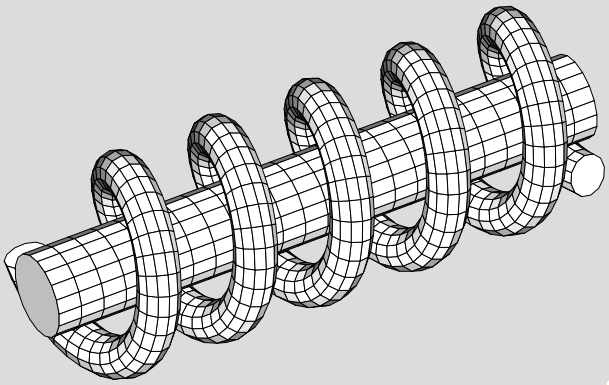
mp-solid

Paquet METAPOST pour la représentation de solides



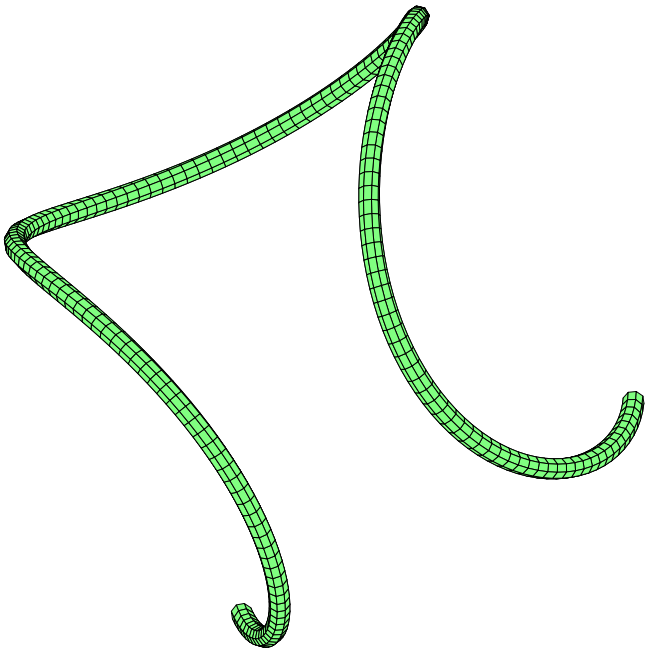
Auteur
Christophe POULAIN

Documentation
Christophe POULAIN
Alex AGUADO

Version 1.01 du 23 Novembre 2011
http://melusine.eu.org/syracuse/G/mp-solid/

Syracuse

mp-solid



Christophe Poulain

Version 1.01 – 23 novembre 2011

Résumé

Package permettant de construire des solides, des courbes de l'espace avec de nombreux apports dus à `pst-solides3d`.[1]

1. Remercions de suite Jean-Paul VIGNAULT et Manuel LUQUE pour ce magnifique (et le mot est *faible*) package `pstricks`.

Table des matières

Chapitre 1

Présentation

Un mathématicien n'a pas parfaitement compris ses propres travaux tant qu'il ne les a pas clarifiés au point de pouvoir aller dans la rue les expliquer à la première personne venue.

JOSEPH-LOUIS LAGRANGE

Avant toutes choses, je tiens particulièrement à remercier chaleureusement les auteurs de `pst-solides3d` pour leur aide et leur accord quant à l'utilisation de la terminologie et de nombreux exemples de leur documentation.

1.1 Constitution du package

– Fichier `mp-solid.mp`.
– Documentation et exemples : `doc.tgz(pdf)`.
Le package est disponible sur un dépôt `git` à l'adresse : http://melusine.eu.org/syracuse/G/mp-solid/

1.2 Introduction

Émerveillé, comme tout à chacun, par les prouesses de `pst-solides3d`, je me suis posé la question de savoir si cela était réalisable avec METAPOST. Malheureuse question, car pas mal de temps il m'a fallu [1] pour aboutir à cette première version... Temps de codage, d'apprentissage, de révision de mes connaissances mathématiques,... ont été très chronophage.
Mes sources d'inspiration ont été `pst-solides3d` (en parcourant le code), les auteurs (*merveilleux auteurs*) de ce package et le livre « Graphisme scientifique sur ordinateur » de Raymond DONY.

1.3 Le package

`mp-solid` contient les macros nécessaires pour les tracés demandés. Certaines macros ont été reprises de `geometriesyr16` [2], de `donymodule` [3]. Je dois remercier également Anthony PHAN et son

1. Yoda est avec moi !
2. Package dédié à la géométrie. Disponible à melusine.eu.org/syracuse/poulecl/geometriesyr16/.
3. Package complémentaire de `geometriesyr16` pour la gestion des figures géométriques de l'espace. Disponible en téléchargeant `geometriesyr16`.

package `m3D`[4] pour m'avoir permis de clarifier certaines conventions dans les représentations spatiales...
`mp-solid` *semble* être moins gourmand en ressources METAPOST que `mp-geo`[5] ; cependant, pour certains exemples, le temps de compilation est assez long[6].

1.4 Généralités

1.4.1 Le choix du point de vue

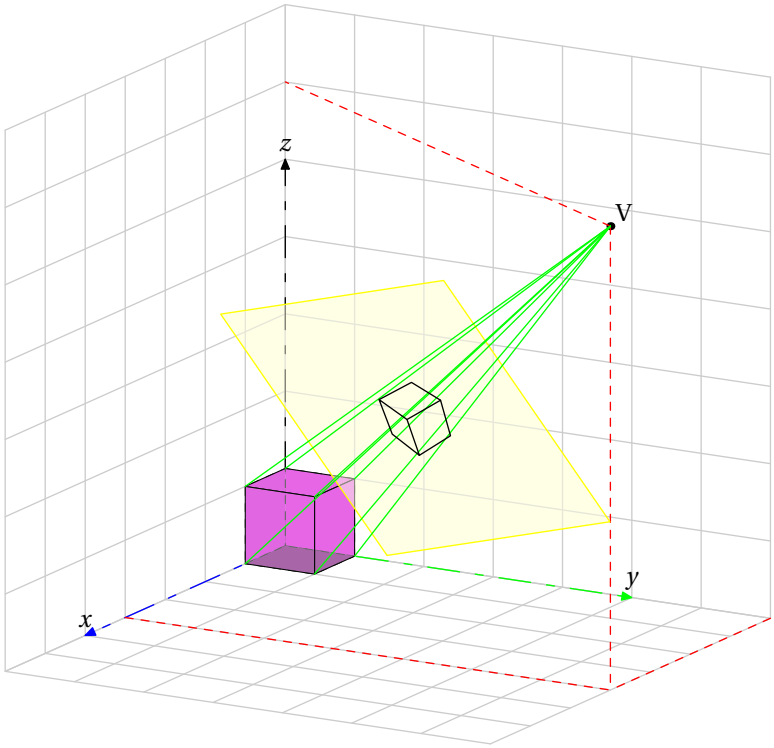


FIGURE 1.1 – Position du point de vue.

Les coordonnées de l'objet, ici le cube violet, sont données sous forme de `color` dans le repère $Oxyz$.
Les coordonnées du point de vue V, sont données dans ce même repère en coordonnées sphériques sous la forme

```
Initialisation(5,30,20,50)
```

5 étant le rayon de la sphère de centre $(0,0,0)$, $\phi = 30°$ et $\theta = 20°$.
L'écran est placé perpendiculairement à la direction $\overrightarrow{OV}$, à une distance de V égale à 50.

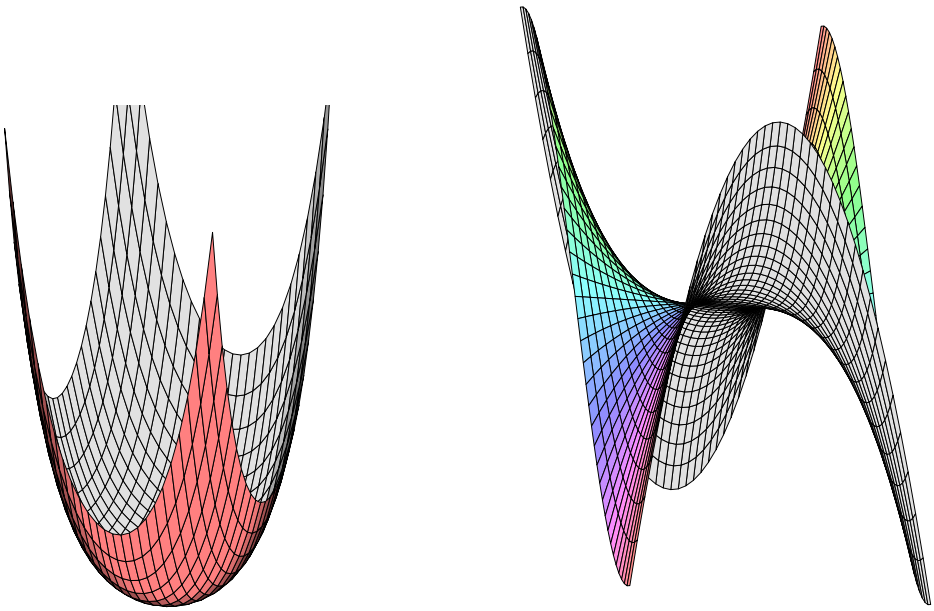
4. Disponible à l'adresse http://www-math.univ-poitiers.fr/~phan/m3Dplain.html
5. Package permettant le tracé de représentations terrestres spatiales ou planes disponibles à l'adresse http://syracuse-dev.org/mpst-globe/browser/trunk/mp-geo
6. Dans la mesure du possible, les temps de compilation importants seront indiqués.

Le rayon de la sphère n'est, *mathématiquement*, pas important. En effet, la projection choisie est une projection perspective. Seuls importent ϕ, θ et la distance à l'écran.
Par contre, il a un intérêt de précision dans les calculs de METAPOST ; c'est pour cela que bien souvent une valeur de 500 voire de 1 000 sera choisie.

Les axes peuvent être représentés *en traits d'axes* par la macro `TraceAxesD(xmax,ymax,zmax)` : les axes $[Ox)$, $[Oy)$, $[Oz)$ seront représenté de l'origine respectivement jusqu'au point (xmax;0;0), (0;ymax;0), (0;0;zmax).
Il existe une autre possibilité : `TraceAxes` qui trace les axes en traits d'axes colorés mais avec xmax = ymax = zmax = 5.

1.4.2 Les couleurs

Pour la représentation des solides, il y a deux façons de définir la coloration des faces :
– soit avec les paramètres de type `color` : `incolor` (couleur intérieure) et `outcolor` (couleur extérieure) ;
– soit avec les paramètres `arcenciel` (de type `boolean`) et `incolor`.



(a) `outcolor` et `incolor` (b) `arcenciel` et `incolor`

FIGURE 1.2 – Exemples d'utilisation de `incolor`, `outcolor` et `arcenciel`.

```
figureespace(-10u,-10u,10u,10u);
Initialisation(500,30,60,120);
outcolor:=0.5[red,white];
incolor:=1.1*gris;
draw Sparam("(u,v,(u**2+v**2)**2)"
    ,-1.25,1.25,0.1,-1.25,1.25,0.1);
finespace;
```

```
figureespace(-10u,-10u,10u,10u);
Initialisation(500,70,-10,75);
arcenciel:=true;
incolor:=1.1*gris;
draw Sparam("(v,u,u*(u**2-3*(v**2)))"
    ,-1,1,0.05,-1,1,0.05);
finespace;
```

Pour colorer, on dispose des espaces de couleur suivants :
– l'espace RGB (classique de METAPOST) ;
– l'espace HSV.

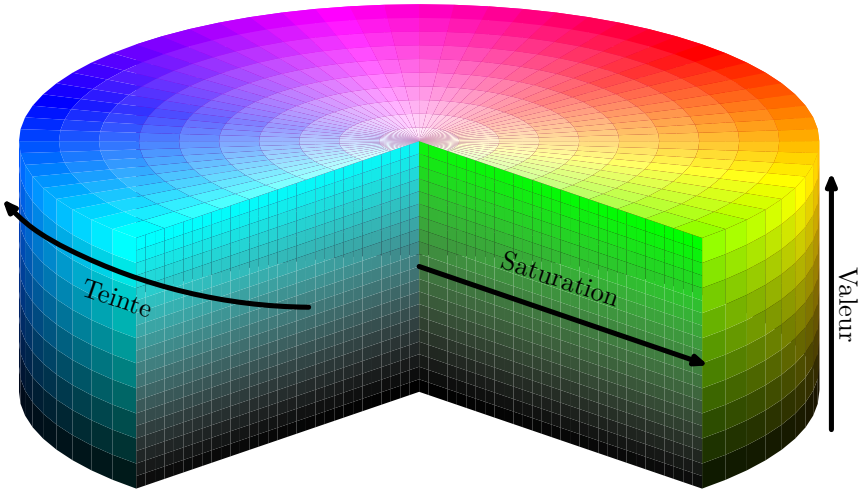
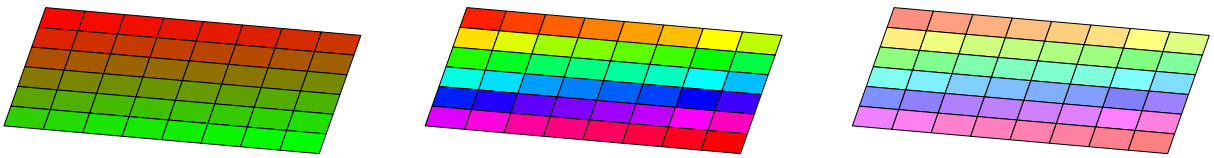


FIGURE 1.3 – Une représentation de l'espace de couleurs HSV.

Ce dernier type n'étant pas implanté dans METAPOST, il s'utilise de la façon suivante [7] :

```
Hsvtorgb(a,s,l)
```

où `a` est la teinte donnée sous la forme d'un angle compris entre 0° et 360°, `s` la saturation (entre 0 (non saturée) et 1 (saturée)), `v` la valeur (qui peut se confondre avec la luminosité) (entre 0 et 1). Par défaut, la saturation est égale à 0.45 et la luminosité à 1. On peut les modifier à l'aide des paramètres numériques `satu` et `lum`. Par exemple, on peut obtenir les dégradés suivants :



(a) degradé de rouge à vert dans l'espace RGB

(b) degradé de 0° à 360° dans l'espace HSV (saturation et luminosité maximale)

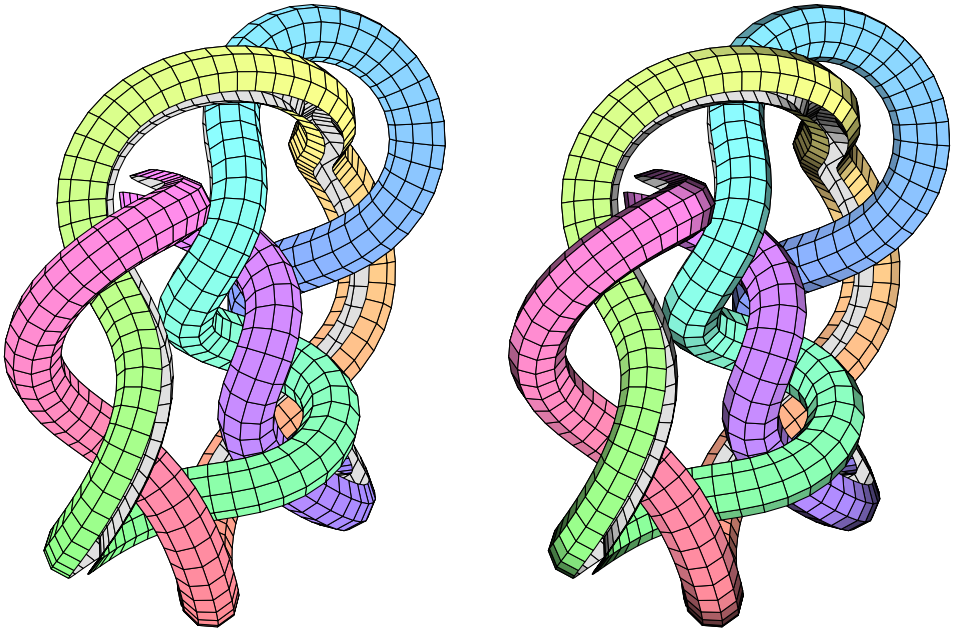
(c) degradé de 0° à 360° dans l'espace HSV (saturation à 0.5 et luminosité maximale)

FIGURE 1.4 – Dégradés dans les espaces de couleurs

1.4.3 Gestion des ombres

La lumière est gérée avec la méthode de LAMBERT. Par défaut, la gestion de la lumière est active. On peut la désactiver grâce au booléen `eclairage`.
Par défaut, la lumière est positionnée au niveau de l'oeil, *mais* on peut la repositionner grâce au type `color` `Lumiere`.
L'intensité lumineuse se règle avec le paramètre numérique `intensite`. Par défaut, elle est réglée à 2.

7. Les formules de conversions ont été obtenues à l'adresse http://en.wikipedia.org/wiki/HSL_color_space.



(a) Sans lumière (b) Avec lumière

FIGURE 1.5 – Utilisation ou non de la lumière

```
figureespace(-10u,-10u,10u,10u);
Initialisation(500,30,20,150);
eclairage:=false;
arcenciel:=true;
LectureOFF("../../data/10-61.off");
finespace;
```

```
figureespace(-10u,-10u,10u,10u);
Initialisation(500,30,20,150);
eclairage:=true;
arcenciel:=true;
LectureOFF("../../data/10-61.off");
finespace;
```

Voici trois exemples où la source lumineuse se déplace sur l'axe z : (0;0;0) puis (0;0;3) et enfin (0;0;10).

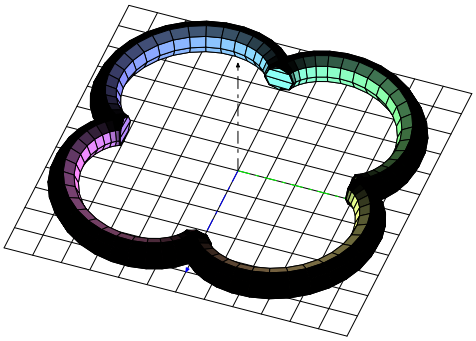


FIGURE 1.6 – Source luminueuse (1)

```
Initialisation(1000,20,45,37.5);
for k=-xm upto xm:
  draw Projette((k,-xm,0))--Projette((k,xm,0));
  draw Projette((-xm,k,0))--Projette((xm,k,0));
endfor;
Lumiere:=(0,0,0); r:=1; q:=4;
arcenciel:=true;
draw Tuben("(r*((q+1)*cos(t)-cos((q+1)*t)),r*((q
    +1)*sin(t)-sin((q+1)*t)),0.5)","(r*(-(q+1)*
    sin(t)+(q+1)*sin((q+1)*t)),r*((q+1)*cos(t)-(
    q+1)*cos((q+1)*t)),0)",0.5,0,102,0.06283);
```

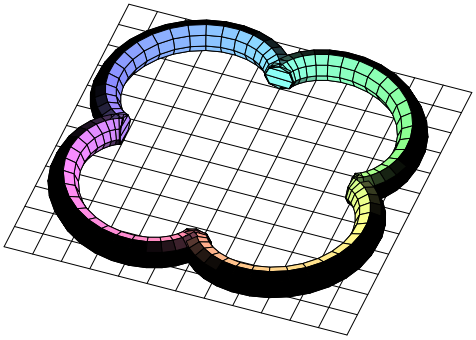


FIGURE 1.7 – Source luminueuse (2)

```
Initialisation(1000,20,45,37.5);
for k=-xm upto xm:
  draw Projette((k,-xm,0))--Projette((k,xm,0));
  draw Projette((-xm,k,0))--Projette((xm,k,0));
endfor;
Lumiere:=(0,0,3); r:=1; q:=4;
arcenciel:=true;
draw Tuben("(r*((q+1)*cos(t)-cos((q+1)*t)),r*((q
    +1)*sin(t)-sin((q+1)*t)),0.5)","(r*(-(q+1)*
    sin(t)+(q+1)*sin((q+1)*t)),r*((q+1)*cos(t)-(
    q+1)*cos((q+1)*t)),0)",0.5,0,102,0.06283);
```

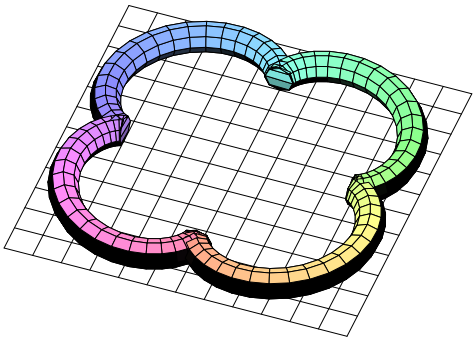


FIGURE 1.8 – Source luminueuse (3)

```
Initialisation(1000,20,45,37.5);
for k=-xm upto xm:
  draw Projette((k,-xm,0))--Projette((k,xm,0));
  draw Projette((-xm,k,0))--Projette((xm,k,0));
endfor;
Lumiere:=(0,0,10); r:=1; q:=4;
arcenciel:=true;
draw Tuben("(r*((q+1)*cos(t)-cos((q+1)*t)),r*((q
    +1)*sin(t)-sin((q+1)*t)),0.5)","(r*(-(q+1)*
    sin(t)+(q+1)*sin((q+1)*t)),r*((q+1)*cos(t)-(
    q+1)*cos((q+1)*t)),0)",0.5,0,102,0.06283);
```

Chapitre 2

Dessinons !

Je compterai toujours, pour ma part, au nombre des heures les plus douces, les plus heureuses de ma vie, celles où j'ai pu saisir dans l'espace et étudier sans trève quelques-uns de ces êtres géométriques qui flottent en quelque sorte autour de nous.

GASTON DARBOUX

2.1 La lecture externe

Avant de se lancer dans les calculs, profitons du fait que de nombreux logiciels de contruction 3D exportent leurs données dans divers formats de fichiers. Pourquoi donc ne pas s'en servir, couplé à METAPOST, pour représenter de tels objets ?

2.1.1 Les fichiers `OFF`

Si vous disposez d'un fichier `toto.off`, on l'utilisera comme ceci :

```
LectureOFF("toto.off")
```

- Avant l'utilisation des fichiers `OFF`, il faudra s'assurer qu'ils aient une syntaxe du type

```
1        nbsommets nbfaces
2
3        x1 y1 z1
4        x2 y2 z2
5        ...
6        nbsommetsface1 1 2 3
7        nbsommetsface2 2 3 4
8        ...
```

- Suivant les fichiers `OFF`, il y aura deux paramètres à modifier : `echelle` qui applique un coefficient divisant les données numériques du fichier par la valeur choisie ; `debut` qui indiquera quel est le chiffre correspondant au premier sommet (bien souvent, c'est 0 mais pour quelques fichiers, cela peut-être 1).
- Le paramètre `invnormale` devra *parfois* être adapté. Par défaut, il est à 1. On devra le mettre parfois comme étant égal à -1.

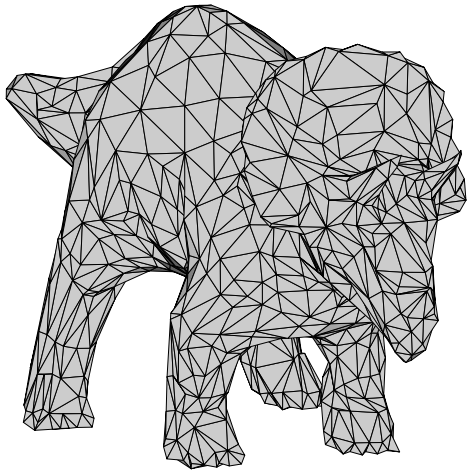
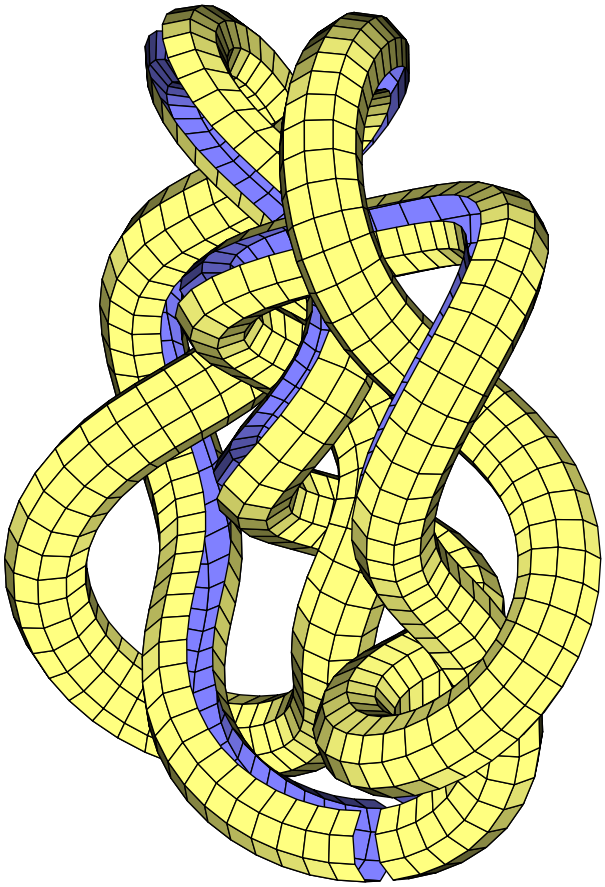


FIGURE 2.1 – Un tricératops.

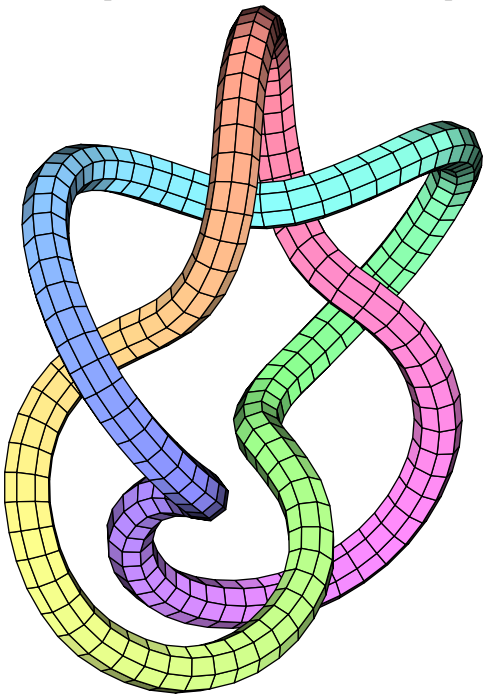
```
%fichier off:http://www.irit.fr/~Loic.Barthe/Enseignements/TPs_OpenGL/L3_IUP_SI/TP7/Maillages/triceratops.off
echelle:=2;
debut:=0;

invnormale:=-1;

figureespace(-10u,-10u,10u,10u);
Initialisation(500,-25,10,50);
outcolor:=0.8*white;
incolor:=gris;
LectureOFF("triceratops.off");
finespace;
```

L'exemple ci-dessous montre un *noeud creusé*. Pour cela, on a un paramètre booléen `Creux` (positionné à `true` par défaut) qui permet d'indiquer si le solide envisagé est creusé ou pas ; creusé dans le sens « ouvert ». Ce paramètre n'est, pour le moment, disponible que pour la lecture des fichiers externes.



```
%www-c.inria.fr/gamma/download/download.php
echelle:=2; debut:=1;
Creux:=false;
outcolor:=0.5[jaune,white];
incolor:=0.5[bleu,white];

figureespace(-10u,-10u,10u,10u);
Initialisation(500,-25,10,35);
LectureOFF("10-61.off");
finespace;
```

Pour ne pas abuser, un dernier exemple.



```
echelle:=5;
debut:=1;
arcenciel:=true;
invnormale:=-1;
figureespace(-10u,-10u,10u,10u);
Initialisation(500,30,20,50);
LectureOFF("5_1.pc.off");
finespace;
```

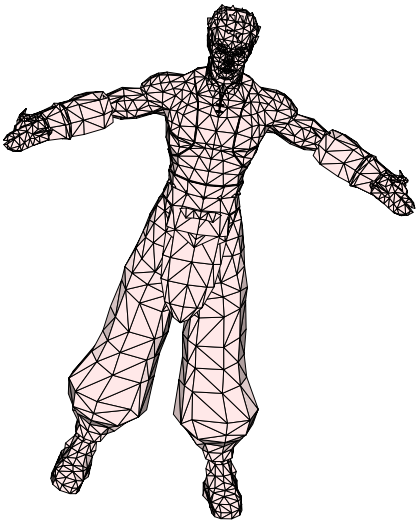
2.1.2 Les fichiers OBJ

Si vous disposez d'un fichier `toto.obj`, on l'utilisera comme ceci : `LectureOBJ("toto.obj")`

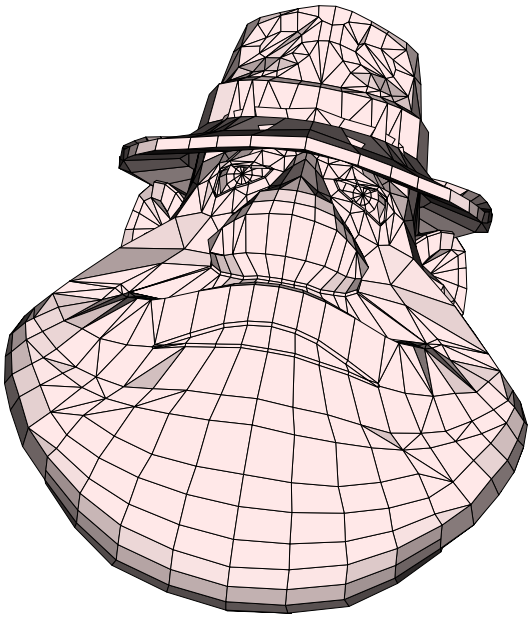
- Avant l'utilisation des fichiers OBJ, il faudra s'assurer qu'ils aient une syntaxe du type

```
v x1 y1 z1
v x2 y2 z2
...
v xn yn zn
f nbsommetsface1 1 2 3 ...
f nbsommetsface2 2 3 4 ...
...
```

  En effet, certains fichiers OBJ contiennent les descriptions des vecteurs normaux aux faces. Ils ne sont pas nécessaires pour nous car METAPOST fait les calculs nécessaires. Il pourrait cependant être utile d'utiliser directement de tels fichiers pour gagner du temps de compilation...
- Suivant les fichiers OBJ, il y aura à modifier le paramètre `echelle`.
- Le paramètre `invnormale` devra *parfois* être adapté. Par défaut, il est à 1. On devra le mettre parfois comme étant égal à -1.



```
%http://www-c.inria.fr/gamma/download/
echelle:=50;

figureespace(-10u,-10u,10u,20u);
Initialisation(2500,80,-70,50);
outcolor:=0.7[rose,white];
incolor:=jaune;
LectureOBJ("Midpoly_04.obj");
finespace;
```



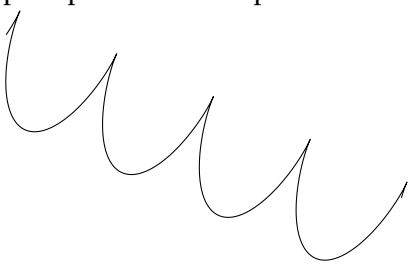
```
%http://www-c.inria.fr/gamma/download/
%avec modification
echelle:=0.25;

figureespace(-10u,-10u,10u,20u);
Initialisation(2500,-80,80,50);
outcolor:=0.7[rose,white];
incolor:=jaune;
LectureOBJ("Y3483.obj");
finespace;
```

2.2 Les courbes

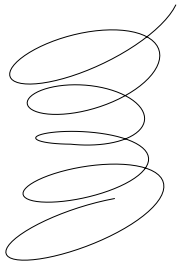
Pour tracer une fonction du type $M(x,y,z) = (f(t), g(t), h(t))$, on utilisera la macro `Fonction`.

```
Fonction(expr fn,tmin,tmax,pas)
```

où `fn` est un type `string` donnant les fonctions f, g et h (ne pas oublier de la mettre sous forme d'un triplet) ; `tmin` et `tmax` sont les bornes de l'intervalle de tracé et `pas` le pas d'incrémentation pour positionner les points et les relier ensuite.



```
figureespace(-10u,-10u,10u,10u);
Initialisation(5,30,50,20);
draw Fonction("(sin(t),cos(t),t/3)",-2,25,0.01);
finespace;
```



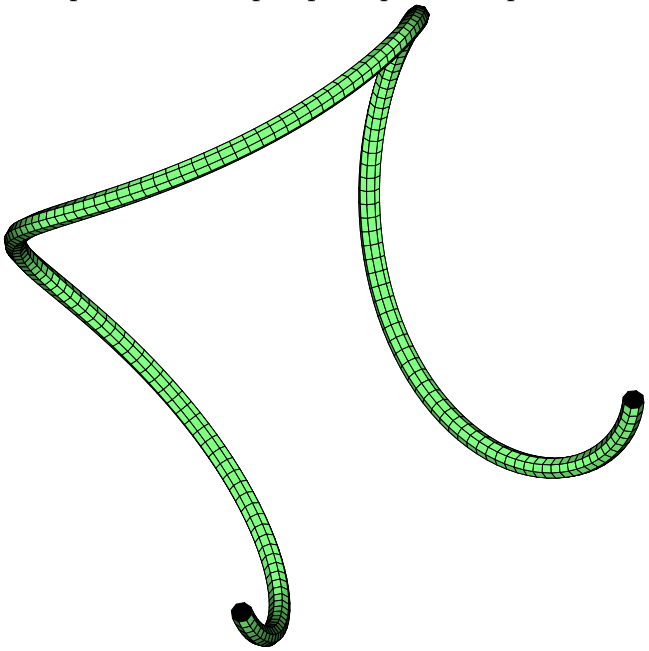
```
figureespace(-10u,-10u,10u,10u);
Initialisation(5,30,20,20);
draw Fonction("(cos(t)*(1+abs(t)/5),1.5*sin(t),0.2*
    t)",-15,15,0.1);
finespace;
```

2.2.1 Les tubes : 1er essai

On a une amélioration graphique : la possibilité de tracer les courbes sous forme de tube avec la macro `Tube` :

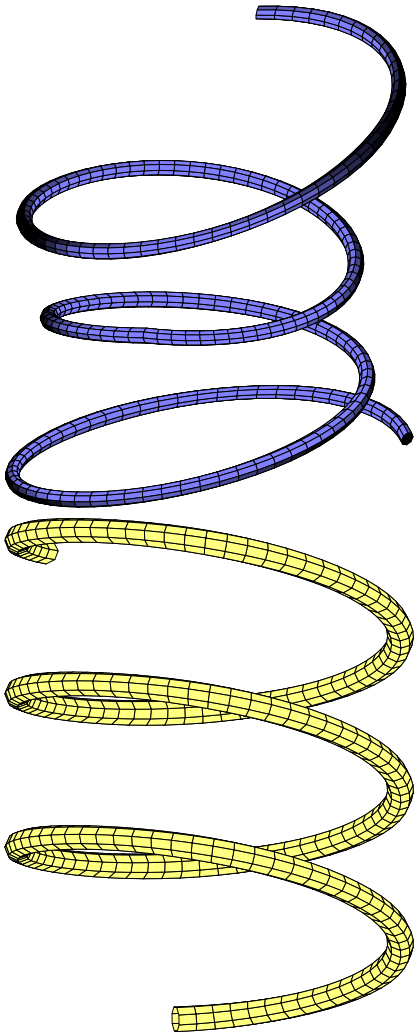
```
Tube(expr Fn,dp,ds,rayon,tmin,nbp,pas)
```

où `Fn`, `dp` et `ds` sont des types `string` représentant respectivement la fonction de référence sur laquelle le tube est construit ; la dérivée première et la dérivée seconde. Pourquoi ? Pour calculer les vecteurs $(\vec{T}, \vec{N}, \vec{B})$ constituant le repère de Frenet local attaché à la courbe. Il est d'ailleurs vivement recommandé de les calculer à la main ou à l'aide d'un logiciel de calcul formel.
Une fois ceci fait, `tmin` la valeur du paramètre de départ, `nbp` le nombre de pas voulus, `pas` le pas pour passer d'un point à un autre.
Voici plusieurs exemples pour que ce soit plus clair :



```
figureespace(-10u,-20u,10u,10u);
Initialisation(1000,15,70,50);
outcolor:=0.5[green,white];
draw Tube("(2*cosd(t)+cosd(3*t),2*
    sind(t)-sind(3*t),2*sqrt(2)*sind
    (2*t))","(-2*sind(t)-3*sind(3*t)
    ,2*cosd(t)-3*cosd(3*t),4*sqrt(2)
    *cosd(2*t))","(-2*cosd(t)-9*cosd
    (3*t),-2*sind(t)+9*sind(3*t),-8*
    sqrt(2)*sind(2*t))"
    ,0.1,90,180,1.5);
finespace;
end
```

```
figureespace(-10u,-10u,10u,10u);
Initialisation(5,30,50,20);
outcolor:=0.5[red,white];
draw Tube("(sin(t),cos(t),t/3)","(cos(t),-sin(t)
    ,1/3)","(-sin(t),-cos(t),0)",0.1,-2,270,0.1);
finespace;
```



```
figureespace(-10u,-20u,10u,10u);
Initialisation(5,30,20,20);
outcolor:=0.5[blue,white];
draw Tube("(cos(t)*(1+abs(t)/5),1.5*sin(t),0.2*t)",
    "if t>=0:(-sin(t)*(1+t/5)+cos(t)/5,1.5*cos(t)
    ,0.2) else: (-sin(t)*(1-t/5)-cos(t)/5,1.5*cos(t
    ),0.2) fi", "if t>=0:(-cos(t)*(1+t/5)-2*sin(t)
    /5,-1.5*sin(t),0) else: (-cos(t)*(1-t/5)+2*sin(
    t)/5,-1.5*sin(t),0) fi",0.075,-15,300,0.1);
finespace;
```

```
figureespace(-10u,-20u,10u,10u);
Initialisation(1000,15,20,50);
outcolor:=0.5[jaune,white];
draw Tube("(cos(t),1.5*sin(t),0.2*t)","(-sin(t)
    ,1.5*cos(t),0.2)","(-cos(t),-1.5*sin(t),0)"
    ,0.1,0,180,0.1);
finespace;
```

2.2.2 Les tubes : 2e essai

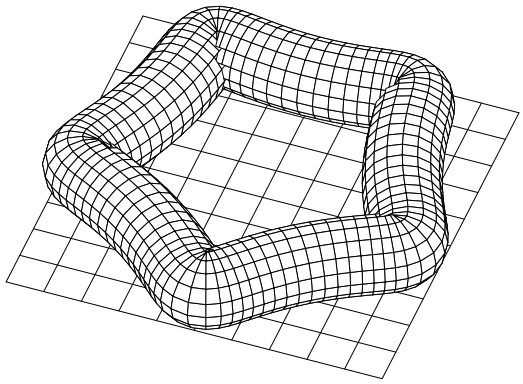


FIGURE 2.2 – Hypocycloïde en tube

Cependant, pour obtenir l'image ci-dessous tirée de la documentation de `pst-solides3d`[1], cela ne

1. Mais celle présente dans cette documentation a été compilée par METAPOST.

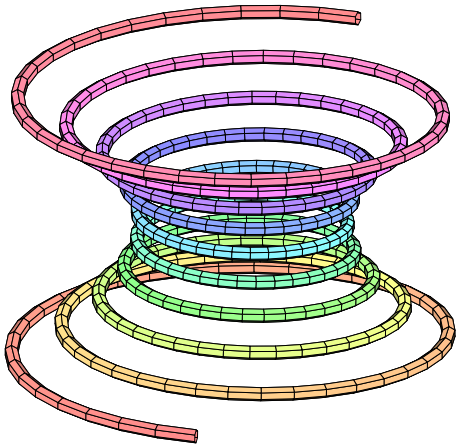
marche pas ! J'ai donc crée une deuxième macro permettant de construire de telles figures. En fait, le problème venait de l'inversion des normales au passage des points d'inflexion. De plus, grâce à un document trouvé sur la toile [2], il n'est plus nécessaire d'utiliser la dérivée seconde. La macro utilisable est

```
Tuben(expr Fn,dp,rayon,tmin,nbp,pas,couleur)
```

avec les mêmes conventions que lors du premier essai. Pour obtenir la figure 2.2, on utilisera alors le code

```
figureespace(-10u,-10u,10u,10u);

nb:=16; unit:=0.8;

Initialisation(1000,20,45,37.5);
for k=-5 upto 5:
 draw Projette((k,-5,0))--Projette((k,5,0));
 draw Projette((-5,k,0))--Projette((5,k,0));
endfor;
eclairage:=false;
outcolor:=blanc;
draw Tuben("(4*cos(t)+cos(4*t)/2,4*sin(t)-sin(4*t)/2,1)","(-4*sin(t)-2*sin(4*t),4*
    cos(t)-2*cos(4*t),0)",1,0,102,0.06283);

finespace;
```

À noter que le paramètre `unit` permet de faire un agrandissement (ou une réduction) de la figure. Par défaut, `unit=1`. De plus, il existe le paramètre `nb` qui permet de moduler le nombre de points sur le cercle permettant de tracer le tube.
Et pour finir, étant fan de Gaston Lagaffe, je me permets de reprendre le ressort-siège de la documentation de `pst-solides3d`.



```
figureespace(-10u,-10u,10u,10u);

nb:=6;
Initialisation(1000,20,20,25);
arcenciel:=true;
draw Tuben("((t**2+3)*sin(15*t),(t**2+3)*cos
    (15*t),2*t)","(2*t*sin(15*t)+15*((t**2)+3)
    *cos(15*t),2*t*cos(15*t)-15*((t**2)+3)*sin
    (15*t),2)",0.2,-2,360,1/90);

finespace;
```

FIGURE 2.3 – Siège-ressort de Gaston Lagaffe

2. Malheureusement, je ne dispose plus de l'adresse. Par contre, l'auteur est Loïc BARTHE de l'équipe Vortex de l'IRIT-UPS Toulouse.

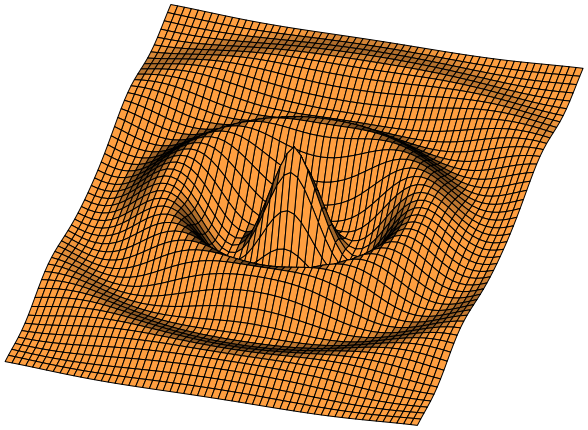
2.3 Les surfaces

2.3.1 Les surfaces en z

De la forme $z = f(x, y)$, on utilisera alors

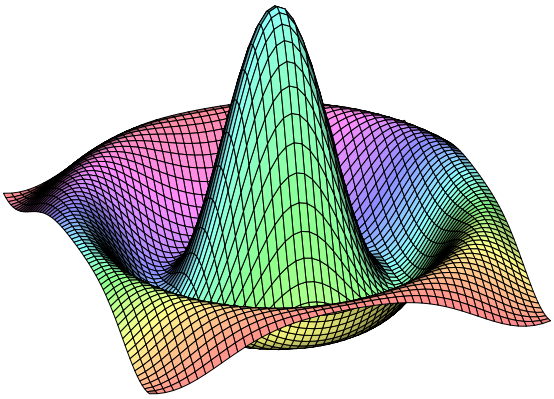
```
Surfz(fn,xmin,xmax,ymin,ymax,nbl,nbp)
```

avec `fn` la fonction écrite sous un type `string`, `xmin`, `xmax`, `ymin` et `ymax` les bornes des intervalles de tracés, `nbl` le nombre de lignes suivant les y[3] et `nbp` le nombre de points utilisés sur chaque ligne pour tracer la surface.
Les trois exemples ci-dessous ont donné des temps de compilation de l'ordre de 40 secondes.



$$f(z) = 4\cos\left(\sqrt{x^2+y^2}\right)e^{-50\sqrt{x^2+y^2}} \text{ avec } \begin{cases} x \in [-15,15] \\ y \in [-10,10] \end{cases}$$

```
1 invnormalelum:=-1;
2
3 figureespace(-10u,-10u,10u,10u);
4 Initialisation(300,15,35.264,10);
5 incolor:=0.5[green,white];
6 outcolor:=0.25[orange,white];
7 draw SurfZ("4*cos(X++Y)*mexp(-(X
      ++Y)*50)"
      ,-15,15,-10,10,60,150);
8 finespace;
```



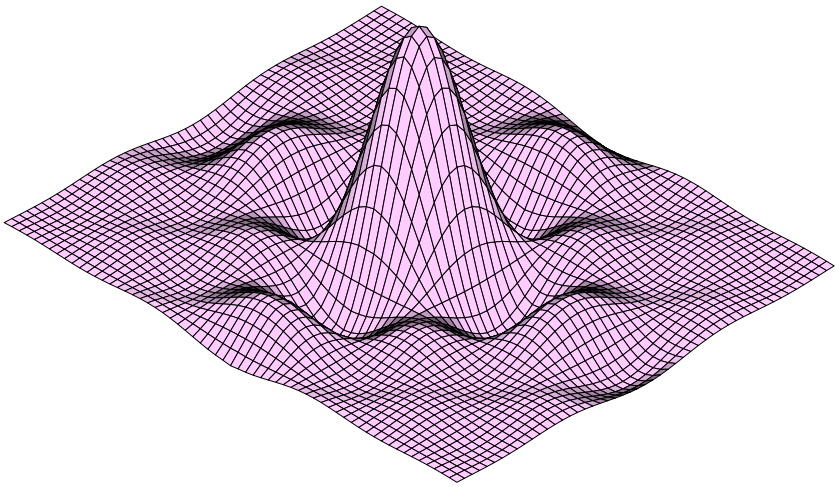
$$f(z) = 10 \times \frac{\sin\left(\sqrt{x^2+y^2}\right)}{\sqrt{x^2+y^2}} \text{ avec } \begin{cases} x \in [-8,8] \\ y \in [-8,8] \end{cases}$$

```
1 invnormalelum:=-1;
2
3 figureespace(-10u,-10u,10u,10u);
4 Initialisation(300,70,30,10);
5 incolor:=0.5[jaune,white];
6 outcolor:=0.8[violet,white];
7 arcenciel:=true;
8 draw SurfZ("if (X=0) and (Y=0):10
       else:10*sin((X++Y))/(X++Y)
      fi",-8,8,-8,8,60,150);
9 finespace;
```

L'exemple précédent montre l'utilisation du paramètre `invnormalelum` permettant d'opposer la normale permettant la détermination de la quantité lumineuse. De plus, la gestion de la lumière est un peu différente lors du tracé des surfaces en z. En effet, on a parfois besoin de voir ce qui se passe dans la partie cachée...

3. Il en est ainsi car j'ai suivi la méthode décrite dans le livre de Raymond Dony. Il est à noter également que pour des impératifs de temps de compilation et de poids des images, les deux paramètres `nbp` et `nbl` sont des multiples de 3.

```
invnormalelum:=-1;

figureespace(-10u,-10u,10u,10u);
Initialisation(300,45,35,10);
incolor:=0.5[jaune,white];
outcolor:=0.8[violet,white];
draw SurfZ("if X=0:if Y=0:10 else:10*sin(Y)/Y fi else:if Y=0:10*sin(X)/X else: 10*
    sin(X)/X*sin(Y)/Y fi fi",-10,10,-12,12,60,150);
finespace;
```



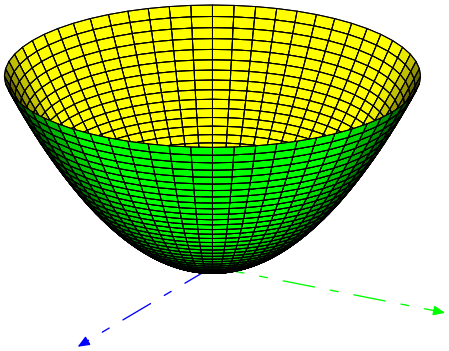
$$f(z) = 10 \times \frac{\sin x}{x}\frac{\sin y}{y} \text{ avec } \begin{cases} x \in [-10,10] \\ y \in [-12,12] \end{cases}$$

Mais on peut également vouloir un maillage circulaire... Dans ce cas,

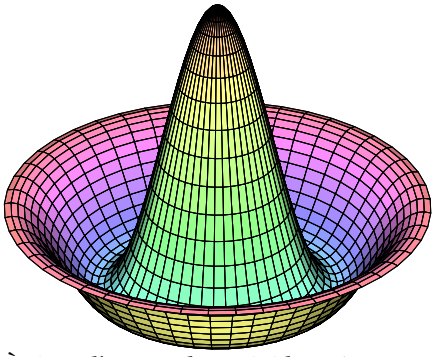
`Surfz(fn,rmax,pray,nbl)`

avec `fn` la fonction écrite sous un type `string`, `rmax` le rayon maximal du maillage circulaire, `pray` le pas entre deux rayons consécutifs et `nbl` le nombre de lignes situés dans la couronne `rmax` et `rmax+pray`.

Dans les deux cas de maillages, la macro à utiliser est la même ; seul le nombre d'arguments diffère.

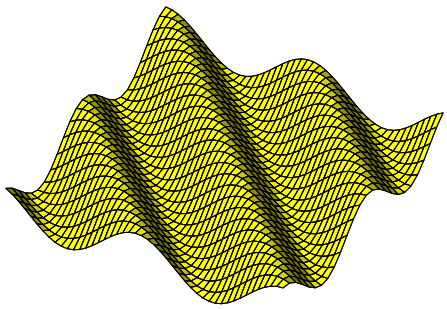


```
figureespace(-100u,-100u,100u,100u);
Initialisation(5000,30,20,60);
TraceAxes;
outcolor:=jaune;
incolor:=vert;
draw SurfZ("(X*X+Y*Y)/4",4,0.1,60);
finespace;
```

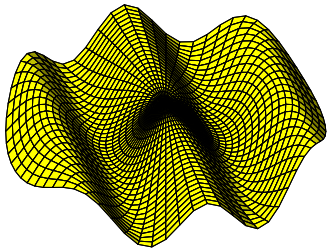


```
figureespace(-10u,-10u,10u,10u);
Initialisation(300,70,30,10);
incolor:=0.5[jaune,white];
outcolor:=0.8[violet,white];
arcenciel:=true;
draw SurfZ("if (X=0) and (Y=0):10 else
    :10*sin((X++Y))/(X++Y) fi"
    ,8,0.25,48);
finespace;
```

À titre d'exemple, voici la même surface représentée avec les deux maillages.

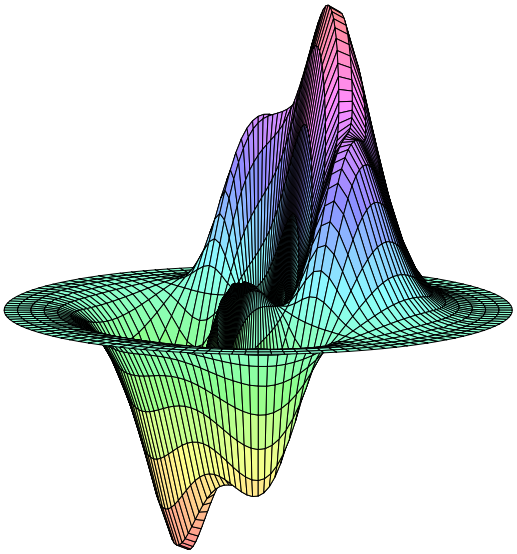


```
figureespace(-100u,-100u,100u,100u);
Initialisation(5000,30,40,10);
outcolor:=jaune;
incolor:=vert;
draw SurfZ("cos(abs(X-Y))"
    ,-6,6,-6,6,45,75);
finespace;
```

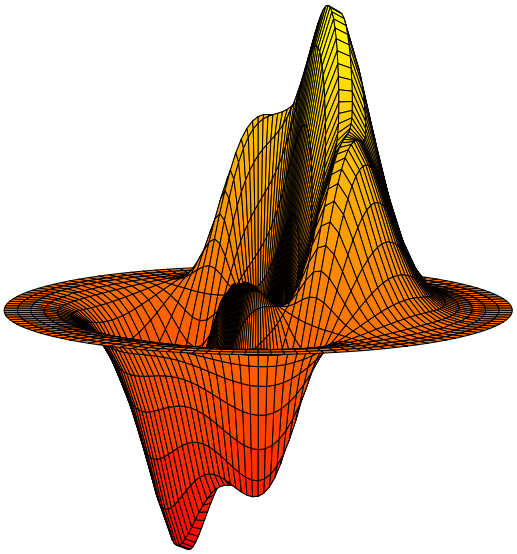


```
figureespace(-100u,-100u,100u,100u);
Initialisation(5000,30,40,10);
outcolor:=jaune;
incolor:=vert;
draw SurfZ("cos(abs(X-Y))",6,0.25,72);
finespace;
```

Une autre option est également disponible : le coloriage en fonction de la coordonnées en z. Les exemples suivants devraient être assez parlant...



```
figureespace(-10u,-10u,10u,20u);
Initialisation(500,-120,10,25);
couleurz:=true;
rayd:=0.01;
draw SurfZ("10*(X**3+X*(Y**4)-(X/5))*exp
    (-(X**2+Y**2))+exp(-((X-1.225)**2+Y
    **2))",4,0.2,120);
finespace;
```



```
figureespace(-10u,-10u,10u,20u);
Initialisation(500,-120,10,25);
couleurz:=true;
cz1:=jaune;
cz2:=rouge;
rayd:=0.01;
draw SurfZ("10*(X**3+X*(Y**4)-(X/5))*exp
    (-(X**2+Y**2))+exp(-((X-1.225)**2+Y
    **2))",4,0.2,120);
finespace;
```

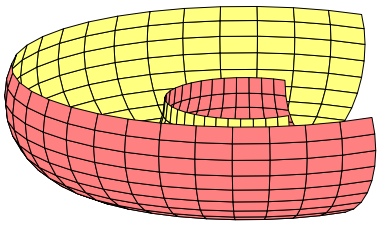
2.3.2 ...paramétrées $M(u,v) = (f(u,v), g(u,v), h(u,v))$

Attention, il faut une orientation correcte des faces créees. Il sera donc nécessaire d'utiliser, *parfois*, `invnormale`.

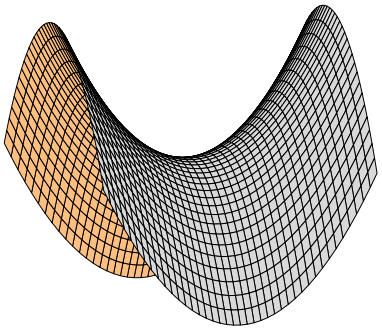
Utilisons la macro

```
Sparam(expr fn,umin,umax,upas,vmin,vmax,vpas)
```

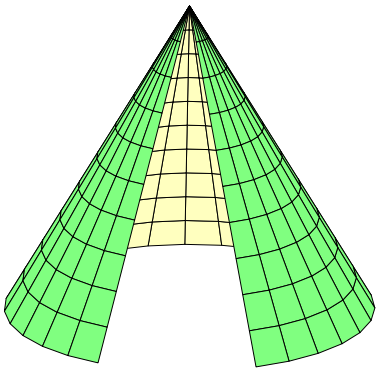
où `fn` est un type `string` représentant la fonction $M(u,v)$ donnée sous forme d'un triplet ; les autres paramètres parlant d'eux mêmes.
Voici plusieurs exemples :



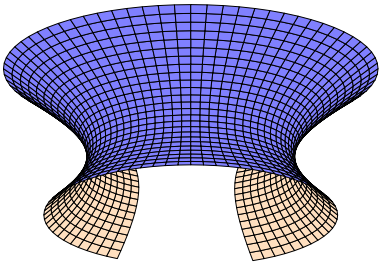
```
figureespace(-10u,-10u,10u,10u);
Initialisation(500,-30,15,75);
outcolor:=0.5[red,white];
incolor:=0.5[jaune,white];
draw Sparam("((1+0.5*cos(u))*cos(v),(1+0.5*cos(u))*
    sin(v),0.5*sin(u))",-pi,pi/12,pi/15,2*pi/3,2*pi
    ,pi/20);
finespace;
```



```
figureespace(-10u,-10u,10u,10u);
Initialisation(50,-110,10,100);
outcolor:=0.5[orange,white];
incolor:=0.25[gris,white];
draw Sparam("(u,v,u*u-v*v)",-1,1,0.05,-1,1,0.05);
finespace;
```



```
figureespace(-10u,-10u,10u,10u);
Initialisation(500,10,20,100);
incolor:=0.75[jaune,white];
outcolor:=0.5[green,white];
draw Sparam("(u*cos(v)*sin(pi/6),u*sin(v)*sin(pi/6)
    ,u*cos(pi/6))",-2,-0.1,0.2,-2.6,2.6,0.2);
finespace;
```



```
figureespace(-10u,-10u,10u,10u);
Initialisation(50,0,-20,100);
incolor:=0.75[orange,white];
outcolor:=0.5[blue,white];
draw Sparam("(cos(v)*sqrt(1+u*u),sin(v)*
    sqrt(1+u*u),0.6*u)",-1,1.5,0.1,-5*pi
    /6,5*pi/6,pi/36);
finespace;
```

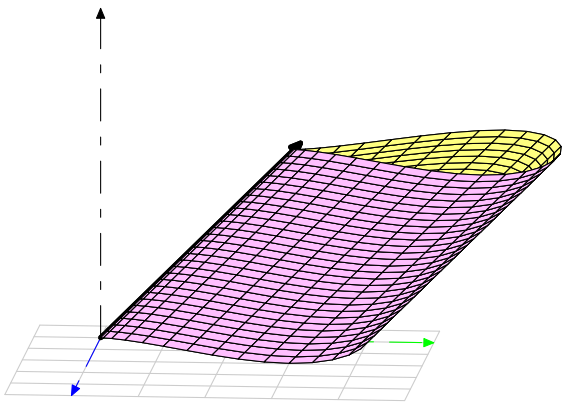


FIGURE 2.4 – Une nappe cylindrique

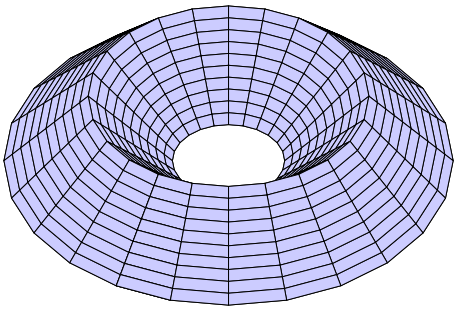
```
figureespace(-10u,-10u,10u,10u);
Initialisation(100,5,10,25);
incolor:=0.5[jaune,white];
outcolor:=0.75[violet,blanc];
drawarrow Projette((0,0,0))--Projette((0,3,3))
    withpen pencircle scaled2bp;
for l=-1 upto 5:
  draw Projette((l,-1,0))--Projette((l,5,0))
      withcolor gris;
  draw Projette((-1,l,0))--Projette((5,l,0))
      withcolor gris;
endfor;
TraceAxes;
drawarrow Projette((0,0,0))--Projette((0,3,3))
    withpen pencircle scaled2bp;
draw Sparam("(6*(cos(u)**3)*sin(u),4*cos(u)*
    cos(u)+v,v)",0,2*pi,pi/30,0,2*sqrt(2),0.1)
    ;
finespace;
```

2.3.3 Les solides de révolution

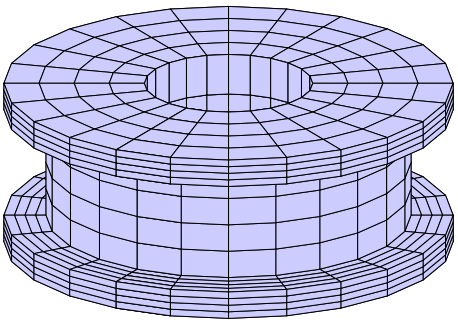
C'est un cas particulier des surfaces paramétrées. La syntaxe est donc très voisine :

```
Revolution(fn,umin,umax,upas,vmin,vmax,vpas)
```

avec les mêmes notations que précédemment *sauf* que `fn` doit représenter un chemin METAPOST. Voici quelques exemples.

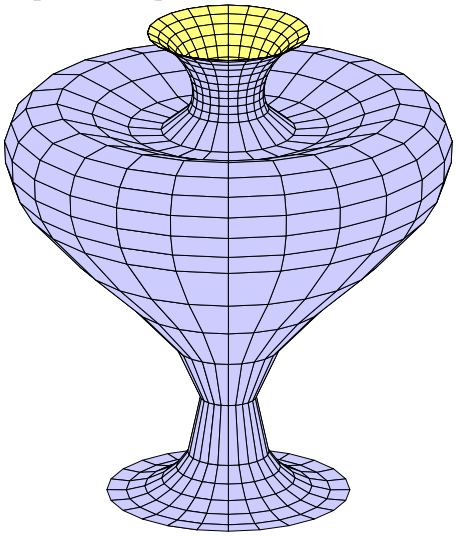


```
r=2; R=8; h=3;

path pp;
pp=(R,-h/2)--((R+r)/2,h/2)--(r,-h/2)--cycle;

incolor:=0.5[jaune,white];
outcolor:=0.8[bleu,white];

figureespace(-10u,-10u,10u,10u);
Initialisation(500,30,40,25);
draw Revolution("pp",0,length pp,0.1,0,2*pi,pi/12);
finespace;
```



```
r=3; R=8; h=6;

path pp;
pp=(r,h/2)--(r,-h/2)--(R,-h/2)--(R,-h/2+1)--(R
    -1.5,-h/2+1)--(R-1.5,h/2-1)--(R,h/2-1)--(R,h/2)
    --cycle;

incolor:=0.5[jaune,white];
outcolor:=0.8[bleu,white];

figureespace(-10u,-10u,10u,10u);
Initialisation(500,30,20,25);
draw Revolution("pp",0,length pp,0.25,0,2*pi,pi/12)
    ;
finespace;
```

Et pour tirer profit des courbes de Bezier

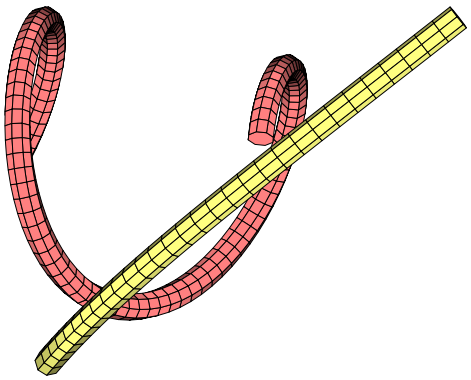


```
path pp;
pp=(3,-7)..(1,-6)..(5,0.5)..(5,3)..(1,3)..(1,4)
    ..(2,5);

incolor:=0.5[jaune,white];
outcolor:=0.8[bleu,white];

figureespace(-10u,-10u,10u,10u);
Initialisation(500,30,20,25);
draw Revolution("pp",0,length pp,0.2,0,2*pi,pi/12);
finespace;
```

Chapitre 3

La fusion d'objets

Un petit aperçu...



Tout ce qui est relatif à la fusion d'objets n'est *valable* que pour ce chapitre. Ce n'est pas implanté pour les surfaces, les courbes,...

3.1 Quels objets fusionner ?

Avant de voir ce que nous permet la fusion, voyons quels objets sont mis à notre disposition : `tétraèdre régulier`, `cube` et `pavé droit`, `octaèdre régulier`, `dodécaèdre régulier`, `icosaèdre régulier`, `prisme` et `prisme creux`, `cylindre de révolution creux` et `plein`, `cône de révolution creux` et `plein`, `tronc de cône de révolution creux` et `plein`, `sphère`, `calotte sphérique creuse` et `pleine`, `tore`, `anneau`, `tube`, `grille`, `ruban`, `biface`, `OFF`, `OBJ` et `New`.

Dans ce chapitre, aucun de ces objets n'est dessiné ! Mais ils sont numérotés. Ainsi pour les afficher, Il faut impérativement utiliser la macro

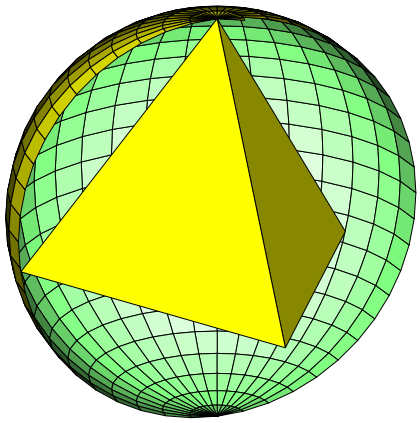
`AffichageObjet1`

1 étant le numéro de l'objet à afficher.

Voici comment les définir :

Tétraèdre régulier

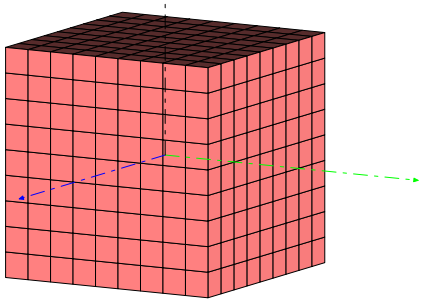
"a=3" (a rayon de la sphère circonscrite)



```
Objettetraedre1("a=1");
AffichageObjet1;
```

Cube

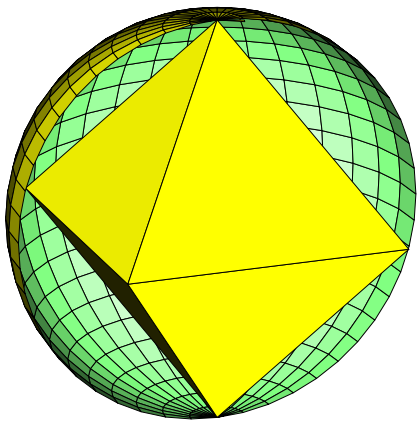
"a=3"



```
%subh:=9;
Objetcube1("a=4");
AffichageObjet1;
```

Octaèdre régulier

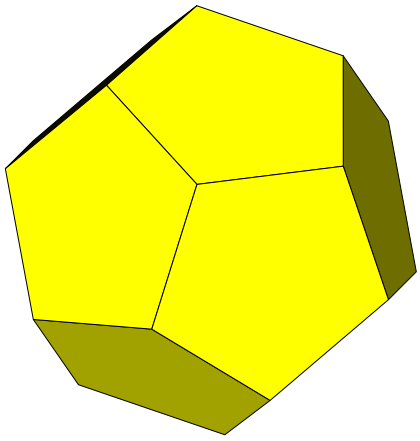
"a=3" (a rayon de la sphère circonscrite)



```
Objetoctaedre1("a=1");
AffichageObjet1;
```

Dodécaèdre régulier

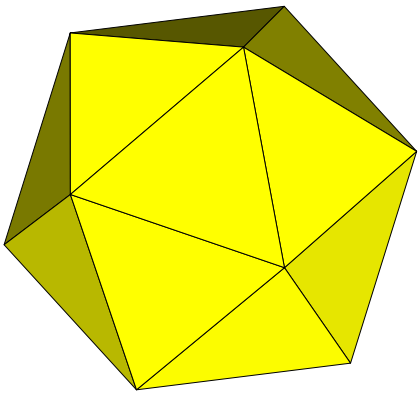
"a=3" (a rayon de la sphère circonscrite)



```
Objetdodecaedre1("a=1");
AffichageObjet1;
```

Icosaèdre régulier

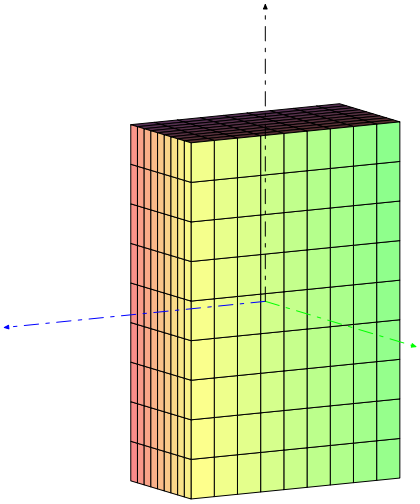
"a=3" (a rayon de la sphère circonscrite)



```
Objeticosaedre1("a=1");
AffichageObjet1;
```

Pavé droit

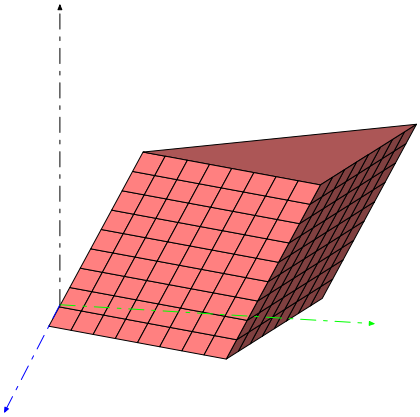
"L=2","H=1", "P=0.5"



```
%subh:=9;
Objetpave1("L=2","H=6","P=4");
AffichageObjet1;
```

Prisme plein

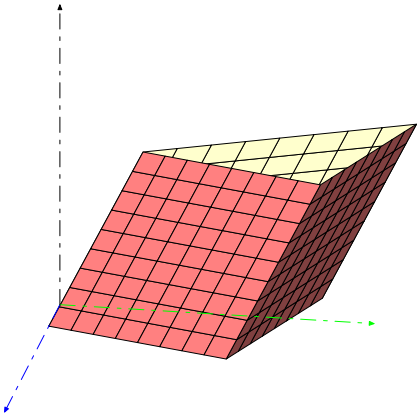
("axe=(0,1,2)" ,"h=3") (Liste sommets)



```
%nb:=4;subh:=20;
ObjetPrisme1("axe=(0,0.5,1)",%
"h=2")%
((1,0,0),(2,3,0),(-1,4,0));
AffichageObjet1;
```

Prisme creux

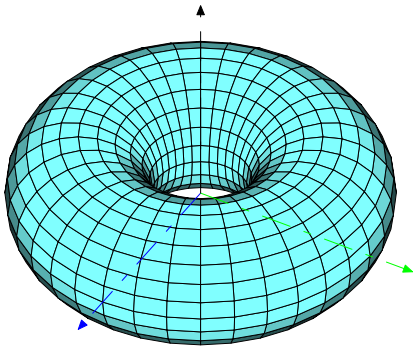
("axe=(0,1,2)" ,"h=3") (Liste sommets)



```
%nb:=4;subh:=20;
creux:true;
Objetprisme1("axe=(0,0.5,1)",%
"h=2")%
((1,0,0),(2,3,0),(-1,4,0));
AffichageObjet1;
```

Tore

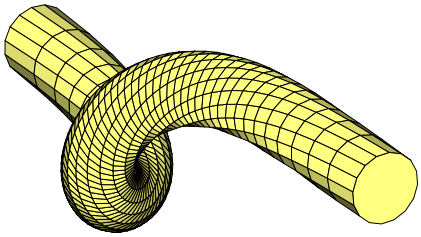
"R=3","r=2"



```
%nb:=24;subh:=36;
Objettore1("R=4","r=1");
AffichageObjet1;
```

Tube plein [1]

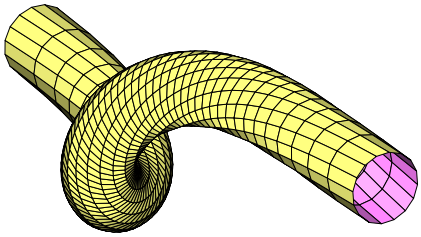
"F(t)","F'(t)", rayon, umin, nb de points, pas



```
%nb:=16;subh:=18;
ObjetTube1("(%
2*(1+cos(t)),2*tan(t/2),2*sin(t)%
)","(%
-2*sin(t),2/((cos(t/2))**2),2*cos(
)", 1,-2.7468,71,0.0763);
AffichageObjet1;
```

Tube creux

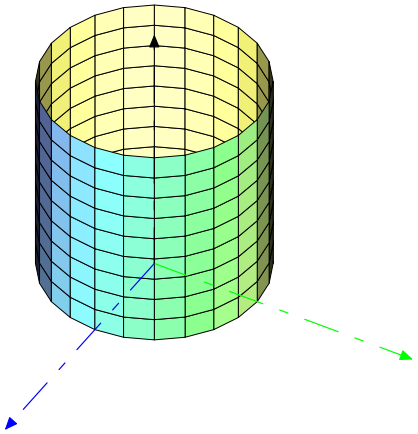
"F(t)","F'(t)", rayon, umin, nb de points, pas



```
%nb:=16;subh:=18;
creux:=true;
ObjetTube1("(%
2*(1+cos(t)),2*tan(t/2),2*sin(t)%
)","(%
-2*sin(t),2/((cos(t/2))**2),2*cos(
)",1,-2.7468,71,0.0763);
AffichageObjet1;
```

Cylindre de révolution creux

"r=1","h=2"

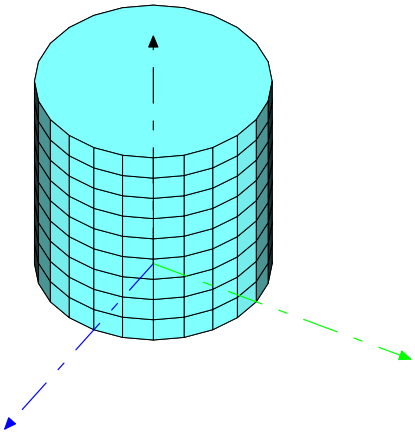


```
creux:=true;
%nb:=24;subh:=9;
Objetcylindre1("r=2","h=4");
AffichageObjet1;
```

1. Le tube représenté ici est relatif à la courbe *horoptère*.

Cylindre de révolution plein

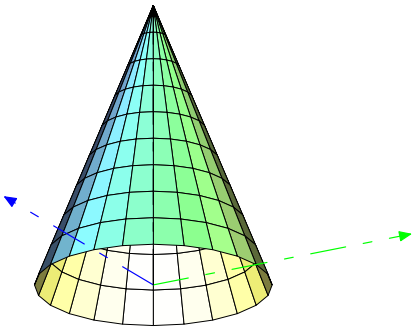
"r=1","h=2"



```
creux:=false;
%nb:=24;subh:=9;
Objetcylindre1("r=2","h=4");
AffichageObjet1;
```

Cône de révolution creux

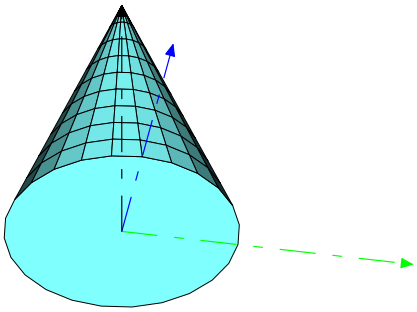
"r=1","h=2"



```
creux:=true;
%nb:=24;subh:=9;
Objetcone1("r=1.5","h=4");
AffichageObjet1;
```

Cône de révolution plein

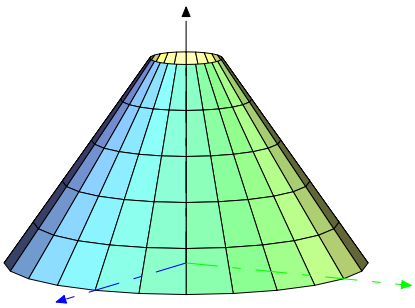
"r=1","h=2"



```
creux:=false;
%nb:=24;subh:=9;
Objetcone1("r=1.5","h=4");
AffichageObjet1;
```

Tronc de cône creux

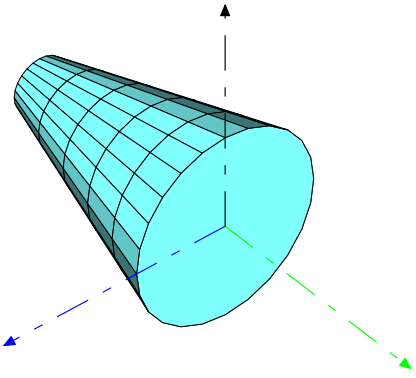
"r=1", "h=2", "H=5"



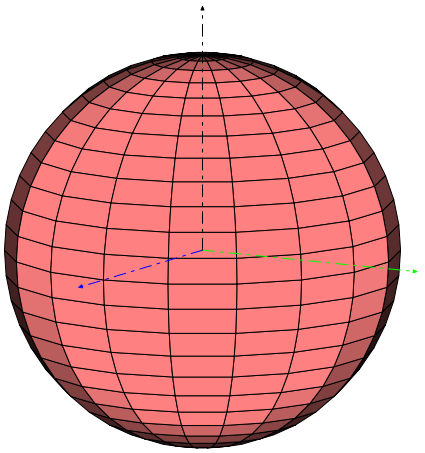
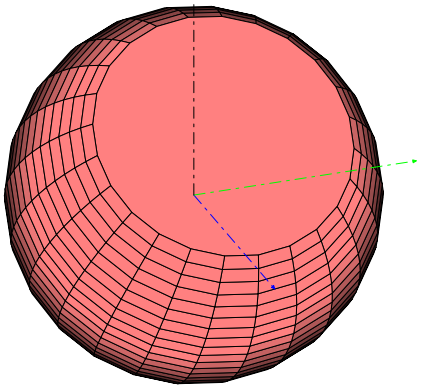
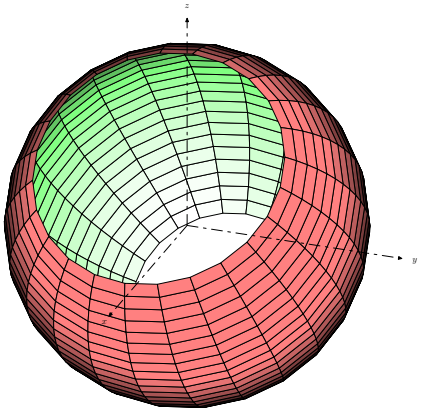
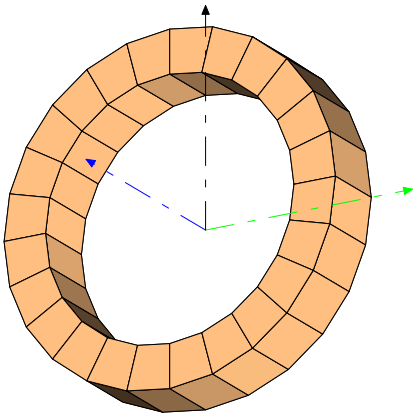
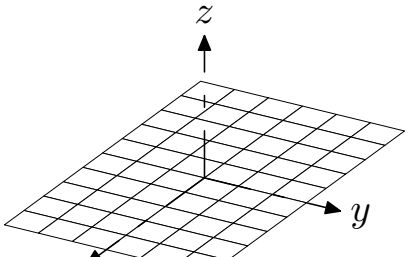
```
creux:=true;
%nb:=24;subh:=9;
Objettronccone1("r=2","h=4","H=5")
AffichageObjet1;
```

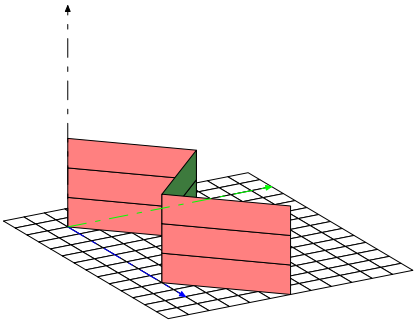
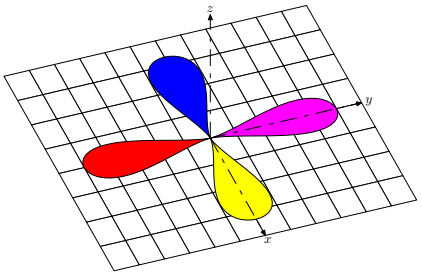
Tronc de cône plein

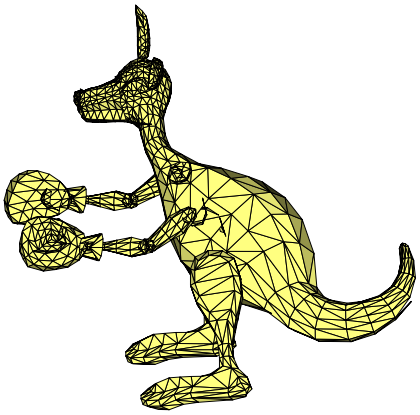
"r=1", "h=2", "H=5"

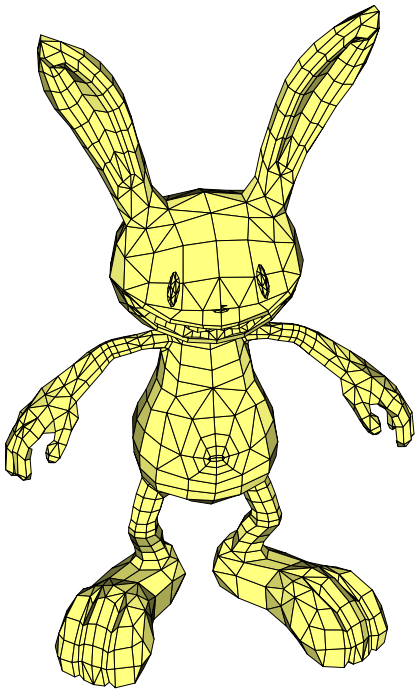


```
creux:=false;
%nb:=24;subh:=9;
Objettronccone1("r=2","h=2","H=3")
AffichageObjet1;
```

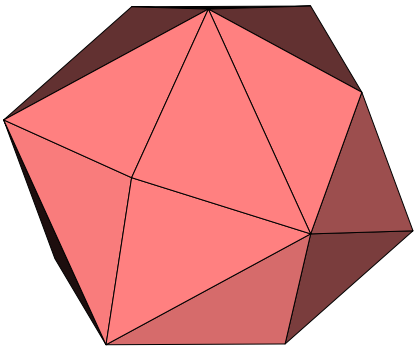
Sphère	"R=3"		%nb:=24;subh:=18; Objetsphere1("R=2"); AffichageObjet1;
Calotte sphérique pleine	"r=3", "phib=-pi/4", "phih=pi/6" (en radians)		%nb:=24;subh:=24; Objetcalotte1(% "R=4","phib=-pi/3","phih=pi/4"); AffichageObjet1;
Calotte sphérique creuse	"r=3", "phib=-pi/4", "phih=pi/6" (en radians)		%nb:=24;subh:=24; Objetcalotte1(% "R=4","phib=-pi/3","phih=pi/4"); AffichageObjet1;
Anneau à section rectangulaire	"R=3", "r=1", "h=2"		%subh:=24; Objetanneau1("R=4","r=3","h=1.5"); AffichageObjet1;
Grille dans le plan (xOy)	"am=-3", "an=2", "bm=-2", "bn=4"		%nb:=10;subh:=6; Objetgrille1("am=-2.5","an=2.5",% "bm=-1.5","bn=1.5"); AffichageObjet1;

Ruban d'axe (Oz)	"h=2", Liste des, sommets		%subh:=3; ObjetRuban2("h=2")((0,0,0),% (2,2,0),(4,0,0),(6,2,0)); AffichageObjet2;
Biface [2]	Liste des sommets		ObjetBiface1((5,0,0) for % t=0.06544 step 0.06544% until pi:% ,(5*(cos(t)**2),% (3*sin(t)*((cos(t))**3))% ,0) endfor); AffichageObjet1;

OFF	Nom du fichier		debut:=0;
			echelle:=15000;
			ObjetOFF1("Kangaroo.off");
			AffichageObjet1;

OBJ	Nom du fichier		echelle:=200; ObjetOFF1("Midpoly_03-1.obj"); AffichageObjet1;

New Liste des sommets, Liste des faces



```
%hexaedre tetrakis
ObjetNew1((0,0,-3*sqrt(6)/4),%
(-sqrt(3)/2,-3/2,-sqrt(6)/4),%
(sqrt(3),0,-sqrt(6)/4),%
(-sqrt(3)/2,3/2,-sqrt(6)/4),%
(sqrt(3)/2,-3/2,sqrt(6)/4),%
(-sqrt(3),0,sqrt(6)/4),%
(sqrt(3)/2,3/2,sqrt(6)/4),%
(0,0,3*sqrt(6)/4),%
(-3*sqrt(3)/4,0,-3*sqrt(6)/8),%
(3*sqrt(3)/8,-9/8,-3*sqrt(6)/8),%
(3*sqrt(3)/8,9/8,-3*sqrt(6)/8),%
(-3*sqrt(3)/8,-9/8,3*sqrt(6)/8),%
(-3*sqrt(3)/8,9/8,3*sqrt(6)/8),%
(3*sqrt(3)/4,0,3*sqrt(6)/8)%
)(3,0,1,8,%
3,0,8,3,%
3,3,8,5,%
3,1,5,8,%
3,0,9,1,%
3,0,2,9,%
3,2,4,9,%
3,1,9,4,%
3,0,3,10,%
3,0,10,2,%
3,2,10,6,%
3,3,6,10,%
3,2,13,4,%
3,2,6,13,%
3,6,7,13,%
3,4,13,7,%
3,3,5,12,%
3,3,12,6,%
3,6,12,7,%
3,5,7,12,%
3,1,11,5,%
3,1,4,11,%
3,4,7,11,%
3,5,11,7%
);
```

Pour beaucoup de ces objets, les `nb` et/ou `subh` sont requis et ont la possibilité d'être modifiés. Ils représentent le maillage sur chacun des objets. De manière générale, `nb` représente le maillage de « la base » du solide et `subh` représente le maillage « vertical » de l'objet.

Pour les objets possédant une version creuse, c'est le paramètre booléen `creux` qui fait la différence. Sa valeur par défaut est `false`.

2. Sur la figure proposée, il y a en fait quatre solides biface.

3.1.1 Compléments sur l'objet `ruban`

Le ruban est un paravent. Il s'agit simplement de positionner les sommets dans le plan (Oxy) et de donner la hauteur du paravent souhaité.
Il n'est pas toujours nécessaire de prendre une ligne « brisée » comme support de base pour le ruban.

Exemple 1 : Un paravent sinusoïdal

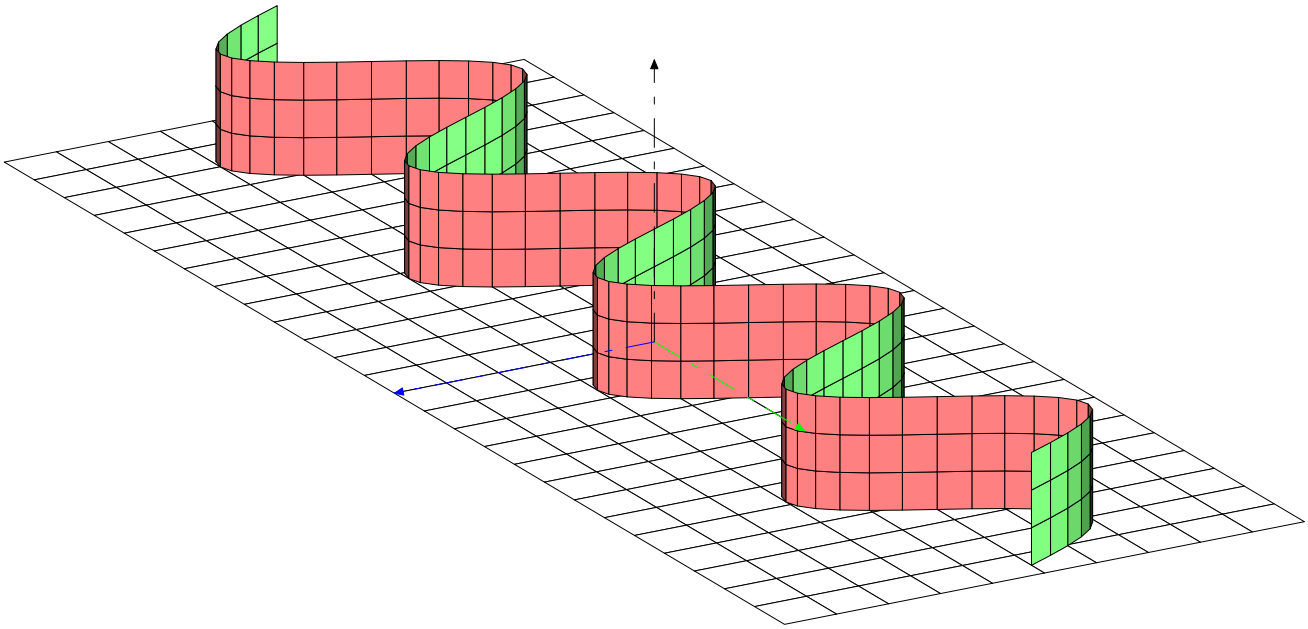


FIGURE 3.1 – Un paravent sinusoïdal

```
figureespace(-10u,-10u,10u,10u);
Initialisation(500,60,20,25);
nb:=10; subh:=26;
eclairage:=false;%<-pour obtenir la grille en blanc
outcolor:=blanc;
Objetgrille1("am=-5","an=5","bm=-13","bn=13");
AffichageObjet1;
eclairage:=true;
outcolor:=0.5[rouge,blanc]; incolor:=0.5[vert,blanc];
subh:=3;
ObjetRuban2("h=2")(for t=-4*pi step (pi/12) until 47*pi/12:(2*sin(t),t,0), endfor
    (0,4*pi,0));
AffichageObjet2;
TraceAxes;
finespace;
```

Exemple 2 : Un paravent d'amour

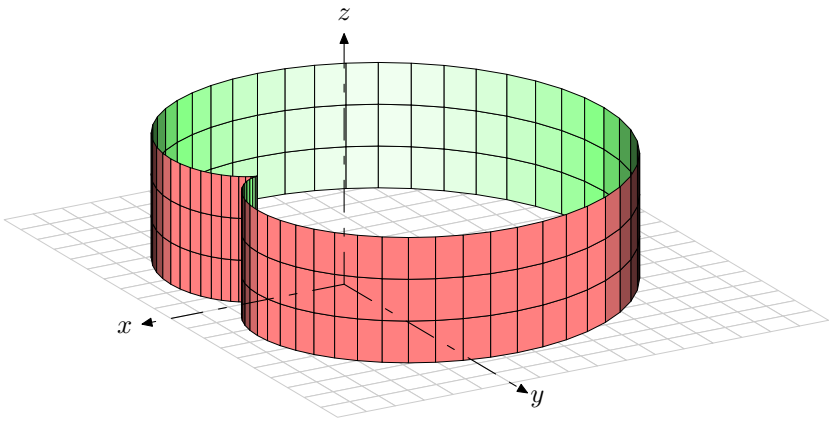


FIGURE 3.2 – Un paravent d'amour

```
figureespace(-10u,-10u,10u,10u);
Initialisation(500,60,20,25);
nb:=17; subh:=20;
eclairage:=false;
drawoptions(withcolor gris);
outcolor:=blanc;
Objetgrille1("am=-5.5","an=3","bm=-5","bn=5");
AffichageObjet1;
drawoptions();
eclairage:=true;
outcolor:=0.5[rouge,blanc]; incolor:=0.5[vert,blanc];
subh:=3;
ObjetRuban2("h=2")(for t=-pi step pi/36 until 35*pi/36:1.5*(2*cos(t)-cos(2*t),2*sin
    (t)-sin(2*t),0), endfor(-4.5,0,0));
AffichageObjet2;
TraceAxesD(3.5,5.5,4);
finespace;
```

3.1.2 Compléments sur l'objet `prisme`

Si l'on souhaite définir un prisme avec une base peu courante, on utilisera la macro `ObjetPrisme` (avec une majuscule) de la même façon que la macro `Objetprisme` en définissant l'axe, la hauteur et la liste des sommets formant la base.

Exemple 1 : Prisme droit à section carrée arrondie

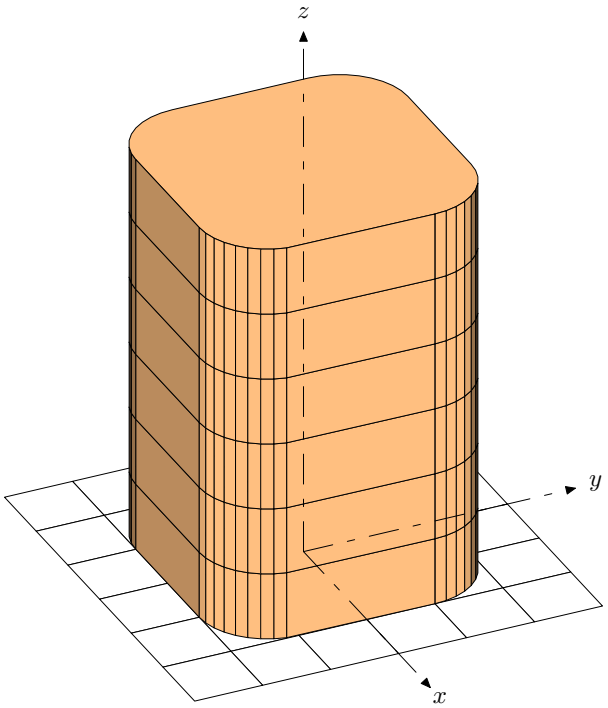
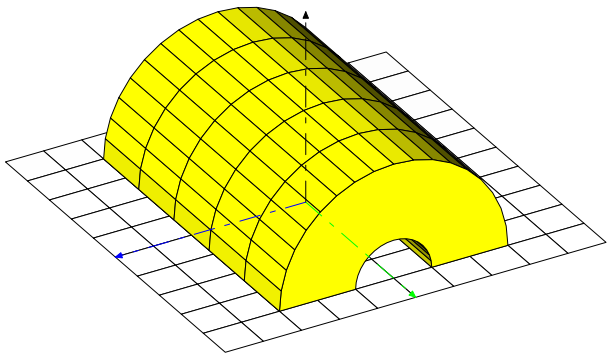


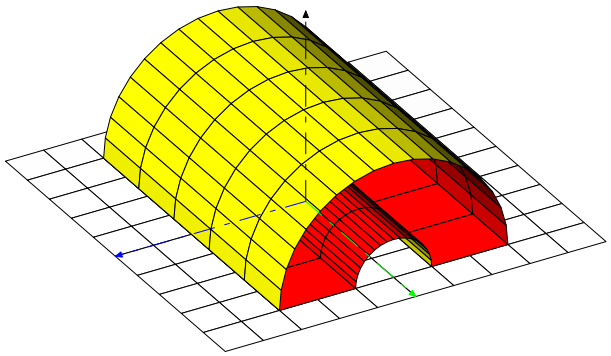
FIGURE 3.3 – Prisme droit à section carrée arrondie

```
outcolor:=0.5[orange,blanc];

figureespace(-10u,-10u,10u,10u);
Initialisation(1500,-25,30,20);
eclairage:=false;
nb:=4;subh:=4;
Objetgrille0("am=-2","an=2","bm=-2","bn=2");
AffichageObjet0;
ObjetPrisme1("axe=(0,0,1)","h=6")(for k=0
    step 10 until 90:(1+cosd(k),1+sind(k),0),
    endfor for k=90 step 10 until 180:(cosd(k)
    -1,1+sind(k),0), endfor for k=180 step 10
    until 270:(cosd(k)-1,sind(k)-1,0), endfor
    for k=270 step 10 until 350:(cosd(k)+1,
    sind(k)-1,0), endfor (2,1,0));
AffichageObjet1;
TraceAxesD(2,2,2);
finespace;
```

Exemple 2 : Demi-prisme droit à section couronne circulaire

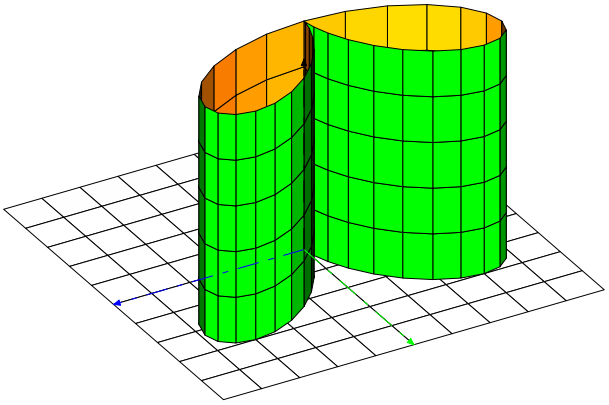


```
figureespace(-10u,-10u,10u,10u);
Initialisation(500,60,30,25);
outcolor:=blanc;
subh:=10;
Objetgrille1("am=-5","an=5","bm=-5","bn=5");
AffichageObjet1;
outcolor:=jaune; incolor:=rouge;
nb:=1; subh:=5;
angx:=90; TR:=(0,4,0);
ObjetPrisme2("axe=(0,0,1)","h=8",for k=0 step
     10 until 180:(3*cosd(k),3*sind(k),0),
    endfor for k=180 step -10 until 0:(cosd(k)
    ,sind(k),0), endfor(3,0,0));
AffichageObjet2;
TraceAxes;
finespace;
```



```
figureespace(-10u,-10u,10u,10u);
Initialisation(500,60,30,25);
outcolor:=blanc;
subh:=10;
Objetgrille1("am=-5","an=5","bm=-5","bn=5");
AffichageObjet1;
outcolor:=jaune; incolor:=rouge;
nb:=1; subh:=5;
angx:=90; TR:=(0,4,0);
creux:=true;%<-changement en solide creux
ObjetPrisme2("axe=(0,0,1)","h=8",for k=0 step
     10 until 180:(3*cosd(k),3*sind(k),0),
    endfor for k=180 step -10 until 0:(cosd(k)
    ,sind(k),0), endfor(3,0,0));
AffichageObjet2;
TraceAxes;
finespace;
```

Exemple 2 : Prisme droit creux dont la section est un bifolium régulier



```
figureespace(-10u,-10u,10u,10u);
Initialisation(500,60,30,25);
outcolor:=blanc;
subh:=10;
Objetgrille1("am=-5","an=5","bm=-5","bn=5");
AffichageObjet1;
outcolor:=vert; incolor:=orange;
nb:=1; subh:=5;
creux:=true;
ObjetPrisme2("axe=(0,0,1)","h=6",for k=0 step 5
    until 175:(12*sind(k)*(cosd(k)**3),12*(sind(k)
    **2)*(cosd(k)**2),0),endfor (0,0,0));
AffichageObjet2;
TraceAxes;
finespace;
```

3.1.3 Compléments sur les objets `cylindre` et `cone`

S'il l'on veut généraliser la notion de cylindre, la macro `Objetcylindre` ne suffit pas. En effet, il devient nécessaire de définir une courbe *directrice* et la direction de *l'axe du cylindre.*
À cet effet, la macro `ObjetCylindre` (*notez la majuscule*) a été défini.

`ObjetCylindre1(fn,umin,umax,vmin,vmax)`

Proposons un exemple. Définissons la courbe spatiale `fn` : `(3*(cos(u)**3),3*(sin(u)**3),-2+v)` de paramètre *obligatoire* `u` et `v`. C'est une astroïde tracé dans le plan $z = -2$. Les paramètres seront donc `umin=` π ; `umax=` $-\pi$, `vmin=0` et (par exemple) `vmax=4` pour aller jusqu'au plan $z = 2$.. Le code dont résulte la figure ci-dessus est donc

```
nb:=24; subh:=12;

figureespace(-10u,-10u,10u,10u);
```

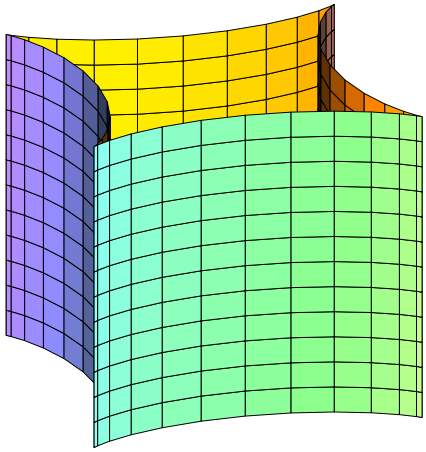
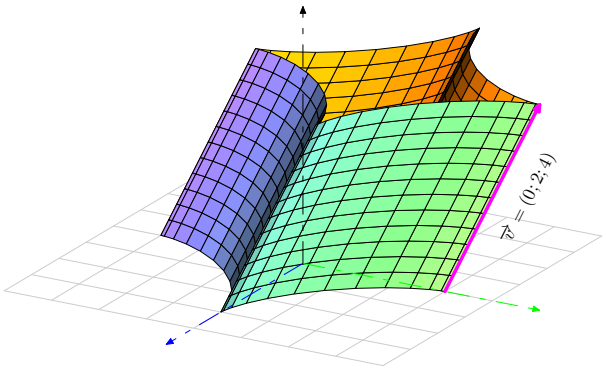


FIGURE 3.4 – Cylindre à base astroïdale et d'axe (0;0;1)

```
Initialisation(500,30,20,50);
arcenciel:=true;
incolor:=0.75[orange,blanc];
ObjetCylindre1("(3*(cos(u)**3),3*(sin(u)**3),-2+v)",pi,-pi,0,4);
AffichageObjet1;
finespace;
```

Si l'on souhaite avoir maintenant le même cylindre partant du plan $z = 0$ mais d'axe $(0;1;2)$, on codera alors

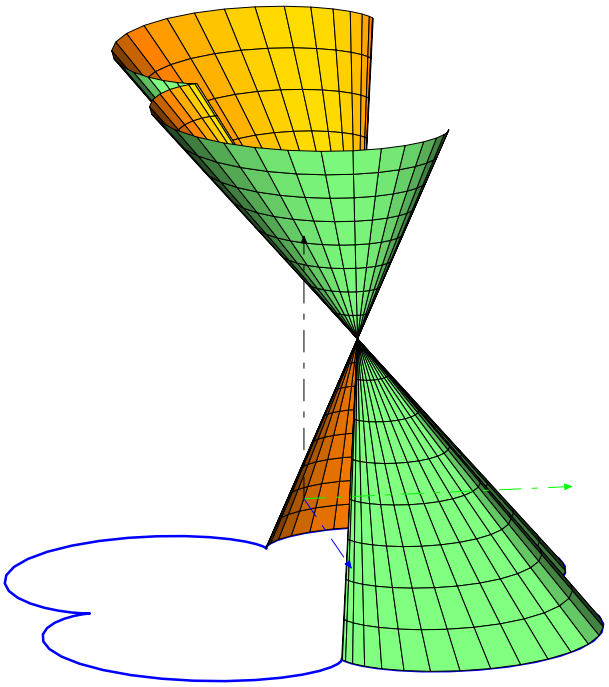



```
nb:=24; subh:=12;

figureespace(-10u,-10u,10u,10u);
Initialisation(500,30,20,50);
arcenciel:=true;
incolor:=0.75[orange,blanc];
ObjetCylindre1("(3*(cos(u)**3),3*(sin(u)**3)+v,2*v)
    ",pi,-pi,0,2);
AffichageObjet1;
finespace;
```

L'usage de la macro `ObjetCone` (toujours avec la majuscule) est le même pour la généralisation des cônes. Toutefois, il faut ici définir une origine et les deux nappes du cône seront symétriques par rapport à cette origine.

```
ObjetCone1(fn,umin,umax,zbas,"orig=...")
```

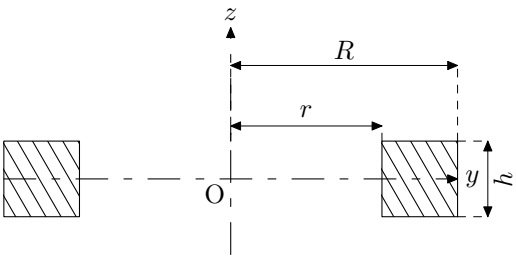
Il ne faut pas oublier de placer la courbe directrice dans le même plan que le plan de base du cône. Pour mieux exploiter cette fonctionnalité, voici l'exemple d'un cône dont la directrice est une branche d'épicycloïde et dont l'origine est le point $(0;1;3)$.

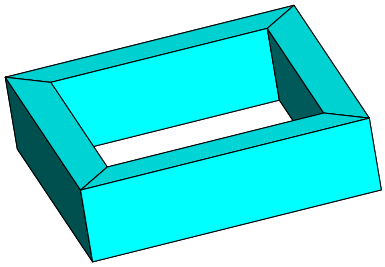


```
nb:=48;subh:=8;

figureespace(-10u,-10u,10u,10u);
Initialisation(500,-10,15,30);
outcolor:=0.5[vert,blanc];
incolor:=orange;
r:=1; q:=4;
draw Fonction("(r*((q+1)*cos(t)-cos((q+1)*t)),r*((q
    +1)*sin(t)-sin((q+1)*t)),0)",-pi,pi,0.0628)
    withpen pencircle scaled1.5bp withcolor bleu;
ObjetCone1("(r*((q+1)*cos(u)-cos((q+1)*u)),r*((q+1)
    *sin(u)-sin((q+1)*u)),0)",0,pi,-2,"orig=(0,1,3)
    ");
AffichageObjet1;
TraceAxes;
finespace;
```

3.1.4 Compléments sur les objets `anneau`

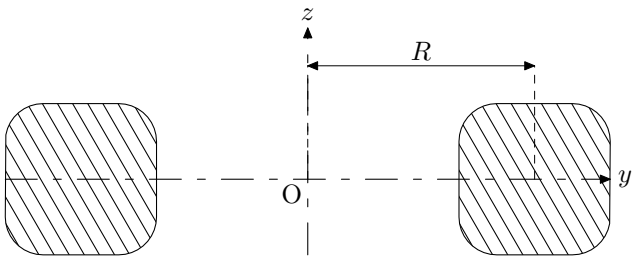
L'objet `Objetanneau` propose *par défaut* une section rectangulaire.





```
Objetannneau1("R=8","r=6","h=3");
```

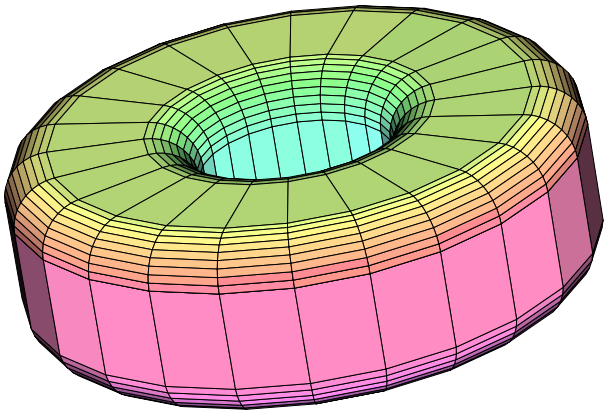
Pour utiliser une autre section telle que celle-ci



on utilisera

```
ObjetAnneau1("nbp=...",pp)
```

où `nbp` est le nombre de sommets de la section et `pp` un chemin METAPOST représentant la section souhaitée.

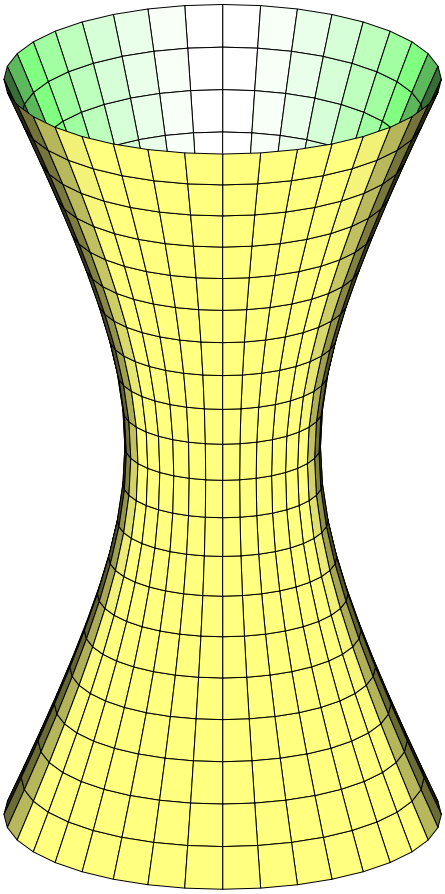


```
figureespace(-10u,-10u,10u,10u);
Initialisation(500,30,30,25);
arcenciel:=true;
incolor:=jaune;
R:=4;
subh:=24;
path pp;
pp=for k=0 step 10 until 90:(R+1+cosd(k),1+sind(k))
    -- endfor for k=90 step 10 until 180:(R+cosd(k)
    -1,1+sind(k))-- endfor for k=180 step 10 until
    270:(R+cosd(k)-1,sind(k)-1)-- endfor for k=270
    step 10 until 350:(R+cosd(k)+1,sind(k)-1)--
    endfor (R+2,1)--cycle;
ObjetAnneau1("nbp=42",pp);
AffichageObjet1;
finespace;
```

3.1.5 Compléments sur l'objet `new`

La déclaration des sommets se fait sous forme de triplets. La déclaration des faces se faisant quant à elle, sous la forme d'un $n+1$-uplet où n est le nombre de sommets composant la face ; le premier nombre étant d'ailleurs égal à n suivi des nombres représentant les sommets de cette face : `3 4 7 8` représente une face triangulaire (3) composée des sommets 4, 7 et 8 *donnés dans le sens trigonométrique si l'on regarde la face de l'extérieur.*

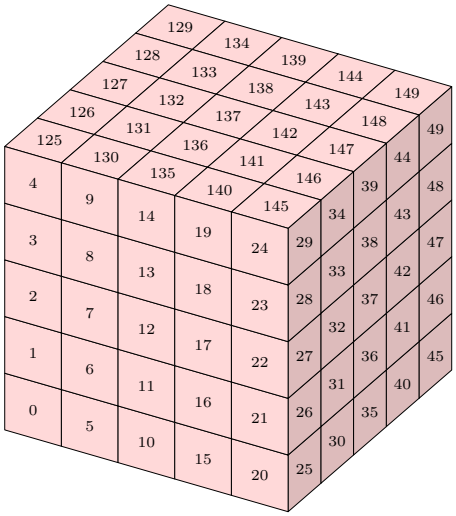
La déclaration des sommets et des faces peut aussi être plus complexes. Cet exemple, repris de la documentation de `pst-solides3d` montre comment on peut définir à l'aide de boucles la liste des sommets et la liste des faces.



```
Initialisation(500,30,20,25);
a=20;b=36;h=8;
zz=-h/2;
ObjetNew1((sqrt(1+((zz+h)**2)/4),0,zz+h)
  for m=a downto 0:
    for n=if m=a:b-1 else:b fi downto 1:
      ,(sqrt(1+((zz+h*m/a)**2)/4)*cosd(n*(360 div b
          )),sqrt(1+((zz+h*m/a)**2)/4)*sind(n*(360
          div b)),zz+h*m/a)
    endfor
  endfor
)(4,a*b,b+a*b,b-1+a*b,a*b-1
for m=a downto 1:
  for n=if m=a:m*b-1 else: m*b fi downto (m-1)*b+2:
    ,4,n,b+n,(b-1)+n,n-1
  endfor
  ,4,m*b+1,(m+1)*b,m*b,(m-1)*b+1
endfor
);
```

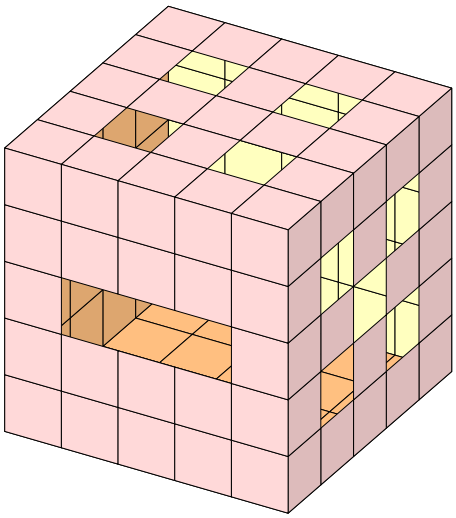
3.1.6 Numéroter et enlever des facettes

Lorsque le paramètre booléen `numeroteface` est positionné à `true`, on numérote *toutes* les faces *visibles* du solide. La police utilisée est `cmr5` ; on peut bien évidemment la modifier grâce à `defaultfont`.




```
defaultfont:="cmr7";
Initialisation(1500,30,20,20);
numeroteface:=true;%<-
subh:=5;
Objetcube1("a=2");
AffichageObjet1;
TraceAxesD(4,4,4);
```

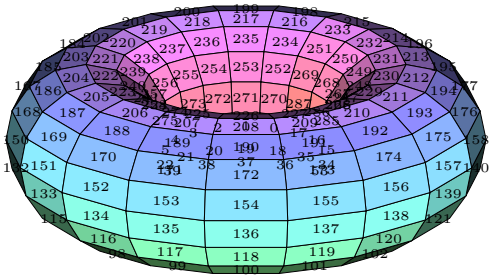
Une fois ceci fait, si le booléen `creux` est positionné à `true` alors on peut enlever des facettes pour voir ce qu'il se passe à l'intérieur de l'objet considéré. On définit alors l'objet puis on indique quelles sont les numéros des faces que l'on souhaite enlever grâce à `ObjetEnleve1(...)`



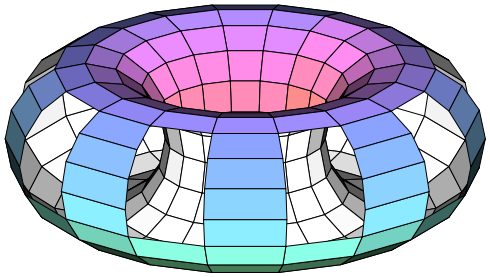
```
Initialisation(1500,30,30,50);
subh:=5;
creux:=true;%<-
Objetcube1("a=3");
ObjetEnleve1(7,12,17,31,33,37,41,43,131,133,141,143);
AffichageObjet1;
```

Les numéros des faces à enlever doivent être donnés dans l'ordre croissant.

Si l'on souhaite enlever beaucoup de faces, une boucle `for` est possible au prix d'un petit changement de syntaxe.



```
nb:=16; subh:=18;

figureespace(-10u,-10u,10u,10u);
Initialisation(1000,30,20,30);
numeroteface:=true;
arcenciel:=true;
incolor:=0.5[gris,white];
Objettore1("R=2","r=1");
AffichageObjet1;
finespace;
```



```
numeroteface:=false;

figureespace(-10u,-10u,10u,10u);
Initialisation(1000,30,20,30);
creux:=true;
arcenciel:=true;
incolor:=0.5[gris,white];
Objettore1("R=2","r=1");
string Face;
Face=""&for k=133 step 2 until 193:
    decimal(k)&","&""& endfor decimal
    (195)&"";
ObjetEnleve1(scantokens Face);
AffichageObjet1;
finespace;
```

3.1.7 Transparence

Elle est implantée grâce au booléen `Transparence` positionné à `false` par défaut. La couleur de transparence choisie est `fillcolor` positionnée à gray par défaut.

C'est une notion gourmande en ressources. Aussi, il est plutôt conseillé de l'utiliser avec des objets *simples*.

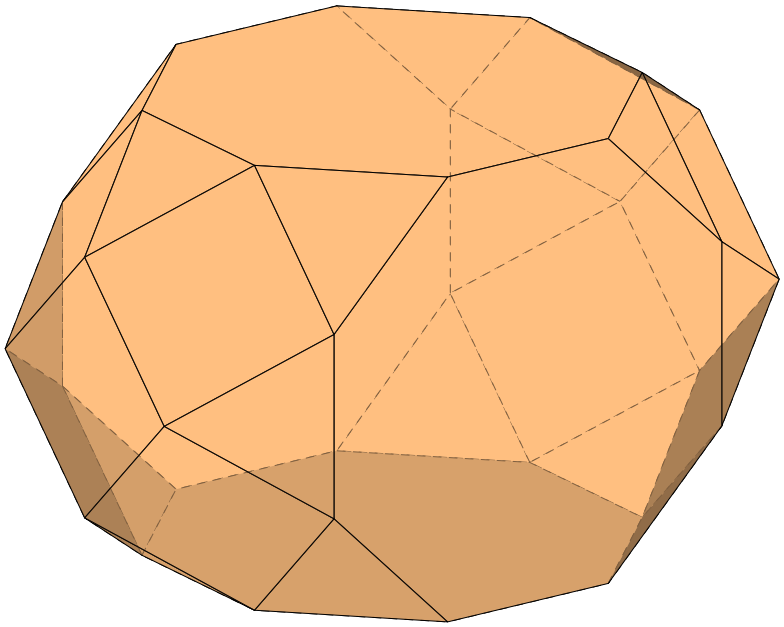


FIGURE 3.5 – Polygone de Johnson : J67

3.2 Et pour les bouger ?

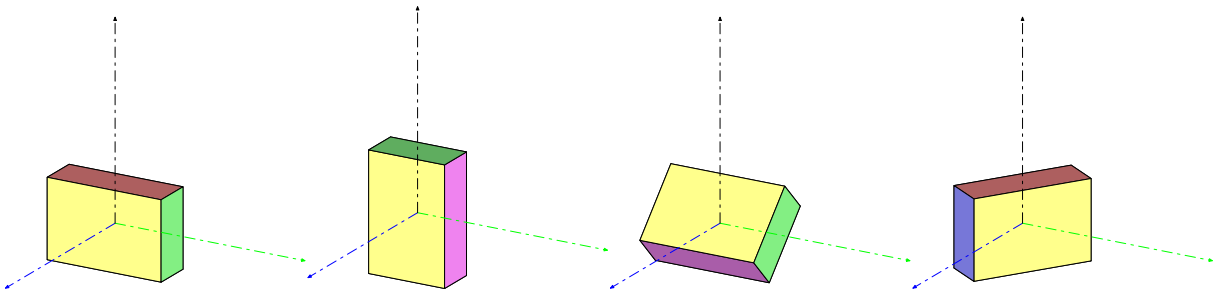
On dispose (et certains codes de la partie précédente l'ont montré) :
– des rotations autour des axes (Ox), (Oy) et (Oz) ;
– des translations ;
– de transformations définies par l'utilisateur.

3.2.1 Les rotations et translations

L'ordre choisi pour les utiliser est la rotation d'axe (Ox) ; puis celle d'axe (Oy) ; puis celle d'axe (Oz) et enfin la translation. Les rotations sont définis par trois angles `angx`, `angy` et `angz`.
Pour les translations, le paramètre est `TR` de type `color`.

Si un fichier METAPOST comporte plusieurs figures, n'oubliez pas de remettre les paramètres angulaires et de translation à leur valeur d'origine. Sinon, vous pourriez bien vous tirer les cheveux pour savoir où se situe votre erreur[3]...

3. Croyez-moi par expérience ;-)



(a) Positioninitiale (b) Position `angx:=90` (c) Position `angy:=-45` (d) Position `angz:=57`

FIGURE 3.6 – Utilisation des rotations.

3.2.2 Les transformations propres à l'utilisateur

On dispose d'un booléen `transformation` (positionné à `false` par défaut) pour indiquer la transformation nécessaire des sommets de l'objet considéré.
La transformation proprement dite, quant à elle, doit être définie par une macro Transform (attention à la majuscule).

Facteurs d'échelle

$$\begin{cases} x' = 1,25x \\ y' = 0,75y \\ z = 0,5z \end{cases}$$

```
1  vardef Transform(expr PT)=
2    save $;
3    color $;
4    Xpart($)=1.25*Xpart(PT);
5    Ypart($)=0.75*Ypart(PT);
6    Zpart($)=0.5*Zpart(PT);
7    $
8  enddef;
9
10 outcolor:=jaune; incolor:=0.5[vert,blanc];
11
12 figureespace(-10u,-10u,10u,10u);
13 Initialisation(500,50,20,15);
14 nb:=24; subh:=36;
15 outcolor:=0.5[rouge,blanc];
16 Objettore1("R=4","r=2");
17 AffichageObjet1;
18 TraceAxes;
19 finespace;
20
21 figureespace(-10u,-10u,10u,10u);
22 Initialisation(500,50,20,15);
23 nb:=24; subh:=36;
24 transformation:=true;%<-Pour appliquer la transformation choisie.
25 outcolor:=0.5[rouge,blanc];
26 Objettore1("R=4","r=2");
27 AffichageObjet1;
```

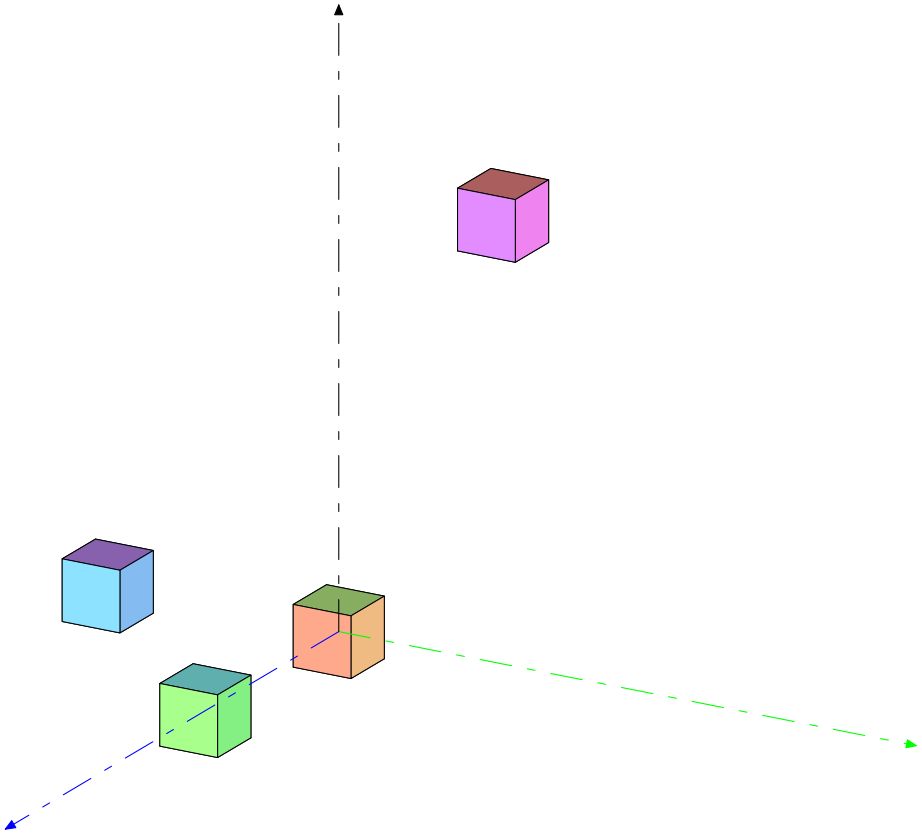
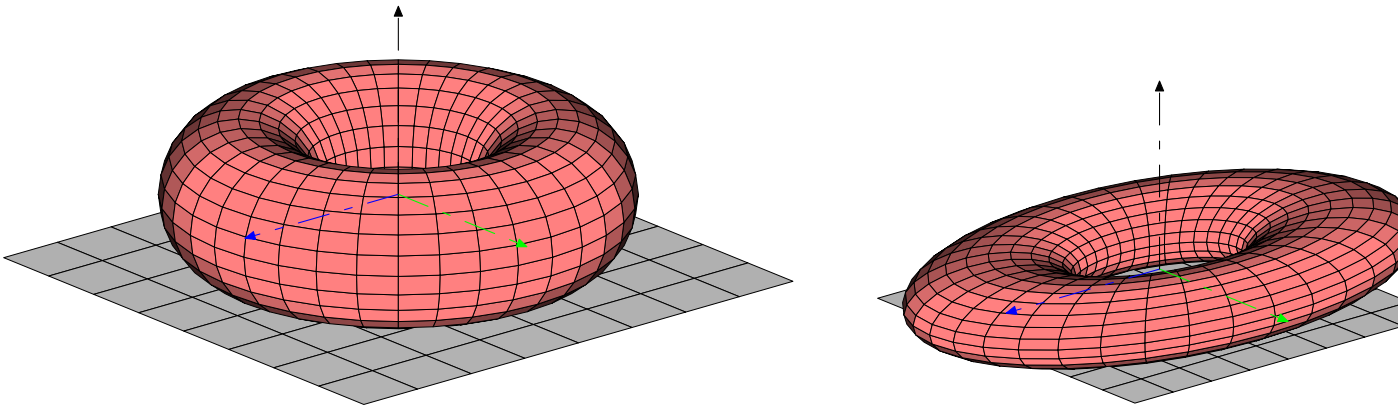


FIGURE 3.7 – Utilisation des translations.

```
28 TraceAxes;
29 finespace;
30 end
```



Transformation liée à la distance du point à l'origine

$$\begin{cases} x' = (0,5\sqrt{x^2+y^2+z^2}+1-0,5\sqrt{3})x \\ y' = (0,5\sqrt{x^2+y^2+z^2}+1-0,5\sqrt{3})y \\ z' = (0,5\sqrt{x^2+y^2+z^2}+1-0,5\sqrt{3})z \end{cases}$$

```
vardef Transform(expr PT)=
  save $; color $;
  Xpart($)=Xpart(PT)*(0.5*Norm(PT)+1-0.5*sqrt(3));
  Ypart($)=Ypart(PT)*(0.5*Norm(PT)+1-0.5*sqrt(3));
  Zpart($)=Zpart(PT)*(0.5*Norm(PT)+1-0.5*sqrt(3));
  $
enddef;

arcenciel:=true;

figureespace(-6u,-6u,6u,6u);
fill feuillet;
Initialisation(500,60,20,50);
subh:=9;
transformation:=true;
Objetcube1("a=3");
AffichageObjet1;
finespace;
```

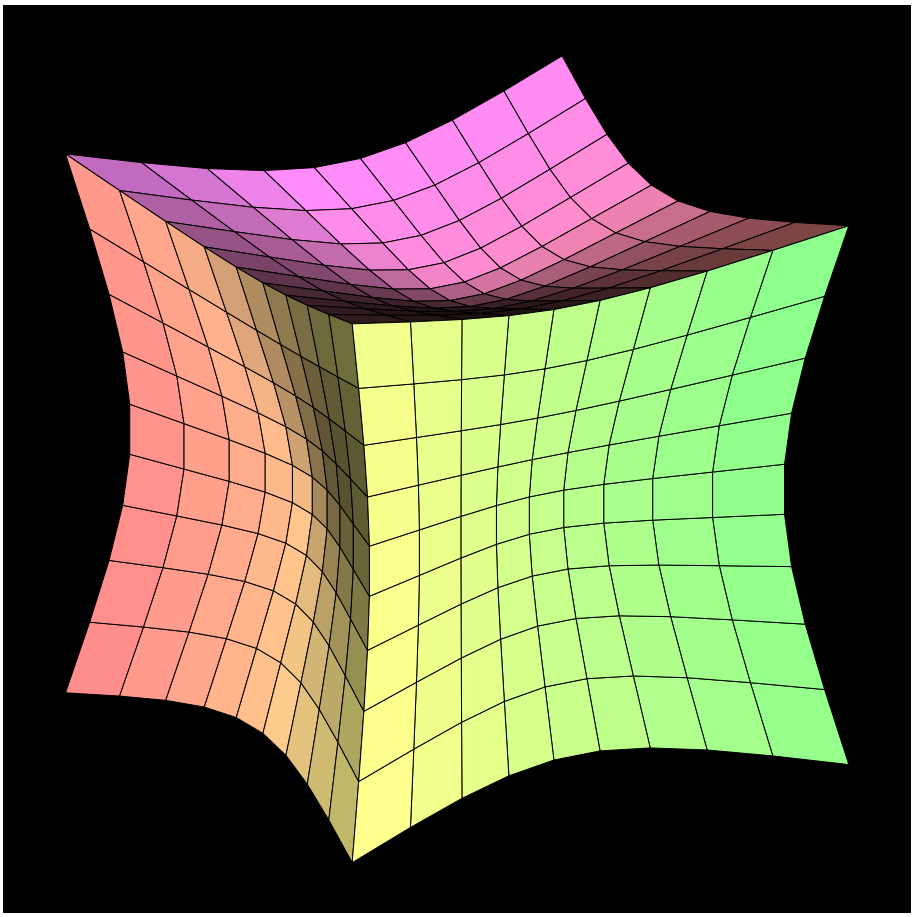
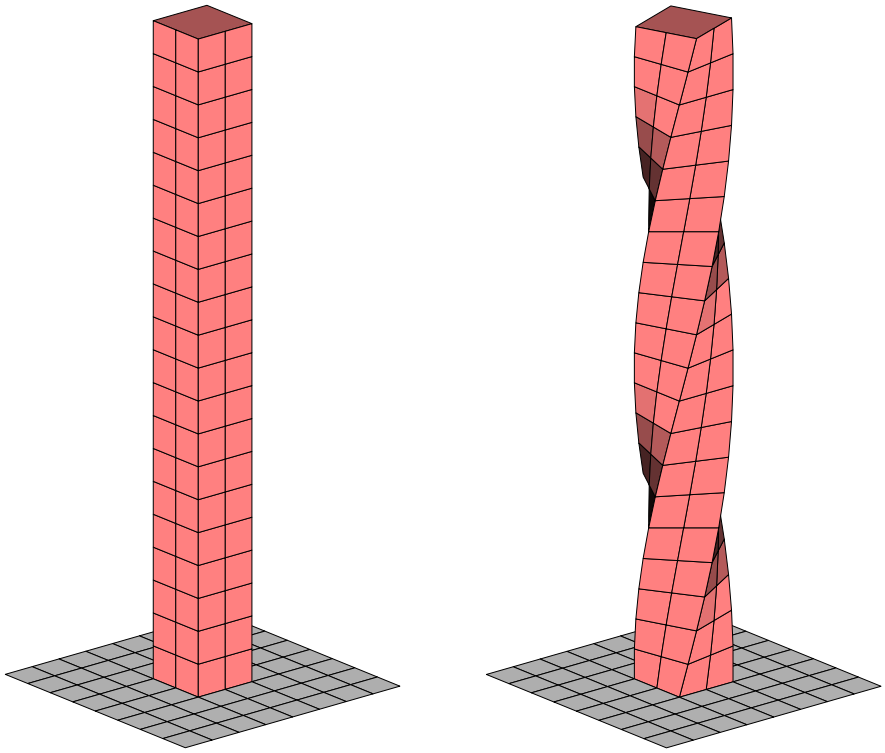


FIGURE 3.8 – Déformation d'un cube

Torsion d'une poutre

$$M' = \text{Rotation}(M, (Oz), 10° * z_M)$$

appliqué à un prisme de hauteur 10.



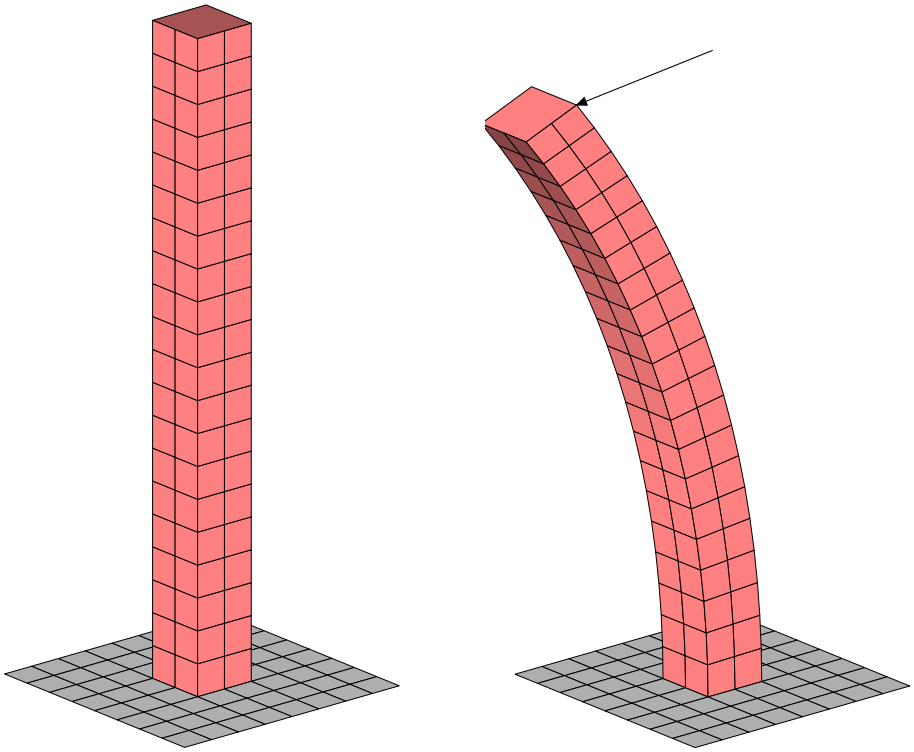
(a) Avant torsion (b) Après torsion

FIGURE 3.9 – Torsion d'une poutre

```
vardef Transform(expr PT)=%pour la torsion.
  save $; color $;
  angz:=subh*(Zpart(PT));
  $=RotXYZ(PT);
  $
enddef;

unit:=0.75;

figureespace(-2u,-2u,2u,10u);
Initialisation(500,50,20,35);
outcolor:=blanc;
subh:=8;
Objetgrille1("am=-2","an=2","bm=-2","bn=2");
nb:=2; subh:=20;
outcolor:=0.5[rouge,blanc];
Objetprisme2("axe=(0,0,1)","h=10")((0.5,-0.5,0),(0.5,0.5,0),(-0.5,0.5,0)
    ,(-0.5,-0.5,0));
nbobj:=2;
DessineFusion;
```

```
finespace;

figureespace(-2u,-2u,2u,10u);
Initialisation(500,50,20,25);
outcolor:=blanc;
subh:=8;
Objetgrille1("am=-2","an=2","bm=-2","bn=2");
nb:=2; subh:=20;
transformation:=true;
outcolor:=0.5[rouge,blanc];
Objetprisme2("axe=(0,0,1)","h=10")((0.5,-0.5,0),(0.5,0.5,0),(-0.5,0.5,0)
    ,(-0.5,-0.5,0));
nbobj:=2;
DessineFusion;
finespace;
```

Flexion d'une poutre encastrée On exerce une force au sommet de la poutre dans la direction (Ox). La rotation se fait alors suivant l'axe (Oy) d'un angle proportionnel à la force exercée et à la distance au sol.



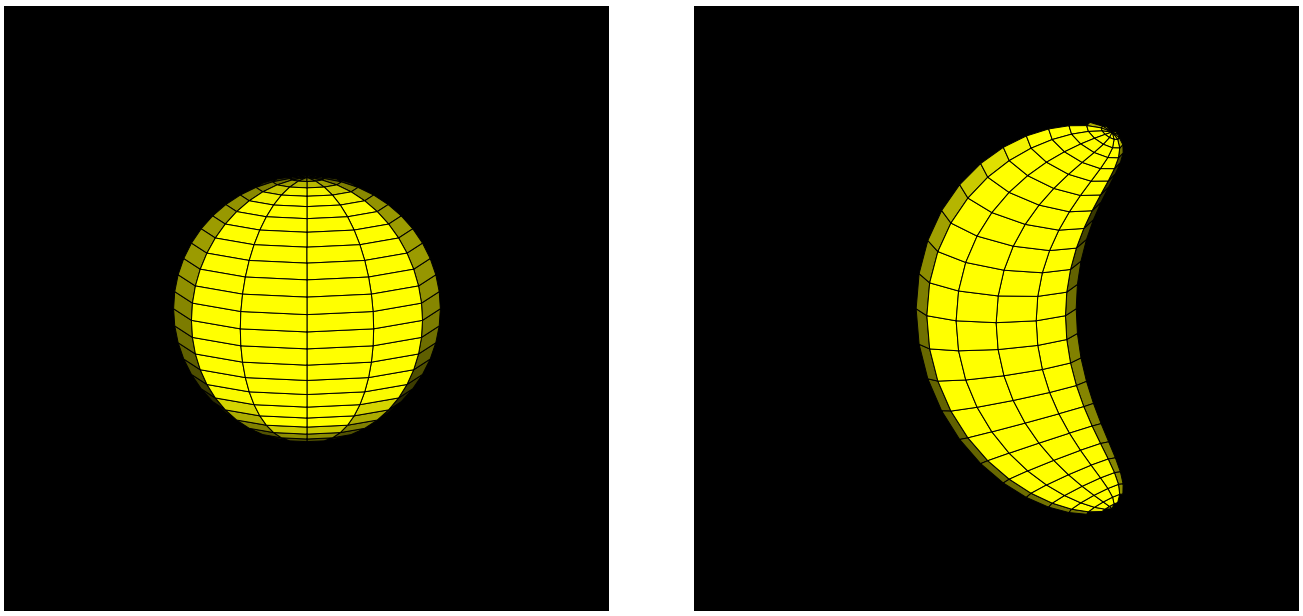
(a) Avant flexion (b) Après flexion

FIGURE 3.10 – Flexion d'une poutre encastrée

```
vardef Transform(expr PT)=%pour la flexion.
  save $; color $;
  angy:=2*(Zpart(PT));
  $=RotXYZ(PT);
```

```
  $
enddef;

unit:=0.75;

figureespace(-3u,-3u,3u,10u);
Initialisation(500,50,20,35);
outcolor:=blanc;
subh:=8;
Objetgrille1("am=-2","an=2","bm=-2","bn=2");
nb:=2; subh:=20;
transformation:=true;
outcolor:=0.5[rouge,blanc];
ObjetPrisme2("axe=(0,0,1)","h=10")((0.5,-0.5,0),(0.5,0.5,0),(-0.5,0.5,0)
    ,(-0.5,-0.5,0));
nbobj:=2;
DessineFusion;
finespace;
```

Voici enfin l'exemple de la banane ! Il m'a été demandé pour un cours de topologie. Je n'y comprends rien :) mais cela montre *l'enchaînement* de deux transformations :
– la première est appliquée à l'objet `sphere` ;
– la deuxième est aplliquée à l'objet résultant de la première transformation.



(a) Avant les transformations

(b) Après les transformations

FIGURE 3.11 – Transformation d'une sphère en « banane »

```
vardef Transform(expr PT)=
save $ ; color $ ;
Xpart($)=0.6*Xpart(PT);
Ypart($)=0.6*Ypart(PT);
Zpart($)=1.75*Zpart(PT);
```

```
  $
enddef ;

figureespace(-7u,-7u,7u,7u);
fill feuillet;
Initialisation(1500,-60,10,50);
nb:=12;
subh:=12;
outcolor:=jaune;
Objetsphere1("R=2");

transformation:=true;
ObjetDeplacement2(1);

vardef Transform(expr PT)=
    save $ ; color $ ;
  angy:=10*(Zpart(PT));
  $=RotXYZ(PT);
    $
enddef ;

outcolor:=jaune;
ObjetDeplacement3(2);
AffichageObjet3;
```

3.3 Ces objets, on peut les couper ?

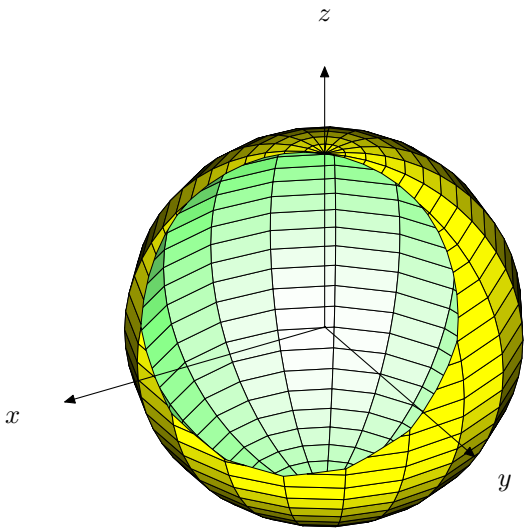
Il n'y a que la section d'un objet par un plan qui est implanté. Et même avec cela, les `arithmetic overflow` sont assez courants. On peut avoir également des erreurs de précisions qui entraînent certaines représentations non conformes.
Malgré ces limitations, METAPOST est capable de nous procurer de bien jolies figures. On peut l'aider en modifiant les paramètres `nb` et `subh` pour permettre de lever ces *petits* soucis.

On va donc pouvoir couper les différents objets par un plan. Les objets seront *nécessairement* creux. Pour cela, on va définir un plan par la macro `ObjetPlan` qui accepte comme paramètres trois points formant le plan. Définir un plan par une équation cartésienne ou par un point et un vecteur normal n'est pas implanté. On écrira donc

```
  ObjetPlan1("An=(0,0,0)","Bn=(1,2,0.25)","Cn=(4,-2,1)");
```

pour définir le plan d'équation $x = 4z$. Une fois le plan défini, on utilise la macro `ObjetSepare2(5,7)` où 2 est le numéro de l'objet à couper ; 5 et 7 étant les numéros des objets résultants de la coupe : 7 pour la partie « haute » 5 pour la partie « base ».

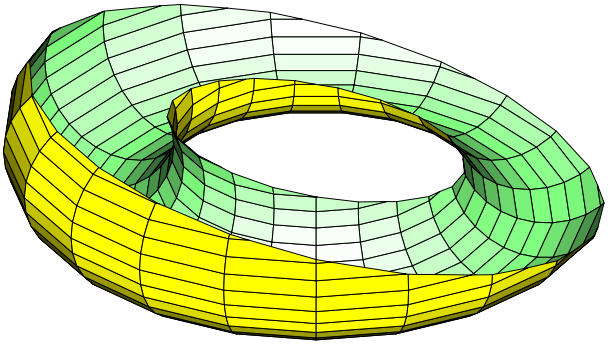
Dans l'exemple ci-dessous, on a coupé une sphère de centre O et de rayon 1 par le plan $x + y + z = 1$.




```
figureespace(-100u,-100u,100u,100u);
Initialisation(2500,60,30,75);
creux:=true;
outcolor:=jaune;
incolor:=0.5[vert,white];
nb:=30;
subh:=19;
Objetsphere15("R=1");
Objetplan21("An=(1,0,0)","Bn=(0,1,0)",%
"Cn=(0,0,1)");
ObjetSepare15(3,4);
AffichageObjet3;
TraceAxesD(2,2,2);
finespace;
```

FIGURE 3.12 – Section d'une sphère.

Voici la section d'un tore par le plan d'équation $x = 4z$.



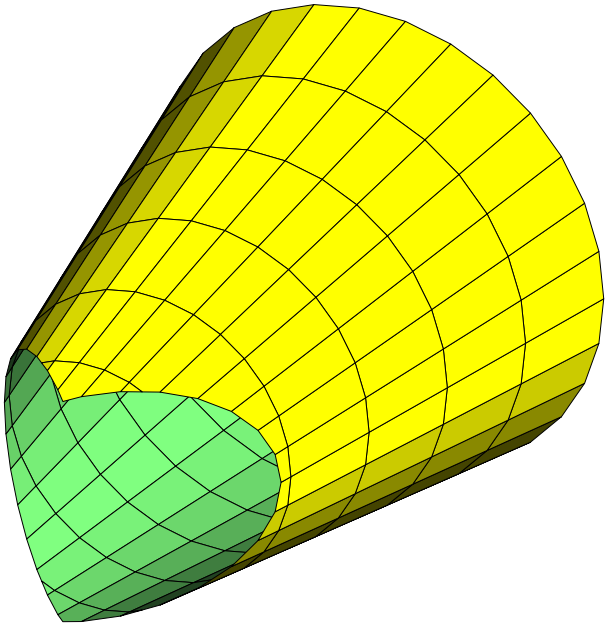
```
figureespace(-100u,-100u,100u,100u);
Initialisation(2500,60,30,75);
creux:=true;
outcolor:=jaune;
incolor:=0.5[vert,white];
nb:=30;
subh:=20;
Objettore15("R=2","r=0.75");
Objetplan21("An=(1,0,0.25)","Bn=(2,1,0.5)","
    Cn=(-1,0,-0.25)");
ObjetSepare15(3,4);
AffichageObjet3;
finespace;
```

FIGURE 3.13 – Section d'un tore.

Un autre exemple (voir figure 3.14) où on coupe un cube d'arête 2 centré sur O par le même plan d'équation $x = 4z$ que ci-dessus.

```
figureespace(-100u,-100u,100u,100u);
Initialisation(2500,60,30,75);
creux:=true;
outcolor:=jaune;
incolor:=0.5[vert,white];
nb:=30;
subh:=5;
Objetcube15("a=2");
Objetplan21("An=(1,0,0.25)","Bn=(0,1,0)","Cn=(4,0,1)");
ObjetSepare15(3,4);
AffichageObjet4;
TraceAxesD(2,2,2);
finespace;
```

On peut également enchaîner les coupes [4].



```
figureespace(-100u,-100u,100u,100u);
Initialisation(2500,-60,30,75);
creux:=true;
outcolor:=jaune;
incolor:=0.5[vert,white];
nb:=30; subh:=11;
angx:=90; TR:=(0,5,0);
Objetcone15("r=2","h=10");
angx:=0; TR:=(0,0,0);
Objetplan21("An=(1,-1,0)","Bn=(2,-2,1)","Cn
    =(-2,2,1)");
ObjetSepare15(1,2);
Objetplan22("An=(1,1,0)","Bn=(2,2,1)","Cn
    =(-2,-2,1)");
ObjetSepare1(3,4);
AffichageObjet3;
finespace;
```

FIGURE 3.15 – Double section d'un cône

3.4 La fusion ? C'est parti !

Une fois que l'on dispose de tous ces objets, on peut les fusionner pour créer une scène. La méthode mise en place est la suivante :

– On définit les objets les uns après les autres en leur attribuant, si besoin est, les couleurs `outcolor` et `incolor` ;
– puis on fusionne tous ces objets. Simple, non ?

Prenons un exemple :

4. Mais là, je ne garantis pas les résultats.

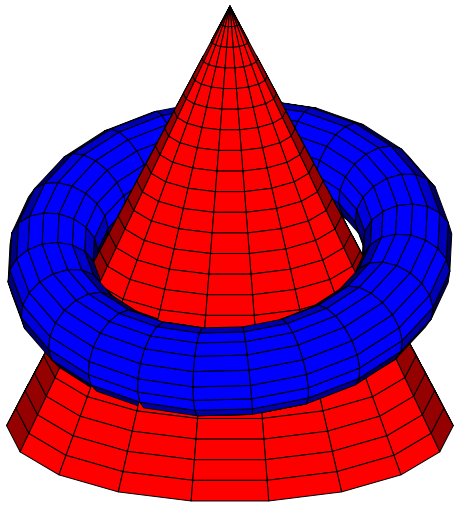


FIGURE 3.16 – Un tore et un cône

```
%5''
figureespace(-10u,-10u,10u,10u);
Initialisation(500,10,20,50);
outcolor:=rouge;
nb:=18; subh:=24;
Objetcone1("r=2.5","h=5");
outcolor:=bleu;
angy:=20; TR:=(0,0,2);
Objettore2("R=2","r=0.5");
nbobj:=2;
DessineFusion;
finespace;
```

Convaincu de la simplicité ? Non ? Alors en voici un autre :

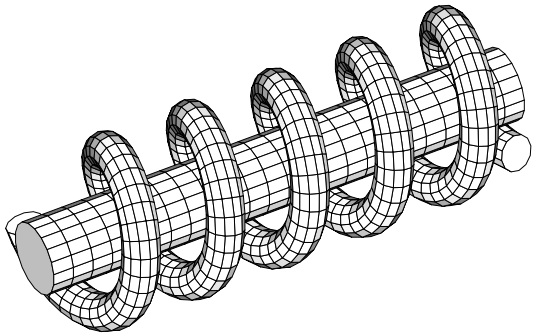
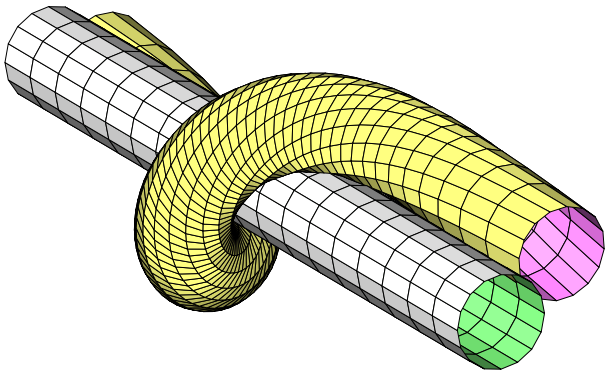


FIGURE 3.17 – Une hélice entourant un cylindre

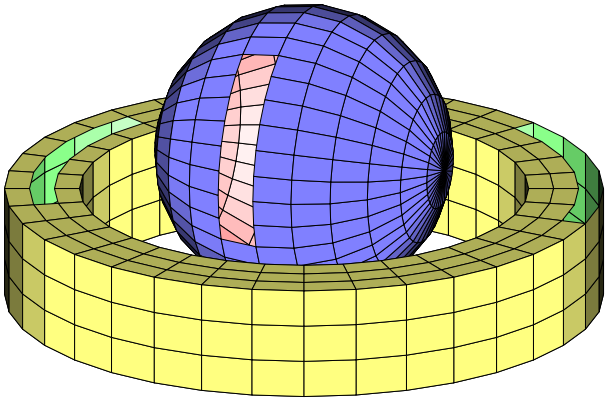
```
%30''
figureespace(-10u,-10u,10u,10u);
Initialisation(500,60,40,50);
angy:=90; TR:=(-1,0,0);
nb:=18; subh:=24;
outcolor:=blanc;
Objetcylindre1("r=0.5","h=7");
TR:=(0,0,0);
nb:=12;
ObjetTube2("(sin(t),cos(t),t/5)","(cos(t),-sin(t)
    ,1/5)",0.25,-5,165,0.2);
nbobj:=2;
DessineFusion;
finespace;
```

La fusion se fait donc par l'usage de la macro `DessineFusion` qui fusionne tous les objets créés précédemment. Elle dépend de `nbobj` qui lui indique le nombre d'objets à fusionner. La numérotation des objets prend ici tout son sens. C'est ce qui permet à la macro de retrouver ses petits...



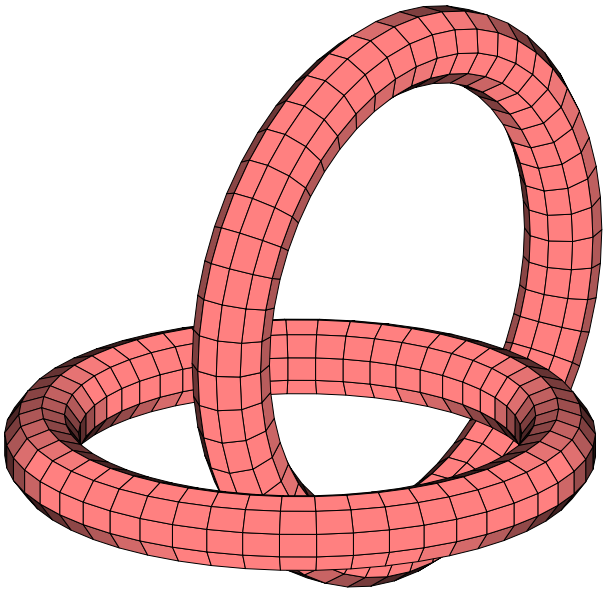
```
figureespace(-20u,-20u,20u,20u);
Initialisation(500,60,20,30);
creux:=true;
nb:=16; subh:=18;
outcolor:=blanc;
incolor:=0.5[vert,white];
angx:=90; TR:=(2,9,0);
Objetcylindre1("r=1","h=18");
outcolor:=0.5[jaune,blanc];
incolor:=0.5[violet,blanc];
angx:=0; TR:=(0,0,0);
ObjetTube2("(2*(1+cos(t)),2*tan(t/2),2*sin(t))","
    (-2*sin(t),2/((cos(t/2))**2),2*cos(t))"
    ,1,-2.7468,71,0.0763);
nbobj:=2;
DessineFusion;
finespace;
```

Un autre exemple avec des solides creusés manuellement.



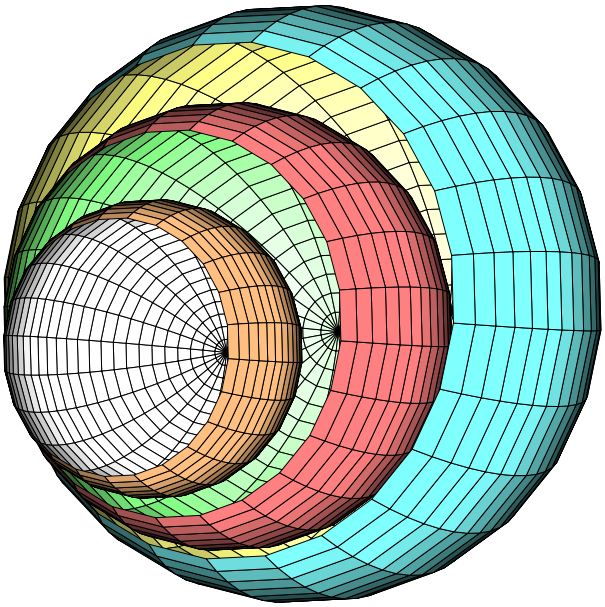
```
nb:=12; subh:=36;

figureespace(-10u,-10u,10u,10u);
Initialisation(1500,20,20,50);
creux:=true;
outcolor:=0.5[jaune,white];
incolor:=0.5[vert,white];
Objetanneau1("R=4","r=3","h=1.5");
ObjetEnleve1(135,136,137,138,%
139,140,150,151,152,153,154,155);
angx:=90;
TR:=(0,0,2);
outcolor:=0.5[bleu,white];
incolor:=0.5[rouge,white];
Objetsphere2("R=2");
ObjetEnleve2(232,233,234,235,%
236,237,238,239);
nbobj:=2;
DessineFusion;
finespace;
```

3.4.1 Deux anneaux

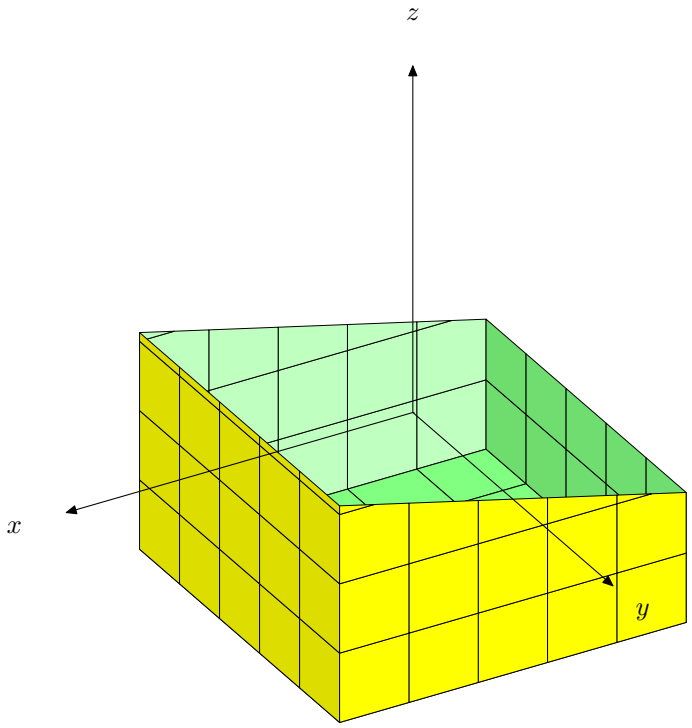


```
outcolor:=0.5[rouge,blanc];

figureespace(-10u,-10u,10u,10u);
Initialisation(500,40,20,100);
nb:=10;
ObjetTube1("(cos(t),sin(t),0.25)","(-sin(t),cos(t)
    ,0)",0.15,0,50,0.12566);
angx:=90; TR:=(-1,-0.1,0.5);
ObjetTube2("(cos(t),sin(t),0.25)","(-sin(t),cos(t)
    ,0)",0.15,0,50,0.12566);
nbobj=2;
DessineFusion;
finespace;
```

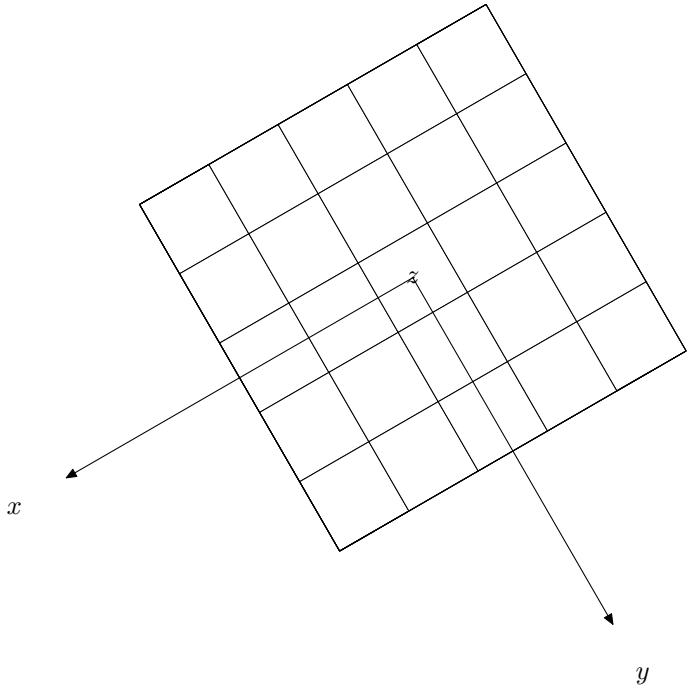
3.4.2 Trois calottes



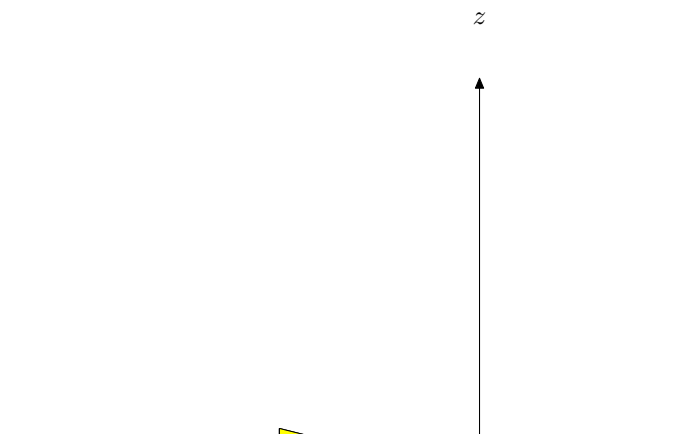
```
outcolor:=0.5[ciel,blanc]; incolor:=0.5[jaune,blanc
    ];

nb:=24;

figureespace(-10u,-10u,10u,10u);
Initialisation(500,30,10,50);
angy:=80;
subh:=24;
creux:=true;
Objetcalotte1("R=2","phib=-pi/2","phih=pi/6");
outcolor:=0.5[rouge,blanc];
incolor:=0.5[vert,blanc];
TR:=(1,0,0);
Objetcalotte2("R=1.5","phib=-pi/2","phih=pi/6");
outcolor:=0.5[orange,blanc];
incolor:=0.5[gris,blanc];
TR:=(2,0,0);
Objetcalotte3("R=1","phib=-pi/2","phih=pi/6");
nbobj:=3;
DessineFusion;
finespace;
```



(a) Section d'un cube



(b) en vue de dessus



Chapitre 4

pst-solides3d

Bien qu'il y ait de nombreux exemples empruntés à pst-solides3d dans les parties précédentes, cette partie constitue un hommage au travail des auteurs de ce package. C'est ma façon de les remercier pour les conseils, les encouragements et les approfondissements qu'ils m'ont apportés.
C'est également ici que l'on trouvera certaines fonctionnalités avancées *non documentées* et/ou *non implantées.*

Exemple 1

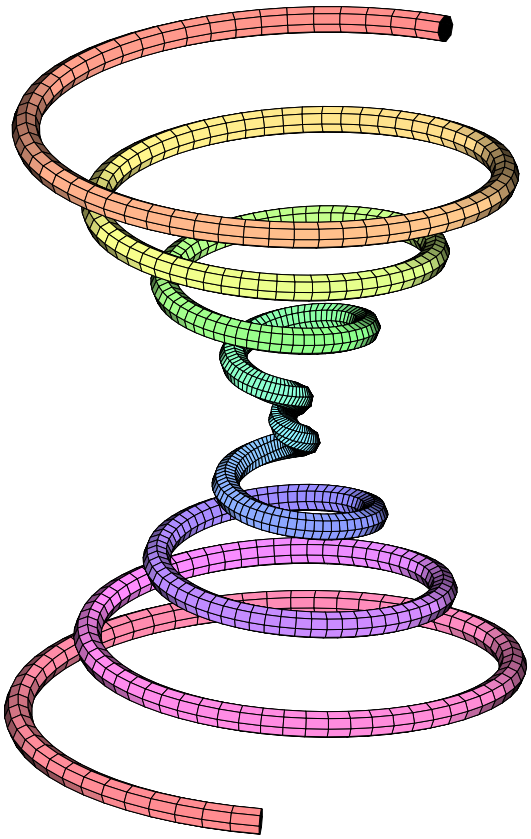


FIGURE 4.1 – Une spirale conique

```
%1'9"
figureespace(-10u,-10u,10u,10u);
Initialisation(200,20,20,15);
arcenciel:=true;
incolor:=gris;
draw Tuben("(0.5*t*sin(0.707)*cos(t),0.5*sin(0.707)
    *t*sin(t),-0.5*t*cos(0.707))","(0.5*sin(0.707)
    *(cos(t)-t*sin(t)),0.5*sin(0.707)*(sin(t)+t*cos
    (t)),-0.5*cos(0.707))",0.4,-25.4,508,0.1);
finespace;
```

Exemple 2

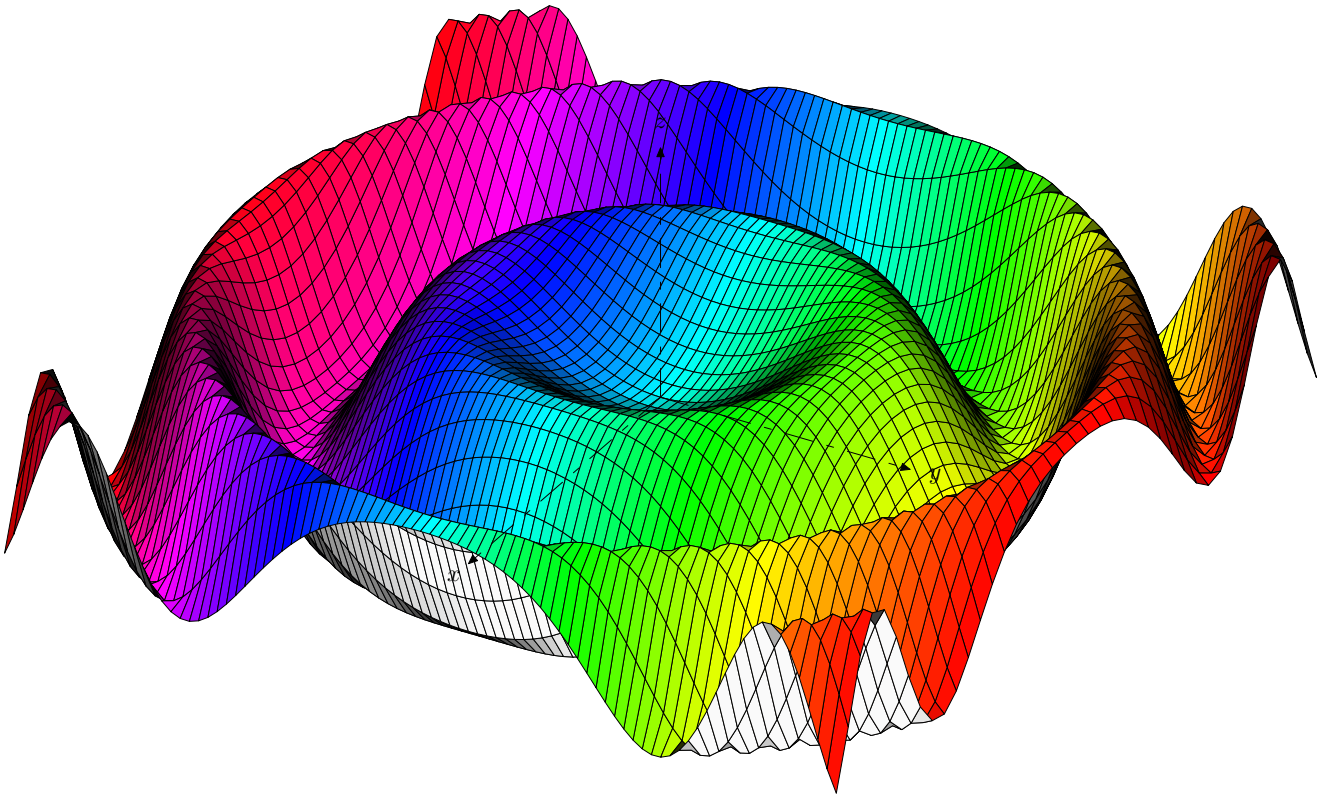


FIGURE 4.2 – La surface $z = \dfrac{\sin(x^2+y^2)}{3}$

```
satu:=1;

%1'9"
figureespace(-10u,-10u,10u,10u);
Initialisation(5000,30,30,40);
arcenciel:=true;
incolor:=0.9[gris,blanc];
draw SurfZ("sin((X**2+Y**2)/3)",-5,5,-5,5,90,120);
TraceAxesD(4,3,3);
finespace;
```

Exemple 3

```
%1"
figureespace(-10u,-10u,10u,10u);
Initialisation(500,50,60,40);
arcenciel:=true;
incolor:=0.9[gris,blanc];
draw SurfZ("X*Y*(X**2-Y**2)*0.1",-3,3,-3,3,30,60);
```

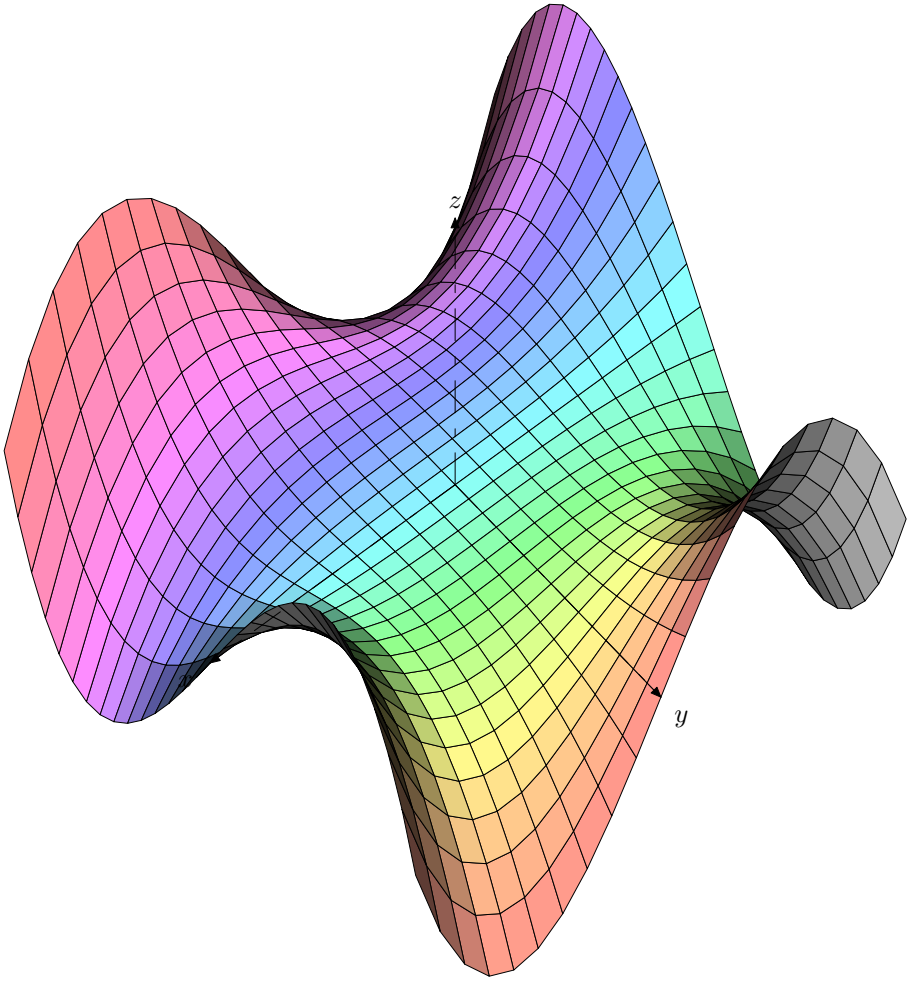


FIGURE 4.3 – La surface $z = xy(x^2 - y^2)$

```
7 TraceAxesD(3,3,5);
8 finespace;
```

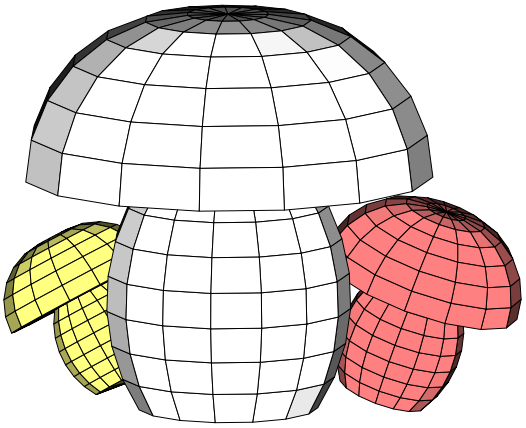


FIGURE 4.5 – Des champignons

Exemple 4

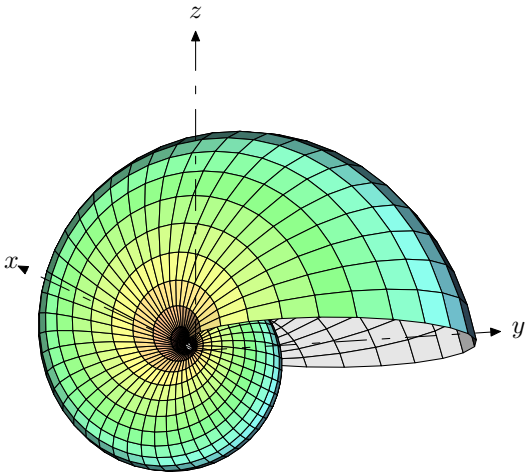


FIGURE 4.4 – Un coquillage

```
%2"
figureespace(-10u,-10u,10u,10u);
Initialisation(2000,20,-10,25);
arcenciel:=true;
incolor:=0.5[gris,white];
draw Sparam("(1.21**v*(sin(u)*cos(u)),1.21**v*(sin(
    u)**2*sin(v)),1.21**v*(sin(u)**2*cos(v)))",pi
    /24,11*pi/12,pi/24,0,10*pi/4,pi/24);
TraceAxesD(8,5,5);
finespace;
```

Exemple 5 Cet exemple nous permet d'utiliser le paramètre `Ferme1` qui indique si le solide n° 1 est « fermé »(c'est-à-dire sans creux visibles) ou non (c'est-à-dire avec des creux visibles ou si l'on souhaite voir l'intérieur du solide). L'intérêt réside dans une diminution du nombre de faces à traiter ; ce qui contribue à un meilleur temps de compilation.
Ici, les paramètres `Ferme1` ... sont tous à `true` : en effet, on ne souhaite pas voir l'intérieur des champignons.

```
Initialisation(500,20,10,10);

vardef Pp(expr R,h)=(0,0)--
  for k=-33 step 10 until 43:
    (cosd(k)*0.5*h-0.2*h,0.3*h+0.5*h*sind(k))--
  endfor
  for l=0 step 15 until 90:
    (0.8*R*cosd(l)+0.2*R,0.8*R*sind(l)+0.6*h)--
```

```
  endfor
  (0,h)
enddef;%pour la section des champignons

outcolor:=blanc;
nb:=30;subh:=15;
h=10;R=5;
Ferme1:=true;%pour ne pas tenir compte des faces non vues
ObjetAnneau1("nbp=18",Pp(R,h));
angx:=-20;
outcolor:=0.5[rouge,blanc];
TR:=(-4,2.5,0);
h:=5;R:=2.5;
Ferme2:=true;
ObjetAnneau2("nbp=18",Pp(R,h));
angx:=30;
TR:=(-4,-4,0);
h:=4;R:=2;
outcolor:=0.5[jaune,white];
Ferme3:=true;
ObjetAnneau3("nbp=18",Pp(R,h));
nbobj:=3;
DessineFusion;
finespace;
```

Exemple 6 Manuel Luque a proposé un document [1] sur la fusée lunaire utilisée par Tintin dans les deux tomes « Objetif Lune » et « On a marché sur la Lune ». J'ai donc repris comme base son travail.

```
enddef;

transformation:=true;

%30"
figureespace(-10u,-10u,10u,10u);
fill feuillet;
for k=1 upto 100:
fill fullcircle scaled (2*uniformdeviate(1)*mm) shifted((uniformdeviate(1))[coinbg,
    coinbd]+uniformdeviate(1)*(coinhg-coinbg)) withcolor jaune;
endfor;

Initialisation(500,-50,20,50);
unit:=0.2;
traits:=false;
angx:=-80;
angy:=20;
outcolor:=rouge;
path rocket[];
rocket1=(0,4.6)--(-0.8,4.6)--(-0.85,5.6)--(-0.95,6.6)--(-1.05,7.6)--(-1.1,8.6)
    --(-1.2,9.6);
rocket2=(-1.2,9.6)--(-1.25,10.6)--(-1.3,11.6)--(-1.35,12.6)--(-1.4,13.6)
    --(-1.4,14.6)--(-1.375,15.6)--(-1.27,16.6)--(-1.2,17.6)--(-1.05,18.6);
```

1. http://melusine.eu.org/syracuse/mluque/solides3d2007/fusee_tintin/objectif_fusee.pdf

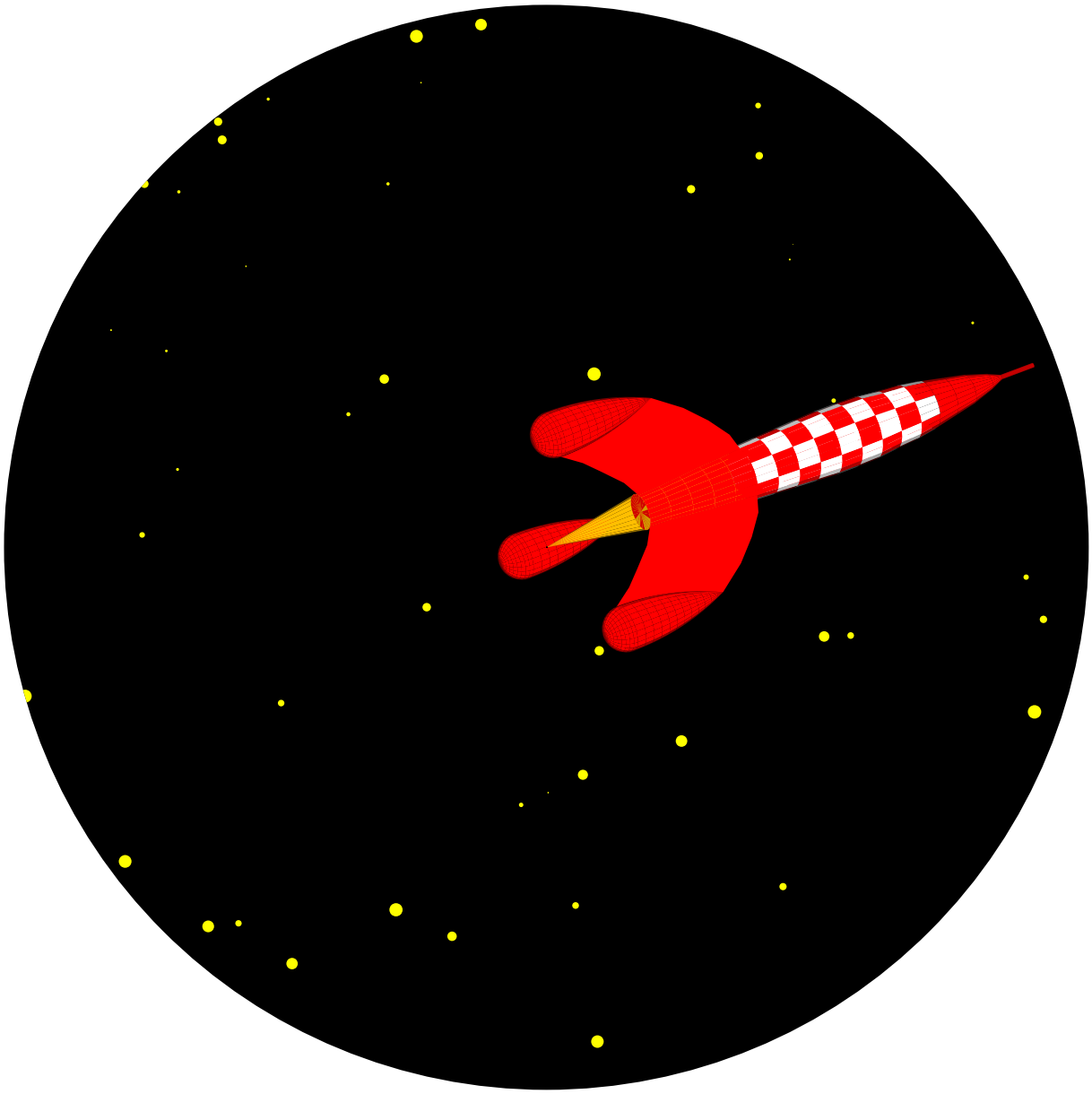
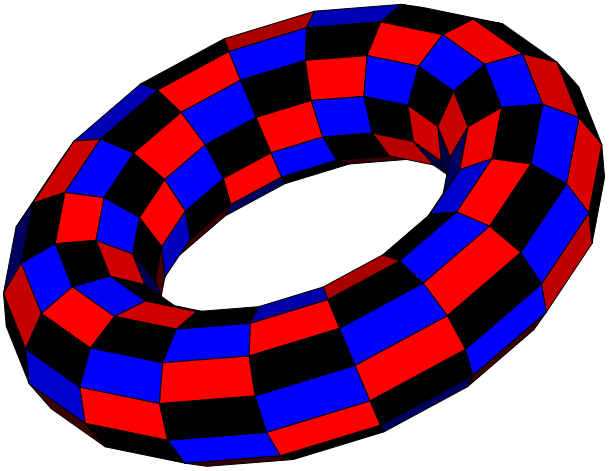


FIGURE 4.6 – La fusée de Tintin

```
21 rocket3=(-1.05,18.6)--(-0.85,19.6)--(-0.65,20.6)--(-0.35,21.6)--(-0.1,22.2)
       --(-0.1,23.75)--(0,23.75);
22 subh:=24;
23 path propulsion;
24 propulsion=(0,4.6)--(-0.8,4.6)--(0,0);
25 perso1:=true;
26 Ferme1:=false;
27 string couleurperso;
28 couleurperso="if ((tapj mod 48)=0) or ((tapj mod 48)=1) or ((tapj mod 48)=2) or ((
       tapj mod 48)=6) or ((tapj mod 48)=7) or ((tapj mod 48)=8) or ((tapj mod 48)=12)
       or ((tapj mod 48)=13) or ((tapj mod 48)=14) or ((tapj mod 48)=18) or ((tapj mod
```

```
    48)=19) or ((tapj mod 48)=20) or ((tapj mod 48)=27) or ((tapj mod 48)=28) or ((
    tapj mod 48)=29) or ((tapj mod 48)=33) or ((tapj mod 48)=34) or ((tapj mod 48)
    =35) or ((tapj mod 48)=39) or ((tapj mod 48)=40) or ((tapj mod 48)=41) or ((tapj
     mod 48)=45) or ((tapj mod 48)=46) or ((tapj mod 48)=47):red else: white fi;";
29 ObjetAnneau1("nbp=10",rocket2);
30 outcolor:=rouge;
31 for k=3 upto 6:
32 Ferme[k]:=true;
33 endfor;
34 incolor:=0.5[jaune,orange];
35 Ferme2:=false;
36 ObjetAnneau2("nbp=7",rocket1);
37 ObjetAnneau3("nbp=7",rocket3);
38 Ferme10:=true;
39 outcolor:=0.5[jaune,orange];
40 ObjetAnneau10("nbp=3",propulsion);
41 outcolor:=rouge;
42 path amortisseur;
43 amortisseur=for k=-90 step 10 until 0:(cosd(k),sind(k)+1)--endfor for k=0 step 2
    until 22:(cosd(k)*11.78-10.78,11.78*sind(k)+1)-- endfor (0,5.75);
44 TR:=Image((0,-5,0));
45 ObjetAnneau4("nbp=22",amortisseur);
46 angz:=120;
47 TR:=(0,0,0);
48 TR:=Image((0,-5,0));
49 ObjetAnneau5("nbp=22",amortisseur);
50 angz:=-120;
51 TR:=(0,0,0);
52 TR:=Image((0,-5,0));
53 ObjetAnneau6("nbp=22",amortisseur);
54 TR:=(0,0,0);
55 angz:=0;
56 subh:=1;
57 ObjetBiface7((0,-0.8,4.6),(0,-0.85,5.6),(0,-0.95,6.6),(0,-1.05,7.6),(0,-1.1,8.6)
    ,(0,-1.2,9.6),(0,-2,9.2),(0,-3,8.3),(0,-4,7.2),(0,-5,5.75),(0,-4.5765,4.6402)
    ,(0,-4.35,3.8498),(0,-4.1789,3.0456),(0,-4.0645,2.2313),(0,-4.0072,1.4111)
    ,(0,-4,1),(0,-3.2,2.2),(0,-2.5,3),(0,-1.6,4));
58 angz:=120;
59 ObjetBiface8((0,-0.8,4.6),(0,-0.85,5.6),(0,-0.95,6.6),(0,-1.05,7.6),(0,-1.1,8.6)
    ,(0,-1.2,9.6),(0,-2,9.2),(0,-3,8.3),(0,-4,7.2),(0,-5,5.75),(0,-4.5765,4.6402)
    ,(0,-4.35,3.8498),(0,-4.1789,3.0456),(0,-4.0645,2.2313),(0,-4.0072,1.4111)
    ,(0,-4,1),(0,-3.2,2.2),(0,-2.5,3),(0,-1.6,4));
60 angz:=-120;
61 ObjetBiface9((0,-0.8,4.6),(0,-0.85,5.6),(0,-0.95,6.6),(0,-1.05,7.6),(0,-1.1,8.6)
    ,(0,-1.2,9.6),(0,-2,9.2),(0,-3,8.3),(0,-4,7.2),(0,-5,5.75),(0,-4.5765,4.6402)
    ,(0,-4.35,3.8498),(0,-4.1789,3.0456),(0,-4.0645,2.2313),(0,-4.0072,1.4111)
    ,(0,-4,1),(0,-3.2,2.2),(0,-2.5,3),(0,-1.6,4));
62 nbobj:=10;
63 DessineFusion;
64 clip currentpicture to (fullcircle scaled 16cm);
65 finespace;
66 end
```

On voit ici apparaître les paramètres `perso`. Ils ont été spécialement crées pour cette fusée. En effet, le principal problème pour définir cette fusée est l'alternance des couleurs blanche et rouge sur le corps de la fusée. Aussi à l'aide de ces paramètres, on peut faire adopter un style spécial à un objet. Par contre, il existe une *contrainte* : la définition de ce style spécial doit se faire par le `string` `couleurperso`.
Voici un autre exemple



```
perso1:=true;
couleurperso:="if ((tapj div subh) mod 3)=0:
    if (tapj mod 3)=0: rouge elseif (tapj mod
    3)=1: noir else: bleu fi elseif ((tapj div
     subh) mod 3)=1:if (tapj mod 3)=1: rouge
    elseif (tapj mod 3)=2: noir else: bleu fi
    elseif ((tapj div subh) mod 3)=2:if (tapj
    mod 3)=2: rouge elseif (tapj mod 3)=0:
    noir else: bleu fi fi;";

nb:=12;
subh:=18;

figureespace(-10u,-10u,10u,10u);
Initialisation(1500,30,20,50);
angy:=25;angz:=-20;
Objettore1("R=3","r=1");
AffichageObjet1;
finespace;
```

Exemple 7 Manuel a également construis une image[2] melant le package `pst-map` et `pst-solides3d`. Il était tentant alors de meler `mp-solides` et `mp-geo`.

```
%figure obtenue en modifiant un petit peu la macro
 %originelle Mappemonde.
 input mp-geo
 input mp-solid;

vardef mappemonde(expr longobs,latobs)=
  projection:="non";
  Initialisation(5,longobs,latobs,500);
  numeric phim,phip,phii;%phi moins -- phi plus - phi intermediaire
  phim=Phi+arcsind(rayon/Rho)-90;
  phip=Phi+90-arcsind(rayon/Rho);
  color pte[];
  pte1=rayon*(cosd(phim)*cosd(Theta),cosd(phim)*sind(Theta),sind(phim));
  pte2=rayon*(cosd(phip)*cosd(Theta),cosd(phip)*sind(Theta),sind(phip));
  pte3=1/2[pte1,pte2];
  pte4-pte3=Normal((0,0,0),pte1,pte2);
  if (Phi>90):
    phip:=180-phip;
```

2. D'autres sont également disponibles à l'adresse

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

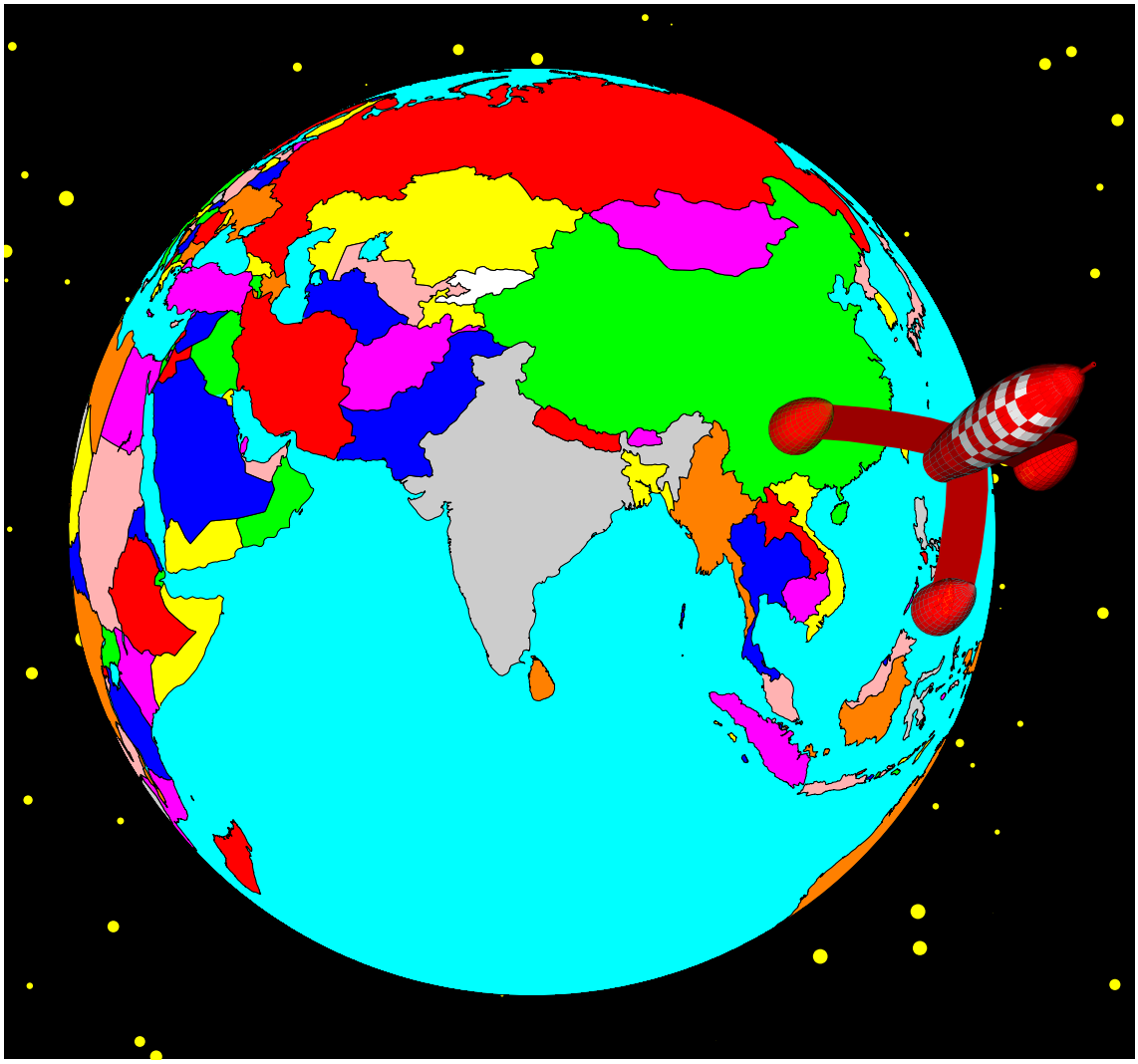


FIGURE 4.7 – La fusée s'éloignant de la Terre

```
19      phii:=180-phim;
20      phim:=phip;
21      phip:=phii;
22    fi;
23    if (Phi<-90):
24      phip:=-180-phip;
25      phii:=-180-phim;
26      phim:=phip;
27      phip:=phii;
28    fi;
29    fill cercles(pte3,pte1,pte3,pte1,pte4) withcolor ciel;
30    Lecture("Cameriquesud.dat");
31    Lecture("Ccaraibes.dat");
32    Lecture("Cameriquecentrale.dat");
33    Lecture("Cameriquenord.dat");
34    Lecture("Casie.dat");
```

```
35   Lecture("Ceurope.dat");
36   Lecture("Cafrique.dat");
37   if volcans=true:
38     Lecturevolcans;
39   fi;
40   if lacs=true:
41     Lecturelacs;
42     Lecturelacssup;
43   fi;
44   Lectureiles;
45   if capitales=true:
46     Lecturecapitales;
47   fi;
48   if fleuves=true:
49     Lecturerivieres;
50   fi;
51   if maillage=true:
52     drawoptions(withcolor gris);
53     MaillageSphere;
54     drawoptions();
55   fi;
56   if maille=true:
57     Maille;
58   fi;
59   draw cercles(pte3,pte1,pte3,pte1,pte4);
60 enddef;
61
62 vardef Transform(expr PT)=RotY(RotX(RotZ(PT)))
63 enddef;
64
65 transformation:=true;
66
67 figureespace(-10u,-10u,10u,10u);
68 picture terre;
69 fleuves:=false;
70 lacs:=false;
71 capitales:=false;
72 terre=image(
73   mappemonde(80,20);
74 );
75 fill feuillet;
76 for k=1 upto 100:
77   fill fullcircle scaled (2*uniformdeviate(1)*mm) shifted((uniformdeviate(1))[coinbg
       ,coinbd]+uniformdeviate(1)*(coinhg-coinbg)) withcolor jaune;
78 endfor;
79 draw terre;
80
81 Initialisation(500,160,20,50);
82 unit:=0.2;
83 traits:=false;
84 angy:=-60;
85 outcolor:=rouge;
86 path rocket[];
```

```
87  rocket1=(0,4.6)--(-0.8,4.6)--(-0.85,5.6)--(-0.95,6.6)--(-1.05,7.6)--(-1.1,8.6)
        --(-1.2,9.6);
88  rocket2=(-1.2,9.6)--(-1.25,10.6)--(-1.3,11.6)--(-1.35,12.6)--(-1.4,13.6)
        --(-1.4,14.6)--(-1.375,15.6)--(-1.27,16.6)--(-1.2,17.6)--(-1.05,18.6);
89  rocket3=(-1.05,18.6)--(-0.85,19.6)--(-0.65,20.6)--(-0.35,21.6)--(-0.1,22.2)
        --(-0.1,23.75)--(0,23.75);
90  subh:=24;
91  perso1:=true;
92  Ferme1:=false;
93  string couleurperso;
94  couleurperso="if ((tapj mod 48)=0) or ((tapj mod 48)=1) or ((tapj mod 48)=2) or ((
        tapj mod 48)=6) or ((tapj mod 48)=7) or ((tapj mod 48)=8) or ((tapj mod 48)=12)
        or ((tapj mod 48)=13) or ((tapj mod 48)=14) or ((tapj mod 48)=18) or ((tapj mod
        48)=19) or ((tapj mod 48)=20) or ((tapj mod 48)=27) or ((tapj mod 48)=28) or ((
        tapj mod 48)=29) or ((tapj mod 48)=33) or ((tapj mod 48)=34) or ((tapj mod 48)
        =35) or ((tapj mod 48)=39) or ((tapj mod 48)=40) or ((tapj mod 48)=41) or ((tapj
         mod 48)=45) or ((tapj mod 48)=46) or ((tapj mod 48)=47):red else: white fi;";
95  TR:=(0,-15,0);
96  ObjetAnneau1("nbp=10",rocket2);
97  outcolor:=rouge;
98  for k=2 upto 6:
99  Ferme[k]:=true;
100 endfor;
101 ObjetAnneau2("nbp=7",rocket1);
102 ObjetAnneau3("nbp=7",rocket3);
103 path amortisseur;
104 amortisseur=for k=-90 step 10 until 0:(cosd(k),sind(k)+1)--endfor for k=0 step 2
        until 22:(cosd(k)*11.78-10.78,11.78*sind(k)+1)-- endfor (0,5.75);
105 TR:=(0,-15,0)+(0,-5,0);
106 ObjetAnneau4("nbp=22",amortisseur);
107 angz:=150;
108 TR:=(0,0,0);
109 TR:=(0,-15,0)+Image((0,-5,0));
110 ObjetAnneau5("nbp=22",amortisseur);
111 angz:=-90;
112 TR:=(0,0,0);
113 TR:=(0,-15,0)+Image((0,-5,0));
114 ObjetAnneau6("nbp=22",amortisseur);
115 TR:=(0,-15,0)+(0,0,0);
116 angz:=0;
117 subh:=1;
118 ObjetBiface7((0,-0.8,4.6),(0,-0.85,5.6),(0,-0.95,6.6),(0,-1.05,7.6),(0,-1.1,8.6)
        ,(0,-1.2,9.6),(0,-2,9.2),(0,-3,8.3),(0,-4,7.2),(0,-5,5.75),(0,-4.5765,4.6402)
        ,(0,-4.35,3.8498),(0,-4.1789,3.0456),(0,-4.0645,2.2313),(0,-4.0072,1.4111)
        ,(0,-4,1),(0,-3.2,2.2),(0,-2.5,3),(0,-1.6,4));
119 angz:=150;
120 ObjetBiface8((0,-0.8,4.6),(0,-0.85,5.6),(0,-0.95,6.6),(0,-1.05,7.6),(0,-1.1,8.6)
        ,(0,-1.2,9.6),(0,-2,9.2),(0,-3,8.3),(0,-4,7.2),(0,-5,5.75),(0,-4.5765,4.6402)
        ,(0,-4.35,3.8498),(0,-4.1789,3.0456),(0,-4.0645,2.2313),(0,-4.0072,1.4111)
        ,(0,-4,1),(0,-3.2,2.2),(0,-2.5,3),(0,-1.6,4));
121 angz:=-90;
122 ObjetBiface9((0,-0.8,4.6),(0,-0.85,5.6),(0,-0.95,6.6),(0,-1.05,7.6),(0,-1.1,8.6)
        ,(0,-1.2,9.6),(0,-2,9.2),(0,-3,8.3),(0,-4,7.2),(0,-5,5.75),(0,-4.5765,4.6402)
```

```
      ,(0,-4.35,3.8498),(0,-4.1789,3.0456),(0,-4.0645,2.2313),(0,-4.0072,1.4111)
      ,(0,-4,1),(0,-3.2,2.2),(0,-2.5,3),(0,-1.6,4));
nbobj:=9;
DessineFusion;
finespace;
end
```

Chapitre 5

Historique

25/01/2011 Modification de l'objet `Deplacement` (gestion des couleurs). Modification mineure de la documentation (Ajout d'un exemple dans la partie transformation de la section 3).

27/11/2010 Corrections de `Redundant equation`.

31/08/2008 Ajout de la transparence pour l'affichage des objets.

21/08/2008 Ajout des objets `OFF`, `OBJ`, `tetraedre`, `octaedre`, `dodecaedre` et `icosaedre`.

16/08/2008 Ajout de l'objet `New`.

13/08/2008 Ajout des objets `Fusion` et `Déplacement`.

12/08/2008 Possibilité de couper un objet par un plan.

05/08/2008 Possibilité de numéroter et d'enlever des faces.

01/08/2008 Ajout des solides `biface` et `calotte sphérique pleine`.

30/07/2008 Redéfinition de l'affichage et de la fusion d'objets. Ajout des solides `grille` et `ruban`.

29/07/2008 Debut de la mise en place des transformations (uniquement disponible pour les objets).

28/07/2008 Ajout d'objets et de la fusion.

19/07/2008 Ajout de la gestion de la lumière.

18/07/2008 Ajout de l'espace de couleurs HSV.

26/06/2008 Ajout des solides de révolution. Premiers essais sur la fusion de deux objets.

25/06/2008 Ajout des surfaces $z = f(x, y)$.

24/06/2008 Ajout « des tubes » dans la représentation des courbes. Ajout des surfaces paramétrées.

23/06/2008 Lancement de la première version très *expérimentale*. Tout est encore en vrac. La lecture de fichiers externes au format `OFF` est possible.

Table des figures