jps2ps : version 0.15

par Jean-Paul Vignault
Groupe des Utilisateurs de Linux Poitevins (GULP)
(**jpv@melusine.eu.org**)
16 Août 2006

Ce document présente les différences entre la version 0.15 et la précédente.

1. Divers

$\{f\}$ **currentpathtransform** $- \longrightarrow$ applique la transformation f au chemin courant, où $f : \mathbb{R}^2 \to \mathbb{R}^2$

$array_1$ **uniqp** $array_2 \longrightarrow array_2$ est le tableau obtenu à partir du tableau de points $array_1$ en suprimant les points identiques consécutifs

$array_1$ $array_2$ **append** $array_3 \longrightarrow array_3$ est le tableau obtenu en concaténant les tableaux $array_1$ et $array_2$

$string_1$ $string_2$ **append** $string_3 \longrightarrow string_3$ est la chaîne de caractères obtenue en concaténant les chapines de caractères $string_1$ et $string_2$

2. Chemins continus paramétrés

Le type *chemin* de la version 0.14 est repris plus modestement en type *chemin continu*, en attendant une version future plus complète.

2.1 - L'objet *chemin continu paramétré*

Un objet *chemin continu paramétré* est une structure complexe représentant une application du type :

$$\begin{aligned} f : I &\to \mathbb{R}^2 \\ t &\mapsto (x, y) \end{aligned}$$

Cet objet possède 2 composantes : le tableau des valeurs du paramètre, et le tableau des points (x, y) correspondants. Dans la pratique, les coordonnées des points sont exprimées soit dans le repère jps, soit dans le repère postscript sous-jacent. Les commandes **cppathtocpath** et **cpathtocppath** permettent de passer d'un type de chemin à l'autre, et les commandes **cpathpointstable** et **cpathparamtable** permettent d'accéder en lecture à chacune des composantes de l'objet.

La commande **drawcpath** permet de dessiner un chemin continu paramétré passé en argument.

$cpathobj_1$ **cppathtocpath** $cpathobj_2 \longrightarrow$ transforme le chemin continu paramétré $cpathobj_1$ exprimé dans le repère postscript en le chemin continu paramétré $cpathobj_2$ exprimé dans le repère jps

$cpathobj_1$ **cpathtocppath** $cpathobj_2 \longrightarrow$ transforme le chemin continu paramétré $cpathobj_1$ exprimé dans le repère jps en le chemin continu paramétré $cpathobj_2$ exprimé dans le repère postscript

$cpathobj$ **cpathpointstable** $array \longrightarrow array$ est le tableau de points du chemin continu paramétré $cpathobj$

$cpathobj$ **cpathparamtable** $array \longrightarrow array$ est le tableau des paramètres du chemin continu paramétré $cpathobj$

$cpathobj$ $string$ **drawcpath** $- \longrightarrow$ Dessine le chemin continu représenté par $cpathobj$. $string$ est un paramètre optionnel indiquant le type de terminaison de ligne

2.2 - Création d'un chemin continu paramétré

Chaque fois que l'une des commandes de tracé du format jps est exécutée (**frame**, **droite**, **cercle**, **ellipse**, **courbe**, **courbeparam**, **sarc**, **sarcn**, **tripointarc**, **draw**, **bezier_curve**), le chemin correspondant est sauvegardé sous forme de chemin continu paramétré, le paramètre appartenant à l'intervalle [0; 100]. Les commandes **lastcpath** et **lastcppath** permettent alors d'accéder à ce chemin, exprimé dans le repère jps ou dans le repère postscript sous-jacent.

On peut également créer un chemin continu paramétré à partir du chemin courant : **stockcurrentcpath** transforme le chemin continu courant en chemin paramétré et réaffecte les commandes **lastcpath** et **lastcppath**. On dispose également de 4 autres commandes permettant d'accéder, sans modifier les variables **lastcpath**, au chemin continu paramétré défini par le chemin continu courant.

$-$ **lastcpath** $cpathobj \longrightarrow$ l'objet cpath, coordonnées dans le repère jps, associé au dernier chemin continu dessiné

– **lastcppath** *cpathobj* ⟶ l'objet cpath, coordonnées dans le repère postcript, associé au dernier chemin continu dessiné

– **stockcurrentcpath** – ⟶ sauvegarde le chemin continu courant sous forme de chemin continu paramétré, et réaffecte **lastcpath** et **lastcppath** en conséquence

– **currentcpathpointstable** *array* ⟶ tableau des points définissant le chemin continu courant dans le repère jps

– **currentcppathpointstable** *array* ⟶ tableau des points définissant le chemin continu courant dans le repère postscript

– **currentcpathobj** *cpathobj* ⟶ chemin continu paramétré courant (dans le repère jps)

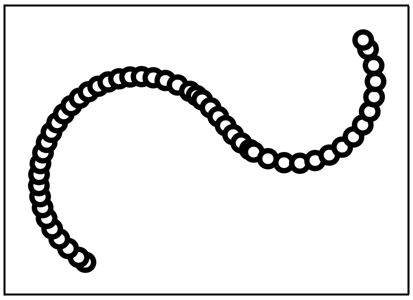
– **currentcppathobj** *cpathobj* ⟶ chemin continu paramétré courant (dans le repère postscript)

2.3 - Points d'un chemin continu paramétré

cpathobj **cpathstartpoint** A ⟶ A est le premier point du chemin continu paramétré *cpathobj*

cpathobj **cpathendpoint** B ⟶ B est le dernier point du chemin continu paramétré *cpathobj*

t *cpathobj* **cpathpoint** M ⟶ M est le point de paramètre t du chemin continu paramétré *cpathobj*



source jps

```
autocrop
[-3 -3 .. -1 0 .. 0 -1 .. 2 1] draw
/path1 lastcpath def
[0 2 100 {} for] {path1 cpathpoint circ2} apply
```

2.4 - Opérations sur les chemins paramétrés

$cpathobj_1$ **reversecpathobj** $cpathobj_2$ ⟶ $cpathobj_2$ est le chemin continu obtenu à partir de $cpathobj_1$ en inversant le sens de parcours

cpathobj **cpathlongueur** ℓ ⟶ longueur de chemin continu parmétré *cpathobj*

t *cpathobj* **cpathlongueurs** ℓ_1 ℓ_2 ⟶ ℓ_1 et ℓ_2 sont les longueurs respectives des sous-chemins $\widehat{AM}$ et $\widehat{MB}$, où M est le point de paramètre t du chemin continu paramétré *cpathobj*, et où A et B sont respectivement les premier et dernier points de *cpathobj*

t *cpathobj* **cpathslongueur** ℓ ⟶ longueur du sous-chemin continu $\widehat{AM}$, où M est le point de paramètre t du chemin continu paramétré *cpathobj*, et A est le premier point de *cpathobj*

t *cpathobj* **cpathelongueur** ℓ ⟶ longueur du sous-chemin continu $\widehat{MB}$, où M est le point de paramètre t du chemin continu paramétré *cpathobj*, et B est le dernier point de *cpathobj*

t *cpathobj* **splitcpath** $cpathobj_1$ $cpathobj_2$ ⟶ sépare, à partir du point de paramètre t, le chemin continu paramétré *cpathobj* en 2 sous-chemins paramétrés

$cpathobj_1$ **normalizecpath** $cpathobj_2$ ⟶ Les chemins décrits par $cpathobj_1$ et $cpathobj_2$ sont confondus, mais le tableau des paramètres de $cpathobj_2$ a été modifié, de telle facon que ceux-ci soient dans l'intervalle [0; 100], et qu'ils soient répartis de façon proportionnelle à la longueur du chemin

$cpathobj_1$ u **translatecpath** $cpathobj_2$ ⟶ $cpathobj_2$ est le chemin continu image de $cpathobj_1$ par la translation de vecteur u

$cpathobj_1$ **projxcpath** $cpathobj_2$ ⟶ $cpathobj_2$ est le chemin continu projeté orthogonal de $cpathobj_1$ sur l'axe Ox

$cpathobj_1$ **projycpath** $cpathobj_2$ ⟶ $cpathobj_2$ est le chemin continu projeté orthogonal de $cpathobj_1$ sur l'axe Oy

$cpathobj_1$ D **orthoprojcpath** $cpathobj_2$ ⟶ $cpathobj_2$ est le chemin continu projeté orthogonal de $cpathobj_1$ sur la droite D

$cpathobj_1$ I α **rotatecpath** $cpathobj_2$ ⟶ $cpathobj_2$ est le chemin continu image de $cpathobj_1$ par la rotation de centr(e I et d'angle α

$cpathobj_1$ I k **homcpath** $cpathobj_2$ ⟶ $cpathobj_2$ est le chemin continu image de $cpathobj_1$ par l'homothétie de centre I et de rapport k

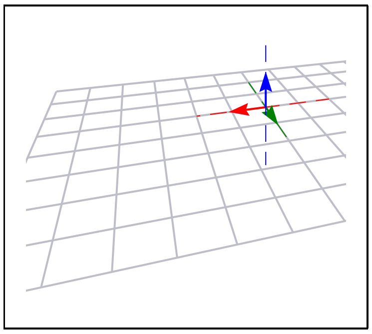
$cpathobj_1$ I **symcpath** $cpathobj_2$ ⟶ $cpathobj_2$ est le chemin continu image de $cpathobj_1$ par la symétrie de centre I

$cpathobj_1$ D **axesymcpath** $cpathobj_2 \longrightarrow cpathobj_2$ est le chemin continu image de $cpathobj_1$ par la symétrie axiale d'axe D

3. Commandes 3d

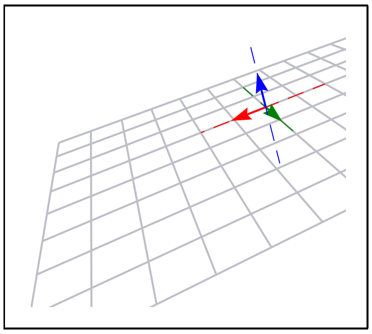
On rappelle que la position de la caméra est donnée par le point *CamPos*, et que son orientation est définie par les 2 vecteurs *CamVec* et *CamUp* qui déterminent la ligne de visée et le plan de visée.

Pour faciliter le positionnement de la caméra, on dispose maintenant de la commande **SetCamView** qui oriente la caméra vers un point fixé et qui recalcule le vecteur *CamUp* pour avoir une base orthonormale du plan de visée.



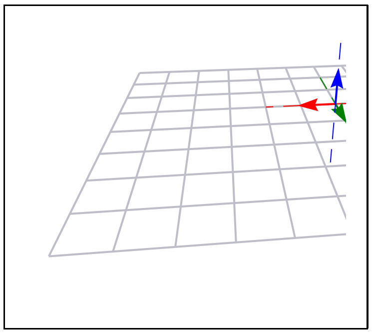
source jps

```
-6 2 setxrange
-5 2 setyrange
15 setxunit
3 10 5 SetCamPos
0 0 0 SetCamView
qplanxy
-2 2 1 axesRVB
```



source jps

```
-6 2 setxrange
-5 2 setyrange
15 setxunit
3 10 5 SetCamPos
0 60 cos 60 sin SetCamUp
0 0 0 SetCamView
qplanxy
-2 2 1 axesRVB
```



source jps

```
-6 2 setxrange
-5 2 setyrange
15 setxunit
3 10 5 SetCamPos
2 0 0 SetCamView
qplanxy
-2 2 1 axesRVB
```

De plus quelques nouvelles commandes sont venues enrichir la librairie 3d :

M **SetCamView** $- \longrightarrow$ Oriente la visée de la caméra vers le point $M(x,y,z)$ et recalcule tous les vecteurs nécessaires

$xmin$ $xmax$ **setxrange3d** $- \longrightarrow$ affecte les variables $xmin3d$ et $xmax3d$ définissant l'intervalle de travail sur l'axe Ox en $3d$

$ymin$ $ymax$ **setyrange3d** $- \longrightarrow$ affecte les variables $ymin3d$ et $ymax3d$ définissant l'intervalle de travail sur l'axe Oy en $3d$

$zmin$ $zmax$ **setzrange3d** $- \longrightarrow$ affecte les variables $zmin3d$ et $zmax3d$ définissant l'intervalle de travail sur l'axe Oz en $3d$

$-$ **qplanxz** $- \longrightarrow$ effectue un quadrillage du plan xOz

$-$ **qplanyz** $- \longrightarrow$ effectue un quadrillage du plan yOz

$[\,A_0 \ldots A_n\,]$ f **papply3d** $[\,b_0 \ldots b_n\,]$ ou $- \longrightarrow$ construit un nouveau tableau en répétant, pour i variant de 0 à n, l'opération suivante : déposer le point A_i puis exécuter f. Si à la fin de cette opération le tableau est vide, alors il est enlevé de la pile.

$[\,A_0 \ldots A_n\,]$ **isobarycentre3d** $G \longrightarrow$ le point G est le barycentre du système $[(A_0,1);\ldots;(A_n,1)]$

$[\,A\ a\ B\ b\,]$ **barycentre3d** $G \longrightarrow$ le point G est le barycentre du système $[(A,a);(B,b)]$

$M\ A\ \alpha$ **hompoint3d** $M' \longrightarrow M'$ est l'image de M par l'homothétie de centre A et de rapport α

$M\ A$ **sympoint3d** $M' \longrightarrow M'$ est l'image de M par la symétrie de centre A

$M\ u$ **translatepoint3d** $M' \longrightarrow M'$ est l'image de M par la translation de vecteur $\vec{u}$

$x\ y\ z\ k_1\ k_2\ k_3$ **scaleOpoint3d** $k_1x\ k_2y\ k_3z \longrightarrow$ opère une « dilatation » des coordonnées du point $M(x,y,z)$ sur les axes Ox, Oy et Oz suivant les facteurs k_1, k_2 et k_3

$M\ \alpha_x\ \alpha_y\ \alpha_z$ **rotateOpoint3d** $M' \longrightarrow M'$ est l'image de M par la rotation de centre O et d'angles respectifs $\alpha_x\ \alpha_y\ \alpha_z$ sur les axes Ox, Oy, Oz

$x\ y\ z$ **3dto2d** $X\ Y \longrightarrow$ calcule les coordonnées du point projeté sur l'écran pour la représentation 3d. cette commande est synonyme de la commande **CamView**

4. Solides

Ce paragraphe présente un ensemble de macros dont le but est la manipulation et la représentation de solides convexes. Un effort particulier a été fait pour respecter la philosophie de la programmation « orientée objet », avec contrôle de typage et messages d'erreurs adéquats.

Ce travail est basé sur les travaux de Manuel Luque [1] et de Christophe Poulain [2] ainsi que le cours d'infographie du Prof. Daniel Thalmann dans la section « *Computer Graphics* » du *Virtual Reality Lab* [3] .

4.1 - Le type *solid*

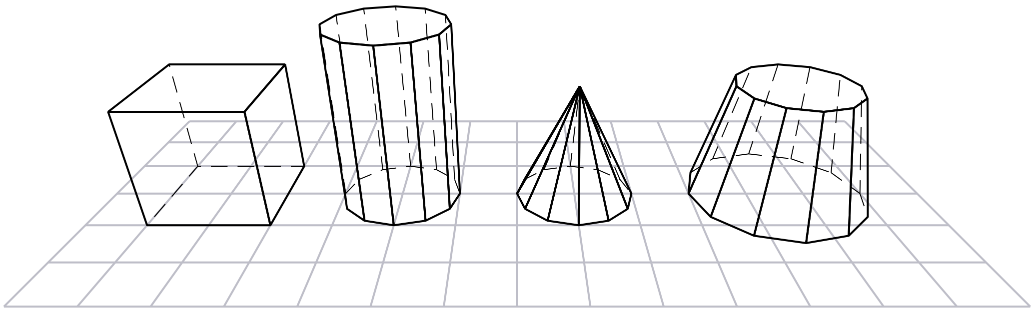
Le type *solid* est un type à plusieurs composantes. Les deux composantes essentielles sont :

- Un tableau des coordonnées (3d) des sommets du solides. Ce tableau est indexé à partir de l'indice 0.
- Un tableau des *faces* du solide. Ce tableau est indexé à partir de l'indice 0.

Et l'on désigne par *face* un tableau des indices des sommets de la face considérée, avec la convention importante suivante : les **sommets sont rangés dans l'ordre trigonométrique** si l'on regarde la face considérée depuis l'extérieur du solide.

4.2 - Solides précalculés

Certains solides simples sont précalculés : le cube, le cylindre, le cône, le tronc de cône, la sphère, le tétraèdre, l'octaèdre, l'icosaèdre, et le dodécaèdre.



Voici la liste des commandes associées :

a **newcube** *solid* $\longrightarrow$ crée un nouveau cube, de type *solid*, de centre O, d'arête a

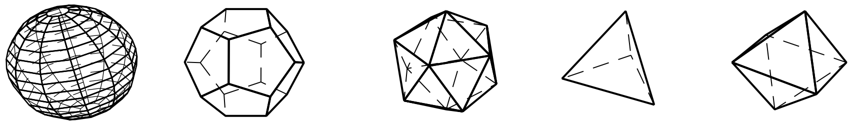
$z_0\ r\ z_1$ **newcylindre** *solid* $\longrightarrow$ crée un nouveau cylindre, de type *solid*, d'axe Oz, de rayon r, allant du plan $z = z_0$ jusqu'au plan $z = z_1$

$z_0\ r\ z_1$ **newcone** *solid* $\longrightarrow$ crée un nouveau cône, de type *solid*, d'axe Oz, de rayon r, allant du plan $z = z_0$ jusqu'au plan $z = z_1$

$z_0\ r_0\ z_1\ r_1$ **newtronccone** *solid* $\longrightarrow$ crée un nouveau tronc de cône, de type *solid*, d'axe Oz, de rayon de base r_0 (sur le plan $z = z_0$) et de rayon au sommet r_1 (sur le plan $z = z_1$)

---

[1] **melusine.eu.org/syracuse/mluque/pst-v3d/**
[2] **melusine.eu.org/syracuse/poulecl/macros/**
[3] **vrlab.epfl.ch/teaching/teaching_index.html**



sphere **dodecaedre** **icosaedre** **tetraedre** **octaedre**

r $\{mode\}$ **newsphere** *solid* $\longrightarrow$ crée une nouvelle sphère, de type *solid*, de centre O, de rayon r. Le paramètre *mode* $\in \{0, 1, 2, 3, 4\}$ est optionnel; il indique le niveau de résolution souhaité (0 = mini, 4 = maxi)

– **newtetraedre** *solid* $\longrightarrow$ crée un nouveau tétraèdre régulier, de type *solid*

– **newoctaedre** *solid* $\longrightarrow$ crée un nouvel octaèdre régulier, de type *solid*, de centre O

– **newicosaedre** *solid* $\longrightarrow$ crée un nouvel isocaèdre régulier, de type *solid*, de centre O

– **newdodecaedre** *solid* $\longrightarrow$ crée un nouveau dodécaèdre régulier, de type *solid*, de centre O

Pour la sphère, plusieurs modes de résolution sont proposés par un argument optionnel. Le mode par défaut est stocké dans la variable $defaultsolidmode$. Ceci permet de préparer son image en basse résolution (mode 0), et de ne calculer la résolution souhaitée, lourde en temps de calcul, qu'au dernier moment.

À terme, il est prévu de proposer un tel mode pour le cylindre, le cône et le tronc de cône.

- $defaultsolidmode$: mode de résolution par défaut pour les solides. **valeur par défaut :** 2

4.3 - Dessiner un solide

Pour dessiner un solide, on dispose des commandes **drawsolid** et **drawsolid***, cette dernière réalisant en plus le coloriage des faces visibles.

solid **drawsolid** $-\longrightarrow$ dessine le solide *solid*

solid **drawsolid*** $-\longrightarrow$ dessine le solide *solid* avec coloriage des faces visibles

On dispose de plus d'un booléen permettant d'activer ou non la représentation des arêtes cachées.

- *aretescachees* : booléen indiquant à la procédure **drawsolid** si l'on doit ou non représenter les arêtes cachées. **valeur par défaut :** $true$

Par défaut, et comme à l'habitude, la couleur de coloriage des faces est définie par $fillstyle$. Pour une scène non éclairée (voir plus loin), 2 autres méthodes sont prévues pour spécifier la couleur d'une face donnée :

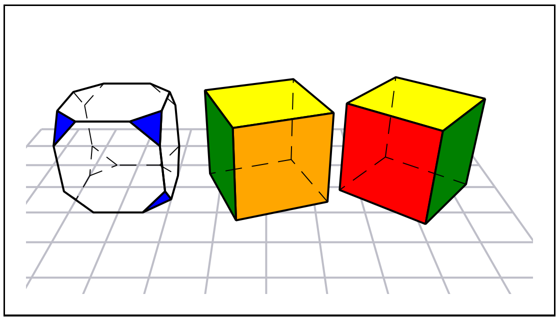
- on spécifie, face par face, la couleur souhaitée avec la commande **solidputfcolors**. La couleur est spécifiée par une chaîne de caractères.
- on spécifie la couleur souhaitée en fonction du nombre de côtés des faces. On utilise pour cela la commande **solidputncolors**.

Pour chacune de ces méthodes, on transmet comme argument un tableau comportant des chaînes de caractères désignant la couleur souhaitée. Ce tableau est ensuite stocké comme un constituant du solide. Pour un solide donné, on obtient ces tableaux avec les commandes **solidgetfcolors** et **solidgetncolors**. L'élement d'indice i du tableau $fcolor$ correspond à la face d'indice i du solide, et l'élement d'indice i du tableau *ncolors* correspond à la couleurs des faces à $3 + i$ côtés.

Dans l'exemple ci-dessous, on dessine un cube tronqué, dont on précise que les faces triangulaires (à 3 côtés) doivent êtres colorées en bleu. On définit ensuite un nouveau cube, que l'on stocke dans la variable *moncube*, dont on définit la couleur pour chacune des 6 faces. On représente ensuite 2 instances de ce cube, l'une ayant subi une rotation de 120° autour de l'axe Oz.

Dans cet exemple, on pourra remarquer un phénomène étonnant au premier abord : dans la variable *moncube* est stocké le cube de centre O et d'arête 2. On fait ensuite subir à ce solide une rotation de 20° autour de l'axe Oz, puis une translation de vecteur $\vec{k}$ (vecteur unitaire dirigeant l'axe Oz). La variable *moncube* désigne alors, non pas le cube d'origine, mais le cube transformé. Ce phénomène s'explique par le fait que pour les objets complexes (les solides, mais aussi les matrices, les tableaux, etc...), ce n'est pas l'objet que postscript dépose sur la pile, mais une référence à l'objet (un *pointeur* comme l'on dit en C, en Pascal ou en ADA). Or les transformations effectuées agissent sur l'objet lui-même...

Si l'on veut éviter ce phénomène, et si l'on souhaite avoir 2 solides identiques, mais avec des instances séparées, on utilisera la commande **dupsolid** qui construit une nouvelle instance, copie conforme de celle passée en argument.



source jps

```
-4.5 5 setxrange
-2 3 setyrange
20 setxunit
0 10 6 SetCamPos
0 0 1 SetCamUp
0 0 0 SetCamView
qplanxy
2 setlinejoin

2 newcube 3 tronque_cube
dup [(bleu)] solidputncolors
{3 0 1 translatepoint3d} solidtransform
drawsolid*

/moncube
   2 newcube dup
   [(jaune) (orange) (bleu) (rouge)
      (vert) (0 0 0 1 setcymkcolor)]
   solidputfcolors
def
moncube
   {0 0 20 rotateOpoint3d} solidtransform
   {0 0 1 translatepoint3d} solidtransform
drawsolid*
moncube
   {0 0 120 rotateOpoint3d} solidtransform
   {-3 0 0 translatepoint3d} solidtransform
drawsolid*
```

solid array **`solidputncolors`** – ⟶ affecte au solide *solid* le tableau *array* en tant que tableau des couleurs des faces à *n* côtés

solid **`solidgetncolors`** *array* ⟶ le tableau *array* est le tableau des couleurs des faces à *n* côtés pour le solide *solid*

solid array **`solidputfcolors`** – ⟶ affecte au solide *solid* le tableau *array* en tant que tableau des couleurs des faces

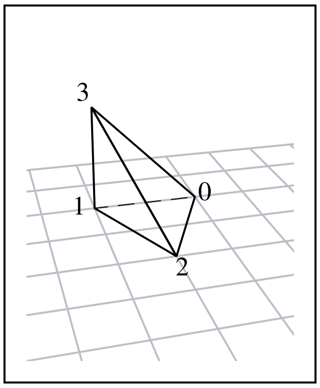
solid **`solidgetfcolors`** *array* ⟶ le tableau *array* est le tableau des couleurs des faces pour le solide *solid*

4.4 - Générer un solide

Un solide peut être *monoface*, auquel cas il sera visible du côté de sa normale, et invisible du côté « arrière ». Il peut également être *biface*, autrement dit aplati.

Plusieurs méthodes pour générer un solide :

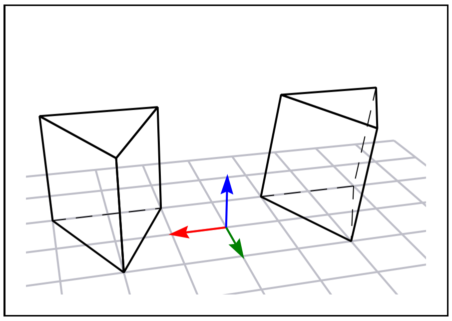
- On crée un tableau des coordonnées des sommets, puis le tableau des faces, qui est lui-même un tableau de tableaux, ces derniers contenant, pour une face donnée, les indices des sommets de la face, rangés par ordre trigonométrique lorsque l'on regarde le solide de l'extérieur. On dépose alors les 2 tableaux (sommets et faces) sur la pile et on invoque la fonction **`generesolid`**.



source jps

```
-2 2 setxrange
-2 3 setyrange
25 setxunit
3 10 6 SetCamPos
0 0 0 SetCamView
qplanxy
2 setlinejoin
[
   -1 -1 0 %% sommet 0
    1 -1 0 %% sommet 1
    0  1 0 %% sommet 2
    1 -1 2 %% sommet 3
]
[
   [2 1 0]
   [0 1 3]
   [1 2 3]
   [2 0 3]
]
generesolid
dup drawsolid
solidnumsommets
```

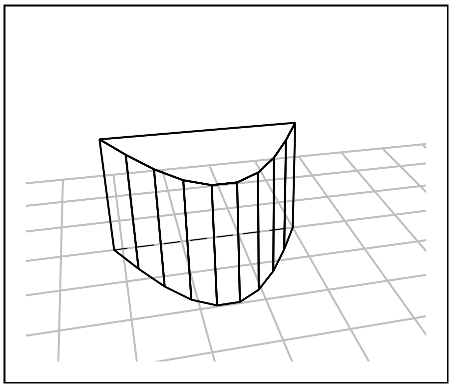
- On construit un tableau des coordonnées 2d des sommets d'une face sur le plan xOy, puis on invoque l'une des fonctions **`generemonoface`**, **`generebiface`**, **`genereprismedroit`** ou **`genereprisme`**.



source jps

```
-3 3 setxrange
-1 3 setyrange
25 setxunit
3 10 6 SetCamPos
0 0 0 SetCamView
qplanxy
0 1 1 axesRVB
2 setlinejoin
/table [-1 -1 1 -1 0 1] def
table 0 2 genereprismedroit
{2 0 0 translatepoint3d} solidtransform
drawsolid
table 0 2 -.25 -.25 1 genereprisme
{-2 0 0 translatepoint3d} solidtransform
drawsolid
```

Remarque : Pour cette dernière méthode, le tableau décrivant la face initiale peut-être obtenue du chemin continu courant en utilisant la commande *currentcpathpointstable*. Par exemple, voici un prisme droit généré à partir de la courbe $y = 2 \sin x$:



source jps

```
-4 2 setxrange
-2 3 setyrange
25 setxunit
3 10 6 SetCamPos
0 0 0 SetCamView
qplanxy
2 setlinejoin
10 setresolution
newpath
   pi 0 smoveto
   pi 0 {Sin 2 mul} Courbe_
currentcpathpointstable 0 2 genereprismedroit
drawsolid
```

En résumé :

array$_1$ *array*$_2$ **generesolid** *solid* $\longrightarrow$ construit un solide dont le tableaux des sommets est *array*$_1$ et le tableau des faces est *array*$_2$

array **generemonoface** *solid* $\longrightarrow$ construit un solide monoface, dans le plan xOy, à partir du tableau de points *array*. Les points sont en 2d, et rangés de manière à décrire l'orientation de la face (sens trigonométrique $\Rightarrow$ face de normale Oz, dans le sens des z croissants)

array **generebiface** *solid* $\longrightarrow$ construit un solide biface, dans le plan xOy, à partir du tableau de points *array*. Les points sont en 2d, et rangés de manière à décrire l'orientation de la face (sens trigonométrique $\Rightarrow$ face de normale Oz, dans le sens des z croissants)

array z_0 z_1 **genereprismedroit** *solid* $\longrightarrow$ construit un prisme droit d'axe Oz à partir du tableau de points *array*. Les points sont en 2d, et rangés de manière à décrire l'orientation de la face (sens trigonométrique $\Rightarrow$ face de normale Oz, dans le sens des z croissants). La base du prisme est sur le plan $z = z_0$ et sa face supérieure sur le plan $z = z_1$

array z_0 z_1 $\vec{u}$ **genereprisme** *solid* $\longrightarrow$ construit un prisme oblique d'axe $(O, \vec{u})$ à partir du tableau de points *array*. Les points sont en 2d, et rangés de manière à décrire l'orientation de la face (sens trigonométrique $\Rightarrow$ face de normale Oz, dans le sens des z croissants). La base du prisme est sur le plan $z = z_0$ et sa face supérieure sur le plan $z = z_1$

4.5 - Opérations sur les solides

La commande **solidtransform** à qui l'on passe en argument une transformation $\mathbb{R}^3 \to \mathbb{R}^3$, applique cette tranformation à chacun des sommets du solide considéré. Ceci permet notamment, par le biais des commandes **translatepoint3d** et **rotate0point3d**, de positionner un solide dans l'espace, mais aussi, par l'intermédiaire de **scale0point3d**, de changer ses dimensions.

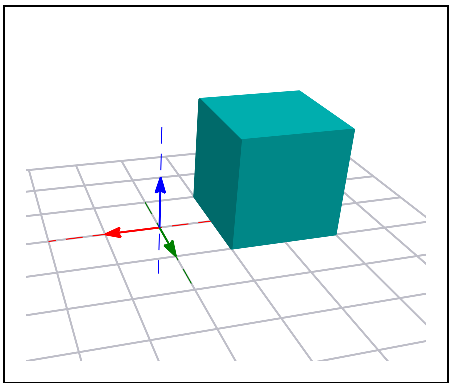
solid $\{f\}$ **solidtransform** *solid* $\longrightarrow$ applique la transformation f au solide *solid*, f étant une application $\mathbb{R}^3 \longrightarrow \mathbb{R}^3$

Pour les cubes, on dispose de la commande **tronque_cube**, qui permet d'obtenir un nouveau solide en coupant les 8 coins du cube d'origine suivant une proportion que l'on transmet en argument (2 $\Rightarrow$ on coupe au milieu de l'arête, 3 $\Rightarrow$ au tiers, 4 $\Rightarrow$ au quart, etc...)

solid$_1$ n **tronque_cube** *solid*$_2$ $\longrightarrow$ *solid*$_2$ est le solide obtenu en coupant chaque coin du parallélépipède *solid*$_1$. n est un réel supérieur à 2 indiquant la proportion à respecter pour la tronquature

4.6 - Éclairage par une source lumineuse ponctuelle

Il est possible d'éclairer la scène par une source ponctuelle d'une couleur et d'une intensité donnée. Dans ce cas, seule cette dernière est prise en compte pour le coloriage des faces des solides présents. L'intensité de la couleur d'une face dépend de l'inclinaison de la face et de sa distance par rapport à la source lumineuse.



source jps

```
-2 4 setxrange
-2 3 setyrange
25 setxunit
3 10 6 SetCamPos
0 0 0 SetCamView

2 setlinejoin
qplanxy
-1 2 1 axesRVB

{cyan} setlight
2 4 2 setlightsrc

/aretescachees false def
2 newcube
{-2 0 1 translatepoint3d} solidtransform
drawsolid*
```

array **setlight** $- \longrightarrow$ Affecte la couleur de la source lumineuse ponctuelle dans l'espace RGB ou CYMK, *array* étant un tableau de 3 ou 4 réels de l'intervalle [0; 1]

$\{f\}$ **setlight** $- \longrightarrow$ Éxécute la fonction f dans un **gsave .. grestore**, y récupère la définition de la couleur dans l'espace RGB, puis l'affecte à la couleur de la source lumineuse ponctuelle.

i **setlightintensity** $-\longrightarrow$ Affecte l'intensité de la source lumineuse ponctuelle

x y z **setlightsrc** $-\longrightarrow$ Affecte la position de la source lumineuse ponctuelle

- *lightintensity* : Intensité de la source lumineuse ponctuelle. **valeur par défaut :** 1

4.7 - Boîte à outils

Ci-dessous une liste des autres fonctions et procédures disponibles concernant le type *solid* :

$-$ **newsolid** *solid* $\longrightarrow$ dépose le solide nul sur la pile

solid **emptysolid** *bool* $\longrightarrow$ *bool* vaut *true* si le solide est vide, *false* sinon

any **issolid** *bool* $\longrightarrow$ *bool* vaut *true* si *any* est de type *solid*, *false* sinon

solid i **solidgetsommetsface** *array* $\longrightarrow$ *array* est le tableau des sommets de la face d'indice *i* du solide *solid*

solid i j **solidgetsommetface** *S* $\longrightarrow$ *S* est le sommet d'indice *i* de la face d'indice *j* du solide *solid*

solid i **solidgetsommet** *S* $\longrightarrow$ *S* est le sommet d'indice *i* du solide *solid*

solid **solidgetpointstable** *array* $\longrightarrow$ *array* est le tableau des sommets du solide

solid **solidgetfaces** *array* $\longrightarrow$ *array* est le tableau des faces du solide

solid i **solidgetface** *array* $\longrightarrow$ *array* est le tableau décrivant la face *i* du solide (tableau d'indices de sommets)

solid array **solidputpointstable** *solid* $\longrightarrow$

solid array **solidputfaces** *array* $\longrightarrow$

solid i **solidfacevisible?** *bool* $\longrightarrow$ *bool* vaut *true* si la face d'indice *i* du solide *solid* est visible

solid i **solidnormaleface** $\vec{u} \longrightarrow \vec{u}$ est un vecteur normal à la face d'indice *i* du solide *solid*. Le vecteur est orientaté vers l'extérieur du solide

solid i **solidcentreface** *G* $\longrightarrow$ le point *G* est le centre de la face d'indice *i* du solide *solid*

solid **solidnombresommets** *n* $\longrightarrow$ nombres de sommets du solide *solid*

solid **solidnombrefaces** *n* $\longrightarrow$ nombres de faces du solide *solid*

solid **solidnumsommets** $-\longrightarrow$ affiche en face de chaque sommet du solide *solid* son indice dans le tableau des sommets

solid **dupsolid** *solid solid* $\longrightarrow$ crée une nouvelle instance du solide déposé sur la pile