

Macros de géométrie spatiale avec **Asymptote**

D. Comin

4 février 2009

Table des matières

1	Les projections	2
2	Les directions	3
3	Les points	3
3.1	Tracé	3
3.2	Construction	3
4	Des faces cachées	4
4.1	Les pavés	4
4.2	Les prismes droits	5
4.3	Les pyramides	5
4.4	Les solides définis par l'utilisateur	6
4.5	Les constantes	7
5	Les solides de revolution	8
5.1	le tracé des solides	8
5.2	Les sections planes	9
6	Mesure et codage	9
6.1	Cotation	9
6.2	Codage des longueurs et angles	10
7	Segments	10
8	Cercles	11

1 Les projections

Ce paragraphe est une réécriture de la documentation de **Asymptote** concernant le module **three.asy**. Toutes les commandes dont il est question ici en sont issues.

Asymptote connaît cinq type de projections, elles conditionnent la façon dont est représenté l'espace en 3D sur une feuille plane.

►**oblique(real angle)** : Le point (x, y, z) est projeté sur $(x - 0.5z, y - 0.5z)$. Les objets contenus dans le plan xOy sont en vraies grandeurs. L'angle est optionnel, il permet de fixé l'angle que fait la partie négative des abscisses avec l'horizontale sur le plan de projection.

►**obliqueX(real angle)** : Le point (x, y, z) est projeté sur $(y - 0.5x, z - 0.5x)$. Les objets contenus dans le plan yOz sont en vraies grandeurs : cette projection est commune en Mathématiques. L'angle est optionnel, il permet de fixé l'angle que fait la partie négative des abscisses avec l'horizontale sur le plan de projection.

►**obliqueY(real angle)** : Le point (x, y, z) est projeté sur $(x + 0.5y, z + 0.5y)$. Les objets contenus dans le plan xOz sont en vraies grandeurs. L'angle est optionnel, il permet de fixé l'angle que fait la partie positive des ordonnées avec l'horizontale sur le plan de projection.

Pour l'instant, les trois projections précédentes fonctionnent mal avec le module **bsp.asy**, qui fait apparaître des bugs dans les faces cachées. Les deux autres fonctionnent, elles, à merveille :

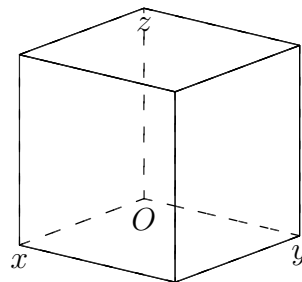
►**orthographic(real x, real y, real z, triple up=Z)** : La caméra (définie par x, y et z) fixe ici un point à l'infini. La direction "up=Z" veut dire que la caméra est debout ..., un autre vecteur pourra la tourner sur un axe horizontal. Les droites parallèles sont représentées parallèlement.

►**perspective(real x, real y, real z, triple up=Z, triple target=0)** est la perspective la plus réaliste, elle possède au moins un point de fuite vers lequel convergent les parallèles. Ce n'est pas la plus conforme aux exigences d'un cours de maths ...

Par défaut, la projection est **currentprojection=perspective(5,4,2)** ;.

Les figures suivantes montrent le cube unité selon les deux perspectives qui vont bien :

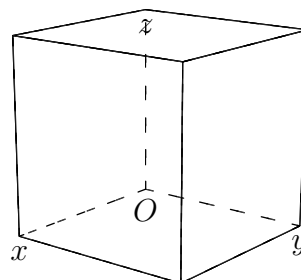
```
import geospace; size(4cm,0);
currentprojection=orthographic(5,4,2);
label("obliqueX",(0,0,1.1),nord);
pave((0,0,0),1,1,1); trace();
label("x",project((1,0,0)),sud);
label("y",project((0,1,0)),sud);
label("z",project((0,0,1)),sud);
label("O",project((0,0,0)),sud);
```



```

import geospace; size(4cm,0);
currentprojection=perspective(5,4,2);
label("obliqueX",(0,0,1.1),nord);
pave((0,0,0),1,1,1); trace();
label("x",project((1,0,0)),sud);
label("y",project((0,1,0)),sud);
label("z",project((0,0,1)),sud);
label("O",project((0,0,0)),sud);

```



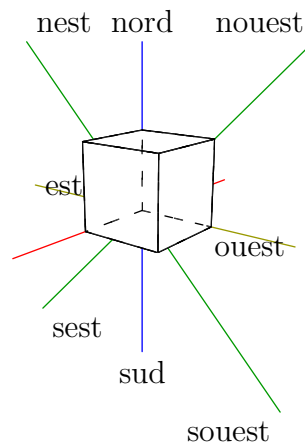
2 Les directions

Asymptote connaît les huit principaux points cardinaux comme variables de type «pair» : N pour le nord, S pour le sud, etc. Ces directions sont pratiques pour nommer les points mais interdisent d'utiliser certaines lettres : N, S, W, E... Ces directions sont renommées comme suit.

```

import geospace; size(4cm,0);
label("nord",(0,0,2),nord);
label("sud",(0,0,-2),sud);
label("est",(0,-2,0),est);
label("ouest",(0,2,0),ouest);
label("nest",(0,-2,2),nest);
label("sest",(0,-2,-2),sest);
label("nouest",(0,2,2),nouest);
label("souest",(0,2,-2),souest);
trace((-2,0,0)-(2,0,0),red);
trace((0,0,-2)-(0,0,2),blue);
trace((0,-2,0)-(0,2,0),yellow*0.6);
trace((0,-2,-2)-(0,2,2),green*0.6);
trace((0,-2,2)-(0,2,-2),green*0.6);
pave((0,0,0),1,1,1); trace();

```



3 Les points

3.1 Tracé

► `pointe(triple A, pen p=currentpen)` trace une croix en A sans nommer le point.

► `nomme(Label L, triple position, pen p=currentpen)` trace le point sur le pair «position» et place le texte contenu dans le «label» L.

3.2 Construction

► `projortho(triple M, triple A, triple B, triple C)` renvoie le projeté de M orthogonal sur le plan passant par A, B et C selon une normale à ce plan.

- `intersectionDP(triple M, triple N, triple A, triple B, triple C)` renvoie l'intersection de (MN) avec le plan passant par A, B et C.
- `intersectionDD(triple M, triple N, triple A, triple B)` renvoie l'intersection de (MN) avec la droite (AB).
- `pointsur(path3 chemin, real r)` retourne un point sur un chemin avec le paramètre `r` entre 0 et 1.
- `milieu(triple A, triple B)` définit le milieu de [AB].
- `triple iso(path3 face)` renvoie l'isobarycentre d'une face

On trouvera des exemples d'utilisation de ces commandes plus loin.

4 Des faces cachées

Le rendu des solides est assuré par le module `bsp.asy`. La solution qui gère les arêtes cachées a été trouvée par Philippe Ivaldi ¹, je n'ai fait que l'adapter à mes besoins.

Un solide est de type `shape` qui est un tableau de `path3`. Tous les solides définis par l'utilisateur et les `path3` *fermés* doivent être insérés dans la scène finale par la commande :

► `void insertscene(shape shp, pen remplissage=white, pen aretes=black)` qui est l'équivalent de `filldraw`. Une fois que tous les solides, `path3` fermés sont définis, on les trace d'un coup par la commande `trace{}`.

Une fois tracé ce qui doit être caché, en pointillé ou pas, on peut faire apparaître les points, les `label`, etc ...

Les `path3` non fermés ne sont, eux, pas gérés par `insertscene` ou `trace()`, mais ils sont tracés par la commande :

► `trace(path3 p, pen p=currentpen)` qui admet les mêmes arguments que `draw`.

4.1 Les pavés

► `pave(triple A, real L, real p, real h, pen remplissage=white, pen aretes=black)` trace un parallélépipède rectangle dont A est un sommet (face arrière, en bas à gauche), de longueur L parallèlement à l'axe (yy'), de profondeur p parallèlement à (xx') et de hauteur h parallèlement à (zz').

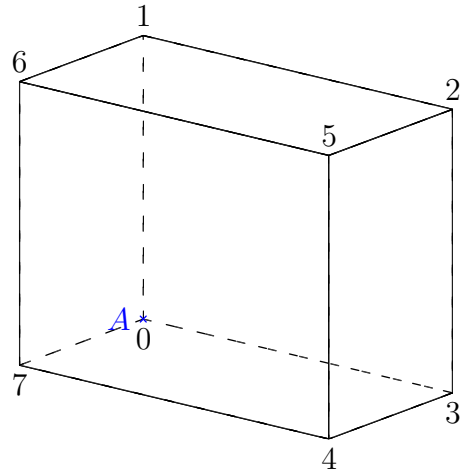
La commande renvoie aussi les sommets du pavé dans un tableau de type `triple[]`.

¹<http://piprim.tuxfamily.org/asymptote/three/index.html>

```

import geospace;
currentprojection=orthographic(5,4,2);
triple A,B,C,D,E;
size(6cm,0);
A=(0,0,0);
triple[] p=pave(A,4,2,3);
trace();
nomme("$A$",A,ouest,blue);
label("0",p[0],sud);
label("1",p[1],nord);
label("2",p[2],nord);
label("3",p[3],sud);
label("4",p[4],sud);
label("5",p[5],nord);
label("6",p[6],nord);
label("7",p[7],sud);

```



4.2 Les prismes droits

Il suffit de définir les points constitutifs de la base (et donc coplanaires), les sommets du prisme sont renvoyés dans un tableau de type `triple[]`.

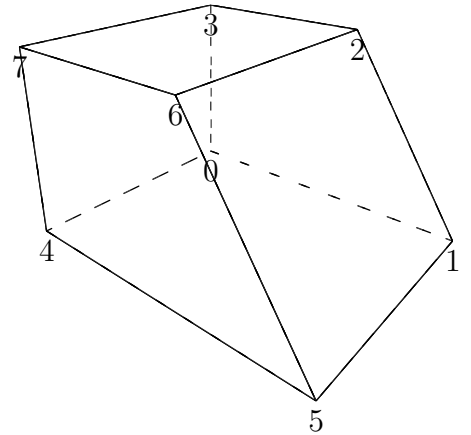
Il y a une commande :

► `triple[] prisme(real h, path3 B, pen remplissage=white, pen aretes=black)` trace un prisme de base B et de hauteur h.

```

import geospace;
currentprojection=perspective(8,7,5);
triple A,B,C,D,E;
size(6cm,0);
A=(0,0,0);
B=(0,5,0);
C=(0,3,3);
D=(0,0,3);
triple[] p=prisme(4,A--B--C--D--cycle);
label("0",p[0],sud);
label("1",p[1],sud);
label("2",p[2],sud);
label("3",p[3],sud);
label("4",p[4],sud);
label("5",p[5],sud);
label("6",p[6],sud);
label("7",p[7],sud);

```

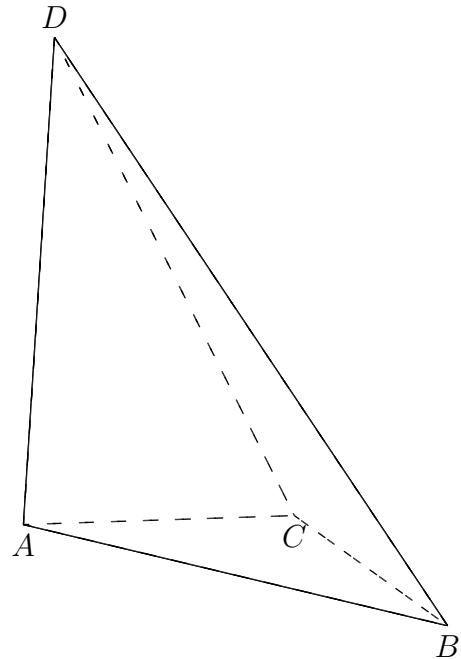


4.3 Les pyramides

► `pyramide(triple S, path3 B, pen remplissage=white, pen aretes=black, real opacite=1, bool pointilles=true)` trace un tétraèdre de sommet D et

de base le `path3` B.

```
import geospace;
triple A,B,C,D;
size(6cm,0);
A=(0,0,0);
B=(0,3,0);
C=(-1.5,1.5,0);
D=(1,1,3);
pyramide(D,A--B--C--cycle);
trace();
label("$A$",A,sud);
label("$B$",B,sud);
label("$C$",C,sud);
label("$D$",D,nord);
```



4.4 Les solides définis par l'utilisateur

On peut définir ses propres solides, il suffit de donner les faces sous la forme de `path3`. On insère le solide avec `insertscene()`.

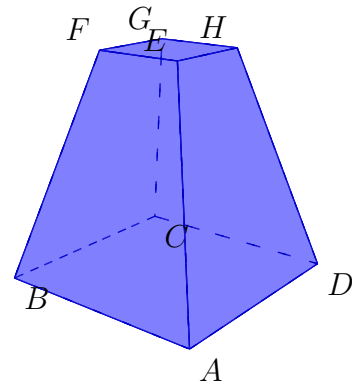
► `triple[] sommet(shape solide)` renvoie les sommets d'un polyèdre.

► `void nomme(shape solide ...string[] txt)` ou `nomme(triple[] M, pair direction=sud ...string[] txt)` nomme les sommets d'un polyèdre ou un tableau de points.

```
import geospace;
```

```
currentprojection=perspective(5,4,3);
size(5cm);
```

```
real a=1;
real h=2;
real k=0.4;
```



```
shape pyra;
pyra[0]=(a,a,0)--(a,-a,0)--(-a,-a,0)--(-a,a,0)--cycle;
pyra[1]=(k*a,k*a,h)--(k*a,-k*a,h)--(-k*a,-k*a,h)--(-k*a,k*a,h)--cycle;
pyra[2]=(a,a,0)--(a,-a,0)--(k*a,-k*a,h)--(k*a,k*a,h)--cycle;
pyra[3]=(a,a,0)--(-a,a,0)--(-k*a,k*a,h)--(k*a,k*a,h)--cycle;
pyra[4]=(-a,a,0)--(-a,-a,0)--(-k*a,-k*a,h)--(-k*a,k*a,h)--cycle;
pyra[5]=(-a,-a,0)--(a,-a,0)--(k*a,-k*a,h)--(-k*a,-k*a,h)--cycle;
```

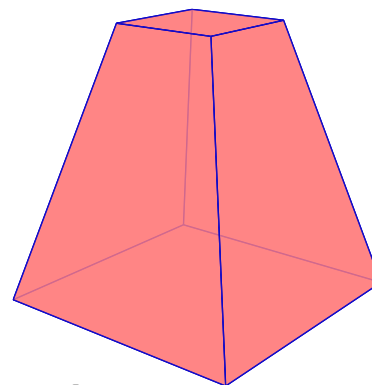
```
insertscene(pyra,lightred+opacity(0.8),blue*0.8);
trace();
```

4.5 Les constantes

- `croix` donne la taille des croix tracées pour les points.
- `bool pointilles` (à `true` par défaut) trace les arêtes cachées ou pas.

```
import geospace;
pointilles=false;
currentprojection=perspective(5,4,3);
size(5cm);
```

```
real a=1;
real h=2;
real k=0.4;
```



```
shape pyra;
pyra[0]=(a,a,0)--(a,-a,0)--(-a,-a,0)--(-a,a,0)--cycle;
pyra[1]=(k*a,k*a,h)--(k*a,-k*a,h)--(-k*a,-k*a,h)--(-k*a,k*a,h)--cycle;
pyra[2]=(a,a,0)--(a,-a,0)--(k*a,-k*a,h)--(k*a,k*a,h)--cycle;
pyra[3]=(a,a,0)--(-a,a,0)--(-k*a,k*a,h)--(k*a,k*a,h)--cycle;
pyra[4]=(-a,a,0)--(-a,-a,0)--(-k*a,-k*a,h)--(-k*a,k*a,h)--cycle;
pyra[5]=(-a,-a,0)--(a,-a,0)--(k*a,-k*a,h)--(-k*a,-k*a,h)--cycle;
```

```
insertscene(pyra,lightblue+opacity(0.8),blue*0.8);
trace();
triple[]p=sommet(pyra);
//nomme les sommets de 0 à 3 (4 exclu)
nomme(p[:4],sest,"$A$","$B$","$C$","$D$");
//nomme les sommets à partir de 4
nomme(p[4:],nouest,"$E$","$F$","$G$","$H$");
```

5 Les solides de revolution

Ces macros utilisent le module `solids.asy` et la possibilité de faire facilement des solides de revolution à partir d'un `path3`.

5.1 le tracé des solides

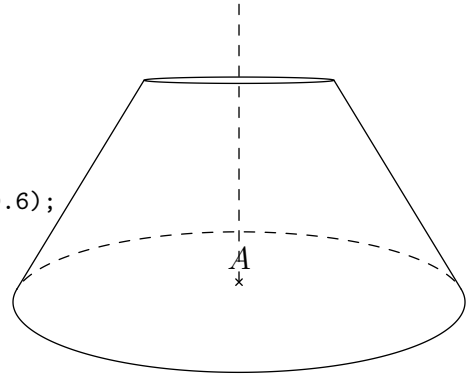
► `revolution conerevolution(triple C, real rayon, real hauteur, real section=1, triple axe=Z)` renvoie comme objet de type `revolution` le cône de centre C, de rayon R, de hauteur h et d'axe axe Z par défaut. Le paramètre `section` -entre 0 et 1- sert à tracer des cônes tronqués.

► `revolution sphere(triple c=0, real rayon, real k=0, real h, int n=32)` définit une calotte spherique comme solide de revolution. k et h doivent etre entre 0 et 1, ils définissent respectivement le debut et la fin de la calotte : 0 correspond au pole sud et 1 au nord de la sphere associée.


```

import geospace;
triple A;
A=(0,0,0);
size(6cm,0);
revolution C=conerevolution(A,2,3,section=0.6);
draw(C);
nomme("$A$",A,nord);
trace(A--(0,0,2.4),dashed);

```



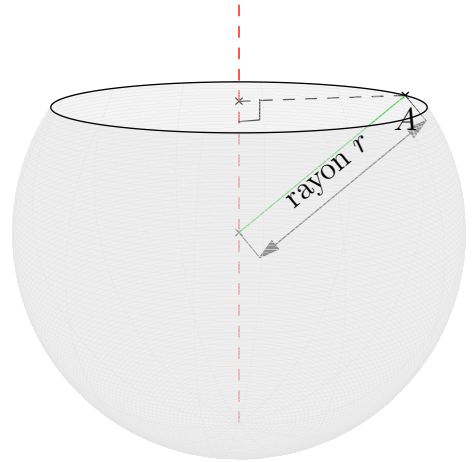
5.2 Les sections planes

► `path3 sectionplane(revolution r, real section, pen p=currentpen)` trace la section plane d'un solide de révolution à une hauteur h entre 0 et 1. La commande retourne aussi cette section sous forme de `path3` : utile pour placer des points sur un solide de révolution.

```

size(6cm,0);
import geospace;
currentprojection=perspective(5,4,2);
triple C=(0,0,0);
revolution c=sphere(C,2,h=0.8);
draw(surface(c),lightgrey+opacity(0.4));
path3 section=sectionplane(c,1);
triple A=pointsur(section,0.4);
nomme("$A$",A,sud);
trace((0,0,1.2)--A,dashed);
trace((0,0,-2)--(0,0,2),dashed+red);
pointe((0,0,0));
triple D=(0,0,1.2);
pointe(D);
angledroit(C,D,A);
trace(C--A,green*0.8);
cote(C,A,"rayon $r$",-3,trait=true);

```



6 Mesure et codage

6.1 Cotation

► `cotemilieu(triple A, triple B, string texte, real d, pen sty=black)` trace une flèche de A à B à d mm au dessus de (AB), le texte est au milieu.

► `cote(triple A, triple B, string texte, real d, trait=false, pen sty=black)` trace une flèche de A à B à d mm au dessus de (AB), le texte est au dessus si d est positif, dessous sinon. `trait` gère le tracé des lignes de rappels ou non.

► `etiquette(triple A, triple B, string txt, bool dessus=true, pen sty=currentpen)`
place le texte `txt` le long de `[AB]`.

6.2 Codage des longueurs et angles

► `code(pen sty=invisible, int trait ... pair[] K)` code une série de segments dont les extrémités sont contenues dans une matrice de type «`triple[]`». Le paramètre "trait" précise le codage :

Si "trait" vaut 1, 2 ou 3 le segment est codé par des traits ...

Si "trait" vaut 4 le segment est codé par un tilde.

Si "trait" vaut 5 le segment est codé par un cercle.

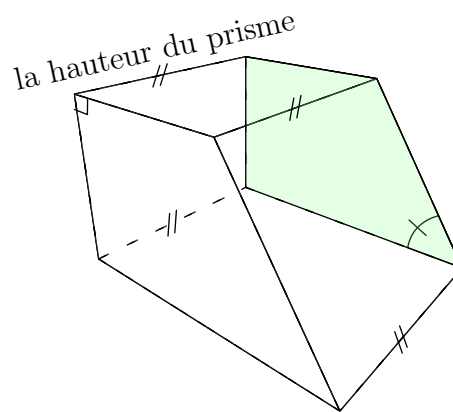
► `codemilieu(triple A, triple B, int trait)` est similaire à `code` mais plus simple syntaxiquement.

► `codeangle(triple D, triple A, triple E, string txt="", int trait, pen remplissage=invisible)` marque l'angle \widehat{ABC} par des traits et un ou plusieurs arcs de cercle.

► `angledroit(triple A, triple B, triple C, real taille=3mm, pen p=black)`

trace un carré (en perspective) codant l'angle \widehat{ABC} qui est supposé droit.

```
import geospace;
croix=0.2;
currentprojection=perspective(8,7,5);
triple A,B,C,D,E,F,G,H;
size(6cm,0);
A=(0,0,0);
B=(0,5,0);
C=(0,3,3);
D=(0,0,3);
triple[] p=prisme(4,A--B--C--D--cycle);
trace();
E=p[7];
F=p[6];
G=p[5];
H=p[4];
angledroit(H,E,F,0.4);
codeangle(A,B,C,trait=1,r=01);
filldraw(project(A--B--C--D--cycle),lightgreen+opacity(0.2));
code(2,E,D,F,C,G,B,H,A);
etiquette(E,D,"la hauteur du prisme");
```



7 Segments

► La commande `segment(triple A, triple B, real a=0)` renvoie le chemin `[AB]` et permet de faire dépasser le trait de `a` cm de part et d'autre des extrémités.

8 Cercles

► `path3 cercle(triple O, triple A, triple n)` renvoie un cercle de centre O et passant par A dans le plan de vecteur normal \vec{n} .

► `path3 cercleR(triple O, real R, triple n)` renvoie un cercle de centre O et de rayon R le plan de vecteur normal \vec{n} .

```
import geospace;
size(8cm,0);
currentprojection=orthographic(5,4,2);
triple A=(0,0,0);
triple B=(1,0,0);
triple C=rotate(60,A,Z)*B;
triple Som=(A+B+C)/3+(0,0,1);
triple[] P = prisme(1,A--B--C--cycle);

croix=0.02;
path3 c=cercleR((A+B+C)/3,1,Z);
insertscene(c,invisible,black);

trace();

nomme("A",A,sud);
nomme("B",B,sud);
nomme("C",C,sud);
pointe(Som);

code(2,A,B,B,C,C,A);
angledroit(Som,(A+B+C)/3,C,0.1,red);
draw(project(Som--(A+B+C)/3),dashed);
```

