

Aide de la bibliothèque `biblioLiaisons3d2d` v.1.0.6

Anthony Meurdefroid

1^{er} août 2024

Table des matières

1	Description de la bibliothèque	3
1.1	Que fait cette bibliothèque ?	3
1.2	Installation	3
1.2.1	Procédure	3
1.2.2	Éditeur WYSIWYG	3
1.2.3	Compilation intégrée VSCode	3
1.3	Méthodologie	4
2	Liste des commandes de la bibliothèque	4
2.1	Les objets	4
2.2	Points, bases et CEC	5
2.2.1	Points	5
2.2.2	Bases	5
2.2.3	CEC	5
2.3	Les liaisons	6
2.3.1	Glissière	6
2.3.2	Pivot	6
2.3.3	Helicoïdale	7
2.3.4	Pivot glissant	8
2.3.5	Rotule à doigt	9
2.3.6	Rotule	10
2.3.7	Appui-plan	11
2.3.8	Linéaire annulaire	11
2.3.9	Linéaire rectiligne	12
2.3.10	Ponctuelle	13
2.4	Transmission de puissance	14
2.4.1	Engrenages	14
2.4.2	Poulie courroie	16
2.4.3	Roue et vis sans fin	17
2.4.4	Pignon crémaillère	17
2.4.5	Chaîne	18
2.5	Habillage	19
2.5.1	Relier	19
2.5.2	Bâti	20
2.5.3	Cylindre	20
2.5.4	Disque	21
2.5.5	Surface conique tronquée	22
2.5.6	Surface cylindrique	23
2.5.7	Sphère	23
2.5.8	U-shape	24

2.5.9	Surface plane	25
2.5.10	Triangulation STL	26
2.6	Paramétrage	26
2.6.1	Bases	27
2.6.2	Paramétrage	29
2.6.3	Texte	32
2.7	Éléments technologiques	36
2.7.1	Ressort de traction/compression	36
3	Animation	36
3.1	Boîtes englobantes	37
3.2	Animation par le package <code>animate</code>	37
3.3	Animation par un script python	37
4	Exemples détaillés	37
4.1	Le système bielle-manivelle	37
4.2	Usage avancé : couplage avec Sympy – direction de camion	39
5	Pour aller plus loin	40
5.1	VSCode	40
5.1.1	Snippets	40
5.1.2	Run bat file from vscode	40
5.2	Rotule à doigt	40
6	Commandes	41

1 Description de la bibliothèque

1.1 Que fait cette bibliothèque ?

Objectifs Cette bibliothèque a pour objectif de tracer en 3d et 2d des schémas cinématiques avec les mêmes commandes. Les différents formats de sortie sont notamment `pdf` et `png`. C'est une bibliothèque utilisant le langage Asymptote, par conséquent elle est parfaitement intégrable à \LaTeX . De plus, comme on code le schéma cinématique, vous pourrez animer vos schémas.

Limitations Le défaut principal d'asymptote serait en autre sa lenteur. Mais surtout que la projection 2d d'une construction 3d n'est pas vectorielle. La 2d quant à elle est bien vectorielle. Après les images sont quand même de bonne voire de très bonne qualité (voir l'ensemble des exemples). Néanmoins, elles prennent plus de mémoire. Limitation supplémentaire, il faut une \LaTeX pour en profiter.

1.2 Installation

1.2.1 Procédure

Pour éviter tout problème, je vous conseille de respecter cette procédure :

1. installer Asymptote – le langage utilisé <https://asymptote.sourceforge.io/> – et ne pas utiliser la version dans votre distribution latex (en tout cas pas celle sous Miktex) ;
2. installer la dernière version de ghostscript <https://ghostscript.com/releases/gsdnld.html>.

Pour un essai « rapide » de vérification après installation :

1. trouver le chemin de l'exécutable `asy.exe` nouvellement installé (pour moi `"C:/Program Files/Asymptote/asy.exe"`) ;
2. dans ce répertoire, copier coller le contenu du dossier `src` : `biblioLiaisons.asy`, `rotuleCreuxDoigt.stl`, `rotuleCreuxDoigtRapide.stl` et `STLforBLM.asy` ;
3. créer un dossier `myTest` ;
4. copier-coller un exemple (par exemple, les fichiers `BM_iso.asy` et `BM_structure.asy` dans le dossier `./examples/bielle_manivelle`) dans `myTest` ;
5. sous Windows, en utilisant la console `cmd` :
 - (a) se placer dans le dossier `myTest` `cd ./path/to/folder/myTest` ;
 - (b) executer `"C:/Program Files/Asymptote/asy.exe"-f pdf -noView BM_iso.asy` ;
 - (c) le `pdf` contenant la figure créée doit s'ouvrir.


1.2.2 Éditeur WYSIWYG

Le langage asymptote n'a pas besoin d'une distribution \LaTeX à ma connaissance. Mais il peut intégrer des commandes \LaTeX comme certaines de ce package (tout ce qui est lié aux chiffres et aux nombres). Il est donc possible pour les utilisateurs de Word de générer le schéma au format `png` et par exemple d'inclure cette image dans leur document et de rajouter à la main les descriptifs des points, des classes d'équivalence cinématique et des paramètres ; ou de modifier les fonctions du package.

Sinon, il existe tout ce qu'il faut en ligne : <http://asymptote.ualberta.ca/>. Néanmoins à ce jour d'écriture, le workspace ne semble pas fonctionner correctement rendant la manipulation de fichiers impossible. Il est sans doute possible de copier/coller l'ensemble des fonctions de la bibliothèque et de vérifier le fonctionnement (non réalisé à ce jour et impossible pour la commande `rotule` à `doigt`).

1.2.3 Compilation intégrée VSCode

Enfin pour l'étape de compilation, deux solutions pour moi sous VSCode :

- créer un fichier `bat` (sous windows évidemment) et exécuter ce fichier `.bat` en appuyant sur  (voir dans le dossier `.vscode`, le fichier `launch.json`) ;

- créer une `task` (utilisant le pdf viewer de VSCode) dont un exemple est également fourni dans le dossier `.vscode`, fichier `tasks.json`.

1.3 Méthodologie

Pour éviter de passer des heures à ajuster (ce qui peut être très ennuyant), je vous conseille la stratégie suivante :

- faire un beau schéma cinématique à la main ; avec les idées claires sur les coordonnées des points, ou du moins comment les obtenir par construction géométrique. Ne pas oublier qu'Asymptote est un outil de construction, il y a donc tous les outils pour translation, pivoter, agrandir etc.
- avoir toutes les longueurs et les bases utiles.
- avoir toutes les lois entrée-sortie. Ce n'est clairement pas indispensable en vrai, mais c'est tout de même plus simple la plupart du temps et indispensable pour réaliser des animations correctes.
- commencer à coder. Partir du fichier avec la structure de base :
 - préciser les variables ;
 - préciser les points et les bases ;
 - préciser les CEC ;
 - placer les liaisons ;
 - tracer la base 0 pour visualisation ;
 - enfin générer une première fois !
- corriger les erreurs éventuelles ;
- améliorer en reliant les CEC – générer – corriger ;
- habiller l'ensemble – générer – corriger ;

Le temps de compilation peut être très rapide en 2d à très lent en 3d (et si en plus il y a des rotules ou pires des rotules à doigt.. voir 5.2). Pour éviter de se décourager, essayer de respecter ces consignes ou du moins trouver rapidement votre façon de faire.

2 Liste des commandes de la bibliothèque

2.1 Les objets

Le langage Asymptote est inspiré du langage C/C++. Il faut déclarer les variables avec leur type :

- `real` : pour les nombres réels.
Ex : `real a = 3.1 ;`.
- `int` : pour les nombres entiers.
Ex : `int a = 3 ;`.
- `triple` : pour les coordonnées des points et les vecteurs.
Ex : `triple A = (0,1,0);` – le point de coordonnées (0,1,0).
Ex : `triple ex = (1,0,0);` – le vecteur directeur (1,0,0).
- `basis` : objet de la bibliothèque `biblioLiaisons`. Par défaut la base `b0` ainsi que le point O (0,0,0) sont définis.
Ex : `basis b1 = rotationBasis(1, b0, theta10, 'z', b0.z);` – pour créer une base par rotation.
- `pen` : pour la gestion des tracés (couleur, épaisseur, type de trait...).
Ex : `pen CEC0 = black + 1 + dashed ;`.
- `path3` : l'objet chemin. En fait tous les tracés sont des objets `path3`. À retenir dès maintenant pour construire le chemin entre deux points, il faudra utiliser `--`.
Exemple : `path3 line = A -- B ;`.

2.2 Points, bases et CEC

2.2.1 Points

Utilisation de l'objet `triple`.

Ex : `triple A = 0 + 3*b0.y` ; – création du point A à une distance 3 suivant \vec{y}_0 de O .

2.2.2 Bases

Construction. La fonction `rotationBasis` permet de créer une base par rotation autour d'un axe.

```
basis rotationBasis(int number, basis parent, real theta, string axis,
triple axisShared) ;
```

- `int number` : entier pour le numéro de la base
- `basis parent` : la base parent
- `real theta` : la valeur de l'angle en radians
- `string axis` : le caractère '`x`' - '`y`' ou '`z`' pour préciser autour de quel axe tourne la nouvelle base
- `triple axisShared` : l'axe de la base parent en commun

Ex : `basis b1 = rotationBasis(1, b0, theta10, 'z', b0.z);`.

— <code>Pens.\bielleManivelle</code>	— <code>Pens.\DAE_anim</code>	— <code>Pens.\jointCardan</code>	— <code>Pens.\pinceCoupeCable</code>
— <code>Pens.\bielleManivelle_anim</code>	— <code>Pens.\directionCamion_anim</code>	— <code>Pens.\limiteurCouple</code>	— <code>Pens.\pompePistonsAxiaux</code>
— <code>Pens.\concasseur</code>	— <code>Pens.\falconHaptic</code>	— <code>Pens.\maxpid</code>	— <code>Pens.\SDP</code>
— <code>Pens.\croixMalte</code>	— <code>Pens.\falconHaptic_anim</code>	— <code>Pens.\maxpid_anim</code>	— <code>Pens.\SDP_anim</code>
— <code>Pens.\croixMalte_anim</code>	— <code>Pens.\faucheuse</code>	— <code>Pens.\pilote5000</code>	— <code>Pens.\sinusmatic</code>
— <code>Pens.\DAE</code>	— <code>Pens.\forcebat</code>	— <code>Pens.\pilote5000_anim</code>	— <code>Pens.\sinusmatic_anim</code>

Il existe une deuxième commande de création de base au besoin :

```
basis createBasis(int number, triple x, triple y) ;
```

- `int number` : entier pour le numéro de la base
- `triple x` : cet axe deviendra le vecteur directeur \vec{e}_x
- `triple y` : deuxième axe directeur (orthonormée par Gram-Schmidt au besoin $\rightarrow \vec{e}_y$)

Remarque : le vecteur directeur \vec{e}_z est construit par produit vectoriel $\vec{e}_x \wedge \vec{e}_y$.

```
— Pens.\jointCardan
```

Utilisation. L'objet `base` a des attributs `x`, `y` et `z`. Par exemple `b0.x` retourne le vecteur directeur \vec{e}_x défini par le triplé `(1,0,0)`..

Par conséquent, on peut positionner des points de manière physique : `triple A = 0 + R*b1.x` par exemple ou bien toutes les lois entrées sortie. Les fonctions mathématiques usuelles sont disponibles.

2.2.3 CEC

Enfin pour les classes d'équivalence cinématique, je conseille simplement d'associer un stylo de couleur à chacune. Par exemple : `pen CEC1 = orange;`. Toutes les couleurs disponibles de base sont sur le site d'asymptote, documentation, chapitre 6.3 <https://asymptote.sourceforge.io/doc/Pens.html>.

2.3 Les liaisons

L'ensemble des fonctions ne font que tracer. Mais elles sont identiques en 2d ou en 3d en fonction du point d'observation.

Toutes les illustrations ci-dessous commencent par le code suivant :

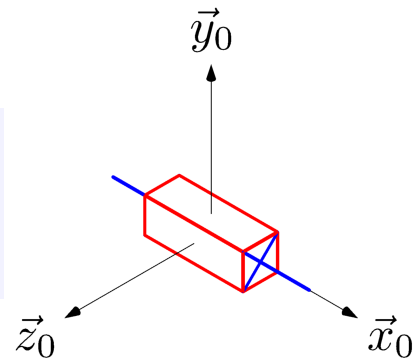
```
1 settings.render = -4 ;
2 settings.prc = false ;
3 import biblioLiaisons ;
4 defaultpen(fontsize(10pt)) ;
5 unitsize(1cm) ;
```

2.3.1 Glissière

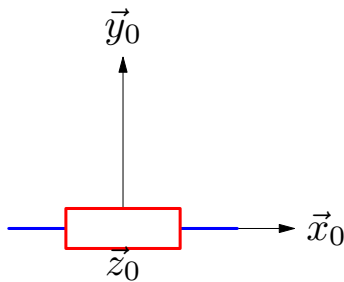
BIB `liaisonGlissiere(triple center, triple direction, triple orientation, pen c1, pen c2) ;`

- `triple center` : le centre de la liaison
- `triple direction` : direction de la liaison
- `triple orientation` : l'orientation d'un côté court du parallélépipède
- `pen c1` : mise en forme de la classe 1 (parallélépipède)
- `pen c2` : mise en forme de la classe 2 (axe + croix)

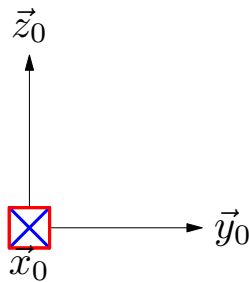
```
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 liaisonGlissiere(0,b0.x, b0.y, red, blue) ;
```



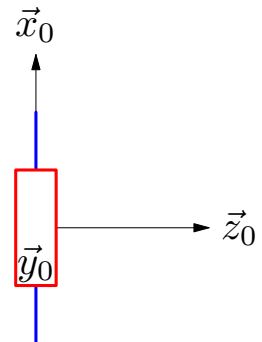
Perspective isométrique



Plan $(0, \vec{x}, \vec{y})$



Plan $(0, \vec{y}, \vec{z})$



Plan $(0, \vec{z}, \vec{x})$

— <code>⊞.\bielleManivelle</code>	— <code>⊞.\DAE</code>	— <code>⊞.\limiteurCouple</code>	— <code>⊞.\SDP</code>
— <code>⊞.\bielleManivelle_anim</code>	— <code>⊞.\DAE_anim</code>	— <code>⊞.\pilote5000</code>	
	— <code>⊞.\fauchouse</code>	— <code>⊞.\pilote5000_anim</code>	— <code>⊞.\SDP_anim</code>

2.3.2 Pivot

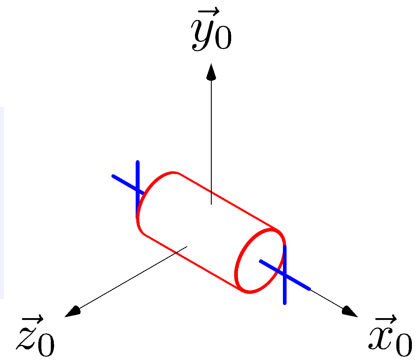
BIB `liaisonPivot(triple point, triple axis, triple stopAxis, pen c1, pen c2) ;`

- `triple point` : le centre de la liaison
- `triple axis` : l'axe de la liaison
- `triple stopAxis` : l'orientation des arrêts axiaux dans la représentation
- `pen c1` : mise en forme de la classe 1 (cylindre)
- `pen c2` : mise en forme de la classe 2 (axe)

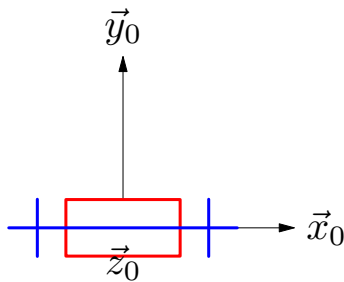
```

1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 liaisonPivot(0,b0.x, b0.y, red, blue) ;

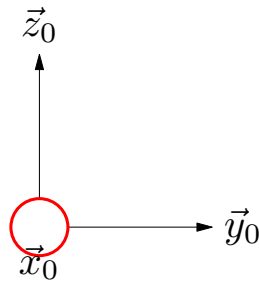
```



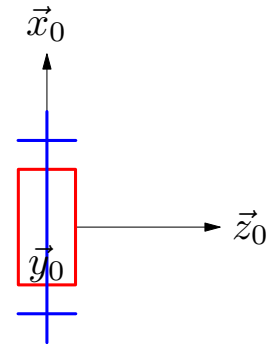
Perspective isométrique



Plan $(0, \vec{x}, \vec{y})$



Plan $(0, \vec{y}, \vec{z})$



Plan $(0, \vec{z}, \vec{x})$

Remarque : malgré l'orientation des arrêts axiaux suivant \vec{y}_0 le code s'adapte en 2d et propose une correction pour voir ces arrêts dans la dernière vue.

— <code>\bielleManivelle</code>	— <code>\directionCamion_anim</code>	— <code>\jointCardan</code>	— <code>\pompePistonsAxiaux</code>
— <code>\bielleManivelle_anim</code>	— <code>\falconHaptic</code>	— <code>\maxpid</code>	— <code>\SDP</code>
— <code>\concasseur</code>	— <code>\falconHaptic_anim</code>	— <code>\maxpid_anim</code>	— <code>\SDP_anim</code>
— <code>\croixMalte</code>	— <code>\faucheuse</code>	— <code>\pilote5000</code>	— <code>\sinusmatic</code>
— <code>\croixMalte_anim</code>	— <code>\forcebat</code>	— <code>\pilote5000_anim</code>	— <code>\sinusmatic_anim</code>
— <code>\DAE</code>	— <code>\I3D</code>	— <code>\piloteSafran</code>	— <code>\sinusmatic_anim</code>
— <code>\DAE_anim</code>	— <code>\I3D_anim</code>	— <code>\pinceCoupeCable</code>	— <code>\trainEpicycloidaux</code>

2.3.3 Helicoïdale

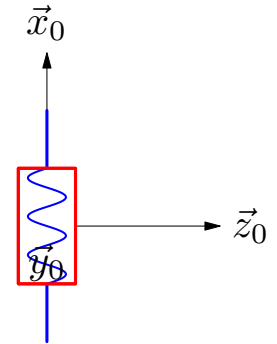
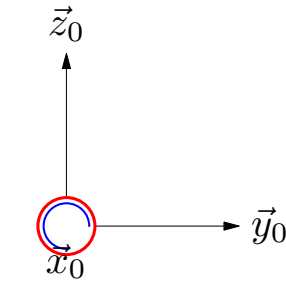
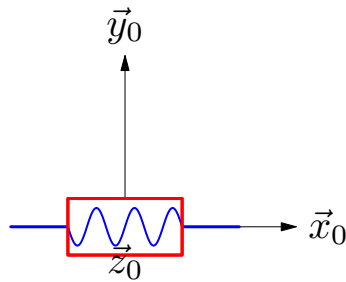
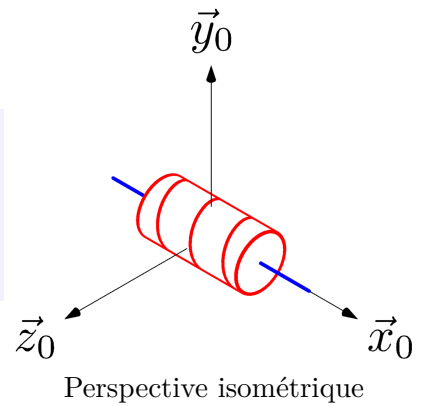
```
liaisonHelicoidale(triple point, triple axis, pen c1, pen c2) ;
```

- `triple point` : le centre de la liaison
- `triple axis` : l'axe de la liaison
- `pen c1` : mise en forme de la classe 1 (cylindre + hélicoïde)
- `pen c2` : mise en forme de la classe 2 (axe)

```

1  triple eye = (1,1,1) ;
2  triple up = (0,1,0) ;
3  currentprojection = orthographic(eye, up, 0) ;
4  currentlight = nolight ;
5  showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6  liaisonHelicoidale(0, b0.x, red, blue) ;

```



— .\maxpid

— .\maxpid_anim

2.3.4 Pivot glissant

BIB

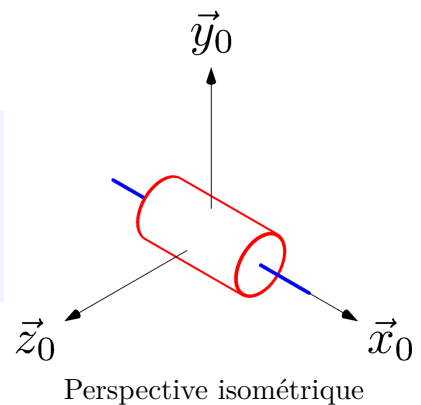
```
liaisonPivotGlissant(triple point, triple axis, pen c1, pen c2) ;
```

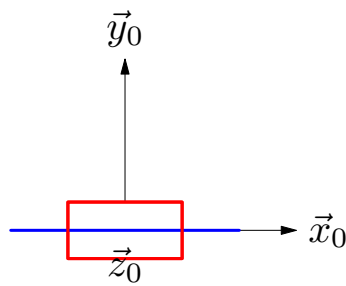
- **triple point** : le centre de la liaison (lindre)
- **triple axis** : l'axe de la liaison
- **pen c1** : mise en forme de la classe 1 (cy- — **pen c2** : mise en forme de la classe 2 (axe)

```

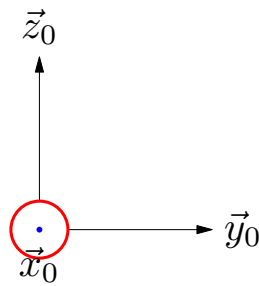
1  triple eye = (1,1,1) ;
2  triple up = (0,1,0) ;
3  currentprojection = orthographic(eye, up, 0) ;
4  currentlight = nolight ;
5  showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6  liaisonPivotGlissant(0, b0.x, red, blue) ;

```

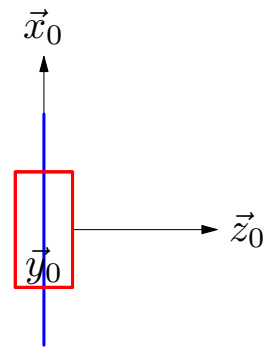




Plan $(0, \vec{x}, \vec{y})$



Plan $(0, \vec{y}, \vec{z})$



Plan $(0, \vec{z}, \vec{x})$

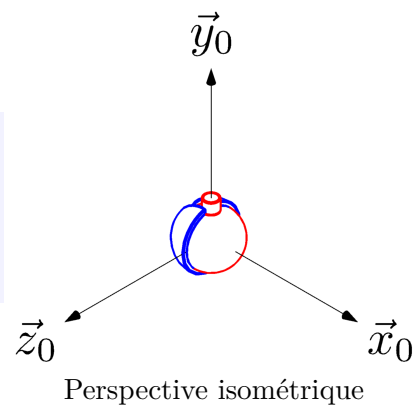
— <code>📁.\bielleManivelle</code>	— <code>📁.\I3D</code>	— <code>📁.\pinceCoupeCable</code>	— <code>📁.\sinusmatic</code>
— <code>📁.\bielleManivelle_anim</code>	— <code>📁.\I3D_anim</code>	— <code>📁.\pompePistonsAxiaux</code>	— <code>📁.\sinusmatic_anim</code>

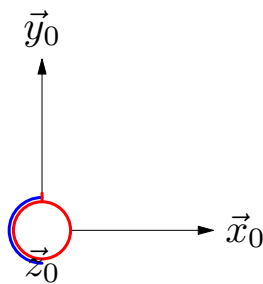
2.3.5 Rotule à doigt

```
liaisonRotuleDoigt(triple centre, triple axis, triple tige, pen c1, pen
c2, bool rapide=true, triple obs=currentprojection.camera) ;
```

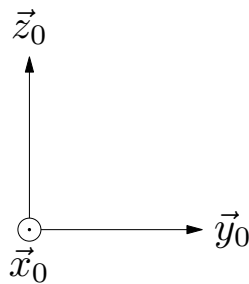
- | | |
|--|---|
| — <code>triple centre</code> : le centre de la liaison | creuse) |
| — <code>triple axis</code> : la « direction » de la rotule (normal au plan de la demi sphère creuse) | — <code>bool rapide=true</code> : permet d'avoir une construction rapide (voir 5.2) au détriment de la qualité visuelle |
| — <code>triple tige</code> : direction de la tige | |
| — <code>pen c1</code> : mise en forme de la classe 1 (sphère + tige) | — <code>triple obs=currentprojection.camera</code> : aucune raison de changer puisqu'il s'adapte à la commande <code>currentprojection</code> |
| — <code>pen c2</code> : mise en forme de la classe 2 (sphère | |

```
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 liaisonRotuleDoigt(0, b0.x, b0.y, red, blue) ;
```

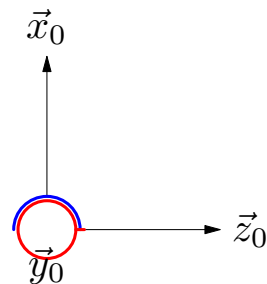




Plan $(0, \vec{x}, \vec{y})$



Plan $(0, \vec{y}, \vec{z})$



Plan $(0, \vec{z}, \vec{x})$

Remarque : la projection 2d, avec l'axe d'orientation de la rotule à doigt orthogonal au plan n'a pas été implémentée (car le concepteur n'en voyait pas l'intérêt, mais il peut le faire au besoin :-)).

— `DAE`

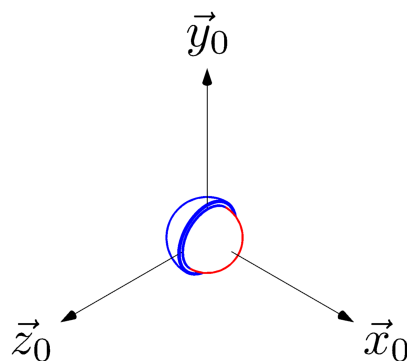
— `DAE_anim`

2.3.6 Rotule

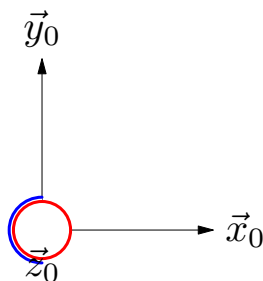
BIB `liaisonRotule(triple center, triple axis, pen c1, pen c2) ;`

- `triple center` : centre de la liaison (sphère)
- `triple axis` : la « direction » de la rotule (normal au plan de la demi sphère creuse)
- `pen c1` : mise en forme de la classe 1
- `pen c2` : mise en forme de la classe 2 (sphère creuse)

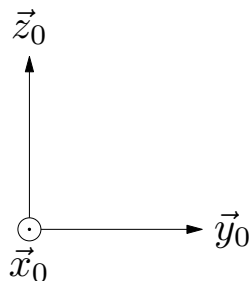
```
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 liaisonRotule(0, b0.x, red, blue) ;
```



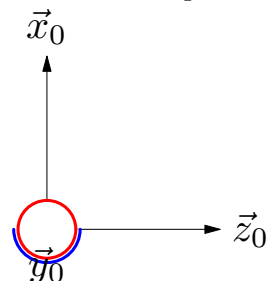
Perspective isométrique



Plan $(0, \vec{x}, \vec{y})$



Plan $(0, \vec{y}, \vec{z})$



Plan $(0, \vec{z}, \vec{x})$

Remarque : la projection 2d, avec l'axe d'orientation de la rotule orthogonal au plan n'a pas été implémentée (car le concepteur n'en voyait pas l'intérêt, mais il peut le faire au besoin :-)).

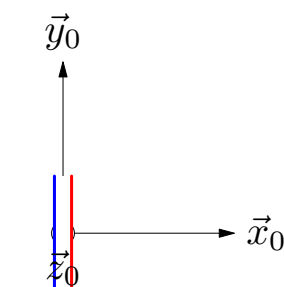
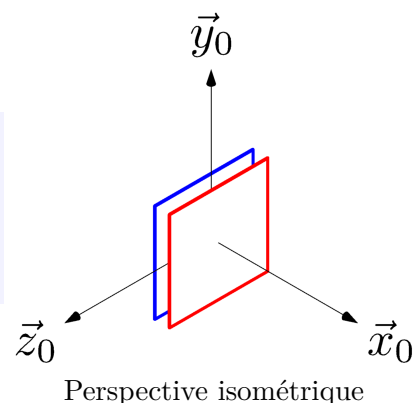
— `\bielleManivelle` — `\DAE_anim` — `\I3D_anim` — `\SDP_anim`
— `\bielleManivelle_anim` — `\directionCamion_anim` — `\piloteSafran` — `\sinusmatic`
— `\concasseur` — `\pompePistonsAxiaux` — `\sinusmatic`
— `\DAE` — `\I3D` — `\SDP` — `\sinusmatic_anim`

2.3.7 Appui-plan

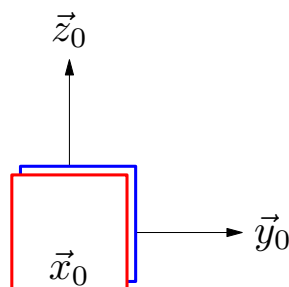
`liaisonAppuiPlan(triple center, triple normal, triple orientation, pen c1, pen c2) ;`

- `triple center` : centre de la liaison
- `triple normal` : normale au plan de contact
- `triple orientation` : une des deux directions d'orientation du plan (la deuxième est
- automatiquement calculée)
- `pen c1` : mise en forme de la classe 1
- `pen c2` : mise en forme de la classe 2

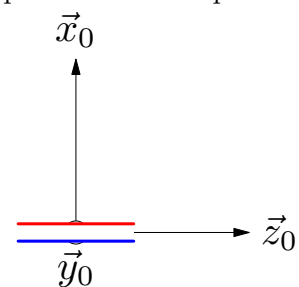
```
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 liaisonAppuiPlan(0,b0.x, b0.y, red, blue) ;
```



Plan $(0, \vec{x}, \vec{y})$



Plan $(0, \vec{y}, \vec{z})$



Plan $(0, \vec{z}, \vec{x})$

— `\limiteurCouple` — `\pompePistonsAxiaux`

2.3.8 Linéaire annulaire

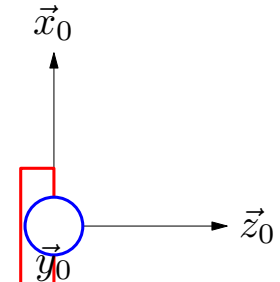
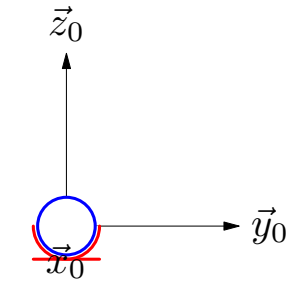
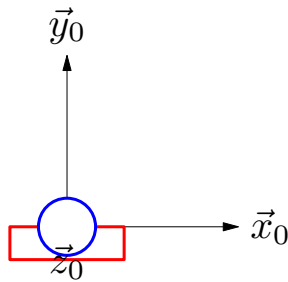
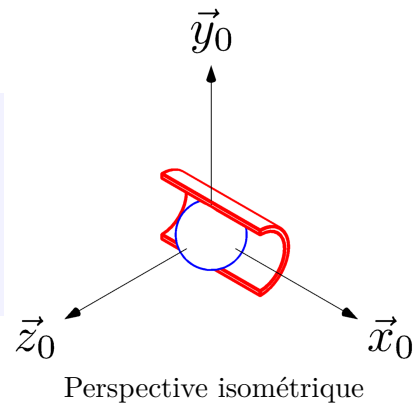
`liaisonLineaireAnnulaire(triple center, triple direction, triple cdc, pen c1, pen c2)`

- `triple center` : centre de la liaison
- `triple direction` : axe de translation possible
- `triple cdc` : représente le côté du demi-
- cylindre
- `pen c1` : mise en forme de la classe 1 (demi-cylindre)
- `pen c2` : mise en forme de la classe 2 (boule)

```

1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 liaisonLineaireAnnulaire(0, b0.x, -b0.z, red,
    blue) ;

```



— \concasseur

— \piloteSafran

— \SDP

— \SDP_anim

2.3.9 Linéaire rectiligne

```

liaisonLineaireRectiligne(triple centre, triple normale, triple
    droiteContact, pen c1, pen c2) ;

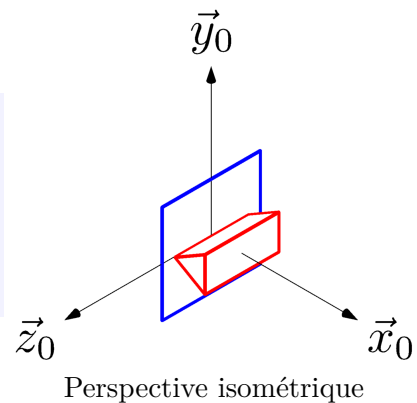
```

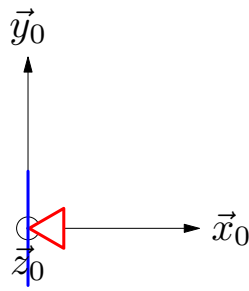
- **triple** centre : centre de la liaison
- **triple** normale : normale au plan de contact
- **triple** droiteContact : vecteur directeur de la droite de contact
- **pen** c1 : mise en forme de la classe 1 (prisme triangulaire)
- **pen** c2 : mise en forme de la classe 2 (plan)

```

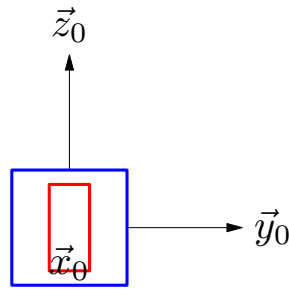
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 liaisonLineaireRectiligne(0, b0.x, b0.z, red,
    blue) ;

```

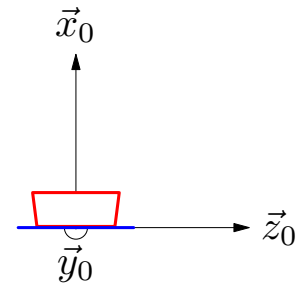




Plan $(0, \vec{x}, \vec{y})$



Plan $(0, \vec{y}, \vec{z})$



Plan $(0, \vec{z}, \vec{x})$

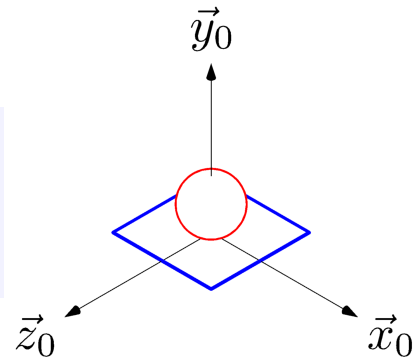
— \pinceCoupeCable

2.3.10 Ponctuelle

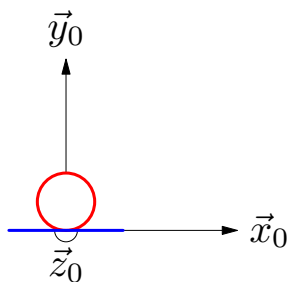
`liaisonPonctuelle(triple center, triple normal, triple orientation, pen
c1, pen c2) ;`

- `triple center` : centre de la liaison
- `triple normal` : normale au plan de contact
- `triple orientation` : vecteur directeur d'un
- des côtés du plan représenté
- `pen c1` : mise en forme de la classe 1 (boule)
- `pen c2` : mise en forme de la classe 2 (plan)

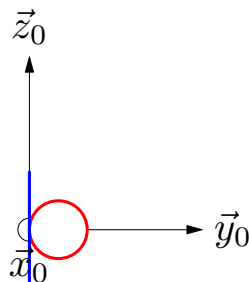
```
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 liaisonPonctuelle(0, b0.y, b0.z, red, blue) ;
```



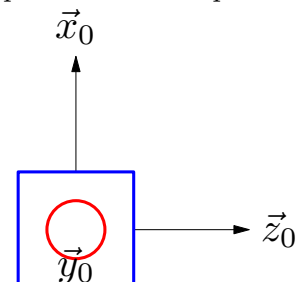
Perspective isométrique



Plan $(0, \vec{x}, \vec{y})$



Plan $(0, \vec{y}, \vec{z})$



Plan $(0, \vec{z}, \vec{x})$

— \limiteurCouple

— \pinceCoupeCable

2.4 Transmission de puissance

2.4.1 Engrenages

BIB

```
transEngrenages(triple c1, triple n1, real r1, pen CEC1, triple c2, triple
n2, pen CEC2) ;
```

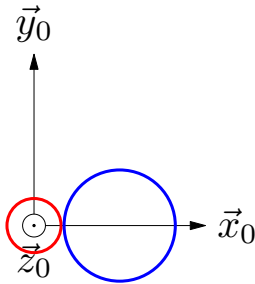
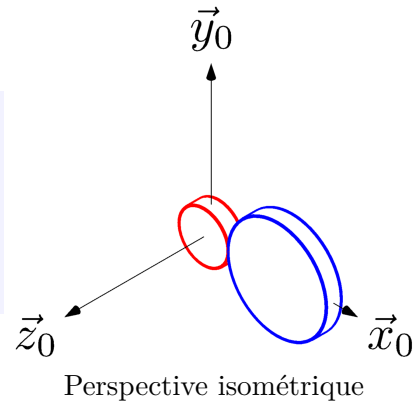
Cette fonction unique gère les contacts intérieur et extérieur, ainsi que les axes parallèles et axes concourants !

Pour des engrenages à axes parallèles à contact intérieur ou extérieur :

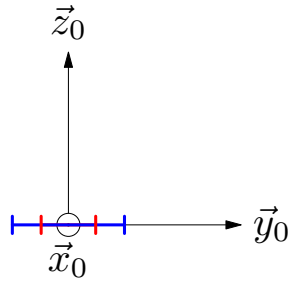
- `triple c1` : centre de la poulie 1
- `triple n1` : vecteur directeur de la poulie 1
- `real r1` : rayon de la poulie 1
- `pen CEC1` : mise en forme de la classe 1
- `triple c2` : centre de la poulie 2
- `triple n2` : vecteur directeur de la poulie 2
- `pen CEC2` : mise en forme de la classe 2

Remarque : `r2` est automatiquement calculé.

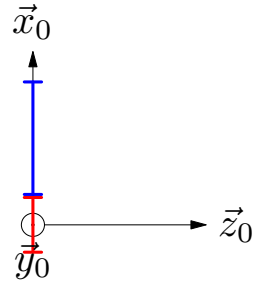
```
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 transEngrenages(0, b0.z, 0.25, red,
0+(0.75,0,0), b0.z, blue) ;
```



Plan $(0, \vec{x}, \vec{y})$



Plan $(0, \vec{y}, \vec{z})$

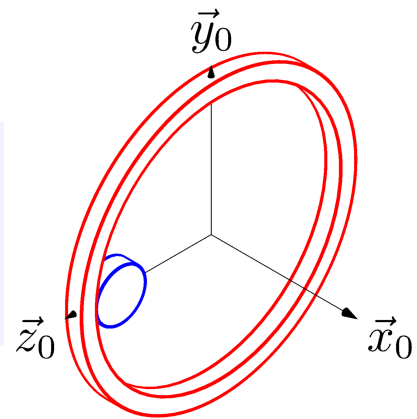


Plan $(0, \vec{z}, \vec{x})$

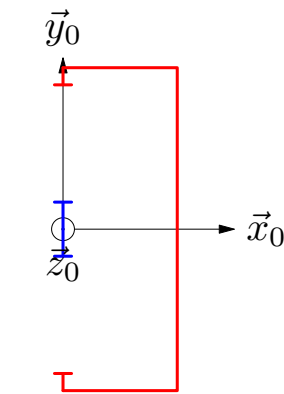
```

1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 transEngrenages(0, b0.x, 1.25, red, 0+(0,0,1),
    b0.x, blue) ;

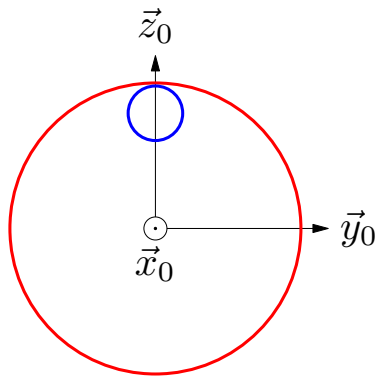
```



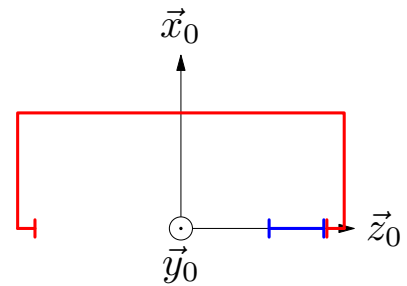
Perspective isométrique



Plan $(0, \vec{x}, \vec{y})$



Plan $(0, \vec{y}, \vec{z})$



Plan $(0, \vec{z}, \vec{x})$

Pour les engrenages à axes concourants :

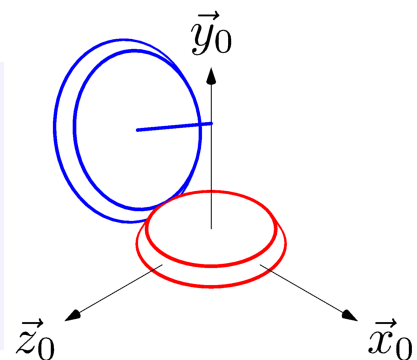
- `triple c1` : centre de la poulie 1
- `triple n1` : vecteur directeur de la poulie 1
- `real r1` : rayon de la poulie 1
- `pen CEC1` : mise en forme de la classe 1
- `triple c2` : un point quelconque de l'axe de rotation (le vrai centre sera calculé)
- `triple n2` : vecteur directeur de la poulie 2
- `pen CEC2` : mise en forme de la classe 2

Remarque : `r2` est également automatiquement calculé.

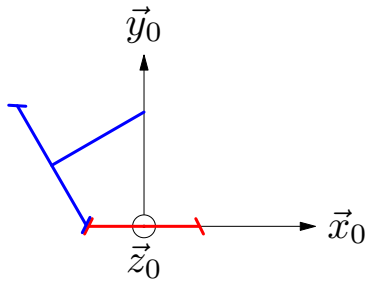
```

1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 basis b1 = rotationBasis(1, b0, pi/6, 'z',
    b0.z) ;
7 transEngrenages(0, b0.y, 0.5, red, 0+(0,1,0),
    b1.x, blue) ;

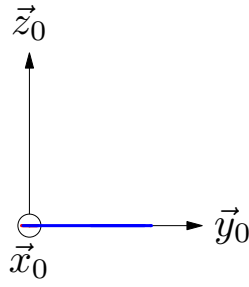
```



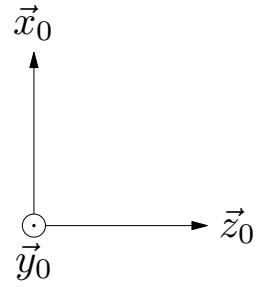
Perspective isométrique



Plan $(0, \vec{x}, \vec{y})$



Plan $(0, \vec{y}, \vec{z})$



Plan $(0, \vec{z}, \vec{x})$

— \concasseur

— \trainEpicycloidaux

2.4.2 Poulie courroie

```
transPoulieCourroie(triple c1, triple n1, real r1, pen CEC1, triple c2,
  triple n2, real r2, pen CEC2) ;
```

— **triple c1** : centre de la poulie 1

— **triple n1** : vecteur directeur de la poulie 1

— **real r1** : rayon de la poulie 1

— **pen CEC1** : mise en forme de la classe 1

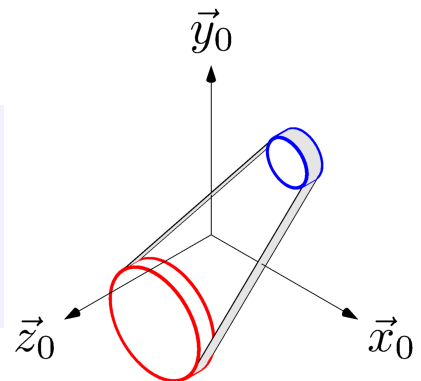
— **triple c2** : centre de la poulie 1

— **triple n2** : vecteur directeur de la poulie 2

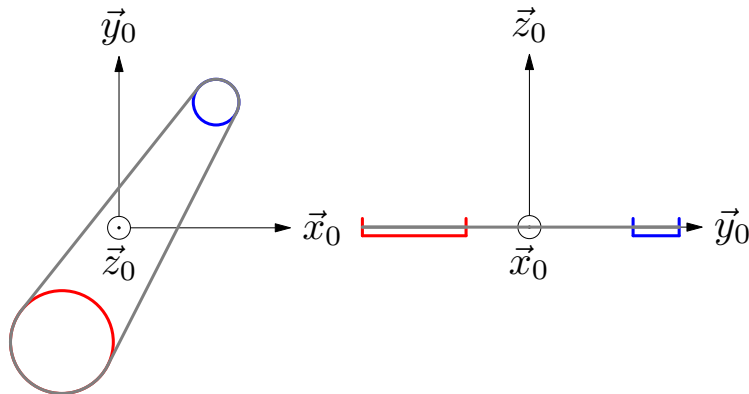
— **real r2** : rayon de la poulie 2

— **pen CEC2** : mise en forme de la classe 2

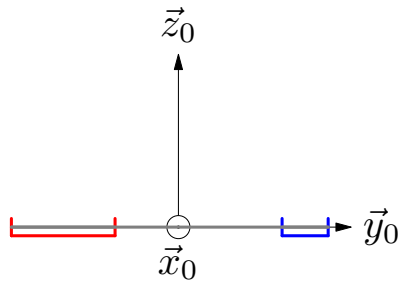
```
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 transPoulieCourroie(0+(-0.5,-1,0), b0.z, 0.45,
  red, 0+(0.85,1.1,0), b0.z, 0.2, blue) ;
```



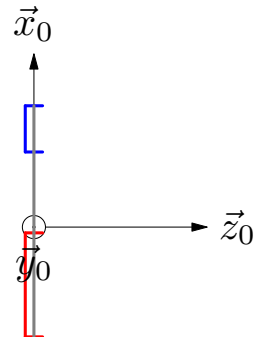
Perspective isométrique



Plan $(0, \vec{x}, \vec{y})$



Plan $(0, \vec{y}, \vec{z})$



Plan $(0, \vec{z}, \vec{x})$

— \concasseur

— \SDP

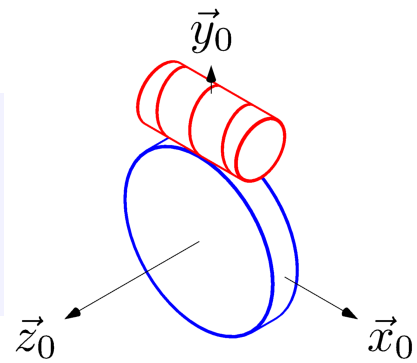
— \SDP_anim

2.4.3 Roue et vis sans fin

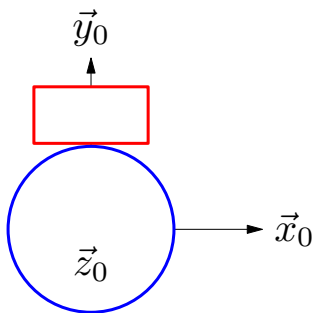
```
transRoueVis(triple c1, triple n1, pen CEC1, triple c2, triple n2, pen
CEC2) ;
```

- `triple c1` : centre de la vis
- `triple n1` : direction vis
- `pen CEC1` : mise en forme de la classe 1 (vis)
- `triple c2` : centre de la roue
- `triple n2` : axe de la roue
- `pen CEC2` : mise en forme de la classe 2 (roue)

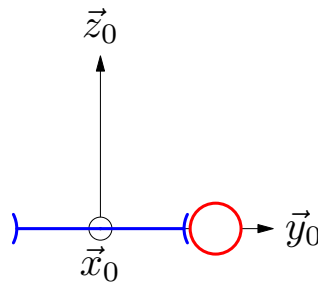
```
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 transRoueVis(0 + (0,1,0), b0.x, red, 0 , b0.z,
blue) ;
```



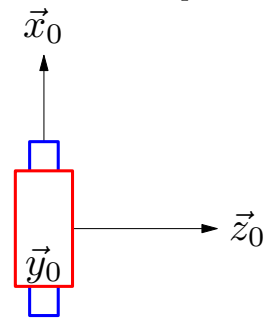
Perspective isométrique



Plan $(0, \vec{x}, \vec{y})$



Plan $(0, \vec{y}, \vec{z})$



Plan $(0, \vec{z}, \vec{x})$

— `forcebat`

2.4.4 Pignon crémaillère

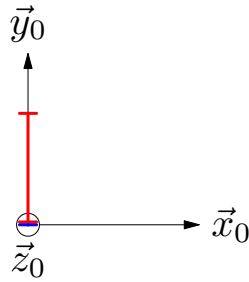
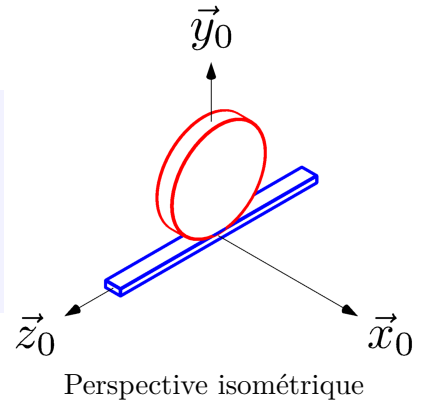
```
transPignonCremailliere(triple c1, triple n1, pen CEC1, triple c2, triple
n2, real r2, pen CEC2) ;
```

- `triple c1` : centre du pignon
- `triple n1` : axe du pignon
- `pen CEC1` : mise en forme de la classe 1 (pignon)
- `triple c2` : « centre » de la crémaillère tel que la distance entre c1 et c2 correspond au rayon de la relation cinématique
- `triple n2` : vecteur directeur de la crémaillère
- `real r2` : longueur de la crémaillère
- `pen CEC2` : mise en forme de la classe 2 (crémaillère)

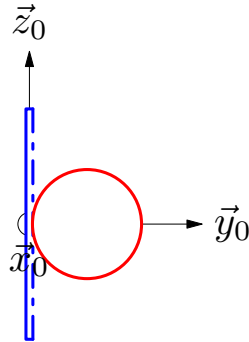
```

1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 transPignonCremailliere(0 + (0,0.5,0), b0.x,
   red, 0 , b0.z, 2, blue) ;

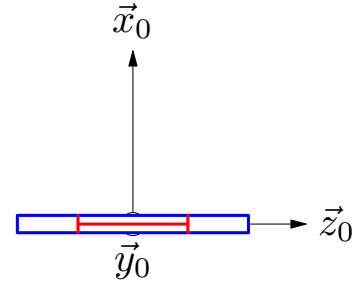
```



Plan $(0, \vec{x}, \vec{y})$



Plan $(0, \vec{y}, \vec{z})$



Plan $(0, \vec{z}, \vec{x})$

— \DAE

— \DAE_anim

2.4.5 Chaîne

```

transChaine(triple c1, triple n1, real r1, pen CEC1, triple c2, triple n2,
  real r2, pen CEC2) ;

```

— **triple c1** : centre de la roue 1

— **triple n1** : axe de la roue 1

— **real r1** : rayon de la roue 1

— **pen CEC1** : mise en forme de la roue 1

— **triple c2** : centre de la roue 2

— **triple n2** : axe de la roue 2

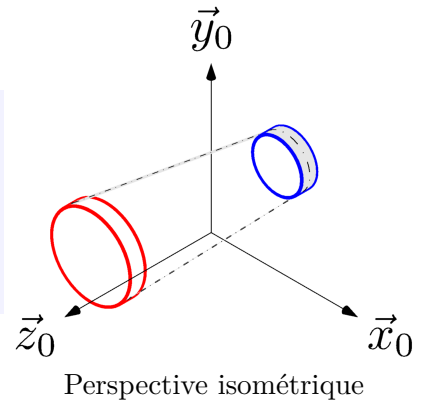
— **real r2** : rayon de la roue 2

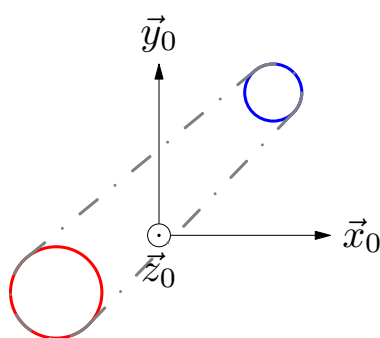
— **pen CEC2** : mise en forme de la roue 2

```

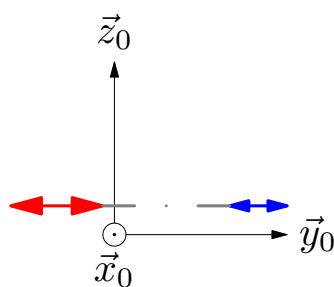
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 transChaine(0+(-0.9,-0.5,0.25), b0.z, 0.4, red,
  0+(1,1.25,0.25), b0.z, 0.25, blue) ;

```

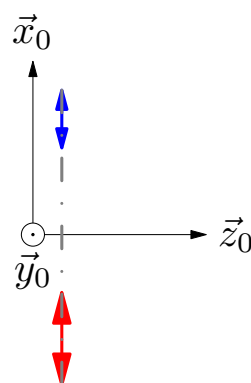




Plan $(0, \vec{x}, \vec{y})$



Plan $(0, \vec{y}, \vec{z})$



Plan $(0, \vec{z}, \vec{x})$

— `forcebat`

2.5 Habillage

2.5.1 Relier

`link(path3 chemin, pen CEC) ;`

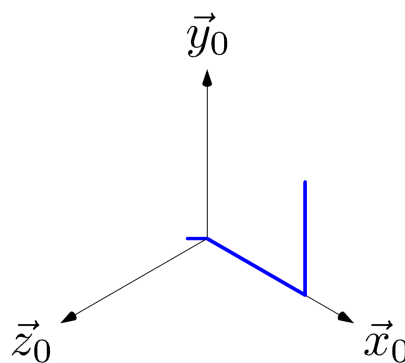
— `path3 chemin` : le chemin à tracer

— `pen CEC` : mise en forme de la classe

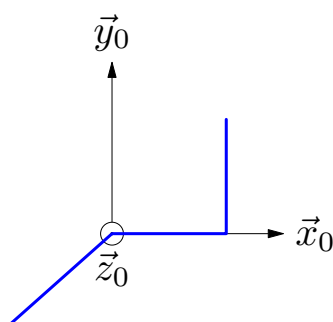
Pour relier une commande unique donc. Un `path3` est donc constitué de points (objet `triple`) reliés entre eux par `--` (en vrai il y a donc possibilités mais pour cette bibliothèque dans un premier temps c'est suffisant).

On peut bien sûr créer son propre chemin. N'hésitez pas à feuilleter les exemples.

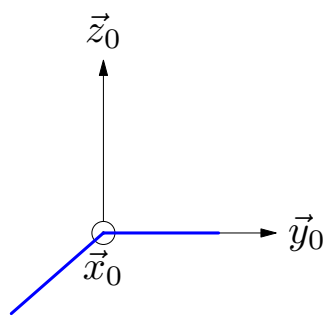
```
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 link((-0.9,-0.8,-0.7) -- 0 -- 0+b0.x --
      0+b0.x+b0.y, blue) ;
```



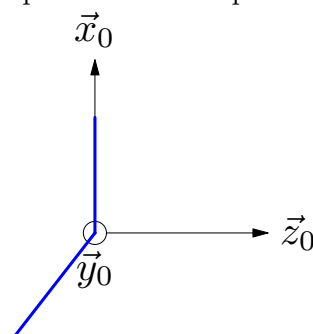
Perspective isométrique



Plan $(0, \vec{x}, \vec{y})$



Plan $(0, \vec{y}, \vec{z})$



Plan $(0, \vec{z}, \vec{x})$

— <code>\bielleManivelle</code>	— <code>\directionCamion_anim</code>	— <code>\jointCardan</code>	— <code>\pinceCoupeCable</code>
— <code>\bielleManivelle_anim</code>	— <code>\falconHaptic</code>	— <code>\limiteurCouple</code>	— <code>\pompePistonsAxiaux</code>
— <code>\concasseur</code>	— <code>\falconHaptic_anim</code>	— <code>\maxpid</code>	— <code>\SDP</code>
— <code>\croixMalte</code>	— <code>\faucheuse</code>	— <code>\maxpid_anim</code>	— <code>\SDP_anim</code>
— <code>\croixMalte_anim</code>	— <code>\forcebat</code>	— <code>\pilote5000</code>	— <code>\sinusmatic</code>
— <code>\DAE</code>	— <code>\I3D</code>	— <code>\pilote5000_anim</code>	— <code>\sinusmatic_anim</code>
— <code>\DAE_anim</code>	— <code>\I3D_anim</code>	— <code>\piloteSafran</code>	— <code>\trainEpicycloidaux</code>

2.5.2 Bâti

BIB `bati(triple point, triple dirGB, triple orPB, pen CEC) ;`

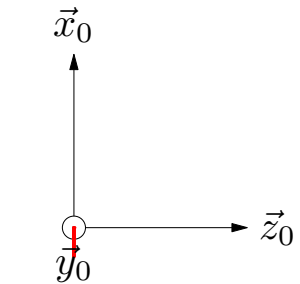
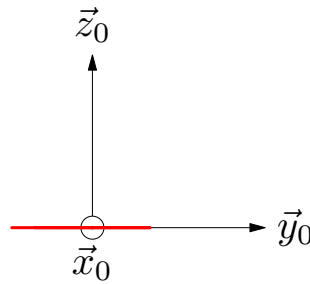
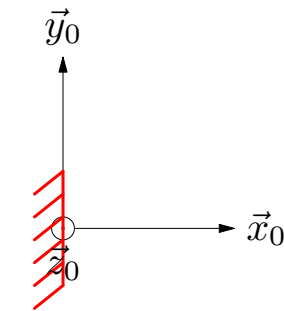
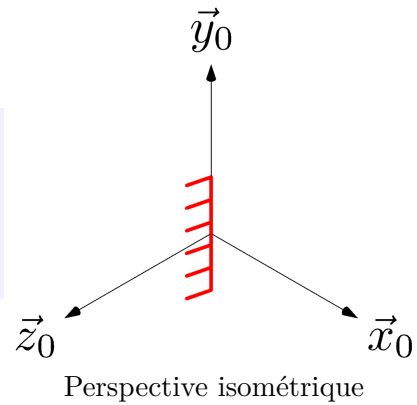
- `triple point` : point milieu de la grande barre
- `triple dirGB` : direction de la grande barre du râteau
- `triple orPB` : direction (côté) où sera dessiné le râteau par rapport à la grande barre
- `pen CEC` : mise en forme de la classe

À terme, et pour faire plaisir, il y aura d'autres designs mais pas dans l'immédiat.

```

1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 bati(0, b0.y, -b0.x, red) ;

```



— <code>\bielleManivelle</code>	— <code>\DAE_anim</code>	— <code>\pilote5000</code>	— <code>\sinusmatic</code>
— <code>\bielleManivelle_anim</code>	— <code>\faucheuse</code>	— <code>\pilote5000_anim</code>	— <code>\sinusmatic_anim</code>
— <code>\concasseur</code>	— <code>\forcebat</code>	— <code>\pinceCoupeCable</code>	— <code>\trainEpicycloidaux</code>
— <code>\croixMalte</code>	— <code>\jointCardan</code>	— <code>\pompePistonsAxiaux</code>	
— <code>\croixMalte_anim</code>	— <code>\maxpid</code>	— <code>\SDP</code>	
— <code>\DAE</code>	— <code>\maxpid_anim</code>	— <code>\SDP_anim</code>	

2.5.3 Cylindre

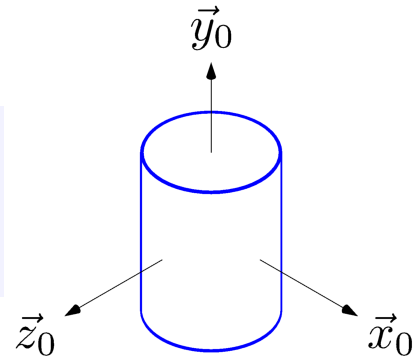
BIB `addCylinder(triple point, triple axis, real r, real h, pen CEC) ;`

- `triple` point : centre du cylindre plein
- `triple` axis : axe de révolution du cylindre
- `real` r : rayon du cylindre
- `real` h : hauteur du cylindre
- `pen` CEC : mise en forme de la classe

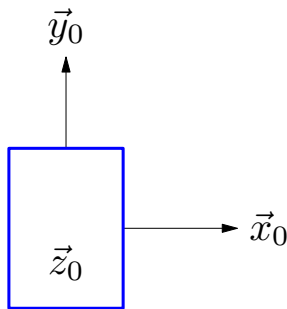
```

1  triple eye = (1,1,1) ;
2  triple up = (0,1,0) ;
3  currentprojection = orthographic(eye, up, 0) ;
4  currentlight = nolight ;
5  showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6  addCylinder(0, b0.y, 0.5, 1.4, blue) ;

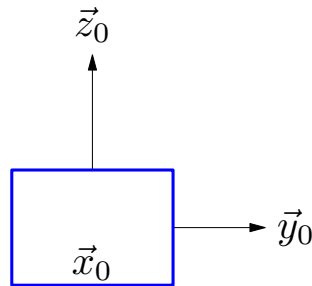
```



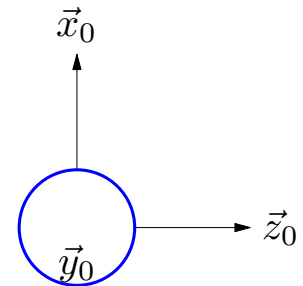
Perspective isométrique



Plan $(0, \vec{x}, \vec{y})$



Plan $(0, \vec{y}, \vec{z})$



Plan $(0, \vec{z}, \vec{x})$

— `⊞.\bielleManivelle` — `⊞.\bielleManivelle_anim` — `⊞.\maxpid` — `⊞.\maxpid_anim`

2.5.4 Disque

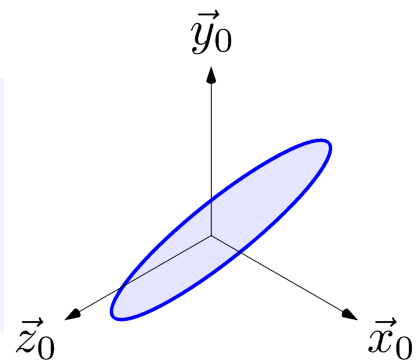
BIB `addDisque(triple centre, triple normal, real R, pen CEC) ;`

- `triple` centre : point milieu du cylindre
- `triple` normal : axe de révolution
- `real` R : rayon du cylindre
- `pen` CEC : mise en forme

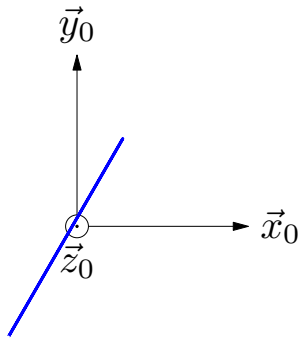
```

1  triple eye = (1,1,1) ;
2  triple up = (0,1,0) ;
3  currentprojection = orthographic(eye, up, 0) ;
4  currentlight = nolight ;
5  showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6  basis b1 = rotationBasis(1, b0, -30/360*2*pi,
7  'z', b0.z) ;
8  addDisque((-0.1,-0.1,-0.2), b1.x, 1, blue) ;

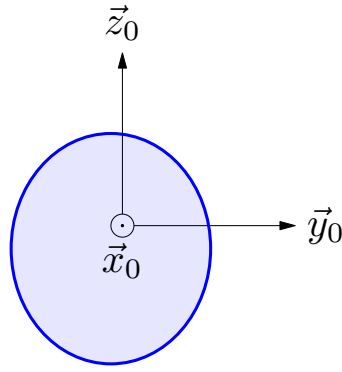
```



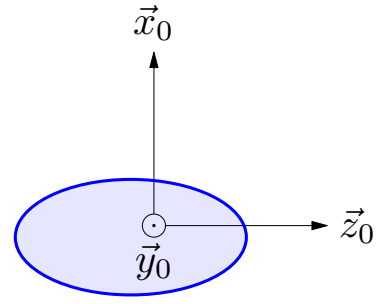
Perspective isométrique



Plan $(0, \vec{x}, \vec{y})$



Plan $(0, \vec{y}, \vec{z})$



Plan $(0, \vec{z}, \vec{x})$

- `triple` centre : centre de la liaison
- `triple` normal : normale de la surface
- `real` R : rayon du disque
- `pen` CEC : mise en forme

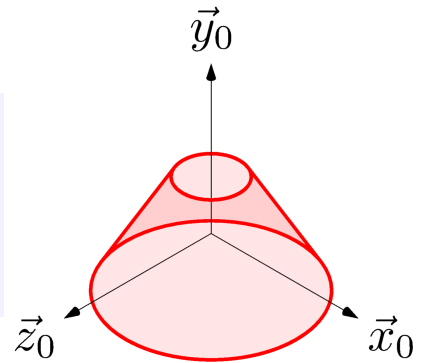
Remarque : dans l'intérêt d'une utilisation 3d, l'intérieur du disque est transparent.

— `directionCamion_anim` — `falconHaptic` — `falconHaptic_anim` — `pompePistonsAxiaux`

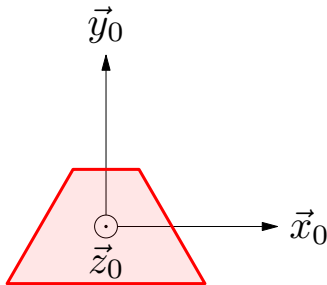
2.5.5 Surface conique tronquée

```
addSurfConiqueTronquee(triple sommet, triple axis, real hauteur1, real
    hauteur2, real demiAngle, pen CEC) ;
```

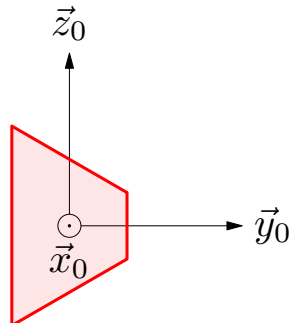
```
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 addSurfConiqueTronquee((0,1,0), -b0.y, 0.5,
    1.5, pi/6, red) ;
```



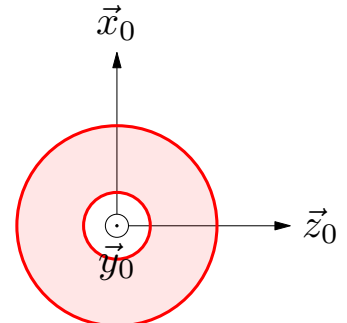
Perspective isométrique



Plan $(0, \vec{x}, \vec{y})$



Plan $(0, \vec{y}, \vec{z})$



Plan $(0, \vec{z}, \vec{x})$

- **triple** sommet : sommet de la surface conique
- **triple** axis : axe de révolution de la surface
- **real** hauteur1 : hauteur du début de la portion
- **real** hauteur2 : hauteur de la fin de la portion
- **real** demiAngle : demi-angle au sommet
- **pen** CEC : mise en forme

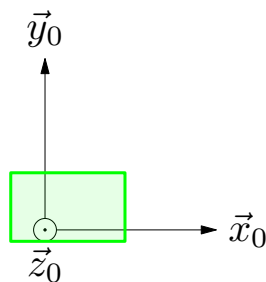
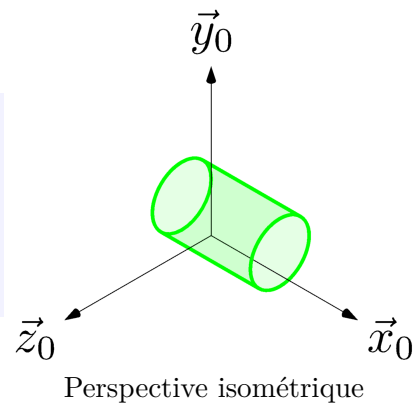
Remarque : dans l'intérêt de voir à travers, la surface est transparente.

—  \concasseur

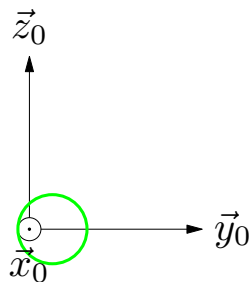
2.5.6 Surface cylindrique

BIB `addSurfCylinder(triple point, triple axis, real r, real h, pen CEC) ;`

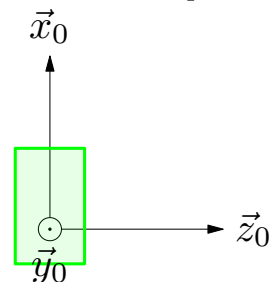
```
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 addSurfCylinder((0.2,0.2,0), b0.x, 0.3, 1,
  green) ;
```



Plan $(0, \vec{x}, \vec{y})$




Plan $(0, \vec{y}, \vec{z})$




Plan $(0, \vec{z}, \vec{x})$

- **triple** point : centre de symétrie de la surface
- **triple** axis : axe de révolution de la surface
- **real** r : rayon de la surface cylindrique
- **real** h : hauteur de la surface cylindrique
- **pen** CEC : mise en forme

Remarque : dans l'intérêt de voir à travers, la surface est transparente.

—  \croixMalte

—  \croixMalte_anim

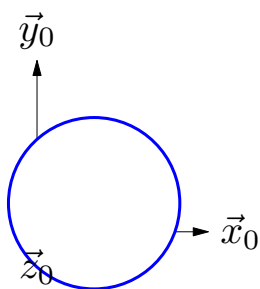
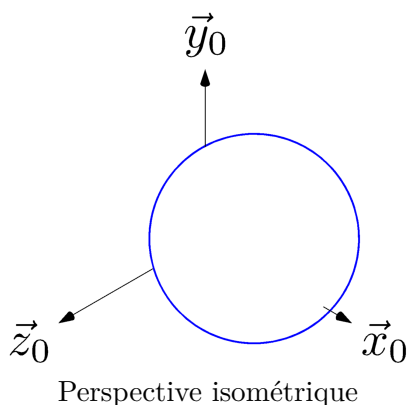
2.5.7 Sphère

BIB `addSphere(triple center, real rayon, pen CEC) ;`

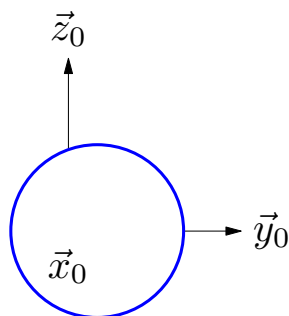
```

1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 addSphere(0+(0.5,0.25,0), 0.75, blue) ;

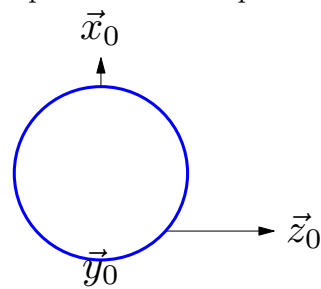
```



Plan $(0, \vec{x}, \vec{y})$



Plan $(0, \vec{y}, \vec{z})$



Plan $(0, \vec{z}, \vec{x})$

— `triple center` : centre de la sphère
— `real rayon` : rayon de la sphère

— `pen CEC` : mise en forme

— `\falconHaptic`

— `\falconHaptic_anim`

— `\fauchouse`

2.5.8 U-shape

```

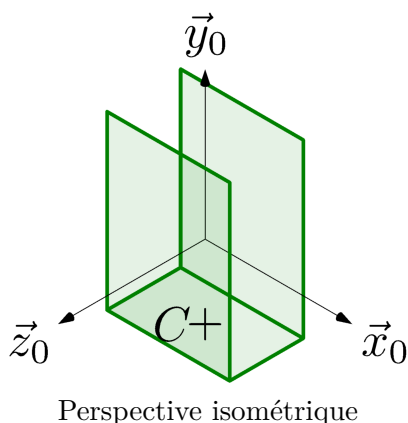
addUshape(triple center, triple axisL, triple axisH, real L, real H,
real E, pen CEC) ;

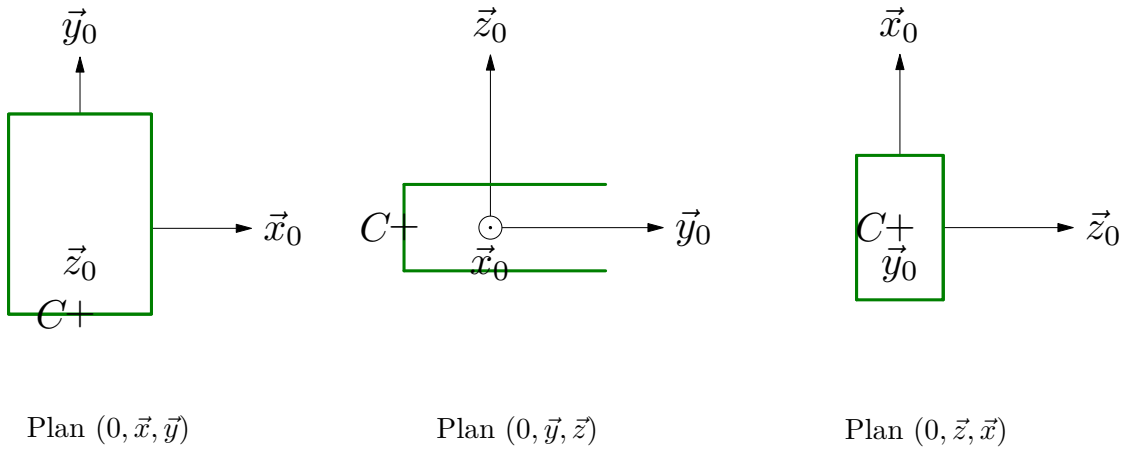
```

```

1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 addUshape(0-0.75*b0.y, b0.x, b0.y, 1.25, 1.75,
0.75, deepgreen) ;
7 namePoint(0-0.75*b0.y, "C", (-1,0)) ;

```





- `triple` center : centre du plat du U
- `triple` axisL : vecteur directeur longueur
- `triple` axisH : vecteur directeur hauteur
- `real` L : longueur
- `real` H : hauteur
- `real` E : épaisseur
- `pen` CEC : mise en forme

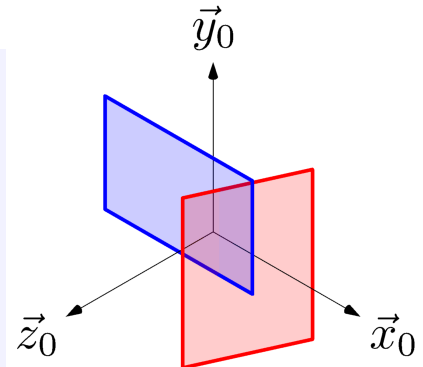
Remarque : dans l'intérêt de voir à travers, les surfaces sont transparentes en 3d et opaques en 2d.

— `⊞.\croixMalte` — `⊞.\croixMalte_anim` — `⊞.\fauchouse`

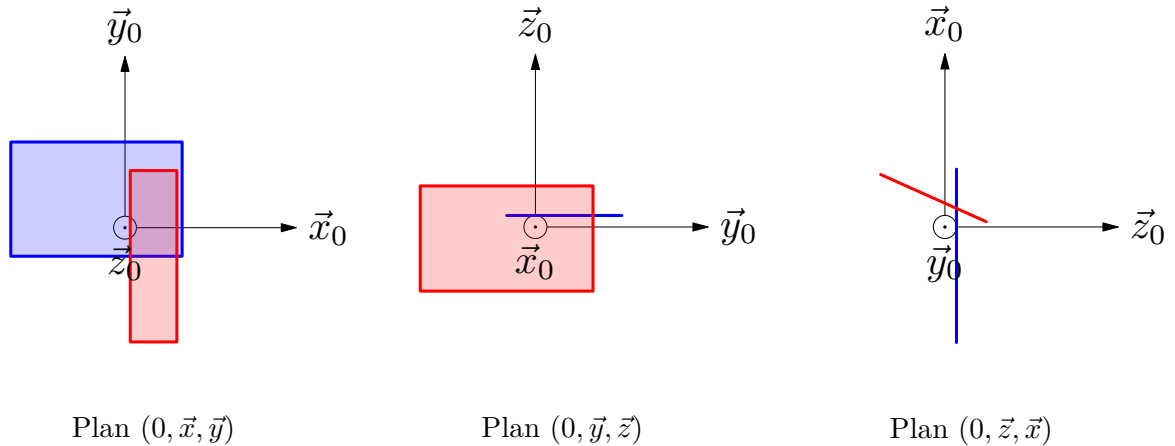
2.5.9 Surface plane

```
addSurfPlane(triple center, triple axis1, real c1, triple axis2, real c2,
             pen CEC) ;
```

```
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 basis b1 = rotationBasis(1, b0, 20, 'y', b0.y) ;
7 addSurfPlane((-0.25, 0.25,0.1), b0.x, 1.5,
               b0.y, 1, blue) ;
8 addSurfPlane((0.25, -0.25,-0.1), b1.x, 1, b1.y,
               1.5, red) ;
```



Perspective isométrique



- `triple center` : centre de la surface
- `triple axis1` : vecteur directeur 1
- `real c1` : côté 1
- `triple axis2` : vecteur directeur 2
- `real c2` : côté 2
- `pen CEC` : mise en forme

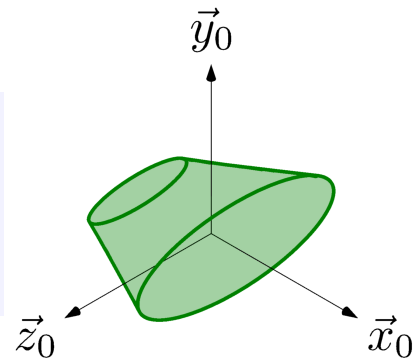
Remarque : dans l'intérêt de voir à travers, les surfaces sont transparentes en 3d et opaques en 2d.

— `\limiteurCouple` — `\piloteSafran`

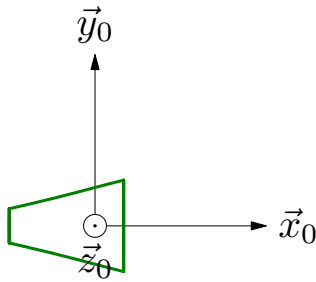
2.5.10 Triangulation STL

```
importSTL(string nameSTL, triple center, triple exSTL, triple eySTL, real
scaleSTL, pen CEC, real valOpa=0.2) ;
```

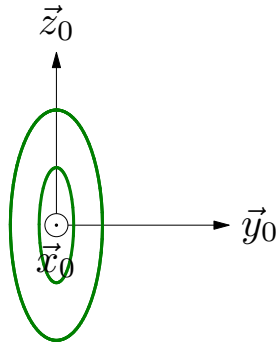
```
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 importSTL('saf.stl', (-0.25,0,0), -b0.z, b0.x,
0.5, deepgreen) ;
```



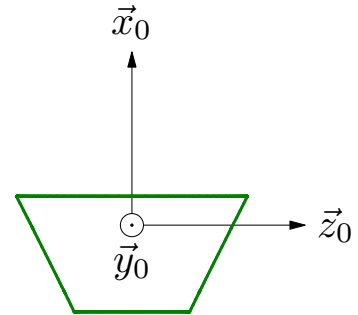
Perspective isométrique



Plan $(0, \vec{x}, \vec{y})$



Plan $(0, \vec{y}, \vec{z})$



Plan $(0, \vec{z}, \vec{x})$

- `string nameSTL` : nom du fichier à importer
- `triple center` : en quel point
- `triple exSTL` : vecteur correspondant au vecteur \vec{e}_x du fichier
- `triple eySTL` : vecteur correspondant au vecteur \vec{e}_y du fichier
- `real scaleSTL` : mise à l'échelle des valeurs numériques
- `pen CEC` : mise en forme
- `real valOpa=0.2` : valeur par défaut de l'opacité (3d uniquement)

Remarque : à utiliser avec précaution pour les mêmes raisons que la liaison rotule à doigt dont on peut trouver l'explication page 40. Je déconseille en 2d car seules les arêtes sont tracées.

— `\piloteSafran`

2.6 Paramétrage

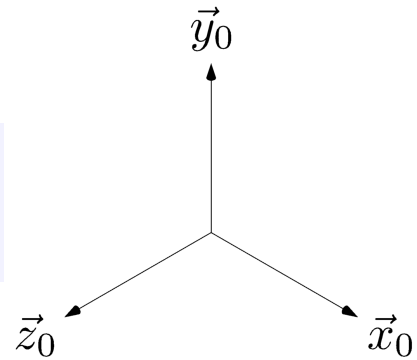
Donc maintenant on va rajouter des axes, du paramétrage angulaire, le nom des points, des flèches, du texte ...

2.6.1 Bases

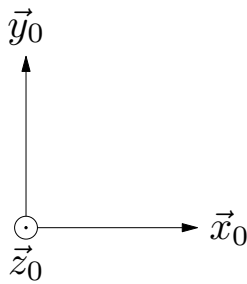
```
showBasis(basis b, triple point, triple coeff=(1,1,1), pen
style=black+0.25) ;
```

- **basis** b : base à tracer
- **triple** point : origine associée à la base
- **triple** coeff : longueurs des axes (valeur par défaut (1,1,1))
- **pen** style : mise en forme de la base (valeur par défaut black+0.25)

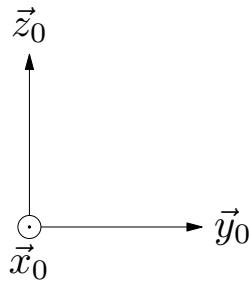
```
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
```



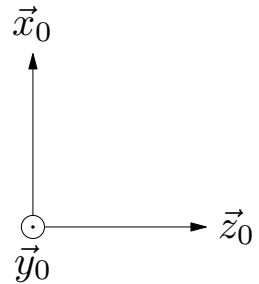
Perspective isométrique



Plan $(0, \vec{x}, \vec{y})$



Plan $(0, \vec{y}, \vec{z})$



Plan $(0, \vec{z}, \vec{x})$

— \bielleManivelle	— \falconHaptic_anim	— \pilote5000	— \SDP_anim
— \concasreur	— \faucheuse	— \pilote5000_anim	— \sinusmatic
— \DAE	— \I3D	— \piloteSafran	— \trainEpicycloidaux
— \DAE_anim	— \I3D_anim	— \pinceCoupeCable	
— \directionCamion_anim	— \limiteurCouple	— \pompePistonsAxiaux	
— \falconHaptic	— \maxpid	— \SDP	

Il est néanmoins intéressant de choisir les axes que l'on souhaite tracer dans beaucoup de cas. La commande suivante le réalise.

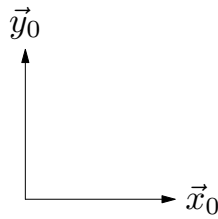
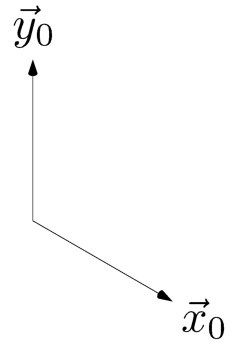
```
showAxis(basis b, int[] tabAxis, triple point, real coeff=1, pen
style=black+0.25)
```

- **basis** b : base à tracer
- **int**[] tabAxis : tableau d'entiers (0 pour l'axe x – 1 pour y – 2 pour z)
- **triple** point : origine associée à la base
- **real** coeff : longueurs des axes sélectionnés (valeur par défaut 1)
- **pen** style : mise en forme de la base (valeur par défaut black+0.25)

```

1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 simpleSphereBounding (1.5) ;
6 int[] tabAxis = {0,1} ;
7 showAxis(b0, tabAxis, 0, coeff=1.5) ;

```



Plan $(0, \vec{x}, \vec{y})$



Plan $(0, \vec{y}, \vec{z})$

Perspective isométrique



Plan $(0, \vec{z}, \vec{x})$

— <code>B.\bielleManivelle</code>	— <code>B.\fauchouse</code>	— <code>B.\maxpid</code>	— <code>B.\pompePistonsAxiaux</code>
— <code>B.\directionCamion_anim</code>	— <code>B.\jointCardan</code>	— <code>B.\pilote5000</code>	— <code>B.\sinusmatic</code>

Souvent, on aimerait préciser le texte en bout de flèche pour illustrer une égalité par exemple.

```

showAxisName(triple axis, triple point, string name, real coeff=1, pen
style=black+0.25) ;

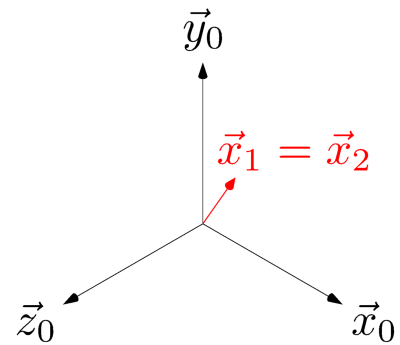
```

— <code>triple axis</code> : axe à afficher	— <code>real coeff=1</code> : longueur de la flèche
— <code>triple point</code> : point de départ du tracé	
— <code>string name</code> : texte à afficher en bout de flèche	— <code>pen style=black+0.25</code> : mise en forme de l'ensemble

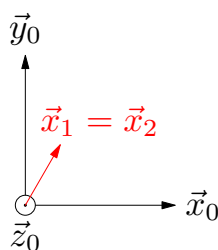
```

1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 simpleSphereBounding (1.5) ;
6 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
7 basis b1 = rotationBasis(1, b0, 60/360*2*pi,
'z', b0.z) ;
8 showAxisName(b1.x, 0, "$\vec{x}_1=\vec{x}_2$",
coeff=0.7, style = red+0.25) ;

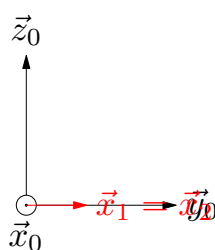
```



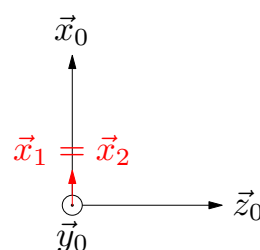
Perspective isométrique



Plan $(0, \vec{x}, \vec{y})$



Plan $(0, \vec{y}, \vec{z})$



Plan $(0, \vec{z}, \vec{x})$

— `\jointCardan`

2.6.2 Paramétrage

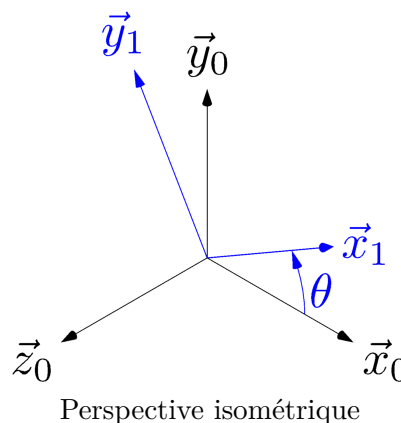
Il est intéressant de montrer le paramétrage angulaire et linéaire afin d'illustrer.

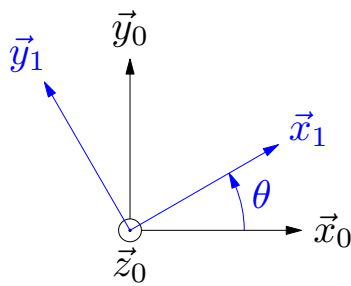
```
showParamAng(triple point, triple axis1, triple axis2, string name, real
             coeff=1, pen style=black+0.25) ;
```

- | | |
|--|---|
| — <code>triple point</code> : origine de l'arc de cercle | — <code>real coeff=1</code> : position de l'arc de cercle suivant l'axe (valeur par défaut 1 – à ajuster) |
| — <code>triple axis1</code> : axe 1 | |
| — <code>triple axis2</code> : axe 2 (vers celui-ci) | |
| — <code>string name</code> : nom du paramètre (LaTeX compatible) | — <code>pen style</code> : mise en forme du paramètre (valeur par défaut <code>black+0.25</code>) |

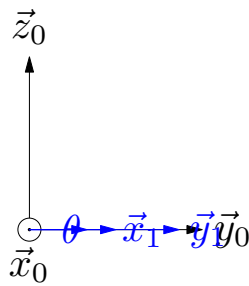
Remarque : La commande `showParamAng` remplace la commande `showParameter` qui devient obsolète à partir de la version v1.0.6. Cette dernière reste néanmoins fonctionnelle mais disparaîtra.

```
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 basis b1 = rotationBasis(1, b0, pi/6, 'z',
7   b0.z) ;
8 int[] tabAxis = {0,1} ;
9 showAxis(b1, tabAxis, 0, coeff=1.5, style =
10   0.25+blue) ;
11 showParamAng(0, b0.x, b1.x, "$\theta$",
12   coeff=1, style=0.25+blue) ;
```

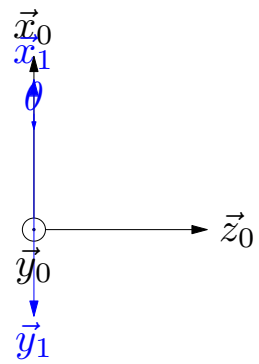




Plan $(0, \vec{x}, \vec{y})$



Plan $(0, \vec{y}, \vec{z})$



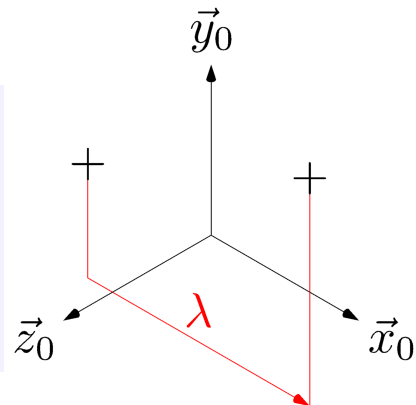
Plan $(0, \vec{z}, \vec{x})$

— `\bielleManivelle` — `\faucheuse` — `\maxpid` — `\pompePistonsAxiaux`
— `\jointCardan` — `\pilote5000` — `\sinusmatic`

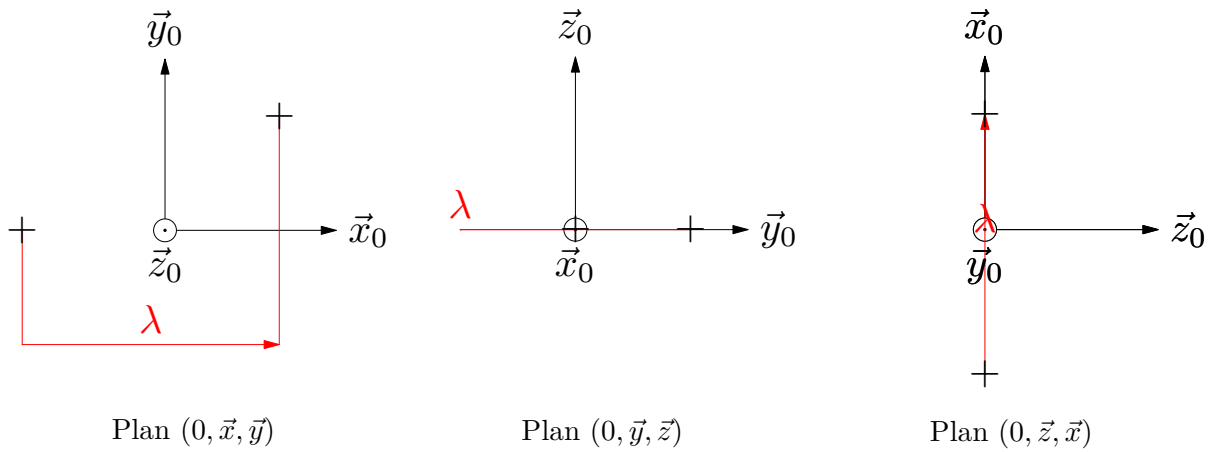
```
showParamLin(triple pointStart, triple pointEnd, triple dirDimLine, triple
  dirExtLine, string name, real offset=1, pair pos=1*N, pen
  style=black+0.25) ;
```

- `triple pointStart` : origine du vecteur
- `triple pointEnd` : fin du vecteur
- `triple dirDimLine` : vecteur directeur du paramètre
- `triple dirExtLine` : vecteur directeur des lignes d'attache
- `string name` : nom du paramètre (LaTeX compatible)
- `real offset=1` : position de la flèche par rapport au point de départ (valeur par défaut 1 – à ajuster)
- `pen style` : mise en forme de l'ensemble (valeur par défaut `black+0.25`)

```
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 showParamLin((-1.25,0,0), (1,1,0), b0.x, -b0.y,
  '$\lambda$', style = red+0.25) ;
7 namePoint((-1.25,0,0), ' ');
8 namePoint((1,1,0), ' ');
```



Perspective isométrique



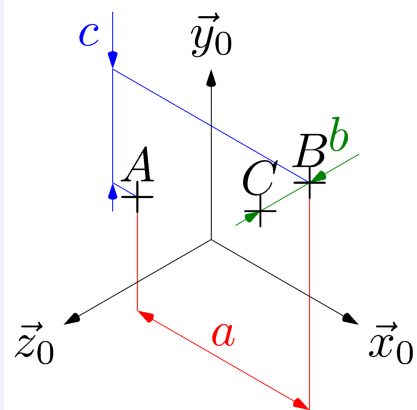
— `\bielleManivelle`

Il est également possible de montrer des cotes spécifiques par la commande :

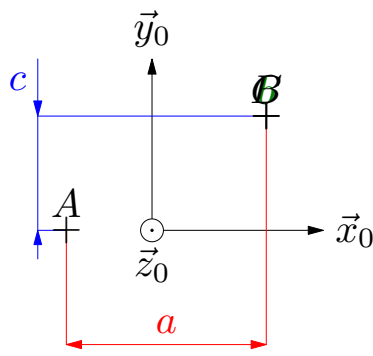
```
showDimension(triple pointStart, triple pointEnd, triple dirDimLine,
triple dirExtLine, string name, real offset=1, pair pos=1*N, pen
style=black+0.25, string posDim="middle") ;
```

- `triple pointStart` : point 1 de référence
- `triple pointEnd` : point 2
- `triple dirDimLine` : vecteur directeur de la cote
- `triple dirExtLine` : vecteur directeur des lignes d'attache
- `string name` : nom du paramètre (LaTeX compatible)
- `real offset=1` : position de la flèche par rapport au point de départ (valeur par défaut 1 – à ajuster)
- `pair pos=1*N` : position du texte par rapport à la cote
- `pen style` : mise en forme de l'ensemble (valeur par défaut `black+0.25`)
- `string posDim="middle"` : position de la cote – 3 positions sont possibles : `"middle"` – `"left"` ou `"right"`

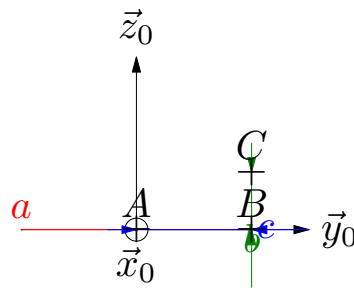
```
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 triple A = (-0.75,0,0) ; triple B = (1,1,0) ;
   triple C = (1,1,0.5);
7 showDimension(A, B, b0.x, -b0.y, '$a$', style =
   red+0.25) ;
8 showDimension(B, C, b0.z, -b0.x, '$b$', offset
   = 0, style = deepgreen+0.25, posDim="left")
   ;
9 showDimension(A, B, b0.y, -b0.x, '$c$', offset
   = 0.25, style = blue+0.25, posDim="right",
   pos=W) ;
10 namePoint(A, 'A', pos=N) ; namePoint(B, 'B',
   pos=N) ; namePoint(C, 'C', pos=N) ;
```



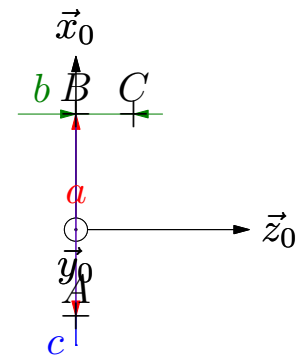
Perspective isométrique



Plan $(0, \vec{x}, \vec{y})$



Plan $(0, \vec{y}, \vec{z})$



Plan $(0, \vec{z}, \vec{x})$

— `\piloteSafran`

2.6.3 Texte

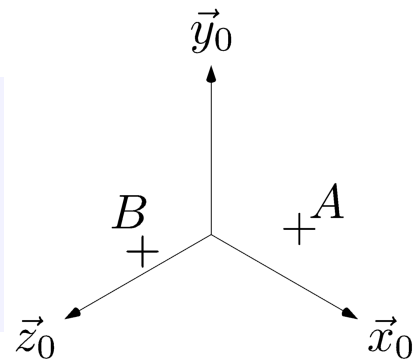
Pour afficher le nom des points, il existe une commande unique :

`namePoint(triple point, string label, pair pos=NE) ;`

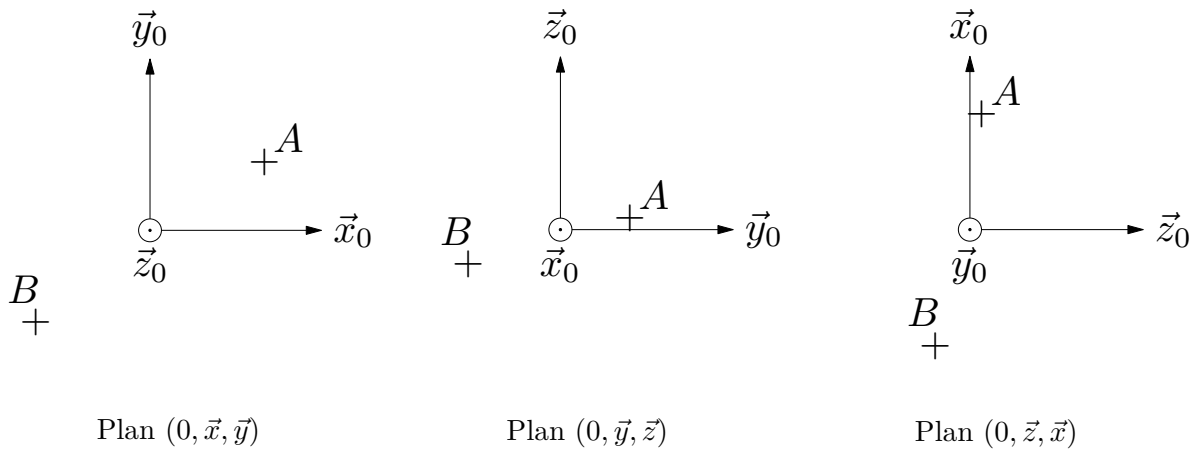
- `triple point` : point à préciser
- `pair pos=NE` : position du texte à afficher par rapport au point (voir explication ci-dessous)
- `string label` : texte à afficher

Le dernier paramètre est de type `pair` (coordonnées en 2d). C'est en fait la position sur la projection 2d de la vue 3d du nom du point. Par exemple, `NE` correspond à $(1, 1)$ soit en haut à droite du point à une distance de $\sqrt{2}$. Il y a une valeur par défaut mais qui ne marche globalement en 3d mais qui faut ajuster en 2d. Dans ce dernier cas, préférez au moins $(2, 2)$. Mais d'une manière le conseil c'est plutôt de laisser la valeur par défaut pour commencer et d'ajuster tous les points d'un coup pour éviter de compiler 30 fois le même projet.

```
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 namePoint(0+(1,0.6,0.1), 'A', pos=NE) ;
7 namePoint(0+(-1,-0.8,-0.3), 'B',
  pos=(-0.5,1.5)) ;
```



Perspective isométrique



```

— \bielleManivelle      — \falconHaptic_anim   — \pilote5000      — \SDP_anim
— \concasseur           — \faucheuse             — \piloteSafran    — 
— \DAE                  — \jointCardan           — \pinceCoupeCable — \sinusmatic
— \directionCamion_anim — \limiteurCouple    — \pompePistonsAxiaux
— \falconHaptic         — \maxpid            — \SDP              — \trainEpicycloidaux

```

Pour afficher le numéro de la classe en un point (le numéro sera entre parenthèse), utiliser la commande suivante (identique finalement à la commande précédente) :

```

nameClasse1point(string label, triple point, pair pos=1.5*NE, pen
p=currentpen) ;

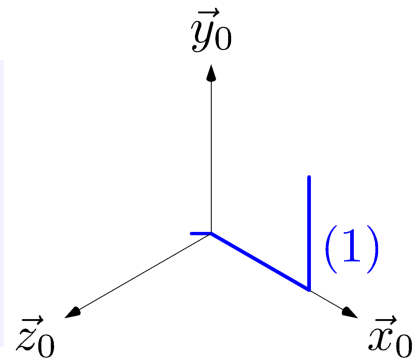
```

- `string label` : nom de la classe
- `triple point` : point origine
- `pair pos=1.5*NE` : position relative du texte par rapport au point origine
- `pen p=currentpen` : mise en forme du text (valeur par défaut le style par défaut d'asymptote)

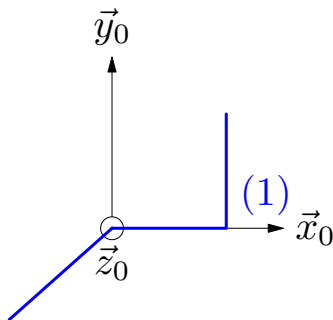
```

1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 link((-0.9,-0.8,-0.7) -- 0 -- 0+b0.x --
7       0+b0.x+b0.y, blue) ;
8 nameClasse1point("1", 0+b0.x, pos=1.5*NE,
9                   p=blue) ;

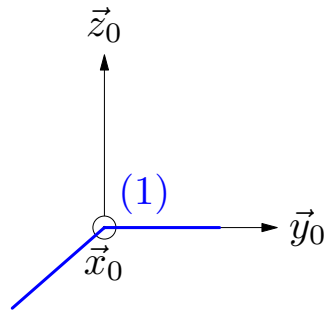
```



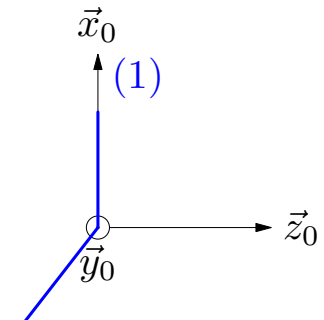
Perspective isométrique



Plan $(0, \vec{x}, \vec{y})$



Plan $(0, \vec{y}, \vec{z})$



Plan $(0, \vec{z}, \vec{x})$

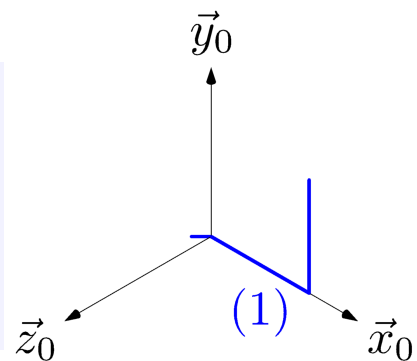
— `\bielleManivelle` — `\pilote5000` — `\pinceCoupeCable` — `\trainEpicycloidaux`
— `\faucheuse` — `\piloteSafran` — `\pompePistonsAxiaux`

Souvent, il est intéressant de placer le nom de la classe en un point milieu de deux points. La commande suivante réalise donc cela :

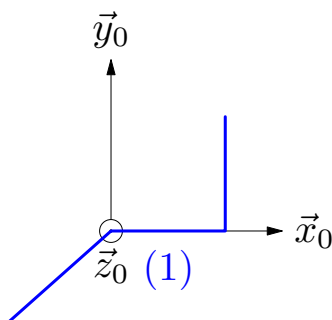
```
nameClasse2points(string label, triple point1, triple point2, pair
pos=1.5*NE, pen p=currentpen) ;
```

— `string label` : nom de la classe par rapport au point origine
— `triple point1` : point 1
— `triple point2` : point 2
— `pair pos=1.5*NE` : position relative du texte
— `pen p=currentpen` : mise en forme du text (valeur par défaut le style par défaut d'asymptote)

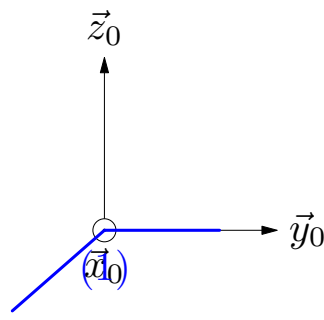
```
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 link((-0.9,-0.8,-0.7) -- 0 -- 0+b0.x --
7      0+b0.x+b0.y, blue) ;
8 nameClasse2points("1", 0, 0+b0.x, pos=1.5*S,
9                  p=blue) ;
```



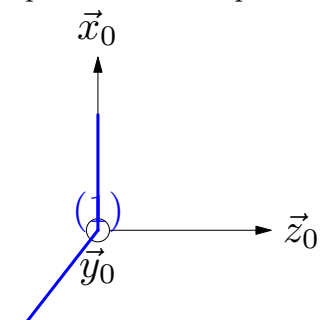
Perspective isométrique



Plan $(0, \vec{x}, \vec{y})$



Plan $(0, \vec{y}, \vec{z})$



Plan $(0, \vec{z}, \vec{x})$

— `\bielleManivelle` — `\pilote5000` — `\pinceCoupeCable` — `\trainEpicycloidaux`
— `\faucheuse` — `\piloteSafran` — `\pompePistonsAxiaux`

On peut également ajouter du texte en un point :

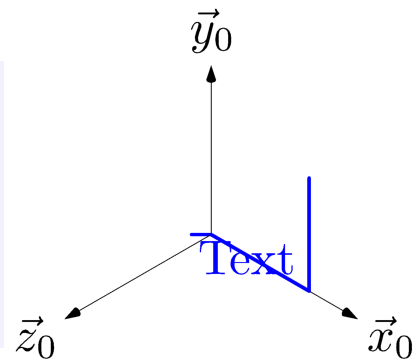
```
addText(string text, triple point, pair pos=1*N, pen style=black) ;
```

— `string text` : texte à afficher au point précisé
— `triple point` : en quel point
— `pair pos=1*N` : position relative par rapport
— `pen style=blacken` : mise en forme par défaut

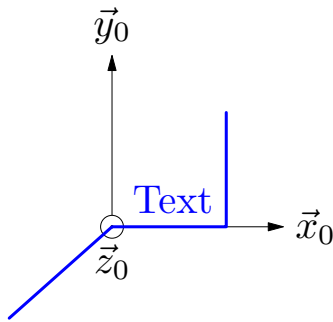
```

1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 link((-0.9,-0.8,-0.7) -- 0 -- 0+b0.x --
7      0+b0.x+b0.y, blue) ;
8 addText("Text", 0+b0.x, pos=1.5*NW, style=blue)
9 ;

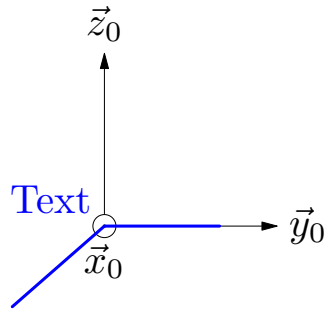
```



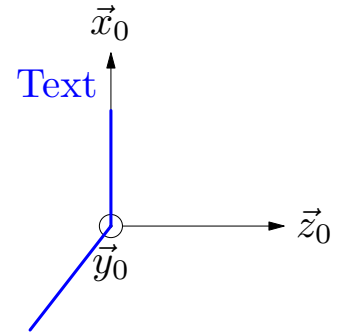
Perspective isométrique



Plan $(0, \vec{x}, \vec{y})$



Plan $(0, \vec{y}, \vec{z})$



Plan $(0, \vec{z}, \vec{x})$

— `\limiteurCouple` — `\piloteSafran`

De plus, il existe la possibilité de tracer une flèche en précisant le texte (vecteur force par exemple).

```

addArrowText(triple startPoint, triple endPoint, string name, real
posText=0.5, pair pos = 1*N, pen style = black+0.25) ;

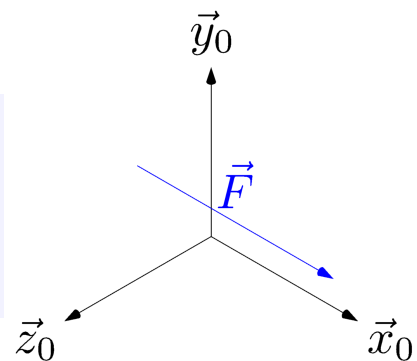
```

- `triple startPoint` : point origine
- `triple endPoint` : point pointé
- `string name` : texte à afficher (L^AT_EX compatible)
- `real posText=0.5` : position relative du texte sur le chemin `startPoint -- endPoint`
- `pair pos=1*N` : position relative du texte par rapport au point précédent
- `pen style=black+0.25` : mise en forme par défaut

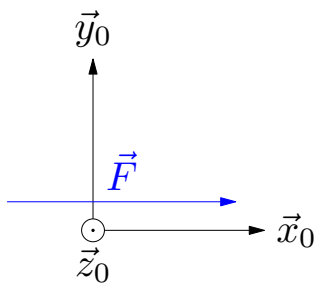
```

1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 addArrowText((-0.75, 0.25, 0), (1.25, 0.25, 0),
7              "$\vec{F}$", style = blue+0.25) ;

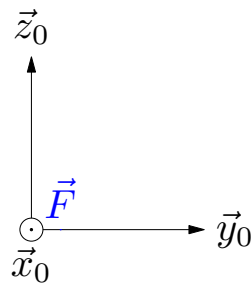
```



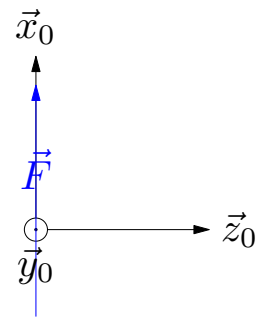
Perspective isométrique



Plan $(0, \vec{x}, \vec{y})$



Plan $(0, \vec{y}, \vec{z})$



Plan $(0, \vec{z}, \vec{x})$

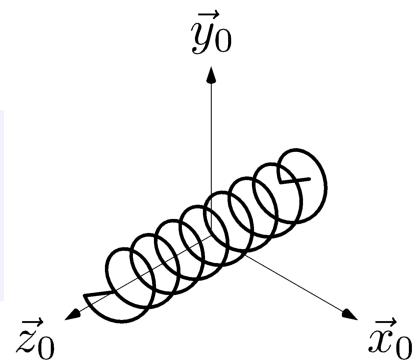
— \piloteSafran

2.7 Éléments technologiques

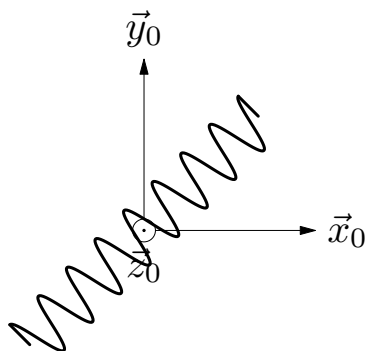
2.7.1 Ressort de traction/compression

BIB `addSpring(triple A, triple B, int N=8) ;`

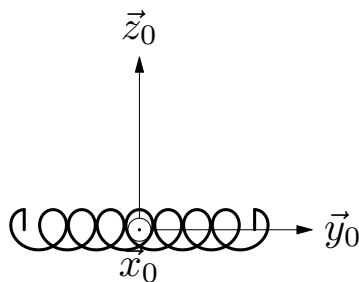
```
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 addSpring((-1,-1,0), (1,1,0)) ;
```



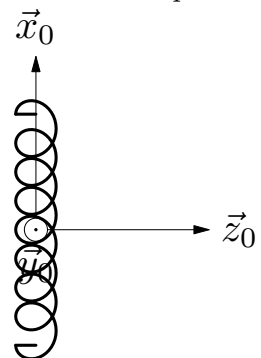
Perspective isométrique



Plan $(0, \vec{x}, \vec{y})$



Plan $(0, \vec{y}, \vec{z})$



Plan $(0, \vec{z}, \vec{x})$

— \limiteurCouple

— \pinceCoupeCable

3 Animation

N'hésitez pas à consulter l'ensemble des exemples ou la présentation rapide de la bibliothèque (avec un lecteur pdf de type adobe reader) afin de vous faire une idée des possibilités.

3.1 Boîtes englobantes

Afin d'éviter que les animations ne « sautent », dû au fait que le découpage de l'image se fait aux éléments présents dans celle-ci, il est possible de créer un élément englobant la scène (invisible). Trois solutions sont proposées.

```
BIB simpleCubeBounding(real lim) ;
BIB simpleSphereBounding(real lim) ;
BIB parallelepipedBounding(triple negativeVertex, triple positiveVertex) ;
```

```
— \bielleManivelle_anim — \DAE_anim — \falconHaptic_anim — \pilote5000
— \directionCamion_anim — \pilote5000_anim
— \croixMalte_anim — \maxpid_anim — \sinusmatic_anim
```

3.2 Animation par le package animate

La première solution naturelle pour générer des animations est d'utiliser la package `animate` et une boucle `for`. La génération doit se terminer par un ensemble de lignes de code résumées dans une seule commande :

```
BIB endAnimationPDF(animation anim) ;
```

```
— \bielleManivelle_anim — \DAE_anim — \maxpid_anim
— \I3D_anim — \sinusmatic_anim
```

3.3 Animation par un script python

Cependant, l'auteur de la bibliothèque n'a pas été convaincu par ce package qui a fait le job partiellement. Les fichiers pdf générés n'ont pas exactement le même formatage en 3d ou en 2d (présence d'un `_` [underscore] ou non) mais surtout une erreur de type `Out Of Memory` très agaçante et vite apparue (voir exemple I3D). Il existe sans aucun doute des solutions propres à Asymptote mais l'auteur a préféré utiliser un script Python.

Suite aux problèmes évoqués précédemment, un script qui vient tout simplement modifier les « bonnes » lignes d'un fichier `asy` a été la réponse à ces problèmes. Il génère (et efface) un fichier `asy` par position souhaitée. Pour avoir un rendu immédiat (sans animer avec le package `animate` de `LATEX`), une fusion de l'ensemble des pdf générés est réalisée. Néanmoins le fait de réécrire chaque fichier rend le processus tout de même plus lent (très visible en 2d, moins en 3d).

```
— \croixMalte_anim — \I3D_anim — \SDP_anim
— \directionCamion_anim — \falconHaptic_anim — \pilote5000_anim
```

4 Exemples détaillés

4.1 Le système bielle-manivelle

Partons du fichier du système bielle-manivelle en perspective isométrique pour s'approprier l'ensemble des commandes.

Ci-dessous l'entête commun à tous les scripts Asymptote de la bibliothèque.

```
1 settings.render = -4 ; // qualité de la sortie en 3D -- en 2D vectoriel
   pour le pdf
2 settings.prc = false ; // non au pdf 3D. Mais ça pourrait être intéressant.
3 import biblioLiaisons ; // import de la biblio
4 defaultpen(fontsize(10pt)); // taille de la police.
5 unitsize(1cm); //unité des grandeurs. Ne pas changer !
```

```

6 triple eye = (1,1,1) ; // point de vue de observateur regardant le point
   (0,0,0)
7 triple up = (0,1,0) ; // axe vertical -- ici +y
8 currentprojection = orthographic(eye, up, 0) ; // projection ortho
9 currentlight = nolight; // pas d'effet de lumière

```

La qualité -4 marche très bien pour moi pour mes sorties en pdf. Pour le png, je pense qu'on peut l'augmenter (-8 ou -16).

Ensuite, on peut définir les différents paramètres et les lois entrée-sortie.

```

1 // Parameters :
2 real R = 2 ;
3 real L = 3 ;
4 real theta10 = 55/360*2*pi ;
5 real theta20 = asin(-R/L*sin(theta10)) ;
6 real pos = R*cos(theta10) + L *sqrt(1-(R/L*sin(theta10))^2) ;

```

Les angles sont définis en radian et les fonctions mathématiques usuelles sont implémentées dans Asymptote.

Puis les bases :

```

1 // Basis
2 basis b1 = rotationBasis(1, b0, theta10, 'z', b0.z) ;
3 basis b2 = rotationBasis(2, b0, theta20, 'z', b0.z) ;

```

Les positions des différents points :

```

1 // Points
2 triple A = R*b1.x ;
3 triple B = pos*b0.x ;
4 real dec = 1 ;
5 triple C = B + 2*dec*b0.x ;
6 triple D = B + 4*dec*b0.x ;

```

On retrouve « notre » façon usuel de représenter des vecteurs.

Les classes d'équivalence cinématiques sont définies par de simples couleurs :

```

1 // CEC
2 pen CEC0 = black ;
3 pen CEC1 = red ;
4 pen CEC2 = purple ;
5 pen CEC3 = deepgreen ;

```

On peut ensuite habiller en reliant les différents points et placer le(s) symbole(s) du bâti :

```

1 // Link and ground link
2 real decBati = 0.75 ;
3 bati(0-decBati*b0.y, b0.x, -b0.y, CEC0) ;
4 link(0-decBati*b0.y -- 0, CEC0) ;
5 bati(C-decBati*b0.y, b0.x, -b0.y, CEC0) ;
6 link(C-decBati*b0.y -- C, CEC0) ;
7 real prof = -1 ;
8 path3 pCEC1 = 0 -- 0+prof*b0.z -- A+prof*b0.z -- A ;
9 link(pCEC1, CEC1) ;
10 link(A--B, CEC2) ;
11 link(B--D, CEC3) ;

```

Ce passage est plus ou moins simple en fonction du mécanisme.

Enfin ajoutons les liaisons (pour qu'elles soient positionnées au-dessus du reste) :

```

1 // Liaisons
2 liaisonPivot(0, b0.z, b0.x, CEC0, CEC1) ;
3 liaisonPivotGlissant(A, b0.z, CEC2, CEC1) ;
4 liaisonRotule(B, -b0.x, CEC2, CEC3) ;
5 liaisonGlissiere(C, b0.x, b0.y, CEC0, CEC3) ;

```

Pour un rendu visuel habituel, ajoutons le cylindre du piston :

```
1 // Formes supplémentaires
2 addCylinder(D, b0.x, 0.35, 0.25, CEC3) ;
```

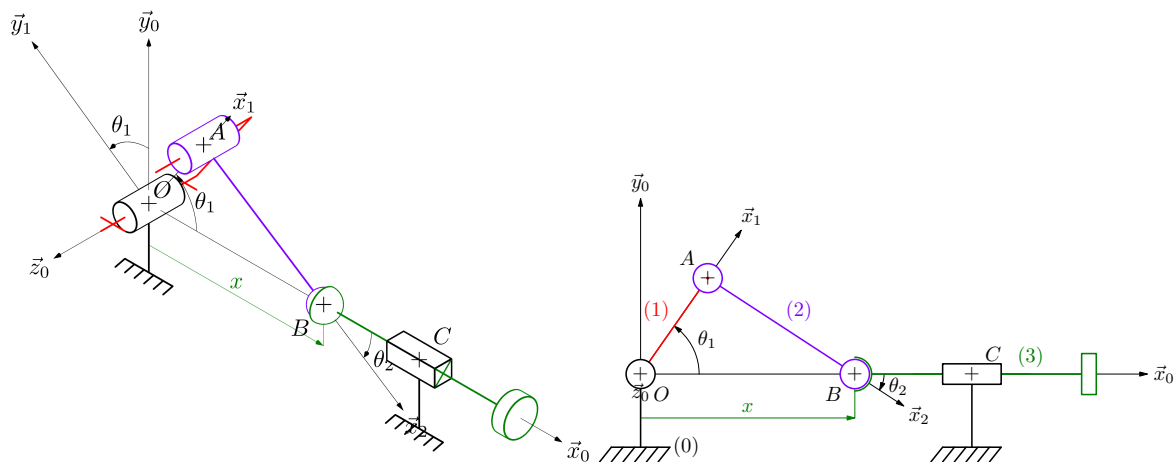
Finissons le script par l'affichage des bases et des paramètres :

```
1 // Bases et paramétrages
2 showBasis(b0, 0, coeff=(length(D-0)+1,R+1,2)) ;
3 int[] tabIndices = {0,1} ;
4 showAxis(b1, tabIndices, 0, R+1, style=black+0.25) ;
5 showParamAng(0, b0.x, b1.x, "$\theta_{1}$") ;
6 showParamAng(0, b0.y, b1.y, "$\theta_{1}$") ;
7 int[] tabIndices = {0} ;
8 showAxis(b2, tabIndices, B, 2, style=black+0.25) ;
9 showParamAng(B, b0.x, b2.x, "$\theta_{2}$") ;
10 showParamLin(0, B, b0.x, -b0.y, "$x$", offset=0.75, pos=1*N,
    style=CEC3+0.25) ;
```

et le noms des points en conclusion :

```
1 // Noms des points
2 namePoint(0,"O",NE) ;
3 namePoint(A,"A",NE) ;
4 namePoint(C,"C",(2,2)) ;
5 namePoint(B,"B",(-2,-2)) ;
```

L'ensemble de ces commandes génère le fichier pdf suivant (à gauche) alors qu'une ne modifiant que la ligne `triple eye = (1,1,1);` par `triple eye = (0,0,1);`, on obtient le schéma en projection plane (à droite).



4.2 Usage avancé : couplage avec Sympy – direction de camion

L'idée était la suivante : proposer un exemple à plusieurs mobilités où l'écriture des lois entrées sorties étaient relativement pénibles mais où il était en plus question de résolution numérique.

Pour cela, je vous laisse fouiller l'exemple sur la direction de camion. Dans un esprit de synthèse :

1. on utilise `sympy` afin de déterminer les lois entrées-sorties ;
2. on pose le problème numérique en utilisant cette fois-ci le module `numpy` (utilisation d'un algorithme hybride de résolution d'équation de type $F(X) = 0$) ;
3. résolution numérique pour une mobilité en fixant l'autre ;
4. création du schéma cinématique dans la position encours en utilisant la bibliothèque proposée dans cette documentation `biblioLiaisons3d2d` ;
5. on recommence pour la seconde mobilité.

À titre d'exemple voici l'animation obtenue pour la mobilité liée au volant :

5 Pour aller plus loin

5.1 VSCode

5.1.1 Snippets

J'ai un fichier `.json` de snippets propre à vscode (si vous l'utilisez d'ailleurs vous êtes chanceux car ça va plus vite quand même). La procédure est ici : You can easily define your own snippets without any extension. To create or edit your own snippets, select Configure User Snippets under File > Preferences (Code > Preferences on macOS), and then select the language (by language identifier) for which the snippets should appear, or the New Global Snippets file option if they should appear for all languages. VS Code manages the creation and refreshing of the underlying snippets file(s) for you.

Vous pouvez ensuite copier coller le contenu.

5.1.2 Run bat file from vscode

To create a launch.json file, click the create a launch.json file link in the Run start view. Puis copier coller le contenu du fichier `launch.json` en adaptant bien sûr le nom du fichier.

5.2 Rotule à doigt

À rédiger un jour proprement mais pour faire simple j'ai dû créer la pièce en CAO pour tracer les arêtes de contour. Pour faire cela, le fichier a été converti au format `stl`, puis j'ai codé un algo de détection de contours. plus tout le reste . Plus encore quelques autres petites choses. Mais globalement ça fait le taf, mais clairement pas performant : un très mauvais $O(n^2)$.

6 Commandes

A	
addArrowText	35
addCylinder	20
addDisque	21
addSphere	23
addSpring	36
addSurfConiqueTronquee	22
addSurfCylinder	23
addSurfPlane	25
addText	34
addUshape	24
B	
bati	20
C	
createBasis	5
E	
endAnimationPDF	37
I	
importSTL	26
L	
liaisonAppuiPlan	11
liaisonGlissiere	6
liaisonHelicoidale	7
liaisonLineaireAnnulaire	11
liaisonLineaireRectiligne	12
liaisonPivot	6
liaisonPivotGlissant	8

liaisonPonctuelle	13
liaisonRotule	10
liaisonRotuleDoigt	9
link	19

N	
nameClasse1point	33
nameClasse2points	34
namePoint	32

P	
parallelepipedBounding	37

R	
rotationBasis	5

S	
showAxis	27
showAxisName	28
showBasis	27
showDimension	31
showParamAng	29
showParamLin	30
simpleCubeBounding	37
simpleSphereBounding	37

T	
transChaine	18
transEngrenages	14
transPignonCremailiere	17
transPoulieCourroie	16
transRoueVis	17