

# Aide de la bibliothèque `biblioLiaisons3d2d`

Anthony Meurdefroid

14 juin 2024

## Table des matières

<b>1</b>	<b>Description de la bibliothèque</b>	<b>3</b>
1.1	Que fait cette bibliothèque ? . . . . .	3
1.2	Installation . . . . .	3
1.3	Méthologie . . . . .	3
<b>2</b>	<b>Liste des commandes de la bibliothèque</b>	<b>4</b>
2.1	Les objets . . . . .	4
2.2	Points, bases et CEC . . . . .	4
2.2.1	Points . . . . .	4
2.2.2	Bases . . . . .	4
2.2.3	CEC . . . . .	5
2.3	Les liaisons . . . . .	5
2.3.1	Glissière . . . . .	5
2.3.2	Pivot . . . . .	6
2.3.3	Helicoïdale . . . . .	6
2.3.4	Pivot glissant . . . . .	7
2.3.5	Rotule à doigt . . . . .	8
2.3.6	Rotule . . . . .	9
2.3.7	Appui-plan . . . . .	10
2.3.8	Linéaire annulaire . . . . .	10
2.3.9	Linéaire rectiligne . . . . .	11
2.3.10	Ponctuelle . . . . .	12
2.4	Transmission de puissance . . . . .	13
2.4.1	Engrenages . . . . .	13
2.4.2	Poulie courroie . . . . .	15
2.4.3	Roue et vis sans fin . . . . .	16
2.4.4	Pignon crémaillère . . . . .	16
2.4.5	Chaîne . . . . .	17
2.5	Habillage . . . . .	18
2.5.1	Relier . . . . .	18
2.5.2	Bâti . . . . .	19
2.5.3	Cylindre . . . . .	19
2.5.4	Disque . . . . .	20
2.5.5	Surface conique tronquée . . . . .	21
2.6	Paramétrage . . . . .	22
2.6.1	Bases . . . . .	22
2.6.2	Texte . . . . .	24
2.7	Éléments technologiques . . . . .	27
2.7.1	Ressort de traction/compression . . . . .	27

<b>3</b>	<b>Animation</b>	<b>27</b>
3.1	Boîtes englobantes . . . . .	27
3.2	Animation par le package <code>animate</code> . . . . .	28
3.3	Animation par un script python . . . . .	28
<b>4</b>	<b>Exemples détaillés</b>	<b>28</b>
4.1	Le système bielle-manivelle . . . . .	28
4.2	Usage avancé : couplage avec Sympy – direction de camion . . . . .	30
<b>5</b>	<b>Pour aller plus loin</b>	<b>30</b>
5.1	VSCode . . . . .	30
5.1.1	Snippets . . . . .	30
5.1.2	Run bat file from vscode . . . . .	31
5.2	Rotule à doigt . . . . .	31

# 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  $\text{\LaTeX}$ . 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 un  $\text{\LaTeX}$  pour en profiter.

## 1.2 Installation

Ma distribution  $\text{\LaTeX}$ , même après une mise à jour, ne proposait pas la dernière version d'asymptote : le langage utilisé <https://asymptote.sourceforge.io/>. J'ai dû installer à la main la dernière version. Il faut également installer la dernière version de ghostscript <https://ghostscript.com/releases/gsdnld.html>.

Pour les utilisateurs de Word, 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`).

Enfin pour l'étape de compilation, je conseille de créer un fichier `bat` (sous windows évidemment). Pour ma part, j'utilise VSCode et je peux exécuter ce fichier `.bat` en appuyant sur  (voir dans le dossier `.vscode`, le fichier `json`).

Enfin dans le répertoire d'Asymptote, copier coller le contenu du dossier `src : biblioLiaisons.asy`, `rotuleCreuxDoigt.stl`, `rotuleCreuxDoigtRapide.stl` et `STLforBLM.asy`.

## 1.3 Méthologie

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...).
- `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

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) ;
```

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

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

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 :

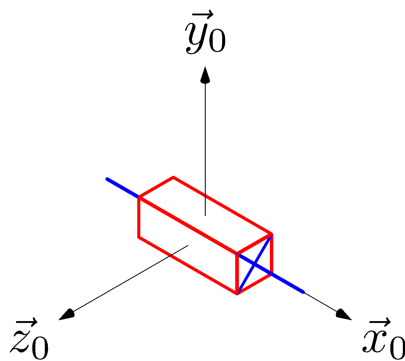
```
1 settings.render = -4 ;
2 settings.prc = false ;
3 import biblioLiaisons ;
4 defaultpen(fontsize(10pt)) ;
5 unitsize(1cm) ;
```

### 2.3.1 Glissière

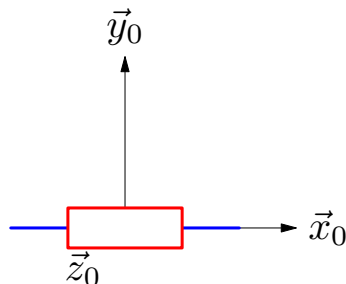
```
liaisonGlissiere(triple center, triple direction, triple orientation, pen
c1, pen c2) ;
```

- `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)

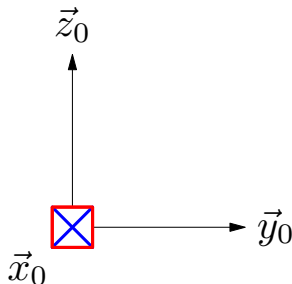
```
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 liaisonGlissiere(0,b0.x, b0.y, red, blue) ;
```



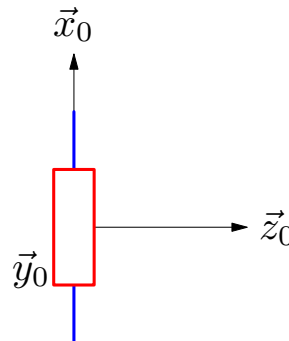
Perspective isométrique



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



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



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

— <code>\bielle_manivelle</code>	— <code>\DAE_anim</code>	— <code>\SDP</code>
— <code>\bielle_manivelle_anim</code>	— <code>\pilote5000</code>	
— <code>\DAE</code>	— <code>\pilote5000_anim</code>	— <code>\SDP_anim</code>

### 2.3.2 Pivot

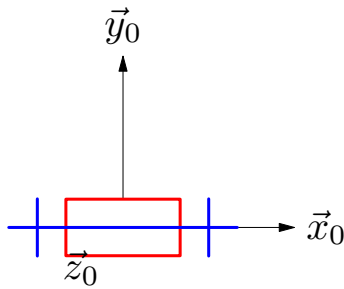
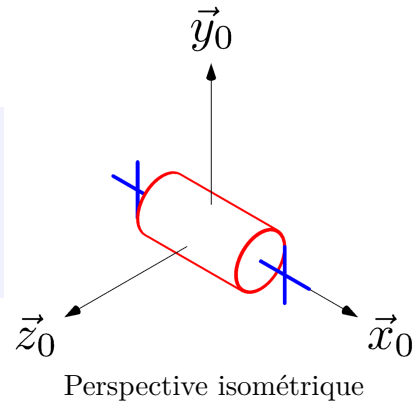
**BIB** `liaisonPivot(triple point, triple axis, triple stopAxis, pen c1, pen c2) ;`

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

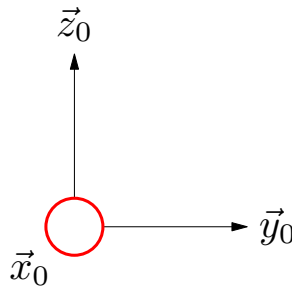
```

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) ;

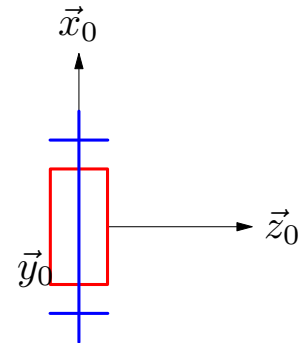
```



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>\bielle_manivelle</code>	— <code>\I3D</code>	— <code>\pompePistonsAxiaux</code>
— <code>\bielle_manivelle_anim</code>	— <code>\I3D_anim</code>	— <code>\SDP</code>
— <code>\concasreur</code>	— <code>\maxpid</code>	— <code>\SDP_anim</code>
— <code>\DAE</code>	— <code>\maxpid_anim</code>	— <code>\sinusmatic</code>
— <code>\DAE_anim</code>	— <code>\pilote5000</code>	— <code>\sinusmatic_anim</code>
— <code>\direction_camion</code>	— <code>\pilote5000_anim</code>	— <code>\trainEpicycloidaux</code>
— <code>\forcebat</code>	— <code>\pinceCoupeCable</code>	

### 2.3.3 Helicoïdale

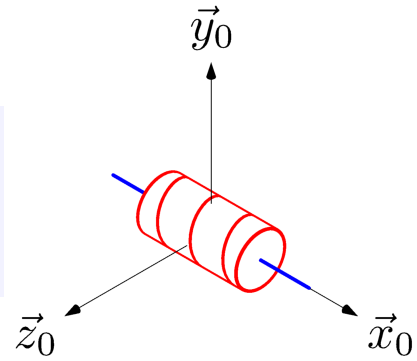
**BIB** `liaisonHelicoidale(triple point, triple axis, pen c1, pen c2) ;`

- `triple point` : le centre de la liaison (lindre + hélicoïde)
- `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)

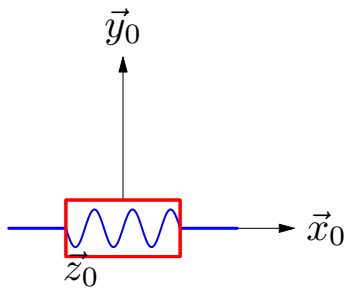
```

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 liaisonHelicoidale(0, b0.x, red, blue) ;

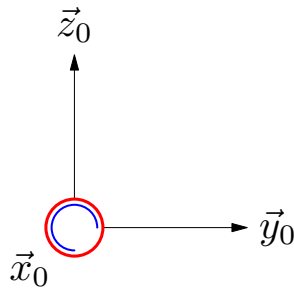
```



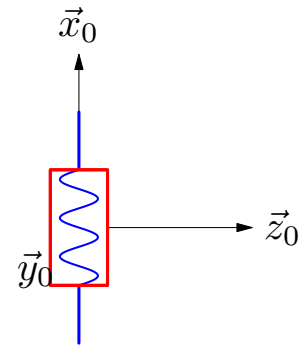
Perspective isométrique



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



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



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

— `\maxpid`

— `\maxpid_anim`

### 2.3.4 Pivot glissant

BIB

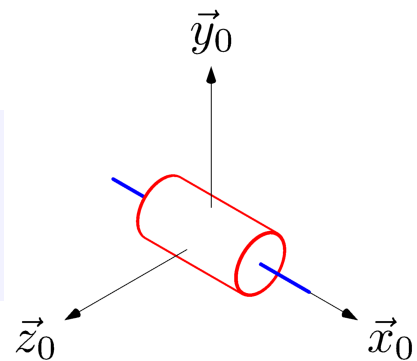
```
liaisonPivotGlissant(triple point, triple axis, pen c1, pen c2) ;
```

- `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)

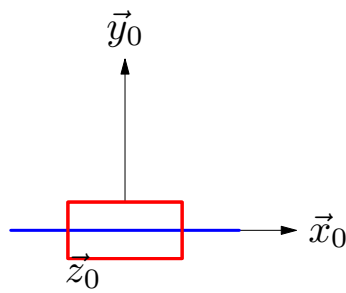
```

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) ;

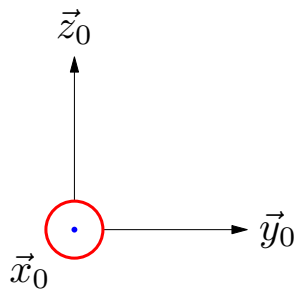
```



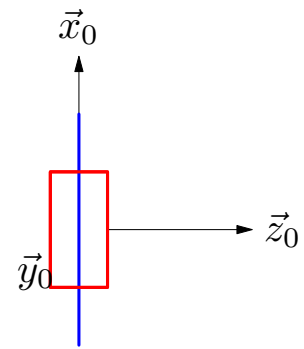
Perspective isométrique



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



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



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

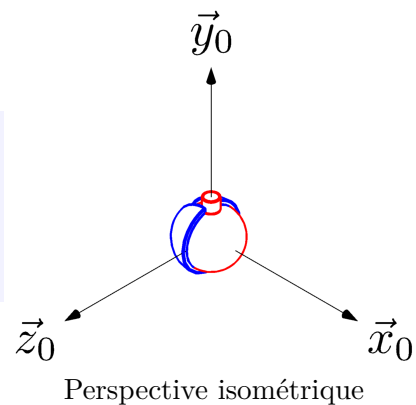
—  \bielle_manivelle	—  \I3D_anim	—  \sinusmatic
—  \bielle_manivelle_anim	—  \pinceCoupeCable	
—  \I3D	—  \pompePistonsAxiaux	—  \sinusmatic_anim

### 2.3.5 Rotule à doigt

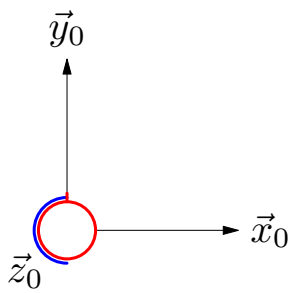
```
liaisonRotuleDoigt(triple centre, triple axis, triple tige, pen c1, pen
c2, bool rapide=true, triple obs=currentprojection.camera) ;
```

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

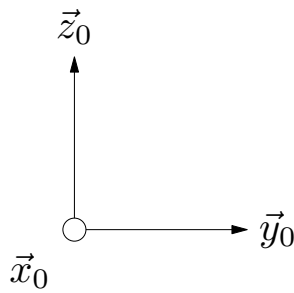
```
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 liaisonRotuleDoigt(0, b0.x, b0.y, red, blue) ;
```



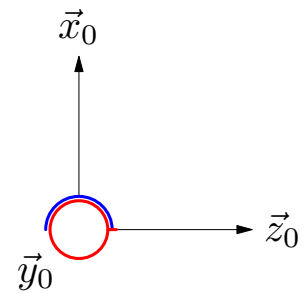




Plan  $(0, \vec{x}, \vec{y})$



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

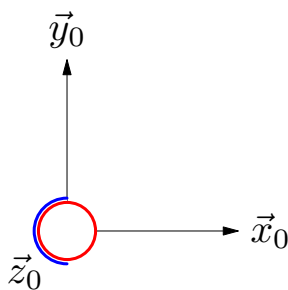
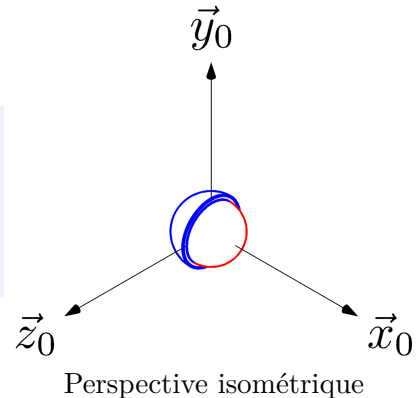
— .\DAE\_anim

### 2.3.6 Rotule

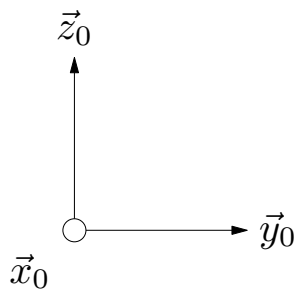
**BIB** liaisonRotule(triple center, triple axis, pen c1, pen c2) ;

- **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)

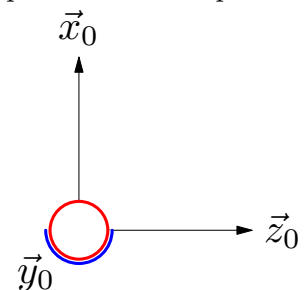
```
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 liaisonRotule(0, b0.x, red, blue) ;
```



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 :-)).

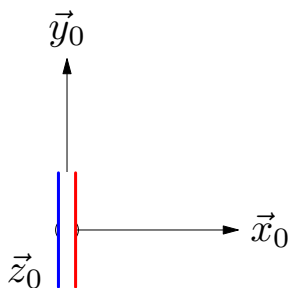
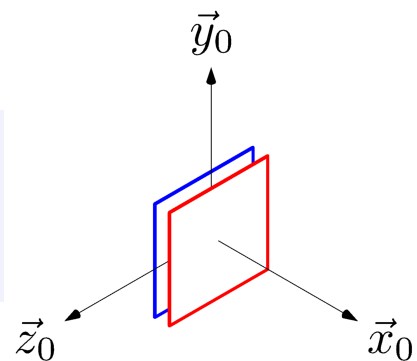
— <code>\bielle_manivelle</code>	— <code>\direction_camion</code>	— <code>\SDP_anim</code>
— <code>\bielle_manivelle_anim</code>	— <code>\I3D</code>	
— <code>\concasseur</code>	— <code>\I3D_anim</code>	— <code>\sinusmatic</code>
— <code>\DAE</code>	— <code>\pompePistonsAxiaux</code>	
— <code>\DAE_anim</code>	— <code>\SDP</code>	— <code>\sinusmatic_anim</code>

### 2.3.7 Appui-plan

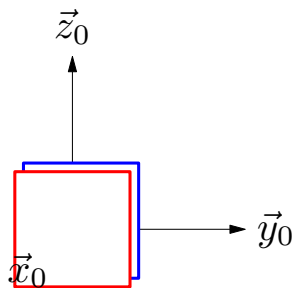
```
liaisonAppuiPlan(triple center, triple normal, triple orientation, pen c1,
                 pen c2) ;
```

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

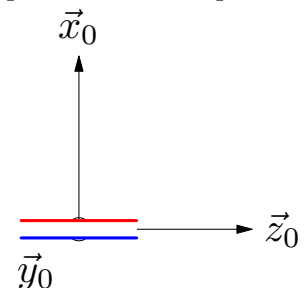
```
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 liaisonAppuiPlan(0,b0.x, b0.y, red, blue) ;
```



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



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



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

— `\pompePistonsAxiaux`

### 2.3.8 Linéaire annulaire

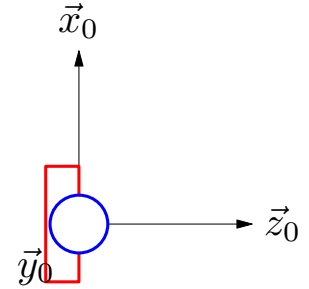
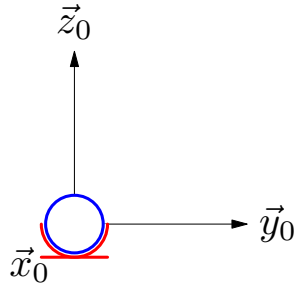
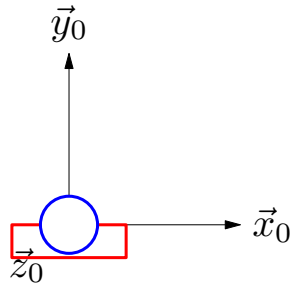
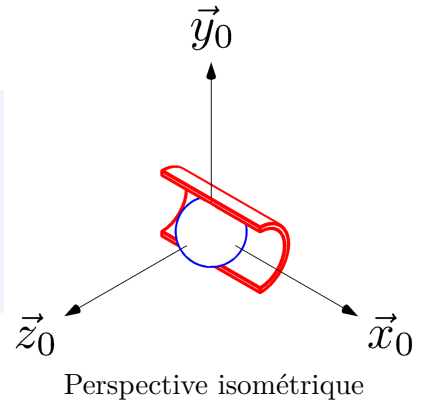
```
liaisonLineaireAnnulaire(triple center, triple direction, triple cdc, pen
                          c1, pen c2)
```

- |   |  |
|---|--|
| — <code>triple center</code> : centre de la liaison           | cylindre   |
| — <code>triple direction</code> : axe de translation possible | — <code>pen c1</code> : mise en forme de la classe 1 (demi-cylindre) |
| — <code>triple cdc</code> : représente le côté du demi-       | — <code>pen c2</code> : 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

— .\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

la droite de contact

— **triple** normale : normale au plan de contact

— **pen** c1 : mise en forme de la classe 1 (prisme triangulaire)

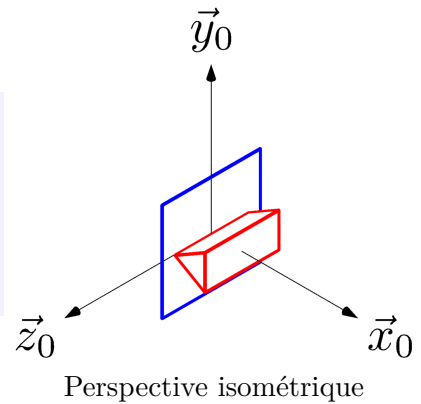
— **triple** droiteContact : vecteur directeur de

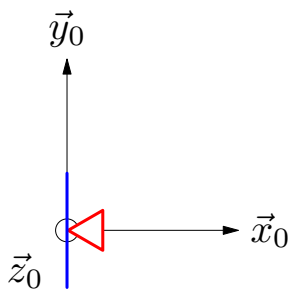
— **pen** c2 : mise en forme de la classe 2 (plan)

```

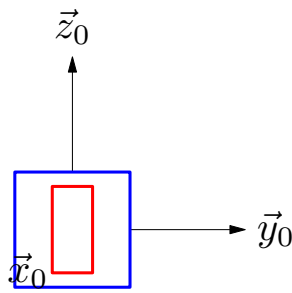
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 liaisonLineaireRectiligne(0, b0.x, b0.z, red,
    blue) ;

```

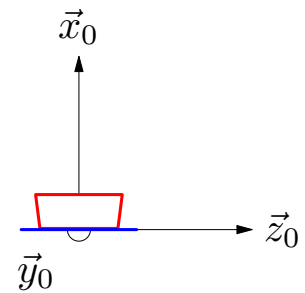




Plan  $(0, \vec{x}, \vec{y})$



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



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

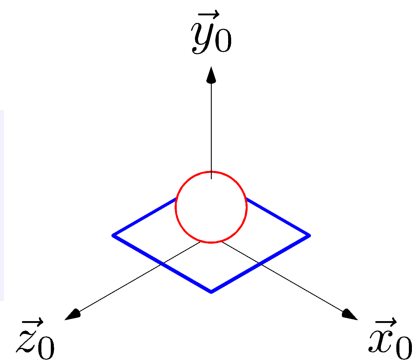
— `\pinceCoupeCable`

### 2.3.10 Ponctuelle

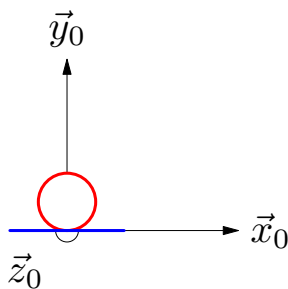
```
liaisonPonctuelle(triple center, triple normal, triple orientation, pen
c1, pen c2) ;
```

- `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)

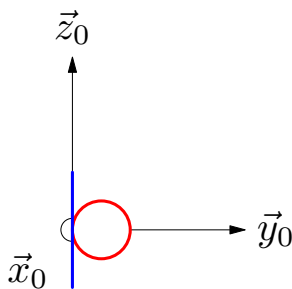
```
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 liaisonPonctuelle(0, b0.y, b0.z, red, blue) ;
```



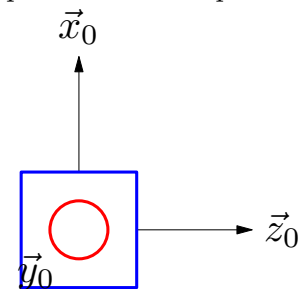
Perspective isométrique



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



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



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

— `\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) ;
```

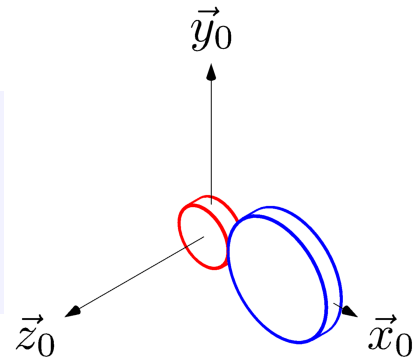
Cette fonction unique gère les contacts intérieur et extérieur, ainsi que les axes parallèles et axes concourants!

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

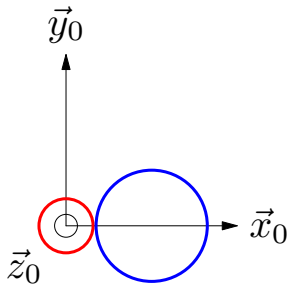
- `triple c1` : centre de la poulie 1
- `triple n1` : vecteur directeur de la poulie 1
- `real r1` : rayon de la poulie 1
- `pen CEC1` : mise en forme de la classe 1
- `triple c2` : centre de la poulie 2
- `triple n2` : vecteur directeur de la poulie 2
- `pen CEC2` : mise en forme de la classe 2

Remarque : `r2` est automatiquement calculé.

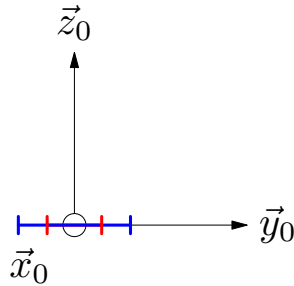
```
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 transEngrenages(0, b0.z, 0.25, red,
0+(0.75,0,0), b0.z, blue) ;
```



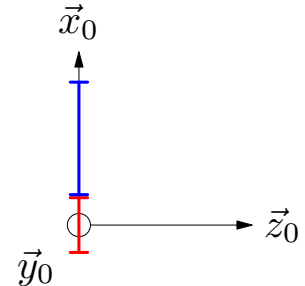
Perspective isométrique



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



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

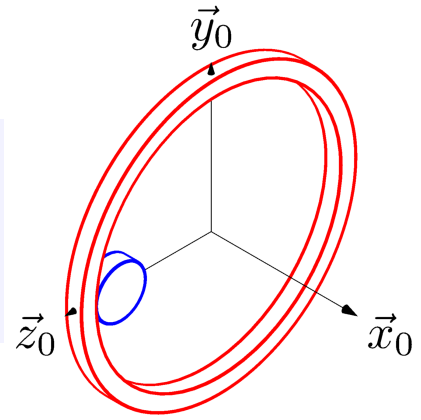


Plan  $(0, \vec{z}, \vec{x})$

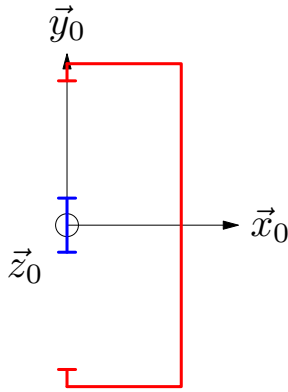
```

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) ;

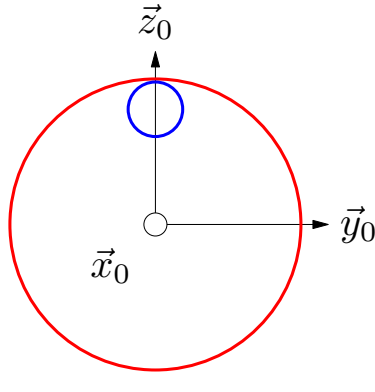
```



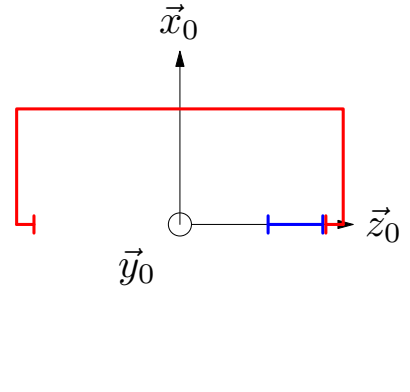
Perspective isométrique



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



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



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

Pour les engrenages à axes concourants :

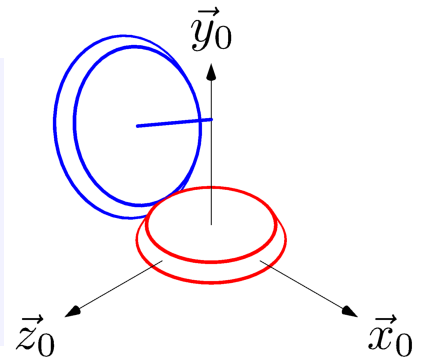
- `triple c1` : centre de la poulie 1
- `triple n1` : vecteur directeur de la poulie 1
- `real r1` : rayon de la poulie 1
- `pen CEC1` : mise en forme de la classe 1
- `triple c2` : un point quelconque de l'axe de rotation (le vrai centre sera calculé)
- `triple n2` : vecteur directeur de la poulie 2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

Remarque : `r2` est également automatiquement calculé.

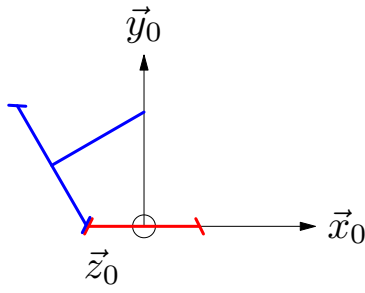
```

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) ;

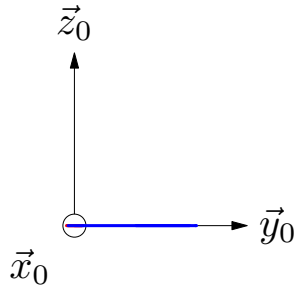
```



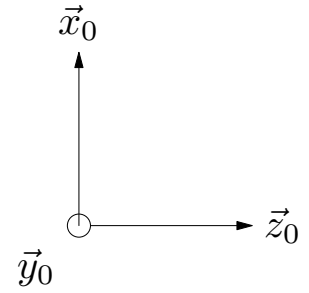
Perspective isométrique



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

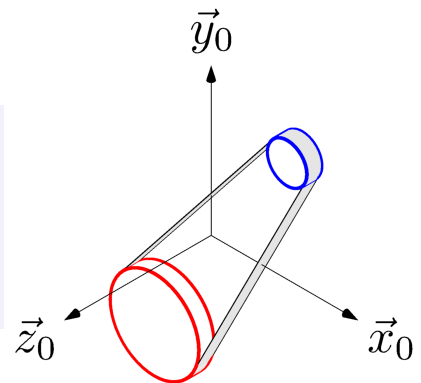
— **triple** c2 : centre de la poulie 2

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

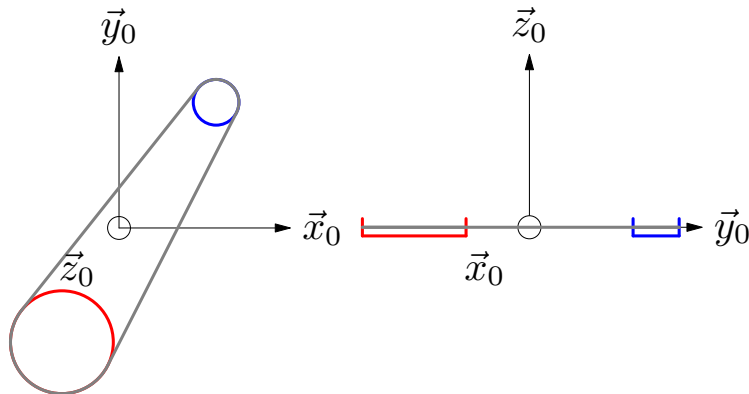
— **real** r2 : rayon de la poulie 2

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

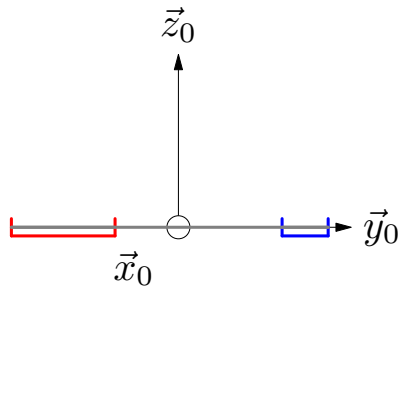
```
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 transPoulieCourroie(0+(-0.5,-1,0), b0.z, 0.45,
    red, 0+(0.85,1.1,0), b0.z, 0.2, blue) ;
```



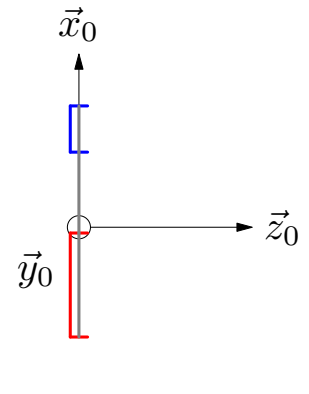
Perspective isométrique



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



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



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

— .\concasseur

— .\SDP

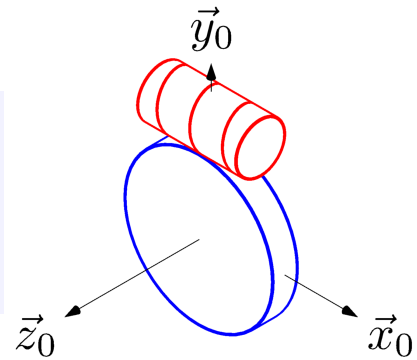
— .\SDP\_anim

### 2.4.3 Roue et vis sans fin

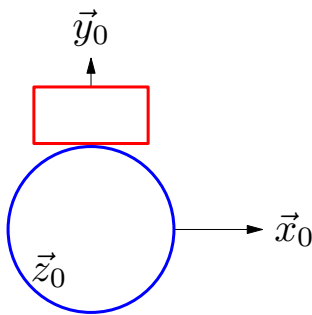
```
transRoueVis(triple c1, triple n1, pen CEC1, triple c2, triple n2, pen
CEC2) ;
```

- `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)

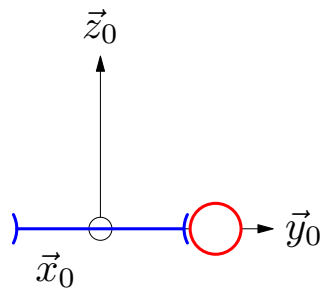
```
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 transRoueVis(0 + (0,1,0), b0.x, red, 0, b0.z,
blue) ;
```



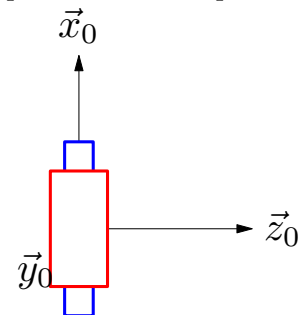
Perspective isométrique



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) ;
```

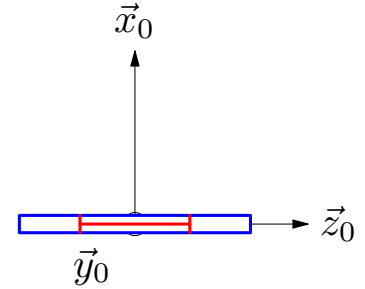
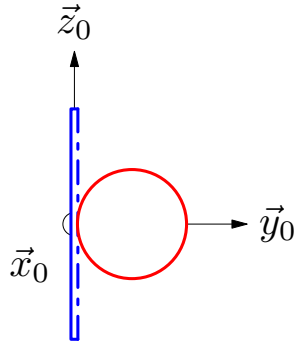
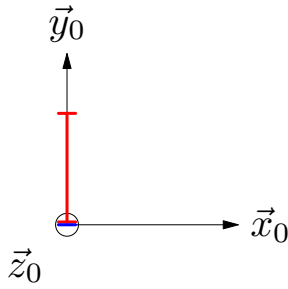
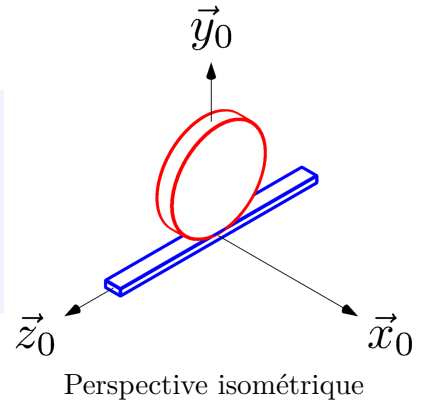
- `triple c1` : centre du pignon
- `triple n1` : axe du pignon
- `pen CEC1` : mise en forme de la classe 1 (pignon)
- `triple c2` : « centre » de la crémaillère tel que la distance entre c1 et c2 correspond au rayon de la relation cinématique
- `triple n2` : vecteur directeur de la crémaillère
- `real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

```



— .\DAE

— .\DAE\_anim

## 2.4.5 Chaîne

```

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

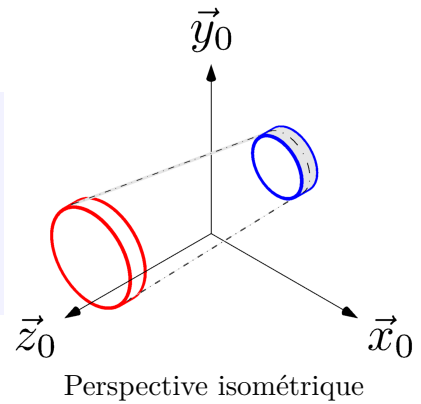
```

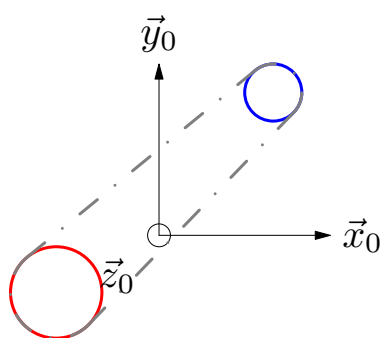
- |  |  |
|--|--|
| — <b>triple</b> c1 : centre de la roue 1       | — <b>triple</b> c2 : centre de la roue 2       |
| — <b>triple</b> n1 : axe de la roue 1          | — <b>triple</b> n2 : axe de la roue 2          |
| — <b>real</b> r1 : rayon de la roue 1          | — <b>real</b> r2 : rayon de la roue 2          |
| — <b>pen</b> CEC1 : mise en forme de la roue 1 | — <b>pen</b> CEC2 : mise en forme de la roue 2 |

```

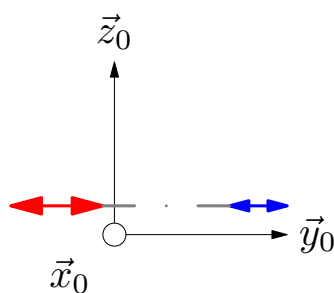
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 transChaine(0+(-0.9,-0.5,0.25), b0.z, 0.4, red,
    0+(1,1.25,0.25), b0.z, 0.25, blue) ;

```

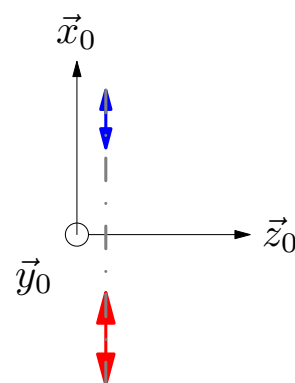




Plan  $(0, \vec{x}, \vec{y})$



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) ;`

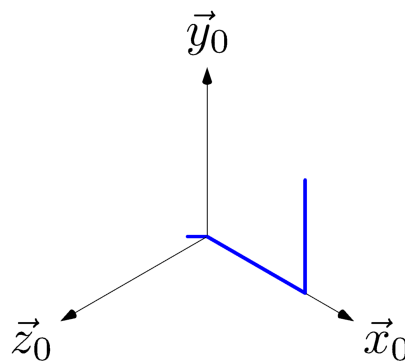
— `path3 chemin` : le chemin à tracer

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

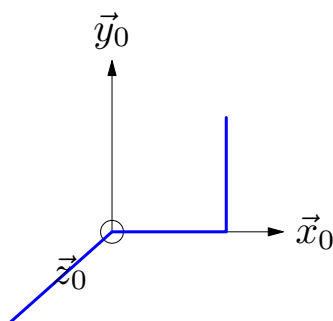
Pour relier une commande unique donc. Un `path3` est donc constitué de points (objet `triple`) reliés entre eux par `--` (en vrai il y a donc possibilités mais pour cette bibliothèque dans un premier temps c'est suffisant).

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

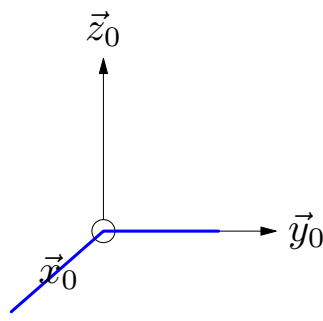
```
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 link((-0.9,-0.8,-0.7) -- 0 -- 0+b0.x --
      0+b0.x+b0.y, blue) ;
```



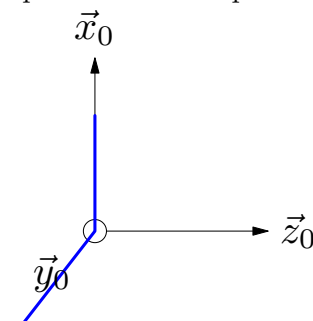
Perspective isométrique



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



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



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

— <code>\bielle_manivelle</code>	— <code>\I3D</code>	— <code>\pompePistonsAxiaux</code>
— <code>\bielle_manivelle_anim</code>	— <code>\I3D_anim</code>	— <code>\SDP</code>
— <code>\concasseur</code>	— <code>\maxpid</code>	— <code>\SDP_anim</code>
— <code>\DAE</code>	— <code>\maxpid_anim</code>	— <code>\sinusmatic</code>
— <code>\DAE_anim</code>	— <code>\pilote5000</code>	— <code>\sinusmatic_anim</code>
— <code>\direction_camion</code>	— <code>\pilote5000_anim</code>	— <code>\trainEpicycloidaux</code>
— <code>\forcebat</code>	— <code>\pinceCoupeCable</code>	

### 2.5.2 Bâti

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

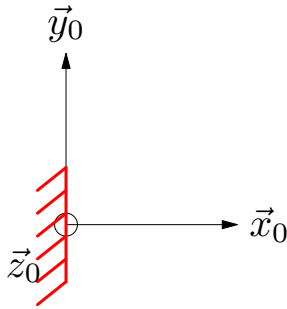
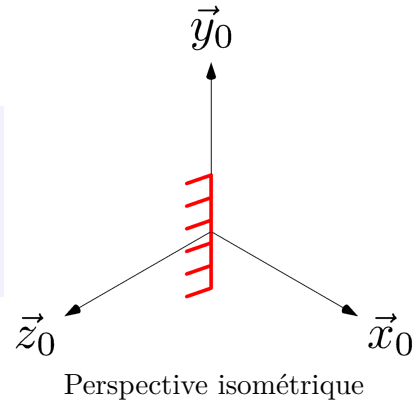
- |  |   |
|--|---|
| — <code>triple point</code> : point milieu de la grande barre        | — <code>triple orPB</code> : direction (côté) où sera dessiné le râteau par rapport à la grande barre |
| — <code>triple dirGB</code> : direction de la grande barre du râteau | — <code>pen CEC</code> : mise en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

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

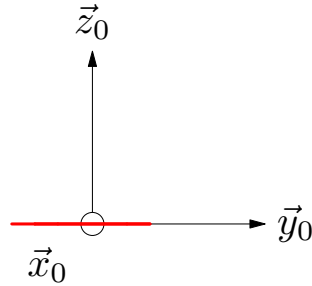
```

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) ;

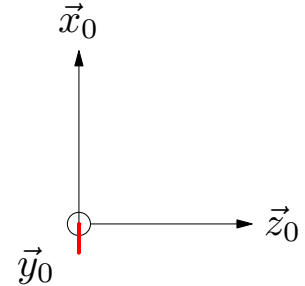
```



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



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



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

— <code>\bielle_manivelle</code>	— <code>\maxpid</code>	— <code>\SDP</code>
— <code>\bielle_manivelle_anim</code>	— <code>\maxpid_anim</code>	— <code>\SDP_anim</code>
— <code>\concasseur</code>	— <code>\pilote5000</code>	— <code>\sinusmatic</code>
— <code>\DAE</code>	— <code>\pilote5000_anim</code>	— <code>\sinusmatic_anim</code>
— <code>\DAE_anim</code>	— <code>\pinceCoupeCable</code>	— <code>\trainEpicycloidaux</code>
— <code>\forcebat</code>	— <code>\pompePistonsAxiaux</code>	

### 2.5.3 Cylindre

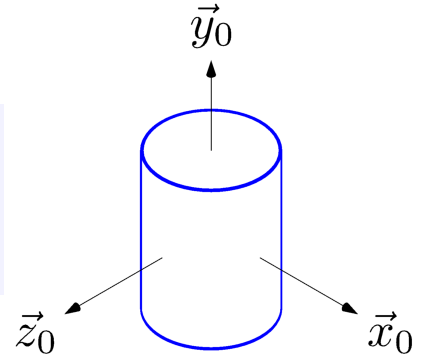
**BIB** `addCylinder(triple point, triple axis, real r, real h, pen CEC) ;`

- **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

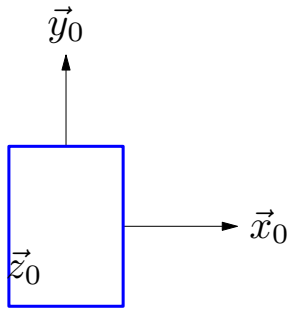
```

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) ;

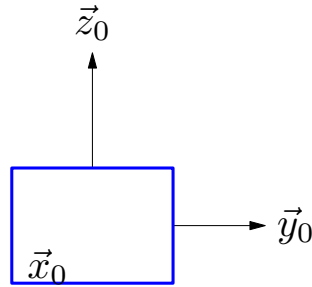
```



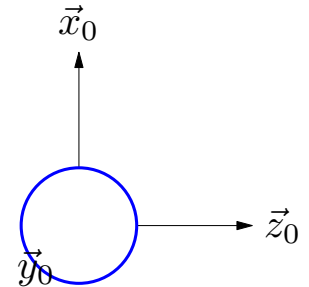
Perspective isométrique



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



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



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

```

— \bielle_manivelle
— \bielle_manivelle_anim
— \maxpid
— \maxpid_anim

```

## 2.5.4 Disque

```

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

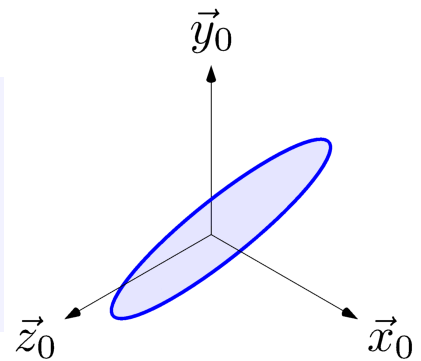
```

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

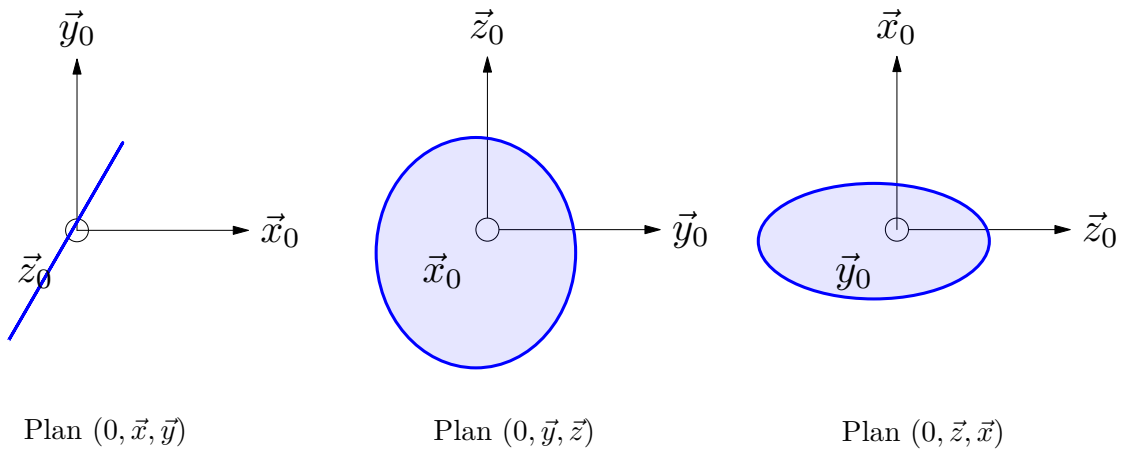
```

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



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

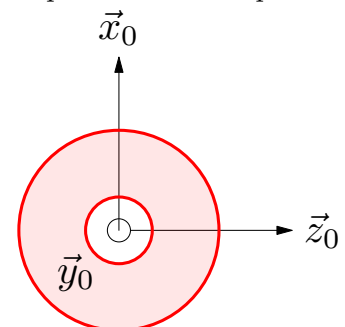
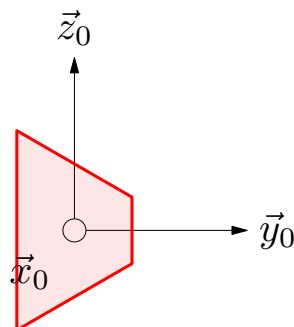
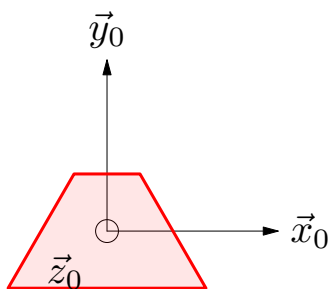
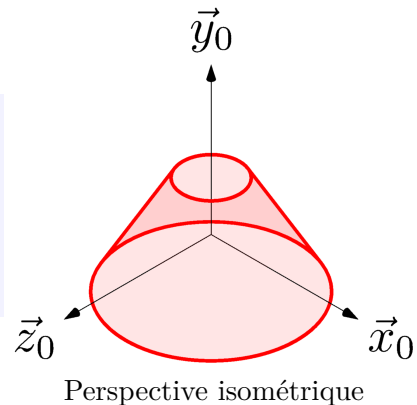
Remarque : dans l'intérêt d'une utilisation 3d, l'intérieur du disque est transparent.

— `\direction_camion`      — `\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) ;
```



- `triple` sommet : sommet de la surface conique
- `triple` axis : axe de révolution de la surface
- `face`
- `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`

## 2.6 Paramétrage

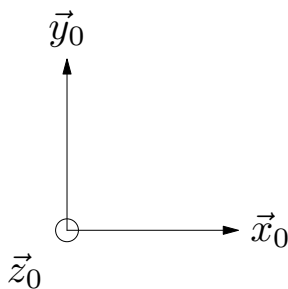
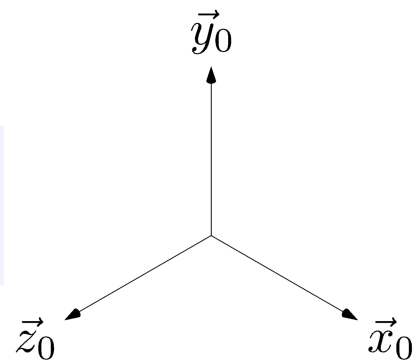
Donc maintenant on va rajouter des axes, du paramétrage angulaire et le nom des points.

### 2.6.1 Bases

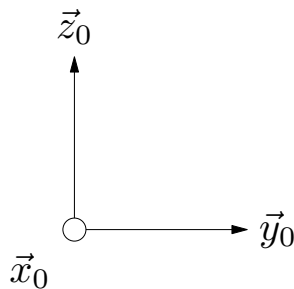
**BIB** `showBasis(basis b, triple point, triple coeff=(1,1,1), pen style=black+0.25) ;`

- `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`)

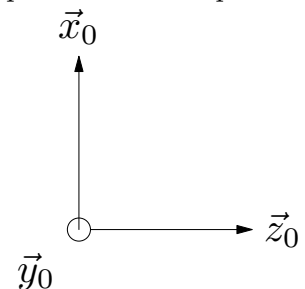
```
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
```



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



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



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

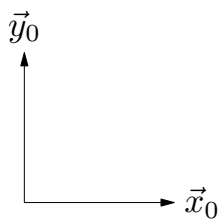
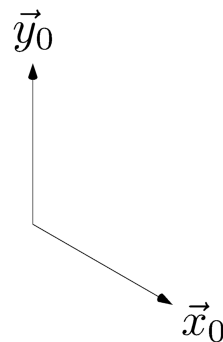
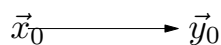
— <code>\bielle_manivelle</code>	— <code>\I3D_anim</code>	— <code>\SDP</code>
— <code>\concasseur</code>	— <code>\maxpid</code>	— <code>\SDP_anim</code>
— <code>\DAE</code>	— <code>\pilote5000</code>	— <code>\sinusmatic</code>
— <code>\DAE_anim</code>	— <code>\pilote5000_anim</code>	— <code>\trainEpicycloidaux</code>
— <code>\direction_camion</code>	— <code>\pinceCoupeCable</code>	
— <code>\I3D</code>	— <code>\pompePistonsAxiaux</code>	

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

-  \bielle\_manivelle
-  \maxpid
-  \pompePistonsAxiaux
-  \direction\_camion
-  \pilote5000
-  \sinusmatic

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

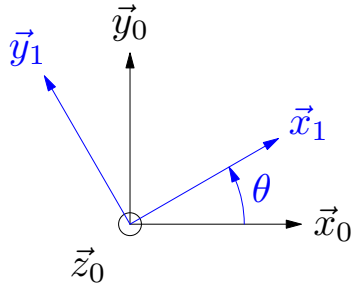
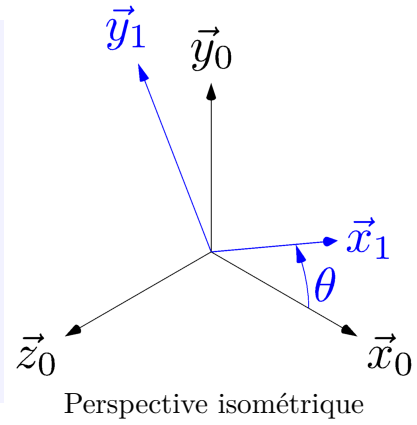
```
showParameter(triple point, triple axis1, triple axis2, string name, real
coeff=1, pen style=black+0.25) ;
```

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

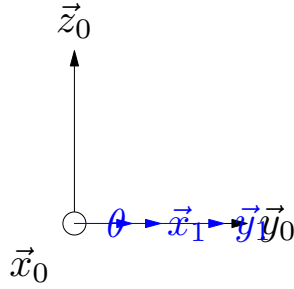
```

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 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 showParameter(0, b0.x, b1.x, "$\theta$",
12   coeff=1, style=0.25+blue) ;

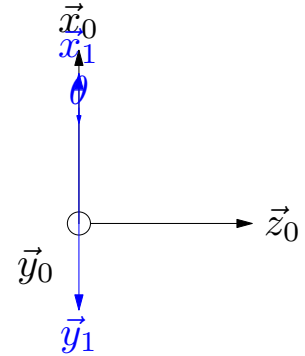
```



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



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



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

— `\bielle_manivelle`

— `\pilote5000`

— `\sinusmatic`

— `\maxpid`

— `\pompePistonsAxiaux`

## 2.6.2 Texte

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

**BIB** `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

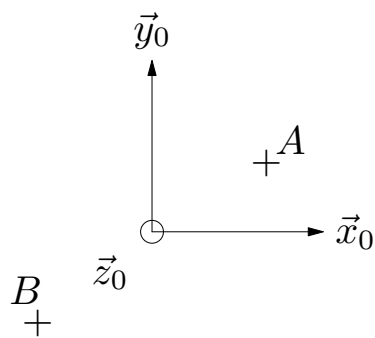
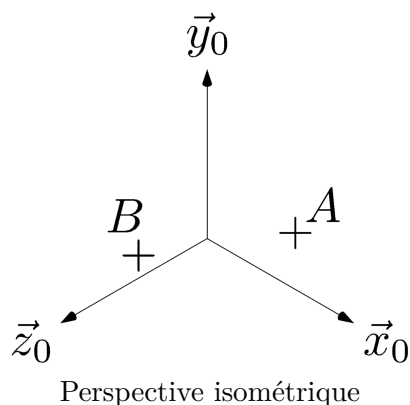
Le dernier paramètre est de type `pair` (coordonnées en 2d). C'est en fait la position sur la projection 2d de la vue 3d du nom du point. Par exemple, `NE` correspond à  $(1, 1)$  soit en haut à droite du point à une distance de  $\sqrt{2}$ . Il y a une valeur par défaut mais qui ne marche globalement 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.



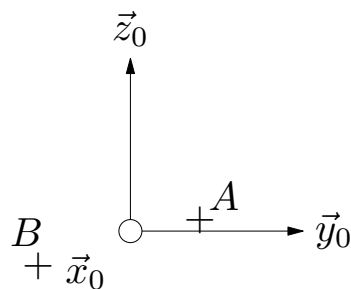
```

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)) ;

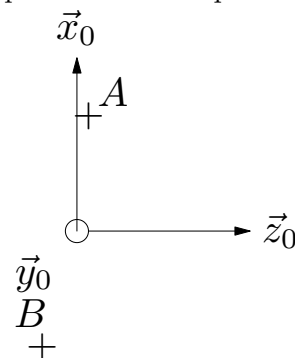
```



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



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



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

—  \bielle_manivelle	—  \maxpid	—  \SDP
—  \concasseur	—  \pilote5000	—  \SDP_anim
—  \DAE	—  \pinceCoupeCable	—  \sinusmatic
—  \direction_camion	—  \pompePistonsAxiaux	—  \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) :

BIB

```

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

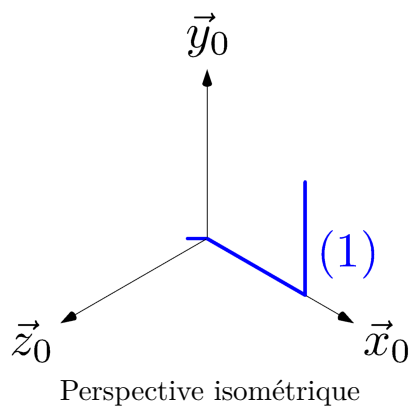
```

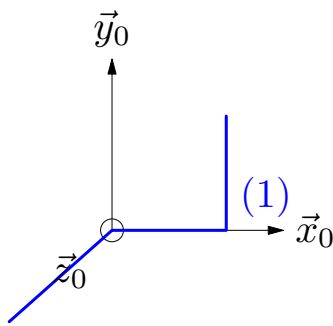
- |   |   |
|---|---|
| — <b>string label</b> : nom de la classe  | — <b>pen p=currentpen</b> : mise en forme du text<br>(valeur par défaut le style par défaut<br>d'asymptote) |
| — <b>triple point</b> : point origine   |   |
| — <b>pair pos=1.5*NE</b> : position relative du texte<br>par rapport au point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

```

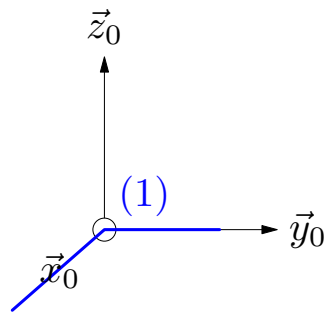
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 link((-0.9,-0.8,-0.7) -- 0 -- 0+b0.x --
  0+b0.x+b0.y, blue) ;
7 nameClasse1point("1", 0+b0.x, pos=1.5*NE,
  p=blue) ;

```

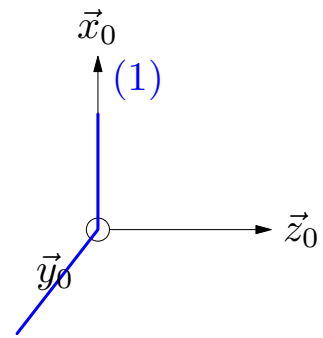




Plan  $(0, \vec{x}, \vec{y})$



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



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

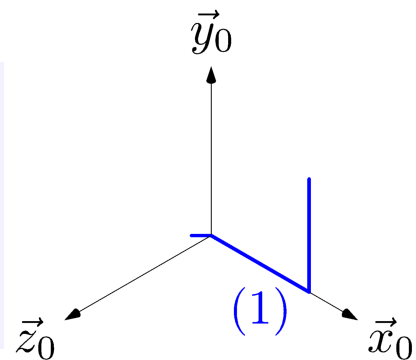
— .\bielle\_manivelle      — .\pinceCoupeCable      — .\trainEpicycloidaux  
— .\pilote5000      — .\pompePistonsAxiaux

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

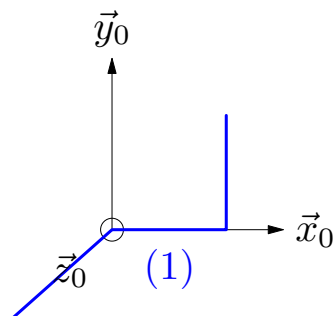
```
nameClasse2points(string label, triple point1, triple point2, pair
pos=1.5*NE, pen p=currentpen) ;
```

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

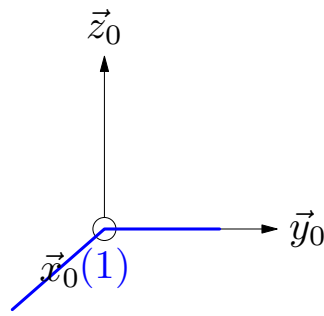
```
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 link((-0.9,-0.8,-0.7) -- 0 -- 0+b0.x --
7 0+b0.x+b0.y, blue) ;
7 nameClasse2points("1", 0, 0+b0.x, pos=1.5*S,
p=blue) ;
```



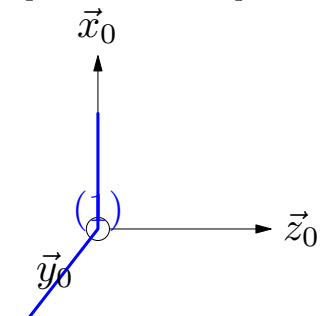
Perspective isométrique



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



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



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

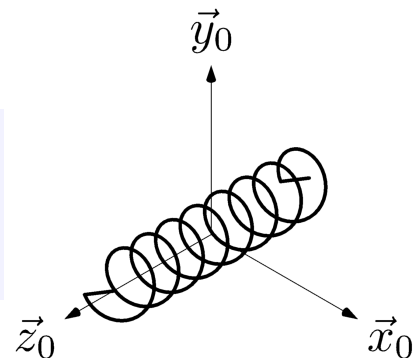
— .\bielle\_manivelle      — .\pinceCoupeCable      — .\trainEpicycloidaux  
— .\pilote5000      — .\pompePistonsAxiaux

## 2.7 Éléments technologiques

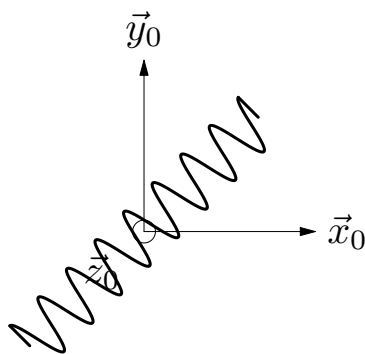
### 2.7.1 Ressort de traction/compression

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

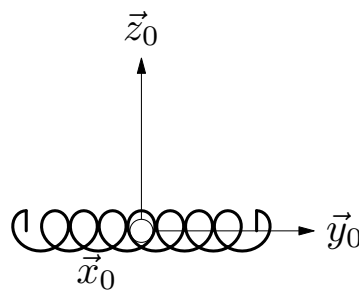
```
1 triple eye = (1,1,1) ;
2 triple up = (0,1,0) ;
3 currentprojection = orthographic(eye, up, 0) ;
4 currentlight = nolight ;
5 showBasis(b0, 0, coeff=1.5*(1,1,1)) ;
6 addSpring((-1,-1,0), (1,1,0)) ;
```



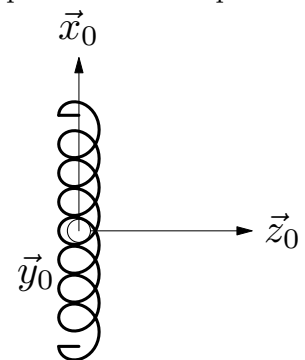
Perspective isométrique



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



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



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

— .\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) ;
```

— .\bielle\_manivelle\_anim

— .\maxpid\_anim

— .\sinusmatic\_anim

— .\DAE\_anim

— .\pilote5000






— .\direction\_camion

— .\pilote5000\_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) ;
```

—  .\bielle_manivelle_anim	—  .\I3D_anim	—  .\sinusmatic_anim
—  .\DAE_anim	—  .\maxpid_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 L<sup>A</sup>T<sub>E</sub>X), une fusion de l'ensemble des pdf générés est réalisée.

—  .\direction_camion	—  .\pilote5000_anim
—  .\I3D_anim	—  .\SDP_anim

# 4 Exemples détaillés

## 4.1 Le système bielle-manivelle

Partons du fichier du système bielle-manivelle en perspective isométrique pour s'appropriier 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 showParameter(0, b0.x, b1.x, "$\theta_{1}$") ;
6 showParameter(0, b0.y, b1.y, "$\theta_{1}$") ;
7 int[] tabIndices = {0} ;
8 showAxis(b2, tabIndices, B, 2, style=black+0.25) ;
9 showParameter(B, b0.x, b2.x, "$\theta_{2}$") ;
10 draw(L=Label("$x$", position=Relative(0.5), align=SW), shift(-0.5*b0.y)*(0
    -- B) , CEC3+0.25, Arrow3) ;
11 draw(B--shift(-0.5*b0.y)*B, CEC3+0.25) ;

```

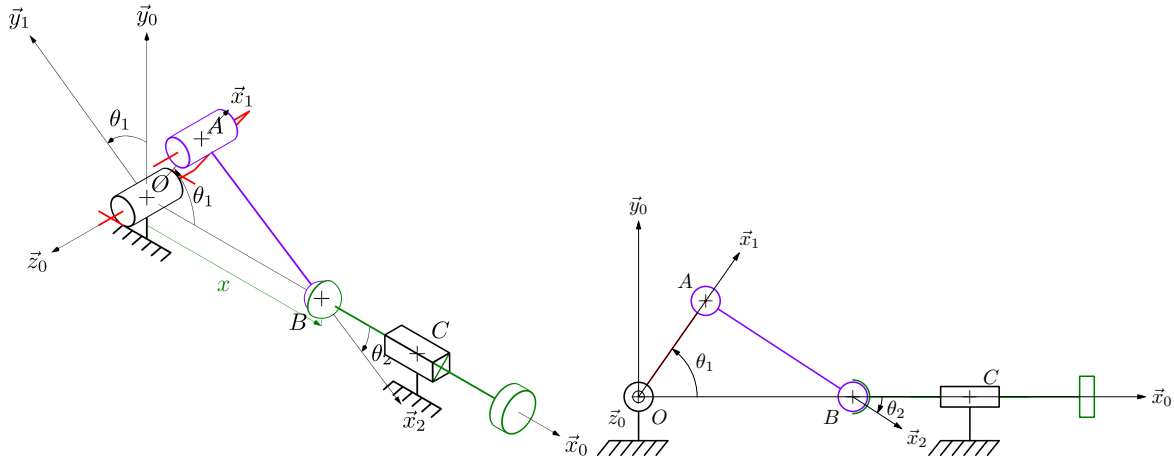
et le noms des points en conclusion :

```

1 // Noms des points
2 namePoint(O,"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  $O(n^2)$ .