and a solid	120
9.2.2. Slicing a solid	121
9.2.3. Slice of a pyramid	123
9.2.4. Slicing an octahedron with a plane parallel to one of its faces	125
9.2.5. Slices of a cube	128
9.2.6. Multiple sections	131
9.2.7. Sections of a torus	134
9.2.8. Some more examples	134
9.3. Fusing solids	135
9.4. Fusing with <i>jps code</i>	136
9.4.1. Using <i>jps code</i>	136
9.4.2. A chloride ion	138
9.4.3. A prototype of a vehicle	140
9.4.4. A wheel – or a space station	142
9.4.5. Intersection of two cylinders	143
9.4.6. Intersection between a sphere and a cylinder	143
9.4.7. Two linked rings	144
9.4.8. The methane molecule: wooden model	144
9.4.9. The thiosulphate ion	146
10. Interaction with PStricks	149
10.1. Positioning a named point	149
10.2. Drawing a line	150
10.3. Drawing a polygon	150
10.4. Transformations to a point	151
10.5. Adding dimensions to the scenery	152
11. Projections	153
11.1. Presentation	153
11.2. The parameter visibility	153
11.3. Defining a projection plane	153
11.4. Points	154
11.4.1. Direct definition	154
11.4.2. Labels	154
11.4.3. Naming and memorising a point	154
11.4.4. Some other definitions	155
11.5. Vectors	157
11.5.1. Direct definition	157
11.5.2. Some more definitions	157
11.6. Lines	158
11.6.1. Direct definition	158
11.6.2. Some other definitions	159
11.7. Circles	159
11.7.1. Direct definition	159
11.7.2. Some other definitions	160
11.8. Polygons	160
11.8.1. Direct definition	160

11.9. Lines	162
11.9.1. Direct definition	162
11.10. Right angle	163
11.10.1. Direct definition	163
11.11. Curves of real-valued and parameterised functions	164
11.11.1. Curve of a real-valued function	164
11.11.2. Parameterised curves	165
11.12. Text	166
11.12.1. The parameters and the options	166
11.12.2. Examples of projecting onto a plane	167
11.12.3. Examples for projecting onto a face of a solid	170
11.12.4. Examples of projecting onto different faces of a solid	171
11.13. Projection of images	174
11.14. A bit of theory	177
12. Possible extensions	179
12.1. Creating your own object	179
12.2. Creating a .u3d file	179
A. Appendix	181
A.1. The parameters of <code>pst-solides3d</code>	181
A.2. Alphabetical list of keywords	184
A.3. Acknowledgments	190
A.4. The poems	190

Contents

CHAPTER 1

Basics for the package

1.1. Constitution of the package – Distribution

- **Required files:** `pst-solides3d.sty`, `pst-solides3d.tex`, `solides.pro` and the latest version of the basic PStricks package.
- **Workflow:** This package is made for dvips and ps2pdf, however pdf_{TEX} won't work.
- **Documentation and examples:** `pst-solides3d-doc.tex`(pdf), `doc-exemples-solides3d.tex`(pdf).

This package is available on: <http://syracuse.eu.org/syracuse/pstricks/pst-solides3d/> as well as on CTAN.

Numerous examples are available on: <http://syracuse.eu.org/lab/bpst/pst-solides3d>

Finally, the actual developer's version is available on the SVN of *mélusine*: <http://syracuse-dev.org/pst-solides3d>

1.2. Installation hints

Here we give some hints on how to install `pst-solides3d` on your T_{EX} system.

The `pst-solides3d` package consists of three main files:

- `solides.pro`: the prolog file for `pst-solides3d`
- `pst-solides3d.sty`: the appropriate style file
- `pst-solides3d.tex`: the appropriate tex file

as well as the actual PStricks base files:

- `pstricks.pro`: the prolog file for pstricks
- `pstricks.tex`: the appropriate tex file

1. Basics for the package

available on CTAN.

Some extension files for **pst-rubans**:

- **pst-rubans.sty**: the appropriate style file
- **pst-rubans.tex**: the appropriate tex file

Save the files **pst-solides3d.sty|tex**, **pst-rubans.sty|tex** and **pstricks.tex** in a directory which is part of your local **T_EX** tree.

However the **solides.pro** and the **pstricks.pro** file should go into the folder **\$TEXMF/dvips/pstricks/**

Do not forget to run **texhash** to update this tree. For MiK_TE_X users, do not forget to update the file name database (FNDB).

For more detailed information see the documentation of your personal L^AT_EX distribution on installing packages to your local **T_EX** system.

1.3. Preface

The package presented in this documentation arose from teamwork initiated via the mailing list of the syracuse web site (<http://melusine.eu.org/syracuse>).

The idea was born of a confrontation between the work of Jean-Paul VIGNAULT on the software package *jps2ps*¹ and Manuel LUQUE's work on PSTricks², especially in relation to the subject of representing solids in three-dimensional space.

The two authors decided to unify their efforts and co-author a PSTricks package dedicated to three-dimensional scenes. The work took place on the “machine *mélusine*” within an environment generated and maintained by Jean-Michel SARLAT.

The team was completed with the addition of Arnaud SCHMITTBUHL, Herbert Voss and Jürgen GILG, the latter specialising in animation-based beta-testing³.

1.4. Presentation

The package **pst-solides3d**, with the help of PSTricks, allows for 3D views of predefined or user-generated solids. You will find most of the usual solids, which can be drawn with or without hidden edges, whose colour can be varied with lighting.

This package can project text or simple graphics (in 2D) onto arbitrarily chosen planes or onto plane faces of solids that are created by the user.

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/>

From the user's standpoint, most of its functionalities are accessible by way of three \TeX macros: `\psSolid`, which can manipulate objects in 3 dimensions, `\psSurface`, related to the first macro and designed to represent surfaces that are defined by an equation of the type $f(x, y) = z$ and `\psProjection` which allows the user to project two-dimensional graphics/text onto any plane face of a 3D solid.

In using this package, two languages come together: on the one hand PSTricks, with its well-known macros and familiar syntax, and on the other PostScript code, which appears within the optional arguments of the former.

We have made the decision to strictly limit the involvement of PSTricks. Its function is only to transmit parameters from \TeX to PostScript. All calculations and displays are done by the latter.

A PostScript library, which was developed for another application (the software package *jps2ps*), is used for all calculations and display routines. The PostScript code used in this library is called *jps code*.

The aim of the present document is to describe PSTricks syntax for each operation provided by the package.

1.5. Changes by comparison with previous versions

1.5.1. Changes compared to version 3.0

- The macro `\psProjection` has been completely rewritten. We now need to use an object of type `plan` to define a projection.
- The object `courbe` now uses the argument r . To reproduce the previous behaviour we now have to specify $r = 0$.
- The option `resolution` of the object `courbe` is replaced with the option `ngrid`
- Suppression of the argument `tracelignedeniveau`.

1.5.2. Changes compared to version 2.0

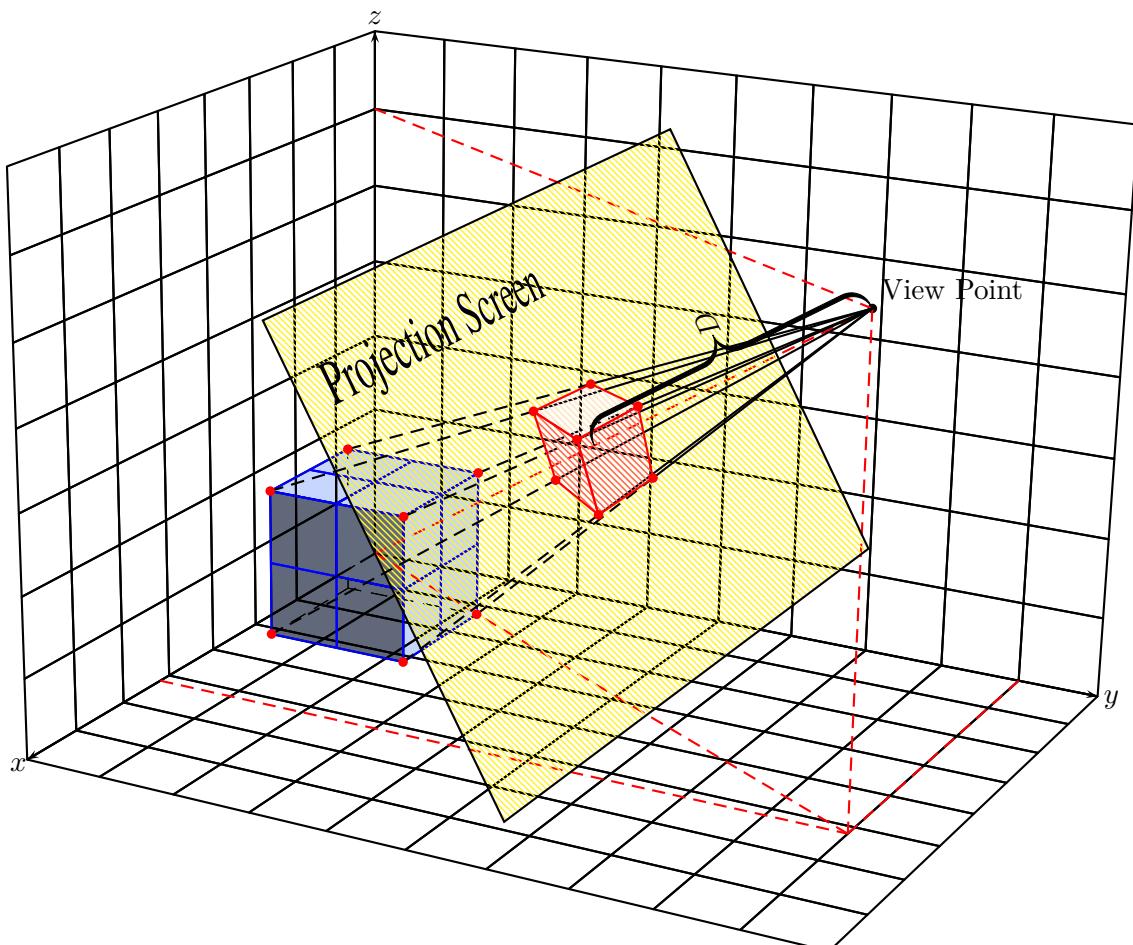
- The option `hue` is not a Boolean anymore.
- The scaling in PostScript will from now on follow the workings of *jps code*. To be consistent, the commands `smoveeto`, `srmoveto`, `slineto`, `srlineto` now respectively replace the commands `moveto`, `rmoveto`, `lineto`, `rlineto`.

1. Basics for the package

CHAPTER 2

Setting the layout of the scenery

2.1. Choice of the view point



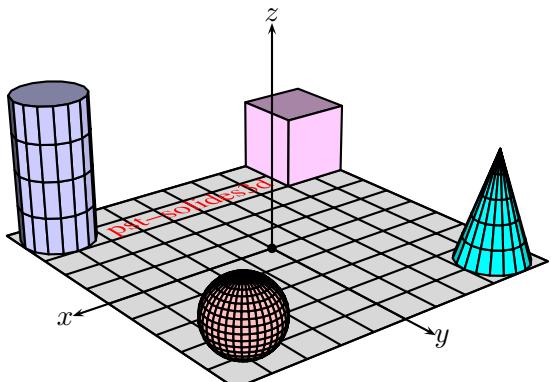
The coordinates of the object, in this case the bluish cube, are setup in the axes of coordinates $Oxyz$. The coordinates of the view point (V), are setup in the same axes of coordinates, either in spherical

2. Setting the layout of the scenery

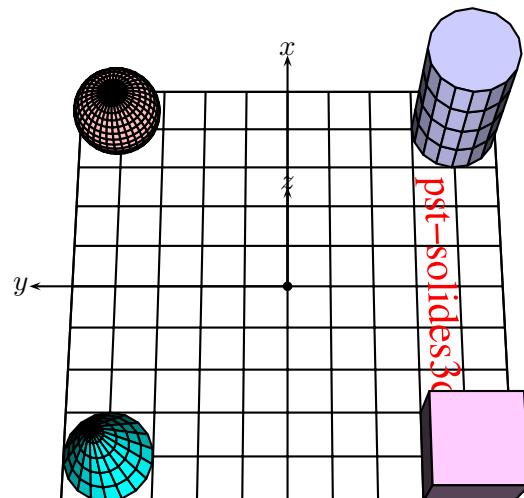
coordinates—with the adding option [rtp2xyz], or in Cartesian coordinates—which is the default option.

Example: [viewpoint=50 30 20 rtp2xyz] (here the notation with spherical coordinates)

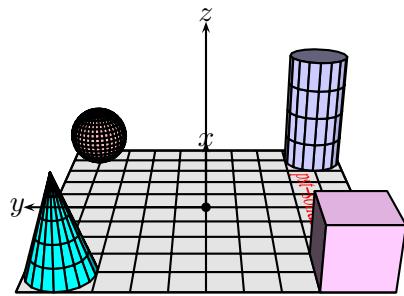
See some examples:



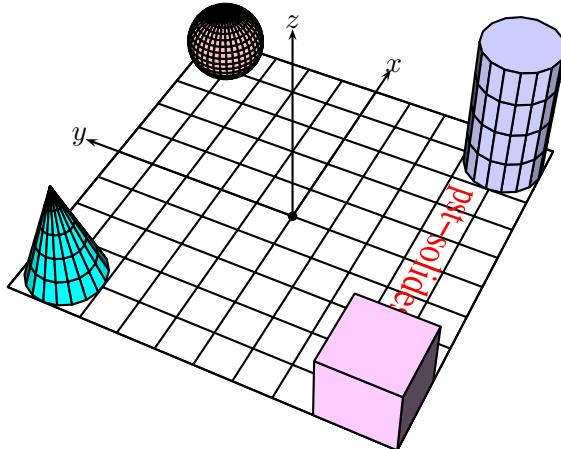
viewpoint=20 25 15



viewpoint=-10 0 30



viewpoint=-20 0 10

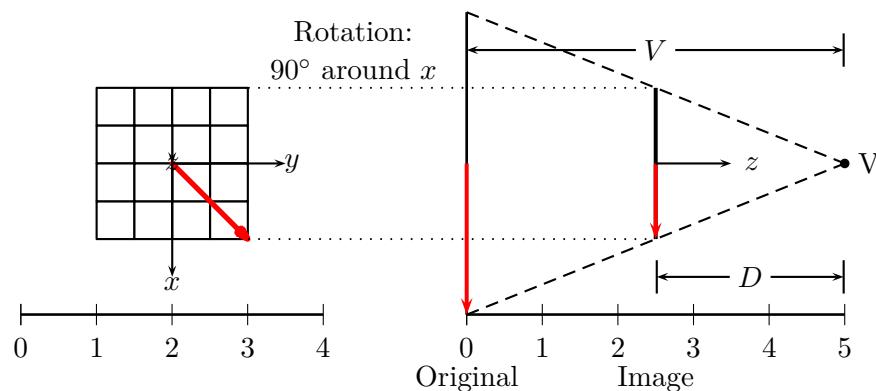
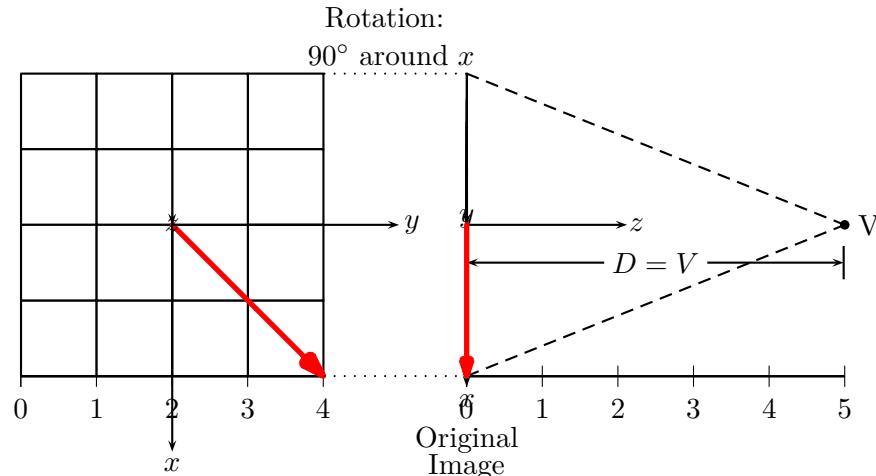


viewpoint=-20 -10 25

2.2. The definition of the option Decran

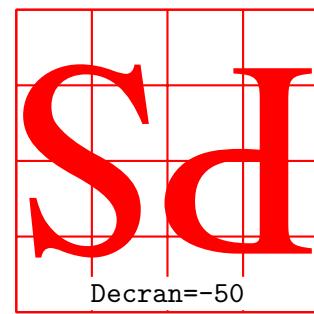
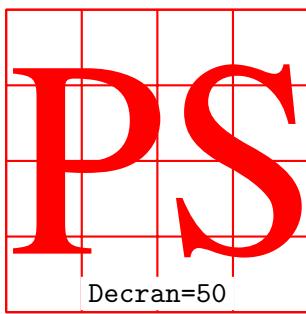
The projection screen is placed perpendicular to the direction OV —central perspective, at a distance D from the view point V : We call that distance ‘Decran’, with the default value of `Decran=50`; this value can either be positive or negative.

The following examples show the behaviour of the parameter `Decran`.



If you keep the view point and make the *Decran* value smaller, then the image gets smaller. If you make the *Decran* value larger, then the image gets larger.

Here are some examples, where we keep the same object, the same view point and just vary the *Decran* value:

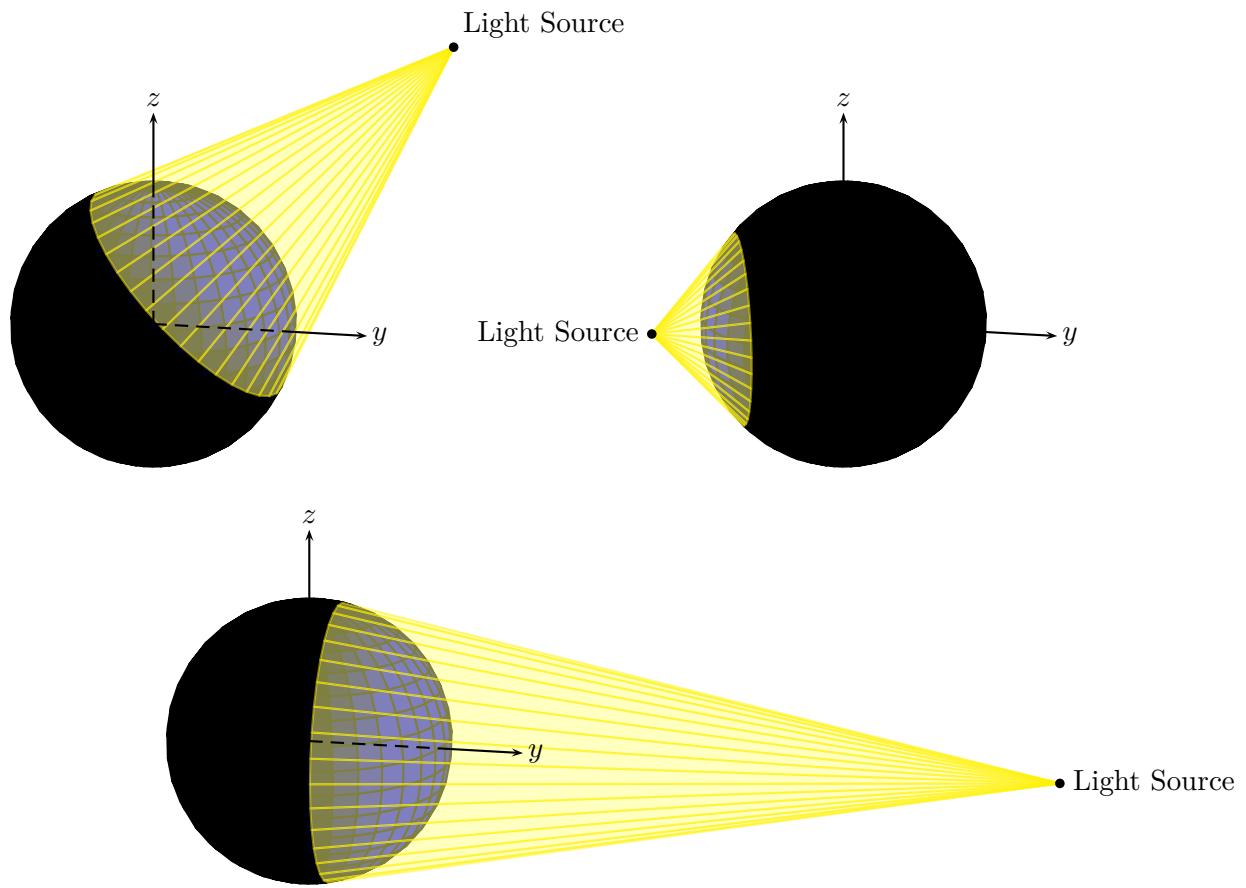


2. Setting the layout of the scenery

2.3. Lighting by a point light source

Two parameters, the first one positions the light source, the second one sets the light intensity:

- `lightsrc=20 30 50` in Cartesian coordinates, or `lightsrc=viewpoint` to put the light source at the view point.
- `lightintensity=2` (default value).

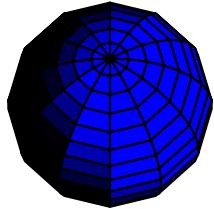


As you can see, the intersecting plane (section of the sphere with the cone of light) divides the object into two half spaces: the first half space (the one on the side of the light source) is illuminated and the other half space is the shadow region referring to this light source position.

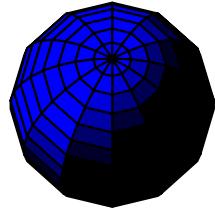
Now it is clear, that if the view point is setup with the same coordinates as the light source, the object is illuminated uniquely.

Note: In order to get some shadow regions to appear in the graphic—which emphasises the 3D character—we would suggest choosing the light source and the view point differently.

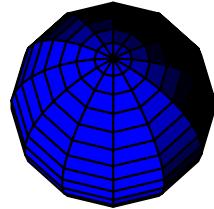
Here follow a few examples:



`lightsrc=10 20 30`

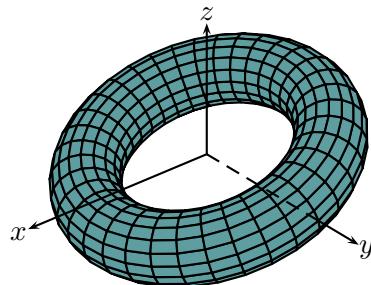


`lightsrc=-10 -20 30`

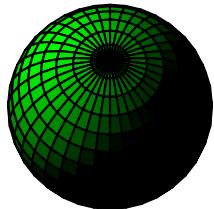


`lightsrc=30 -20 30`

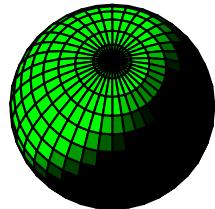
When the option `[lightsrc=value1 value2 value3]` is not specified, the object is uniformly illuminated.



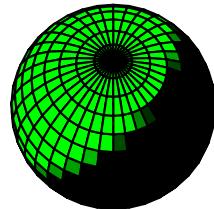
Here are some examples, where we always keep the same object, the same view point, the same light source coordinates and just vary the `lightintensity` value:



`lightintensity=2`



`lightintensity=3`



`lightintensity=8`

Here we can see, that by increasing the `lightintensity` value, the shading nuances of the solid are decreasing.

2. Setting the layout of the scenery

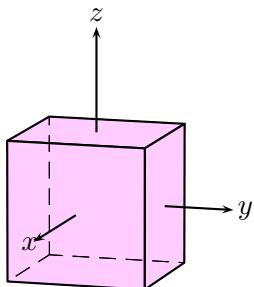
2.4. The axes in 3d

The command `\axesIID[options](x1,y1,z1)(x2,y2,z2)` draws the axes Ox , Oy and Oz dashed from the origin O to the coordinates $(x_1, 0, 0)$ for the x -axis, $(0, y_1, 0)$ for the y -axis and $(0, 0, z_1)$ for the z -axis and from there continues drawing the axes as lines to the points $(x_2, 0, 0)$, $(0, y_2, 0)$ and $(0, 0, z_2)$.

The options are the following:

- all colour options, line width as well as all types of arrows.
- `labelsep=length` which allows you to position the label in a self defined distance away from the extremity of the arrow of the axis, the default value is `labelsep=5pt`—this is a real distance in three dimensions and not on screen.
- the choice of the labels on each of the axes with the option:
`axisnames=a,b,c`, the default values are `axisnames=x,y,z`.
- the potential to specify the style of the labels with the option:
`axisemph=\boldmath\Large\color{red}`. By default there is no style predefined, which means, if no style is chosen one will get `x`, `y`, `z`.
- `showOrigin` is a Boolean, `true`—by default. If it is set to `showOrigin=false` the dashed lines aren't drawn to the origin anymore.
- `mathLabel` is a Boolean, `true`—by default, in which case the math mode is activated. Set to `mathLabel=false` the labels are set in text mode.

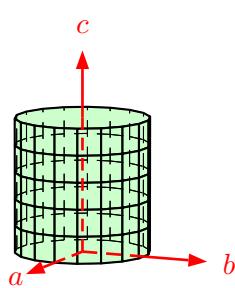
Note: The labels are placed at the extemities of the axes.



```

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

```



```

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

```

CHAPTER 3

Predefined solids and their positioning

3.1. The predefined solids and their parameters

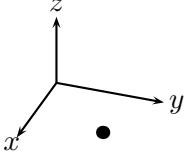
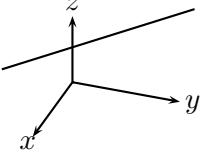
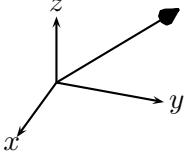
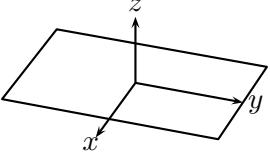
The basic command is: `\psSolid[object=name](x,y,z)` which allows us to translate the chosen object to the point with the coordinates (x, y, z) .

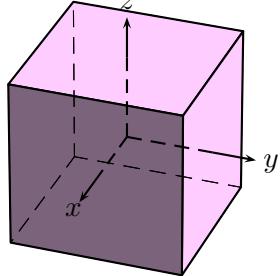
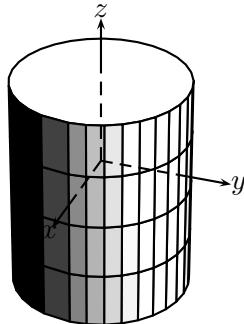
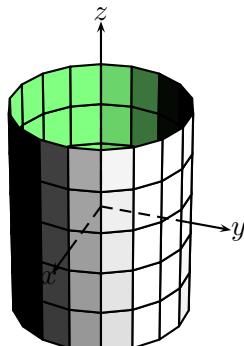
The available predefined names for the objects are:

`point, line, vector, plan, grille, cube, cylindre, cylindrecreux, cone, conecreux, tronccone, troncconecreux, sphere, calottesphere, calottespheredeuse, tore, tetrahedron, octahedron, dodecahedron, isocahedron, anneau, prisme, prismecreux, parallelepiped, face, polygonregulier, ruban, surface, surface*, surfaceparametree, pie, fusion, geode, load, offfile, objfile, datfile, new.`

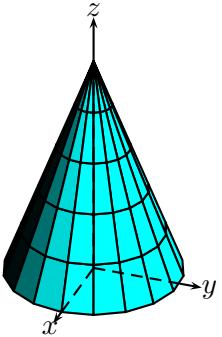
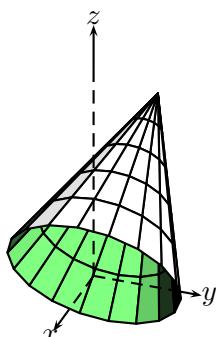
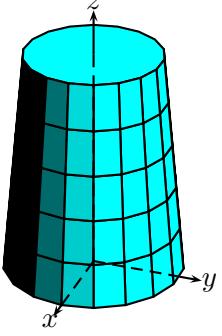
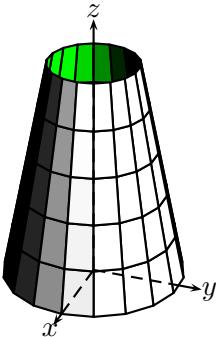
The following table gives an example of every one of the above named solids with their specified parameters:

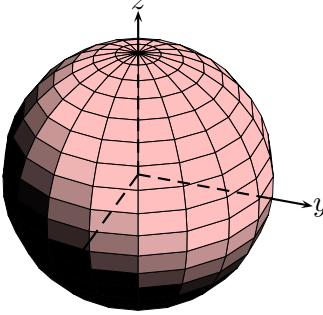
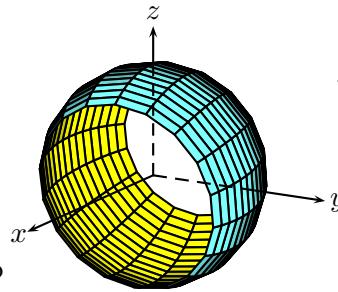
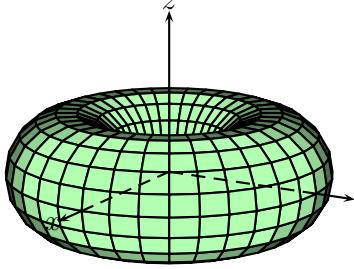
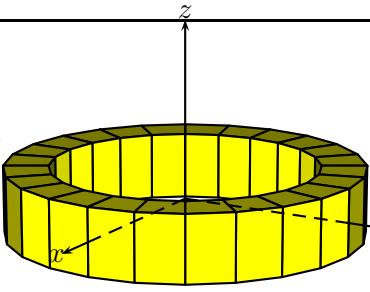
3. Predefined solids and their positioning

Solid	Default Parameters	View	Code
Point	[args=1 1 0] coordinates		\psSolid[object=point, args=1 1 0]%
Line	[args=0 -1 0 1 2 2] coordinates of the end points		\psSolid[object=line, args=0 -1 0 1 2 2]
Vector	[args=1 2 2] components of the vector		\psSolid[object=vecteur, args=1 2 2]
Plane	[base=-x x -y y] range of plane args=[0 0 1 0] equation of plane		\psSolid[object=plan, definition=equation, args={[0 0 1 0]}, base=-1 1 -1.5 1.5]

Solid	Default Parameters	View	Code
Cube	[a=4] edge's length		\psSolid[object=cube, a=2, action=draw*, fillcolor=magenta!20]
Cylinder	[h=6,r=2] height and radius grid: [ngrid=n1 n2]		\psSolid[object=cylindre, h=5,r=2, fillcolor=white, ngrid=4 32] (0,0,-3)
Hollow Cylinder	[h=6,r=2] height and radius grid: [ngrid=n1 n2]		\psSolid[object=cylindrecieux, h=5,r=2, fillcolor=white, mode=4, incolor=green!50] (0,0,-3)

3. Predefined solids and their positioning

Solid	Default Parameters	View	Code
Cone	[h=6,r=2] height and radius grid: [ngrid=n1 n2]		\psSolid[object=cone, h=5,r=2, fillcolor=cyan, mode=4]%
Hollow Cone	[h=6,r=2] height and radius grid: [ngrid=n1 n2]		\psSolid[object=conecreux, h=5,r=2, RotY=-60, fillcolor=white, incolor=green!50, mode=4]%
Truncated Cone	[h=6,r0=4,r1=1.5] height and radii grid: [ngrid=n1 n2]		\psSolid[object=tronccone, r0=2,r1=1.5,h=5, fillcolor=cyan, mode=4]%
Truncated Hollow Cone	[h=6,r0=4,r1=1.5] height and radii grid: [ngrid=n1 n2]		\psSolid[object=troncconecreux, r0=2,r1=1,h=5, fillcolor=white, mode=4]%

Solid	Default Parameters	View	Code
Sphere	[r=2] radius grid: [ngrid=n1 n2]		\psSolid[object=sphere, r=2,fillcolor=red!25, ngrid=18 18]%
Spherical zone	[r=2] radius [phi=0,theta=90] latitude for slicing the zone respectively from the bottom and top		\psSolid[object=calottesphere, r=3,ngrid=16 18, theta=45,phi=-30, hollow,RotY=-80]%
Torus	[r0=4,r1=1.5] inner radius mean radius grid: [ngrid=n1 n2]		\psSolid[r1=2.5,r0=1.5, object=tore, ngrid=18 36, fillcolor=green!30, action=draw*]%
Cylindric Ring	[R=4,r=3 inner and outer radius h=6,section=...] height cross section		\psSolid[object=anneau, fillcolor=yellow, h=1.5,R=4,r=3]%

3. Predefined solids and their positioning

Solid	Default Parameters	View	Code
Tetrahedron	[r=2] radius of the circumscribed sphere		\psSolid[object=tetrahedron, r=3, linecolor=blue, action=draw]%
Octahedron	[a=2] radius of the circumscribed sphere		\psSolid[object=octahedron, a=3, linecolor=blue, fillcolor=Turquoise]%
Dodecahedron	[a=2] radius of the circumscribed sphere		\psSolid[object=dodecahedron, a=2.5, RotZ=90, action=draw*, fillcolor=OliveGreen]%
Icosahedron	[a=2] radius of the circumscribed sphere		\psSolid[object=icosahedron, a=3, action=draw*, fillcolor=green!50]%
Prism	[axe=0 0 1] direction of the axis [base= -1 -1 1 -1 0 1] coordinates of the vertices of the base [h=6] height		\psSolid[object=prisme, action=draw*, linecolor=red, h=4]%

Solid	Default Parameters	View	Code
Grid	[base=-X +X -Y +Y]		\psSolid[object=grille, base=-5 5 -3 3]%
Cuboid	[a=4,b=3,c=2] edge lengths with center in 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.] the coordinates of the vertices		\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)
Strip	[base=x0 y0 x1 y1 x2 y2 etc.] [h=height] [ngrid=value] number of gridlines [axe=0 0 1] direction of inclination of the strip		\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])

3. Predefined solids and their positioning

	Solid	Default Parameters	View	Code
Surface	see the related paragraph in the documentation			\psSurface[ngrid=.25 .25, incolor=white,axesboxed] (-4,-4)(4,4){% x dup mul y dup mul 3 mul sub x mul 32 div}
New	solid defined by the coordinates of the vertices and the vertices of each face			\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]}]%
Curve	curve of a function $\mathbb{R} \rightarrow \mathbb{R}^3$ defined by its paramterised equations			\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]%

Some information about rings and parallelepipeds is available in the documents:

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

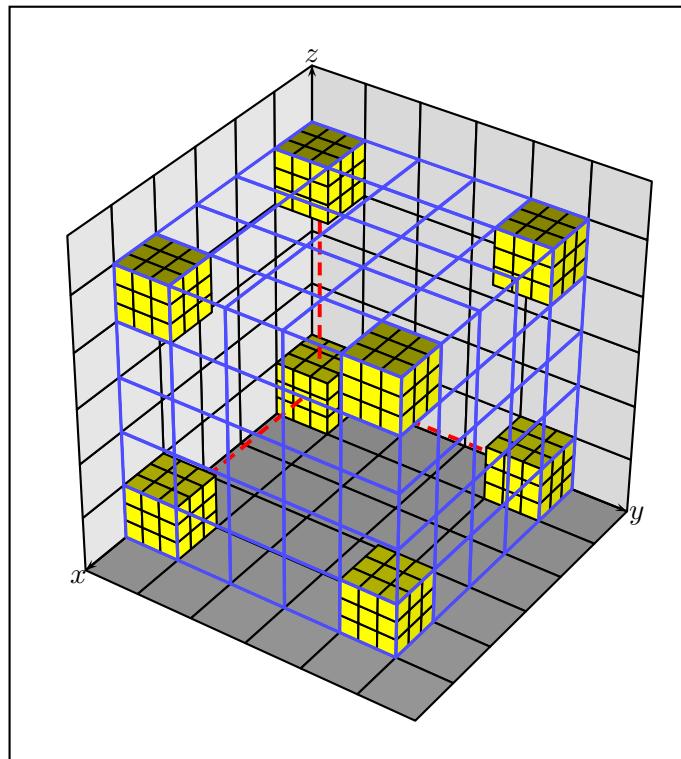
3.2. Positioning a solid

3.2.1. Translation

The following command `\psSolid[object=cube,+options](x,y,z)` shifts the centre of the cube to the point with the coordinates (x, y, z) .



The next example will copy the cube with edge length of 1 to the points with the coordinates $(0.5, 0.5, 0.5)$, $(4.5, 0.5, 0.5)$ etc. so that the copied cubes setup the vertices of a new cube with the edge length 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)
```

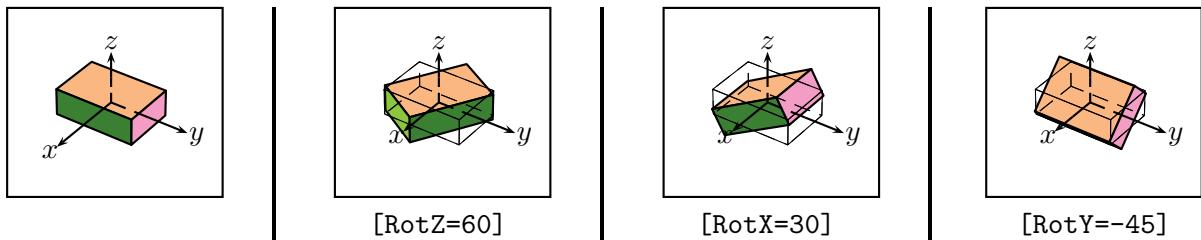
3. Predefined solids and their positioning

```
\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)
```

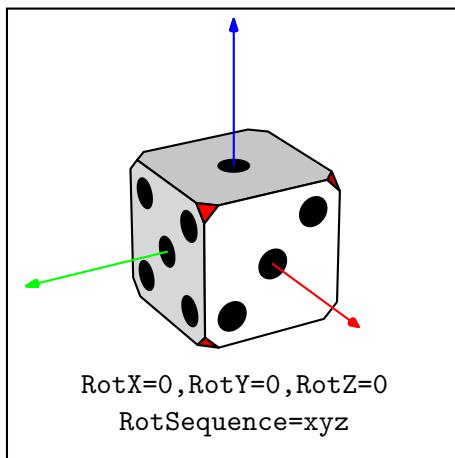
3.2.2. Rotation

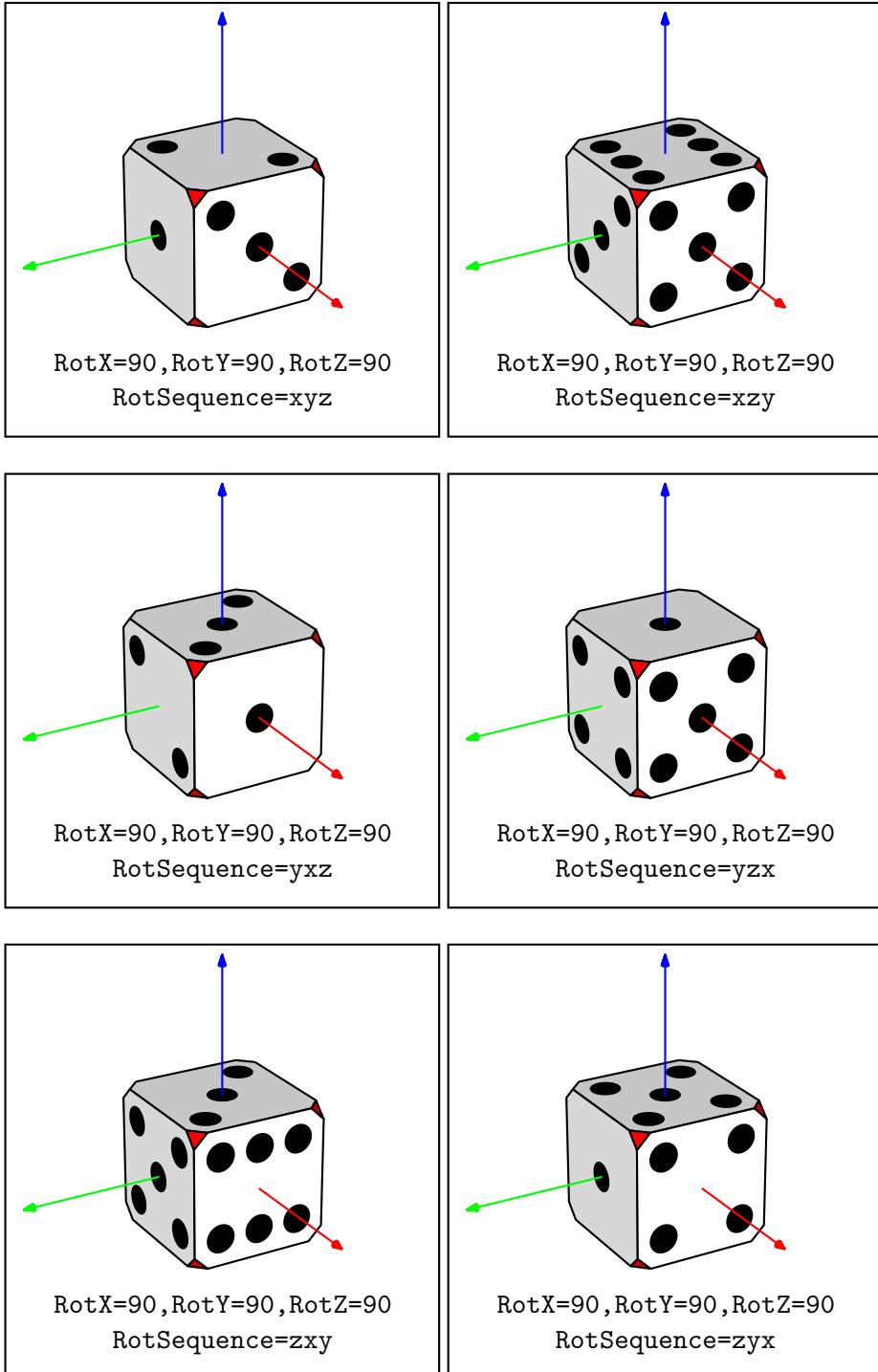
3.2.3. Default sequence xyz

The rotation is effected around the three axes Ox , Oy and Oz . Let's take a cuboid as an example,



3.2.4. Rotations Sequence





3. Predefined solids and their positioning

CHAPTER 4

More options of \psSolid

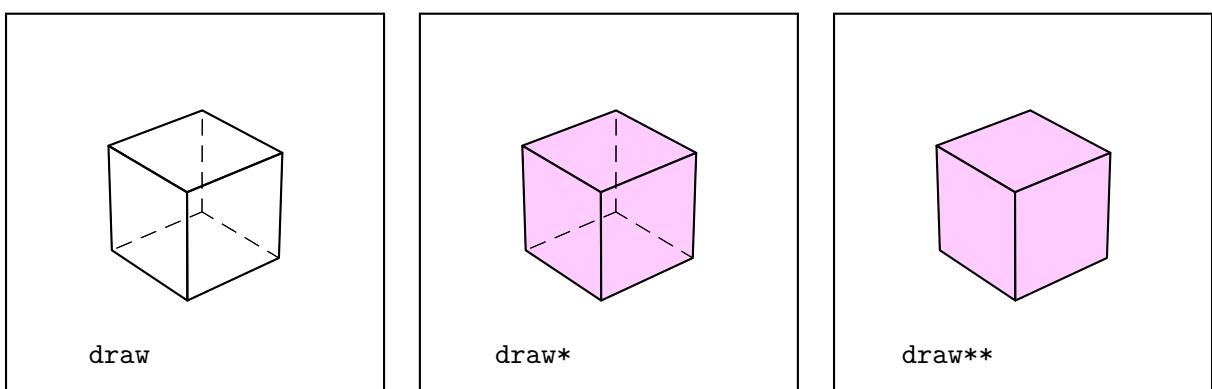
4.1. Commands for drawing

The parameter for drawing comes with the key value `action` within the command `\psSolid`.

Four values are possible:

- `none`: nothing is drawn.
- `draw`: draws the solid as a framework and sets up dashed lines for the hidden edges.
- `draw*`: draws the solid with dashed lines for the hidden edges and colours the visible faces.
- `draw**`: draws the solid with a painting algorithm, without the hidden edges and with colouration of the visible faces.

Note: The key values `draw` and `draw*` only make sense for convex solids.



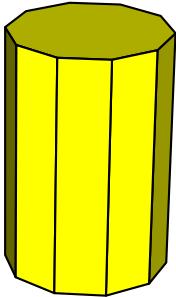
4.2. Emptying a solid

Several of the predefined solids have a “*hollow*” relative which is naturally associated with it (the cone, the truncated cone, the cylinder, the prism and the spherical zone). For all those, the option

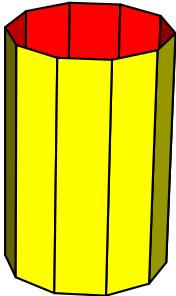
4. More options of `\psSolid`

`hollow=true` is provided. Set to `false`, we get the “filled” solid; set to `true` we get the “hollow” version.

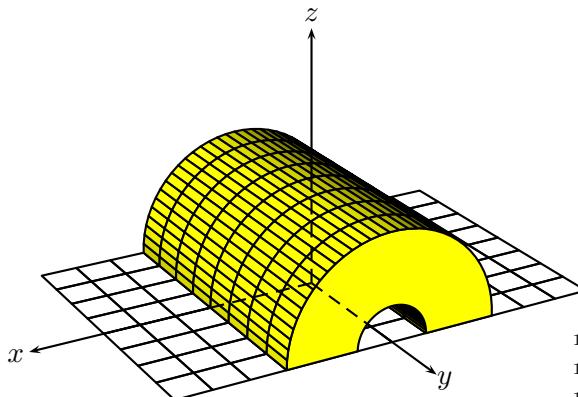
Example 1: a cylinder and a hollow cylinder



```
1 \psset{unit=0.5}
2 \psset{lightsrc=viewpoint,viewpoint=50 60 25 rtp2xyz,Decran=50}
3 \begin{pspicture}(-2,-3)(6,6)
4 \psSolid[object=cylindre,h=6,r=2,
5     fillcolor=yellow,
6     ](0,4,0)
7 \end{pspicture}
```

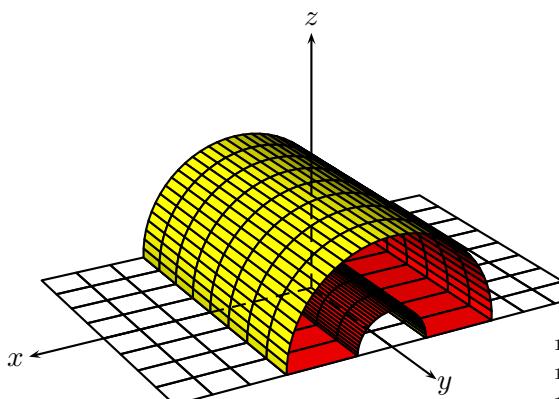


```
1 \psset{unit=0.5}
2 \psset{lightsrc=viewpoint,viewpoint=50 60 25 rtp2xyz,Decran=50}
3 \begin{pspicture}(-2,-3)(6,6)
4 \psSolid[object=cylindre,h=6,r=2,
5     fillcolor=yellow,incolor=red,
6     hollow](0,4,0)
7 \end{pspicture}
```

Example 2: a prism and a hollow prism

```

1 \psset{unit=0.5}
2 \psset{lightsrc=viewpoint,viewpoint=50 60 25
rtp2xyz,Decran=50}
3 \begin{pspicture}(-9,-4)(4,8)
4 \defFunction[F](t){t cos 3 mul}{t sin 3 mul
}{}
5 \defFunction[G](t){t cos}{t sin){}
6 \psSolid[object=grille,base=-6 6 -4 4,action=
draw]%
7 \psSolid[object=prisme,
8 h=8,fillcolor=yellow,
9 RotX=90,ngrid=8 18,
10 base=0 180 {F} CourbeR2+
11 180 0 {G} CourbeR2+](0,4,0)
12 \axesIIID(3,4,3)(8,6,7)
13 \end{pspicture}
```

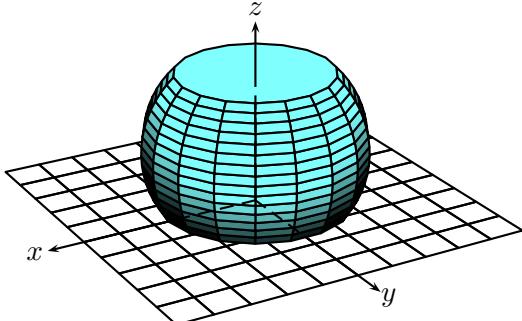


```

1 \psset{unit=0.5}
2 \psset{lightsrc=viewpoint,viewpoint=50 60 25
rtp2xyz,Decran=50}
3 \begin{pspicture}(-9,-4)(3,8)
4 \defFunction[F](t){t cos 3 mul}{t sin 3 mul
}{}
5 \defFunction[G](t){t cos}{t sin){}
6 \psSolid[object=grille,base=-6 6 -4 4,action=
draw]%
7 \psSolid[object=prisme,
8 h=8,fillcolor=yellow,incolor=red,
9 RotX=90,hollow,ngrid=8 18,
10 base=0 180 {F} CourbeR2+
11 180 0 {G} CourbeR2+](0,4,0)
12 \axesIIID(3,4,3)(8,6,7)
13 \end{pspicture}
```

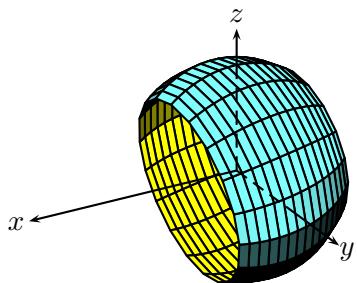
4. More options of `\psSolid`

Example 3: a spherical zone and a hollow spherical zone



```

1 \psset{unit=0.5}
2 \psset{lightsrc=10 20 30,viewpoint=50 60 25 rtp2xyz,
Decran=50}
3 \begin{pspicture}(-7,-4)(5,7)
4 \psSolid[object=grille,
5   base=-5 5 -5 5,
6   action=draw]%
7 \psSolid[object=calottesphere,
8   r=3,ngrid=16 18,
9   fillcolor=cyan!50,
10  incolor=yellow,
11  theta=45,phi=-30](0,0,1.5)%
12 \axesIIID(3,3,3.6)(6,6,5)
13 \end{pspicture}
```



```

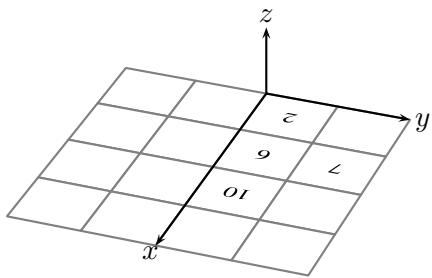
1 \psset{unit=0.5}
2 \psset{lightsrc=10 20 30,viewpoint=50 60 25 rtp2xyz,
Decran=50}
3 \begin{pspicture}(-7,-5)(7,5)
4 \psSolid[object=calottesphere,
5   r=3,ngrid=16 18,
6   fillcolor=cyan!50,
7   incolor=yellow,
8   theta=45,phi=-30,
9   hollow,
10  RotY=-80]%
11 \axesIIID(0,3,3)(6,5,4)
12 \end{pspicture}
```

4.3. Numbering of the faces

The option `numfaces` gives permission to number every face with its correspondent index number.

- `numfaces=all` all faces are numbered;
- `numfaces=0 1 2 3` only the faces that have index 0, 1, 2 and 3 are numbered.

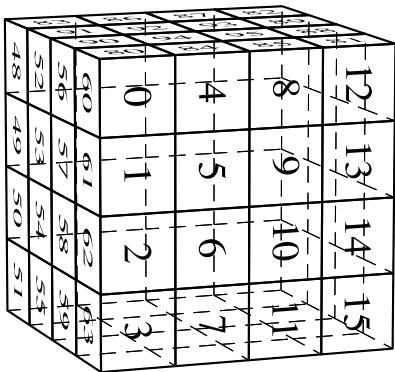
The option `fontsize` allows to fix the measurement of the used character set. Finally, the Boolean `visibility` the numbering of faces that are not visible. By default, the Boolean is set to `visibility=true`, so the visibility is set up (e. g. numbers are not set to invisible faces).



```

1 \psset{unit=1}
2 \begin{pspicture}(-4,-3)(3,1.5)
3 \psSolid[object=grille,
4   base=0 4 -2 2,
5   numfaces=2 6 7 10,
6   linecolor=gray](0,0,0)
7 \axesIIID(0,0,0)(4,2,1)
8 \end{pspicture}

```



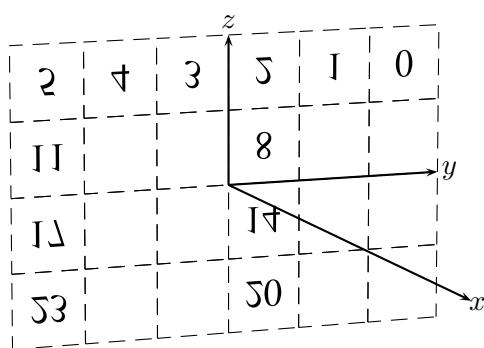
```

1 \begin{pspicture*}(-4,-3)(4,3)
2 \psSolid[object=cube,
3   RotY=90,
4   ngrid=4,
5   fontsize=15,
6   action=draw,
7   numfaces=all,](0,0,0)
8 \end{pspicture*}

```

The options of `\psSolid` accept PostScript commands, in particular the `for` loop.

With the instruction `numfaces=0 1 5 {} for` all faces with the index numbers between 0 and 5 are set up. The instruction `numfaces=8 3 23 {} for` sets up every third index number between 8 and 23.



```

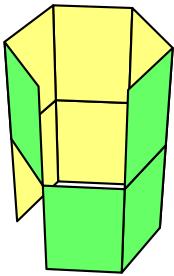
1 \begin{pspicture*}(-4,-3)(4,3)
2 \axesIIID(0,0,0)(8,3,2)
3 \psSolid[object=grille,
4   RotY=90,
5   RotZ=180,
6   ngrid=1.,
7   fontsize=15,
8   numfaces=
9     0 1 5 {} for
10    8 3 23 {} for,
11   base=-2 2 -3 3,
12   visibility=false,
13   action=draw](0,0,0)
14 \end{pspicture*}

```

4.4. Removing faces

The key value `rm=1 2 8` allows to suppress the drawing of the faces with the index numbers 1, 2 and 8, to be able to have a look inside a hollow solid.

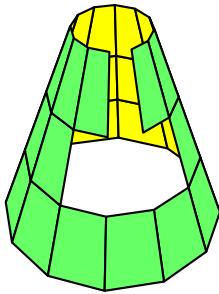
4. More options of `\psSolid`



```

1 \psset{Decran=12,grid=true,viewpoint=15 10 15}
2 \begin{pspicture}(-2.5,-2.5)(2.5,2.5)
3 \psSolid[object=cylindrecieux,
4   ngrid=2 6,
5   h=6,r=2,
6   fillcolor=green!60,
7   incolor=yellow!50,
8   RotZ=-60,
9   rm=1 3 6,](0,0,-3)
10 \end{pspicture}

```



```

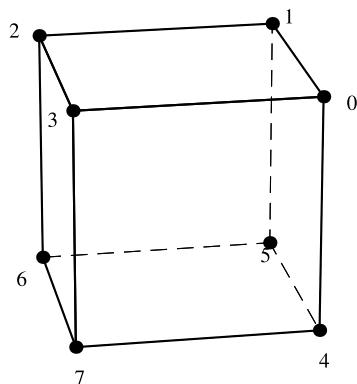
1 \psset{Decran=12,grid=true,viewpoint=15 10 15}
2 \begin{pspicture}(-2.5,-2.5)(2.5,2.5)
3 \psSolid[object=troncconecreux,
4   rm=1 12 13 14,
5   r0=3,r1=1,h=6,
6   fillcolor=green!60,
7   incolor=yellow,
8   mode=3](0,0,-3)
9 \end{pspicture}

```

4.5. Numbering of the vertices

There is an option that permits the marking of the vertices (with a black circle) and/or numbers them either globally or individually.

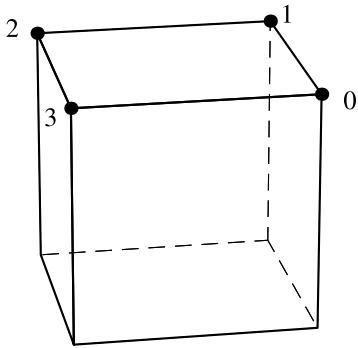
- `show=all` marks all the vertices;
- `num=all` numbers all the vertices;
- `show=0 1 2 3` marks the vertices with the index number 0, 1, 2 and 3;
- `num=0 1 2 3` numbers the vertices with the index number 0, 1, 2 and 3.



```

1 \begin{pspicture}(-3,-2.5)(7,2.5)
2 \psset{viewpoint=50 20 20 rtp2xyz,Decran=40}
3 \psSolid[
4   action=draw,
5   object=cube,
6   RotZ=30,
7   show=all,
8   num=all
9   ]%
10 \end{pspicture}

```



```

1 \begin{pspicture}(-3,-2.5)(7,2.5)
2 \psset{viewpoint=50 20 20 rtp2xyz,Decran=40}
3 \psSolid[action=draw,
4   object=cube,
5   RotZ=30,
6   show=0 1 2 3,
7   num=0 1 2 3
8   ]%
9 \end{pspicture}

```

4.6. Colours and the nuances of a colour

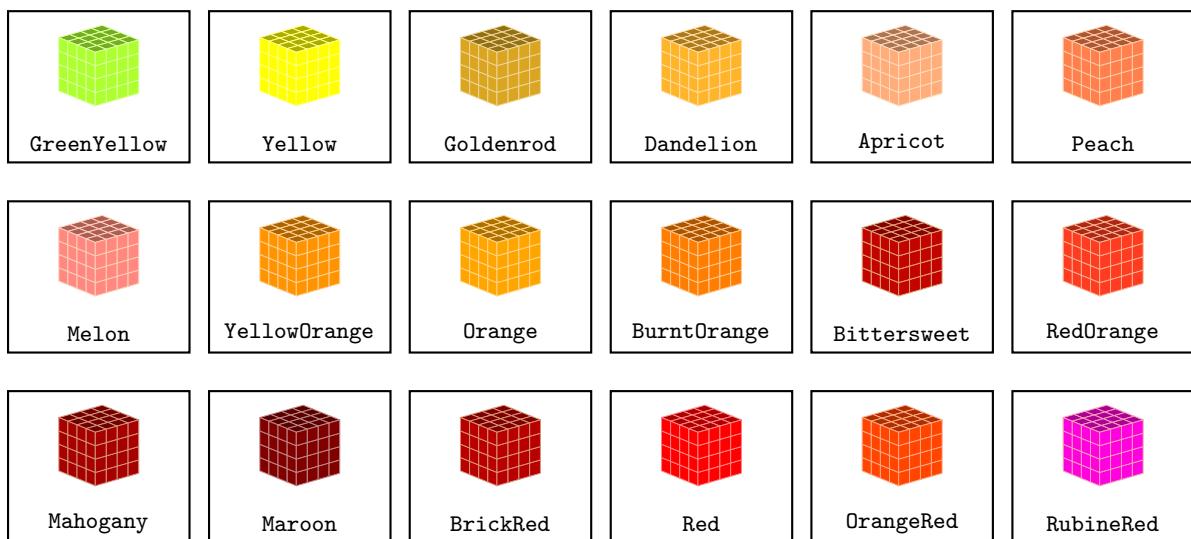
The key word `fillcolor=colourname` allows us to specify the wanted colour for the outer faces of a solid. The key word `incolor=colourname` allows us to specify the wanted colour for the inner faces of a solid.

The possible values for *name* are those known to PStricks (and particularly those of the package `xcolor`).

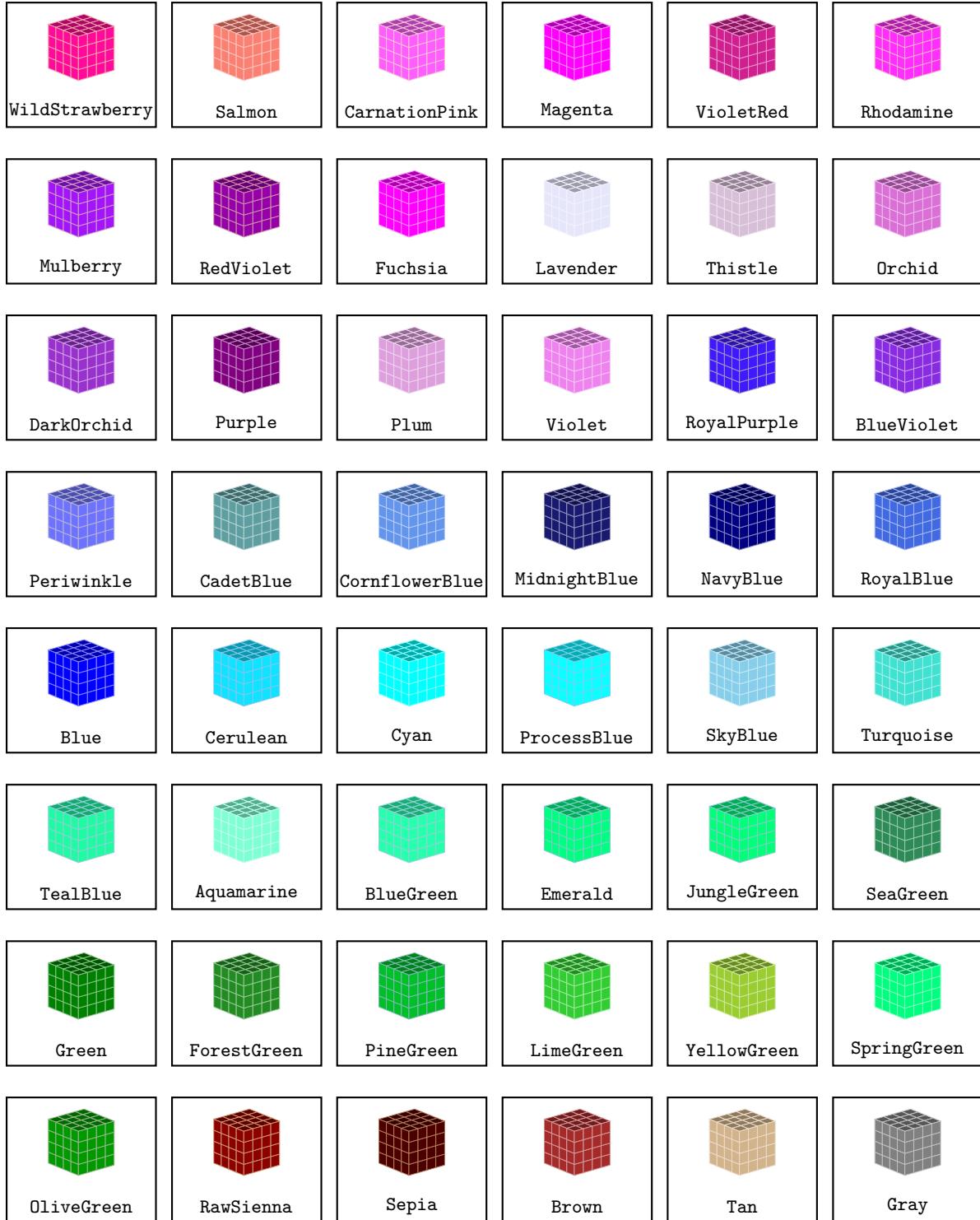
We can directly use the colour nuances in the color schemes of HSB, RGB or CMYK. In that case we use the key values `hue`, `inhue` or `inouthue` for the outer faces, the inner faces, or for all the faces. The number of arguments `hue` determines nuances.

4.6.1. Predefined colours by the option `dvipsnames`

There are 68 predefined colours, which are identified by `solides.pro`: `Black`, `White`, and the 66 colours below.



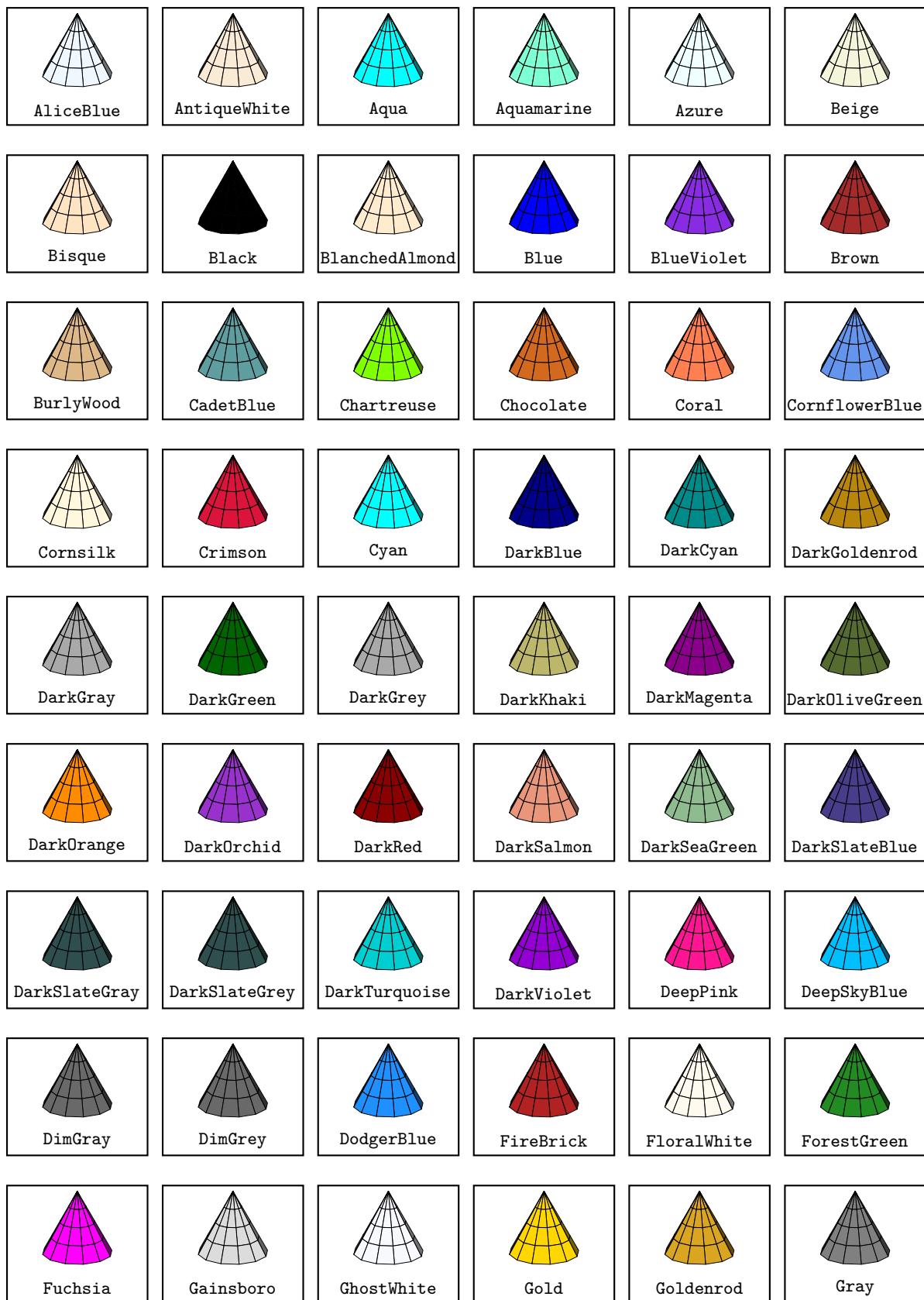
4. More options of `\psSolid`



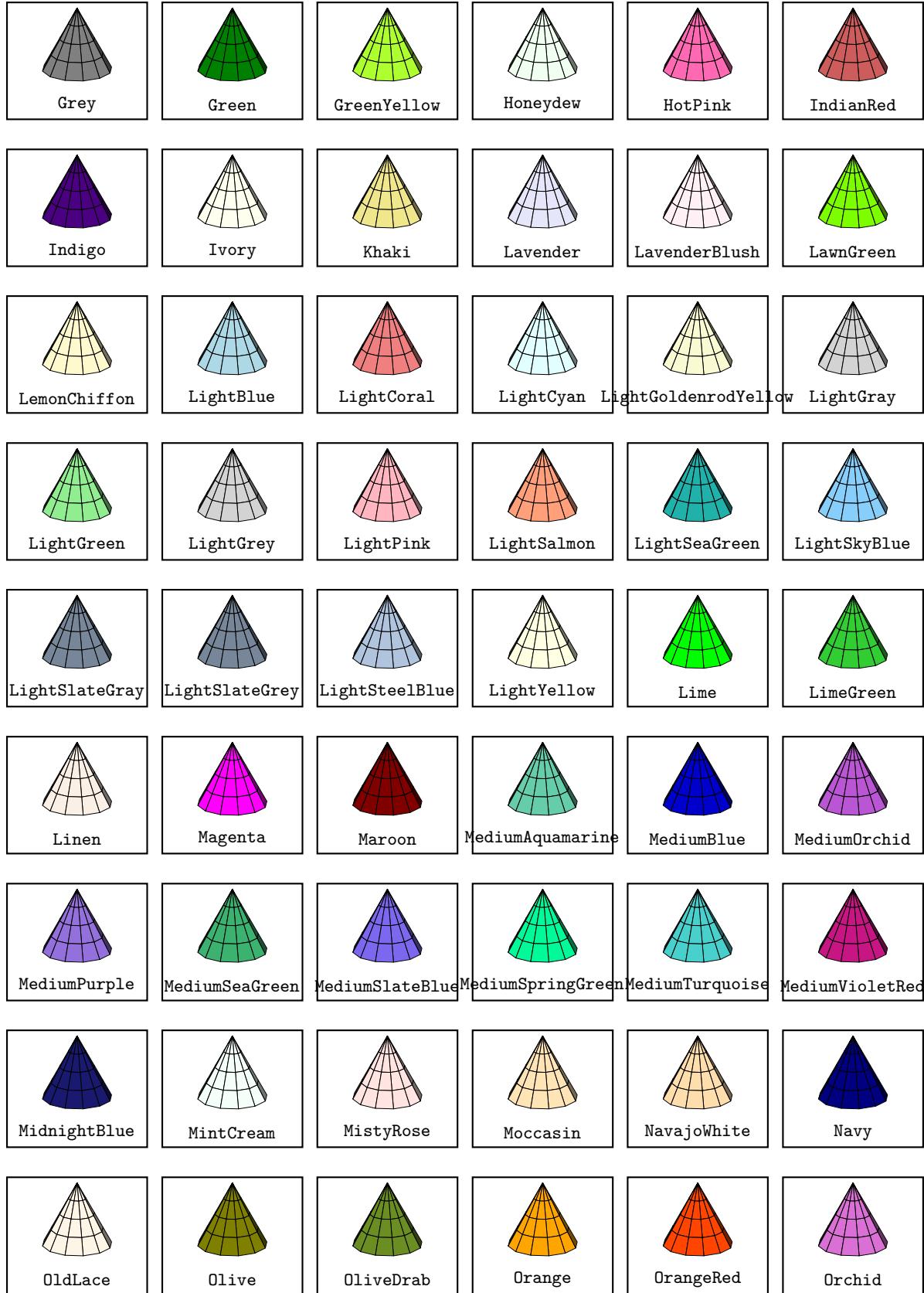
4.6.2. Predefined colours by the option `svgnames`

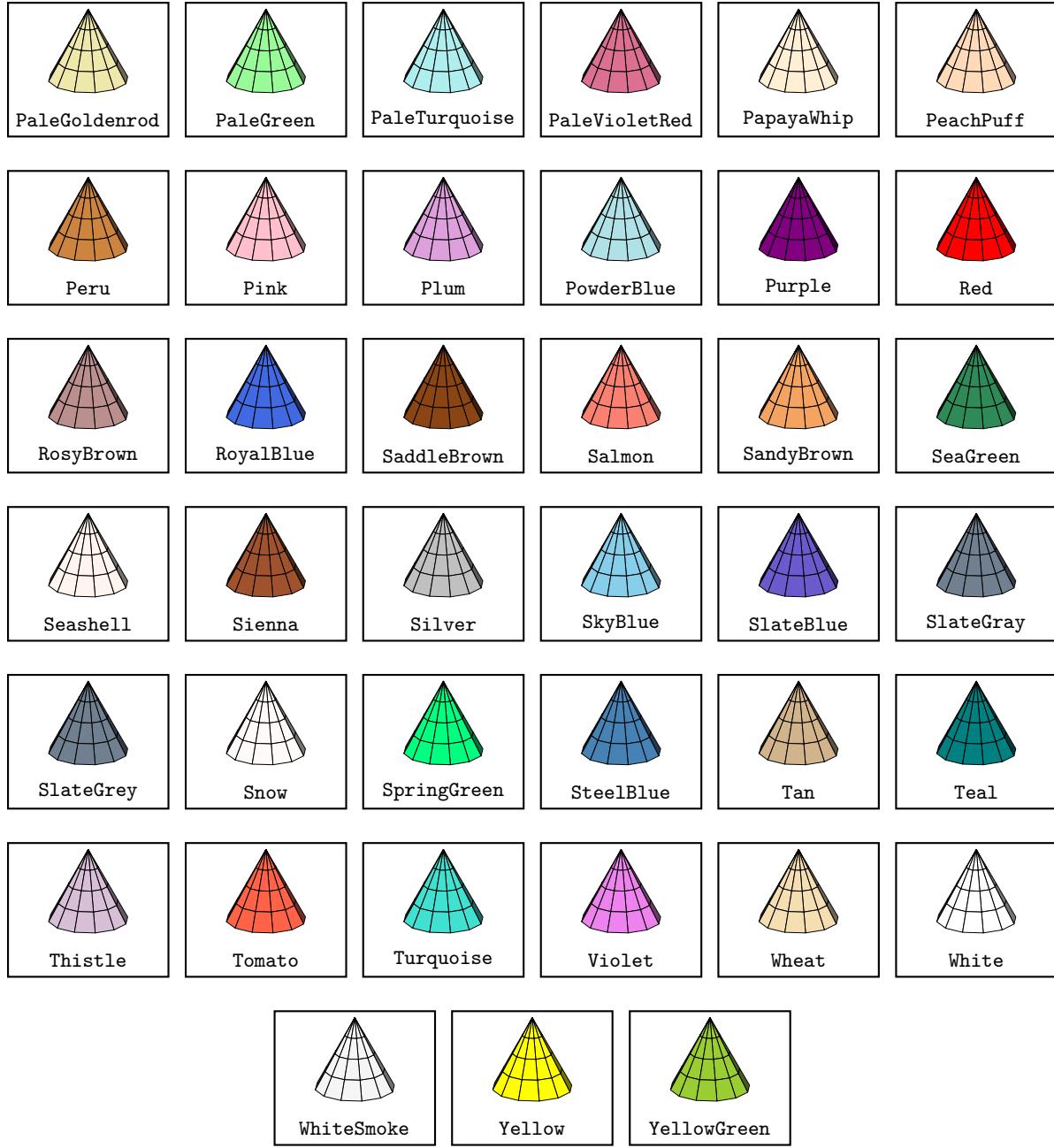
The following colours are known by PSTricks, when the option `svgnames` is given. These ones are not identified by the file `solides.pro`: we can use them directly with the option `fcol`.

These colours are delivered from the package `xcolor`.



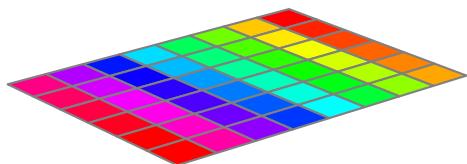
4. More options of \psSolid





4.6.3. Nuances in the colour scheme of HSB, saturation and maximum brilliance

There are 2 key values: `hue=h0 h1` where the numbers h_0 and h_1 with $0 \leq h_0 < h_1 \leq 1$ respect the limits of the colour scheme of HSB.

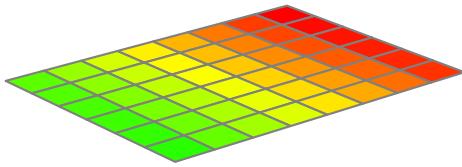


```

1 \psset{unit=1}
2 \begin{pspicture}(-4,-1.5)(3,1)
3 \psSolid[object=grille,
4   base=-3 5 -3 3,
5   linecolor=gray,
6   hue=0 1](0,0,0)
7 \end{pspicture}

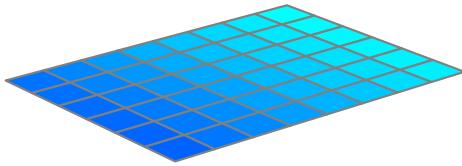
```

4. More options of `\psSolid`



```

1 \psset{unit=1}
2 \begin{pspicture}(-4,-1.5)(3,1)
3 \psSolid[object=grille,
4   base=-3 5 -3 3,
5   linecolor=gray,
6   hue=0 .3](0,0,0)
7 \end{pspicture}
```

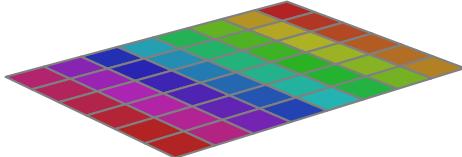


```

1 \psset{unit=1}
2 \begin{pspicture}(-4,-1.5)(3,1)
3 \psSolid[object=grille,
4   base=-3 5 -3 3,
5   linecolor=gray,
6   hue=.5 .6](0,0,0)
7 \end{pspicture}
```

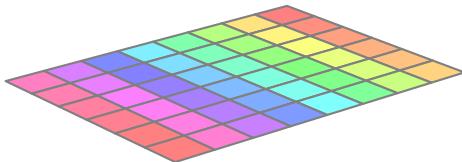
4.6.4. Nuances in the colour scheme of HSB, saturation and fixed brilliance

There are 4 key values: `hue=h0 h1 s b` or the numbers h_0 and h_1 with $0 \leq h_0 < h_1 \leq 1$ respect the limits of the colour scheme HSB and s and b are the values for **saturation** and **brilliance**.



```

1 \psset{unit=1}
2 \begin{pspicture}(-4,-1.5)(3,1)
3 \psSolid[object=grille,
4   base=-3 5 -3 3,
5   linecolor=gray,
6   hue=0 1 .8 .7](0,0,0)
7 \end{pspicture}
```

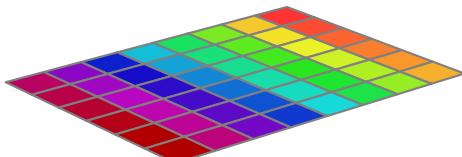


```

1 \psset{unit=1}
2 \begin{pspicture}(-4,-1.5)(3,1)
3 \psSolid[object=grille,
4   base=-3 5 -3 3,
5   linecolor=gray,
6   hue=0 1 .5 1](0,0,0)
7 \end{pspicture}
```

4.6.5. Nuances in the colour scheme of HSB, general case

There are 7 key values: `hue=h0 s0 b0 h1 s1 b1 (hsb)` or the numbers h_i , s_i and b_i respecting the limits of the parameters of HSB.

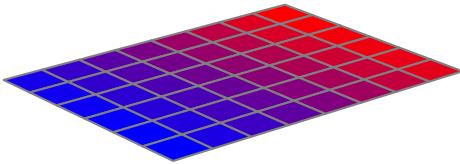


```

1 \psset{unit=1}
2 \begin{pspicture}(-4,-1.5)(3,1)
3 \psSolid[object=grille,
4   base=-3 5 -3 3,
5   linecolor=gray,
6   hue=0 .8 1 1 1 .7 (hsb)](0,0,0)
7 \end{pspicture}
```

4.6.6. Nuances in the colour scheme of RGB

There are 6 key values: `hue=r0 g0 b0` `r1 g1 b1` or the numbers r_i , g_i and b_i respecting the limits of the 3 parameters of RGB.

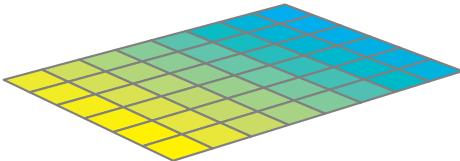


```

1 \psset{unit=1}
2 \begin{pspicture}(-4,-1.5)(3,1)
3 \psSolid[object=grille,
4   base=-3 5 -3 3,
5   linecolor=gray,
6   hue=1 0 0 0 0 1](0,0,0)
7 \end{pspicture}
```

4.6.7. Nuances in the colour scheme of CMYK

There are 8 key values: `hue=c0 m0 y0 k0` `c1 m1 y1 k1` or the numbers c_i , m_i , y_i and k_i respecting the limits of the 4 parameters of CMYK.

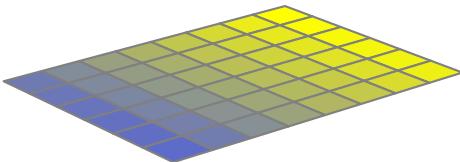


```

1 \psset{unit=1}
2 \begin{pspicture}(-4,-1.5)(3,1)
3 \psSolid[object=grille,
4   base=-3 5 -3 3,
5   linecolor=gray,
6   hue=1 0 0 0 0 0 1 0](0,0,0)
7 \end{pspicture}
```

4.6.8. Nuances between 2 named colours

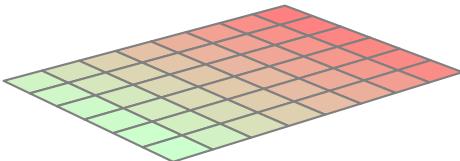
There are 2 key values `hue=(color1) (color2)` where `color1` and `color2` are the names of colours known by `solides.pro`.



```

1 \psset{unit=1}
2 \begin{pspicture}(-4,-1.5)(3,1)
3 \psSolid[object=grille,
4   base=-3 5 -3 3,
5   linecolor=gray,
6   hue=(jaune) (CadetBlue)](0,0,0)
7 \end{pspicture}
```

If we like to use some defined colours of `xcolor`, we use the key values `color1`, `color2`, etc. from `\psSolid`.



```

1 \psset{unit=1}
2 \begin{pspicture}(-4,-1.5)(3,1)
3 \psSolid[object=grille,
4   base=-3 5 -3 3,
5   linecolor=gray,
6   color1=red!50,
7   color2=green!20,
8   hue=(color1) (color2)](0,0,0)
9 \end{pspicture}
```

4. More options of `\psSolid`

4.6.9. Deactivation of the colour application

For specific purposes it is possible to disable the application of colour. This is particularly the case, when an object is already memorized or defined in external files. Within these configurations, if we do not deactivate the colours and if we do not define some new colours, these will be the colours by default that overwrite the colours that were defined.

To deactivate the colour application we use the option `deactivatecolor`.

4.7. Colouring some single faces

The key value `fcol=i0 (c0) i1 (c1) ... in (cn)`, where i_k are integers and c_k the names of the colours, permits to specify a colour for special faces. To the face with the index i_k corresponds the colour c_k . The integer n must be lower than the maximum of the number of faces of the chosen solid.

The colour names c_k , there are 68 predefined values, are defined names in the `color.pro`. These values are: 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. The list of these 68 colours is available in the command `\colorfaces` (see an example in the section about the grating of a cube).

Thinking on that case, the number of the faces $n_1 \times n_2 + 2$ (outer faces inner faces) must be lower than 68!

However users can define their own colours. There are two methods:

- They can use one of the 4 optional arguments `color1`, `color2`, `color3`, `color4` from `\psSolid`, then transmit to `fcol` a pair of the type i (`color1`), where i is the index of the chosen face. The arguments `color1`, etc. are used in the same way as the arguments from `color` and `inColor`. A possible command could be the following:

```
\psSolid[a=1,object=cube,color1=red!60!yellow!20,fcol=0 (color1)]%
```

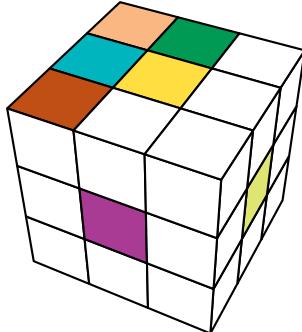
- They define their own colour names with the command `\pstVerb`, and then use these names with the argument `fcol`. For example:

```
\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
}%
And therefore:
```

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

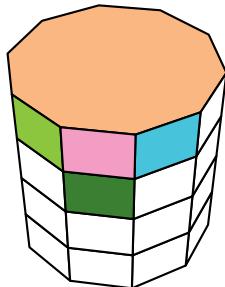
The 4 arguments `color1`, `color2`, `color3`, `color4` have default values:

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



```

1 \psset{Decran=20,viewpoint=10 5 10,unit=0.5}
2 \begin{pspicture}(-5,-5)(5,5)
3 \psSolid[
4   fcol=0 (Apricot) 1 (Aquamarine) 2 (Bittersweet)
5     3 (ForestGreen) 4 (Goldenrod)
6     13 (GreenYellow)
7     40 (Mulberry),
8   object=cube,mode=3]%
9 \end{pspicture}
```



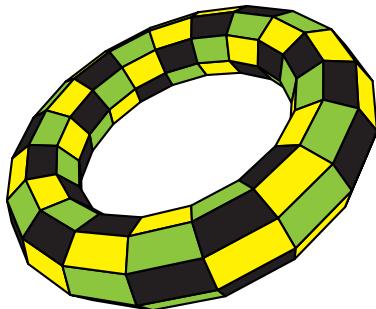
```

1 \psset{Decran=20,viewpoint=10 5 10,unit=0.5}
2 \begin{pspicture}(-5,-5)(5,5)
3 \psSolid[
4   fcol=0 (Apricot) 2 (Lavender) 3 (SkyBlue) 11 (LimeGreen) 12 (
5     OliveGreen),
6   object=cylindre,
7   h=4,
8   ngrid=4 10](0,0,-2)
9 \end{pspicture}
```

The choice of the faces to be coloured can be specified with some PostScript code,

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

which will alternately colour the faces in black, green and yellow.

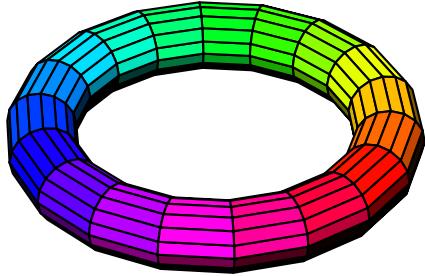


```

1 \begin{pspicture}(-3,-3)(3.5,2.5)
2 \psset{Decran=7.5,viewpoint=10 10 5}
3 \pstVerb{/iface 0 store}%
4 \psSolid[
5   fcol=48 {iface (Black)
6   iface 1 add (LimeGreen)
7   iface 2 add (Yellow) /iface
8   iface 3 add store} repeat,
9   r1=4,r0=1,
10  object=tore,
11  ngrid=8 18,
12  RotY=30]
13 \end{pspicture}
```

When the option `hue` is activated, the faces of the solid are coloured with the nuance of the rainbow colours.

4. More options of `\psSolid`



```
1 \begin{pspicture}(-3,-2.5)(3,2.5)
2 \psset[pst-solides3d]{viewpoint=50 50 50,Decran=40,lightsrc=50
20 1e2}
3 \psSolid[r1=5,r0=1,object=tore,ngrid=16 18,hue=0 1]%
4 \end{pspicture}
```

4.8. Nuances of transparency

The key value `opacity=k` with $k \in \mathbb{R}$ and $0 \leq k \leq 1$, allows you to define the level of opacity.

Within *jps code*, we use an equivalent expression `k setfillopacity`. The last expression finds its application in the option `fcol`. For example the instruction, `fcol=0 (.5 setfillopacity yellow)`, which defines the face with the index number 0, sets it to yellow with an opacity of 50%.

4.9. Definition of grating

The user can specify the grating of the solid with the option `ngrid` within the command `\psSolid`.

For the objects `cube`, `prisme`, `prismecreux`, the syntax is `ngrid=n1` where n_1 represents the number of vertical gridlines.

For the objects `cylindre`, `cylindrecréux`, `cone`, `conecréux`, `tronccone`, `tronconeconcreux`, the syntax is `ngrid=n1 n2` where n_1 is an integer greater or equal to 1 (2 for `tore`) representing the number of the vertical gridlines, and n_2 is an integer representing the number of divisions on the circle.

For the object `sphere`, the syntax is `ngrid=n1 n2` where n_1 is an integer, representing the number of divisions on the vertical axis, and n_2 is an integer representing the number of divisions on the circle horizontally.

For the object `tore`, the syntax is `ngrid=n1 n2` where n_1 and n_2 are integers.

Here are some examples:

4.9.1. The cube



For the first example, the grid is fixed to 4×4 facettes/faces and the command is the following:

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

In the second example, the face grid is set to 3×3 and the colours of the faces are different. We use the package `arrayjob` to easily save the colours:

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

The list of the colours is given by the command:

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

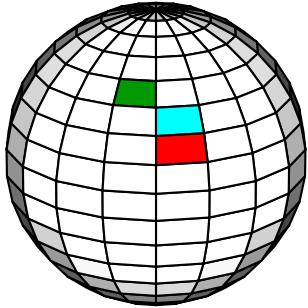
One sets up: `fcol=\colorfaces`. The gridded cube now is called with:

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

The option `grid` suppresses the drawing of the gridlines.

4.9.2. Sphere

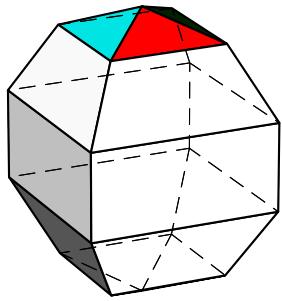
4. More options of `\psSolid`



```

1 \begin{pspicture}(-3,-3)(3,3)
2 \psset{viewpoint=50 50 20 rtp2xyz,Decran=50,lightsrc=viewpoint
3 }
4 \psset{color1=cyan,color2=red}
5 \psSolid[
6   fcol=251 (OliveGreen) 232 (color1) 214 (color2),
7   object=sphere,
8   ngrid=16 18,
9   RotX=180,RotZ=30]%
9 \end{pspicture}

```

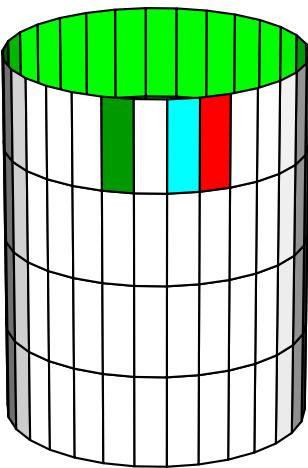


```

1 \begin{pspicture}(-3,-3)(3,3)
2 \psset{viewpoint=50 50 20 rtp2xyz,Decran=50,lightsrc=viewpoint
3 }
4 \psset{color1=cyan,color2=red}
5 \psSolid[
6   action=draw*,
7   fcol=0 (OliveGreen) 2 (color1) 3 (color2),
8   object=sphere,
9   ngrid=4 4,
9   RotX=180,RotZ=30]%
10 \end{pspicture}

```

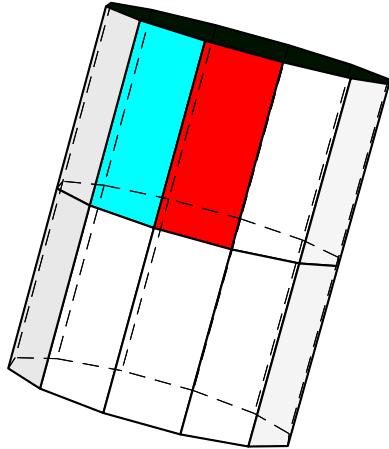
4.9.3. Cylinders



```

1
2 \begin{pspicture}(-3,-4)(3,4)
3 \psset{viewpoint=50 50 20 rtp2xyz,Decran=50,lightsrc=viewpoint
3 }
4 \psset{color1=cyan,color2=red}
5 \psSolid[
6   fcol=0 (OliveGreen) 2 (color1) 3 (color2),
7   h=5,r=2,
8   object=cylindrecieux,
9   ngrid=4 30,
10  RotZ=30
11 ](0,0,-2.5)
12 \end{pspicture}

```

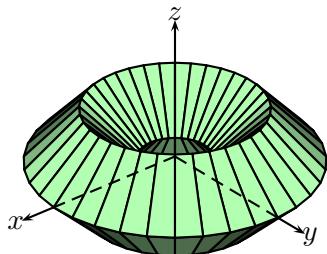


```

1 \begin{pspicture}(-3,-4)(4,4)
2 \psset{viewpoint=50 50 20 rtp2xyz,Decran=50,lightsrc=
      viewpoint}
3 \psset{color1=cyan,color2=red}
4 \psSolid[
5   action=draw*,
6   fcol=0 (OliveGreen) 2 (color1) 3 (color2),
7   h=5,r=2,
8   object=cylindre,
9   ngrid=2 12,
10  RotY=-20
11 ](0,0,-2.5)
12 \end{pspicture}

```

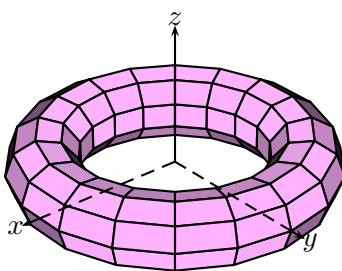
4.9.4. Torus



```

1 \begin{pspicture}(-3,-2)(3,2)
2 \psset{viewpoint=50 50 30 rtp2xyz,Decran=25,lightsrc=viewpoint
      }
3 \psSolid[r1=2.5,r0=1.5,
4   object=tore,
5   ngrid=4 36,
6   fillcolor=green!30,
7   action=draw**]%
8 \axesIIID(4,4,0)(5,5,4)
9 \end{pspicture}

```



```

1 \begin{pspicture}(-3,-2)(3,2)
2 \psset{viewpoint=50 50 30 rtp2xyz,Decran=25,lightsrc=viewpoint
      }
3 \psSolid[r1=3.5,r0=1,
4   object=tore,
5   ngrid=9 18,
6   fillcolor=magenta!30,
7   action=draw**]%
8 \axesIIID(4.5,4.5,0)(5,5,4)
9 \end{pspicture}

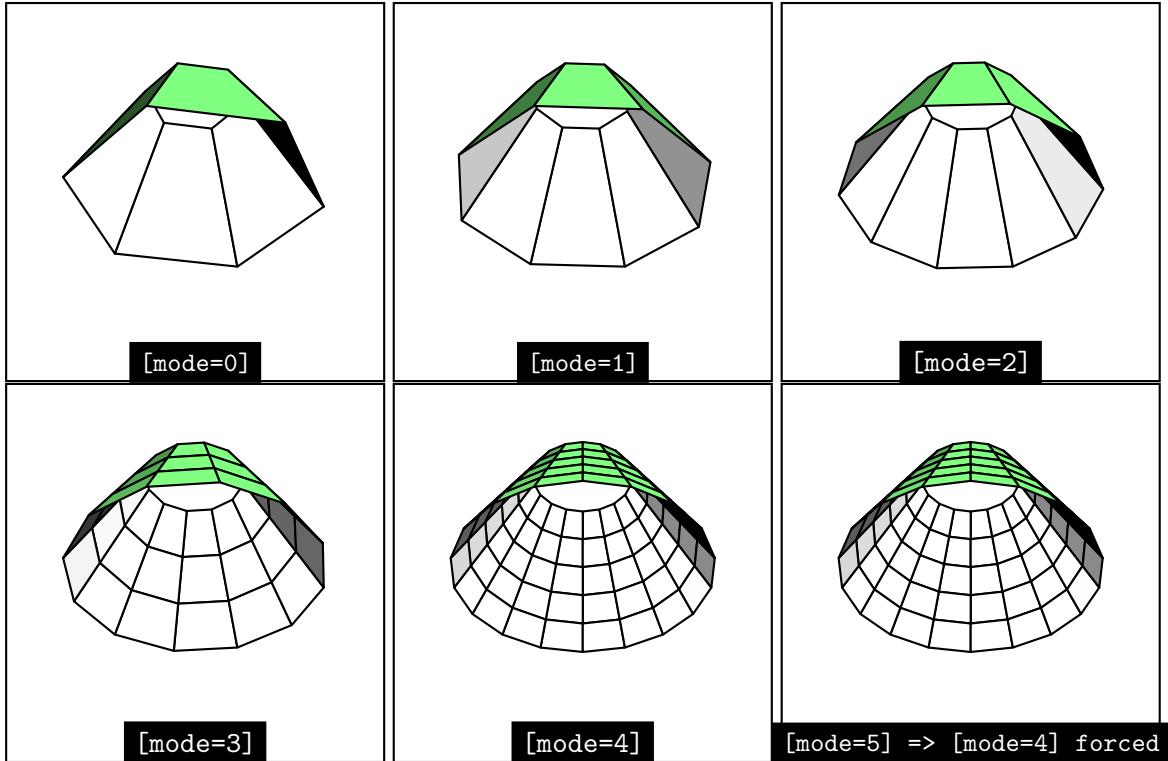
```

4.10. The modes

For some solids, there are certain gratings predefined. We can setup the key values to `mode=0, 1, 2, 3 or 4` which allows to have some some gratings from very coarse `mode=0` up to very fine `mode=4`.

This permits us to have a draft version of a solid with `mode=0` (fewer calculations) and then refine it with `mode=4` for the final version.

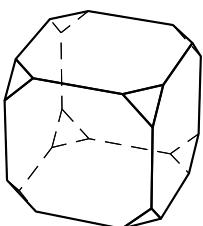
4. More options of `\psSolid`



4.11. Truncate a solid's vertices

The option `trunc` allows us to truncate a solid's vertices either globally or individually. This option uses the key `trunccoeff` (value 0.25 by default) which indicates the ratio k used for the truncation ($0 < k \leq 0.5$).

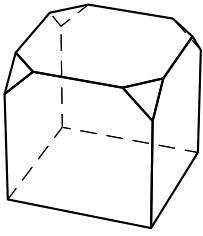
- `trunc=all` truncates all the vertices;
- `trunc=0 1 2 3` truncates the vertices 0, 1, 2 and 3.



```

1 \psset{viewpoint=50 50 30 rtp2xyz,Decran=25,lightsrc=viewpoint
        }
2 \begin{pspicture}(-3,-2)(2,2)
3 \psSolid[
4   action=draw,
5   object=cube,
6   RotZ=30,
7   trunccoeff=.2,
8   trunc=all]\%
9 \end{pspicture}

```



```

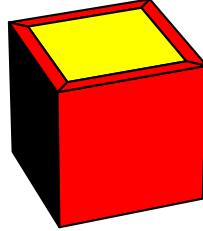
1 \psset{viewpoint=50 50 30 rtp2xyz,Decran=25,lightsrc=viewpoint
        }
2 \begin{pspicture}(-3,-2)(2,2)
3 \psSolid[action=draw,
4   object=cube,
5   RotZ=30,
6   trunccoeff=.2,
7   trunc=0 1 2 3]%
8 \end{pspicture}

```

4.12. Hollowing out a solid's faces

We call *hollowing by the ratio k* an operation, which for a given face with the center G , executes a dilation on that face with the ratio k , then divides the original face with using this new face.

For example, a cube with a hollow of its top face with a ratio of 0.8:



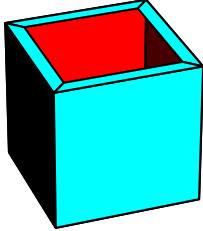
The option **affinage** allows us to hollow a solid's faces either globally or individually. This option uses the key **affinagecoeff** (value 0.8 by default) which indicates the ratio k used for the hollow ($0 < k < 1$).

- **affinage=all** hollows all the faces;
- **affinage=0 1 2 3** hollows the faces 0, 1, 2 and 3;

When a face is hollowed out, the default behaviour suppresses the resulting central face. However, the option **affinagerm** allows us to conserve that central face.

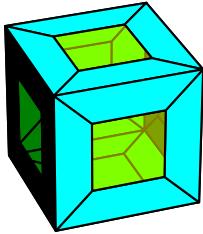
When we conserve the centre face, it is—by default—drawn with the same colour as the original. The option **fcolor** permits to specify another colour.

4. More options of `\psSolid`



```

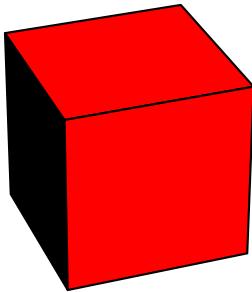
1 \psset{unit=0.5}
2 \begin{pspicture*}(-5,-4)(6,5)
3 \psSolid[object=cube,
4   fillcolor=cyan,
5   incolor=red,
6   hollow,
7   affinage=0]
8 \end{pspicture*}
```



```

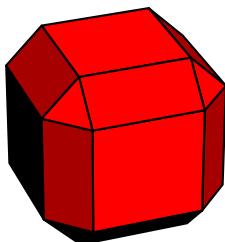
1 \psset{unit=0.5}
2 \begin{pspicture*}(-5,-4)(6,5)
3 \psSolid[object=cube,
4   fillcolor=cyan,
5   affinagecoeff=.5,
6   affinagerm,
7   fcolor=.5 setfillopacity Yellow,
8   hollow,
9   affinage=all]
10 \end{pspicture*}
```

4.13. Chamfering a solid



```

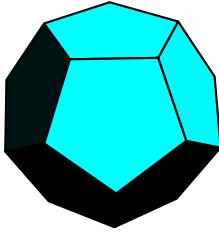
1 \psset{unit=0.5}
2 \begin{pspicture*}(-4,-4)(4,4)
3 \psSolid[object=cube,
4   a=5,
5   fillcolor=red]
6 \end{pspicture*}
```



```

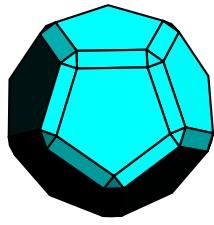
1 \psset{unit=0.5}
2 \begin{pspicture*}(-4,-4)(4,4)
3 \psSolid[object=cube,
4   a=5,
5   fillcolor=red,
6   chanfrein,
7   chanfreincoeff=.6]
8 \end{pspicture*}
```

The option `chanfrein` allows us to chamfer a solid. This option uses the key `chanfreincoeff` (value 0.8 by default) which indicates the ratio k with $(0 < k < 1)$. This ratio is the one of a centre dilation with the centre in the middle of the chosen face.



```

1 \psset{unit=0.5}
2 \begin{pspicture*}(-4,-4)(4,4)
3 \psSolid[object=dodecahedron,
4   a=5,
5   fillcolor=cyan]
6 \end{pspicture*}
```



```

1 \psset{unit=0.5}
2 \begin{pspicture*}(-4,-4)(4,4)
3 \psSolid[object=dodecahedron,
4   a=5,
5   fillcolor=cyan,
6   chanfrein,
7   chanfreincoeff=.8]
8 \end{pspicture*}
```

4.14. The option `transform`

The option `transform`, which is nothing else than a formula $\mathbb{R}^3 \rightarrow \mathbb{R}^3$, which is applied to every point of the solid. In the first example, the object that accepts the transformation is a cube. The referenced cube is yellow, the transformed cube is green and the cube before the transformation is setup with a reticule.

4.14.1. Identical scaling factor in the three coordinates

The scaling factor is set to 0.5. It is either introduced within the PostScript variable ‘/Facteur’:

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

and then passed to the option `transform`:

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

or directly passed to the option:

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

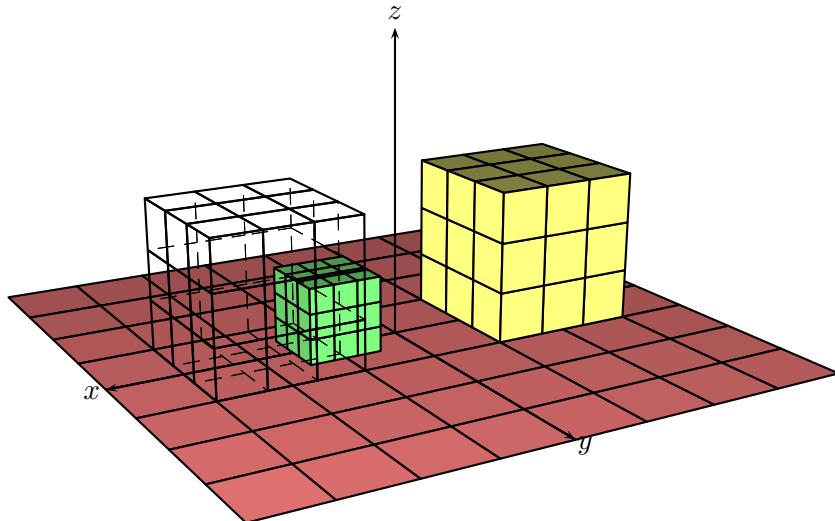
Here the *jps* abbreviation `transform={.5 mulv3d}` for a function $\mathbb{R}^3 \rightarrow \mathbb{R}^3$ was used.

Another method would be to use the code

4. More options of `\psSolid`

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

and then pass it to the option `transform=mattransformation`.

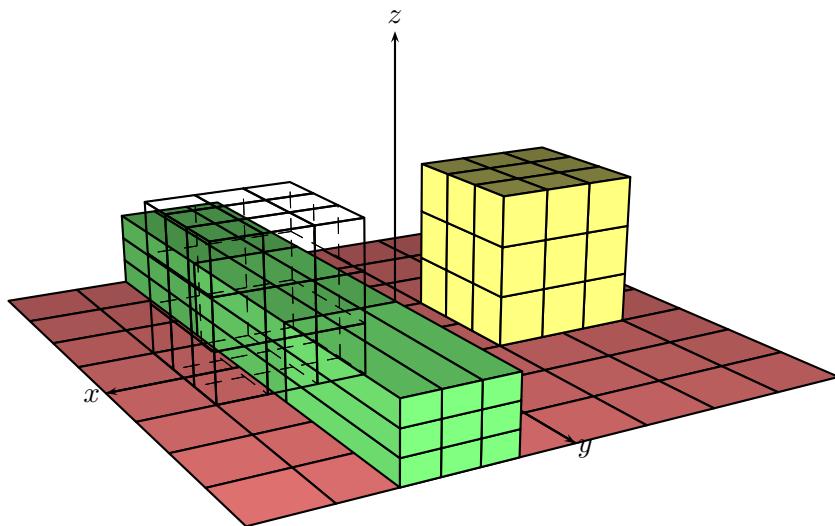


```
1 \psset{viewpoint=20 60 20 rtp2xyz,lightsrc=viewpoint,Decran=20}
2 \begin{pspicture}(-5,-3)(6,5)
3 \psSolid[object=grille,base=-4 4 -4 4,fillcolor=red!50]%
4 \axesIIID(0,0,0)(4,4,4)%
5 \psSolid[object=cube,fillcolor=yellow!50,
6     a=2,ngrid=3](-2,0,1)
7 \psSolid[object=cube,fillcolor=green!50,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

Note: The scaling factor also affects the position coordinates of the cube's center.

4.14.2. Different scaling factors for the three coordinates

Let's for example use a factor 0.75 for x , 4 for y and 0.5 for z using the function `scale0point3d` from the *jps* library—so a cube will be transformed to a cuboid.



```

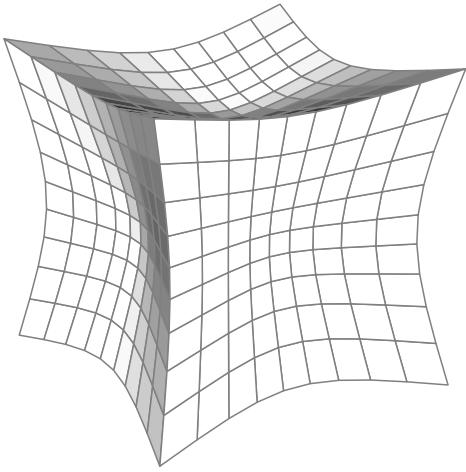
1 \psset{viewpoint=20 60 20 rtp2xyz,lightsrc=viewpoint,Decran=20}
2 \begin{pspicture}(-5,-3)(6,5)
3 \psSolid[object=grille,base=-4 4 -4 4,fillcolor=red!50]%
4 \axesIIID(0,0,0)(4,4,4)%
5 \psSolid[object=cube,fillcolor=yellow!50,
6     a=2,ngrid=3](-2,0,1)
7 \psSolid[object=cube,fillcolor=green!50,
8 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

4.14.3. Transformation associated with the distance to the origin

Here an example applied to a cube:

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

4. More options of `\psSolid`



```

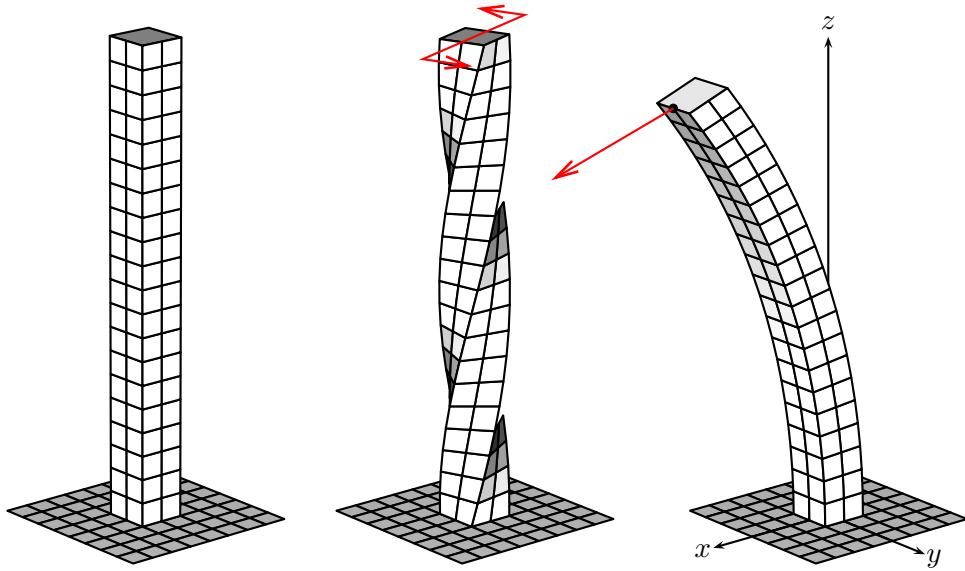
1 \begin{pspicture}(-3,-4)(4,3)
2 \psset{viewpoint=20 60 20 rtp2xyz,lightsrc=10 15 7,
         Decran=20}
3 \pstVerb{
4 /gro {
5 4 dict begin
6   /M defpoint3d
7   /a .5 def
8   /b 1 a 3 sqrt mul sub def
9   /k M norme3d a mul b add def
10  M k mulv3d
11 end
12 } def}%
13 \psset{linewidth=.02, linecolor=gray}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

```

4.14.4. Bending and torsion of beams

The solid to the left is a prism of the height 10 cm with 20 floors (`ngrid=20 2`). In every floor, an additional angle of rotation—for example 10° around the Oz axis—is given. Now that the adjacent floors have a distance of 0.5 cm, one multiplies $z \times 20$.

La flexion est envisagée dans le plan xOz sous l'action d'une force perpendiculaire à la poutre appliquée en son extrémité.



```

1 \psset{viewpoint=100 50 20 rtp2xyz,lightsrc=viewpoint,Decran=100,unit=0.65}
2 \begin{pspicture}(-3,-1)(3.5,11)
3 \psSolid[object=grille,base=-2 2 -2 2,ngrid=8]%
4 \psSolid[object=prisme,h=10,ngrid=20 2,
5      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]%
6 \end{pspicture}
7 \begin{pspicture}(-3,-1)(3.5,11)
8 \psSolid[object=grille,base=-2 2 -2 2,ngrid=8]%
9 \pstVerb{
10 /torsion {\% on tourne de 10 degre's suivant l'axe Oz `a chaque niveau
11 2 dict begin
12   /M defpoint3d \% on r\`e cup're les coordonn\`ees
13   M /z exch def pop pop
14   M 0 0 z 20 mul rotate0point3d
15 end} def}%
16 \psSolid[object=prisme,h=10,ngrid=20 2,
17      base=0.5 0 0.5 0 0.5 -0.5 0.5 -0.5 0 -0.5 -0.5 0 -0.5 0.5 -0.5,
18      transform=torsion]%
19 \psTransformPoint [RotZ=20](2 0 10)(0,0,0){A}
20 \psTransformPoint [RotZ=20](2 1 10)(0,0,0){A'}
21 \psTransformPoint [RotZ=20](-2 0 10)(0,0,0){B}
22 \psTransformPoint [RotZ=20](-2 -1 10)(0,0,0){B'}
23 \psline[linecolor=red]{v-v}(A')(A)(B)(B')
24 \end{pspicture}
25 \begin{pspicture}(-3.5,-1)(3,11)
26 \psSolid[object=grille,base=-2 2 -2 2,ngrid=8]%
27 \pstVerb{\% id\`e de Christophe Poulain
28 /flexion {\% on tourne de 2 degre's suivant l'axe Oy `a chaque niveau
29 2 dict begin
30   /M defpoint3d \% on r\`e cup're les coordonn\`ees
31   M /z exch def pop pop
32   M 0 z 2 mul 0 rotate0point3d
33 end} def}%
34 \axesIID(0,0,0)(3,3,10)
35 \psSolid[object=prisme,h=10,ngrid=20 2,
36      base=0.5 0 0.5 0 0.5 -0.5 0.5 -0.5 0 -0.5 -0.5 0 -0.5 0.5 -0.5,
37      transform=flexion]%
38 \psTransformPoint [RotY=20](0.5 0 10)(0,0,0){A}
39 \psPoint(3 20 cos mul 20 sin 10 mul add 0.5 add,0, 20 cos 10 mul 20 sin 3 mul sub){A'}
40 \psdot(A)\psline[linecolor=red]{-v}(A)(A')
41 \end{pspicture}

```

4.15. Lines of intersecting planes

For every object of the type `\psSolid`, it is possible to draw the lines of intersection between a chosen solid and one or more planes.

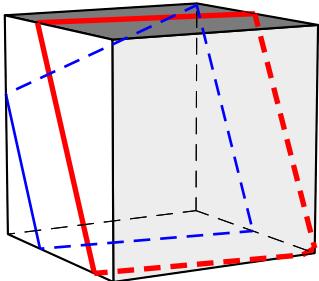
The numeric argument `intersectiontype=k` (value -1 by default) determines whether or not to draw the intersection lines. Set to 0 , the intersection lines are drawn.

There are three keys to be handled:

- `intersectionplan={[eq1] ... [eqn]}` defines a list of the equations eq_i of the intersecting planes. The eq_i could as well be some objects from the type `plan` (see the related section).

$ax + by + cz + d = 0$ that would deliver $[a\ b\ c\ d]$ as one of the n equations

- `intersectionlinewidth=w1 ... wn` defines a list of the thickness in picas w_i for each of the intersection lines.
- `intersectioncolor=color1 ... colorn` defines a list for the colors of the intersection lines.



```

1 \psset{lightsrc=20 -20 10,viewpoint=50 -20 10 rtp2xyz,Decran
         =50}
2 \psset{unit=0.5}
3 \begin{pspicture}*(-5,-4)(5,5)
4 \psSolid[object=cube,
5   intersectiontype=0,
6   intersectionplan={[1 0 .5 2] [1 0 .5 -1]},
7   intersectionlinewidth=1 2,
8   intersectioncolor=(bleu) (rouge),
9   RotX=20,RotY=90,RotZ=30,
10  a=6,
11  action=draw*]
12 \end{pspicture}

```

CHAPTER 5

Usage of external files

5.1. Using the data file types .obj and .off

Sometimes it will be helpful to use external files, either for reading or writing. When there is a solid which needs a long time to be calculated and which will be tested in different views or different colors, it is very interesting to save them externally and then only reread them by avoiding the time expensive recalculations. In particular, this technique is often used to generate some animations. One can also export a solid by that method to reuse with another software.

For **pst-solides3d**, all the procedures of reading/writing are delegated to the PostScript interpreter (and not to **TeX** or **L^AT_EX**). Consequently it is not the **L^AT_EX** compilation that will cause the execution of reading/writing, but the visualisation of the PostScript file that is produced.

Generally the reading of external files by a PostScript interpreter doesn't cause any trouble normally. The writing of files however, can cause some security problems and it is often the case that the PostScript viewer forbids the writing by default. So the system must be configured to get authorisation for that writing.

Note: By default, under Windows and Linux, the security of files on the hard drive is activated and doesn't allow to write on the drive. To deactivate that security option, more or less temporarily, here the two corresponding procedures:

Linux: The advice from Jean-Michel SARLAT: the simplest will be to use GhostScript directly, within the console. As there is no image to wait for:

```
$> gs -dNOSAFER monfichier.ps quit.ps
```

Windows: Within the menu **Options**, the option **Security of files** must be turned to unchecked.

5. Usage of external files

5.1.1. .dat files (specific to `pst-solides3d`)

In `pst-solides3d`, the data structure used for a solid has 4 fields. It can be stored in a set of 4 `.dat` files.

Writing .dat files

One uses the action `writesolid` within `\psSolid`, and one uses the option `file` to specify the name of the file.

For example, let's look at the code below:

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

The command chain `LaTeX->dvips->GSview (Windows) or gv (Linux)` first compiles, then transforms into PostScript to finally get visualised.

That last operation creates 4 files:

- `montore-sommets.dat` → the list of the vertices;
- `montore-faces.dat` → the list of the faces;
- `montore-couleurs.dat` → the colors of the faces;
- `montore-io.dat` → the limits of the indices of the external and internal faces.

Note: All these four files will automatically be saved within the same folder as the generating file.

Reading .dat files

We use the object `datfile` of `\psSolid`, with the argument `file` to specify the name. Now the code

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

will allow us to use the object—now saved in the `.dat` files generated—as described in the previous paragraph.

5.1.2. .obj files

We use only a simplified form of the `.obj` format. In particular, the files should not contain a character like `#` (the character for a comment in that format).

This format just uses a single file and permits within this file to specify the vertices and the faces.

Writing .obj files

One uses the action `writeobj` in `\psSolid`, and one uses the option `file` to specify the name of the file.

For example, the code below:

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

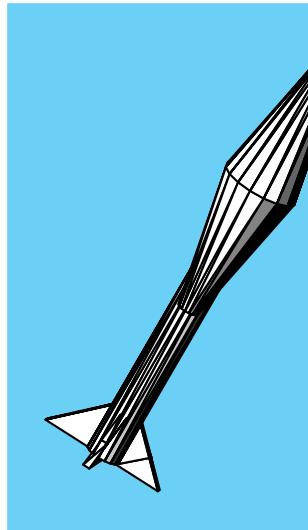
will produce a single file `montore.obj` (after compilation and visualisation of the `.ps` that was produced).

Reading .obj files

One uses the option `objfile` of `\psSolid`, with the argument `file` to specify the name of the file. Now the following code

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

will allow to use the object—now saved in the `.obj` file generated—as described in the previous paragraph.



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

5.1.3. .off files

We use only a simplified form of the `.off` format. In particular, these files only comprise `v` and `f` entries.

This format just uses a single file and permits within this file to specify the vertices and the faces.

5. Usage of external files

Writing .off files

We use the action `writeobj` in `\psSolid`, and we use the option `file` to specify the name of the file.

For example the code below:

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

will produce the `montore.off` file (after compilation and visualisation of the `.ps` that was produced).

Reading.off files

We use the option `offfile` of `\psSolid`, with the argument `file` to specify the name of the file. Now the following code

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

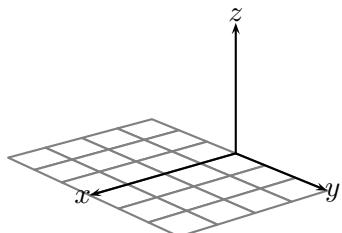
will allow to use the object—now saved in the `.off` file generated—like described in the previous paragraph.

CHAPTER 6

Some special objects

6.1. The grid

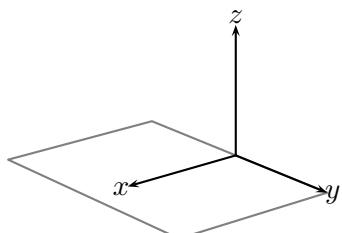
The object `grille` allows you to obtain a solid plane. The key `[base=xmin xmax ymin ymax]` lets you specify the dimension of the grid.



```
1 \begin{pspicture}(-3.5,-1.5)(3.5,2.5)
2 \psSolid[object=grille,
3   base=0 4 -3 3,
4   linecolor=gray](0,0,0)
5 \axesIIID(0,0,0)(4,3,3)
6 \end{pspicture}
```

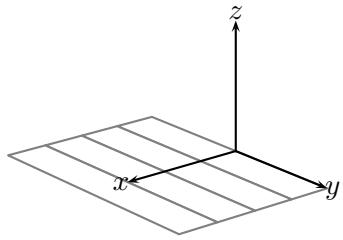
The key `[ngrid=n1 n2]` lets you specify fineness of the grid. If n_2 is not set up, it is considered that $n_2 = n_1$.

If n_1 is an integer, it represents the number of grid points along the Ox axis. If it is a real, it represents the step size along the Ox axis. For example, the number 1 is an integer, the number 1. is real (note the decimal point).



```
1 \begin{pspicture}(-3.5,-1.5)(3.5,2.5)
2 \psSolid[object=grille,
3   ngrid=1,
4   base=0 4 -3 3,
5   linecolor=gray](0,0,0)
6 \axesIIID(0,0,0)(3,3,3)
7 \end{pspicture}
```

6. Some special objects



```

1 \begin{pspicture}(-3.5,-1.5)(3.5,2.5)
2 \psSolid[object=grille,
3   ngrid=1. 1,
4   base=0 4 -3 3,
5   linecolor=gray](0,0,0)
6 \axesIIID(0,0,0)(3,3,3)
7 \end{pspicture}

```

6.2. The object point

6.2.1. Definition via coordinates

The object **point** defines a point. The simplest method is to use the argument `args=x y z` to specify its coordinates. If we have already named a point $M(x, y, z)$ (see chapter “Advanced usage”), we can easily use the argument `args=M`.

6.2.2. Some other definitions

There are some other possibilities for defining a point. Here a list of possible definitions with the appropriate arguments:

- `definition=solidgetsommet; args= solid k.`

The vertex with index k of the solid *solid*.

- `definition=solidcentreface; args=solid k.`

The centre of the face with index k of the solid *solid*.

- `definition=isobarycentre3d; args={[A0 ... An]}.`

The isobarycentre of the system $[(A_0, 1); \dots; (A_n, 1)]$.

- `definition=barycentre3d; args= {[A a B b] }.`

The barycentre of the system $[(A, a); (B, b)]$.

- `definition=hompoint3d; args=M A α.`

The image of M via a homothety with centre A and ratio α .

- `definition=sympoint3d; args= M A.`

The image of M via the center of symmetry A

- **definition=translatepoint3d; args= M u.**

The image of M under the translation via the vector \vec{u}

- **definition=scale0point3d; args= x y z k1 k2 k3.**

This gives a “dilation” of the coordinates of the point $M(x, y, z)$ on the axes Ox , Oy and Oz each multiplied by an appropriate factor k_1 , k_2 and k_3

- **definition=rotate0point3d; args= M alpha_x alpha_y alpha_z.**

The image of M through consecutive rotations—centered at O —and with respective angles α_x , α_y and α_z around the axes Ox , Oy and Oz .

- **definition=orthoprojplane3d; args= M A v.**

The projection of the point M to the plane P which is defined by the point A and the vector \vec{v} , perpendicular to P .

- **definition=milieu3d; args= A B.**

The midpoint of $[AB]$

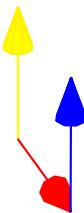
- **definition=addv3d; args= A u.**

Gives the point B so that $\overrightarrow{AB} = \vec{u}$

6.3. The object vecteur

6.3.1. Definition with components

The object **vecteur** allows us to define a vector. The simplest way to do that is to use the argument **args=x y z** to specify its components.



```

1 \begin{pspicture*}(-1,-1)(1,2)
2 \psSolid[object=vecteur,
3   action=draw*,
4   args=0 0 1,
5   linecolor=yellow]%
6 \psSolid[object=vecteur,
7   args=1 0 0,
8   linecolor=red]
9 \psSolid[object=vecteur,
10  args=0 0 1,
11  linecolor=blue](1,0,0)
12 \end{pspicture*}

```

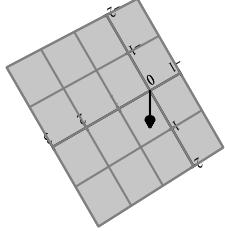
6. Some special objects

6.3.2. Definition with 2 points

We can also define a vector with 2 given points A and B of \mathbb{R}^3 .

We then use the arguments `definition=vecteur3d` and `args=xA yA zA xB yB zB` where (x_A, y_A, z_A) and (x_B, y_B, z_B) are the appropriate coordinates of the points A and B

If the points A and B were already defined, we can easily use the named variables: `args=A B`.



```

1 \begin{pspicture*}(-3,-3)(4.5,2)
2 \psSolid[object=plan,
3   linecolor=gray,
4   definition=equation,
5   args={[0 1 1 0]},
6   base=-1 3 -2 2,
7   planmarks,
8   plangrid]
9 \psSolid[object=vecteur,
10  definition=vecteur3d,
11  args=0 0 1 1 1 1]%
12 \end{pspicture*}

```

6.3.3. Some other definitions of a vector

There are some other possibilities to define a vector. Here a list of some possible definitions with the appropriate arguments:

- `definition=addv3d; args= \vec{u} \vec{v} .`

Addition of 2 vectors.

- `definition=subv3d; args= \vec{u} \vec{v} .`

Difference of 2 vectors.

- `definition=mulv3d; args= \vec{u} λ .`

Multiplication of a vector with a real.

- `definition=vectprod3d; args= \vec{u} \vec{v} .`

Vector product of 2 vectors.

- `definition=normalize3d; args= \vec{u} .`

Normalized vector $\|\vec{u}\|^{-1}\vec{u}$.

6.4. The object `plan`

6.4.1. Presentation: type `plan` and type `solid`

The object `plan` is special in `pst-solides3d`. However, all the objects presented until now have had a common structure: they are of type `solid`: in other words, they are defined by a list of vertices, faces and colours.

For many applications, it is necessary to have some additional information for a plane: an origin, an orientation, a reference base etc.

To fulfill all these requirements, another data structure of type `plan` was created, which allows one to save all this necessary information. These manipulations of the plane will be controlled by such an object. Only when rendering takes place will an object of type `plan` be converted to an object of type `solid` which conforms to the macro `\psSolid`.

An object of type `plan` is used to describe an oriented affine plane. For a complete definition of such an object, an origin I , a basis (\vec{u}, \vec{v}) for that plane, a scaling of the axis (I, \vec{u}) and a scaling of the axis (I, \vec{v}) are needed. In addition, we can specify the fineness of the grid—in other words, the number of faces—used to represent that portion of the affine plane while transforming in an object of the type `solid`.

This type of object can be used to define planes of section; it is then necessary to define a plane for projection.

Its usage is quite easy to understand for users of PSTRicks. The only thing that you need to know is that, if we manipulate a `object=plan` with the macro `\psSolid`, we manipulate two objects at the same time: one of type `plan` and the other of type `solid`. When we select a backup of that object (see chapter “*Advanced usage*”) with the name `monplan` for example with the option `name=monplan`, there are in fact 2 backups that are effected. The first, with the name `monplan`, is an object of type `plan`, and the second, with the name `monplan_s`, is an object of type `solid`.

6.4.2. Defining an oriented plane

To generate such an object, one uses `object=plan` which comes with a few arguments:

- `definition` which specifies the method to defining the plane.
- `args` which specifies the necessary arguments for the method chosen.
- `base=xmin xmax ymin ymax` which specifies the dimensions of each axis.
- `[phi]` (value 0 by default) which specifies the angle of rotation (in degrees) of the plane around its normal.

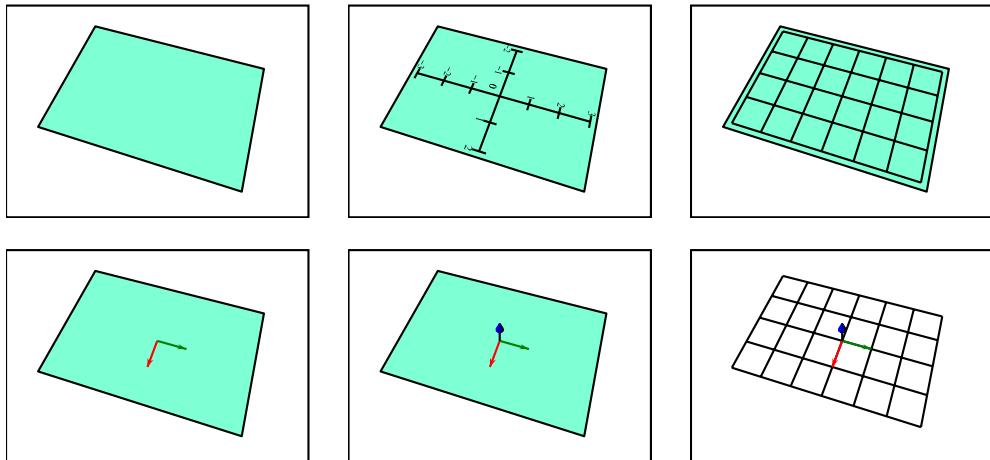
6. Some special objects

6.4.3. Special options

The object `plan` comes with some special options for viewing:

- `planmarks` which shows axes and scaling (with ticks),
- `plangrid` which shows the grid,
- `showbase` which shows the basis vectors for the plane, and
- `showBase` (note the capital letters) which shows the basis vectors of the plane and draws the associated normal vector.

These options apply regardless of the method of definition of the plane.



These options can be used, even if the plane is not drawn.

6.4.4. Defining a plane with a cartesian equation

The *cartesian equation* of a plane is of the form

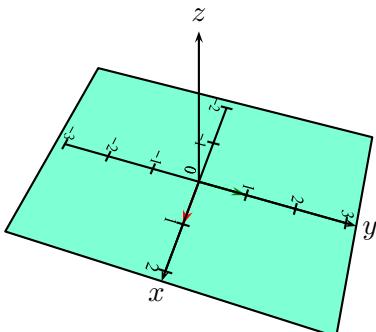
$$ax + by + cz + d = 0$$

The coefficients a , b , c and d determine an affine plane.

Usage with default orientation and origin

To define an affine plane, we can use `definition=equation`, and `args={[a b c d]}`. The orientation and origin of the affine plane must be given.

For example, the quadruple $(a, b, c, d) = (0, 0, 1, 0)$ determines the plane with the equation $z = 0$:



```

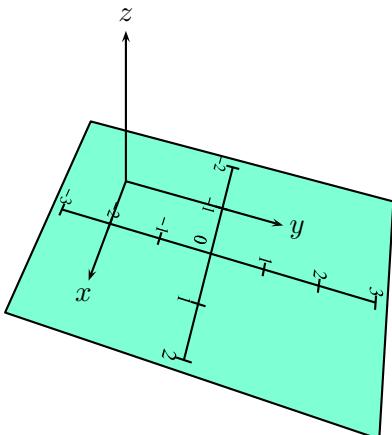
1 \psset{viewpoint=10 18 60 rtp2xyz,Decran=10,
2   fontsize=10,unit=0.65}
3 \begin{pspicture*}(-5,-4)(5,4)
4 \psSolid[object=plan,
5   definition=equation,
6   args={[0 0 1 0]},
7   fillcolor=Aquamarine,
8   planmarks,
9   base=-2.2 2.2 -3.2 3.2,
10  showbase]
11 \axesIIID(0,0,0)(2.2,3.2,4)
12 \end{pspicture*}
```

The parameter `base=xmin xmax ymin ymax` specifies the extent along each axis.

Specifying the origin

The parameter `origine=x0 y0 z0` specifies the origin of the affine plane. If the chosen point (x_0, y_0, z_0) doesn't fit the equation of the plane, it will be ignored.

For example, a plane with the equation $z = 0$ for which $(1, 2, 0)$ has been chosen as a possible origin:



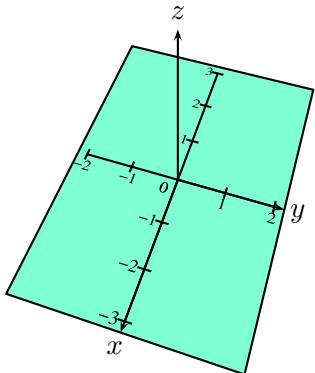
```

1 \psset{viewpoint=10 18 60 rtp2xyz,Decran=10,
2   fontsize=10,unit=0.65cm}
3 \begin{pspicture*}(-4,-5.5)(6,4)
4 \psSolid[object=plan,
5   definition=equation,
6   args={[0 0 1 0]},
7   fillcolor=Aquamarine,
8   origine=1 2 0,
9   base=-2.2 2.2 -3.2 3.2,
10  planmarks]
11 \axesIIID(0,0,0)(2.2,3.2,4)
12 \end{pspicture*}
```

Specifying the orientation

If the chosen orientation is unsatisfactory, we can specify an angle of rotation α (in degrees) around the normal of the plane with the syntax `args={[a b c d] alpha}`.

6. Some special objects



```

1 \psset{viewpoint=10 18 60 rtp2xyz,
2 Decran=10,fontsize=10,unit=0.65cm}
3 \begin{pspicture*}(-5,-4)(5,4)
4 \psSolid[object=plan,
5 definition=equation,
6 args={[0 0 1 0] 90},
7 fillcolor=Aquamarine,
8 base=-2.2 2.2 -3.2 3.2,
9 planmarks]
10 \axesIIID(0,0,0)(3.2,2.2,4)
11 \end{pspicture*}
```

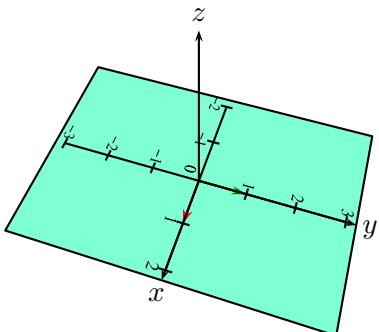
6.4.5. Defining a plane using a normal vector and a point

It is also possible to define a plane by giving a point and a normal vector. In this case one uses the parameter `definition=normalpoint`.

If wanted, we can specify the orientation, but it can be omitted.

First Method: orientation Unspecified

We use `args={ $x_0\ y_0\ z_0\ [a\ b\ c]$ }` where (x_0, y_0, z_0) is the origin of the affine plane, and (a, b, c) is a vector normal to that plane.

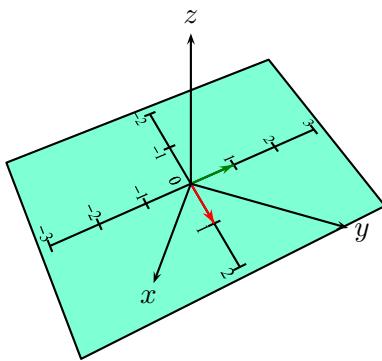


```

1 \psset{viewpoint=10 18 60 rtp2xyz,
2 Decran=10,fontsize=10,unit=0.65cm}
3 \begin{pspicture*}(-5,-4)(5,4)
4 \psSolid[object=plan,
5 definition=normalpoint,
6 args={0 0 0 [0 0 1]},
7 fillcolor=Aquamarine,
8 planmarks,
9 base=-2.2 2.2 -3.2 3.2,
10 showbase]
11 \axesIIID(0,0,0)(2.2,3.2,4)
12 \end{pspicture*}
```

Second Method: Specifying an angle of rotation

We use `args={ $x_0\ y_0\ z_0\ [a\ b\ c\ \alpha]$ }` where (x_0, y_0, z_0) is the origin of the affine plane, (a, b, c) a normal vector of that plane, and α the angle of rotation (in degrees) around the normal vector of that plane.

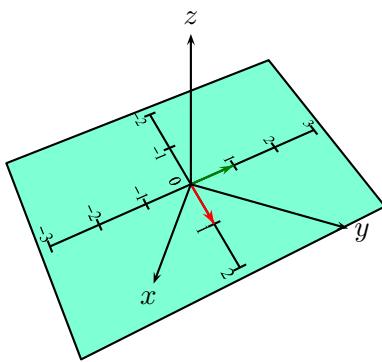


```

1 \psset{viewpoint=10 18 60 rtp2xyz,
2   Decran=10,fontsize=10,unit=0.65}
3 \begin{pspicture*}(-5,-4)(5,4)
4 \psSolid[object=plan,
5   definition=normalpoint,
6   args={0 0 0 [0 0 1 45]},
7   fillcolor=Aquamarine,
8   planmarks,
9   base=-2.2 2.2 -3.2 3.2,
10  showbase]
11 \axesIIID(0,0,0)(2.2,3.2,4)
12 \end{pspicture*}
```

Third Method: Specifying the first basis vector

We use `args={ $x_0 \ y_0 \ z_0 \ [u_x \ u_y \ u_z \ a \ b \ c]$ }` where (x_0, y_0, z_0) is the origin of the affine plane, (a, b, c) a normal vector of that plane, and (u_x, u_y, u_z) the first basis vector for that plane.

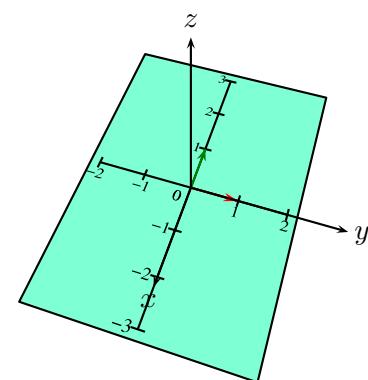


```

1 \psset{viewpoint=10 18 60 rtp2xyz,
2   Decran=10,fontsize=10,unit=0.65cm}
3 \begin{pspicture*}(-5,-4)(5,4)
4 \psSolid[object=plan,
5   definition=normalpoint,
6   args={0 0 0 [1 1 0 0 0 1]},
7   fillcolor=Aquamarine,
8   planmarks,
9   base=-2.2 2.2 -3.2 3.2,
10  showbase,
11 ]
12 \axesIIID(0,0,0)(2.2,3.2,4)
13 \end{pspicture*}
```

Fourth Method: Specifying the first basis vector and an angle of rotation

We use `args={ $x_0 \ y_0 \ z_0 \ [u_x \ u_y \ u_z \ a \ b \ c \ \alpha]$ }` where (x_0, y_0, z_0) is the origin of the affine plane, (a, b, c) is a normal vector of that plane, (u_x, u_y, u_z) is the first basis vector for that plane and α (in degrees) is a rotation around the axis of the normal vector.



```

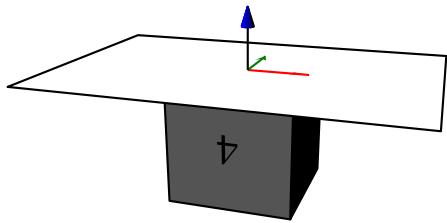
1 \psset{viewpoint=10 18 60 rtp2xyz,
2   Decran=10,fontsize=10,unit=0.65cm}
3 \begin{pspicture*}(-5,-4)(5,4)
4 \psSolid[object=plan,
5   definition=normalpoint,
6   args={0 0 0 [1 1 0 0 0 1 45]},
7   fillcolor=Aquamarine,
8   planmarks,
9   base=-2.2 2.2 -3.2 3.2,
10  showbase]
11 \axesIIID(0,0,0)(2.2,3.2,4)
12 \end{pspicture*}
```

6. Some special objects

6.4.6. Defining a plane from a face of a solid

We use `definition=solidface` with the arguments `args=name i` where `name` is the name of the designated solid and `i` is the index of the face. The origin is taken as the centre of the chosen face.

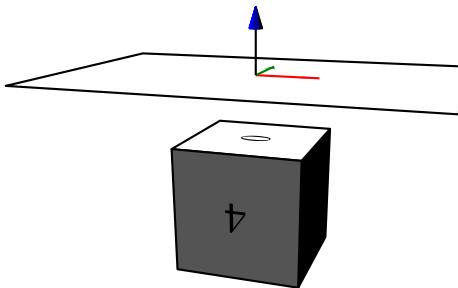
In the example below, the plane is defined through the face with the index 0 from the cube named `A`.



```

1 \psset{viewpoint=10 18 20 rtp2xyz,Decran=8}
2 \begin{pspicture}(-3.5,-2)(3,2.5)
3 \psset{solidmemory}
4 \psSolid[object=cube,a=2,fontsize=20,numfaces=all,name=A]
5 \psSolid[object=plan,
6   definition=solidface,
7   args=A 0,
8   showBase]
9 \end{pspicture}
```

If the user specifies the coordinates (x, y, z) within the macro `\psSolid[...](x,y,z)`, a plane is generated parallel to the face with index `i` of the solid `name`, and translated to the point (x, y, z) which now is taken as the origin.



```

1 \psset{viewpoint=10 18 20 rtp2xyz,Decran=8}
2 \begin{pspicture}(-3.5,-1.5)(3,3)
3 \psset{solidmemory}
4 \psSolid[object=cube,a=2,fontsize=20,numfaces=all,name=A]
5 \psSolid[object=plan,
6   definition=solidface,
7   args=A 0,
8   showBase](0,0,2)
9 \end{pspicture}
```

6.5. The object `geode`

6.5.1. Mathematical presentation

Some excellent tutorials about geodes and their duals are available on the following websites:

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

The parametrisation of a geode complies with that given on the website:

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

“We can define a geode with two parameters: a number N indicating the type of the initial polyhedron ($N = 3$ for the tetrahedron, $N = 4$ for the octahedron and $N = 5$ for the icosahedron) and a number n indicating the number of divisions along the edge’s length.”

The article *Indexing the Sphere with the Hierarchical Triangular Mesh* describes a method that allows us to obtain a representation of geodes:

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

6.5.2. Construction with `pst-solides3d`

Two approaches are possible to generate a geode or its dual: either via `\codejps`, or via the objects of `\psSolid`.

For a geode, the codes

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

and

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

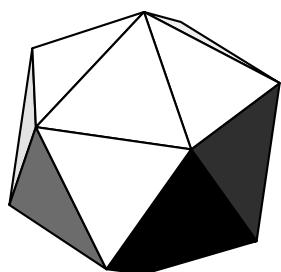
are equivalent. And for its dual, the codes

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

and

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

6.5.3. Some examples of geodes and their duals

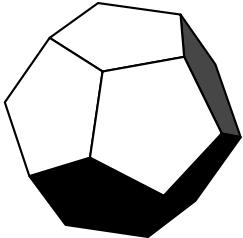


N=5 n=0

```

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

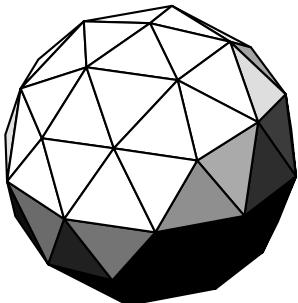
6. Some special objects



```

1 \begin{pspicture}(-3,-3)(3,3)
2 \psset{viewpoint=50 -20 30 rtp2xyz,Decran=100}
3 \psSolid[object=geode,
4   dualreg,
5   ngrid=5 0]
6 \%code{jpsf5 0 newdualgeode drawsolid**}
7 \psframe*(-2,-2.8)(2,-2.2)
8 \rput(0,-2.5){\textcolor{white}{\textsf{N=5 n=0}}}
9 \end{pspicture}
```

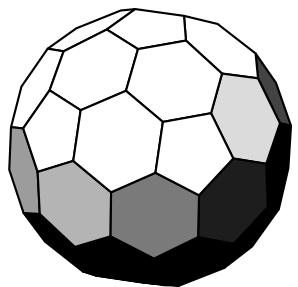
N=5 n=0



```

1 \begin{pspicture}(-3,-3)(3,3)
2 \psset{viewpoint=50 -20 30 rtp2xyz,Decran=100}
3 \psSolid[object=geode,
4   ngrid=5 1]
5 \%code{jpsf5 1 newgeode drawsolid**}
6 \psframe*(-2,-2.8)(2,-2.2)
7 \rput(0,-2.5){\textcolor{white}{\textsf{N=5 n=1}}}
8 \end{pspicture}
```

N=5 n=1



```

1 \begin{pspicture}(-3,-3)(3,3)
2 \psset{viewpoint=50 -20 30 rtp2xyz,Decran=100}
3 \psSolid[object=geode,
4   dualreg,
5   ngrid=5 1]
6 \%code{jpsf5 1 newdualgeode drawsolid**}
7 \psframe*(-2,-2.8)(2,-2.2)
8 \rput(0,-2.5){\textcolor{white}{\textsf{N=5 n=1}}}
9 \end{pspicture}
```

N=5 n=1

6.5.4. The parameters of the geodes

The radius of the sphere is fixed at 1, so to vary the dimensions of the geodes one plays around with one or the other of the two following parameters:

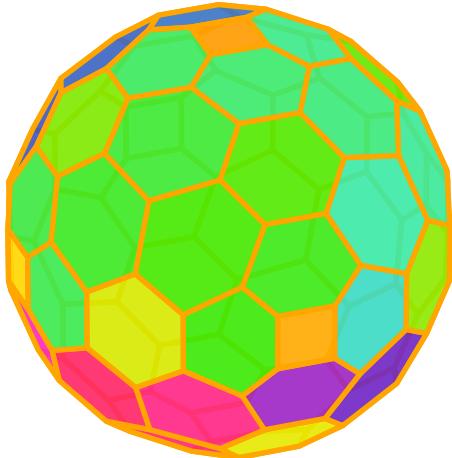
- The unit: `\psset{unit=2}`

- The position of the screen: `viewpoint=50 -20 30,Decran=100`, if the distance to the screen is twice as far as the distance to the viewer, one scales the scenery by a factor of two.

Note: Within `jps`, the setup for the geode is `\codejps{N n newgeode}` and for its dual it is `\codejps{N n newdualgeode}`.

Note: With `\psSolid`, the parameters N and n are transmitted via the argument `ngrid`

The color and transparency options are available for the geodes as well.



```

1 \psset{viewpoint=50 -20 30 rtp2xyz,Decran=150}
2 \begin{pspicture}(-3.5,-3.5)(3.5,3.5)
3 \psset{linewidth=2pt}
4 \codejps{
5 /geode42{4 2 newdualgeode} def
6 .7 setfillopacity
7 orange
8 /geodetransparente{
9 geode42
10 dup videsolid
11 dup (orange) inputcolors
12 dup [.1 .9] solidputhuecolors} def
13 geodetransparente
14 drawsolid**}
15 \end{pspicture}

```

6.5.5. Advice for a ‘fast’ construction of a geode

The calculation time for the geodes and their duals depends on the number of divisions of an edge (the second parameter n) and will increase rapidly with n which is really uncomfortable, because one has to wait more or less patiently, until the result of the transformation `dvips->ps2pdf` is ready.

As happens for all other solids, it is possible to save the calculation in external files, which then saves calculation time when one has to make a test run of colours or view point.

We have to operate in two stages:

Backup the parameters of the geodes in a .dat file

```

\documentclass{article}
\usepackage{pst-solides3d}
\begin{document}
\codejps{
4 4 newdualgeode
dup {[.5 .6]} exec solidputhuecolors

```

6. Some special objects

```
(geodedual44) writesolidfile  
}  
\end{document}
```

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

The last operation will generate 4 files:

- `geodedual44-couleurs.dat` → the colors of the faces;
- `geodedual44-faces.dat` → the list of the faces;
- `geodedual44-sommets.dat` → the list of the vertices;
- `geodedual44-io.dat` → the number of the faces and vertices.

Note: By default, under Windows and Linux, the security of files on the hard drive is activated and doesn't allow you to write on the drive. To deactivate that security option, more or less temporarily, here the two corresponding procedures:

Linux: Advice from Jean-Michel SARLAT: the simplest will be to use GhostScript directly, within the console. As there is no image to wait for:

```
$> gs -dNOSAFER monfichier.ps quit.ps
```

Windows: Within the menu Options, the option Security of files must be unchecked.

Reading the data and drawing the geode

The advantage of this method becomes even more evident when one compares the compilation of two files producing the same result by different methods:

The file `geode42_direct.tex` calculates the solid and its view. The file `geode42_precalcul.tex` uses the file `.dat` including the precalculated data of the file `calc_geode42.tex`. These three files are included in the distribution.

6.5.6. Some other examples

You will find numerous other examples of geodes on the website:

<http://melusine.eu.org/lab/bpst/pst-solides3d/geodes>

CHAPTER 7

Generating some new solids

7.1. The *jps* code

jps code contains all the PostScript code that is used by the library developed for the software *jps2ps*.

The **solides.pro** file of the **solides3d** package contains all the elements native to that library, which contains about 4500 functions and procedures.

It allows us to have available some adapted commands in mathematical form, without having to construct them with the primitives **moveto**, **lineto**, **curveto**, etc.

For example, we can define a function F with $F(t) = (3 \cos^3 t, 3 \sin^3 t)$, and draw its curve with the *jps code* `0 360 {F} CourbeR2`.

If we only want to have the path of that curve, we use the code `0 360 {F} CourbeR2_`, and if we want to add this to the stack of points of the curve, we use `0 360 {F} CourbeR2+`.

In all of the 3 examples below, the number of points is declared by the global variable **resolution**.

In other words, with the function F named above and a fixed resolution of 36, the *jps code*

```
0 360 {F} CourbeR2+
```

is equivalent to the PostScript code

```
0 10 360 {
    /angle exch def
    3 angle cos 3 exp mul
    3 angle sin 3 exp mul
} for
```

We haven't yet developed documentation for the library hidden in the **solides.pro** file. For the moment we refer the *Guide de l'utilisateur de jps2ps* for the interested user available at the website

7. Generating some new solids

melusine.eu.org/syracuse/bbgraf.

7.2. Defining a function

It is possible to define functions usable in a PostScript environment.

The domain can be \mathbb{R} , \mathbb{R}^2 or \mathbb{R}^3 , and the codomain can be \mathbb{R} , \mathbb{R}^2 or \mathbb{R}^3 .

The definition is made with the macro `\defFunction`. This macro comes with six arguments, where the first is optional.

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

- <options> We insert the options typical to PStricks, like `linewidth` etc., and, some of them defined by `pst-solides3d`. A very nice and helpful option is `algebraic`, with which one can avoid RPN (Reverse Polish Notation). All the options are key value pairs separated with commas.
- <name> This is a unique name of your choice—but be careful: avoid names that contain accents, PostScript doesn't like them at all.
- <var> We insert at most three variables, arbitrarily chosen and separated with commas.
- <x(var)> Here, we place functions defining the three Euclidean components
- <y(var)> x, y, z . If one of the three components is not wanted, just enter
- <z(var)> a 0 within parentheses—this will also allow you to define some projections of the lines of functions.

Once you have defined a function, this function is always called by its chosen name <name>.

Here some examples:

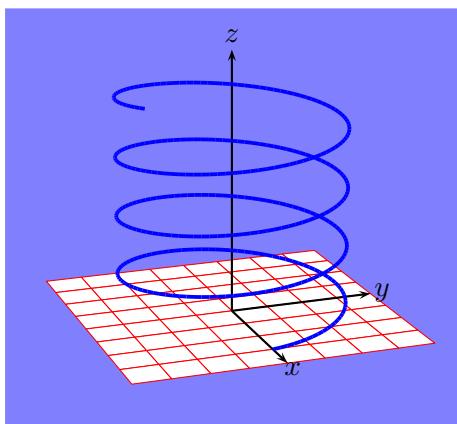
- `\defFunction{moncercle}(t){t cos 3 mul}{0}{t sin 3 mul}`
draws a circle with radius 3 in the xOz plane (notation RPN).
- `\defFunction[algebraic]{helice}(t){cos(t)}{sin(t)}{t}`
draws a helix in algebraic notation.
- `\defFunction[algebraic]{F}(t){t}{}`
draws a function from \mathbb{R} in \mathbb{R}
- `\defFunction[algebraic]{F}(t){t}{t}{}`
draws a function from \mathbb{R} in \mathbb{R}^2
- `\defFunction[algebraic]{F}(t){t}{t}{t}`
draws a function from \mathbb{R} in \mathbb{R}^3

There remains work to be done on this macro. For the moment it does not permit an arbitrary choice of names of variables, as this risks conflict with existing names. Please use names analogous to those used in the documentation. A good strategy is to systematically use one or more numerical characters at the end of the names of your variables.

7.3. Curves of functions from \mathbb{R} in \mathbb{R}^3

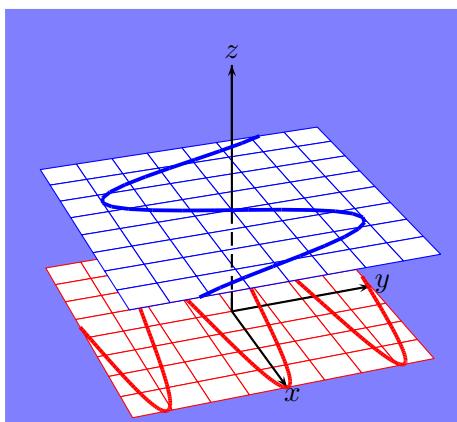
The line of a defined function calls the object `courbe` and the option `function`. We can realize a helix in algebraic notation with the function:

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



```

1 \psset{unit=0.5}
2 \begin{pspicture}(-6,-3)(6,8)
3 \psframe*[linecolor=blue!50](-6,-3)(6,8)
4 \psSolid[object=grille,base=-4 4 -4 4, linecolor=red,
           linewidth=0.5\pslinewidth]%
5 \axesIIID(0,0,0)(4,4,7)
6 \defFunction[algebraic]{helice}(t){3*cos(4*t)},{3*sin(4*t)},{t}
7 \psSolid[object=courbe,
           r=0,
           range=0 6,
           linecolor=blue, linewidth=0.1,
           resolution=360,
           function=helice]%
13 \end{pspicture}
```



```

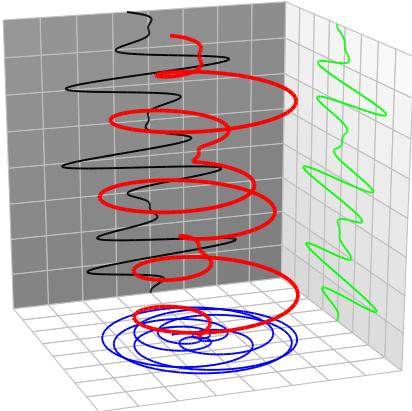
1 \psset{unit=0.5}
2 \begin{pspicture}(-6,-3)(6,8)
3 \psframe*[linecolor=blue!50](-6,-3)(6,8)
4 \psset{lightsrc=10 -20 50,viewpoint=50 -20 30 rtp2xyz,
        Decran=50}
5 \psSolid[object=grille,base=-4 4 -4 4, linecolor=red,
           linewidth=0.5\pslinewidth]%
6 \axesIIID(0,0,0)(4,4,7)
7 \psset{range=-4 4}
8 \defFunction{cosRad}(t){ t 2 mul Cos 4 mul }{ t }{ 0 }
9 \psSolid[object=courbe, linewidth=0.1,
10   r=0, linecolor=red,
11   resolution=360,
12   function=cosRad]
13 \psSolid[object=grille,base=-4 4 -4 4, linecolor=blue,
           linewidth=0.5\pslinewidth](0,0,3)
14 \psPoint(0,0,3){O1}\psPoint(0,0,7){Z1}\psline(O1)(Z1)\psline[linestyle=dashed](O1)(0)
15 \pstVerb{/tmin -4 def /tmax 4 def}%
16 \defFunction{sinRad}(t){ t }{ t Sin 3 mul }{ 3 }
17 \psSolid[object=courbe, linewidth=0.1,
18   r=0, linecolor=blue,
19   resolution=30,
20   function=sinRad]
21 \end{pspicture}
```

7. Generating some new solids

```

1 \psset{unit=0.5}
2 \begin{pspicture}(-6.5,-3)(7,11)
3 \psset{lightsrc=10 -20 50,viewpoint=50 -20 20 rtp2xyz,
        Decran=50}
4 \psSolid[object=grille,base=-4 4 -4 4,
5   linecolor=lightgray,linewidth=0.5\pslinewidth]%
6 \psSolid[object=grille,base=-4 4 0 8,
7   linecolor=lightgray,RotX=90,
8   linewidth=0.5\pslinewidth](0,4,0)
9 \psSolid[object=grille,base=-4 4 -4 4,
10  linecolor=lightgray,RotY=90,
11  linewidth=0.5\pslinewidth](-4,0,4)
12 \defFunction[algebraic]{helice}(t)%
13 {1.3*(1-cos(2.5*t))*cos(6*t)}
14 {1.3*(1-cos(2.5*t))*sin(6*t)}{t}
15 \defFunction[algebraic]{helice_xy}(t)%
16 {1.3*(1-cos(2.5*t))*cos(6*t)}
17 {1.3*(1-cos(2.5*t))*sin(6*t)}{0}
18 \defFunction[algebraic]{helice_xz}%
19 (t){1.3*(1-cos(2.5*t))*cos(6*t)}{4}{t}
20 \defFunction[algebraic]{helice_yz}%
21 (t){-4}{1.3*(1-cos(2.5*t))*sin(6*t)}{t}
22 \psset{range=0 8}
23 \psSolid[object=courbe,r=0,linecolor=blue,
24 linewidth=0.05,resolution=360,
25 normal=0 0 1,function=helice_xy]
26 \psSolid[object=courbe,r=0,
27 linecolor=green,linewidth=0.05,
28 resolution=360,normal=0 0 1,
29 function=helice_xz]
30 \psSolid[object=courbe,r=0,
31 linewidth=0.05,resolution=360,
32 normal=0 0 1,function=helice_yz]
33 \psSolid[object=courbe,r=0,
34 linecolor=red,linewidth=0.1,
35 resolution=360,function=helice]
36 \end{pspicture}

```



These last function lines are found in an animated form on the website:

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

7.4. Tubes

This section is about to substitute a curve in two or three dimensions (2D or 3D), that are setup parameterised, by a tube, where the initial curve is the axes and we can choose the radius and grid. We find some mathematical elements concerning these objects on the following websites:

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

As usual, the `pst-solides3d` package offers two possibilities to draw the tubes:

- via PStricks and the argument `object` of `\psSolid`
- directly with `\codejps`

Note: It is often advisable to calculate in advance, by hand or with a preferred software, the first derivatives of the parametric functions which define the coordinates.

However, if this derivative isn't defined explicitly by the user, the package makes some approximate calculations, but the result then is not always sufficient.

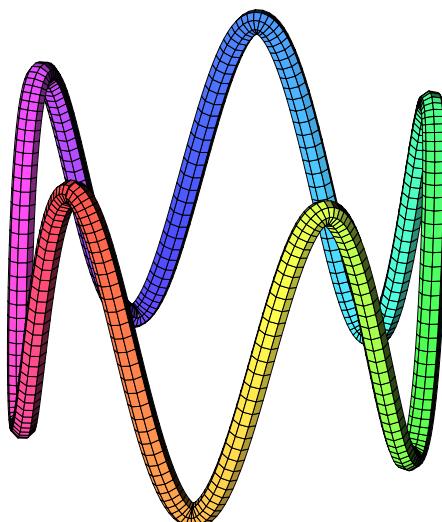
7.4.1. Usage with PSTricks

Give your curves a relief

“Donnez du relief à vos courbes”, this is the title of the article from Robert FERRÉOL, available on:

<http://mapage.noos.fr/r.ferreol/atelecharger/textes/relief/courbes%20en%20relief.html>

from who the following functions were borrowed and which are analogous to a Lissajous figure enrolled around a cylinder.



```

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

The argument `object=courbe` with the parameters `r`, `function` and `range` is used to specify the radius of the tube, the name of the function to be used and the range.

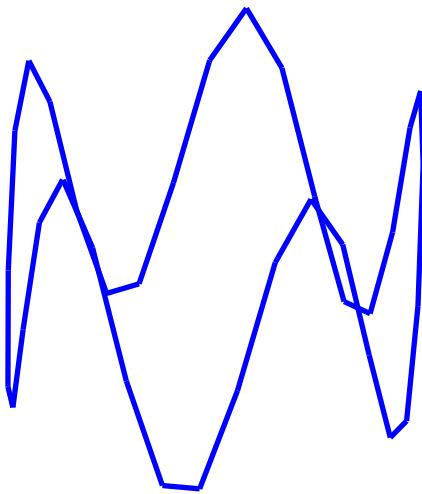
We can also refine the grid with the optional argument `ngrid=n1 n2` where n_1 represents the number of vertices of a section of a tube (if $n_1 = 6$, this gives a tube with a hexagonal section) and n_2 represents

7. Generating some new solids

the number of divisions along it.

A hairline curve is produced with the radius r=0

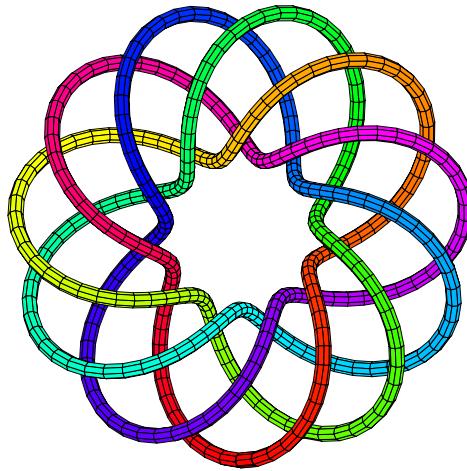
And thus, no fear to specify the derived function.



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

7.4.2. Usage with \code{jps}

The syntax is \code{jps\{t_min t_max (name_function) radius_tube [n1 n2] newtube\}}.



```

1 \begin{pspicture}(-3.5,-3.5)(4,3.5)
2 \psset{lightsrc=80 30 30,viewpoint=100 45 90 rtp2xyz,Decran=100,linewidth=0.2pt}
3 \codejps{
4 /rpn {tx@AlgToPs begin AlgToPs end cvx exec} def
5 /xc {{(2+1*cos(2.75*t))*cos(t)} rpn } def
6 /yc {{(2+1*cos(2.75*t))*sin(t)} rpn } def
7 /zc {{1*sin(2.75*t)} rpn } def
8 /xc' {{(-2.75*sin(2.75*t)*cos(t)-(2+cos(2.75*t))*sin(t)) rpn } def
9 /yc' {{(-2.75*sin(2.75*t)*sin(t)+(2+cos(2.75*t))*cos(t)) rpn } def
10 /zc' {{(2.75*cos(2.75*t)) rpn } def
11 /g { 3 dict begin /t exch def xc yc zc end } def
12 /g' { % first derivative
13 3 dict begin /t exch def xc' yc' zc' end } def
14 /solenoide{
15 % t_min t_max (name_function) radius_tube [resolution]
16 0 25.2 (g) 0.1 [360 8] newtube dup [0 1] solidputhuecolors} def
17 solenoide
18 drawsolid**
19 }%
20 \end{pspicture}

```

7.4.3. Improving the speed of readout

The curve with the name “*horopter*” is the subject of this website:

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

Obtaining the curve directly

The following lines allow us to calculate the points and draw the curve. The resolution `ngrid=72 12` of the curve was increased, so some more calculation time to produce the result, which some will judge as very long.

```
\begin{pspicture}(-7,-2)(7,4)
\psset{lightsrc=80 30 30}
```

7. Generating some new solids

```
\psset{viewpoint=1000 60 20 rtp2xyz,Decran=1000}
\psframe(-7,-2)(7,4)
\psset{solidmemory}
\codejps{/a 2 def /b 2 def}%
\defFunction[algebraic]{F3}(t)
  {a*(1+cos(t))}%
  {b*tan(t/2)}%
  {a*sin(t)}%
\defFunction[algebraic]{F3'}(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=F3,hue=0 1 0.7 1,
  action=none,name=H1,
  r=1]%
\psSolid[object=cylindrecieux,
  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}
```

Saving the parameters of the curve

If this curve is used several times, it is advisable to backup all the characteristics of that curve, like: co-ordinates of the vertices, list of colours of the faces with placing the last command `action=writesolid`:

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

The following sequence `LaTeX fichier.tex->dvips->GSview (Windows) or gv (Linux)` will generate 4 files:

- `horoptere-couleurs.dat` → the colours of the faces;
- `horoptere-faces.dat` → the list of faces;
- `horoptere-sommets.dat` → the list of vertices;
- `horoptere-io.dat` → the number of faces and vertices.

then read and execute the files with the command: `\psSolid[object=datfile,file=horoptere]`, the time saved can be quite significant

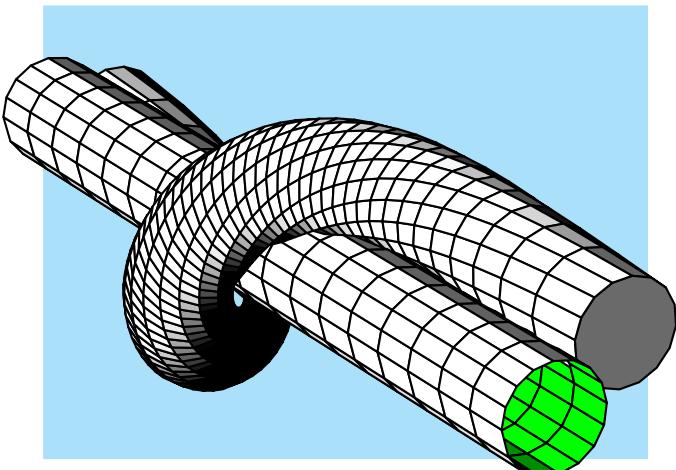
Note: By default, under Windows and Linux, the security of files on the hard drive is activated and doesn't allow to write on the drive. To deactivate that security option, more or less temporarily, here the two corresponding procedures:

Linux: The advice from Jean-Michel SARLAT: the simplest will be to use GhostScript directly, within the console. As there is no image to wait for:

```
$> gs -dNOSAFER monfichier.ps quit.ps
```

Windows: Within the menu Options, the option Security of files must be turned to unchecked.

The plot of the curve



```

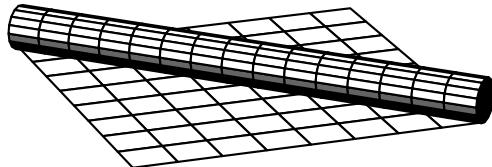
1 \begin{pspicture}(-5,-3.5)(4,3)
2 \psset{lightsrc=80 30 30}
3 \psset{viewpoint=100 60 20 rtp2xyz,
        Decran=75}
4 \psframe*[linecolor=cyan!30](-4.5,-3)(3.5,3)
5 \psSolid[object=datfile,file=../horoptere]
6 \end{pspicture}

```

7. Generating some new solids

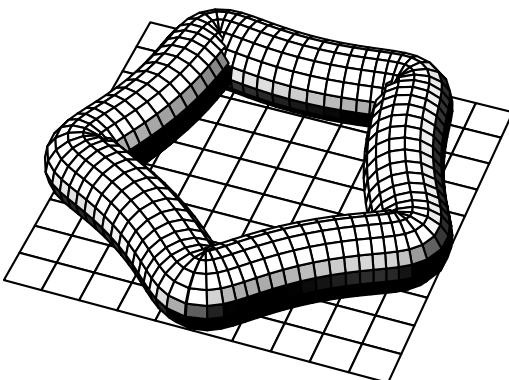
7.4.4. Some other examples

A straight line



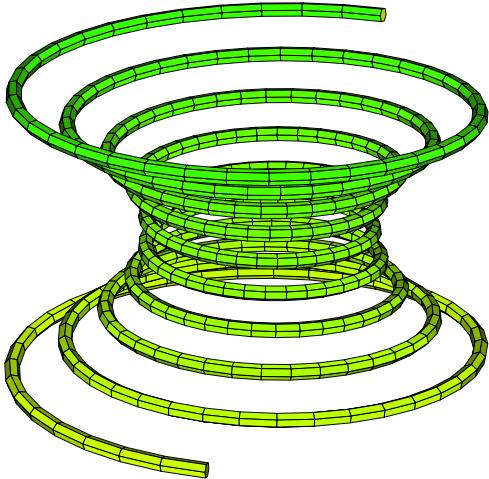
```
1 \begin{pspicture}(-3.5,-2)(3.5,2)
2 \psset{viewpoint=100 -20 20 rtp2xyz,
3 Decran=75,unit=0.8}
4 \psSolid[object=grille,base=-4 4 -4 4]%
5 \defFunction[algebraic]{FIV}(t){t}{t}{t}{0.5}
6 \defFunction[algebraic]{FIV'}(t){1}{1}{1}{0}
7 \psSolid[object=courbe,
8 range=-4 4, ngrid=16 16,
9 function=FIV,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

A hypocycloid



```
1 \begin{pspicture}(-3.5,-3)(3.5,3)
2 \psset{viewpoint=100 20 45 rtp2xyz,
3 Decran=75,unit=0.7}
4 \psSolid[object=grille,base=-5 5 -5 5]%
5 \defFunction[algebraic]{FII}(t)
6 {4*cos(t)+cos(4*t)/2}
7 {4*sin(t)-sin(4*t)/2}
8 {1}
9 \defFunction[algebraic]{FII'}(t)
10 {-4*sin(t)-2*sin(4*t)}
11 {4*cos(t)-2*cos(4*t)}
12 {0}
13 \psSolid[object=courbe,
14 range=0 6.28,ngrid=90 16,
15 function=FII,r=1]
16 \end{pspicture}
```

The spring of Gaston



```

1 \begin{pspicture}(-3.5,-4)(3.5,4.5)
2 \psset{lightsrc=80 30 30,
3   viewpoint=100 20 20 rtp2xyz,Decran=50}
4 \defFunction[algebraic]{FIII}(t)
5   {(t^2+3)*sin(15*t)}
6   {(t^2+3)*cos(15*t)}{2*t}
7 \defFunction[algebraic]{FIII'}(t)
8   {2*t*sin(15*t)+15*(t^2+3)*cos(15*t)}
9   {2*t*cos(15*t)-15*(t^2+3)*sin(15*t)}{2}
10 \psSolid[object=courbe,
11   range=-2 2,ngrid=360 6,
12   function=FIII,hue=0.2 0.3,
13   linewidth=0.1pt,r=0.2]
14 \end{pspicture}
```

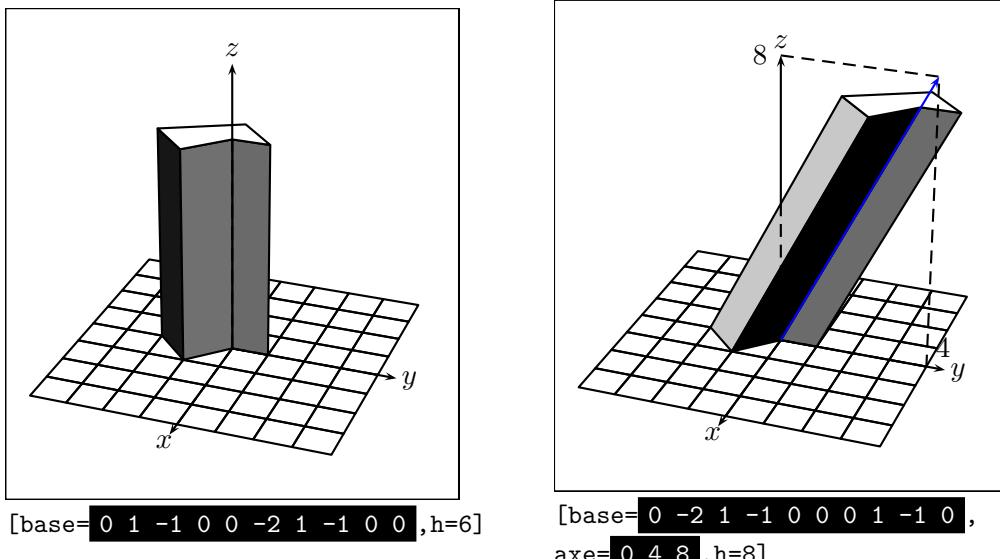
7.5. The prism

A prism is determined by two parameters:

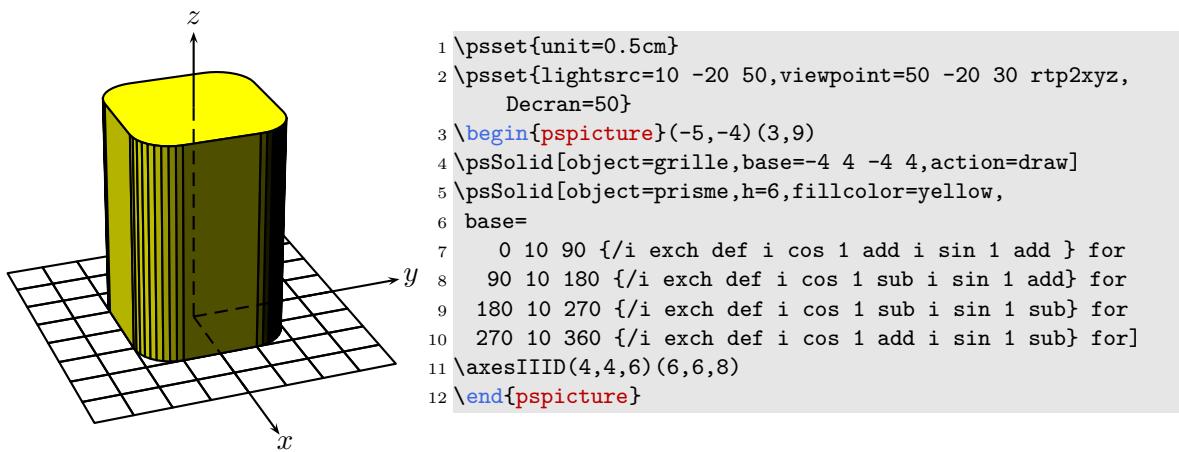
- The base of the prism can be defined by the coordinates of the vertices in the xy -plane. Note that it is necessary that the four vertices be given in counterclockwise order with respect to the barycentre of the base;
- the direction of the prism axis (the components of the shearing vector).

7. Generating some new solids

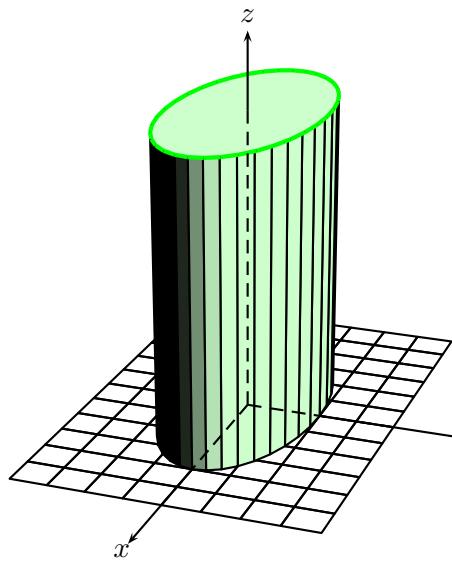
Example 1: a right and oblique prisms with polygonal section



Example 2: a right prism with cross-section a rounded square



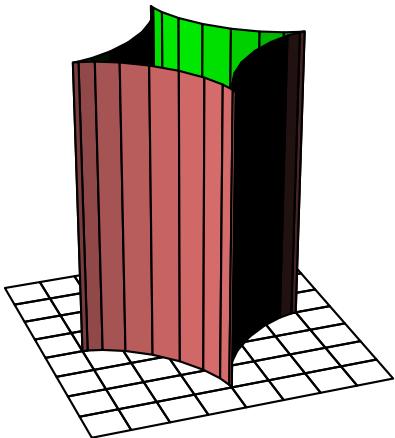
Example 4: a prism with an elliptic section



```

1 \psset{unit=0.5cm}
2 \begin{pspicture}(-6,-5)(4,12)
3 \psset{lightsrc=10 20 30,viewpoint=50 20 25 rtp2xyz,Decran
      =50}
4 \psSolid[object=grille,base=-6 6 -4 4,action=draw]
5 \defFunction{FuncI}(t){t cos 4 mul}{t sin 2 mul}{}
6 \psSolid[object=prisme,h=8,fillcolor=green!20,
      base=0 350 {FuncI} CourbeR2+]
7 \defFunction{FuncII}(t){t cos 4 mul}{t sin 2 mul}{8}
8 \psSolid[object=courbe,r=0,
9   function=FuncII,range=0 360,
10  linewidth=2\pslinewidth,
11  linecolor=green]
12 \axesIIID(6,4,8)(8,6,10)
13 \end{pspicture}
14 
```

Example 3: a right prism with a star-shaped section



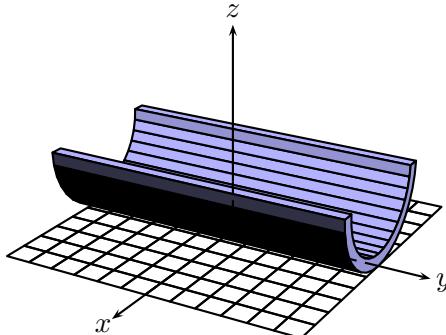
```

1 \psset{unit=0.5cm}
2 \psset{lightsrc=10 -20 50,viewpoint=50 -20 30 rtp2xyz,
      Decran=50}
3 \begin{pspicture*}(-5,-4)(6,9)
4 \defFunction{F}(t){3 t cos 3 exp mul}{3 t sin 3 exp mul}{}
5 \psSolid[object=grille,base=-4 4 -4 4,action=draw]
6 \psSolid[object=prismecreux,h=8,fillcolor=red!50,
      resolution=36,
8   base=0 350 {F} CourbeR2+
9 ]
10 \end{pspicture*}

```

7. Generating some new solids

Example 5: a roof gutter with a semi-circular section



```

1 \psset{unit=0.35cm}
2 \psset{lightsrc=10 20 30,viewpoint=50 30 25 rtp2xyz,
Decran=50}
3 \begin{pspicture}(-10,-5)(6,10)
4 \defFunction[algebraic]{F}(t)
5 {3*cos(t)}{3*sin(t)}{}
6 \defFunction[algebraic]{G}(t)
7 {2.5*cos(t)}{2.5*sin(t)}{}
8 \psSolid[object=grille,
9 base=-6 6 -6 6,action=draw]%
10 \psSolid[object=prisme,h=12,
11 fillcolor=blue!30,RotX=-90,
12 resolution=19,
13 base=0 pi {F} CourbeR2+
14 pi 0 {G} CourbeR2+](0,-6,3)
15 \axesIIID(6,6,2)(8,8,8)
16 \end{pspicture}

```

We draw the exterior face (semicircle of radius 3 cm) in counterclockwise order: $0 \text{ pi } \{F\} \text{ CourbeR2+}$
Then the interior face (semicircle of radius 2.5 cm), is drawn in clockwise order: $\text{pi } 0 \{G\} \text{ CourbeR2+}$

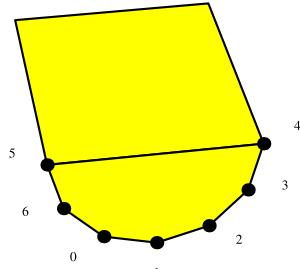
We can turn the solid -90° and place it at the point $(0, -6, 3)$. If we use the `algebraic` option to define the functions F and G , the functions sin and cos are in radians.

The parameter `decal`

We wrote above that the first four vertices must be given in counterclockwise order with respect to the barycentre of the vertices of the base. In fact, this is the default version of the following rule: If the base has $n + 1$ vertices, and if G is their barycentre, then (s_0, s_1) on one hand and (s_{n-1}, s_n) on the other, should be in counterclockwise order with respect to G .

This rule puts constraints on the coding of the base of a prism which sometimes renders the latter unaesthetically. For this reason we have introduced the argument `decal` (default value= -2) which allows us to consider the list of vertices of the base as a circular file which you will shift round if needed.

An example: default behavior with `decal=-2`:



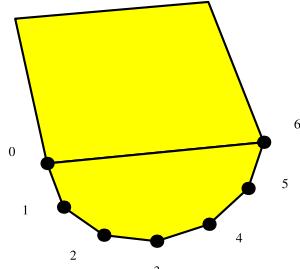
```

1 \psset{unit=0.5}
2 \begin{pspicture}(-6,-4)(6,7)
3 \defFunction{F}(t){t cos 3 mul}{t sin 3 mul}{}
4 \psSolid[object=prisme,h=8,
5     fillcolor=yellow,RotX=-90,
6     num=0 1 2 3 4 5 6,
7     show=0 1 2 3 4 5 6,
8     resolution=7,
9     base=0 180 {F} CourbeR2+
10 ](0,-10,0)
11 \end{pspicture}

```

We see that the vertex with index 0 is not where we expect to find it.

We start again, but this time suppressing the renumbering:



```

1 \psset{unit=0.5}
2 \begin{pspicture}(-6,-4)(6,7)
3 \defFunction{F}(t){t cos 3 mul}{t sin 3 mul}{}
4 \psSolid[object=prisme,h=8,
5     fillcolor=yellow,RotX=-90,
6     decal=0,
7     num=0 1 2 3 4 5 6,
8     show=0 1 2 3 4 5 6,
9     resolution=7,
10    base=0 180 {F} CourbeR2+
11 ](0,-10,0)
12 \end{pspicture}

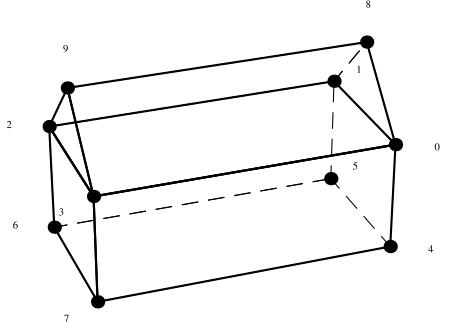
```

7.6. Construction from scratch

The object `new` constructs a solid. Two parameters are used: `sommets` which indicates the list of coordinates of the different vertices, and `faces` which gives the list of faces of the solid; a face is characterized by a list of the indices of its vertices, listed in counterclockwise order when the face is viewed from the exterior of the solid.

7. Generating some new solids

7.6.1. Example 1: a house

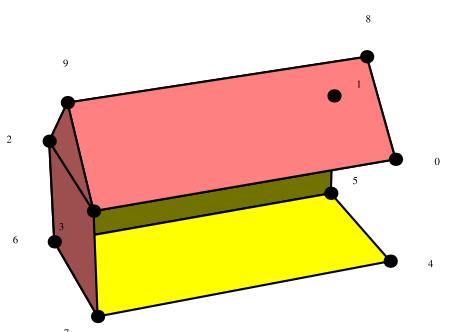


```

1 \psset{unit=0.5}
2 \psset{lightsrc=10 -20 50,viewpoint=50 -20 30 rtp2xyz,
      Decran=50}
3 \begin{pspicture}(-7,-4)(7,7)
4 \psSolid[object=new,
5   sommets=
6     2 4 3 -2 4 3 -2 -4 3 2 -4 3
7     2 4 0 -2 4 0 -2 -4 0 2 -4 0
8     0 4 5 0 -4 5,
9   faces={
10    [0 1 2 3] [7 6 5 4] [0 3 7 4]
11    [3 9 2] [1 8 0] [8 9 3 0]
12    [9 8 1 2] [6 7 3 2] [2 1 5 6]},
13   num=all,show=all,action=draw]
14 \end{pspicture}

```

Note that the solid `new` uses the same options as the other solids. For example, we give the same solid as above below, using the parameters `hollow`, `incolor`, `fillcolor`, and `rm`.



```

1 \psset{unit=0.5}
2 \psset{lightsrc=10 -20 50,viewpoint=50 -20 30 rtp2xyz,
      Decran=50}
3 \begin{pspicture}(-7,-3.5)(7,7.5)
4 \psSolid[object=new,fillcolor=red!50,incolor=yellow,
5   action=draw**,hollow,rm=2,
6   sommets=
7     2 4 3 -2 4 3 -2 -4 3 2 -4 3
8     2 4 0 -2 4 0 -2 -4 0 2 -4 0
9     0 4 5 0 -4 5,
10   faces={[0 1 2 3][7 6 5 4][0 3 7 4]
11   [3 9 2][1 8 0][8 9 3 0][9 8 1 2]
12   [6 7 3 2][2 1 5 6]},
13   num=all,show=all]
14 \end{pspicture}

```

7.6.2. Example 2: a hyperboloid with a fixed radius

As always, the options of the macro `\psSolid` can handle Postscript code, even *jps code*

Unlike an example in pure PostScript, where we use the parameters a , b and h which are transmitted by the options of PStricks. In this way one obtains a variable solid constructed from scratch.

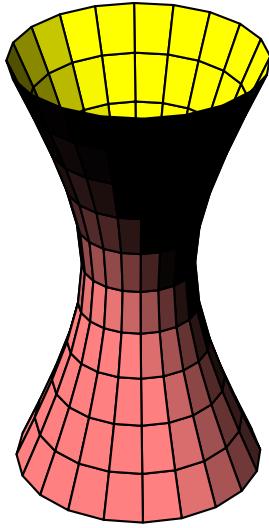
Remark: the code being used comes from a *jps* source used in practice, as in:

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

```

1 \psset{unit=0.75}
2 \psset{lightsrc=10 -20 50,viewpoint=50 -20 30 rtp2xyz,
Decran=50}
3 \begin{pspicture}(-5,-5)(3,5)
4 \psSolid[object=new,fillcolor=red!50,incolor=yellow,
5 hollow, a=10, %% nb d'etages
6 b=20, %% diviseur de 360, nb de meridiens
7 h=8, %% hauteur
8 action=draw**,sommets=
9 /z0 h neg 2 div def
10 a -1 0 {
11 /k exch def
12 0 1 b 1 sub {
13 /i exch def
14 /r z0 h a div k mul add dup mul 4 div 1 add sqrt def
15 360 b idiv i mul cos r mul 360 b idiv i mul sin r mul
16 z0 h a div k mul add
17 } for
18 } for,
19 faces={
20 0 1 a 1 sub {
21 /k exch def
22 k b mul 1 add 1 k 1 add b mul 1 sub {
23 /i exch def
24 [i i 1 sub b i add 1 sub b i add]
25 } for
26 [k b mul k 1 add b mul 1 sub k 2 add b mul 1 sub k 1 add
b mul]
27 } for
28 }]
29 \end{pspicture}

```

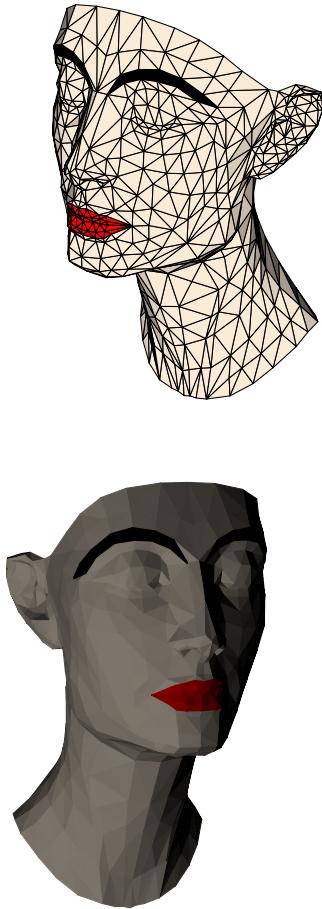


7.6.3. Example 3: importing external files

From a file describing a solid in a particular format (other than .obj or .off), we can create a .dat file containing the coordinates of the vertices, and another .dat file containing the tables of indices of the vertices on each face. These files can then be entered as parameters **sommets** and **faces** when using the PostScript instruction **run**.

In the example below, the files **sommets_nefer.dat** and **faces_nefer.dat** have been placed in the directory of the compiler.

7. Generating some new solids



```

1 \psset{unit=0.4}
2 \definecolor{AntiqueWhite}{rgb}{0.98,0.92,0.84}
3 \begin{pspicture}(-7,-9)(7,7)
4 \psset{lightsrc=30 -40 10}
5 \psset{viewpoint=50 -50 20 rtp2xyz,Decran=50}
6 \psSolid[object=new,fillcolor=AntiqueWhite,linewidth=0.5\
    pslinewidth,
8   faces={./faces_nefer.dat} run]%
9 \psSolid[object=new,fillcolor=red,linewidth=0.5\pslinewidth,
10  faces={./faces_nefer_levres.dat} run]%
11 \psSolid[object=new,fillcolor=black,
12   faces={./faces_nefer_sourcils.dat} run]%
13 \end{pspicture}
14 \hfill
15 \begin{pspicture}(-7,-9)(7,7)
16 \psset{lightsrc=-10 -40 -5,lightintensity=.5}
17 \psset{viewpoint=50 -80 10 rtp2xyz,Decran=50}
18 \psset{RotX=90,RotZ=30,sommets= (./sommets_nefer.dat) run}
19 \psSolid[object=new,fillcolor=AntiqueWhite,linewidth=0.5\
    pslinewidth,
20   grid,faces={./faces_nefer.dat} run]
21 \psSolid[object=new,fillcolor=red,linewidth=0.5\pslinewidth,grid,
22   faces={./faces_nefer_levres.dat} run]
23 \psSolid[object=new,fillcolor=black,
24   faces={./faces_nefer_sourcils.dat} run]
25 \end{pspicture}

```

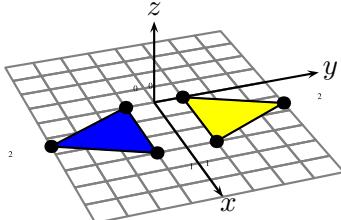
7.7. One- and two-sided solids

The contour of `face` is defined in the plane Oxy by

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

The edge of `face` is defined in the plane Oxy by the coordinates of its vertices, given in counterclockwise order by the parameter `base`:

7.7.1. Triangular ‘faces’

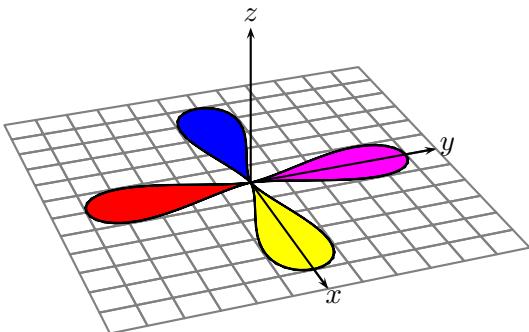


```

1 \psset{unit=0.4}
2 \psset{viewpoint=50 -20 30 rtp2xyz,Decran=50}
3 \begin{pspicture}(-5.5,-4.5)(7,3.5)
4 \psSolid[object=grille,base=-4 6 -4 4,action=draw,linecolor=gray](0,0,0)
5 \psSolid[object=face,fillcolor=yellow,action=draw*,incolor=blue,biface,base=0 0 3 0 1.5 3, num=all,show=all](0,1,0)
6 \psSolid[object=face,fillcolor=yellow,action=draw*,incolor=blue,base=0 0 3 0 1.5 3,num=all,show=all,biface,RotX=180](0,-1,0)
7 \axesIIID(0,0,0)(6,6,3)
8 \end{pspicture}

```

7.7.2. ‘face’ defined by a function



```

1 \psset{unit=0.45}
2 \psset{viewpoint=50 -20 30 rtp2xyz,Decran=50}
3 \def\BASE{0 10 360{/Angle ED 5 Angle cos dup mul mul
% x
4 3 Angle cos 3 exp Angle sin mul mul } for}% y
5 \begin{pspicture}(-7,-5.5)(9,6)
6 \defFunction[algebraic]{F}(t){5*(cos(t))^2
7 {3*(sin(t))*(cos(t))^3}{}
8 \psSolid[object=grille,base=-6 6 -6 6,action=draw,
9 linecolor=gray](0,0,0)
10 \psSolid[object=face,fillcolor=magenta,action=draw*,incolor=blue,biface,RotZ=90,
11 base=0 2 pi mul {F} CourbeR2+](0,0,0)
12 \psSolid[object=face,fillcolor=yellow,action=draw*,incolor=blue,biface,
13 base=0 2 pi mul {F} CourbeR2+](0,0,0)
14 \psSolid[object=face,fillcolor=yellow,action=draw*,incolor=blue,biface,RotY=180,
15 base=0 2 pi mul {F} CourbeR2+](0,0,0)
16 \psSolid[object=face,fillcolor=yellow,action=draw*,incolor=red,biface,RotY=180,RotZ=90,
17 base=0 2 pi mul {F} CourbeR2+](0,0,0)
18 \psSolid[object=ruban,h=3,base=x1 y1 x2 y2 x3 y3 ...xn yn,ngrid=n](0,0,0)%
19 \axesIIID(0,0,0)(6,6,5)
20 \end{pspicture}

```

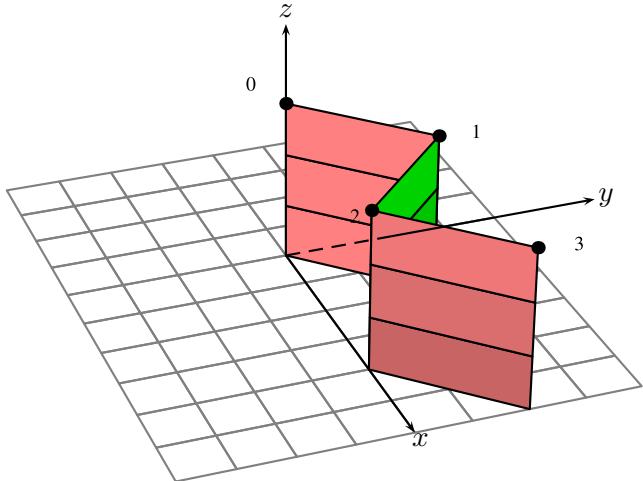
7.8. Solid strip

The strip is a folding screen positioned horizontally on the floor. The base of the folding screen is defined in the plane Oxy by the coordinates of its vertices by the parameter `base`:

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

7. Generating some new solids

7.8.1. A simple folding screen

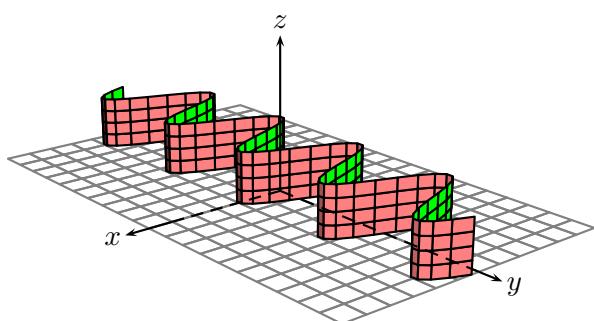


```

1 \psset{lightsrc=10 0 10,viewpoint=50 -20
         30 rtp2xyz,Decran=50,unit=0.75}
2 \begin{pspicture}(-5.5,-4.5)(7,5)
3 \psSolid[object=grille,base=-4 6 -4 4,
           action=draw,linecolor=gray](0,0,0)
4 \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)
10 \axesIIID(0,2,0)(6,6,4.5)
11 \end{pspicture}

```

7.8.2. A sinusoidal folding screen



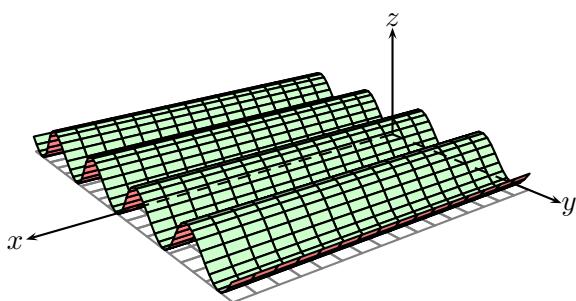
```

1 \psset{unit=0.35}
2 \begin{pspicture}(-10,-6)(12,8)
3 \defFunction[F]{t}{2 t 4 mul cos mul}{t 20 div
}{}
4 \psSolid[object=grille,base=-6 6 -10 10,action=
           draw,linecolor=gray](0,0,0)
5 \psSolid[object=ruban,h=2,fillcolor=red!50,
           resolution=72,
           base=-200 200 {F} CourbeR2+, %% -200 5
           200 {/Angle ED 2 Angle 4 mul cos mul
           Angle 20 div } for,
           ngrid=4](0,0,0)
9 \axesIIID(5,10,0)(7,11,6)
10 \end{pspicture}

```

7.8.3. A corrugated surface

This is the same object as before with an additional rotation of 90° around Oy .



```

1 \psset{unit=0.4}
2 \begin{pspicture}(-14,-7)(8,5)
3 \defFunction[F](t){t 4 mul cos}{t 20 div}{}{}
4 \psSolid[object=grille,base=0 16 -10 10,action=
   draw,linecolor=gray](0,0,0)
5 \psSolid[object=ruban,h=16,fillcolor=red!50,
   RotY=90,incolor=green!20,
   resolution=72,
7   base=-200 200 {F} CourbeR2+,
8   ngrid=16](0,0,1)
9 \axesIID(16,10,0)(20,12,6)
10 \end{pspicture}

```

We can then imagine it to be like a corrugated iron roof of a shed.

7.8.4. An asteroidal folding screen: version 1

The contour of the folding screen is defined within a loop:

```

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

```

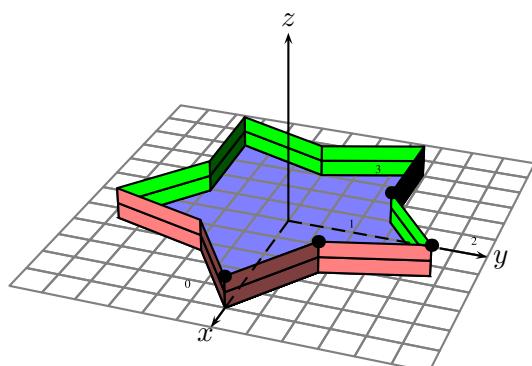
the blueish surface on the bottom is defined with the help of a polygon, where the vertices are calculated by the command

`\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)

```



```

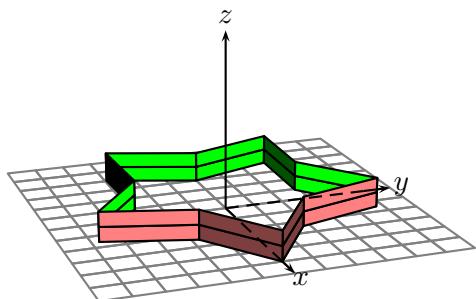
1 \psset{unit=0.45}
2 \begin{pspicture}(-9,-5)(9,7)
3 \multido{\iA=0+72,\iB=36+72,\i=0+1}{6}{%
4   \psPoint(\iA\space cos 5 mul,\iA\space sin 5 mul
   ,0){P\i}
5   \psPoint(\iB\space cos 3 mul,\iB\space sin 3 mul
   ,0){p\i}
}
7 \pspolygon[fillstyle=solid,fillcolor=blue!50] (P0)(p0)
(P1)(p1)(P2)(p2)(P3)(p3)(P4)(p4)(P5)(p5)
8 \defFunction[F](t){t cos 5 mul}{t sin 5 mul}{}{}
9 \defFunction[G](t){t 36 add cos 3 mul}{t 36 add sin 3
mul}{}{}
10 \psSolid[object=grille,base=-6 6 -6 6,action=draw,
   linecolor=gray](0,0,0)
11 \psSolid[object=ruban,h=1,fillcolor=red!50,
   base=0 72 360 {/Angle exch def Angle F Angle G}
   for,
13   num=0 1 2 3,show=0 1 2 3,ngrid=2](0,0,0)
14 \axesIID(5,5,0)(6,6,6)
15 \end{pspicture}

```

7. Generating some new solids

7.8.5. An asteroidal folding screen: version 2

The bottom of the pot is defined by the object `face` with the option `biface`:



```

1 \psset{unit=0.4}
2 \begin{pspicture}(-9,-4)(9,7)
3 \defFunction{F}(t){t cos 5 mul}{t sin 5 mul}{}
4 \defFunction{G}(t){t 36 add cos 3 mul}{t 36 add sin 3
   mul}{}
5 \psSolid[object=face,fillcolor=blue!50,biface,
6   base=0 72 360{/Angle exch def Angle F Angle G}
   for,](0,0,0)
7 \psSolid[object=grille,base=-6 6 -6 6,action=draw,
   linecolor=gray](0,0,0)
8 \psSolid[object=ruban,h=1,fillcolor=red!50,
9   base=0 72 360{/Angle exch def Angle F Angle G}
   for,
10  ngrid=2](0,0,0)
11 \axesIIID(5,5,0)(6,6,6)
12 \end{pspicture}
```

7.9. Solid rings

This paragraph discusses the cylindric rings. Within the macro `\psSolid`, this object is passed with the option: `object=anneau`, that comes with 3 parameters:

- the inner radius `r=1.5` (value by default);
- the outer radius `R=4` (value by default);
- the height `h=6` (value by default).

The argument `ngrid` defines the number of sections used to make a complete rotation of 360 degrees. Its default value is 24.

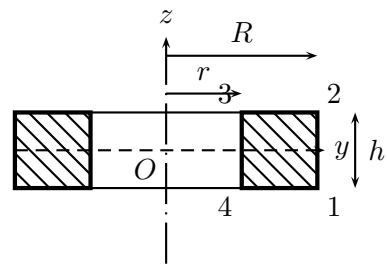
The section of the ring, whose shape is *rectangular* was chosen as default, and can be redesigned by the user. We will discuss different examples of sections for rings.

7.9.1. Predefined command: the ring with a rectangular section

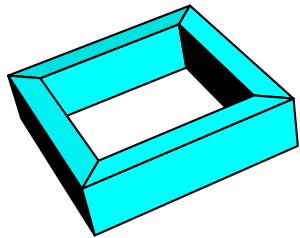
This section is defined in the plane Oyz , it is parameterized with the triple (r, R, h) . The values of the outer radius R , inner radius r and the height h are passed in the macro `\psSolid`. By default, one has a ring with a variable rectangular section, and the definition takes place at the time of the transmission of the values (r, R, h) into the options of `\psSolid`.

If the user redefines the TeX macro `\Section` with some numeric values instead of the parameters r , R and h , then the ring won't be variable anymore and it is not necessary to transmit the values r , R , and h into the options of `\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)
}
```



7. Generating some new solids

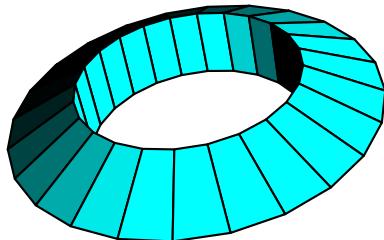


```

1 \psset{unit=0.5}
2 \begin{pspicture}(-5,-4)(5,4)
3 \psset[pst-solides3d]{viewpoint=50 20 40 rtp2xyz,
4 Decran=25,lightsrc=10 20 20}
5 \psSolid[object=anneau,fillcolor=cyan,
6 h=3,R=8,r=6,ngrid=4,RotX=10](0,0,0)
7 \end{pspicture} \\
8 \begin{pspicture}(-5,-4)(5,4)
9 \psset[pst-solides3d]{viewpoint=50 -20 -40 rtp2xyz,
10 Decran=25,lightsrc=-10 -20 -20}
11 \psSolid[object=anneau,
12 fillcolor=yellow,h=3,R=8,r=6,
13 RotX=90,RotZ=10](0,0,0)
14 \end{pspicture}
```

7.9.2. Example 1: a simple ring with a triangular section

Below is a very simple ring with a fixed triangular section. The section is defined by 3 points $(6, -2)$, $(10, 0)$ and $(6, 2)$ within the option `section` of `\psSolid`.

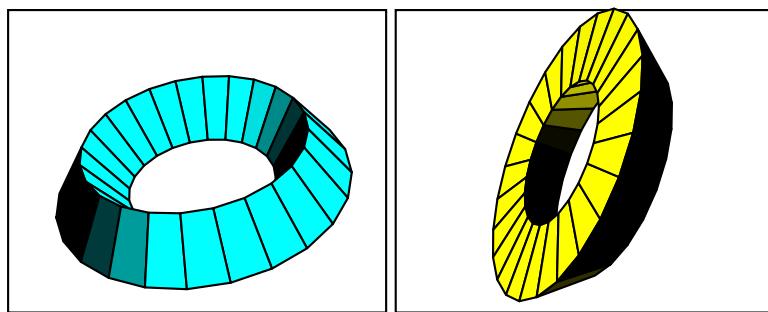
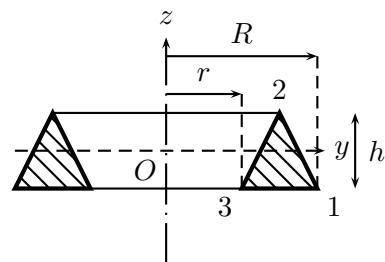


```

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

7.9.3. Example 2: a ring with a variable triangular section

```
\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)
}
```

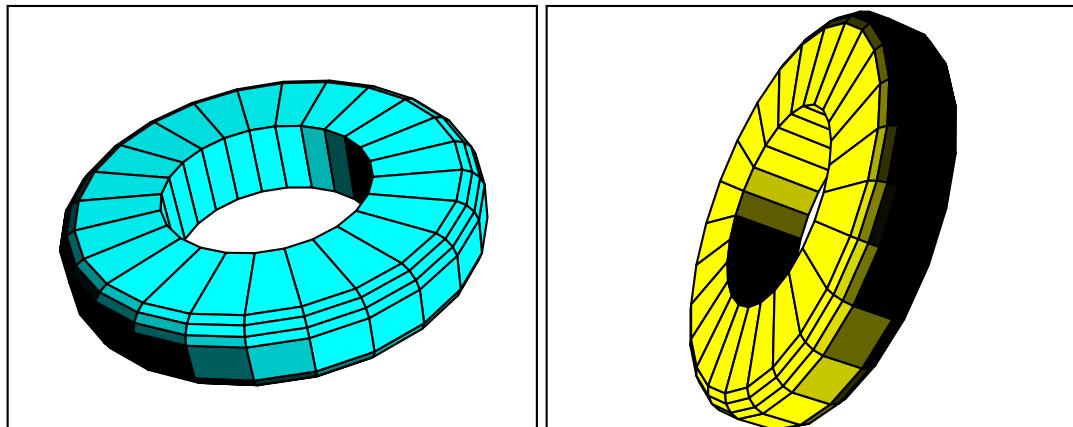
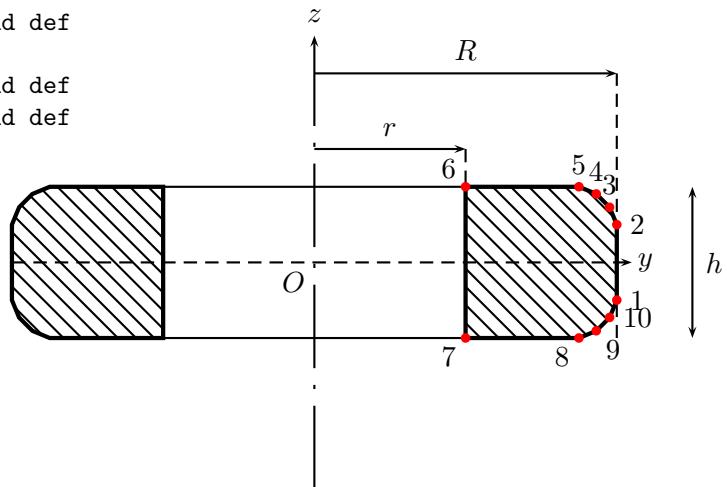


```
\psSolid[object=anneau,section=\SectionTriangulaire,%
        fillcolor=cyan,h=3,R=8,r=4,RotX=10](0,0,0)
\psSolid[object=anneau,section=\SectionTriangulaire,%
        fillcolor=yellow,h=3,R=8,r=4,RotX=-90,RotZ=10](0,0,0)
```

7. Generating some new solids

7.9.4. Example 3: a ring with a “tyre”-like section: cylindric ring with chamfered edges

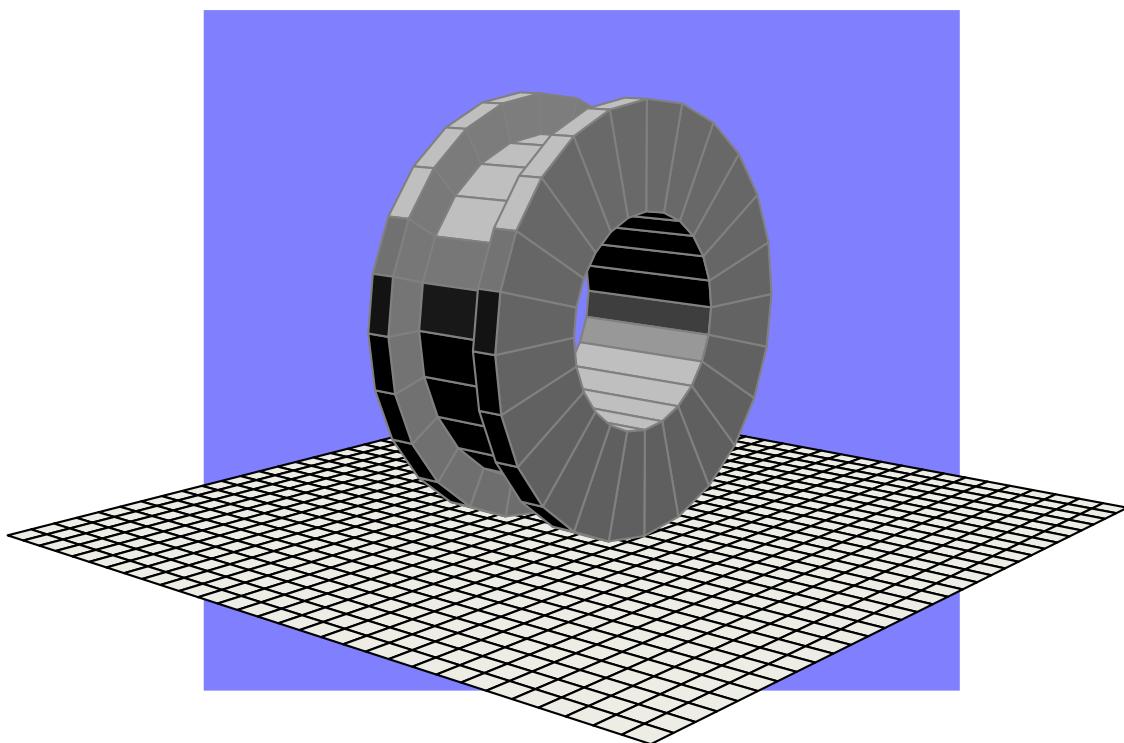
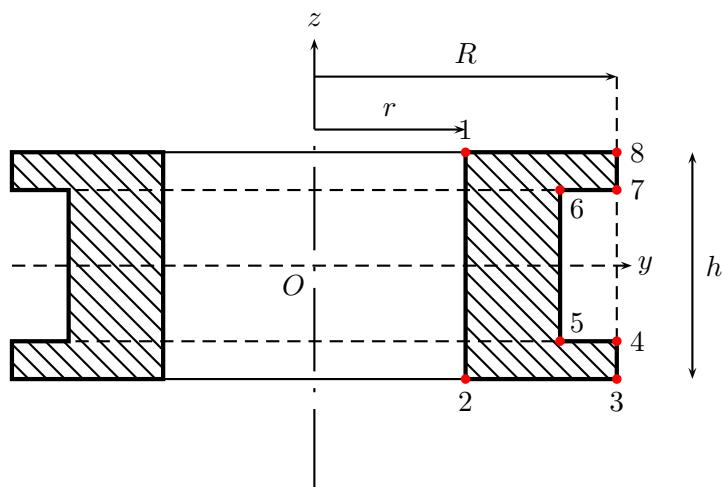
```
\renewcommand{\SectionPneu{
/m {90 4 div} bind def
/Scos {m cos 2 m mul cos add 3 m mul cos add} bind def
/Z0 {h 4 div} bind def
/c {Z0 Scos div} bind def
/Z1 {Z0 c m cos mul add} bind def
/Z2 {Z1 c m 2 mul cos mul add} bind def
/R1 {R c m sin mul sub} bind def
/R2 {R1 c m 2 mul sin mul sub} bind 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]%
```

7.9.5. Example 4: an empty bobbin

```
\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]%
```

7.9.6. Some other rings

Three other examples are available on the website:

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

7. Generating some new solids

7.10. Generalization of the notion of a cylinder and a cone

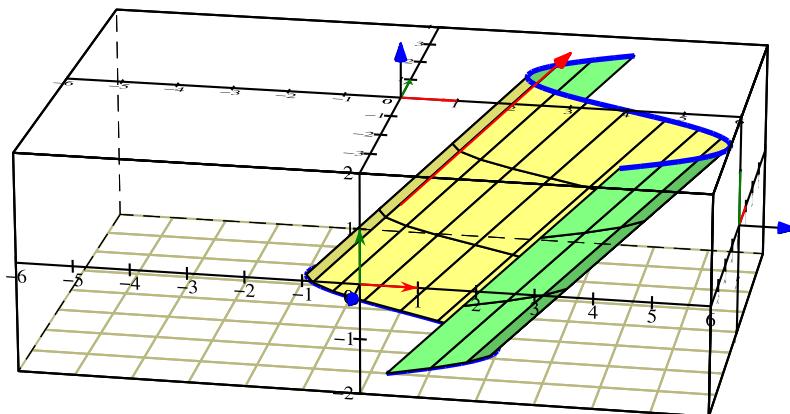
7.10.1. Cylinder or cylindric area

This paragraph generalizes the notion of a cylinder, or a cylindric area¹. A *routing curve* has to be defined by a function and the direction of the *cylinder* axis needs to be arranged. In the example below the routing curve is sinusoidal, situated in the plane $z = -2$:

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

The direction of the cylinder is defined by the components of a vector `axe=0 1 1`. The drawing calls `object=cylindre` which in addition to the usual parameters—which is the height `h=4`—is about the **length of the generator** and not of the distance between the two base planes, and needs to define the routing curve `function=G1` and the interval of the variable `t range=-3 3`.

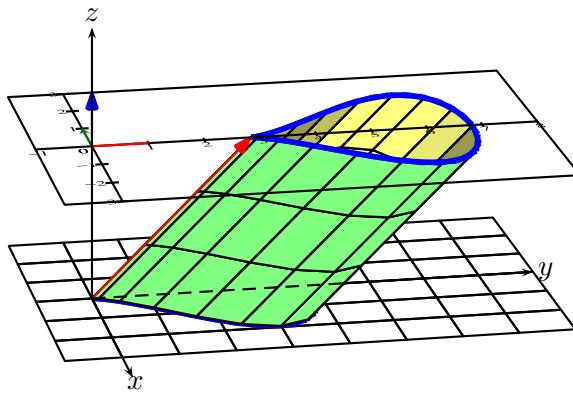
```
\psSolid[object=cylindre,
h=4,function=G1,
range=-3 3,
ngrid=3 16,
axe=0 1 1,
incolor=green!50,
fillcolor=yellow!50]
```



In the following example, before drawing the horizontal planes, we calculate the distance between these two planes.

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

¹This was written by Maxime CHUPIN, as a result of a question on the list <http://melusine.eu.org/cgi-bin/mailman/listinfo/syracuse>



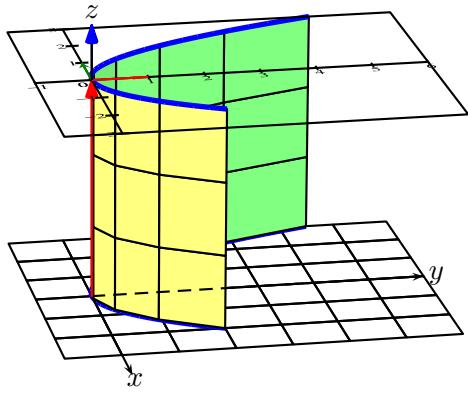
axe=0 1 1

```

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

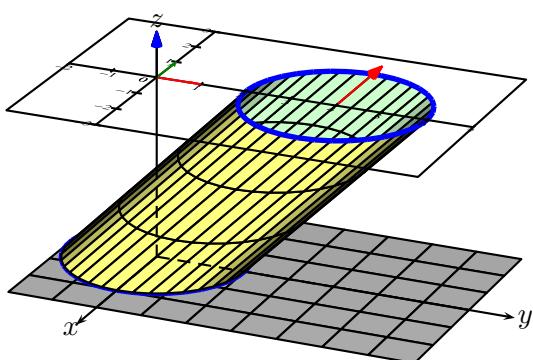
```

7. Generating some new solids



```

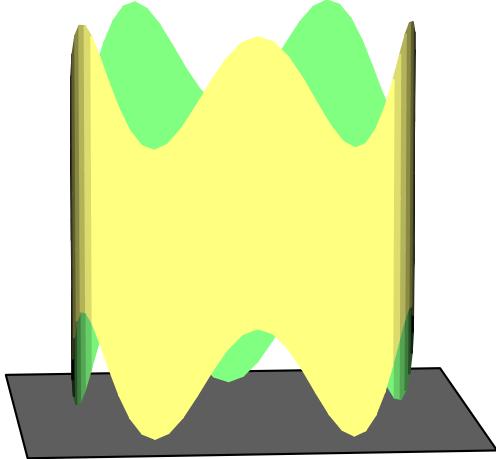
1 \psset{unit=0.75,lightsrc=viewpoint,
2   viewpoint=100 -10 20 rtp2xyz,Decran=100}
3 \begin{pspicture}(-1.5,-3)(6.5,6)
4 \psSolid[object=grille,base=-3 3 -1 6,action=draw]
5 \defFunction[algebraic]{G5}(t)
6 {t}{0.5*t^2}{0}
7 \defFunction[algebraic]{G6}(t)
8 {t}{0.5*t^2}{4}
9 \psSolid[object=courbe,function=G5,
10  range=-3 2,r=0,linecolor=blue,
11  linewidth=2pt]
12 \psSolid[object=cylindre,
13  range=-3 2,h=4,
14  function=G5,
15  axe=0 0 1, % valeur par d\{e}faut
16  incolor=green!50,
17  fillcolor=yellow!50,
18  ngrid=3 8]
19 \psSolid[object=courbe,function=G6,
20  range=-3 2,r=0,linecolor=blue,
21  linewidth=2pt]
22 \axesIIID(0,4.5,0)(4,6,5)
23 \psSolid[object=vecteur,
24  linecolor=red,args=0 0 4]
25 \psSolid[object=plan,action=draw,
26  definition=equation,
27  args={[0 0 1 -4] 90},
28  base=-1 6 -3 3,planmarks,showBase]
29 \end{pspicture}
```



```

1 \psset{unit=0.75,lightsrc=viewpoint,
2   viewpoint=100 -10 20 rtp2xyz,Decran=100}
3 \begin{pspicture}(-3.5,-3)(6.5,6)
4 \psset{lightsrc=viewpoint,viewpoint=100 45 45,
      Decran=100}
5 \psSolid[object=grille,base=-3 3 -2 7,fillcolor=
      gray!30]
6 \defFunction[algebraic]{G7}(t)
7 {2*cos(t)}{2*sin(t)}{0}
8 \defFunction[algebraic]{G8}(t)
9 {2*cos(t)}{2*sin(t)+4}{4}
10 \psSolid[object=courbe,function=G7,
11  range=0 6.28,r=0,
12  linecolor=blue,linewidth=2pt]
13 \psSolid[object=cylindre,
14  range=0 6.28,h=5.65685,
15  function=G7,axe=0 1 1,
16  incolor=green!20,
17  fillcolor=yellow!50,
18  ngrid=3 36]
19 \psSolid[object=courbe,function=G8,
20  range=0 6.28,r=0,linecolor=blue,
21  linewidth=2pt]
22 \axesIIID(2,4.5,2)(4,8,5)
23 \psSolid[object=vecteur,
24  linecolor=red,args=0 1 1](0,4,4)
25 \psSolid[object=plan,action=draw,
26  definition=equation,
27  args={[0 0 1 -4] 90},
28  base=-2 7 -3 3,planmarks,showBase]
29 \end{pspicture}
```

Note: The routing curve can be any curve and need not necessarily be a plane horizontal.



```

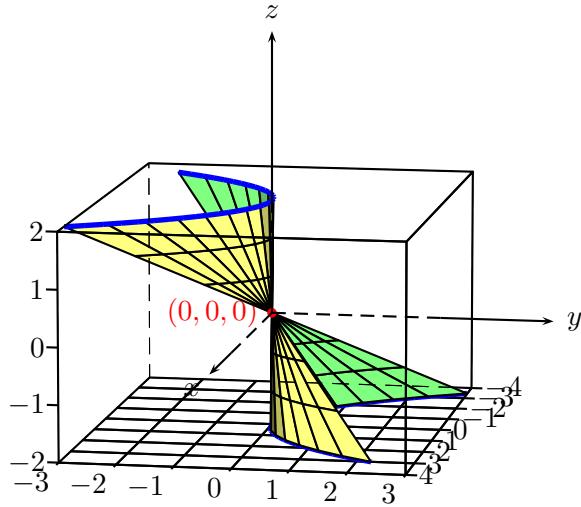
1 \begin{pspicture}(-3.5,-2)(4,5)
2 \psset{unit=0.75,lightsrc=viewpoint,viewpoint=100
-5 10 rtp2xyz,Decran=100}
3 \psSolid[object=grille,base=-4 4 -4 4 ,ngrid=8.
8.](0,0,-1)
4 \defFunction[algebraic]{G9}(t)
5 {3*cos(t)}{3*sin(t)}{1*cos(5*t)}
6 \psSolid[object=cylindre,
7 range=0 6.28,h=5,function=G9,
8 axe=0 0 1,incolor=green!50,
9 fillcolor=yellow!50,
10 ngrid=4 72,grid]
11 \end{pspicture}
```

7.10.2. Cone or conic area

This paragraph generalizes the notion of a cone, or a conic area². A *routing* curve needs to be defined by a function which defines the base of the cone, and the vertex of the *cone* which is by default `origine=0 0 0`. The parts above and below the cone are symmetric concerning the vertice. In the example below, the routing curve is a parabolic arc, situated in the plane $z = -2$.

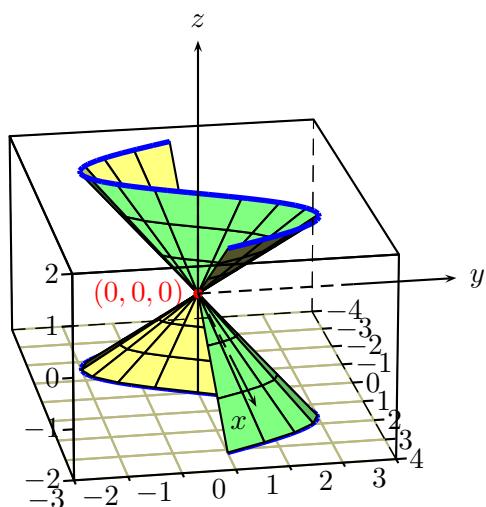
²This was written by Maxime CHUPIN, as the result of a question on the list <http://melusine.eu.org/cgi-bin/mailman/listinfo/syracuse>

7. Generating some new solids



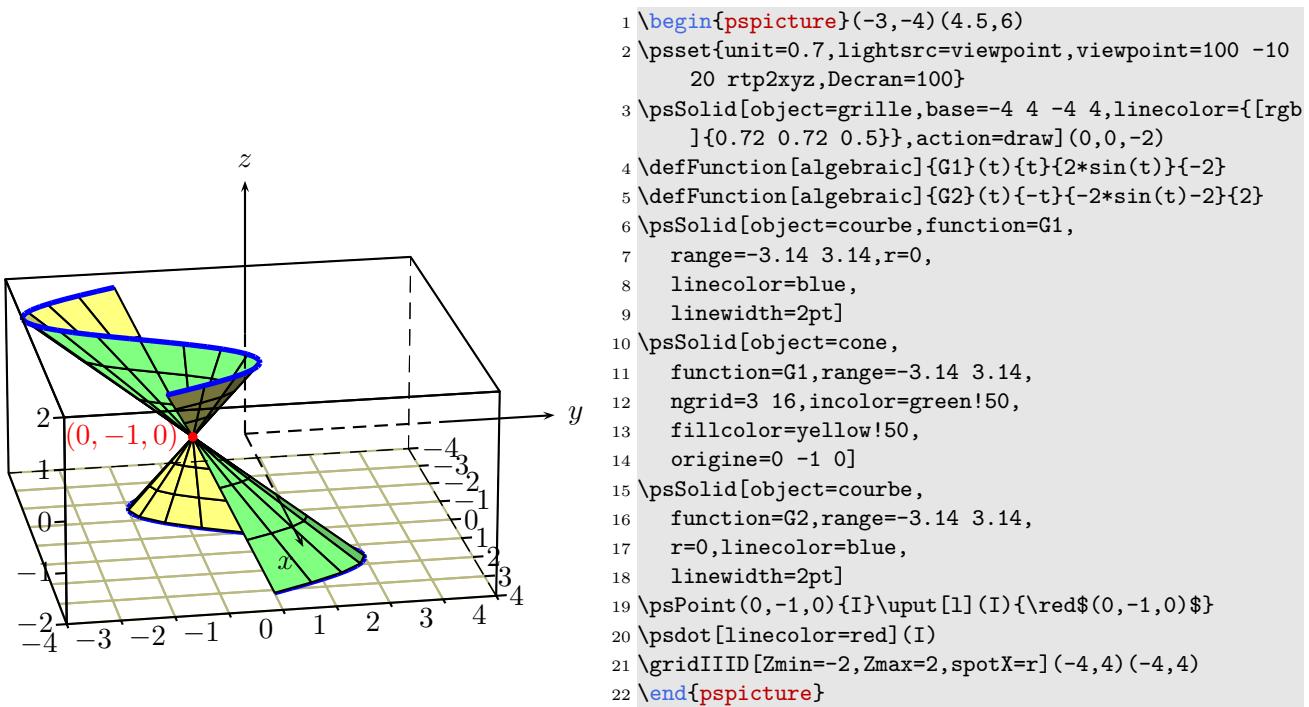
```

1 \begin{pspicture}(-3,-4)(4.5,6)
2 \psset{unit=0.75,lightsrc=viewpoint,viewpoint=100 10
      10 rtp2xyz,Decran=100}
3 \psSolid[object=grille,base=-4 4 -3 3,action=draw
          ](0,0,-2)
4 \defFunction[algebraic]{G1}(t){t}{0.25*t^2}{-2}
5 \defFunction[algebraic]{G2}(t){-t}{-0.25*t^2}{2}
6 \psSolid[object=courbe,function=G1,
7   range=-3.46 3,r=0,
8   linecolor=blue,linewidth=2pt]
9 \psSolid[object=cone,function=G1,
10  range=-3.46 3,ngrid=3 16,
11  incolor=green!50,
12  fillcolor=yellow!50,
13  origine=0 0 0]
14 \psSolid[object=courbe,
15   function=G2,range=-3.46 3,
16   r=0,linecolor=blue,
17   linewidth=2pt]
18 \psPoint(0,0,0){I}
19 \uput[1](I){\red$(0,0,0)$}
20 \psdot[linecolor=red](I)
21 \gridIIID[Zmin=-2,Zmax=2,spotX=r](-4,4)(-3,3)
22 \end{pspicture}
```



```

1 \begin{pspicture}(-3,-4)(4.5,6)
2 \psset{unit=0.7,lightsrc=viewpoint,viewpoint=100 -10
      20 rtp2xyz,Decran=100}
3 \psSolid[object=grille,base=-4 4 -3 3,
4   linecolor={[rgb]{0.72 0.72 0.5}},action=draw
          ](0,0,-2)
5 \defFunction[algebraic]{G1}(t){t}{2*sin(t)}{-2}
6 \defFunction[algebraic]{G2}(t){-t}{-2*sin(t)}{2}
7 \psSolid[object=courbe,function=G1,
8   range=-3.14 3.14,r=0,
9   linecolor=blue,
10  linewidth=2pt]
11 \psSolid[object=cone,function=G1,
12  range=-3.14 3.14,ngrid=3 16,
13  incolor=green!50,
14  fillcolor=yellow!50,
15  origine=0 0 0]
16 \psSolid[object=courbe,
17   function=G2,range=-3.14 3.14,
18   r=0,linecolor=blue,
19   linewidth=2pt]
20 \psPoint(0,0,0){I} \uput[1](I){\red$(0,0,0)$}
21 \psdot[linecolor=red](I)
22 \gridIIID[Zmin=-2,Zmax=2,spotX=r](-4,4)(-3,3)
23 \end{pspicture}
```



Note: For the cones as well, the routing curve can be any curve and need not necessarily be a plane horizontal curve, as the following example, written by Maxime CHUPIN, will show.

http://melusine.eu.org/lab/bpst/pst-solides3d/cone/cone-dir_02.pst

7.11. Parameterised surfaces

7.11.1. The method

The parameterised surfaces are setup as $[x(u, v), y(u, v), z(u, v)]$ and administered thanks to the macro `\psSolid` by the option `object=surfaceparametree` and defined either in *Reverse Polish Notation (RPN)*:

```
\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)}
```

or in *algebraic notation*:

```
\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)}
```

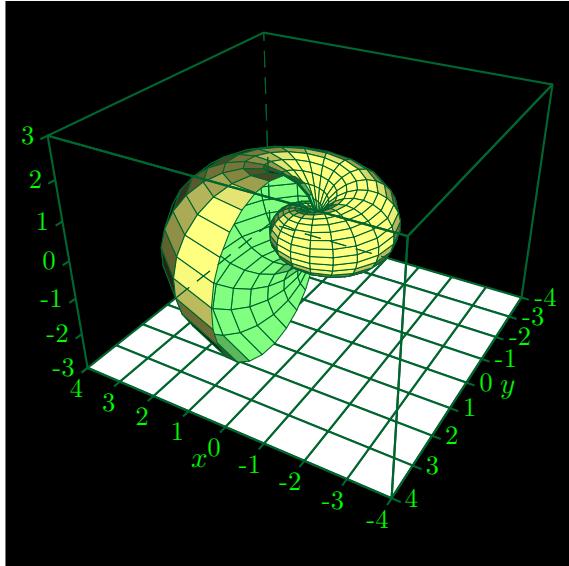
The range for the values of u and v are defined within the option `range=umin umax vmin vmax`.

7. Generating some new solids

The drawing of the function is activated with `function=name`, this name is implied when the parametric equations are written: `\defFunction{name}...`

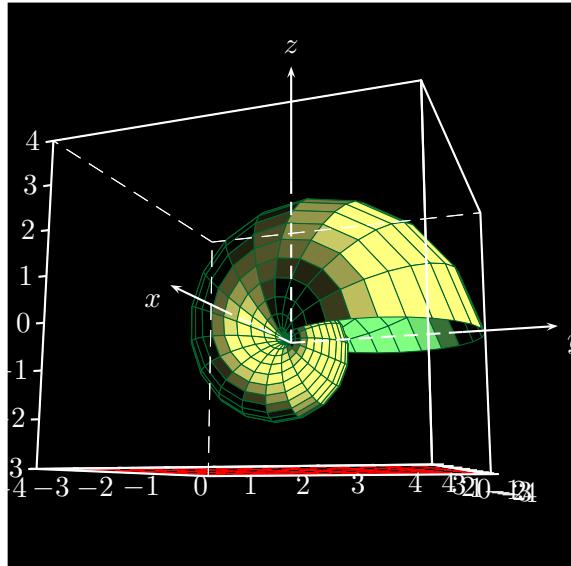
Any other choice of u and v are accepted. Let's remind that the argument of `Sin` and `Cos` must be in radians those of `sin` and `cos` in degrees if *RPN* is used. Within the algebraic notation, the argument is in radians.

7.11.2. Example 1: a sea shell



```

1 \psset{unit=0.75}
2 \begin{pspicture}(-5.5,-6)(4.5,4)
3 \psframe*(-5.5,-6)(4.5,4)
4 \psset[pst-solides3d]{viewpoint=20 120 30 rtp2xyz,
5 Decran=15,lightsrc=-10 15 10}
6 % Parametric Surfaces
7 \psSolid[object=grille,base=-4 4 -4 4,
8 action=draw*,linecolor={[cmyk]{1,0,1,0.5}}]
9 (0,0,-3)
10 \defFunction{shell}(u,v)
11 {1.2 v exp u Sin dup mul v Cos mul mul}
12 {1.2 v exp u Sin dup mul v Sin mul mul}
13 {1.2 v exp u Sin u Cos mul mul}
14 \psSolid[object=surfaceparametree,
15 linecolor={[cmyk]{1,0,1,0.5}},
16 base=0 pi pi 4 div neg 5 pi mul 2 div,
17 fillcolor=yellow!50,incolor=green!50,
18 function=shell,linewidth=0.5\pslinewidth,ngrid
=25]%
19 \psSolid[object=parallelepiped,a=8,b=8,c=6,
20 action=draw,linecolor={[cmyk]{1,0,1,0.5}}]%
21 \quadrillage
22 \end{pspicture}
```

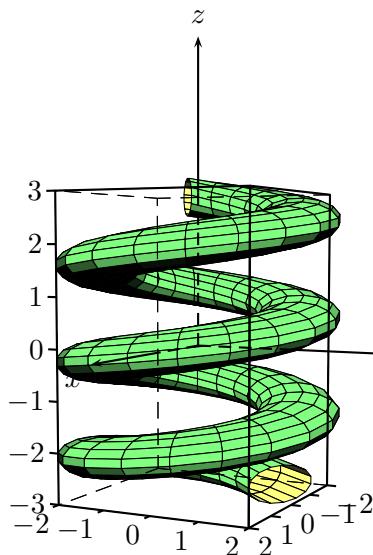


```

1 \psset{unit=0.75}
2 \begin{pspicture}(-5,-4)(5,6)
3 \psframe*(-5,-4)(5,6)
4 \psset[pst-solides3d]{viewpoint=20 20 -10 rtp2xyz,
5 Decran=15,lightsrc=5 10 2}
6 % Parametric Surfaces
7 \psSolid[object=grille,base=-4 4 -4 4,
8 action=draw*,linecolor=red](0,0,-3)
9 \defFunction[algebraic]{shell}(u,v)
10 {1.21^v*(sin(u)*cos(u))}
11 {1.21^v*(sin(u)^2*sin(v))}
12 {1.21^v*(sin(u)^2*cos(v))}
13 %% \defFunction[shell](u,v)
14 %% {1.2 v exp u Sin u Cos mul mul}
15 %% {1.2 v exp u Sin dup mul v Sin mul mul}
16 %% {1.2 v exp u Sin dup mul v Cos mul mul}
17 \psSolid[object=surfaceparametree,
18 linecolor={[cmyk]{1,0,1,0.5}},
19 base=0 pi pi 4 div neg 5 pi mul 2 div,
20 fillcolor=green!50,incolor=yellow!50,
21 function=shell,linewidth=0.5\pslinewidth,
22 ngrid=25]%
23 \white%
24 \gridIIID[Zmin=-3,Zmax=4,linecolor=white,
25 QZ=0.5](-4,4)(-4,4)
26 \end{pspicture}

```

7.11.3. Example 2: a helix



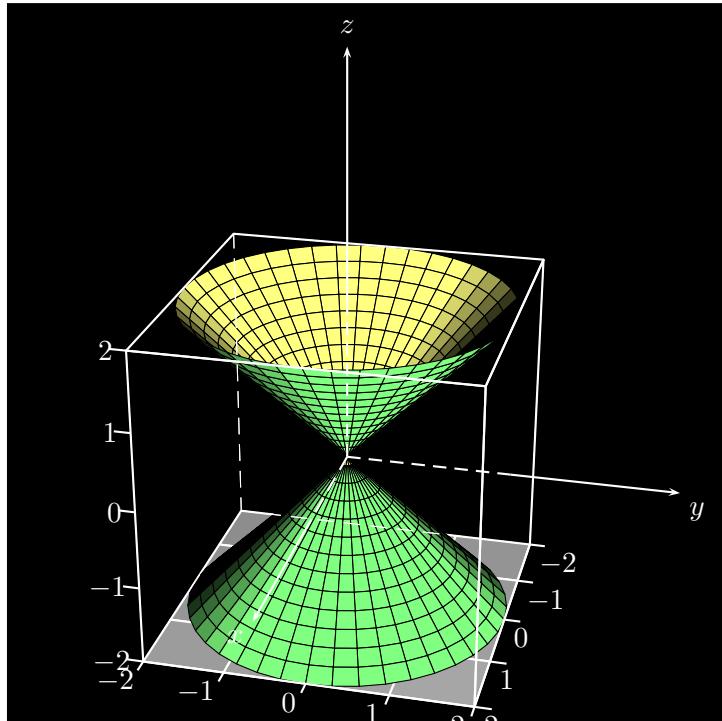
```

1 \psset{unit=0.75}
2 \begin{pspicture}(-3,-4)(3,6)
3 \psset[pst-solides3d]{viewpoint=20 10 2,Decran=20,
4 lightsrc=20 10 10}
5 % Parametric Surfaces
6 \defFunction[helix](u,v)
7 {1 .4 v Cos mul sub u Cos mul 2 mul}
8 {1 .4 v Cos mul sub u Sin mul 2 mul}
9 {.4 v Sin mul u .3 mul add}
10 \psSolid[object=surfaceparametree,linewidth=0.5\pslinewidth,
11 base=-10 10 0 6.28,fillcolor=yellow!50,incolor=green!50,
12 function=helix,
13 ngrid=60 0.4]%
14 \gridIIID[Zmin=-3,Zmax=3](-2,2)(-2,2)
15 \end{pspicture}

```

7.11.4. Example 3: a cone

7. Generating some new solids



```
1 \psset{unit=0.5}
2 \begin{pspicture}(-9,-7)(10,12)
3 \psframe*(-9,-7)(10,12)
4 \psset[pst-solides3d]{
5   viewpoint=20 5 10,
6   Decran=50,lightsrc=20 10 5}
7 \psSolid[
8   object=grille,base=-2 2 -2 2,
9   linecolor=white](0,0,-2)
10 % Parametric Surfaces
11 \defFunction[cone](u,v)
12 {u v Cos mul}{u v Sin mul}{u}
13 \psSolid[object=surfaceparametre,
14   base=-2 2 0 2 pi mul,
15   fillcolor=yellow!50,
16   incolor=green!50,function=cone,
17   linewidth=0.5\pslinewidth,
18   ngrid=25 40]%
19 \psset[linecolor=white]\white
20 \gridIIID[Zmin=-2,Zmax=2]
21 (-2,2)(-2,2)
22 \end{pspicture}
```

7.11.5. An advised website

You will find on the website:

<http://k3dsurf.sourceforge.net/>

an excellent software to represent surfaces with numerous examples of parameterised surfaces and others.

CHAPTER 8

Surfaces defined by a function of the form $z = f(x, y)$

8.1. Presentation

The command has the following form:

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

with the same options which apply to solids, and these additional ones:

- The surface grid is defined by the parameter `ngrid=n1 n2`, which has these specifics:
 - If `n1` and/or `n2` are integers, the number(s) represent(s) the number of grids following Ox and/or Oy .
 - If `n1` and/or `n2` are decimals, the number(s) represent(s) the incrementing steps following Ox and/or Oy .
 - If `ngrid=n`, with only one parameter value, the number of grids, or the incrementing steps, are identical on both axes.
- `algebraic`: this option allows you to write the function in algebraic notation; `pstricks.pro` meanwhile contains the code `AlgToPs` from Dominique RODRIGUEZ, which allows this notation and which is included in the `pstricks-add.pro` file. This version of `pstricks` is provided with `pst-solides3d`. If necessary, you must load the `pstricks-add` package in the document preamble.
- `grid`: by default the grid is activated. If the option `grid` is used, the grid will be deactivated!
- `axesboxed`: this option allows you to draw the 3D coordinate axes in a semi-automatic way, but because of the need to specify the limits of z by hand this option is deactivated by default:
 - `Zmin`: minimum value;
 - `Zmax`: maximum value;

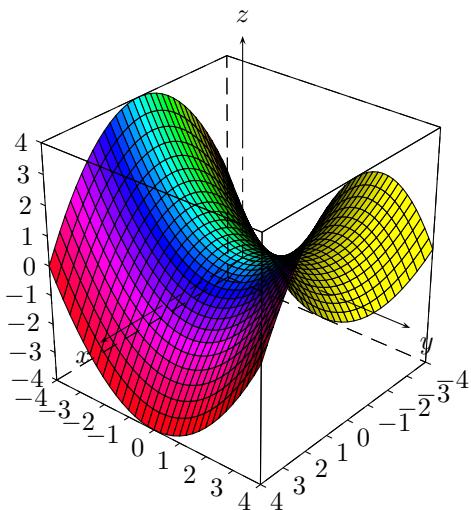
8. Surfaces defined by a function of the form $z = f(x, y)$

- **QZ:** allows a vertical shift of the coordinate axes with the value `QZ=value`;
- **spotX:** alters the placing of the x -axis tick values at the end of ticks, if the default behaviour is unsatisfactory. The positioning can be altered with the command `\uput [angle](x,y){ticklabel}`;
- **spotY:** is similar;
- **spotZ:** likewise.

If the option `axesboxed` doesn't meet your needs, it is possible to adapt the following command, which is appropriate for the first example:

```
\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[d1](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}
}
```

8.2. Example 1: a saddle

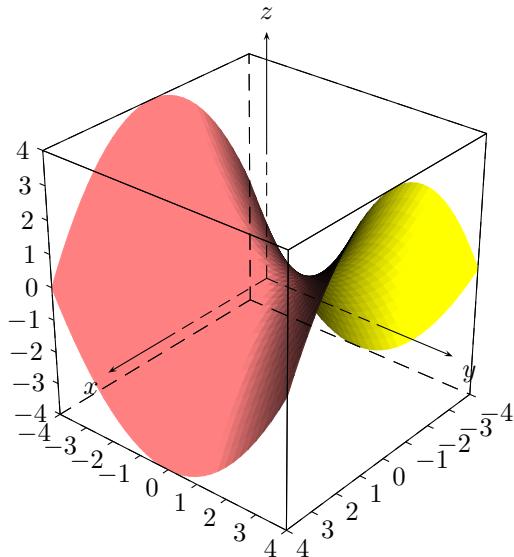


```

1 \psset{unit=0.45}
2 \psset{viewpoint=50 40 30 rtp2xyz,Decran=50}
3 \psset{lightsrc=viewpoint}
4 \begin{pspicture}(-7,-8)(7,8)
5 \psSurface[ngrid=.25 .25,incolor=yellow,
6 linewidth=0.5\pslinewidth,axesboxed,
7 algebraic,hue=0 1](-4,-4)(4,4)%
8 ((y^2)-(x^2))/4 }
9 \end{pspicture}
```

8.3. Example 2: a saddle without a grid

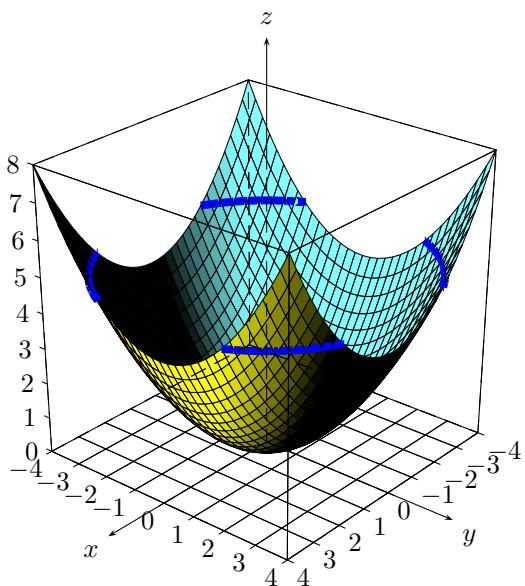
The grid lines are suppressed, when using in the option: `grid`.



```

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

8.4. Example 3: a paraboloid

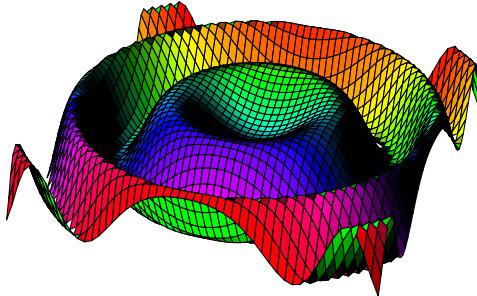


```

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

8. Surfaces defined by a function of the form $z = f(x, y)$

8.5. Example 4: a sinusoidal wave



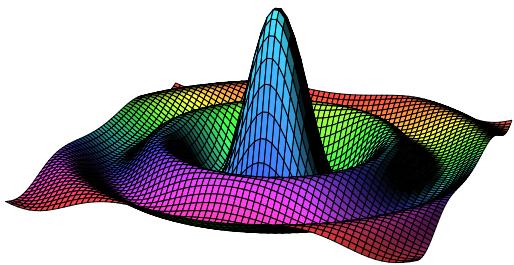
```

1 \psset{unit=0.35}
2 \psset{lightsrc=30 -10 10}
3 \psset{viewpoint=50 20 30 rtp2xyz,Decran=70}
4 \begin{pspicture}(-11,-8)(7,8)
5 \psSurface[ngrid=.2 .2,algebraic,Zmin=-1,Zmax=1,
6 linewidth=0.5\pslinewidth,spotX=r,spotY=d,
7 spotZ=l,
8 hue=0 1](-5,-5)(5,5){%
9   sin((x^2+y^2)/3)
10 \end{pspicture}

```

8.6. Example 5: another sinusoidal wave

In this example we show how to colour the faces, each with a different coloration, directly using PostScript code.



```

1 \psset{unit=0.25}
2 \psset{lightsrc=30 -10 10}
3 \psset{viewpoint=100 20 20 rtp2xyz,Decran=80}
4 \begin{pspicture}(-15,-10)(7,12)
5 \psSurface[ngrid=0.4 0.4,algebraic,Zmin=-2,Zmax=10,QZ
=4,
6 linewidth=0.25\pslinewidth,
7 fcol=0 1 4225
8 {/iF ED iF [iF 4225 div 0.75 1] (
9   sethsbcolor) astr2str} for
10 ](-13,-13)(13,13){%
11   10*sin(sqrt((x^2+y^2)))/(sqrt(x^2+y^2))
12 \end{pspicture}

```

8.7. Example 6: a hyperbolic paraboloid with the equation $z = xy$

In this example we combine the graph of the surface and the curves of intersection of the paraboloid with the planes $z = 4$ and $z = -4$. In this case we use `\psSolid[object=courbe]`.

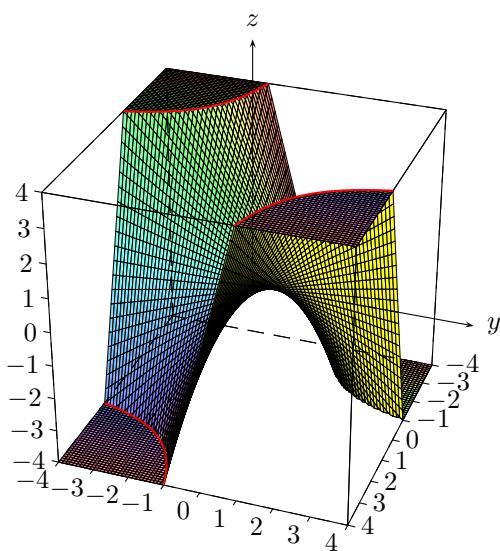
```

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

```

You will note the use of the functions `min` and `max`, which return the minimum and the maximum, respectively, of two values.

8.8. Example 7: a surface with the equation $z = xy(x^2 + y^2)$

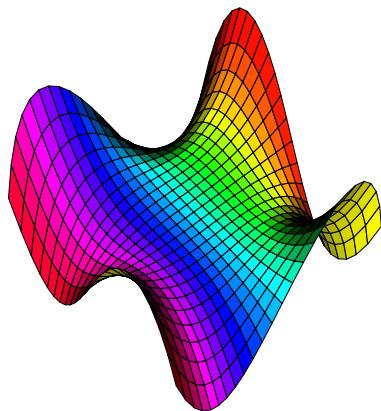


```

1 \psset{unit=0.5}
2 \psset{viewpoint=50 20 30 rtp2xyz,Decran=50}
3 \psset{lightsrc=viewpoint,linewidth=0.5\pslinewidth}
4 \begin{pspicture}(-7,-8)(7,8)
5 \psSolid[object=datfile,file=../paraboloid,hue=0 1 0.5
           1,incolor=yellow]
6 \gridIIID[Zmin=-4,Zmax=4,spotX=r](-4,4)(-4,4)
7 \defFunction{F}(t){t}{4 t div 4 min}{4}
8 \psSolid[object=courbe,range=1 4,r=0,
9   linecolor=red,linewidth=2\pslinewidth,
10  function=F]
11 \defFunction{G}(t){t}{4 t div -4 max}{4}
12 \psSolid[object=courbe,range=-1 -4,r=0,
13   linecolor=red,linewidth=2\pslinewidth,
14  function=G]
15 \defFunction{H}(t){t neg}{4 t div -4 max}{-4}
16 \psSolid[object=courbe,range=-1 -4,r=0,
17   linecolor=red,linewidth=2\pslinewidth,
18  function=H]
19 \end{pspicture}

```

8.8. Example 7: a surface with the equation $z = xy(x^2 + y^2)$



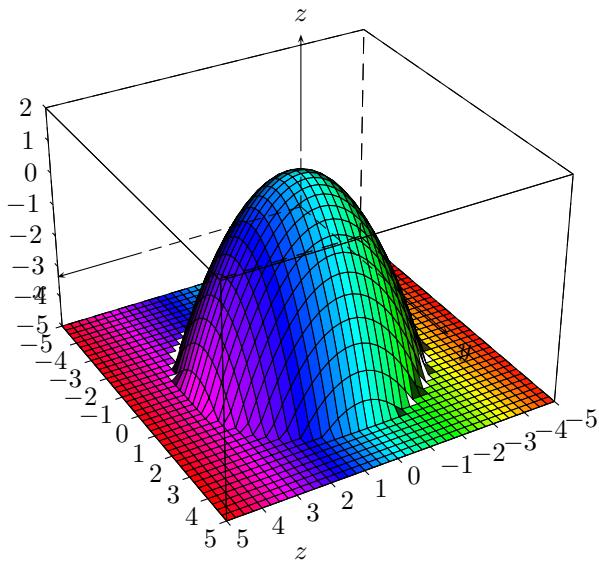
```

1 \psset{unit=0.35}
2 \psset{lightsrc=10 12 20,linewidth=0.5\pslinewidth}
3 \psset{viewpoint=30 50 60 rtp2xyz,Decran=50}
4 \begin{pspicture}(-10,-10)(12,10)
5 \psSurface[
6   fillcolor=cyan!50,algebraic,
7   ngrid=.25 .25,incolor=yellow,hue=0 1,
8   Zmin=-3,Zmax=3](-3,-3)(3,3){%
9   x*y*(x^2-y^2)*0.1}
10 \end{pspicture}

```

8.9. Example 8: a surface with the equation $z = \left(1 - \frac{x^2+y^2}{2}\right)^2$

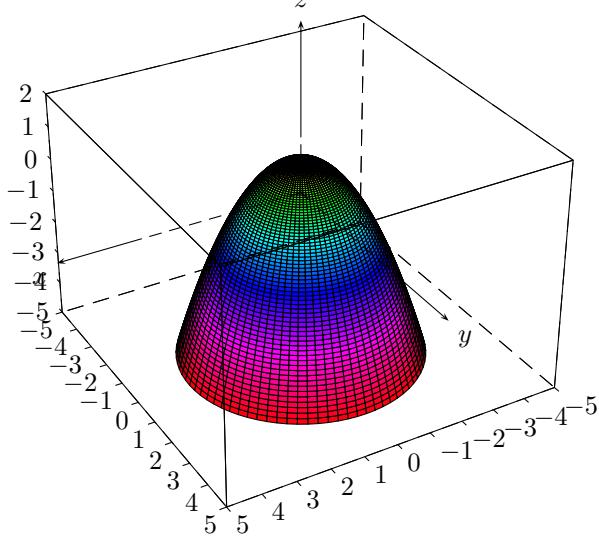
8. Surfaces defined by a function of the form $z = f(x, y)$



```

1 \psset{unit=0.5cm,viewpoint=50 60 30 rtp2xyz,Decran
=50}
2 \psset{lightsrc=viewpoint}
3 \begin{pspicture}(-4,-5)(6,8)
4 \psSurface*[ngrid=.25 .25,incolor=yellow,linewidth
=0.5\pslinewidth,
5 base= -2 2 -2 2, axesboxed, Zmin=-5,Zmax=2,hue=0
1](-5,-5)(5,5){%
6 1 0.5 x dup mul y dup mul add mul sub dup -5 lt {
pop -5 }if }
7 \end{pspicture}

```



```

1 \psset{unit=0.5cm,viewpoint=50 60 30 rtp2xyz,Decran
=50,
2 lightsrc=viewpoint}
3 \begin{pspicture}(-4,-5)(6,8)
4 \psSurface*[ngrid=.25 .25,incolor=yellow,
5 linewidth=0.5\pslinewidth,
6 r = 3 sqrt 2 mul, axesboxed, Zmin=-5,Zmax=2,hue=0
1](-5,-5)(5,5){%
7 1 0.5 x dup mul y dup mul add mul sub dup -5 lt {
pop -5 }if }
8 \end{pspicture}

```

CHAPTER 9

Advanced usage

9.1. Naming a solid

For certain purposes, it is helpful to save a solid in working storage to allow it to be referenced later on. To do so, we activate the Boolean `solidmemory`, which allows the transmission of a variable throughout the code.

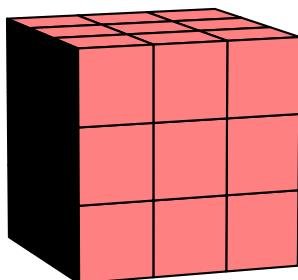
Consequently, activation of this Boolean deactivates drawing by the macros `\psSolid`, `\psSurface` and `\psProjection` immediate. To obtain the drawing, we use the macro `\at` at the end of the code.

When `\psset{solidmemory}` is set up, we can use the option `name` of the macro `\psSolid`.

In the example below, a coloured solid is constructed, which is named `A`. It is drawn using the object `object=cube` with the parameter `load=A`.

Note that `linecolor=blue`, used while constructing our cube, has no effect on the drawing: only the structure of the solid is stored (vertices, faces, colours of faces), not the thickness of any line, nor its colour, nor the position of the light source. The settings of those parameters are taken into account at the time the solid is rendered.

Finally, we demonstrate the use of the option `deactivatecolor` which allows the cube to keep its original red colour (otherwise the default colours would be used within the object `load`).



```
1 \psset{unit=0.75}
2 \begin{pspicture}(-4,-4)(5,4)
3 \psset{solidmemory}
4 \psSolid[object=cube,
5   linecolor=blue,
6   a=4,fillcolor=red!50,
7   ngrid=3,
8   action=none,
9   name=A,
10 ](0,0,0)
11 \psSolid[object=load,
12   deactivatecolor,
13   load=A]
14 \composeSolid
15 \end{pspicture}
```

9. Advanced usage

With the option `solidmemory`, the names of variables are relatively well encapsulated, and there will be no conflict with the variables of the dvips driver. There remains however the risk of a collision with the names used in the `solides.pro` file. You could use only single letter variable names, for example, and it is necessary to avoid names like `vecteur`, `distance`, `droite`, etc. which are already defined in the package.

9.2. Sectioning a solid with a plane

9.2.1. Drawing the intersection between a plane and a solid

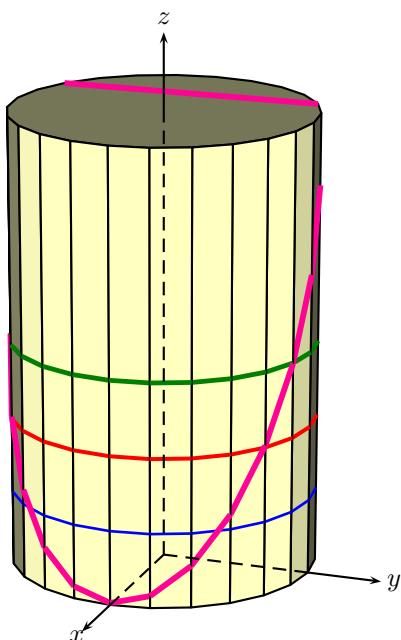
The parameters

The option `intersectionplan={[a b c d]}` allows the user to draw the intersection between a plane and a solid. The numbers between the braces are the coefficients of the affine plane with equation: $ax+by+cz+d=0$. It is possible to draw the intersection between a solid and more than one plane by placing the appropriate parameters in order, as in the following example.

The drawing is activated with `intersectiontype=0` or any value ≥ 0 .

The colour of the intersection line is chosen with the option `intersectioncolor=(bleu) (rouge) etc..` In the same order, the thickness of the appropriate line `intersectionlinewidth=1 2 etc.` (dimensions in picas) is set up.

The hidden parts, drawn with dashed lines, will be shown with `action=draw`.



```

1 \begin{pspicture}(-3,-2)(3,7.5)
2 \psset{viewpoint=50 20 20 rtp2xyz,Decran=50}
3 \psset{lightsrc=viewpoint} \psSolid[object=cylindre,
4   ngrid=1 24,
5   r=2,
6   fillcolor=yellow!25,
7   intersectiontype=0,
8   intersectionplan={
9     [0 0 1 -1]
10    [0 0 1 -2]
11    [0 0 1 -3]
12    [0.894 0 0.447 -1.8]},
13   intersectioncolor=(bleu) (rouge) (vert) (rose),
14   intersectionlinewidth=1 1.5 1.8 2.2]
15 \axesIIID(2,2,6)(3,3,7)
16 \end{pspicture}

```

9.2.2. Slicing a solid

Slicing a filled solid

The object under consideration is a cylinder. The plane that slices the object is defined by:

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

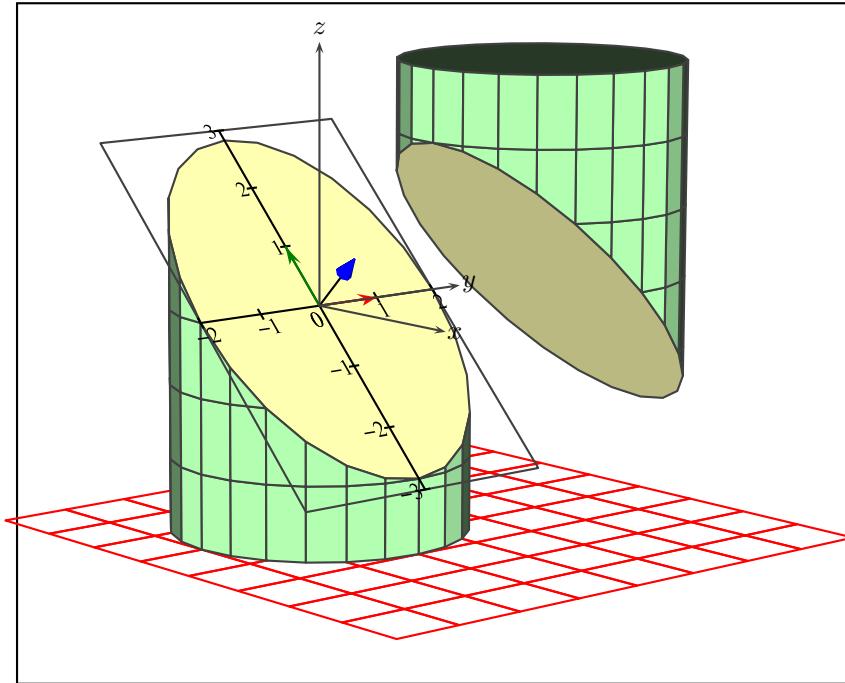
The two parts are not drawn, but memorised with the name `name=partiescylindre`:

```
\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)
```

Then they are displayed separately using their respective index numbers. The numbering of the two parts is determined by the direction of the normal to the slicing plane: 0 if above the normal, 1 if below. For both parts, the sliced face carries the number 0. If there are several sliced faces, as may happen in the case of a torus, they are numbered 0, 1 etc.

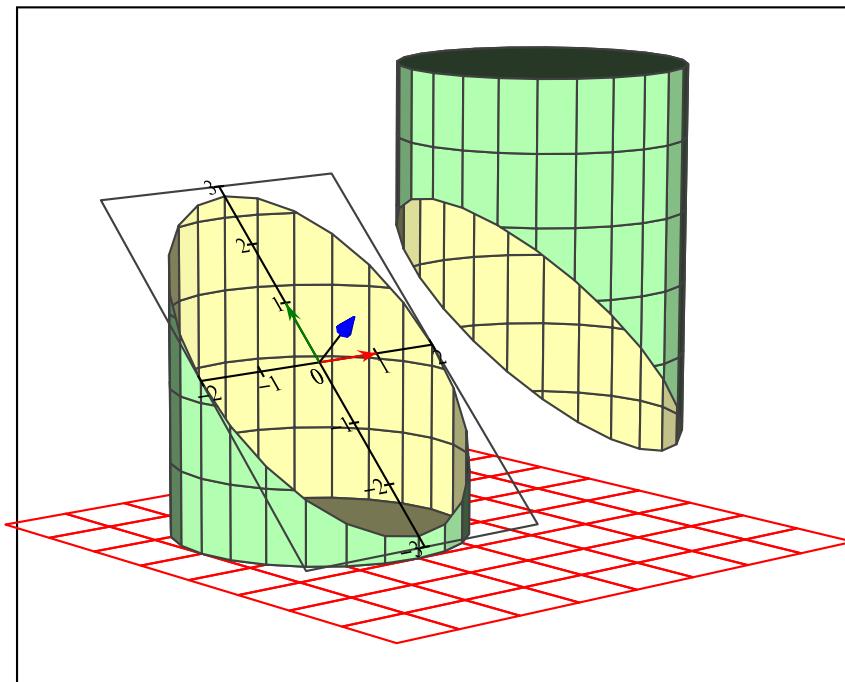
```
\psSolid[object=load,
load=partiescylindre1,
fillcolor={[rgb]{0.7 1 0.7 }},
fcol=0 (1 1 0.7 setrgbcolor)]
\psSolid[object=load,
load=partiescylindre0,RotZ=60,
fillcolor={[rgb]{0.7 1 0.7 }},
fcol=0 (1 1 0.7 setrgbcolor)](0,4,0)
```

9. Advanced usage



Slicing a hollow solid

The options `rm=0,hollow` allow us to not only remove a face `rm=0` but also to see inside it `hollow`.



9.2.3. Slice of a pyramid

Highlighting the contour lines and first slice

This pyramid is generated as `object=new` by giving a list of the coordinates of the vertices, and the vertices of each face.

```
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]
}
```

In the first diagram, the slicing lines are highlighted.

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

Then we cut off the upper part, and draw the slicing plane as well.

```
\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]
```

9. Advanced usage

To avoid having to repeatedly type the vertices and faces of the pyramid, we save these data to the files:

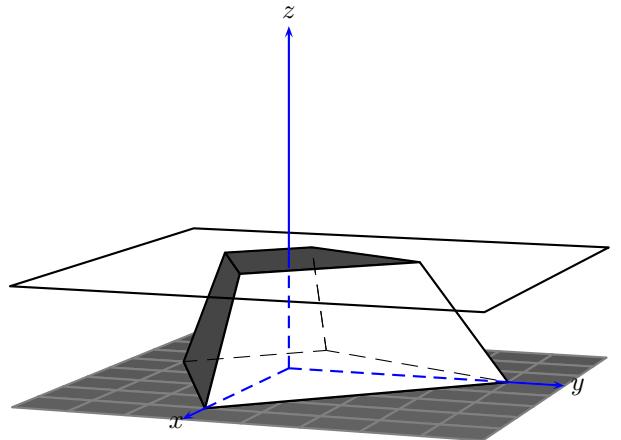
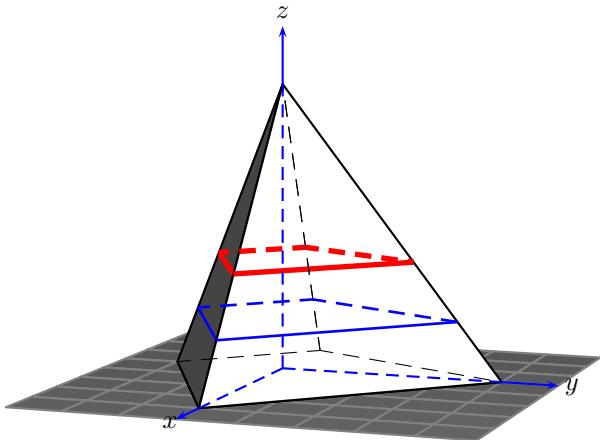
- `Pyramid-couleurs.dat`
- `Pyramid-faces.dat`
- `Pyramid-sommets.dat`
- `Pyramid-io.dat`

thanks to the command `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]
```

All these lines of code could then be removed and, thereafter, we would recall the data with the command:

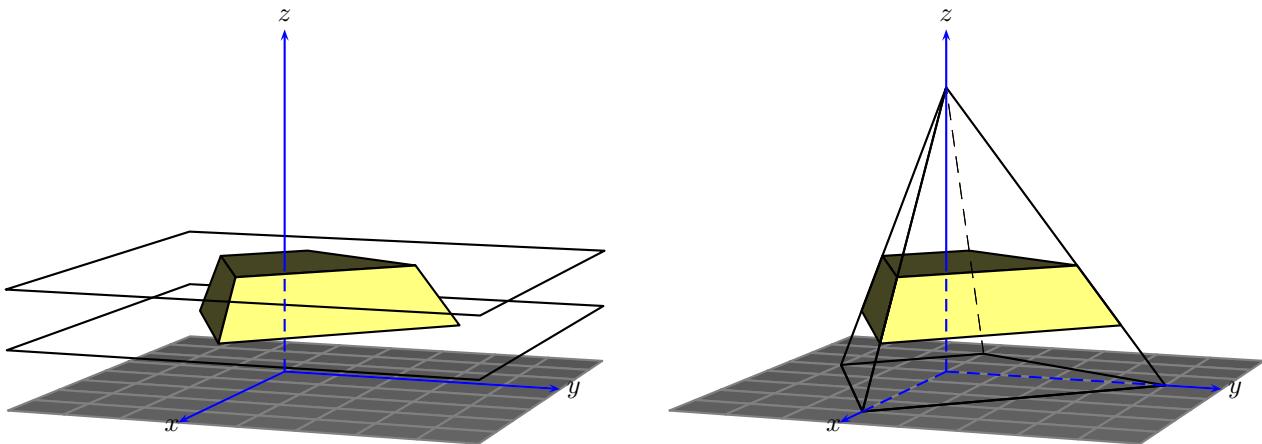
```
\psSolid[object=datfile,
file=./Pyramid]
```



The second slice and its insertion within the pyramid

Having removed the upper part `firstSlice0` (which no longer appears), we slice the frustum of the pyramid `firstSlice1`, and keep the upper part of this as `secondSlice0`, then we record it and insert it into a wire frame model of the pyramid:

```
\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]
```



9.2.4. Slicing an octahedron with a plane parallel to one of its faces

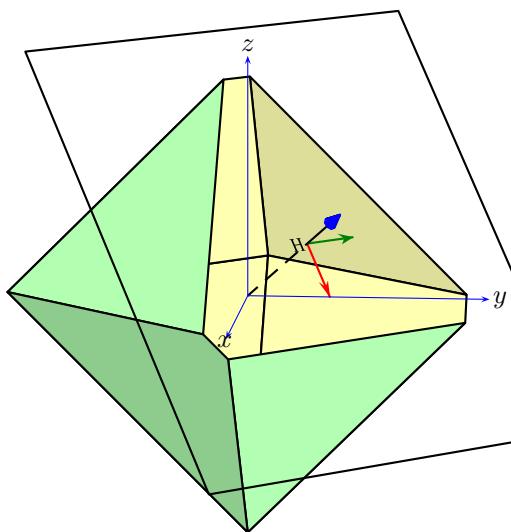
The view inside

Recall that there are options `rm=0,hollow` that allow us, on the one hand, to remove a face `rm=0` and, on the other, to look inside `hollow`.

9. Advanced usage

In the following example, we shall start by generating the required objects without drawing them (`action=none`).

We construct the octahedron, giving the center of the face with index 1 the name G , then define the point H which satisfies $\overrightarrow{OH} = 0.8 \overrightarrow{OG}$. After that we define P to be the plane through H parallel to the face of the octahedron with index 1. Finally, we slice the octahedron using the plane P .



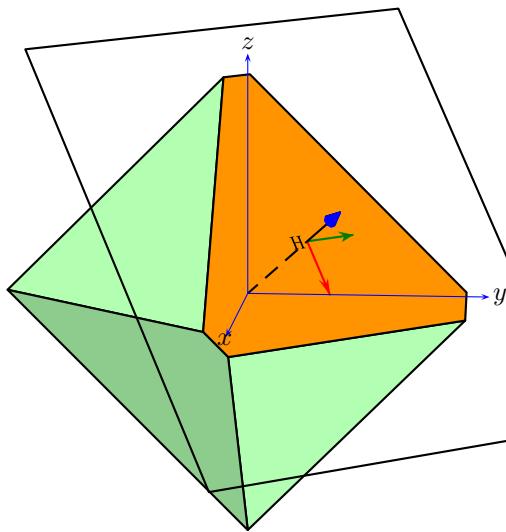
```

1 \begin{pspicture}(-3.5,-3)(4.5,5)
2 \psset{viewpoint=100 5 10 rtp2xyz,Decran=80,
3   lightsrc=viewpoint,solidmemory,action=none}
4 \psSolid[object=octahedron,
5   a=4,name=my_octahedron,]
6 \psSolid[object=point,
7   definition=solidcentreface,
8   args=my_octahedron 1,
9   name=G,]
10 \psSolid[object=point,
11   definition=mulv3d,
12   args=G .8,
13   name=H,]
14 \psSolid[object=plan,
15   definition=solidface,
16   args=my_octahedron 1,
17   base=-4 4 -4 4,
18   name=P,](H,,)
19 \psSolid[object=load,
20   load=my_octahedron,
21   planseparate=P,
22   name=part]
23 \psSolid[object=load,load=part1,
24   rm=0,hollow,action=draw**,
25   fillcolor={[rgb]{0.7 1 0.7}},,
26   incolor={[rgb]{1 1 0.7}},]
27 \psSolid[object=plan,args=P,
28   action=draw,showBase]
29 \psSolid[object=line,
30   args=0 0 0 H,
31   linestyle=dashed,]
32 \psProjection[object=point,plan=P,args=0 0,
33   fontsize=20, pos=cl, text=H, phi=90,]
34 \axesIIID [linecolor=blue,linewidth=0.4pt](0,0,0)
(4,4,4)
35 \end{pspicture}

```

Regarding the solid as filled

The option `fcol=0` (`YellowOrange`) allows us to colour the face with index 0.



```

1 \begin{pspicture}(-3.5,-3)(4.5,5)
2 \psset{viewpoint=100 5 10 rtp2xyz,Decran=80,
3   lightsrc=viewpoint,solidmemory,action=none}
4 \psSolid[object=octahedron,
5   a=4,name=my_octahedron,]
6 \psSolid[object=point,
7   definition=solidcentreface,
8   args=my_octahedron 1,
9   name=G,]
10 \psSolid[object=point,
11   definition=mulv3d,
12   args=G .8,
13   name=H,]
14 \psSolid[object=plan,
15   definition=solidface,
16   args=my_octahedron 1,
17   base=-4 4 -4 4,
18   name=P,](H,,)
19 \psSolid[object=load,
20   load=my_octahedron,
21   plansepare=P,
22   name=part]
23 \psSolid[object=load,
24   load=part1,
25   fcol=0 (YellowOrange),
26   action=draw**,
27   fillcolor={[rgb]{0.7 1 0.7}},]
28 \psSolid[object=plan,args=P,
29   action=draw,showBase]
30 \psSolid[object=line,
31   args=0 0 H,
32   linestyle=dashed,]
33 \psProjection[object=point,plan=P,args=0 0,
34   fontsize=20, pos=cl, text=H, phi=90,]
35 \axesIIID[linecolor=blue,linewidth=0.4pt](0,0,0)
36 (4,4,4)
36 \end{pspicture}

```

The two parts of a sliced solid

You will recall that the direction of the normal of the slicing plane determines the numbering of the two parts: 0 if above the normal, 1 if below. For both parts, the sliced face carries the number 0. If there are several sliced faces, as in the case of the torus, they are numbered 0, 1 etc.

Using two steps, we memorise both parts of the sliced solid:

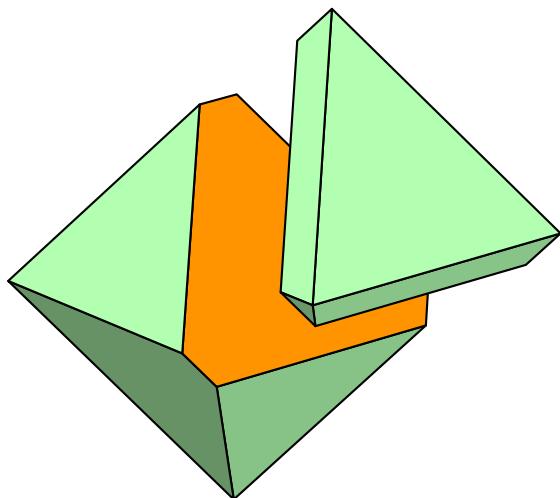
```
\psSolid[object=load,
load=my_octahedron,
plansepare=P,
name=part]
```

Then we position and render each part:

```
\psSolid[object=load,
fcol=0 (YellowOrange),
```

9. Advanced usage

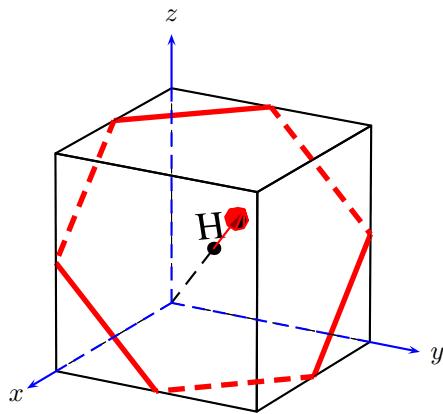
```
fillcolor={[rgb]{0.7 1 0.7}},
load=part1]
\psSolid[object=load,
fillcolor={[rgb]{0.7 1 0.7}},
load=part0](H 2 mulv3d,,)
\composeSolid
```



```
1 \begin{pspicture}(-3.5,-3)(4.5,5)
2 \psset{viewpoint=100 5 20 rtp2xyz,Decran=150,
3 lightsrc=viewpoint,solidmemory,action=none}
4 \psSolid[object=octahedron,
5 a=2,name=my_octahedron,]
6 \psSolid[object=point,
7 definition=solidcentreface,
8 args=my_octahedron 1,
9 name=G,]
10 \psSolid[object=point,
11 definition=mulv3d,
12 args=G .7,
13 name=H,]
14 \psSolid[object=plan,
15 definition=solidface,
16 args=my_octahedron 1,
17 base=-4 4 -4 4,
18 name=P,](H,,)
19 \psSolid[object=load,
20 load=my_octahedron,
21 planseparate=P,
22 name=part]
23 \psset{action=draw**}
24 \psSolid[object=load,
25 load=part1,
26 fcol=0 (YellowOrange),
27 fillcolor={[rgb]{0.7 1 0.7}},]
28 \psSolid[object=load,
29 fillcolor={[rgb]{0.7 1 0.7}},
30 load=part0](H 2 mulv3d,,)
31 \composeSolid
32 \end{pspicture}
```

9.2.5. Slices of a cube

Highlighting the edges of the cut

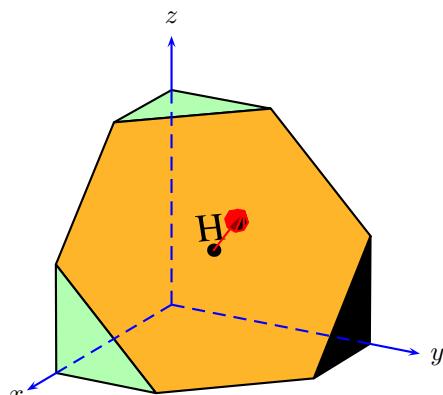


```

1 \psset{viewpoint=100 30 20 rtp2xyz,Decran=150}
2 \begin{pspicture}(-4,-3)(4,5)
3 \psset{solidmemory}
4 \psSolid[object=plan,definition=normalpoint,
5   args={1 1 1 [1 1 1]},action=none,name=P]
6 \psSolid[object=cube,a=2,action=draw,
7   intersectiontype=0,
8   intersectionplan=P,
9   intersectionlinewidth=2,
10  intersectioncolor=(rouge),
11 ](1,1,1)
12 \psProjection[object=point,
13   args=0 0,fontsize=10,pos=dc,
14   text=H,phi=-30,plan=P,
15 ]
16 \psSolid[object=line,
17   linestyle=dashed,
18   args=0 0 0 1 1 1]
19 \psSolid[object=vecteur,
20   linecolor=red,
21   args=1 1 1 .7 mulv3d](1,1,1)
22 \axesIIID[linecolor=blue](2,2,2)(2.5,2.5,2.5)
23 \end{pspicture}

```

Showing the sliced cube with its hexagonal cut face



```

1 \psset{viewpoint=100 30 20 rtp2xyz,Decran=150}
2 \begin{pspicture}(-4,-3)(4,5)
3 \psset{solidmemory}
4 \psSolid[object=plan,action=none,definition=
      normalpoint,
5   args={1 1 1 [1 1 1]},name=P]
6 \psSolid[object=cube,a=2,
7   plansepare=P,
8   action=none,
9   name=parts_cube,
10 ](1,1,1)
11 \psSolid[object=load,
12   load=parts_cube1,
13   fcol=0 (Dandelion),
14   fillcolor={[rgb]{0.7 1 0.7}},
15 ]
16 \psProjection[object=point,
17   args=0 0,fontsize=10,pos=dc,
18   text=H,phi=-30,plan=P,
19 ]
20 \psSolid[object=vecteur,
21   linecolor=red,
22   args=1 1 1 .7 mulv3d](1,1,1)
23 \axesIIID[linecolor=blue](2,2,2)(2.5,2.5,2.5)
24 \end{pspicture}

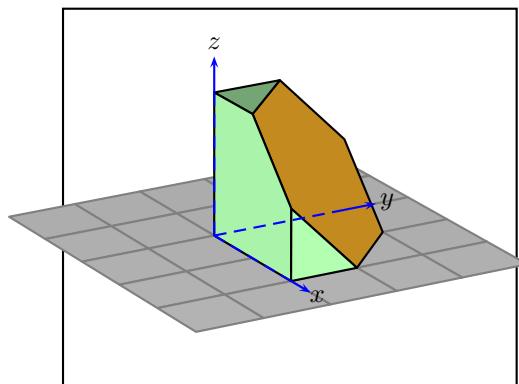
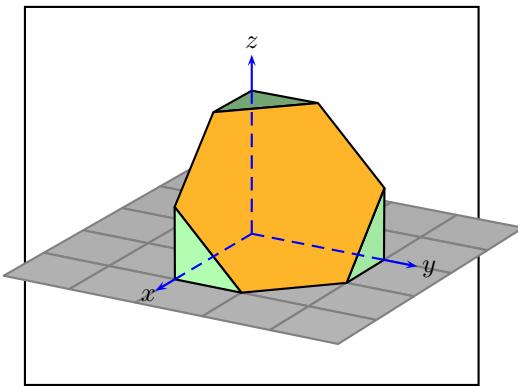
```

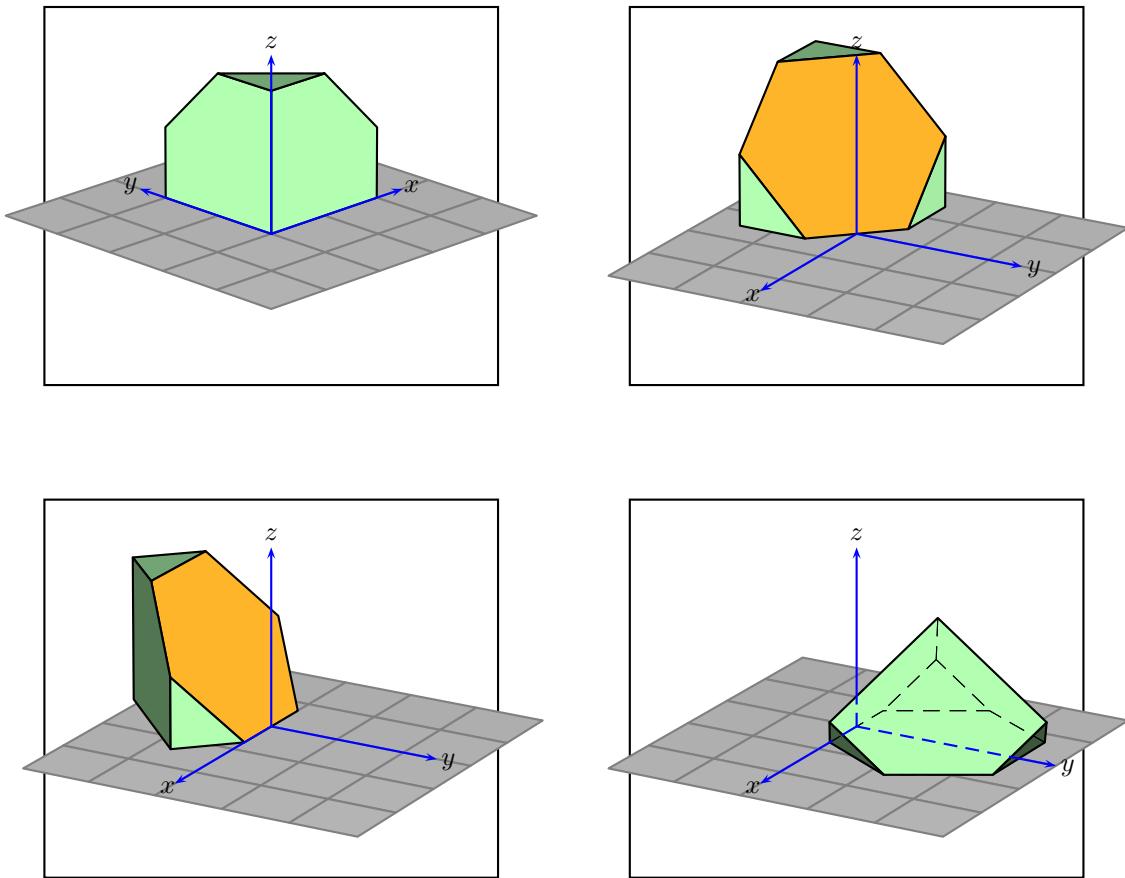
9. Advanced usage

The sliced cube in various positions

Where we use the option that allows us to memorise a solid, in order to put the truncated cube, after undergoing various transformations, down on its cut face.

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

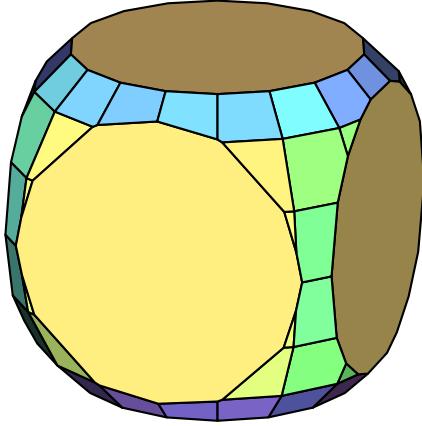




9.2.6. Multiple sections

Slicing a sphere with PStricks

9. Advanced usage

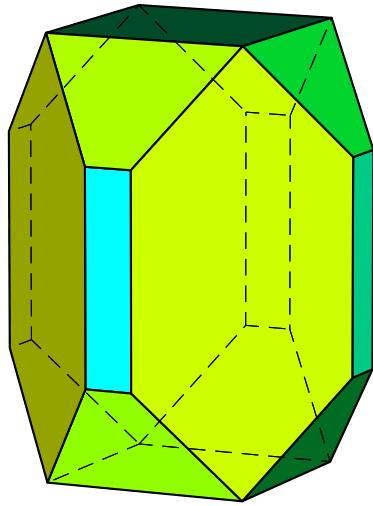


```
1 \begin{pspicture}(-4,-4)(4,4)
2 \psset{viewpoint=100 20 20 rtp2xyz,Decran=75}
3 \psset{solidmemory,lightsrc=viewpoint}
4 \codejps{
5   /coeff 0.75 def /r0 4 def /OH coeff r0 mul neg
6   def}%
6 \psSolid[object=sphere,
7   r=r0,ngrid=9 18,
8   plansepare={[1 0 0 OH]},
9   name=part,
10  action=none]
11 \psSolid[object=load,
12   load=part1,plansepare={[-1 0 0 OH]},action=none
13   ,name=part]
13 \psSolid[object=load,
14   load=part1,plansepare={[0 1 0 OH]},action=none,
15   name=part]
15 \psSolid[object=load,
16   load=part1,plansepare={[0 -1 0 OH]},action=none
17   ,name=part]
17 \psSolid[object=load,
18   load=part1,plansepare={[0 0 1 OH]},action=none,
19   name=part]
19 \psSolid[object=load,
20   load=part1,plansepare={[0 0 -1 OH]},action=none
21   ,name=part]
21 \psSolid[object=load,hue=.1 .8 0.5 1,
22   load=part1](0,0,0)
23 \composeSolid
24 \end{pspicture}
```

Multiple sections of a parallelepiped

Multiple sections are better carried out inside a PostScript loop, within `\codejps`; it's easier and quicker!

In this example, the original solid is a parallelepiped. Truncations of the vertices and chamfering of the edges are effected by means of slicing planes, starting off with the vertices and finishing with the edges.



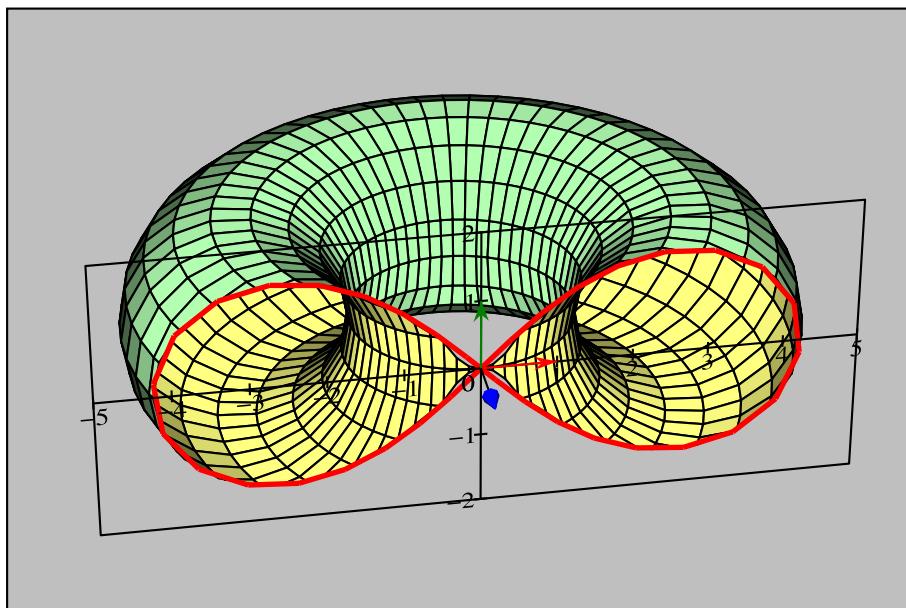
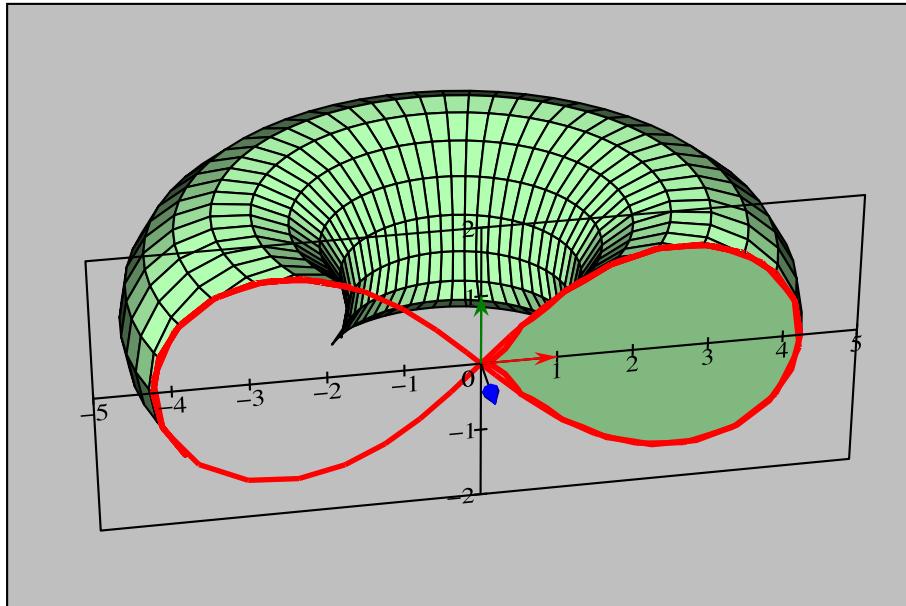
```

1 \begin{pspicture}(-3.5,-4)(3.5,4)
2 \psset{viewpoint=100 -20 10 rtp2xyz,Decran=100}
3 %\lightsource
4 \psset{lightsrc=viewpoint}
5 \codejps{
6 4 4 6 newparallelepiped
7 45 90 360 {
8 /iAngle exch def
9 /n_x iAngle cos 35.2644 cos mul def
10 /n_y iAngle sin 35.2644 cos mul def
11 /n_z 35.2644 sin def
12 /distance 2 3 add 3 sqrt div neg def
13 [ n_x n_y n_z distance]
14 solidplansepare
15 } for
16 45 90 360 {
17 /iAngle exch def
18 /n_x iAngle cos 35.2644 cos mul def
19 /n_y iAngle sin 35.2644 cos mul def
20 /n_z 35.2644 sin neg def
21 /distance 2 3 add 3 sqrt div neg def
22 [ n_x n_y n_z distance]
23 solidplansepare
24 } for
25 45 90 360 {
26 /iAngle exch def
27 % plan : ax+by+cz-d=0
28 [ iAngle cos % a
29   iAngle sin % b
30   0 % c
31   -2.5 % -d
32 ] solidplansepare
33 } for
34 dup [.5 .2] solidputhuecolors
35 solidlightOn
36 drawsolid*}
37 \end{pspicture}

```

9. Advanced usage

9.2.7. Sections of a torus



9.2.8. Some more examples

1. You will find a *jps* coded version of this document within the \code{jps} command in the following document:

<http://melusine.eu.org/syracuse/mluque/solides3d2007/sections>

2. A lesson about conic sections on:

<http://melusine.eu.org/syracuse/mluque/solides3d2007/sections/sections-cone>

3. A lesson about cylindrical sections on:

<http://melusine.eu.org/syracuse/mluque/solides3d2007/sections/section-cylindre>

4. A lesson about sections of a torus on:

<http://melusine.eu.org/syracuse/mluque/solides3d2007/sections/section-tore>

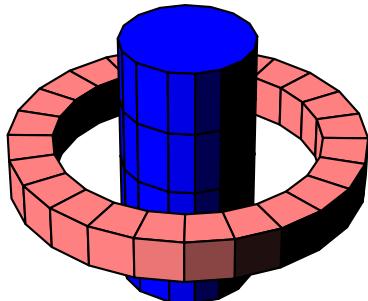
9.3. Fusing solids

It is possible to arrange several solids within the same structure: this is done with the operation **fusion** of solids. This technique uses the painting algorithm for the whole scene.

To do so, you must activate the option `\psset{solidmemory}` to memorize the structures of the different solids within `\psSolid`, with each of them given a separate name.

You use the object **fusion** of `\psSolid`, by indicating in the parameter `base` the list of names of the solids to be fused.

To draw the scene, don't forget to conclude the code with `\composeSolid`.

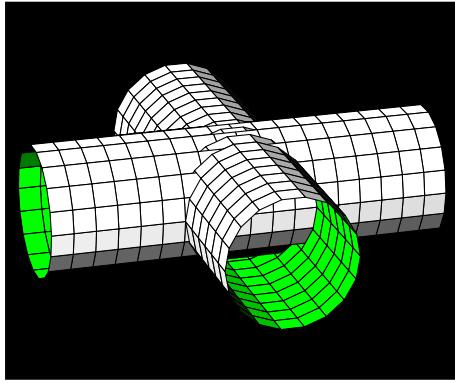


```

1 \psset{unit=.6}
2 \begin{pspicture}(-6,-5)(6,7)
3 \psset{solidmemory}
4 \psSolid[object=cylindre,h=6,fillcolor=blue,
5      r=1.5,
6      ngrid=4 16,
7      action=none,
8      name=A1,
9      ](0,0,-4)
10 \psSolid[object=anneau,h=6,fillcolor=red!50,
11      R=4,r=3,h=1,
12      action=none,
13      name=B1,
14      ](0,0,-1)
15 \psSolid[object=fusion,
16      action=draw**,
17      base=A1 B1,
18      ](0,0,0)
19 \composeSolid
20 \end{pspicture}

```

9. Advanced usage



```

1 \psset{unit=0.5}
2 \begin{pspicture}(-6,-5)(6,5)
3 \psset{solidmemory}
4 \psset{lightsrc=50 -50 50,viewpoint=100 -30 40,
5 Decran=100,linewidth=0.5\pslinewidth,
6 ngrid=18 18,fillcolor=white,
7 h=12,r=2,RotX=90}
8 \psframe*[linecolor=black](-6,-5)(6,5)
9 \psSolid[object=cylindrecreux,
10 action=none,
11 name=cylindre1](0, 6, 0)
12 \psSolid[object=cylindrecreux,
13 RotZ=90,
14 action=none,
15 name=cylindre2](-6, 0, 0)
16 \psSolid[object=fusion,
17 base=cylindre1 cylindre2,RotX=0]
18 \composeSolid
19 \end{pspicture}

```

9.4. Fusing with *jps code*

We can also fuse solids by passing the code directly using *jps code*. The calculation of the hidden parts is carried out by the PostScript routines of the `solides.pro` file, but the lines of code are “encapsulated” within a `pspicture` environment thanks to the command `\code{jps{ps code}}`.

9.4.1. Using *jps code*

The choice of object

- `[section] n newanneau`: choice of a cylindrical ring defined by the coordinates of the vertices of its intersection with the plane Oyz .
- `2 1.5 6 [4 16] newcylindre`: choice of a vertical cylinder with the following parameters:
 - `z0=2`: the position of the base centre on the axis Oz ;
 - `radius=1.5`: radius of the cylinder;
 - `z1=6`: the position of the top centre on the axis Oz ;
 - `[4 16]`: the cylinder is sliced horizontally into 4 pieces and vertically into 16 sectors.

The transformations

- `{-1 2 5 translatepoint3d} solidtransform`: the object previously chosen is translated to the point with the coordinates $(x = -1, y = 2, z = 5)$.
- `{90 0 45 rotate0point3d} solidtransform`: the object previously chosen is rotated around the axes (Ox, Oy, Oz) , in this order: rotation of 90° about (Ox) followed by a rotation of 45° about (Oz) .

The choice of object colour

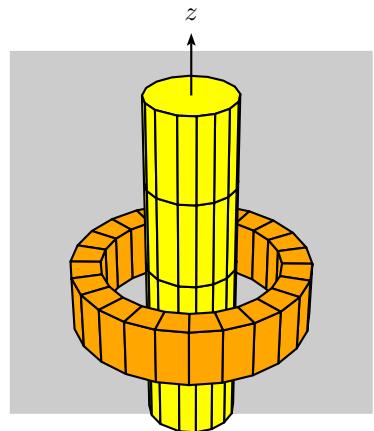
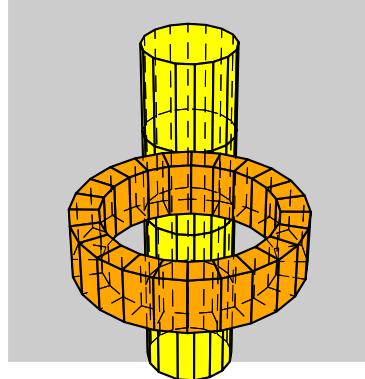
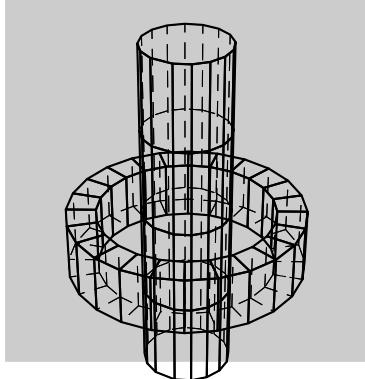
- dup (yellow) outputcolors: a yellow object illuminated in white light.

Fusing objects

- The fusion is finally made with the instruction **solidfuz**.

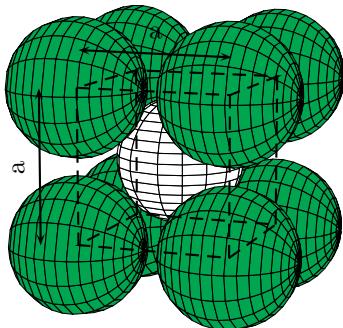
Designing objects

- There are three drawing options:
 - **drawsolid**: only draw edges; hidden edges are drawn dashed;
 - **drawsolid***: draw and fill solids in their coded order (not a very interesting option at first glance); hidden edges are drawn dashed;
 - **drawsolid****: draw and fill solids with the painting algorithm; only those parts seen by the observer are drawn.



```
\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}
```

9.4.2. A chloride ion



```

1 \begin{pspicture}(-3,-4)(3,4)
2 \psset{lightsrc=100 -50 -10,lightintensity=3,viewpoint=200 20
10 rtp2xyz,Decran=20}
3 {\psset{linewidth=0.5\pslinewidth}
4 \codejps{/C1 {9.02 [18 16] newsphere
5 {-90 0 0 rotate0point3d} solidtransform
6 dup (Green) outputcolors} def
7 /C11 { C1 {10.25 10.25 10.25 translatepoint3d} solidtransform
} def
8 /C12 { C1 {10.25 -10.25 10.25 translatepoint3d} solidtransform
} def
9 /C13 { C1 {-10.25 -10.25 10.25 translatepoint3d}
solidtransform } def
10 /C14 { C1 {-10.25 10.25 10.25 translatepoint3d} solidtransform
} def
11 /C15 { C1 {10.25 10.25 -10.25 translatepoint3d} solidtransform
} def
12 /C16 { C1 {10.25 -10.25 -10.25 translatepoint3d}
solidtransform } def
13 /C17 { C1 {-10.25 -10.25 -10.25 translatepoint3d}
solidtransform } def
14 /C18 { C1 {-10.25 10.25 -10.25 translatepoint3d}
solidtransform } def
15 /Cs {8.38 [18 16] newsphere
16 dup (White) outputcolors} def
17 /C112{ C11 C12 solidfuz} def
18 /C1123{ C112 C13 solidfuz} def
19 /C11234{ C1123 C14 solidfuz} def
20 /C112345{ C11234 C15 solidfuz} def
21 /C1123456{ C112345 C16 solidfuz} def
22 /C11234567{ C1123456 C17 solidfuz} def
23 /C112345678{ C11234567 C18 solidfuz} def
24 /C_Cs { C112345678 Cs solidfuz} def
25 C_Cs drawsolid**}}%
26 \psPoint(0,0,0){P}
27 \psPoint(10.25,10.25,10.25){C11}
28 \psPoint(10.25,-10.25,10.25){C12}
29 \psPoint(-10.25,-10.25,10.25){C13}
30 \psPoint(-10.25,10.25,10.25){C14}
31 \psPoint(10.25,10.25,-10.25){C15}
32 \psPoint(10.25,-10.25,-10.25){C16}
33 \psPoint(-10.25,-10.25,-10.25){C17}
34 \psPoint(-10.25,10.25,-10.25){C18}
35 \pspolygon [linestyle=dashed](C11)(C12)(C13)(C14)
36 \pspolygon [linestyle=dashed](C15)(C16)(C17)(C18)
37 \psline [linestyle=dashed](C12)(C16)
38 \psline [linestyle=dashed](C13)(C17)
39 \psline [linestyle=dashed](C11)(C15)
40 \psline [linestyle=dashed](C14)(C18)
41 \pcline[offset=0.5]{<->}(C12)(C11)
42 \aput{:U}{a}
43 \pcline[offset=0.5]{<->}(C16)(C12)
44 \aput{:U}{a}
45 \end{pspicture}
```

We define the chloride ion Cl^- :

```
/C1 {9.02 [12 8] newsphere
{-90 0 0 rotate0point3d} solidtransform
dup (Green) outputcolors} def
```

which we shift to each vertex of a cube:

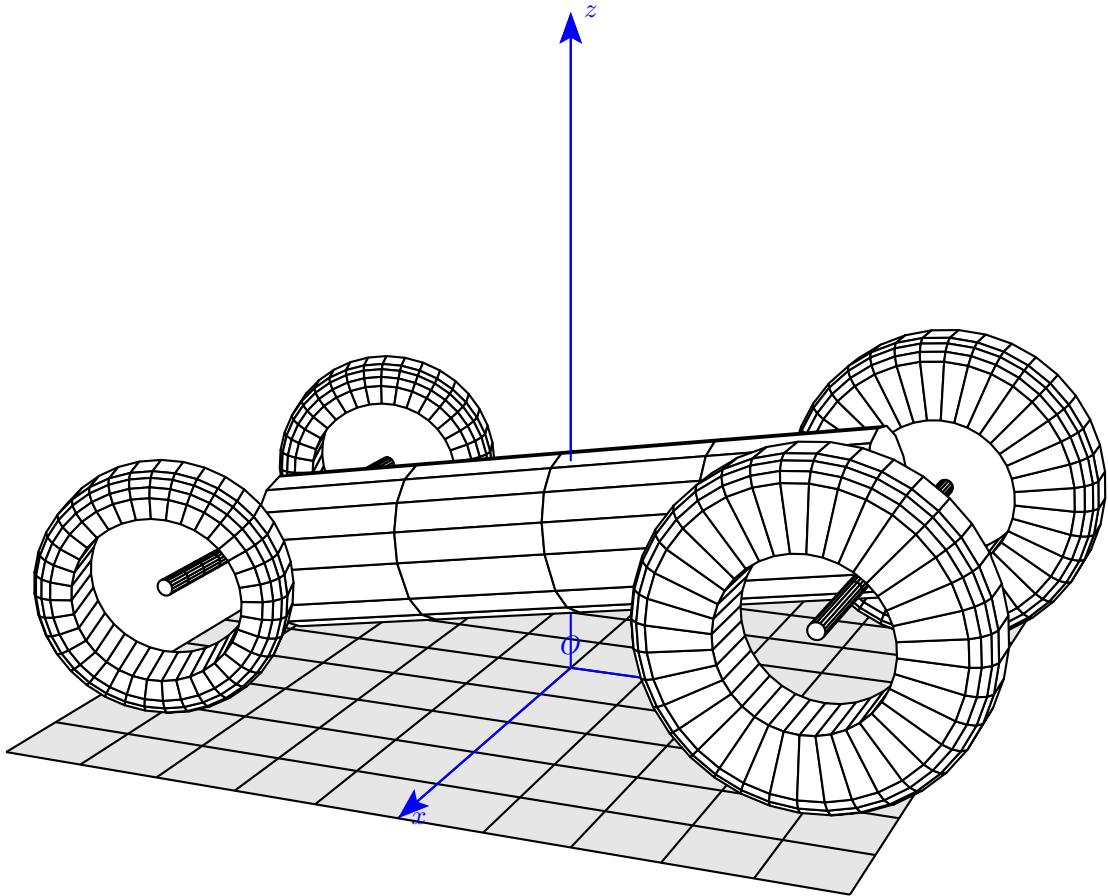
```
/C11 { C1 {10.25 10.25 10.25 translatepoint3d} solidtransform } def
/C12 { C1 {10.25 -10.25 10.25 translatepoint3d} solidtransform } def
/C13 { C1 {-10.25 -10.25 10.25 translatepoint3d} solidtransform } def
/C14 { C1 {-10.25 10.25 10.25 translatepoint3d} solidtransform } def
/C15 { C1 {10.25 10.25 -10.25 translatepoint3d} solidtransform } def
/C16 { C1 {10.25 -10.25 -10.25 translatepoint3d} solidtransform } def
/C17 { C1 {-10.25 -10.25 -10.25 translatepoint3d} solidtransform } def
/C18 { C1 {-10.25 10.25 -10.25 translatepoint3d} solidtransform } def
```

Then a caesium ion Cs^+ is placed in the center:

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

Finally we fuse the separate spheres in pairs.

9.4.3. A prototype of a vehicle



We have to operate in several steps to fuse the solids in pairs:

- We first fuse the two front wheels `roue12`:

```
/roue12 {
% solide 1
/R 2 def /r 1 def /h 1 def
[Pneu] 36 newanneau
{90 0 90 rotate0point3d} solidtransform
{3 4 2 translatepoint3d} solidtransform
dup (White) outputcolors
% solide 2
[Pneu] 36 newanneau
{90 0 90 rotate0point3d} solidtransform
{-3 4 2 translatepoint3d} solidtransform
dup (White) outputcolors
% fusion
solidfuz } def
```

- Then the two wheels and their axis:

```
/axe12{
0 0.1 6 [4 16] newcylindre
{90 0 90 rotate0point3d} solidtransform
{-3 4 2 translatepoint3d} solidtransform
```

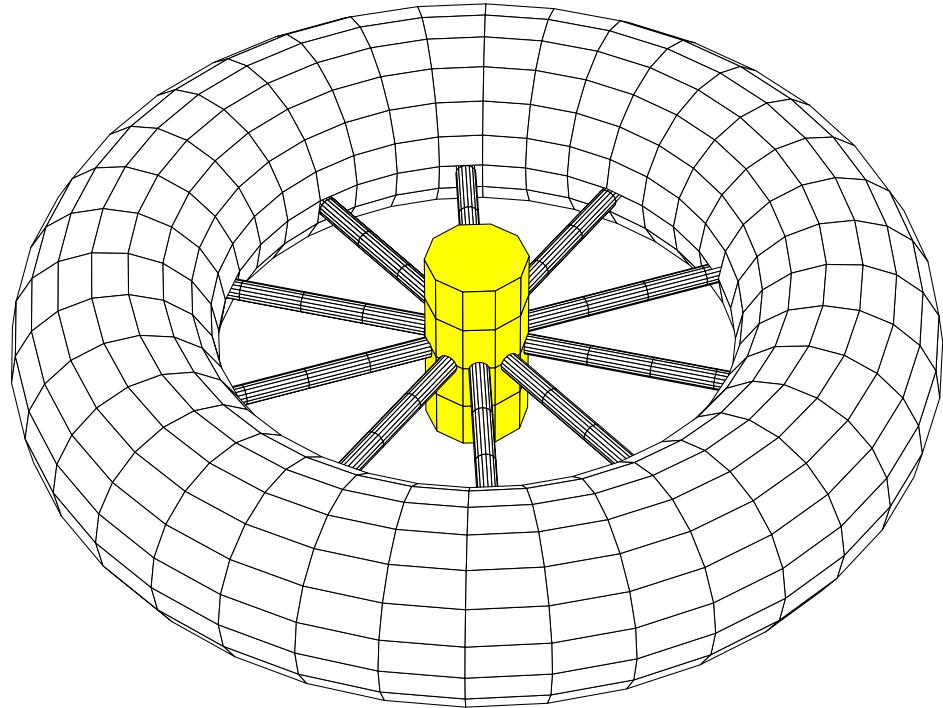
```

dup (White) outputcolors
} def
/roue12axes {
roue12 axe12 solidfuz } def
• After that the rear wheels and their axis:
/roue34 {
% solide 3
/R 1.5 def /r 1 def /h 1 def
[Pneu] 36 newanneau
{90 0 110 rotate0point3d} solidtransform
{3 -4 1.5 translatepoint3d} solidtransform
dup (White) outputcolors
% solide 4
[Pneu] 36 newanneau
{90 0 110 rotate0point3d} solidtransform
{-3 -4 1.5 translatepoint3d} solidtransform
dup (White) outputcolors
% fusion
solidfuz } def
/axe34{
0 0.1 6 [16 16] newcylindre
{90 0 90 rotate0point3d} solidtransform
{-3 -4 1.5 translatepoint3d} solidtransform
dup (White) outputcolors
} def
/roue34axes34 {
roue34 axe34 solidfuz } def
• Then fuse the two wheel assemblies:
/roues {roue34axes34 roue12axes solidfuz} def
• The final step is to fuse the previously generated solid with the chassis:
/chassis {
0 1 8 [4 16] newcylindre
{100 0 0 rotate0point3d} solidtransform
{0 4 2.5 translatepoint3d} solidtransform
dup (White) outputcolors
} def
roues chassis solidfuz
drawsolid**}

```

9. Advanced usage

9.4.4. A wheel – or a space station



We define the first spoke:

```
/rayon0 {
    1 0.2 6 [4 16] newcylindre
    {90 0 0 rotate0point3d} solidtransform
    dup (White) outputcolors
} def
```

Then, with a loop, we fuse all the spokes of the wheel:

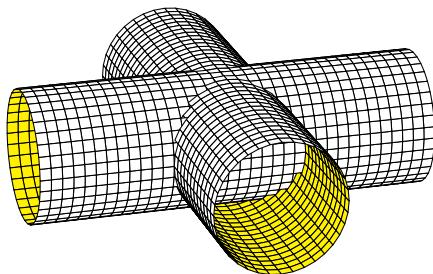
```
36 36 360 {
    /angle exch def
    /rayon1 {
        1 0.2 6 [4 16] newcylindre
        {90 0 angle rotate0point3d} solidtransform
        dup (White) outputcolors
    } def
    /rayons {rayon0 rayon1 solidfuz} def
    /rayon0 rayons def
} for
```

After that, we draw the hub and the tyre of the wheel, and finally fuse all of them:

```
/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**
```

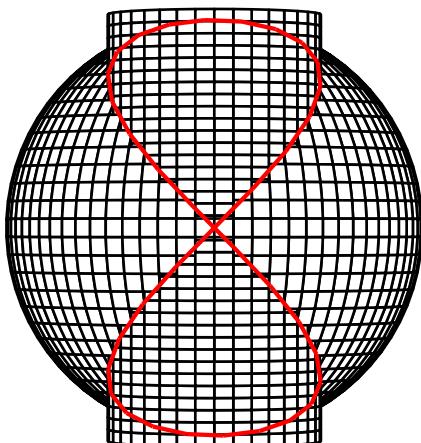
9.4.5. Intersection of two cylinders



```
1 \begin{pspicture}(-4,-3)(6,3)
2 \psset{lightsrc=50 -50 50,viewpoint=100 -30
3 40,Decran=100,linewidth=0.5\pslinewidth, unit=0.5}
4 \codejps{
5 /cylindre1 {
6 -6 2 6 [36 36] newcylindrecreux %newcylindre
7 {90 0 0 rotate0point3d} solidtransform
8 dup (Yellow) (White) inoutputcolors
9 } def
10 /cylindre2 {
11 -6 2 6 [36 36] newcylindrecreux %newcylindre
12 {90 0 90 rotate0point3d} solidtransform
13 dup (Yellow) (White) inoutputcolors
14 } def
15 /UnionCylindres {cylindre1 cylindre2 solidfuz}
def
16 UnionCylindres drawsolid**}
17 \end{pspicture}
```

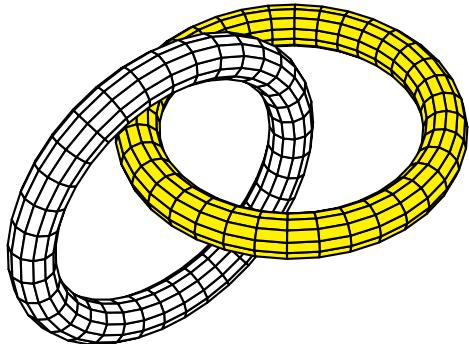
9.4.6. Intersection between a sphere and a cylinder

This time we draw the curve of intersection using `\psSolid[object=courbe]`.



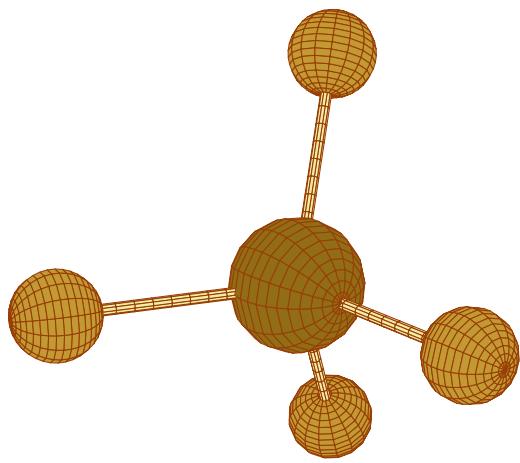
```
1 \psset{unit=0.5,lightsrc=50 -50 50,viewpoint=100 0
0 rtp2xyz,Decran=110,linewidth=0.5\
pslinewidth}
2 \begin{pspicture}(-7,-6)(5,6)
3 \defFunction[F](t){t cos dup mul 5 mul}{t cos t
sin mul 5 mul}{t sin 5 mul}
4 \codejps{%
5 /cylindre1 {
6 -5 2.5 5 [36 36] newcylindre
7 {2.5 0 0 translate0point3d} solidtransform
8 dup (White) outputcolors
9 } def
10 /sphere1 {
11 5 [36 72] newsphere
12 dup (White) outputcolors
13 } def
14 /CS {cylindre1 sphere1 solidfuz} def
15 CS drawsolid**}
16 \psPoint(0,0,0){0}
17 \psSolid[object=courbe,r=0,
18 function=F,
19 range=0 360,
20 linecolor=red,linewidth=4\pslinewidth]
21 \end{pspicture}
```

9.4.7. Two linked rings



```
1 \begin{pspicture}(-5,-4)(3,3)
2 \psset{lightsrc=50 50 50,viewpoint=40 50 60,Decran
   =30,unit=0.85}
3 \codejps{
4 /anneau1 {1 7 [12 36] newtore
5 {0 0 0 translatepoint3d} solidtransform
6 dup (Yellow) outputcolors} def
7 /anneau2 {1 7 [12 36] newtore
8 {90 0 0 rotate0point3d} solidtransform
9 {7 0 0 translatepoint3d} solidtransform
10 dup (White) outputcolors} def
11 /collier {anneau1 anneau2 solidfuz} def
12 collier drawsolid**}
13 \end{pspicture}
```

9.4.8. The methane molecule: wooden model



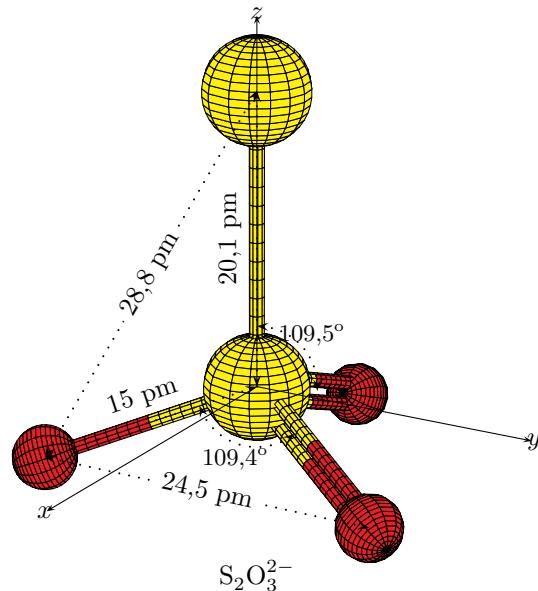
```

1 \begin{pspicture}(-4.5,-4)(3.2,5)
2 \psset{lightsrc=50 50 10,lightintensity=2,
         viewpoint=100 50 20 rtp2xyz,
         Decran=30}
3 \psset{linecolor={[cmyk]{0,0.72,1,0.45}}, linewidth
        =0.5\pslinewidth,
        unit=1}
6 %\psframe[fillstyle=solid,fillcolor=green
           !20](-4,-4)(3.2,5)
7 \pstVerb{/hetre {0.764 0.6 0.204 setrgbcolor} def
8          /chene {0.568 0.427 0.086 setrgbcolor} def
9          /bois {0.956 0.921 0.65 setrgbcolor} def
10 }%
11 \codejps{
12 /H1 {
13 2 [18 16] newsphere
14 {-90 0 0 rotate0point3d} solidtransform
15 {0 10.93 0 translatepoint3d} solidtransform
16 dup (hetre) outputcolors} def
17 /L1 {
18 0 0.25 10 [12 10] newcylindre
19 {-90 0 0 rotate0point3d} solidtransform
20 dup (bois) outputcolors
21 } def
22 /HL1{ H1 L1 solidfuz} def
23 /HL2 { HL1 {0 0 -109.5 rotate0point3d}
          solidtransform } def
24 /HL3 { HL2 {0 -120 0 rotate0point3d}
          solidtransform } def
25 /HL4 { HL2 {0 120 0 rotate0point3d} solidtransform
          } def
26 /C {3 [18 16] newsphere
27 {90 0 0 rotate0point3d} solidtransform
28 dup (chene) outputcolors} def
29 /HL12 { HL1 HL2 solidfuz} def
30 /HL123 { HL12 HL3 solidfuz} def
31 /HL1234 { HL123 HL4 solidfuz} def
32 /methane { HL1234 C solidfuz} def
33 methane drawsolid**}
34 \end{pspicture}

```

9. Advanced usage

9.4.9. The thiosulphate ion



We first define the two sulphur atoms and place them on the Oz axis. S_1 is placed at the origin O .

```
\code{jps{
/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
```

Then the single bond S-O using the following convention: half red—the half connected to O, and half yellow—the half connected to 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
```

The oxygen atom, its bond, and the setting of the combined unit:

```
/Ox {2.17 [20 16] newsphere
    {0 0 15 translatepoint3d} solidtransform
    dup (Red) outputcolors} def
/L0 { Liaison Ox solidfuz} def
/L01 { L0 {0 -109.5 0 rotate0point3d} solidtransform } def
/L0x1 { L01 {0 0 120 rotate0point3d} solidtransform } def
% fin liaison simple S-O
```

For the double bond S=O, we take the single bond above and duplicate it with shifts of 0.75 cm along the *Ox* axis.

```
% 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
```

Connecting it to the O atom:

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

and with two successive rotations we position the two bonds =O:

```
/LiaisonDOx1 {LiaisonDOx {0 -109.5 0 rotate0point3d} solidtransform } def
/LiaisonDOx2 {LiaisonDOx1 {0 0 -120 rotate0point3d} solidtransform } def
```

The following step consists of fusing the two connections:

```
/L012 { LiaisonDOx1 LiaisonDOx2 solidfuz} def
/L0123 {L012 L0x1 solidfuz} def
```

Then the single bond S-S is created:

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

and fused with the two atoms S-S:

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

The last step will be to fuse the two S-S and the three O already equipped with their bonds:

```
/S203 { S1S2L4 L0123 solidfuz} def
S203  drawsolid**}
```

9. Advanced usage

CHAPTER 10

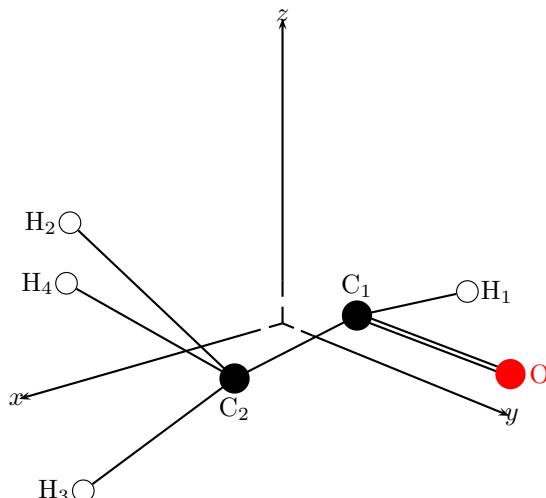
Interaction with PSTricks

10.1. Positioning a named point

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

This is a command similar to `\pnode(! x y){name}`. It places the node (`name`) at the point with the coordinates (x, y, z) , viewed with the chosen point of view `viewpoint=vx vy vz`. We can now use the point to mark it, draw lines, polygons, etc.

Let's place the centres of the atoms of the methanol molecule CH₃COH.



```
1 \begin{pspicture}(-4,-4)(4,5)
2 \psset{viewpoint=100 50 20 rtp2xyz,Decran=20}
3 \axesIIID(3,3,3)(20,20,20)
4 \psPoint(-4.79,2.06,0){C1}
5 \psPoint(-4.79,15.76,0){Ox}
6 \psPoint(8.43,5.57,0){C2}
7 \psPoint(-14.14,3.34,0){H3}
8 \psPoint(14.14,-2.94,8.90){H6}
9 \psPoint(14.14,-2.94,-8.90){H7}
10 \psPoint(6.43,-16.29,0){H8}
11 \psline(C1)(H3)\psline(C2)(H7)
12 \psline(C2)(H8)\psline(C1)(C2)
13 \psline[doubleline=true](C1)(Ox)
14 \psline(C2)(H6)
15 \uput[r](H3){$\mathrm{H_1}$}
16 \uput[l](H6){$\mathrm{H_2}$}
17 \uput[l](H7){$\mathrm{H_3}$}
18 \uput[l](H8){$\mathrm{H_4}$}
19 \uput{0.25}[u](C1){$\mathrm{C_1}$}
20 \uput{0.25}[d](C2){$\mathrm{C_2}$}
21 \uput{0.25}[r](Ox){$\mathrm{O}$}
22 \psdots[dotstyle=o,dotsize=0.3](H3)(H6)(H7)(H8)
23 \psdots[dotsize=0.4](C1)(C2)
24 \psdot[linecolor=red,dotsize=0.4](Ox)
25 \end{pspicture}
```

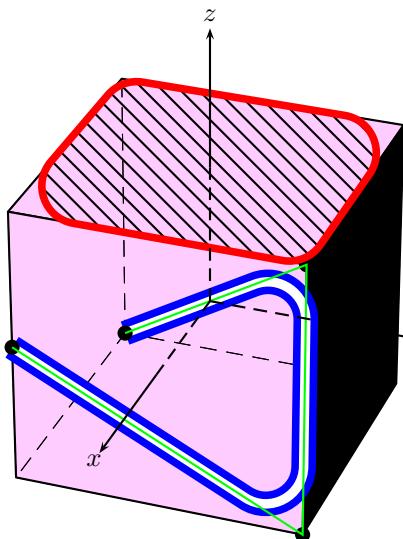
10.2. Drawing a line

This command is adapted from the macro `\pstThreeDLine` from the package `pst-3dplot` of Herbert Voss¹

We use `\psLineIID[options](x0,y0,z0)(x1,y1,z1)…(xn,yn,zn)`, with the following possible options:

- `linecolor=colour;`
- `doubleline=true;`
- `lineararc=value.`

It is not possible to put arrowheads at the ends of the lines.



```

1 \psset{viewpoint=50 20 30 rtp2xyz,Decran=50}
2 \begin{pspicture}(-3,-4)(4,4)
3 \psSolid[object=cube,a=4,action=draw*,
4   fillcolor=magenta!20]%
5 \psLineIID[linecolor=blue,
6   linewidth=0.1,lineararc=0.5,
7   doubleline=true](-2,-2,-2)(2,2,2)(2,2,-2)(2,-2,0)
8 \psPoint(2,-2,0){A}\psPoint(-2,-2,-2){B}
9 \psPoint(2,2,2){C}\psPoint(2,2,-2){D}
10 \psdot[dotsize=0.2](A)\psdot[dotsize=0.2](B)
11 \psdot[dotsize=0.2](C)\psdot[dotsize=0.2](D)
12 \psLineIID[linecolor=green]%
13 (-2,-2,-2)(2,2,2)(2,2,-2)(2,-2,0)
14 \psPolygonIID[linecolor=red,
15   fillstyle=vlines,lineararc=0.5,
16   linewidth=0.1](-2,-2,2)(-2,2,2)(2,2,2)(2,-2,2)
17 \axesIID(2,2,2)(4,4,4)
18 \end{pspicture}
```

10.3. Drawing a polygon

We use: `\psPolygonIID[options](x0,y0,z0)(x1,y1,z1)…(xn,yn,zn)`, with the possible options that follow:

- `linecolor=color;`
- `doubleline=true;`
- `lineararc=value;`
- `fillstyle=solid;`
- `fillstyle=vlines or fillstyle=hlines or fillstyle=crosshatch.`

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

10.4. Transformations to a point

Given is an initial point $A(x, y, z)$. Now we make some rotations around the axes Ox , Oy and Oz with the appropriate angles (in degrees): [RotX=valueX, RotY=valueY, RotZ=valueZ], in this order, then translate it with the vector (v_x, v_y, v_z) . The problem is to get back the coordinates of the image (final point) $A'(x', y', z')$.

The code `\psTransformPoint [RotX=valueX, RotY=valueY, RotZ=valueZ] (x y z) (vx,vy,vz){A'}` now allows us to save the node A' , the coordinates of the transformed point.

In the following example, $A(2, 2, 2)$ is one of the vertices of the initial cube, where the centre is placed at the origin.

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

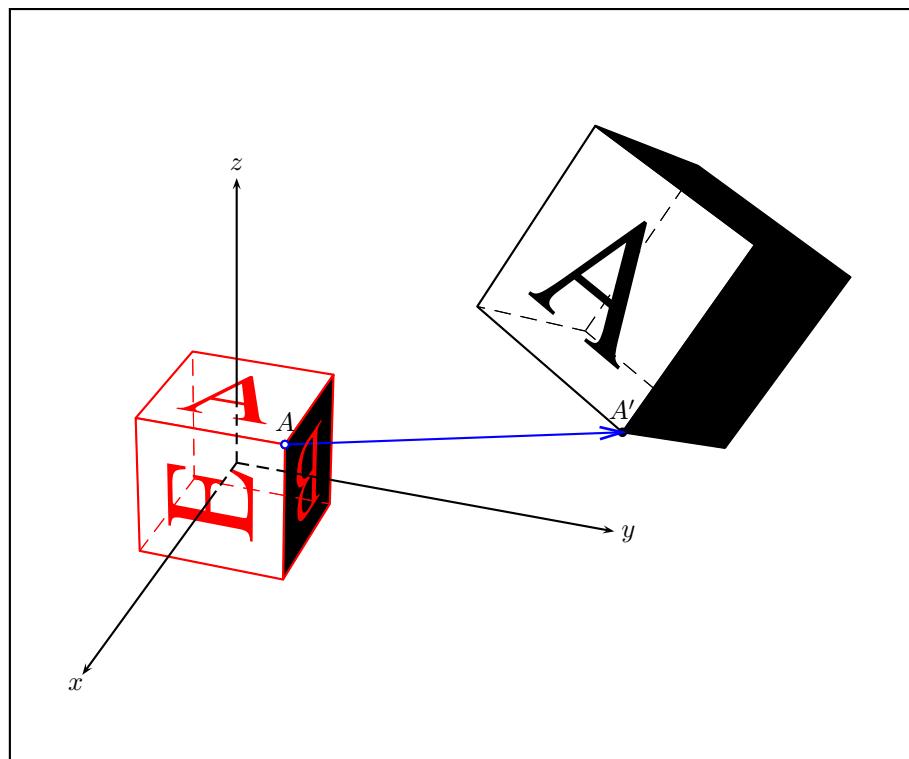
Some transformations are applied to the cube:

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

To obtain the image of A , we use the following command:

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

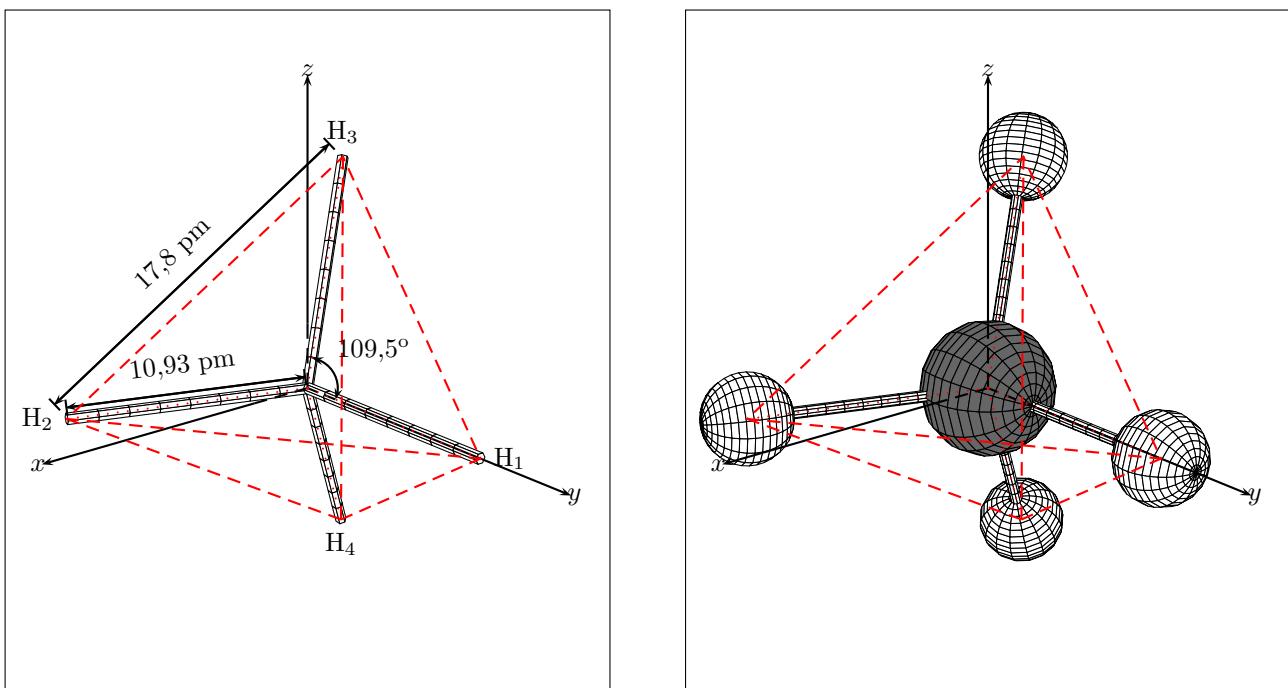
This allows us, for example, to name these points and then draw the vector $\overrightarrow{AA'}$.



10.5. Adding dimensions to the scenery

It is very interesting to add dimensions to the scenery. We take the example of the methane molecule, where we want to insert the distances and angles.

The first step consists of representing the molecule with its bonds and characteristic dimensions, and then draw it in a good looking way.



The construction of the molecule is detailed in the document `molecules.tex`. To add a dimensioning you only need to find the vertices of the tetrahedron:

```
\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}
```

and then use the power of the package `pst-node`. For the distances:

```
\pcline[offset=0.25]{<->}(H2)(H3)
\aput[:U]{17,8 pm}
\pcline[offset=0.15]{<->}(H2)(0)
\aput[:U]{10,93 pm}
\psPoint(-5.15,-3.64,-8.924){H4}
```

Then, for the angles, we take help from the package `pst-eucl`

```
\pstMarkAngle[arrows=<->]{H1}{0}{H3}{\small 109,5^{\circ}}
```

CHAPTER 11

Projections

11.1. Presentation

The package allows the representation and manipulation of some simple objects in two dimensions (2D). The macro `\psProjection` can project these 2D objects onto a chosen plane.

The syntax is analogous to that of `\psSolid`, with an obligatory option `object`, that allows us to specify the type of object to be projected.

The general syntax is `\psSolid[object=objectname,plan=plantype,<options>](x,y)`

11.2. The parameter visibility

For all projections, the Boolean `visibility` (`true` by default) specifies whether or not to have the projection made visible.

Set to `false`, the projection is always carried out. Set to `true`, the projection is only carried out when the plane of projection is visible from the viewpoint of the observer.

11.3. Defining a projection plane

The plane of projection is defined with the option `plan=plantype` which expects an argument *type of plane*. The creation of such an argument invariably happens through the command `\psSolid[object=plan]` (see the relevant paragraph of chapter 4 and the example below in sub-paragraph *Labels* of the paragraph *Points*).

11.4. Points

11.4.1. Direct definition

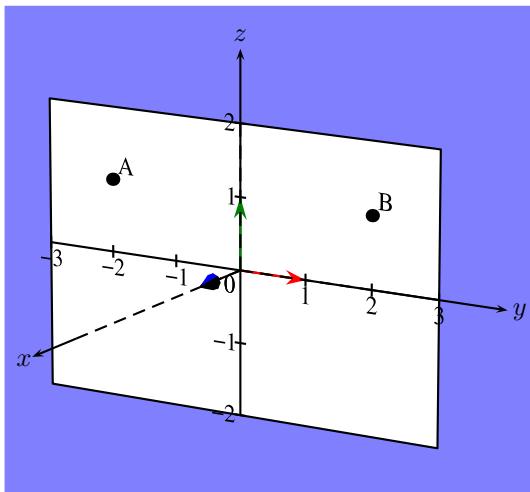
The object point defines a point. The values (x, y) of its coordinates can be passed directly to the macro `\psProjection` or indirectly via the option `args`.

Thus the two commands `\psProjection[object=point](1,2)` and `\psProjection[object=point,arg=1 2]` are equivalent and lead to the projection of the point with coordinates $(1, 2)$ onto the chosen plane.

11.4.2. Labels

The option `text=my text` allows us to project a string of characters onto the chosen plane next to a chosen point. The positioning is made with the argument `pos=value` where `value` is one of the following `{ul, cl, bl, dl, ub, cb, bb, db, uc, cc, bc, dc, ur, cr, br, dr}`.

The details of the parameter `pos` will be discussed in a later paragraph.



```

1 \begin{pspicture}(-3,-3)(4,3.5)%
2 \psframe*[linecolor=blue!50](-3,-3)(4,3.5)
3 \psset{viewpoint=50 30 15,Decran=60}
4 \psset{solidmemory}
5 %% definition du plan de projection
6 \psSolid[object=plan,
7   definition=equation,
8   args={[1 0 0 0] 90},
9   name=monplan,
10  planmarks,
11  showBase]
12 \psset{plan=monplan}
13 %% definition du point A
14 \psProjection[object=point,
15   args=-2 1,
16   text=A,
17   pos=ur]
18 \psProjection[object=point,
19   text=B,
20   pos=ur](2,1)
21 \composeSolid
22 \axesIIID(4,2,2)(5,4,3)
23 \end{pspicture}

```

11.4.3. Naming and memorising a point

If the option `name=myName` is given, the coordinates (x, y) of the chosen point are saved under the name `myName` and so can be reused.

11.4.4. Some other definitions

There are other methods to define a point in 2D. The options `definition` and `args` support the following methods:

- `definition=milieu; args=A B.`

The midpoint of the line segment $[AB]$

- `definition=parallelopoin; args=A B C.`

The point D for which $(ABCD)$ is a parallelogram.

- `definition=translatepoint; args=M u.`

The image of the point M shifted by the vector \vec{u}

- `definition=rotatepoint; args=M I r.`

The image of the point M under a rotation about the point I through an angle r (in degrees)

- `definition=hompoint; args=M A k.`

The point M' satisfying $\overrightarrow{AM'} = k\overrightarrow{AM}$

- `definition=orthoproj; args=+M d.`

The orthogonal projection of the point M onto the line d .

- `definition=projx; args=M.`

The projection of the point M onto the Ox axis.

- `definition=projy; args=M.`

The projection of the point M onto the Oy axis.

- `definition=sympoint; args=M I.`

The point of symmetry of M with respect to the point I .

- `definition=axesympoint; args=M d.`

The axially symmetrical point of M with respect to the line d .

- `definition=cpoint; args=α C.`

The point corresponding to the angle $α$ on the circle C

- `[definition=xdpoin]; args=x d.`

11. Projections

The Ox intercept x of the line d .

- **definition=ydpoint; args=y d.**

The Oy intercept y of the line d .

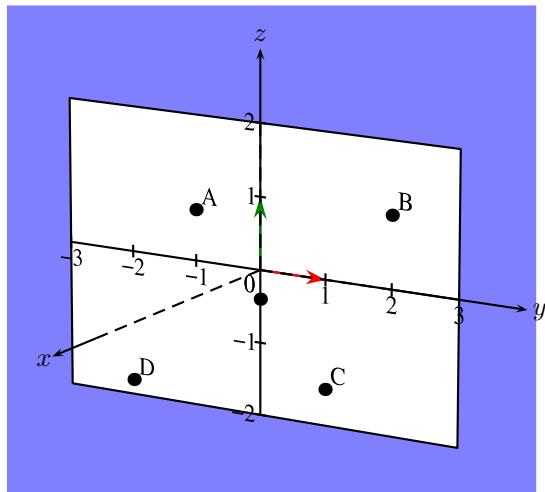
- **definition=interdroite; args=d₁ d₂.**

The intersection point of the lines d_1 and d_2 .

- **definition=interdroitecercle; args=d I r.**

The intersection points of the line d with a circle of centre I and radius r .

In the example below, we define and name three points A , B and C , and then calculate the point D for which $(ABCD)$ is a parallelogram together with the centre of this parallelogram.



```

1 \begin{pspicture}(-3,-3)(4,3.5)%
2 \psframe*[linecolor=blue!50](-3,-3)(4,3.5)
3 \psset{viewpoint=50 30 15,Decran=60}
4 \psset{solidmemory}
5 %% definition du plan de projection
6 \psSolid[object=plan,
7   definition=equation,
8   args={[1 0 0 0] 90},
9   name=monplan,
10  planmarks,
11  showbase]
12 \psset{plan=monplan}
13 %% definition du point A
14 \psProjection[object=point,
15   text=A, pos=ur, name=A](-1,.7)
16 %% definition du point B
17 \psProjection[object=point,
18   text=B, pos=ur, name=B](2,1)
19 %% definition du point C
20 \psProjection[object=point,
21   text=C, pos=ur, name=C](1,-1.5)
22 %% definition du point D
23 \psProjection[object=point,
24   definition=parallelpoint,
25   args=A B C,
26   text=D, pos=ur, name=D]
27 %% definition du point G
28 \psProjection[object=point,
29   definition=milieu,
30   args=D B]
31 \composeSolid
32 \axesIIID(4,2,2)(5,4,3)
33 \end{pspicture}

```

11.5. Vectors

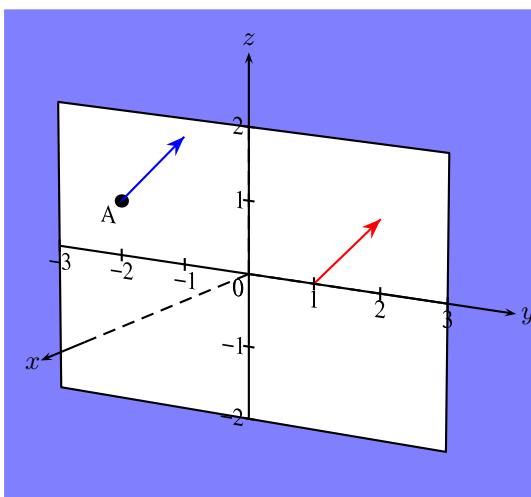
11.5.1. Direct definition

The object `vecteur` allows us to define and draw a vector. To do so in a simple way, we use the option `args` to define its components (x, y) and we specify the point from where the vector starts with the macro `\psProjection` (or we may use a named point).

As with points, we can save the components of a vector using the option `name`.

```

1 \begin{pspicture}(-3,-3)(4,3.5)%
2 \psframe*[linecolor=blue!50](-3,-3)(4,3.5)
3 \psset{viewpoint=50 30 15,Decran=60}
4 \psset{solidmemory}
5 %% definition du plan de projection
6 \psSolid[object=plan,
7   definition=equation,
8   args={[1 0 0 0] 90},
9   planmarks,
10  name=monplan]
11 \psset{plan=monplan}
12 %% definition du point A
13 \psProjection[object=point,
14   args=-2 0.75,
15   name=A,text=A,
16   pos=d1]
17 \psProjection[object=vecteur,
18   linecolor=red,
19   args=1 1,
20   name=U](1,0)
21 \psProjection[object=vecteur,
22   args=U,
23   linecolor=blue](A)
24 \composeSolid
25 \axesIIID(4,2,2)(5,4,3)
26 \end{pspicture}
```



11.5.2. Some more definitions

There are other methods to define a vector in 2D. The options `definition` and `args` allow us a variety of supported methods:

- `definition=vecteur; args=A B.`

The vector \overrightarrow{AB}

- `definition=orthovecteur; args=u.`

A vector perpendicular to \vec{u} with the same length.

- `definition=normalize; args=u.`

11. Projections

The vector $\|\vec{u}\|^{-1}\vec{u}$ if $\vec{u} \neq \vec{0}$, and $\vec{0}$ otherwise.

- **definition=addv;** **args=u v.**

The vector $\vec{u} + \vec{v}$

- **definition=subv;** **args=u v.**

The vector $\vec{u} - \vec{v}$

- **definition=mulv;** **args=u α .**

The vector $\alpha\vec{u}$

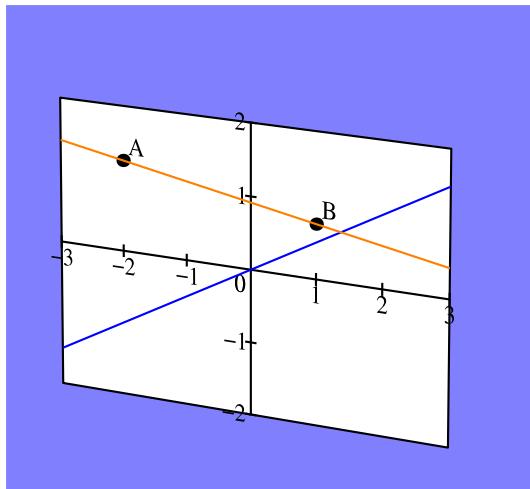
11.6. Lines

11.6.1. Direct definition

The object **droite** allows us to define and draw a line. In the **pst-solides3d** package, a line in 2D is defined by its two end-points.

We use the option **args** to specify the end-points of the chosen line. We can use coordinates or named points.

As with points and vectors, we can save the coordinates of the line with the option **name**.



```

1 \begin{pspicture}(-3,-3)(4,3.5)%
2 \psframe*[linecolor=blue!50](-3,-3)(4,3.5)
3 \psset{viewpoint=50 30 15,Decran=60}
4 \psset{solidmemory}
5 %% definition du plan de projection
6 \psSolid[object=plan,
7   definition=equation,
8   args={[1 0 0 0] 90},
9   planmarks,name=monplan]
10 \psset{plan=monplan}
11 %% definition du point A
12 \psProjection[object=point,
13   name=A,text=A,
14   pos=ur](-2,1.25)
15 \psProjection[object=point,
16   name=B,text=B,
17   pos=ur](1,.75)
18 \psProjection[object=droite,
19   linecolor=blue,
20   args=0 0 1 .5]
21 \psProjection[object=droite,
22   linecolor=orange,
23   args=A B]
24 \composeSolid
25 \end{pspicture}

```

11.6.2. Some other definitions

There are other methods to define a line in 2D. The options `definition` and `args` are used in these variants:

- `definition=horizontale; args=b.`

The line with equation $y = b$.

- `definition=verticale; args=a.`

The line with equation $x = a$.

- `definition=paral; args=d A.`

A line parallel to d passing through A .

- `definition=perp; args=d A.`

A line perpendicular to d passing through A .

- `definition=mediatrice; args=A B.`

The perpendicular bisector of the line segment $[AB]$.

- `definition=bissectrice; args=A B C.`

The bisector of the angle \widehat{ABC} .

- `definition=axesymdroite; args=d D.`

The reflection of the line d in the line D .

- `definition=rotatedroite; args=d I r.`

The image of the line d after a rotation with centre I through an angle r (in degrees)

- `definition=translatedroite; args=d u.`

The image of the line d shifted by the vector \vec{u} .

11.7. Circles

11.7.1. Direct definition

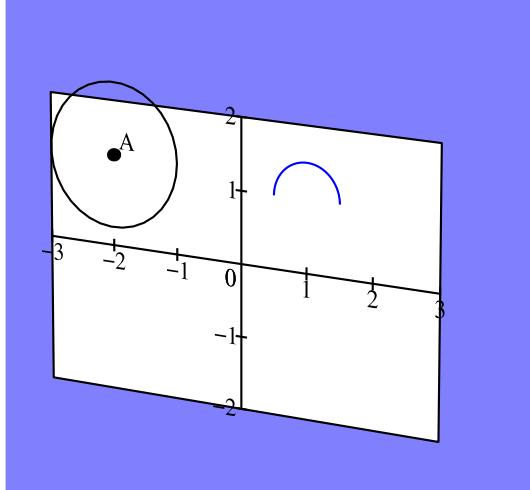
The object `cercle` allows us to define and draw a circle. In the `pst-solides3d` package, a circle in 2D is defined by its centre and radius.

We use the option `args` to specify the centre and radius of the chosen circle. We can use coordinates or named variables.

11. Projections

The argument `range=tmin tmax` allows us to specify an arc of the chosen circle.

As for all the other object, we can save the circle data using the option `name`.



```
1 \begin{pspicture}(-3,-3)(4,3.5)%
2 \psframe*[linecolor=blue!50](-3,-3)(4,3.5)
3 \psset{viewpoint=50 30 15,Decran=60}
4 \psset{solidmemory}
5 %% definition du plan de projection
6 \psSolid[object=plan,
7   definition=equation,
8   args={[1 0 0 0] 90},
9   planmarks,
10 name=monplan]
11 \psset{plan=monplan}
12 %% definition du point A
13 \psProjection[object=point,
14   name=A,
15   text=A,
16   pos=ur](-2,1.25)
17 \psProjection[object=cercle,
18   args=A 1,
19   range=0 360]
20 \psProjection[object=cercle,
21   args=1 1 .5,linecolor=blue,
22   range=0 180]
23 \composeSolid
24 \end{pspicture}
```

11.7.2. Some other definitions

There are additional methods to define a circle in 2D. The options `definition` and `args` give the following supported methods:

- `definition=AB cercle; args=A B C.`

A circle through the points A , B and C .

- `definition=diam cercle; args=A B.`

A circle with diameter $[AB]$.

11.8. Polygons

11.8.1. Direct definition

The object `polygone` allows us to define a polygon. We use the option `args` to specify the list of vertices: `[object=polygone,args=A0 A1 ... An]`

There are other ways to define a polygon in 2D. The options `definition` and `args` support these methods:

- **definition=translatepol; args=pol u.**

Translation of the polygon pol by the vector \vec{u}

- **definition=rotatepol; args=pol I α.**

Image of the polygon pol after a rotation with centre I and angle $α$

- **definition=homopol; args=pol I α.**

Image of the polygon pol after a homothety (dilation) with centre I and ratio $α$.

- **definition=sympol; args=pol I.**

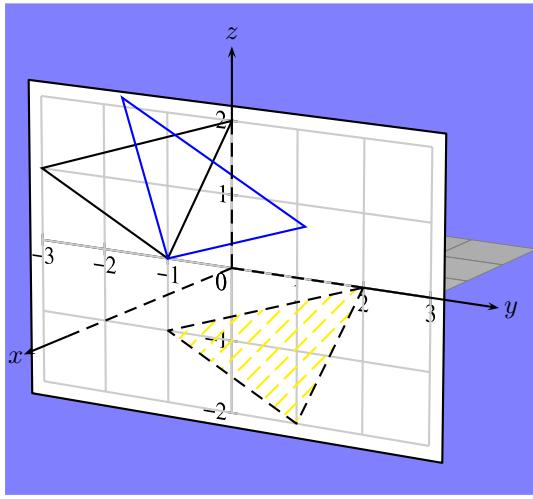
Image of the polygon pol after a reflection in the point I .

- **definition=axesympol; args=pol d.**

Image of the polygon pol after a reflection in the line d .

In the following example we define, name and draw the polygon with vertices $(-1, 0)$, $(-3, 1)$, $(0, 2)$, then—in blue—the image after a rotation about the point $(-1, 0)$ through an angle -45 . Finally, we translate the polygon with the vector shift $(2, -2)$ by directly incorporating *jps code* within the argument of **definition**.

11. Projections



```

1 \begin{pspicture}(-3,-3)(4,3.5)%
2 \psframe*[linecolor=blue!50](-3,-3)(4,3.5)
3 \psset{lightsrc=50 20 20,viewpoint=50 30 15,Decran
       =60}
4 \psset{solidmemory}
5 \psSolid[object=grille,
6   base=-3 0 -3 3,
7   linewidth=0.5\pslinewidth,linecolor=gray,]
8 %% definition du plan de projection
9 \psSolid[object=plan,
10   definition=equation,
11   args={[1 0 0 0] 90},
12   base=-3.2 3.2 -2.2 2.2,
13   name=monplan,
14   planmarks,
15 ]
16 \psset{plan=monplan}
17 \psSolid[object=plan,
18   args=monplan,
19   linecolor=gray!40,
20   plangrid,
21   action=none,
22 ]
23 \psProjection[object=polygone,
24   args=-1 0 -3 1 0 2,
25   name=P,
26 ]
27 \psProjection[object=polygone,
28   definition=rotatepol,
29   linecolor=blue,
30   args=P -1 0 -45,
31 ]
32 %% du code jps dans la definition
33 \psProjection[object=polygone,
34   definition={2 -2 addv} papply,
35   fillstyle=hlines,hatchcolor=yellow,
36   linestyle=dashed,
37   args=P,
38 ]
39 \composeSolid
40 \axesIIID(4,2,2)(5,4,3)
41 \end{pspicture}

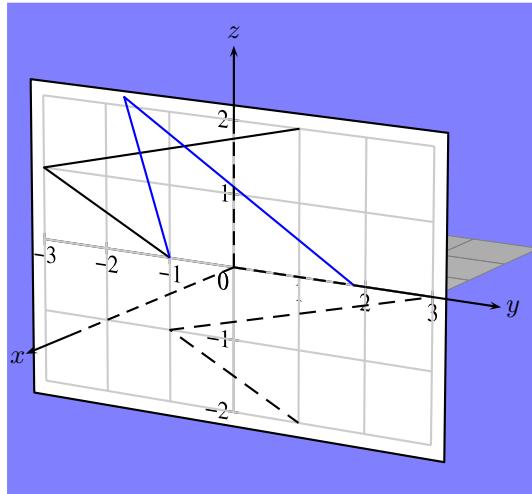
```

11.9. Lines

11.9.1. Direct definition

The object `line` defines a line (or a series of line segments). We use the option `args` to specify the points: `[object=line,args=A0 A1 ... An]`

We can also define a line that has been transformed using a translation, a rotation, a homothety, etc., as though it were a polygon.



```

1 \begin{pspicture}(-3,-3)(4,3.5)%
2 \psframe*[linecolor=blue!50](-3,-3)(4,3.5)
3 \psset{lightsrc=50 20 20,viewport=50 30 15,Decran
       =60}
4 \psset{solidmemory}
5 \psSolid[object=grille,
6   base=-3 0 -3 3,
7   linewidth=0.5\pslinewidth,linecolor=gray,]
8 %% definition du plan de projection
9 \psSolid[object=plan,
10   definition=equation,
11   args={[1 0 0 0] 90},
12   base=-3.2 3.2 -2.2 2.2,
13   name=monplan,
14   planmarks]
15 \psset{plan=monplan}
16 \psSolid[object=plan,
17   args=monplan,
18   linecolor=gray!40,
19   plangrid,
20   action=none]
21 \psProjection[object=line,
22   args=-1 0 -3 1 1 2,
23   name=P]
24 \psProjection[object=line,
25   definition=rotatepol,
26   linecolor=blue,
27   args=P -1 0 -45]
28 %% du code jps dans la definition
29 \psProjection[object=line,
30   definition={2 -2 addv} papply,
31   linestyle=dashed,
32   args=P]
33 \composeSolid
34 \axesIIID(4,2,2)(5,4,3)
35 \end{pspicture}

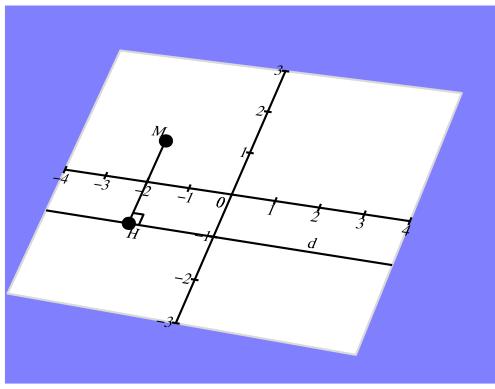
```

11.10. Right angle

11.10.1. Direct definition

The object `rightangle` allows us to specify and draw a right angle. The syntax is: `[object=rightangle,args=A B C]`

11. Projections



```

1 \begin{pspicture}(-3,-2.5)(3.5,2.5)%
2 \psframe*[linecolor=blue!50](-3,-2.5)(3.5,2.5)
3 \psset{lightsrc=viewpoint,viewpoint=50 30 15,Decran
=40}
4 \psset{solidmemory}
5 %% definition du plan de projection
6 \psSolid[object=plan,
7   definition=equation,
8   args={[1 0 1 0] 90},
9   base=-4 4 -3 3,
10  fillcolor=white,
11  linecolor=gray!30,
12 % plangrid,
13  planmarks,
14  name=monplan]
15 \psset{plan=monplan,visibility=false}
16 %% definition droite d
17 \psProjection[object=droite,
18   definition=horizontale,
19   args=-1,name=d]
20 \psset{fontsize=15}
21 %% definition du point M
22 \psProjection[object=point,
23   args=-2 1,
24   name=M,text=M,
25   pos=ul]
26 %% definition du point H
27 \psProjection[object=point,
28   definition=orthoproj,
29   args=M d,
30   name=H,text=H,
31   pos=dr]
32 %% definition du point H' pour orienter l'angle droit
33 %% et mettre la legende
34 \psProjection[object=point,
35   definition=xzpoint,
36   args=2 d,name=H',
37   action=none,
38   text=d,pos=ur]
39 %% definition d'une ligne
40 \psProjection[object=line,
41   args=M H]
42 %% dessin angle droit
43 \psProjection[object=rightangle,
44   args=M H H']
45 \composeSolid
46 \%axesIIID(4,4,2)(5,5,6)
47 \end{pspicture}

```

11.11. Curves of real-valued and parameterised functions

11.11.1. Curve of a real-valued function

The object `courbe` allows us to draw a curve, where the name is given with the option `function`. This function, with values in \mathbb{R} , has to be defined by the macro `\defFunction` (see the appropriate paragraph for more details).

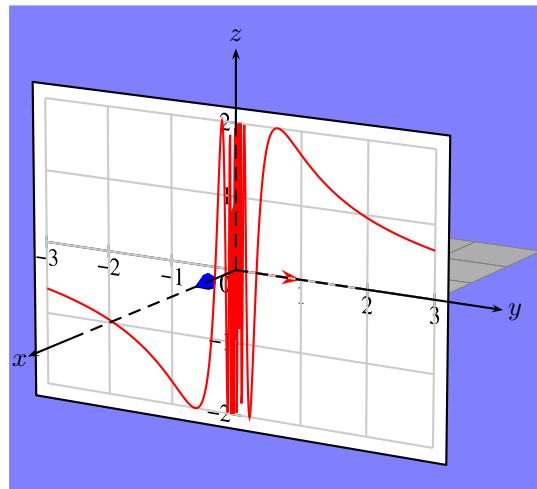
We can define this function either in algebraic notation, with the option `algebraic`, or in Reverse Polish Notation (RPN), with variables like $(x, u, t \dots)$, using an expression of the following form:

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

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

Note: This expression needs to be included within a `pspicture` environment.

The limits of the variables are defined by the option `range=xmin xmax`, and the option `argument=n` defines the number of points to be plotted when drawing the curve.



```

1 \begin{pspicture}(-3,-3)(4,3.5)%
2 \psframe*[linecolor=blue!50](-3,-3)(4,3.5)
3 \psset{lightsrc=50 20 20,viewpoint=50 30 15,Decran
       =60}
4 \psset{solidmemory}
5 \defFunction[algebraic]{1_sin}(x){2*sin(1/x)}{}{}
6 \psSolid[object=grille,
7   base=-3 0 -3 3,
8   linewidth=0.5\pslinewidth,linecolor=gray,]
9 %% definition du plan de projection
10 \psSolid[object=plan,
11   definition=equation,
12   args={[1 0 0 0] 90},
13   base=-3.2 3.2 -2.2 2.2,
14   planmarks,
15   showBase,
16   name=monplan]
17 \psset{plan=monplan}
18 \psSolid[object=plan,
19   args=monplan,
20   linecolor=gray!40,
21   plangrid,
22   action=none]
23 \psProjection[object=courbe,
24   linecolor=red,
25   range=-3 3,resolution=720,
26   function=1_sin]
27 \composeSolid
28 \axesIIID(4,2,2)(5,4,3)
29 \end{pspicture}
```

11.11.2. Parameterised curves

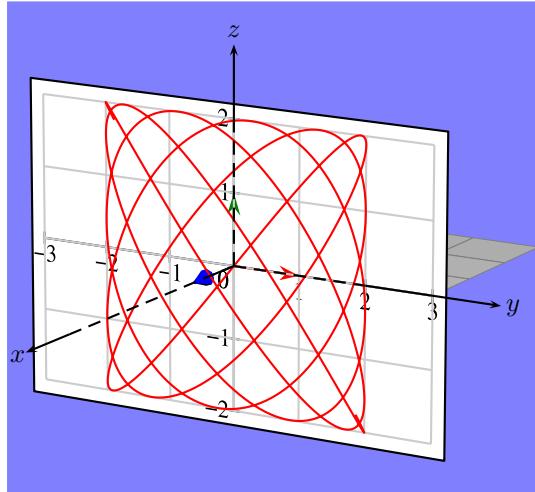
The technique used here is analogous to the above, with the difference that the values now come from \mathbb{R}^2 , and the object for the macro `\psProjection` is now `courbeR2`.

For example, to draw a circle of radius 3 and centre O , we type:

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

Another example: Lissajous curves.

11. Projections



```

1 \begin{pspicture}(-3,-3)(4,3.5)%
2 \psframe*[linecolor=blue!50](-3,-3)(4,3.5)
3 \psset{lightsrc=50 20 20,viewport=50 30 15,Decran
       =60}
4 \psset{solidmemory}
5 \defFunction[algebraic]{F}(t){2*sin(0.57735*t)}{2*sin
   (0.707*t){}}
6 \psSolid[object=grille,
7   base=-3 0 -3 3,
8   linewidth=0.5\pslinewidth,linecolor=gray,]
9 %% definition du plan de projection
10 \psSolid[object=plan,
11   definition=equation,
12   args={[1 0 0 0] 90},
13   base=-3.2 3.2 -2.2 2.2,
14   name=monplan,
15   planmarks,
16   showBase]
17 \psset{plan=monplan}
18 \psSolid[object=plan,
19   args=monplan,
20   linecolor=gray!40,
21   plangrid,
22   action=none]
23 \psProjection[object=courbeR2,
24   range=-25.12 25.12,resolution=720,
25   normal=1 1 2,linecolor=red,
26   function=F]
27 \composeSolid
28 \axesIIID(4,2,2)(5,4,3)
29 \end{pspicture}

```

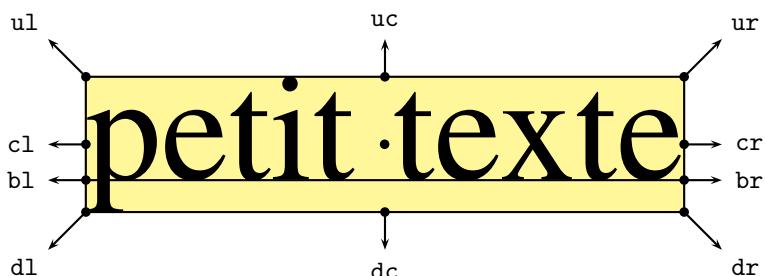
11.12. Text

The object `texte` of the macro `\psProjection` allows us to project character strings onto planes.

11.12.1. The parameters and the options

There are three parameters: `text` which defines the string, `fontsize`, which gives the dimension of the font in points (remember: 28.45 pts correspond to 1 cm), and finally `pos`, which defines the position of the text. By default, the text is centred at the origin of the plane.

This last parameter needs some explanation. See the string `petit texte` represented below.



We have 4 horizontal reference lines: the bottom line (**d**)own, the base line (**b**)aseline, the median line, or centre line (**c**)enter, and the upper line (**u**)p.

There are as well 4 vertical reference lines: the left line (**l**)eft, the base line (**b**)aseline, the centre line (**c**)enter and the right line (**r**)ight. In the case of strings, the two vertical lines **l** and **b** might be indistinguishable and easily confounded.

The intersection of the 4 horizontal lines with the 4 vertical lines gives us 16 positioning point possibilities **d1**, **b1**, **cl**, **ul**, **db**, **bb**, **cb**, **ub**, **dc**, **bc**, **cc**, **uc**, **dr**, **br**, **cr**, **ur**.

Of these, 4 are considered as *inner points*: **bb**, **bc**, **cb** and **cc**.

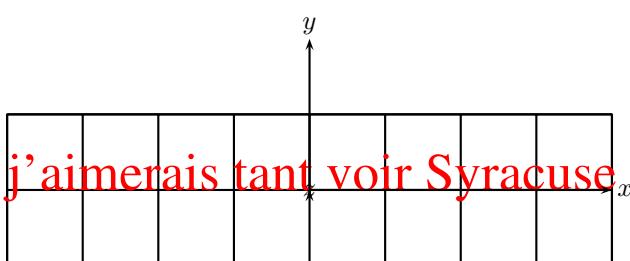
When the parameter **pos** of `\psProjection` is assigned one of these four inner points, it means that the latter will be situated at the origin of the plane of projection.

When the parameter **pos** of `\psProjection` is assigned one of the twelve remaining points, it indicates the direction in which the text will be positioned relative to the origin of the plane of projection.

For example, `\psProjection[...,pos=uc](0,0)` indicates that the text will be centred relative to the point $(0,0)$ and situated above it.

11.12.2. Examples of projecting onto a plane

Example 1: projection onto Oxy , with the option **pos=bc**

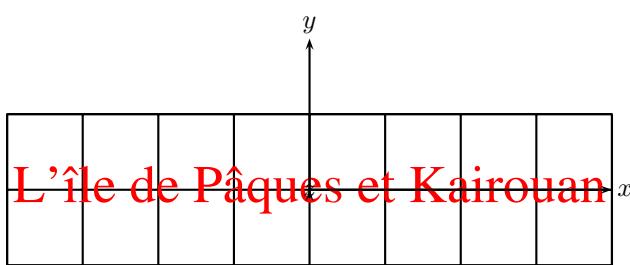


```

1 \begin{pspicture}(-4,-1.5)(4,1.5)
2 \psset{solidmemory}
3 \psset{lightsrc=10 0 10,
4   viewpoint=50 -90 89.99 rtp2xyz,Decran=50}
5 \psSolid[object=plan,definition=normalpoint,
       plangrid,
6   base=-4 4 -1 1,args={0 0 0 [0 0 1]},name=
       monplan,]
7 \psProjection[object=texte,
8   fontsize=20,linecolor=red,
9   pos=bc,plan=monplan,
10  text=j'aimerais tant voir Syracuse,
11 ](0,0)%
12 \axesIIID(0,0,0)(4,2,1)
13 \composeSolid
14 \end{pspicture}
```

Example 2: projection onto Oxy , centred text

11. Projections

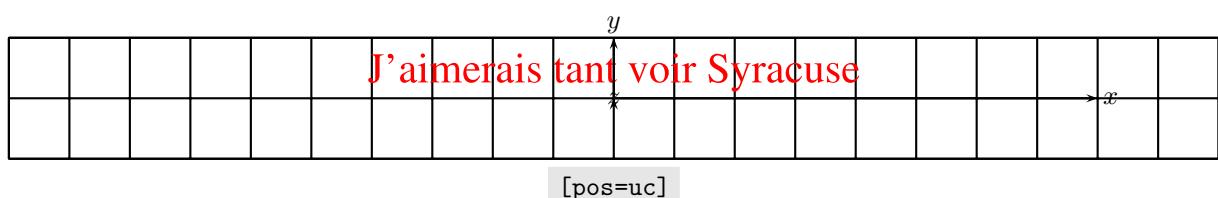
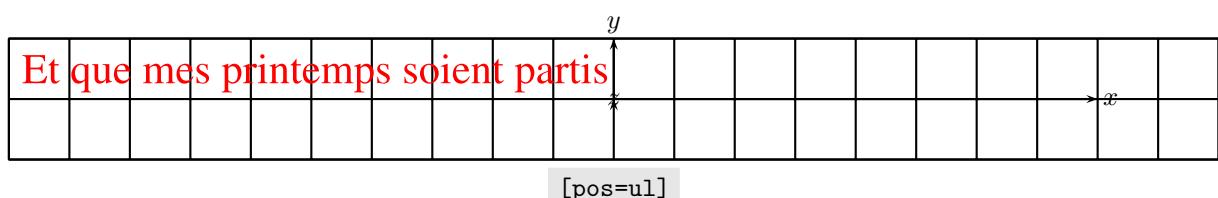
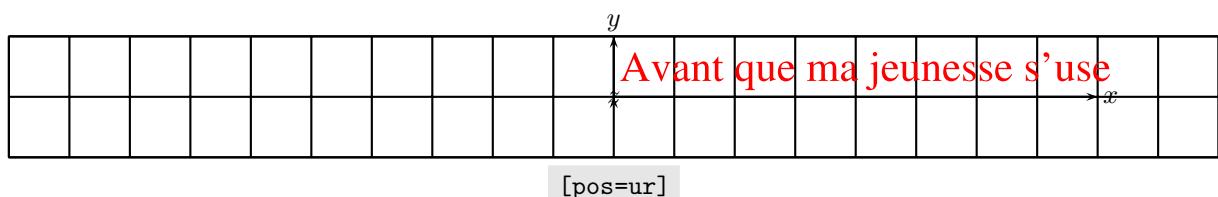
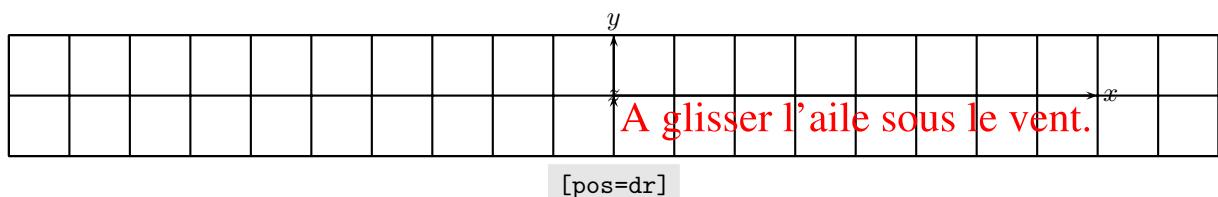
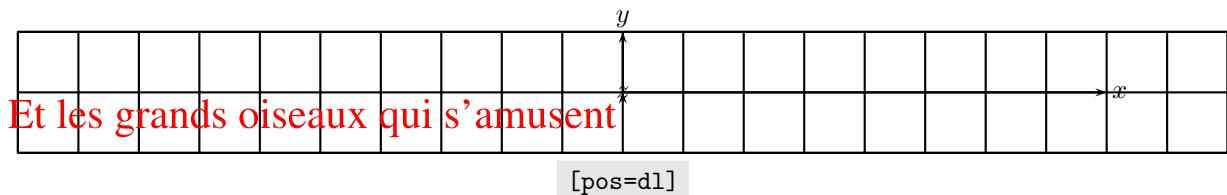


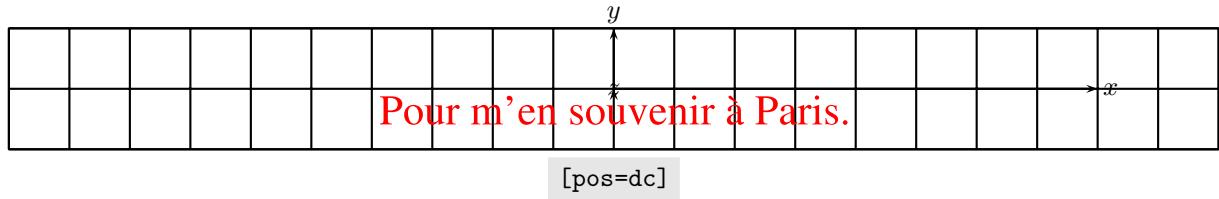
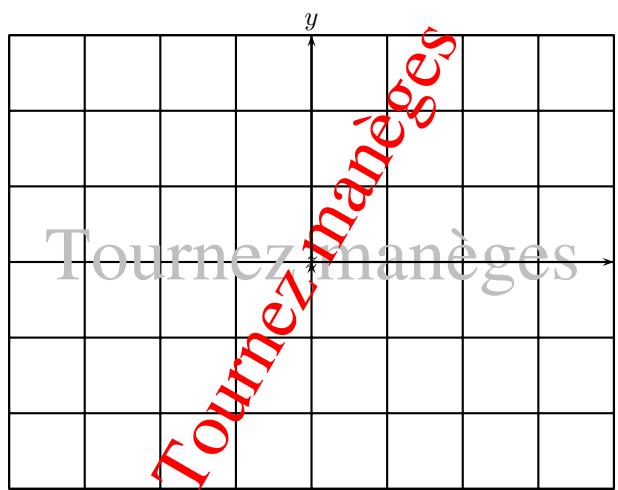
```

1 \begin{pspicture}(-4,-1.5)(4,1.5)
2 \psset{solidmemory}
3 \psset{lightsrc=10 0 10,
4   viewpoint=50 -90 89.99 rtp2xyz,,Decran=50}
5 \psSolid[object=plan,definition=normalpoint,
6   plangrid,
7   base=-4 4 -1 1,args={0 0 0 [0 0 1]},name=
8   monplan,]
9 \psProjection[object=texte,
10   fontsize=20,linecolor=red,
11   text=L'\^{\i}le de P\^{\a}ques et Kairouan,
12   plan=monplan]%
13 \axesIIID(0,0,0)(4,2,1)
14 \end{pspicture}

```

Example 3: projection onto Oxy , with different options pos=dl, etc.



**Example 4: projection onto Oxy with text rotation**

```

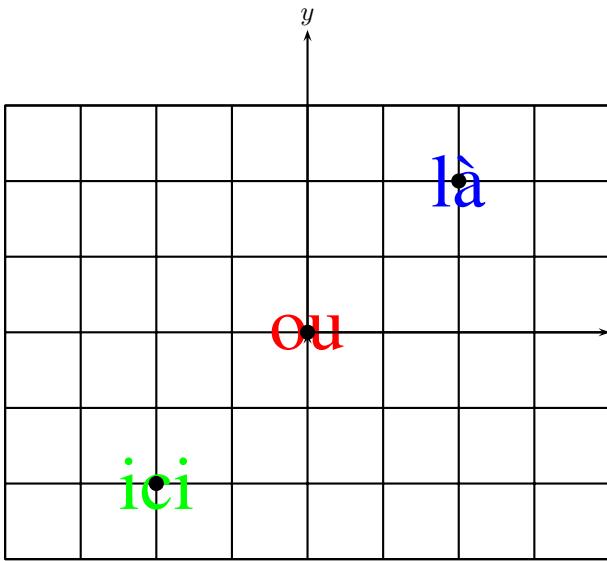
1 \begin{pspicture}(-4,-3)(4,3)
2 \psset{solidmemory}
3 \psset{lightsrc=10 0 10,
4   viewpoint=50 -90 89.99 rtp2xyz,Decran=50}
5 \psSolid[object=plan,definition=normalpoint,
6   plangrid,
7   base=-4 4 -3 3,args={0 0 0 [0 0 1]},name=
8     monplan,]
9 \psset{plan=monplan}
10 \psProjection[object=texte,
11   fontsize=28.45,linecolor=gray!50,
12   text=Tournez man\`{e}ges]%
13 \psProjection[object=texte,
14   fontsize=28.45,linecolor=red,
15   text=Tournez man\`{e}ges,
16   phi=60]%
17 \axesIIID(0,0,0)(4,3,1)
18 \end{pspicture}

```

The text rotation is introduced by the parameter phi=60.

Example 5: positioning text at a point

11. Projections



```

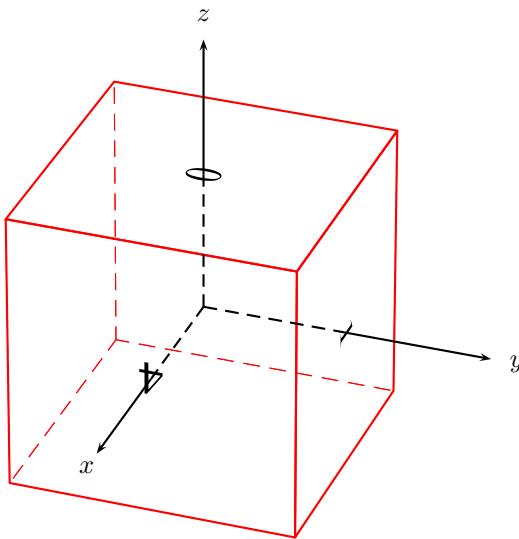
1 \begin{pspicture}(-4,-3)(4,3)
2 \psset{solidmemory}
3 \psset{viewpoint=50 -90 89.99 rtp2xyz,Decran=50}
4 \psSolid[object=plan,definition=normalpoint,
      plangrid,
5   base=-4 4 -3 3,args={0 0 0 [0 0 1]},name=
      monplan,]
6 \psset{fontsize=28.45,plan=monplan}
7 \psProjection[object=texte,
8   linecolor=green,
9   text=ici](-2,-2)
10 \psProjection[object=texte,
11   linecolor=red,
12   text=ou]%
13 \psProjection[object=texte,
14   linecolor=blue,
15   text=l\`{a}](2,2)
16 \psPoint(0,0,0){0}
17 \psPoint(-2,-2,0){01}
18 \psPoint(2,2,0){02}
19 \psdots[dotsize=0.2](0)(01)(02)
20 \axesIIID(0,0,0)(4,4,1)
21 \end{pspicture}

```

11.12.3. Examples for projecting onto a face of a solid

Method

The solid must be memorised with the general option `\psset{solidmemory}`. The first thing to do is to find the numbers of the faces of the solid with the option `numfaces=all`.



```

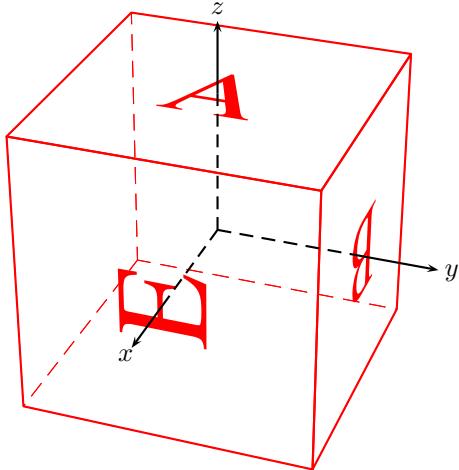
1 \psset{viewpoint=50 20 30 rtp2xyz,Decran=100}
2 \begin{pspicture}(-4,-4)(4,4)
3 \psSolid[object=cube,a=2,action=draw,
4   linecolor=red,numfaces=all]%
5 \axesIIID(1,1,1)(2,2,2)
6 \end{pspicture}

```

Then we define the projection plane as the chosen face, where in this case we put A on the face with the index number 0:

Then we define the projection plane by a chosen face, there we put A on the face with the index number 0:

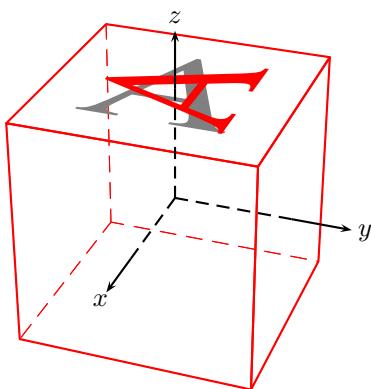
```
\psSolid[object=plan,definition=solidface,args=A 0,name=P0]
\psProjection[object=texte,linecolor=red,text=A,plan=P0]%
```



```

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

Text rotation with the option phi



```

1 \psset{viewpoint=50 20 30 rtp2xyz,Decran=50}
2 \psset{unit=0.4}
3 \begin{pspicture}(-8,-7)(4,9)
4 \psset{solidmemory}
5 \psSolid[object=cube,a=8,action=draw,linecolor=red,
         ,name=A]%
6 \psset{fontsize=200}
7 \psSolid[object=plan,action=none,
8   definition=solidface,args=A 0,name=P0]
9 \psProjection[object=texte,linecolor=gray,text=A,
    plan=P0]%
10 \psset{phi=90}
11 \psProjection[object=texte,linecolor=red,text=A,
    plan=P0]%
12 \axesIIID(4,4,4)(6,6,6)
13 \end{pspicture}
```

11.12.4. Examples of projecting onto different faces of a solid

We project a poem, verse by verse, onto 4 faces of a cube. It is necessary to use the option `solidmemory` at the beginning

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

11. Projections

of the code. We then define the cube, which is memorised with the help of the command `name=A`:

```
\psset{solidmemory}
\psProjection[object=texte,text=po\`eme,fontsize=30,plan=P0](0,3)%
\psSolid[object=cube,a=8,name=A](0,0,4.2)%
```

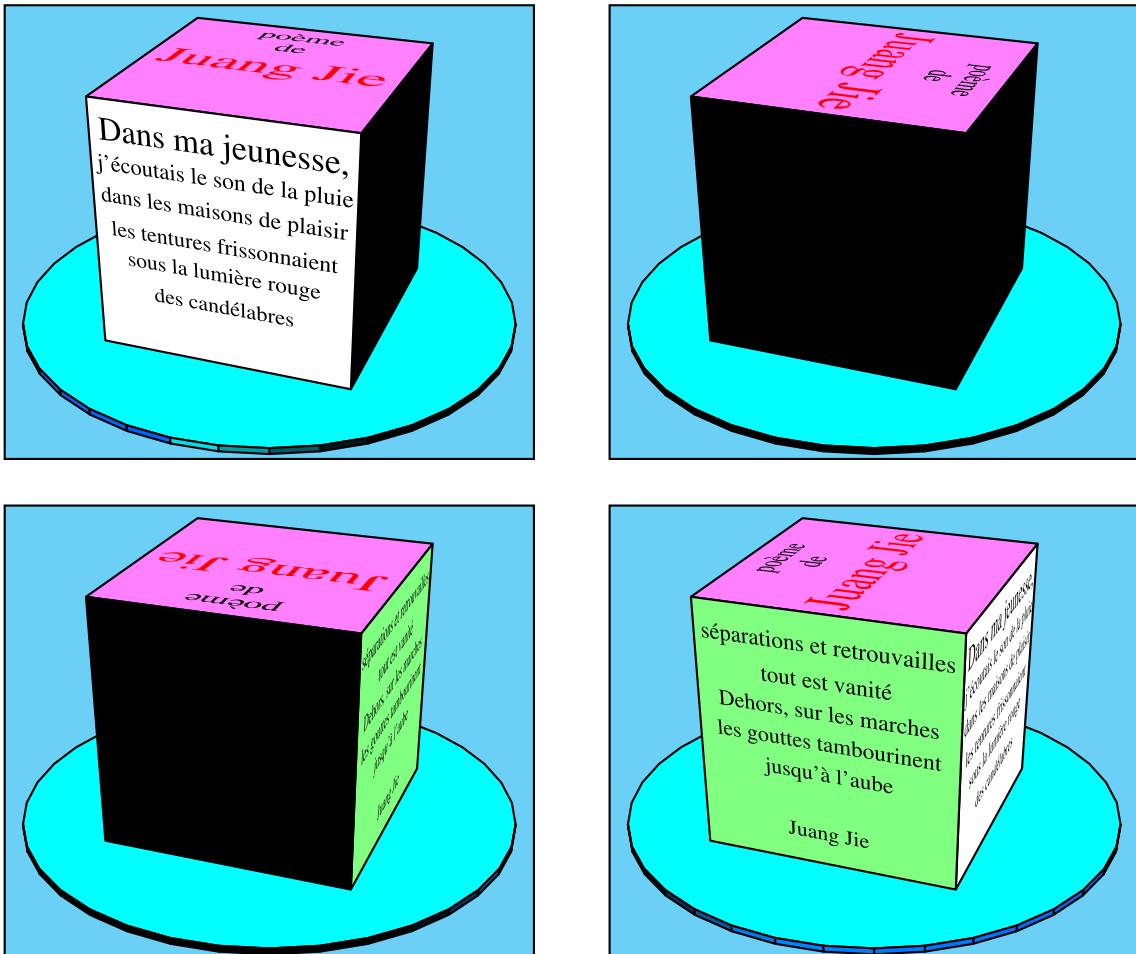
The number of each face needs to be known—from a previous run of the code with the option `numfaces=all`. The following commands:

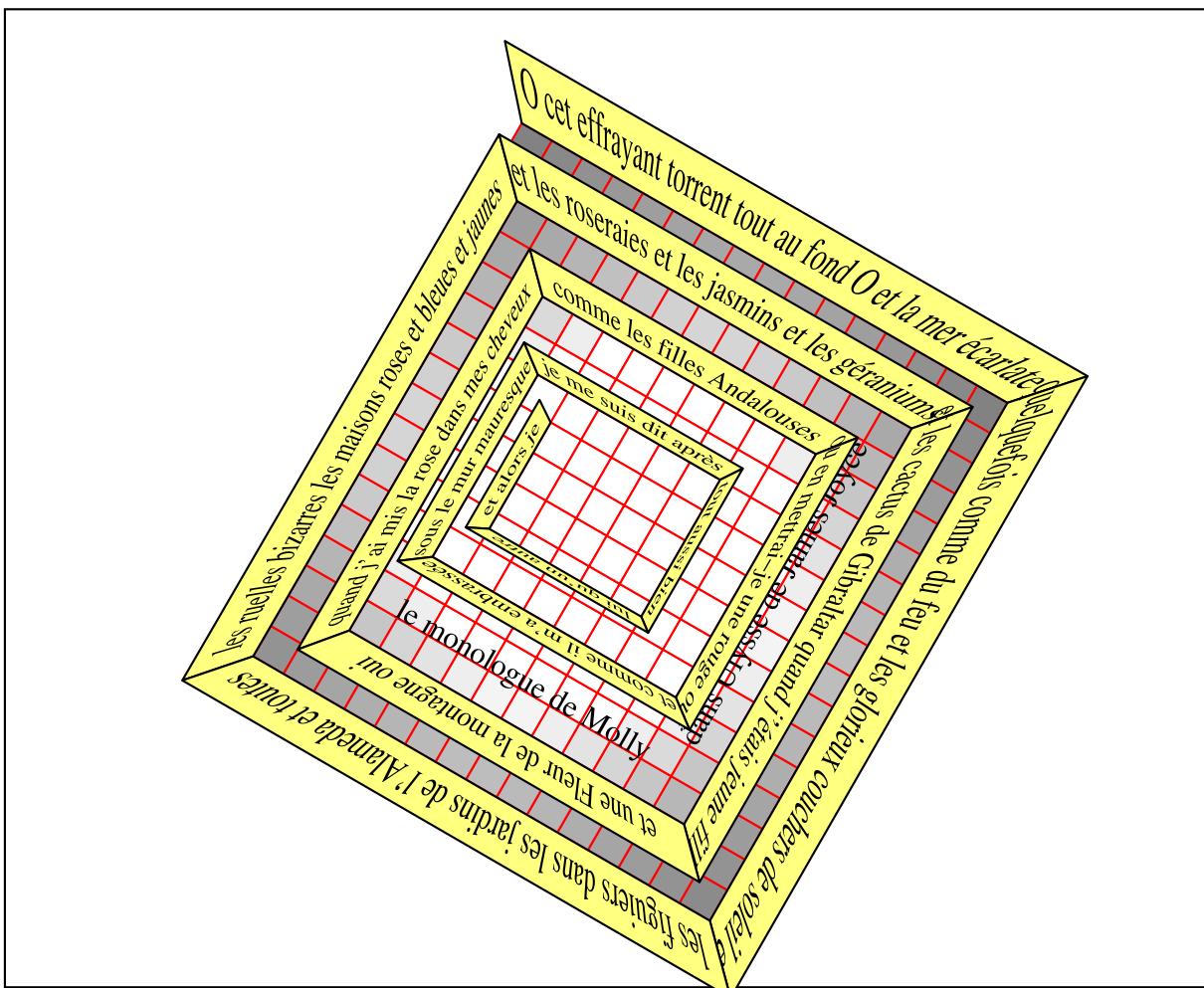
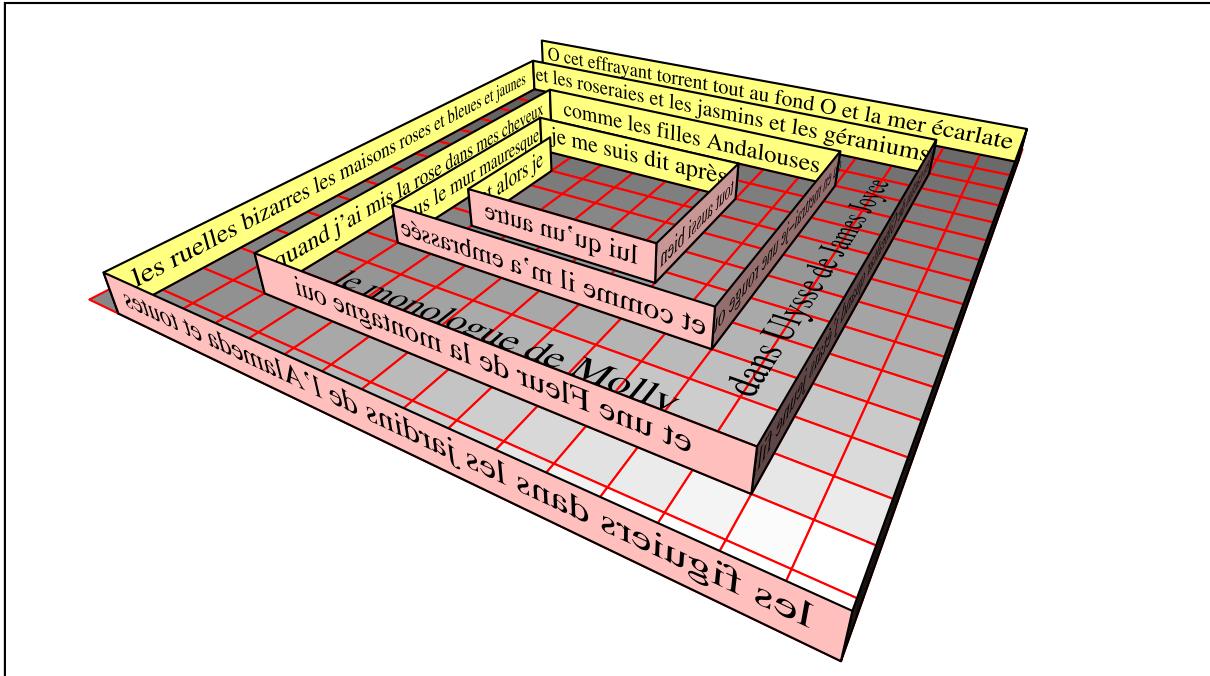
```
\psSolid[object=plan,action=none,definition=solidface,args=A 0,name=P0]%
\psProjection[object=texte,text=po\`eme,fontsize=30,plan=P0](0,3)%
```

define the plane P_0 as the oriented plane of the face with index number 0 of the solid A , before the word `poème` is projected onto P_0 , with a font size of 30 pts, to the point with coordinates $(0,3)$ (within the coordinate system of that plane). We could have changed the orientation of the text to `phi=-90` for example, in the one or other of the commands.

By default, if the face is not visible, its text stays hidden. By putting `visibility` in the options, the text is shown when it would otherwise not be, as in the following example.

You must not forget to write `\composeSolid` at the end of the text-writing commands for all these lines to be taken into account. Any other PStricks command will have the usual effect and `\composeSolid` will be unnecessary.





11.13. Projection of images

This command displays an eps image on a plane defined by an origin and a normal, this plan can be the face of a predefined object: a cube for example. The eps image must be prepared according to the method described in the documentation for ‘pst-anamorphosis’¹.

The macro includes various options:

```
\psImage[file=<filename with extension>,
         divisions=10,
         normale=nx ny nz,
         origine=x0 y0 z0,
         phi=angle,
         unitPicture=28.45] (x,y)
```

It focuses the image on the plane at the point defined by the origin, it may be moved to another point by setting the *optional* values (x,y). You can omit these values if we do not translate the image into another point than the origin of the plan.

divisions=20 selects the number of sub-segments for `lineto` in the image file to display. The higher the number, the higher the projected image will be faithful to the original. However, the projection takes place on a plane, the deformation will be small in all cases except one approaches very close to the plane, therefore a small number of sub-divisions will generally give a correct result and will perform calculations quickly .

phi can rotate the image of a fixed value in degrees.

unitImage=28.45 allows to resize the size of the eps image that is generally points per cm, a larger value will give a smaller image.

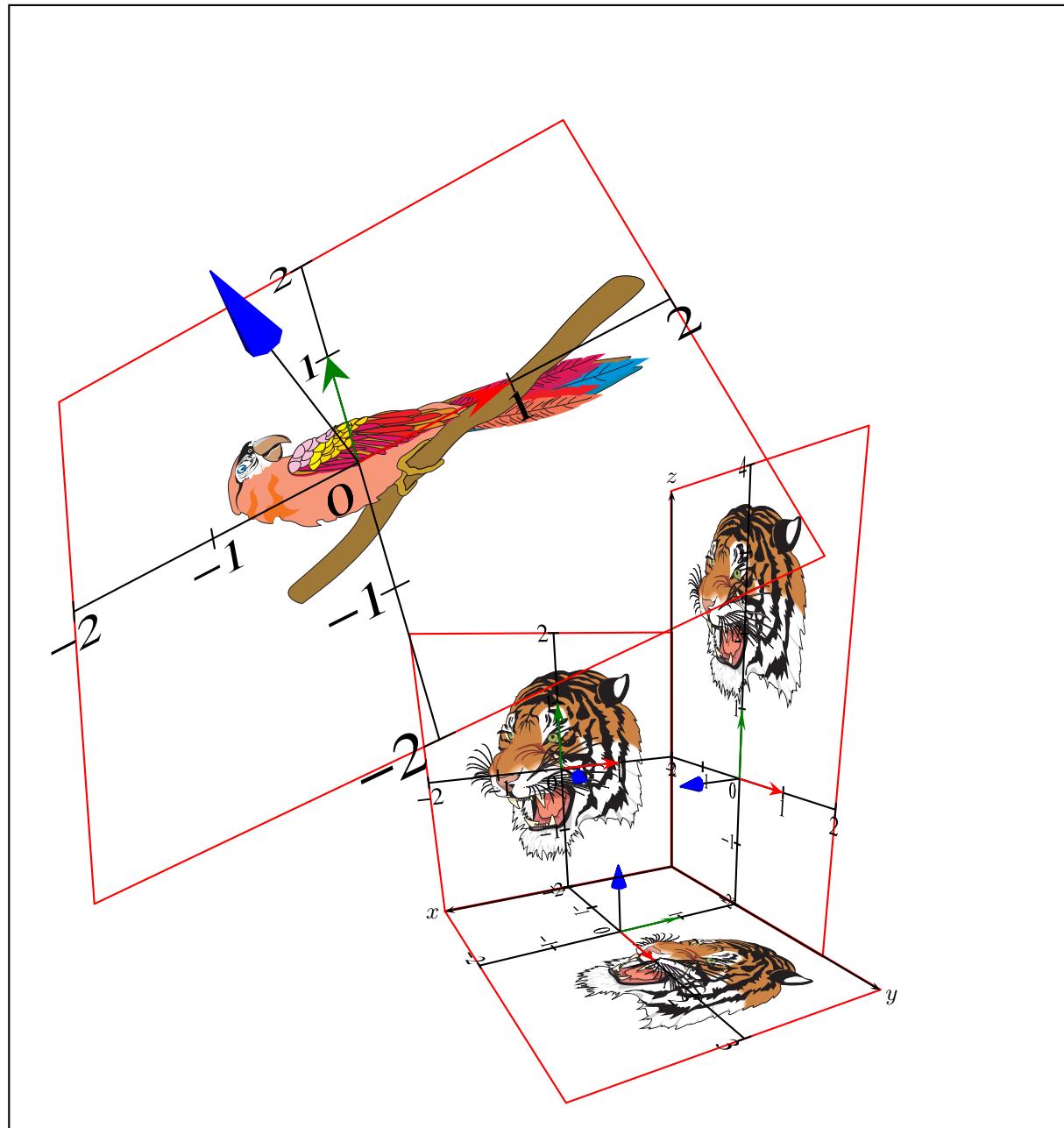
If you want to place the image on the front of an object, it will follow the following procedure:

- determine the number of faces of the object, see the documentation of ‘pst-solides3d ’;
- give to the normal of the face in question and origin at the center of that face. We can always shift the image with (x, y).

```
\begin{pspicture}(-5,-5)(5,5)
\psset{solidmemory}
\psSolid[object=cube,a=8,action=draw,name=OBJECT,linecolor=red]%
\psImage[file=tiger.eps,normal=OBJECT 0 solidnormaleface,
          origine=OBJECT 0 solidcentreface,unitPicture=75]
\psImage[file=tiger.eps,normal=OBJECT 1 solidnormaleface,
          origine=OBJECT 1 solidcentreface,unitPicture=75]
\psImage[file=tiger.eps,normal=OBJECT 4 solidnormaleface,
          origine=OBJECT 4 solidcentreface,unitPicture=75]
\psImage[file=tiger.eps,normal=OBJECT 3 solidnormaleface,
          origine=OBJECT 3 solidcentreface,unitPicture=75]
\psImage[file=tiger.eps,normal=OBJECT 2 solidnormaleface,
          origine=OBJECT 2 solidcentreface,unitPicture=75]
\end{pspicture}
```

¹<http://melusine.eu.org/syracuse/G/pst-anamorphosis/doc/>

If the selected plan is not visible to the set position, it may, if desired, force the display of the image with the **visibility**.



11. Projections

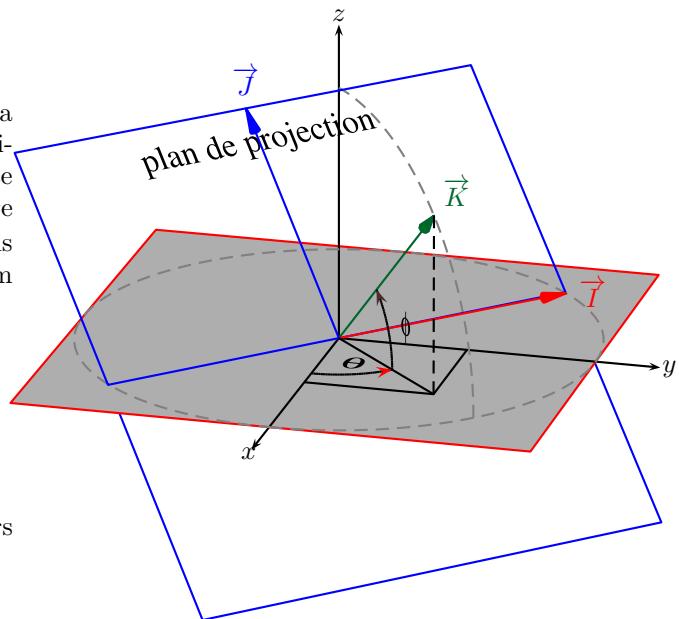


11.14. A bit of theory

The image is projected into a plane defined by a normal \vec{K} and origin $O'(x_O, y_O, z_O)$. The coordinates of points in each image are given in reference to a benchmark plan (O, \vec{I}, \vec{J}) whose vectors are determined from \vec{K} as follows: This vector \vec{K} is defined by θ and φ , we calculate these values from the coordinates. With $(O, \vec{i}, \vec{j}, \vec{k})$

$$\vec{K} = \begin{pmatrix} \cos \varphi \cos \theta \\ \cos \varphi \sin \theta \\ \sin \varphi \end{pmatrix}$$

You must then choose the other two basis vectors $(\vec{I}, \vec{J}, \vec{K})$. I choose to keep \vec{I} at the plane Oxy



11. Projections

CHAPTER 12

Possible extensions

12.1. Creating your own object

It is possible to create your own object in a separate file and import it into the list of objects recognized by **pst-solides3d**. Create a text file with the extension of **.pro** (myObj.pro) and enter the PostScript commands to define your **pst-solides3d** object.

Reference your **.pro** file in the preamble with

```
\pstheader{myObj.pro}
```

Following this line, add this new object to the list of objects recognized by **pst-solides3d** with

```
\addtosolideslistobject{myObj}
```

For some examples of this technique, see the following web pages:

<http://melusine.eu.org/syracuse/mluque/solides3d2007/cristaux/>
<http://melusine.eu.org/syracuse/mluque/solides3d2007/rhombicuboctaedre/>

12.2. Creating a **.u3d** file

You can manipulate 3D objects created with **pst-solides3d**; the following three steps are necessary:

1. Save your designed 3D object in the **.off** or **.obj** format—see the chapter “*Usage of external files*”.
2. Then use, for example, *Meshlab*—an open source software—(<http://meshlab.sourceforge.net/>) to convert these files into the **.u3d** format.
3. The L^AT_EX package **movie15** of Alexander GRAHN embeds files in the **.u3d** format into a PDF document, the document can then be viewed using Adobe® Reader® 7 or later.

You will find some examples on the following web pages:

<http://melusine.eu.org/syracuse/mluque/solides3d2007/pdf3d/>

12. Possible extensions

<http://melusine.eu.org/syracuse/mluque/solides3d2007/zeolithes/>

APPENDIX A

Appendix

A.1. The parameters of `pst-solides3d`

Parameter	Default	Description
<code>object</code>		predefined objects for use with <code>\psSolid</code> and <code>\psProjection</code> : <code>object=myName</code> where <code>myName</code> is the type of object
<code>viewpoint</code>	10 10 10	the coordinates of the point of view
<code>a</code>	2	the value of <code>a</code> has several interpretations: the edge length of a cube, the radius of the circumscribed sphere of regular polyhedrons, the length of one of the edges of a parallelepiped
<code>r</code>	2	the radius of a cylinder or sphere
<code>h</code>	6	the height of a cylinder, cone, truncated cone, or prism
<code>r0</code>	1.5	the inner radius of a torus
<code>r1</code>	4	the mean radius of a torus
<code>phi</code>	0	the lower latitude of a spherical zone
<code>theta</code>	90	the upper latitude of a spherical zone
<code>a,b and c</code>	4	the lengths of three incident edges of a parallelepiped
<code>base</code>	-1 -1 1 -1 0 1	the coordinates of vertices in the <i>xy</i> -plane for specified shapes
<code>axe</code>	0 0 1	the direction of the axis of inclination of a prism
<code>action</code>	<code>draw**</code>	uses the painting algorithm to draw the solid without hidden edges and with coloured faces
<code>lightsrc</code>	20 30 50	the Cartesian coordinates of the light source
<code>lightintensity</code>	2	the intensity of the light source
<code>ngrid</code>	<code>n1 n2</code>	sets the grid for a chosen solid
<code>mode</code>	0	sets a predefined grid: values are 0 to 4. <code>mode=0</code> is a large grid and <code>mode=4</code> is a fine grid
<code>grid</code>	<code>true</code>	if <code>grid</code> is used then gridlines are suppressed
<code>biface</code>	<code>true</code>	draw the interior face; if you only want the exterior shown write <code>biface=false</code>
<code>algebraic</code>	<code>false</code>	<code>algebraic=true</code> (also written as <code>[algebraic]</code>) allows you to give the equation of a surface in algebraic form (otherwise RPN is enabled); the package <code>pstricks-add</code> must be loaded in the preamble
<code>fillcolor</code>	<code>white</code>	specifies a colour for the outer faces of a solid

Continued on next page

A. Appendix

Parameter	Default	Description
<code>incolor</code>	green	specifies a colour for the inner faces of a solid
<code>hue</code>		the colour gradient used for the outer faces of a solid
<code>inhue</code>		the colour gradient used for internal faces
<code>inouthue</code>		the colour gradient used for both internal and external faces as a single continuation
<code>fcol</code>		permits you to specify, in order of face number 0 to $n - 1$ (for n faces) the colour of the appropriate face: <code>fcol=0 (Apricot) 1 (Aquamarine) etc.</code>
<code>rm</code>		removes visible faces: <code>rm=1 2 8</code> removes faces 1, 2 and 8
<code>show</code>		determines which vertices are shown as points: <code>show=0 1 2 3</code> shows the vertices 0, 1, 2 and 3, <code>show=all</code> shows all the vertices
<code>num</code>		numbers the vertices; for example <code>num=0 1 2 3</code> numbers the vertices 0,1,2 and 3, and <code>num=all</code> numbers all the vertices
<code>name</code>		the name given to a solid
<code>solidname</code>		the name of the active solid
<code>RotX</code>	0	the angle of rotation of the solid around Ox (in degrees)
<code>RotY</code>	0	the angle of rotation of the solid around Oy (in degrees)
<code>RotZ</code>	0	the angle of rotation of the solid around Oz (in degrees)
<code>hollow</code>	false	draws the inside of hollow solids: cylinder, cone, truncated cone and prism
<code>decal</code>	-2	reassign the index numbers of the vertices within a <code>base</code>
<code>axesboxed</code>	false	this option for surfaces allows semi-automatic drawing of the 3D coordinate axes, since the limits of z must be set by hand; enabled with <code>axesboxed</code>
<code>Zmin</code>	-4	the minimum value of z
<code>Zmax</code>	4	the maximum value of z
<code>QZ</code>	0	shifts the coordinate axes vertically by the chosen value
<code>spotX</code>	dr	the position of the tick labels on the x -axis
<code>spotY</code>	dl	the position of the tick labels on the y -axis
<code>spotZ</code>	1	the position of the tick labels on the z -axis
<code>resolution</code>	36	the number of points used to draw a curve
<code>range</code>	-4 4	the limits for function input
<code>function</code>	f	the name given to a function
<code>path</code>	<code>newpath 0 0 moveto</code>	the projected path
<code>text</code>		the projected text
<code>visibility</code>	false	if false the text applied to a hidden face is not rendered
<code>chanfreincoeff</code>	0.2	the chamfering coefficient
<code>trunccoeff</code>	0.25	the truncation coefficient
<code>dualregcoeff</code>	1	the dual solid coefficient
<code>affinagecoeff</code>	0.8	the hollowing coefficient
<code>affinage</code>		determines which faces are hollowed out: <code>affinage=0 1 2 3</code> recesses faces 0, 1, 2 and 3, <code>affinage=all</code> recesses all faces
<code>affinagerm</code>		keep the central part of hollowed out faces
<code>intersectiontype</code>	-1	the type of intersection between a plane and a solid; a positive value draws the intersection
<code>plansection</code>		list of equations of intersecting planes, when used only for their intersections
<code>plansepare</code>		the equation of the separating plane for a solid
<code>intersectionlinewidth</code>	1	the thickness of an intersection in pt; if there are several intersections of different thicknesses then list them like so: <code>intersectionlinewidth=1 1.5 1.8 etc.</code>

Continued on next page

Parameter	Default	Description
intersectioncolor	(rouge)	the colour used for intersections; if several intersections in different colours are required, list them as follows: <code>intersectioncolor=(rouge) (vert) etc.</code>
intersectionplan	[0 0 1 0]	the equation of the intersecting plane
definition		defines a point, a vector, a plane, a spherical arc, etc.
args		arguments associated with definition
section	\Section	the coordinates of the vertices of a cross-section of a solid ring
planmarks	false	scales the axes of the plane
plangrid	false	draws the coordinate axes of the plane
showbase	false	draws the unit vectors of the plane
showBase	false	draws the unit vectors of the plane and the normal vector to the plane
deactivatecolor	false	disables the colour management of PSTricks
transform		a formula, applied to the vertices of a solid, to transform it
axisnames	{x,y,z}	the labels of the axes in 3D
axisemph		the style of the axes labels in 3D
showOrigin	true	draws the axes from the origin, or not if set to false
mathLabel	true	draws the axes labels in math mode, or not if set to false
file		the name of the data file having .dat extension written with <code>action=writesolid</code> or read with <code>object=datfile</code>
load		the name of the object to be loaded
fcolor		the colour of the refined parts of the faces of an object
sommets		the list of vertices of a solid for use with <code>object=new</code>
faces		the list of faces of a solid for use with <code>object=new</code>
stepX	1	a positive integer giving the interval between ticks on the <i>x</i> -axis of \gridIID
stepY	1	a positive integer giving the interval between ticks on the <i>y</i> -axis of \gridIID
stepZ	1	a positive integer giving the interval between ticks on the <i>z</i> -axis of \gridIID
ticklength	0.2	the length of tickmarks for \gridIID

End of table

A. Appendix

A.2. Alphabetical list of keywords

Glossary of symbols	
Symbol	Use/meaning
object, sommets, ...	keywords
A, B, C, I, P	names of points
$x y$	coordinates of a point in a plane
$x y z$	coordinates of a 3d point
$r \theta \phi$	spherical coordinates of a 3d point
L, M	names of lines
C, r	circle, centre name C , radius r
$a b c$	components of a normal
$[a b c d]$	the plane $ax + by + cz + d = 0$
a, b	intercepts of lines
u, v	names of vectors
α	angle/angle of rotation
k	scaling factor
S	name of a solid
i	index number of a vertex/face
w	linewidth
num	integer
$value$	real number
$length$	positive real number
$string$	text string
$a b c ...$	alternatives

Name	Command/Object	Value	Default
a	<code>\psSolid</code> object=cube tetrahedron octahedron dodecahedron icosahedron	$length$	2
a, b and c	<code>\psSolid</code> object=parallelepiped	$length$	4
action	<code>\psSolid</code>	none draw draw* draw** writeobj writeoff writesolid	draw**
affinage	<code>\psSolid</code>	all $i_0 i_1 \dots i_n$	
affinagecoef	<code>\psSolid</code>	$value$	0.8
affinagerm	<code>\psSolid</code>	$boolean$	true
algebraic	<code>\psFunction, \psSurface</code>	$boolean$	false
args	<code>\psSolid</code> object=plan definition =equation =normalpoint =solidface object=point definition =addv3d =barycentre3d	$\{[a b c d]\} \{[a b c d] \alpha\}$ $\{x_0 y_0 z_0 [a b c]\}$ $\{x_0 y_0 z_0 [a b c \alpha]\}$ $\{x_0 y_0 z_0 [u_x u_y u_z a b c]\}$ $\{x_0 y_0 z_0 [u_x u_y u_z a b c \alpha]\}$ $S i$ $x y z P$ $x_1 y_1 z_1 x_2 y_2 z_2 u v$ $\{[A i_A B i_B]\}$	

Continued on next page

Name	Command/Object	Value	Default
args	<ul style="list-style-type: none"> =hompoint3d =isobarycentre3d =milieu3d =mulv3d =normalize3d =orthoprojplane3d =rotate0point3d =scale0point3d =solidcentreface =solidgetssommet =subv3d =sympoint3d =translatepoint3d =vectprod3d object=vecteur object=vecteur3d \psProjection object=cercle definition <ul style="list-style-type: none"> =ABcercle =diamcercle object=droite definition <ul style="list-style-type: none"> =axesymdroite =bissectrice =horizontale =mediatrice =paral =perp =rotatedroite =translatedroite =verticale object=line object=point definition <ul style="list-style-type: none"> =axesympoint =cpoint =hompoint =interdroite =interdroitecercle =milieu =orthoproj =parallelpoint =projx =projy =rotatepoint 	<ul style="list-style-type: none"> $P A k$ $\{[A_0 A_1 \dots A_n]\}$ $A B$ $x y z k \mid u k$ $x y z \mid u$ $P A v$ $P \alpha_x \alpha_y \alpha_z$ $x y z k_x k_y k_z \mid name k_x k_y k_z$ $S i$ $S i$ $x_1 y_1 z_1 x_2 y_2 z_2 \mid u v$ $P A$ $P v$ $x_1 y_1 z_1 x_2 y_2 z_2 \mid u v$ $x y z \mid$ $x_1 y_1 z_1 x_2 y_2 z_2 \text{ addv3d} \mid$ $x_1 y_1 z_1 x_2 y_2 z_2 \text{ subv3d} \mid$ $x y z k \text{ mulv3d} \mid$ $x y z \text{ normalize3d} \mid$ $x_1 y_1 z_1 x_2 y_2 z_2 \text{ vectprod3d}$ $x_A y_A z_A x_B y_B z_B \mid A B$ $x y r \mid C r$ $A B C$ $A B$ $x_1 y_1 x_2 y_2 \mid A B$ $L M$ $A B C$ b $A B$ $L A$ $L A$ $L A \alpha$ $L u$ a $A_0 A_1 \dots A_n$ $P L$ $\alpha C r$ $P A k$ $L M$ $L C r$ $A B$ $P L$ $A B C$ P P $P I \alpha$ 	

Continued on next page

A. Appendix

Name	Command/Object	Value	Default
	=sympoint =translatepoint =xdpoint =ydpoint object=polygone definition =axesympol =hompol =rotatepol =sympol =translatepol object=rightangle object=vecteur definition =addv =mulv =normalize =orthovecteur =subv =vecteur	$P I$ $P u$ $x L$ $y L$ $A_0 A_1 \dots A_n$ $pol L$ $pol I \alpha$ $pol I \alpha$ $pol I$ $pol u$ $A B C$ $A B$ $u k$ u u $u v$ $A B$	
axe	\psSolid object=cylindre prisme ruban	$x y z$	0 0 1
axesboxed	\psSolid	boolean	false
axisemph	\axesIID \gridIID	{text style}	
axisnames	\axesIID	{a,b,c}	{x,y,z}
base	\psSolid object=face prisme ruban object=fusion object=grille	$x_1 y_1 x_2 y_2 x_3 y_3 \dots x_n y_n$ $S_1 S_2$ $x_{\min} x_{\max} y_{\min} y_{\max}$	-1 -1 1 -1 0 1
biface	\psSolid object=face	boolean	true
chanfrein	\psSolid	boolean	false
chanfrein-coeff	\psSolid	value	0.2
deactiv-atecolor	\psSolid	boolean	false
decal	\psSolid	num	-2
definition	\psSolid object=plan object=point object=vecteur	equation normalpoint solidface addv3d barycentre3d hompoint3d isobarycentre3d milieu3d orthoprojplane3d rotate0point3d scale0point3d solidcentreface solidgetsommets vecteur3d addv3d subv3d mulv3d normalize3d vectprod3d	

Continued on next page

Name	Command/Object	Value	Default
definition	\psProjection object=cercle object=droite object=point object=polygone object=vecteur	ABcercle diamcercle axesymdroite bissectrice horizontale mediatrice paral perp rotatedroite translatedroite verticale axesympoint cpoint hompoint interdroite interdroite cercle milieu orthoproj parellelopoin projx projy rotatepoint sympoint translatepoint xdpoint ydpoint axesympol hompol rotatepol sympol translatepol addv normalize mulv orthovecteur subv vecteur	
dualreg	\psSolid object=geode	boolean	false
faces	\psSolid object=new	{[i ₁ i ₂ ... i _n][i' ₁ i' ₂ ... i' _m] ... }	
fcol	\psSolid	i ₀ (color ₀) i ₁ (color ₁) ...	
fcolor	\psSolid affinagerm	color	
file	\psSolid action=writesolid object=datfile objfile offfile	filename filename	
fillcolor	\psSolid, \psSurface	color	white
function	\psSolid, \defFunction object=cone courbe courbeR2 cylindre surfaceparametree	name	
grid	\psSolid	boolean	true
h	\psSolid object=cone cylindre prisme tronccone	length	6
hollow	\psSolid object=cone cylindre prisme tronccone	boolean	false
hue, inhue, inouthue	\psSolid, \psSurface	h ₀ h ₁ h ₀ h ₁ s b h ₀ s ₀ b ₀ h ₁ s ₁ b ₁ (hsb) r ₀ g ₀ b ₀ r ₁ g ₁ b ₁ c ₀ m ₀ y ₀ k ₀ c ₁ m ₁ y ₁ k ₁ (color ₁) (color ₂)	
incolor	\psSolid, \psSurface	color	green
intersec- tioncolor	\psSolid	(color ₁) ... (color _n)	(rouge)
intersec- tionlinewidth	\psSolid	w ₁ ... w _n	1

Continued on next page

A. Appendix

Name	Command/Object	Value	Default
<code>intersectionplan</code>	<code>\psSolid, \psSurface</code>	$name \mid \{eq_1 \dots eq_n\}$ where $eq_i=[a_i \ b_i \ c_i \ d_i]$	
<code>labelsep</code>	<code>\axesIID</code>	$length[unit]$	
<code>light-intensity</code>	<code>\psSolid, \psSurface</code>	$value$	2
<code>lightsrc</code>	<code>\psSolid, \psSurface</code>	$x \ y \ z$	20 30 50
<code>load</code>	<code>\psSolid</code> <code>object=load</code>	$name$	
<code>mathLabel</code>	<code>\axesIID</code>	$boolean$	true
<code>mode</code>	<code>\psSolid</code>	0 1 2 3 4	0
<code>name</code>	<code>\psSolid, \psProjection</code>	$name$	
<code>ngrid</code>	<code>\psSolid</code> <code>object=cube prisme prismecreux</code> <code>object=cone conecreux cylindre cylindrecréux tore tronccone troncconecreux</code> <code>object=grille surface surface* surfaceparametree</code>	n_1 $n_1 \ n_2$ $n_1 \mid n_1 \ n_2$	
<code>num</code>	<code>\psSolid</code>	$all \mid i_0 \ i_1 \dots i_n$	
<code>object</code>	<code>\psSolid</code>	<code>new anneau calottesphere cone conecreux cube cylindre cylindrecréux datfile dodecahedron face fusion geode grille icosahedron load octahedron objfile parallelepiped plan prisme ruban sphere surfaceparametree tetrahedron tore tronccone troncconecreux</code>	
<code>object</code>	<code>\psProjection</code>	<code>cercle courbe courbeR2 droite line point polygone rightangle texte vecteur</code>	
<code>opacity</code>	<code>\psSolid</code>	$value$	1
<code>origine</code>	<code>\psSolid</code> <code>object=plan</code>	$x_0 \ y_0 \ z_0$	0 0 0
<code>path</code>	<code>\psProjection</code>	<code>pscode</code>	<code>newpath 0 0 moveto</code>
<code>phi</code>	<code>\psSolid, \psProjection</code>	α	0
<code>plangrid</code>	<code>\psSolid</code> <code>object=plan</code>	$boolean$	false
<code>planmarks</code>	<code>\psSolid</code> <code>object=plan</code>	$boolean$	false
<code>plansection</code>	<code>\psSolid</code>	$\{plan_1 \dots plan_n\}$ where $plan_i=[a_i \ b_i \ c_i \ d_i]$	
<code>plansepare</code>	<code>\psSolid</code>	$\{[a \ b \ c \ d]\}$	

Continued on next page

Name	Command/Object	Value	Default
<code>pos</code>	<code>\psProjection</code> <code>object=point</code>	<code>ul cl bl dl ub cb bb db uc cc bc dc ur cr br dr</code>	<code>cc</code>
<code>QZ</code>	<code>\psSolid, \psSurface</code>	<code>value</code>	<code>0</code>
<code>RotX, RotY, RotZ</code>	<code>\psSolid</code>	α	<code>0</code>
<code>r</code>	<code>\psSolid</code> <code>object=anneau courbe</code>	<code>length</code>	<code>2</code>
<code>R</code>	<code>\psSolid</code> <code>object=anneau</code>	<code>length</code>	<code>4</code>
<code>r0</code>	<code>\psSolid</code> <code>object=tore troncone troncconecreux</code>	<code>length</code>	<code>1.5</code>
<code>r1</code>	<code>\psSolid</code> <code>object=tore troncone troncconecreux</code>	<code>length</code>	<code>4</code>
<code>range</code>	<code>\psSolid</code> <code>object=cercle courbe courbeR2</code> <code>object=surfaceparametree</code>	$t_{\min} t_{\max}$ $u_{\min} u_{\max} v_{\min} v_{\max}$	<code>-5 5</code>
<code>resolution</code>	<code>\psSolid</code> <code>object=courbe courbeR2 raban</code>	n	<code>36</code>
<code>rm</code>	<code>\psSolid</code>	$i_0 i_1 \dots i_n$	
<code>section</code>	<code>\psSolid</code> <code>object=anneau</code>	<code>macro{pscode}</code>	<code>\Section</code>
<code>show</code>	<code>\psSolid</code>	<code>all $i_0 i_1 \dots i_n$</code>	
<code>showBase</code>	<code>\psSolid</code> <code>object=plan</code>	<code>boolean</code>	<code>false</code>
<code>showbase</code>	<code>\psSolid</code> <code>object=plan</code>	<code>boolean</code>	<code>false</code>
<code>showOrigin</code>	<code>\axesIID</code>	<code>boolean</code>	<code>true</code>
<code>sommets</code>	<code>\psSolid</code> <code>object=new</code>	$x_1 y_1 z_1 x_2 y_2 z_2 \dots x_n y_n z_n$	
<code>spotX,spotY, spotZ</code>	<code>\psSurface, \gridIID</code>	<code>u ul l dl d dr r ur</code>	
<code>stepX,stepY, stepZ</code>	<code>\gridIID</code>	n	<code>1</code>
<code>text</code>	<code>\psProjection</code> <code>object=point</code>	<code>string</code>	
<code>theta</code>	<code>\psSolid</code> <code>object=calottesphere</code>	α	<code>90</code>
<code>ticklength</code>	<code>\gridIID</code>	<code>length</code>	<code>0.2</code>
<code>transform</code>	<code>\psSolid, \defFunction</code>	<code>{pscode} function</code>	
<code>trunc</code>	<code>\psSolid</code>	<code>all $i_0 i_1 \dots i_n$</code>	
<code>trunccoeff</code>	<code>\psSolid</code>	<code>value</code>	<code>0.2</code>

Continued on next page

A. Appendix

Name	Command/Object	Value	Default
<code>viewpoint</code>	<code>\psset</code>	$x \ y \ z \ \ r \ \theta \ \phi \ \text{rtp2xyz}$	10 10 10
<code>visibility</code>	<code>\psSolid, \psProjection</code>	<code>boolean</code>	<code>true</code>
<code>Zmin</code>	<code>\psSurface, \gridIID</code>	<code>value</code>	-4
<code>Zmax</code>	<code>\psSurface, \gridIID</code>	<code>value</code>	4

End of table

A.3. Acknowledgments

Spontaneous and diligent proofreading assistance from various members of the PSTricks list made it possible to produce this English version of the `pst-solides3d` documentation. We hope that this will help and encourage more of you to set about depicting your own 3D solids.

So, many thanks from the “équipe solide” go to:

Gerry COOMBES, Zbiginiew NITECKI, D. P. STORY and Herbert VOSS.

Additional thanks go to Gerry COOMBES, who generated a keyword glossary for the `pst-solides3d` package and who proofed the terminology for consistency.

A.4. The poems

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 from *Les idées de autres* by 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 of *Molly Bloom* from *Ulysses* by James Joyce

A. Appendix

Index

Symbols

.dat, 60, 75, 93
.obj, 61, 93
.off, 62, 93

A

AB cercle, 160
action, 31, 84, 120, 124, 126
addv, 158
addv3d, 65, 66
affinage, 51
affinagecoeff, 51
affinagerm, 51
algebraic, 113, 165
all, 34, 50, 51, 170, 172
anneau, 98
args, 64–72, 154–162
axe, 104
axes, 18
axesboxed, 113, 114
axesymdroite, 159
axesympoint, 155
axesympol, 161
axisemph, 18
axisnames, 18

B

barycentre3d, 64
base, 67, 69, 94, 95, 135
biface, 98
bissectrice, 159
brilliance, 41, 42

C

cercle, 159
chamfer, 52
Chamfering, 52
chanfrein, 52
chanfreincoeff, 52
chloride ion, 138
circle, 159
CMYK, 43
color, 44
colour, 44
colour scheme, 41–43
Colouring, 44

colours, 37, 38, 44

components, 65
Cone, 22
cone, 111
cone, 46
conecreux, 46
conic area, 107
contour lines, 123
coordinates, 13
corrugated surface, 96
courbe, 11, 79, 81, 164
courbeR2, 165
cpoint, 155
crosshatch, 150
Cube, 21
cube, 46
Cuboid, 25
Curve, 26
curve, 165
Cylinder, 21
cylinder, 32
cylindre, 46, 104
cylindrecrues, 46
cylindric area, 104

D

datfile, 60
deactivatecolor, 44, 119
decal, 90
decal, 90
Decran, 14
Decran, 14, 15, 75
definition, 64–68, 70, 72, 155–161
diam cercle, 160
dimensions, 152
Dodecahedron, 24
doubleline, 150
draw, 31, 120
draw*, 31
draw**, 31
drawing, 31

E

equation, 68

F

Index

Face, 25
face, 94, 98
faces, 34, 44
faces, 91, 93
fcol, 38, 44, 46, 47
fcolor, 51
file, 60–62
fillcolor, 37, 92
fillstyle, 150
folding screen, 96
fontsize, 34, 166
function, 77, 79, 164
function, 79, 81, 104, 110, 164
functions, 78
fuse solids, 136
fusion, 137
fusion, 135

G

geodes, 74
grating, 46
Grid, 25
grid, 47, 113, 115
gridlines, 46

H

h, 98, 104
hairline curve, 82
helix, 111
hlines, 150
hollow, 35
hollow, 32, 92, 122, 125
Hollow Cone, 22
Hollow Cylinder, 21
hollow cylinder, 32
hollow prism, 33
hollow solid, 122
hollow spherical zone, 34
Hollowing out, 51
hompoint, 155
hompoint3d, 64
hompol, 161
horizontale, 159
HSB, 41, 42
hue, 11, 37, 41–43, 45
hyperbolic paraboloid, 116
hyperboloid, 92
hypocycloid, 86

I

Icosahedron, 24
incolor, 37, 44, 92
inhue, 37
inouthue, 37
interdroite, 156
interdroitecercle, 156
intersecting planes, 58

intersection, 120
intersectioncolor, 58, 120
intersectionlinewidth, 58, 120
intersectionplan, 58
intersectiontype, 58, 120
isobarycentre3d, 64

J

jps code, 77

K

Keyword
– ABcercle, 160
– action, 31, 84, 120, 124, 126
– affinage, 51
– affinagecoeff, 51
– affinagerm, 51
– algebraic, 113, 165
– args, 64–72, 154–162
– axe, 104
– axesboxed, 113, 114
– axesymdroite, 159
– axesympol, 161
– axisemph, 18
– axisnames, 18
– base, 67, 69, 94, 95, 135
– biface, 98
– bissectrice, 159
– cercle, 159
– chanfrein, 52
– chanfreincoeff, 52
– color, 44
– courbe, 11, 164
– courbeR2, 165
– datfile, 60
– deactivatecolor, 44, 119
– decal, 90
– Decran, 14, 15, 75
– definition, 64–68, 70, 72, 155–161
– diamcercle, 160
– doubleline, 150
– draw*, 31
– draw**, 31
– face, 94, 98
– faces, 91, 93
– fcol, 38, 44, 46, 47
– fcicolor, 51
– file, 60–62
– fillcolor, 37, 92
– fillstyle, 150
– fontsize, 34, 166
– function, 79, 81, 104, 110, 164
– fusion, 135
– grid, 47, 113, 115
– h, 98, 104
– hollow, 32, 92, 122, 125

- hompol, 161
 - hue, 11, 41–43, 45
 - incolor, 37, 44, 92
 - intersectioncolor, 58, 120
 - intersectionlinewidth, 58, 120
 - intersectionplan, 58
 - intersectiontype, 58, 120
 - labelsep, 18
 - lightintensity, 16, 17
 - lightsrc, 16
 - line, 162
 - lineararc, 150
 - linecolor, 119, 150
 - load, 119
 - mathLabel, 18
 - mode, 49
 - name, 67, 119, 121, 154, 157, 158, 160
 - new, 91, 92
 - ngrid, 11, 46, 56, 75, 81, 83, 98, 113
 - num, 36
 - numfaces, 34, 170, 172
 - object, 67, 80, 81, 98, 104, 109, 119, 123, 153, 181
 - objfile, 61
 - offfile, 62
 - opacity, 46
 - origine, 69, 107
 - perp, 159
 - plan, 11, 58, 153
 - plangrid, 68
 - planmarks, 68
 - point, 64, 154
 - polygone, 160
 - pos, 154, 166, 167
 - QZ, 114
 - R, 98
 - r, 81, 98
 - range, 81, 104, 109, 160, 165
 - resolution, 11, 77
 - rightangle, 163
 - rm, 35, 92
 - rotatedroite, 159
 - rotatepol, 161
 - run, 93
 - section, 100
 - show, 36
 - showBase, 68
 - showbase, 68
 - showOrigin, 18
 - solidmemory, 119, 120
 - sommets, 91, 93
 - spotX, 114
 - spotY, 114
 - spotZ, 114
 - sympol, 161
 - text, 154, 166
 - texte, 166
 - tracelignedeniveau, 11
 - transform, 53
 - translatedroite, 159
 - translatepol, 161
 - trunc, 50
 - trunccoeff, 50
 - vecteur, 65, 157
 - viewpoint, 75, 149
 - visibility, 34, 153, 172
 - writeobj, 61, 62
 - writesolid, 60
 - Zmax, 113
 - Zmin, 113
- L**
- label, 18
 - labels, 18
 - labelsep, 18
 - light, 16
 - light intensity, 16
 - light source, 16
 - lightintensity, 16, 17
 - lightsrc, 16
 - Line, 20
 - line, 150, 158, 162
 - line, 162
 - lineararc, 150
 - linecolor, 119, 150
 - Lissajous, 165
 - load, 119
- M**
- Macro
 - \psProjection, 11, 119, 153, 154, 157, 165–167
 - \psSolid, 11, 19, 27, 31, 35, 43, 44, 46, 58, 60–62, 67, 73, 75, 80, 92, 98, 100, 109, 119, 135, 153
 - \psSurface, 11, 119
 - mathLabel, 18
 - mediatrice, 159
 - methane molecule, 144
 - milieu, 155
 - milieu3d, 65
 - mode, 49
 - modes, 49
 - Multiplication, 66
 - mulv, 158
 - mulv3d, 66
- N**
- name, 67, 119, 121, 154, 157, 158, 160
 - New, 26
 - new, 91, 92, 123
 - ngrid, 11, 46, 56, 75, 81, 83, 98, 113
 - none, 31, 126
 - normalize, 157
 - normalize3d, 66

Index

Normalized vector, 66

normalpoint, 70

num, 36

numfaces, 34, 170, 172

O

object, 67, 80, 81, 98, 104, 109, 119, 123, 153, 181
objfile, 61
oblique prisms, 88
Octahedron, 24
octahedron, 125
offfile, 62
opacity, 46
opacity, 46
origine, 69, 107
orthoproj, 155
orthoprojplane3d, 65
orthovecteur, 157

P

paraboloid, 115
paral, 159
parallelepiped, 132
parallelpoint, 155
Parameterised curves, 165
parameterised functions, 164
perp, 159
plan, 11, 58, 67, 153
Plane, 20
plane, 16, 67, 153
plangrid, 68
planmarks, 68
Point, 20
point, 64, 154
point, 64, 154
polygon, 150, 160
polygone, 160
pos, 154, 166, 167
Prism, 24
prism, 33, 87
prisme, 46
prismecreux, 46
project, 166
projecting, 170, 171
projection, 153, 167–169
projection plane, 153
projection screen, 14
projx, 155
projy, 155
\psProjection, 11, 119, 153, 154, 157, 165–167
\psSolid, 11, 19, 27, 31, 35, 43, 44, 46, 58, 60–62,
 67, 73, 75, 80, 92, 98, 100, 109, 119, 135,
 153
\psSurface, 11, 119
pyramid, 123, 125

Q

QZ, 114

R

R, 98
r, 81, 98
range, 81, 104, 109, 160, 165
resolution, 83
resolution, 11, 77
RGB, 43
right angle, 163
right prism, 88
rightangle, 163
rings, 98, 144
rm, 35, 92
roof gutter, 90
rotatedroite, 159
rotate0point3d, 65
rotatepoint, 155
rotatepol, 161
rotation, 28
run, 93

S

saddle, 114
saturation, 41, 42
scale0point3d, 65
sea shell, 110
section, 100
show, 36
showBase, 68
showbase, 68
showOrigin, 18
sinusoidal wave, 116
Slice, 123
slice, 125
slicing, 121
solid, 150
Solid strip, 95
solidcentreface, 64
solidface, 72
solidgetssommet, 64
solidmemory, 119, 120
sommets, 91, 93
Sphere, 23
sphere, 46
spherical coordinates, 14
spherical zone, 34
spotX, 114
spotY, 114
spotZ, 114
Strip, 25
subv, 158
subv3d, 66
Surface, 26
surfaceparametree, 109
surfaces, 109

sympoint, 155
sympoint3d, 64
sympol, 161

T

Tetrahedron, 24
text, 166
text, 154, 166
texte, 166
thiosulphate ion, 146
tore, 46
torsion, 56
Torus, 23
tracelignedeniveau, 11
transform, 53
transformation, 53
Transformations, 151
translatedroite, 159
translatepoint, 155
translatepoint3d, 65
translatepol, 161
Translation, 27
transparency, 46, 75
troncone, 46
tronconecreux, 46
trunc, 50
truncate, 50
Truncated Cone, 22
truncation, 50
trunccoef, 50
tube, 80

V

Value
– addv, 158
– addv3d, 65, 66
– all, 34, 50, 51, 170, 172
– anneau, 98
– axesympoint, 155
– barycentre3d, 64
– cone, 46
– conecreux, 46
– courbe, 79, 81
– cpoint, 155
– crosshatch, 150
– cube, 46
– cylindre, 46, 104
– cylindrecréux, 46
– draw, 31, 120
– draw*, 31
– equation, 68
– hlines, 150
– hompoint, 155
– hompoint3d, 64
– horizontale, 159
– hue, 37

– inhue, 37
– inouthue, 37
– interdroite, 156
– interdroite cercle, 156
– isobarycentre3d, 64
– mediatrice, 159
– milieu, 155
– milieu3d, 65
– mulv, 158
– mulv3d, 66
– new, 123
– none, 31, 126
– normalize, 157
– normalize3d, 66
– normalpoint, 70
– orthoprod, 155
– orthoprojplane3d, 65
– orthovecteur, 157
– paral, 159
– parallelopunkt, 155
– plan, 67
– prisme, 46
– prismecreux, 46
– projx, 155
– projy, 155
– rotate0point3d, 65
– rotatepoint, 155
– scale0point3d, 65
– solid, 150
– solidcentreface, 64
– solidface, 72
– solidgetssommet, 64
– sphere, 46
– subv, 158
– subv3d, 66
– surfaceparametree, 109
– sympoint, 155
– sympoint3d, 64
– tore, 46
– translatepoint, 155
– translatepoint3d, 65
– troncone, 46
– tronconecreux, 46
– vecteur, 157
– vecteur3d, 66
– vectprod3d, 66
– verticale, 159
– viewpoint, 16
– vlines, 150
– writesolid, 84, 124
– ydpoint, 156
vecteur, 65, 157
vecteur3d, 66
Vector, 20
vector, 65, 66, 157
Vector product, 66

Index

vectprod3d, 66
vehicle, 140
verticale, 159
vertices, 36, 91
view point, 13
viewpoint, 16, 75, 149
visibility, 153
visibility, 34, 153, 172
vlines, 150

W

wheel, 142
writeobj, 61, 62
writesolid, 60, 84, 124

Y

ydpoint, 156

Z

Zmax, 113
Zmin, 113

Bibliography

- [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, 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 L^AT_EX Graphics Companion*. Addison-Wesley Publishing Company, Reading, Mass., 2nd edition, 2007.
- [5] Alan Hoenig. *T_EX Unbound: L^AT_EX & T_EX Strategies, Fonts, Graphics, and More*. Oxford University Press, London, 1998.
- [6] Frank Mittelbach and Michel Goosens et al. *The L^AT_EX Companion*. Addison-Wesley Publishing Company, Boston, 2nd 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 T_EX und L^AT_EX*. DANTE – Lehmanns, Heidelberg/Hamburg, 4th edition, 2007.
- [10] Timothy Van Zandt and Denis Girou. Inside PSTRicks. *TUGboat*, 15:239–246, September 1994.