V.
Scène 3D

1. Transformations Géométriques de base
2. Espace projectif et coordonnées homogènes
3. Transf. Géométrique sous OpenGL
4. Point de vue
5. Description hiérarchique des objets

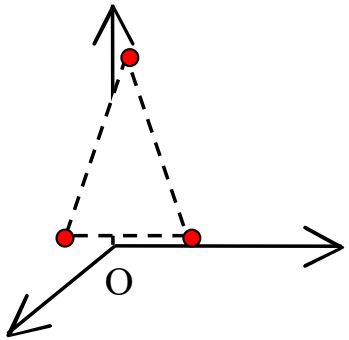
Laboratoire des Sciences de l'Information et des Systèmes
UMR CNRS 6168

Rappel :

4 étapes principales pour le tracé d'une primitive

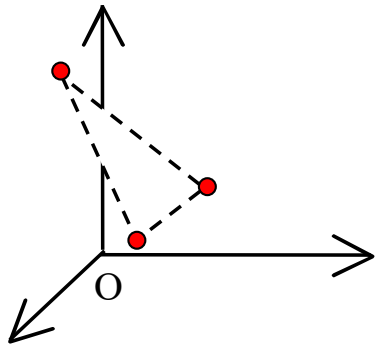
1 : Primitive : point, segment, triangle

décrite sous forme de **sommets**



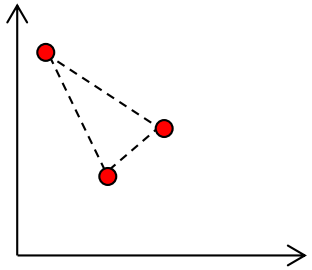
Espace 3D continu non borné

2: Transformation géométrique 3D



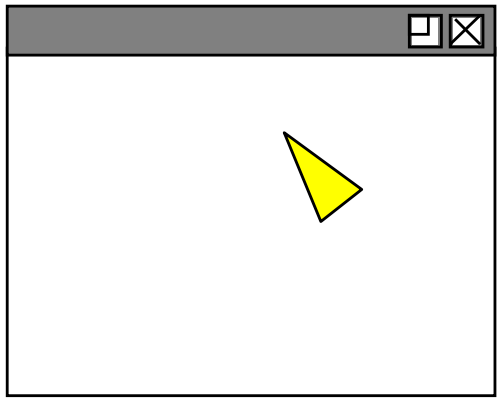
Espace 3D continu non borné

3 : Projection 2D



Espace 2D continu non borné

4 : Pixelisation-remplissage



Espace 2D discret borné

Généralités

Pour construire des objets et les positionner dans la scène :

➔ transformations géométriques 3D

Un objet complexe : constitué de plusieurs pièces

➔ Utilisation de modèles hiérarchiques

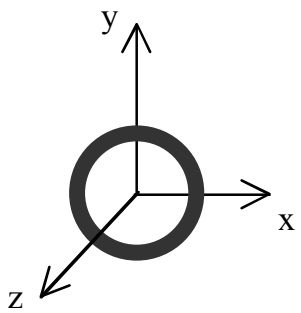
- définir un référentiel pour l'objet à tracer
- positionner les pièces de cet objet par rapport à ce référentiel
- cet objet pourra être lui même une pièce d'un objet plus complexe

  => ses primitives subiront une transformation géométrique avant d'être projetées

Exemple :

Tracer une roue de vélo :

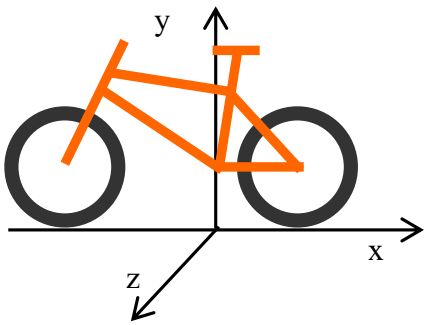
1. choisir un référentiel roue : par exemple le centre
2. tracer la jante dans le plan X-Y et centrée sur Z
3. tracer le pneu dans le plan X-Y et centré sur Z



Tracer un vélo :

1. Choisir un référentiel vélo
2. Tracer un cadre dans ce référentiel
3. Déterminer la position de la roue avant dans ce référentiel
4. Tracer roue
5. Déterminer la position de la roue arrière dans ce référentiel
6. Tracer roue

Tracer un vélo dans une scène : référentiel absolu



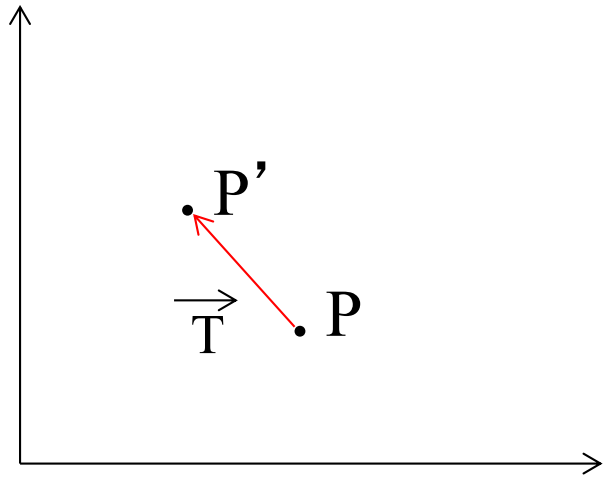
1. Transformations géométriques de base

- la **translation**
- la **rotation** (en degré) autour d'un axe porté par un vecteur,
- l'**homothétie** suivant les trois axes X, Y et Z.

a) Translation 2D

Soit un vecteur T(dx, dy) et un point P(x, y),

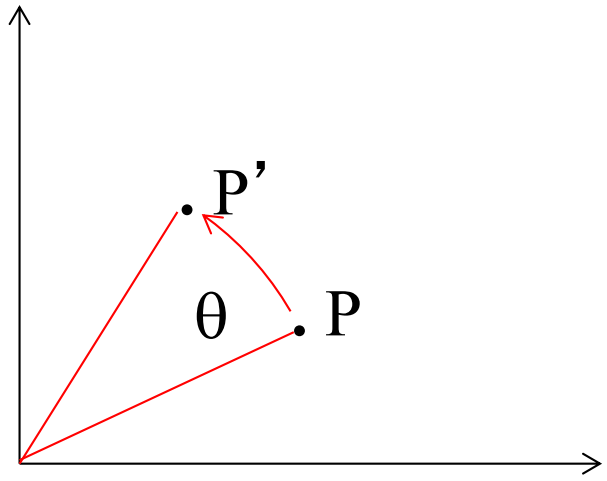
$$\begin{vmatrix} x' \\ y' \end{vmatrix} = \begin{vmatrix} x \\ y \end{vmatrix} + \begin{vmatrix} dx \\ dy \end{vmatrix}$$



b) Rotation 2D autours de l'origine

Une rotation d'angle θ <u>autour de l'origine</u> est définie par :

$$\begin{vmatrix} x' \\ y' \end{vmatrix} = \begin{vmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{vmatrix} * \begin{vmatrix} x \\ y \end{vmatrix}$$

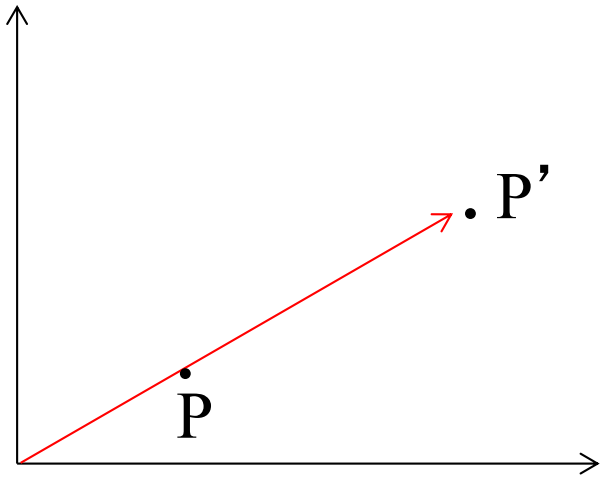


c) Homothétie 2D par rapport à l'origine

Une homothétie par rapport à l'origine est définie par :

$$\begin{vmatrix} x' \\ y' \end{vmatrix} = \begin{vmatrix} h & 0 \\ 0 & k \end{vmatrix} * \begin{vmatrix} x \\ y \end{vmatrix}$$

l'homothétie est dite **uniforme** si h = k, **différentielle** sinon



Homothétie uniforme

d) Composition des transformations géométriques

Rappels :

- le produit matriciel est **associatif** $(F \cdot G) \cdot H = F \cdot (G \cdot H)$
- le produit matriciel **n'est pas commutatif**, $F \cdot G \neq G \cdot F$

- on peut précalculer la composition des rotations et des homothéties

  $f \circ g \circ h$ $\quad\quad$ $\mathbf{E} = F \cdot G \cdot H$

  $f \circ g \circ h\,(P) = \mathbf{E} \cdot P$

- on peut précalculer la composition de plusieurs translations

  $t_1 \circ t_2$ $\quad\quad$ $\mathbf{E} = T_1 + T_2$

  $t_1 \circ t_2(P) = \mathbf{E} + P$

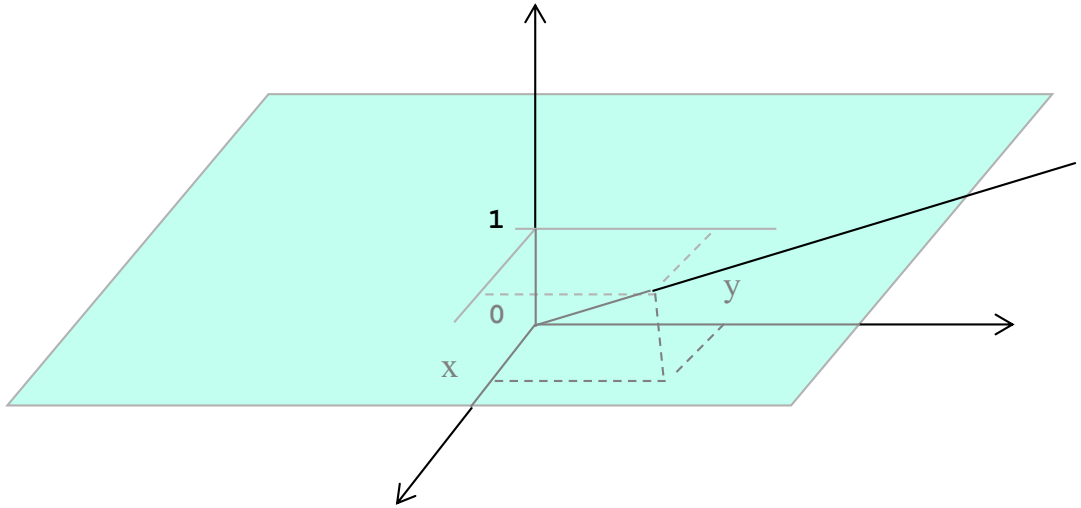
- on ne peut pas précalculer la composition d'une translation avec une rotation ou une homothétie

  :o(

2. Coordonnées homogènes en 2D

Espace projectif : ajout d'une 3ème coordonnée

- tout point (x, y, w) est un représentant de (x/w, y/w) dans l'espace initial
- $(x_1, y_1, \mathbf{1})$ est un représentant privilégié : **coordonnées homogènes**



- un point du plan initial est associé à une droite dans l'espace projectif
- w = 0 : point image à l'infini dans la direction de la droite

Transformations géométrique dans l'espace projectif :

$$\begin{vmatrix} x' \\ y' \\ 1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{vmatrix} . \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$

Translation

$$\begin{vmatrix} x' \\ y' \\ 1 \end{vmatrix} = \begin{vmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix} . \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$

Rotation

$$\begin{vmatrix} x' \\ y' \\ 1 \end{vmatrix} = \begin{vmatrix} h & 0 & 0 \\ 0 & k & 0 \\ 0 & 0 & 1 \end{vmatrix} . \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$

Homothétie

Composition de transformations 2D

Dans l'espace projectif, les 3 transformations géométriques se composent sous la forme de **produit matriciel**.

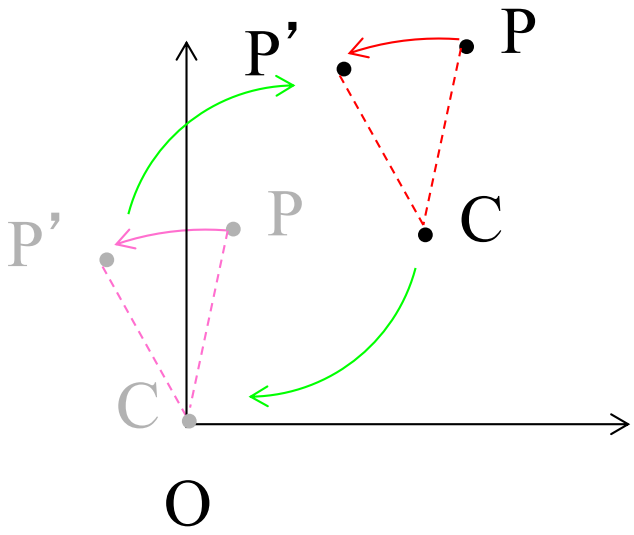
➔ architecture spécifique des processeurs graphiques

Ex : rotation d'un point P autours d'un point C

1. translation CO : $\mathcal{T}_{CO}$
2. rotation de θ autours de O : $\mathcal{R}_{\theta}$
3. translation OC : $\mathcal{T}_{OC}$

$$P' = \mathcal{T}_{OC} \circ \mathcal{R}_{\theta} \circ \mathcal{T}_{CO}(P)$$

$$P' = (T_{OC} \cdot R_{\theta} \cdot T_{CO}) \cdot P$$



Calcul

$$\begin{vmatrix} 1 & 0 & cx \\ 0 & 1 & cy \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} 1 & 0 & -cx \\ 0 & 1 & -cy \\ 0 & 0 & 1 \end{vmatrix}$$

$$\begin{vmatrix} 1 & 0 & cx \\ 0 & 1 & cy \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} \cos\theta & -\sin\theta & -\cos\theta.cx+\sin\theta.cy \\ \sin\theta & \cos\theta & -\sin\theta.cx-\cos\theta.cy \\ 0 & 0 & 1 \end{vmatrix}$$

$$\begin{vmatrix} \cos\theta & -\sin\theta & (1-\cos\theta).cx+\sin\theta.cy \\ \sin\theta & \cos\theta & -\sin\theta.cx+(1-\cos\theta).cy \\ 0 & 0 & 1 \end{vmatrix}$$

On montre que la composition de transformations géométriques 2D est toujours de la forme :

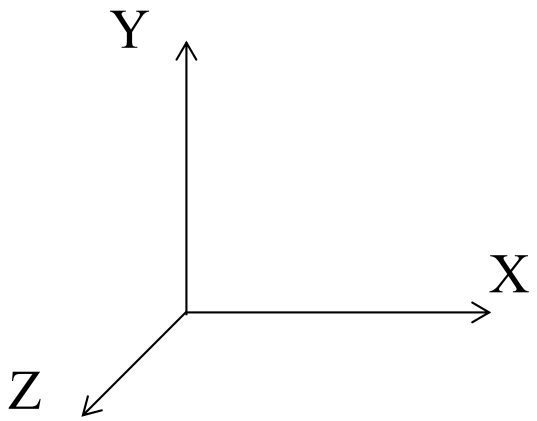
$$\begin{vmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{vmatrix}$$

Extension au 3D

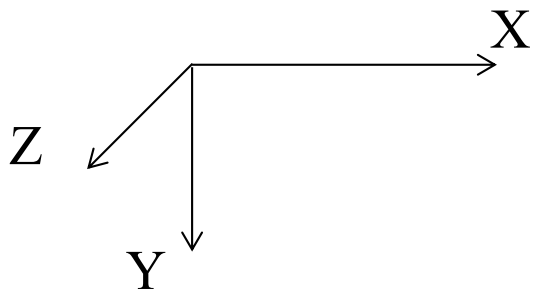
Rappel : **repère orthonormé direct**

- axes de rotation
- direction d'une rotation positive

X	Y	Z
Y à Z	Z à X	X à Y



Repère direct



Repère indirect

Matrices des transformations dans l'espace projectif 3D

Translations et homothéties :

$$\begin{vmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{vmatrix} \qquad \begin{vmatrix} hx & 0 & 0 & 0 \\ 0 & hy & 0 & 0 \\ 0 & 0 & hz & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Rotations autour de Oz, Ox, et Oy

$$\begin{vmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \quad \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \quad \begin{vmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

3. Transformations Géométriques sous OpenGL

La **matrice de modélisation** (3D -> 3D) : **`GL_MODELVIEW`**

- matrice de transformation courante pour positionner les objets
- appliquée à toutes les primitives avant d'être tracées

  `GL_MODELVIEW . P`

- construite par produit matriciel avec les transformations de base

Initialisation avec l'identité :

```
glLoadIdentity() ;
```

$$\begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Composition :

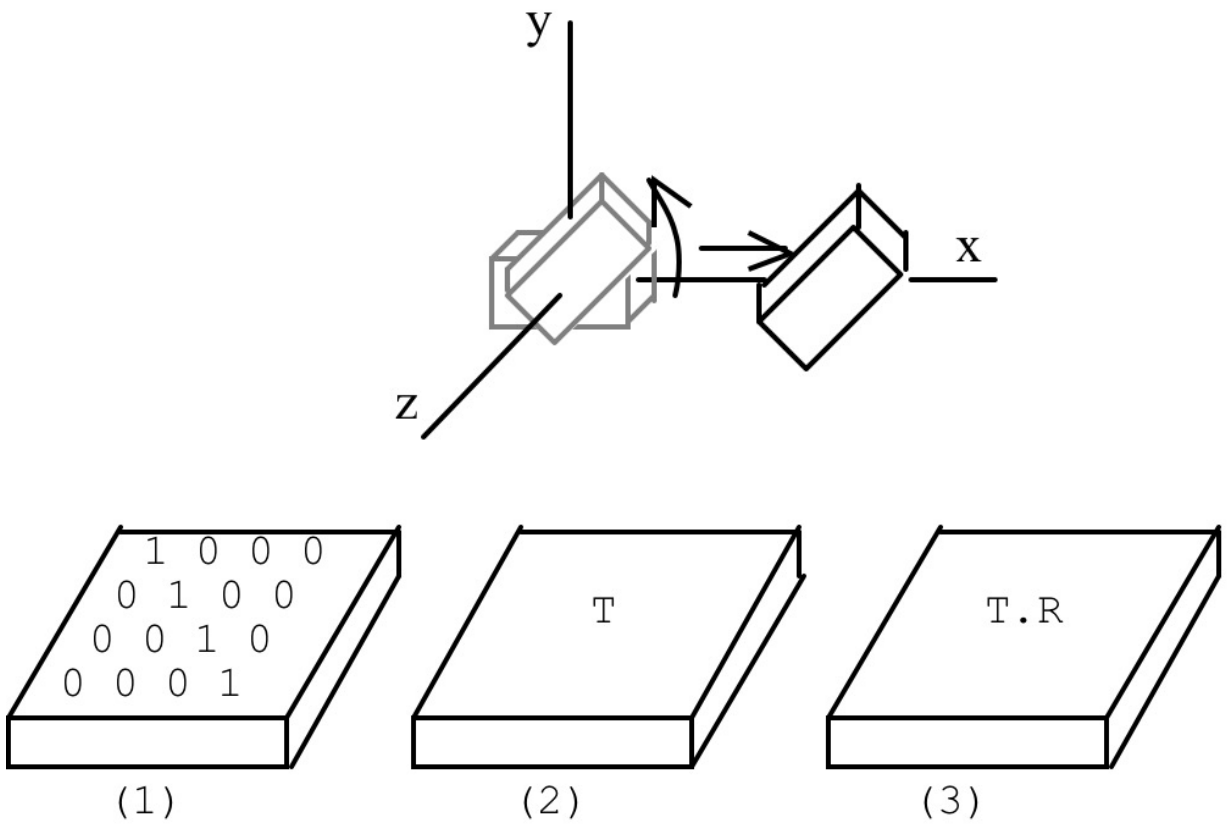
`GL_MODELVIEW <- GL_MODELVIEW .` *transformation de base*

- **`glTranslatef`**`(`*dx, dy, dz*`)`
- **`glScalef`**`(`k_x`,` k_y`,` k_z`)`
- **`glRotatef`**`(`*angle, x, y, z*`)`

Remarques :

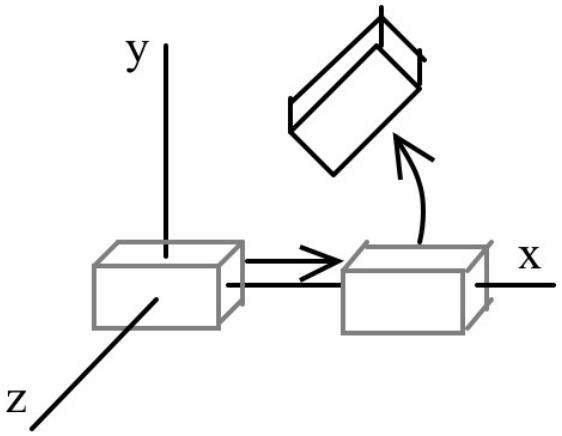
- Tous les paramètres sont des `GLfloat` (`float` autorisés)
- L'angle est exprimé en degré
- Attention à l'utilisation de 0 dans **`glScalef`** !!!
- La dernière transformation de base énoncée est la première appliquée

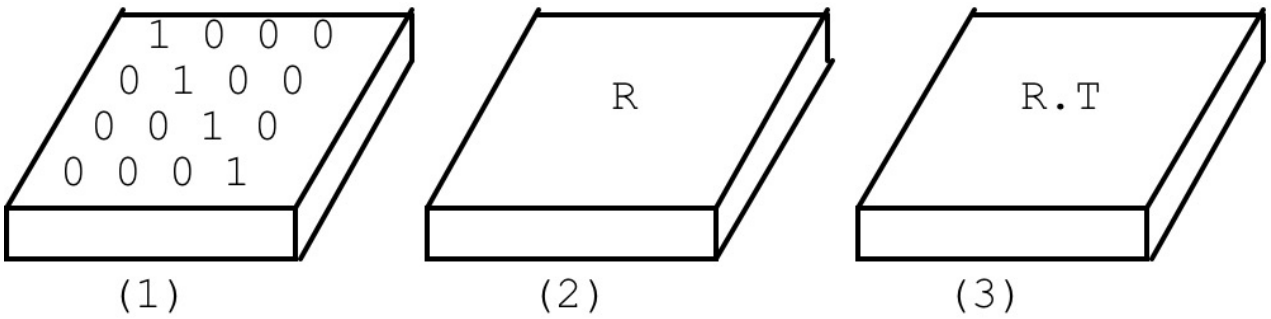
```
/* matrice de transformation A */
glLoadIdentity() ;                    (1)
glTranslatef(5, 0, 0);                (2)
glRotatef(45, 0, 0, 1);               (3)
/* objet qui subira la transformation*/
dessineBoite();
```



Evolution de la matrice de modélisation (A)

```
/* matrice de transformation B */
glLoadIdentity() ;                    (1)
glRotatef(45, 0, 0, 1);               (2)
glTranslatef(5, 0, 0);                (3)
/* objet qui subira la transformation*/
dessineBoite();
```





Evolution de la matrice de modélisation (B)

4. Point de vue

Le point de vue se situe par défaut :

- à l'origine,
- visant vers les Z négatifs.
- le haut orienté vers les Y positifs

Pour visualiser une scène, on peut :

- soit la déplacer pour la mettre dans le champ de vision,
- soit déplacer le point de vue avec **`gluLookAt`**

Ces deux opérations sont strictement équivalentes.

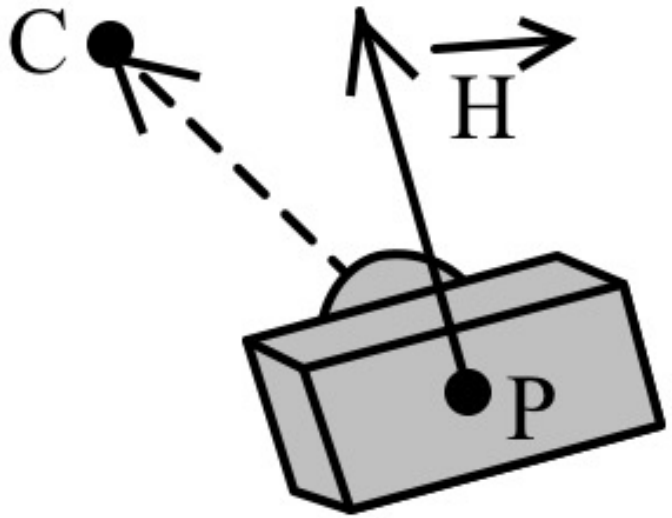
Déplacer le point de vue :

```
gluLookAt(GLdouble Px, GLdouble Py, GLdouble Pz,
          GLdouble Cx, GLdouble Cy, GLdouble Cz,
          GLdouble Hx, GLdouble Hy, GLdouble Hz)
```

P : position

C : point visé

H : verticale appareil



Visualiser une scène simple contenant un objet :

```
void afficheMaScene()
{ glClear(GL_COLOR_BUFFER_BIT) ;

  glLoadIdentity() ;
  gluLookAt(position, point visé, verticale) ;
  tracer un objet
  glutSwapBuffer() ;
}
```

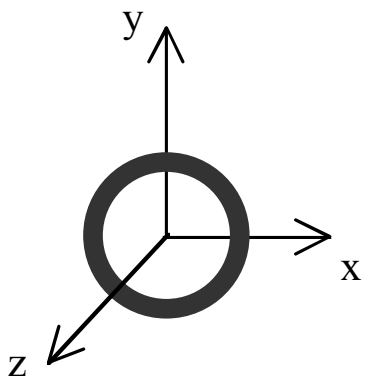
5. Gestion des transformations

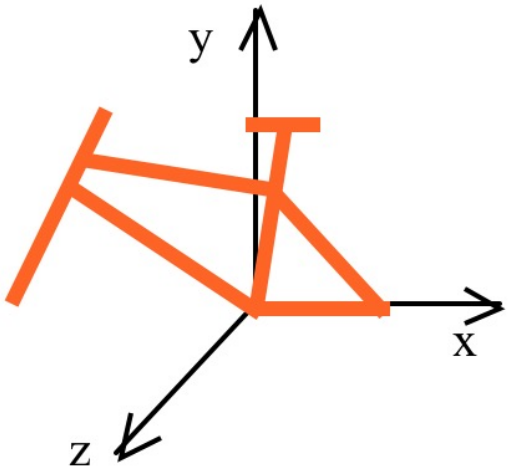
OpenGL utilise une **pile de matrices de modélisation**

➔ description hiérarchique d'un objet complexe.

Exemple : construction d'un vélo

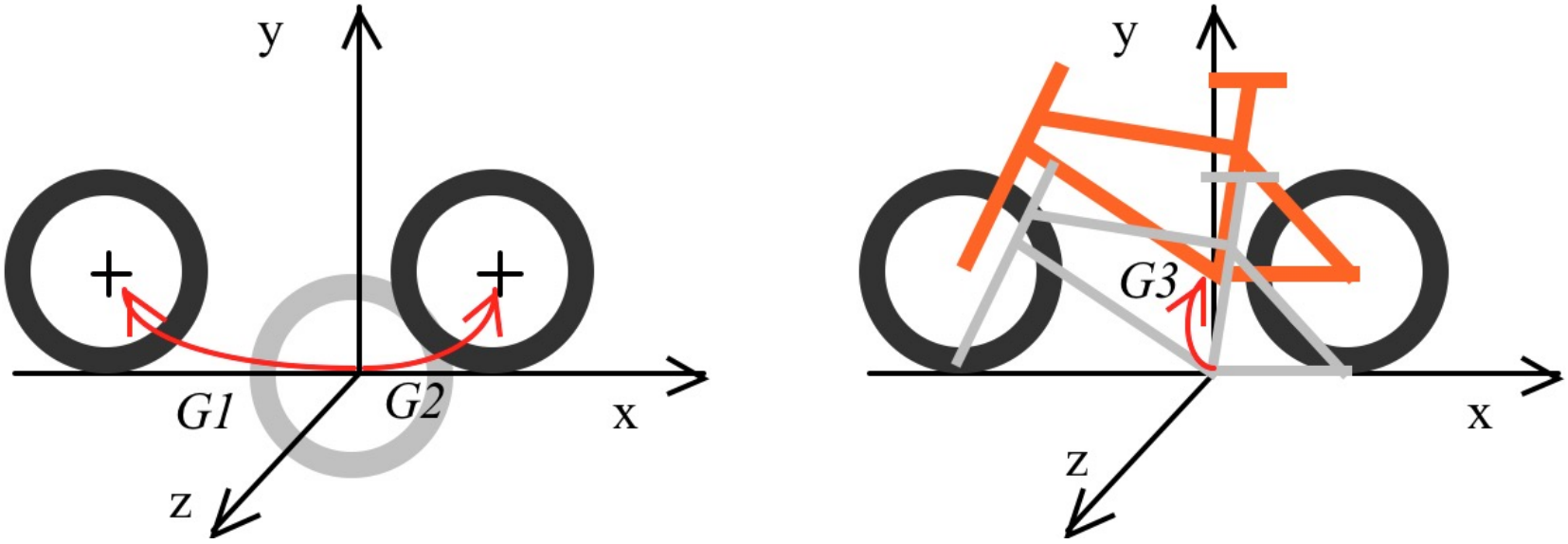
- Soient les fonctions `dessineRoue()` et `dessineCadre()`
- Choix arbitraire :
  - référentiel roue = centre de la roue
  - référentiel cadre = axe du pédalier



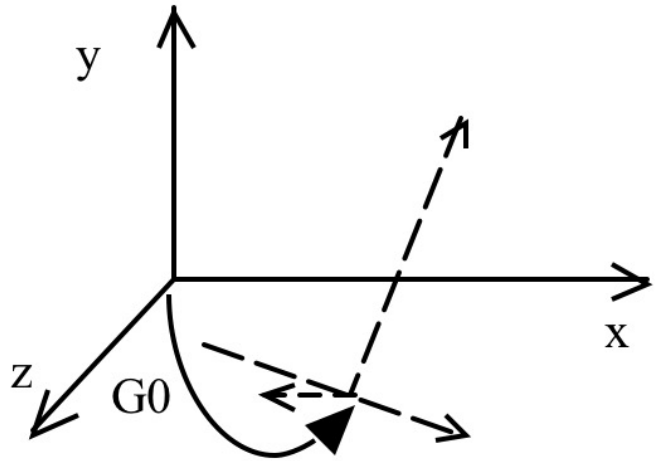


Assemblage d'un vélo : deux roues et un cadre

- choisir un référentiel vélo
- 3 transformations / ce référentiel : G1, G2, G3



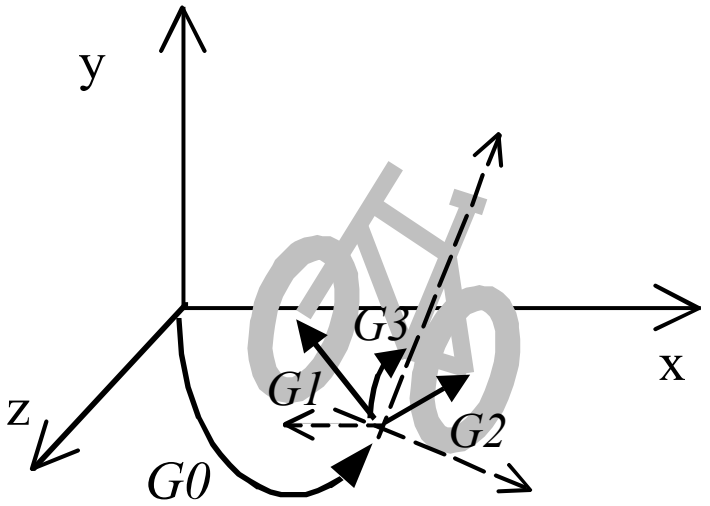
Positionner le vélo dans la scène : transformation G0



Transformations à appliquer :

- G1’ = G0.G1 à la roue avant
- G2’ = G0.G2 à la roue arrière
- G3’ = G0.G3 au cadre

➔ *il faut conserver G0*



`GL_MODELVIEW` est en fait une pile de transformations géométriques qui permet de conserver des transformations intermédiaires

2 opérations de base pour gérer la pile **`GL_MODELVIEW`** :

`glPushMatrix()` : `duplication du sommet de pile`

`glPopMatrix()` : `dépiler`

Après **`glPushMatrix`** :

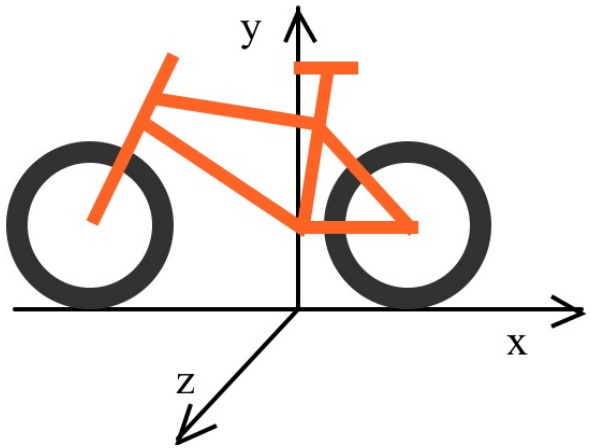
- le sommet de pile est une copie de la transformation « mère »
- sauf si on exécute ensuite `glLoadIdentity()`
- les transformations géométriques se composent avec le sommet
- la transformation « mère » est préservée

Après **`glPopMatrix`** :

- on oublie la transformation courante
- on retrouve la transformation « mère »

Algorithme de tracé du vélo:

```
/*on suppose que GL_MODELVIEW contient G0*/
glPushMatrix() ;      /* roue avant */
  GL_MODELVIEW <- GL_MODELVIEW.G1;
  dessineRoue() ;
glPopMatrix() ;    /* on retrouve G0 */
glPushMatrix() ;      /* roue arrière */
  GL_MODELVIEW <- GL_MODELVIEW.G2;
  dessineRoue() ;
glPopMatrix() ;    /* on retrouve G0 */
glPushMatrix() ;      /* cadre */
  GL_MODELVIEW <- GL_MODELVIEW.G3;
  dessineCadre() ;
glPopMatrix() ;    /* GL_MODELVIEW contient G0 */
```



Affichage obtenu si G0 = I

```
void dessineVelo(void)
{ glPushMatrix() ; /* transformation relative */
    glTranslatef(dx1, rayon, 0) ;
    dessineRoue() ;
  glPopMatrix() ;
  glPushMatrix() ;
    glTranslatef(dx2, rayon, 0) ;
    dessineRoue() ;
  glPopMatrix() ;
  glPushMatrix() ;
    glTranslatef(0, rayon, 0) ;
    dessineCadre() ;
  glPopMatrix() ;
}
```

```
void afficheMaScene(void)
{ glClear(GL_COLOR_BUFFER_BIT) ;
  glLoadIdentity() ;
  gluLookAt(position, point visé, verticale) ;
  glPushMatrix() ;  /* vélo n° 1 */
    glTranslatef(V1x, V1y, V1z) ;
    glRotatef(cap1, 0, 1, 0) ;
    dessineVelo() ;
  glPopMatrix() ;
  glPushMatrix() ;  /* vélo n° 2 */
    glTranslatef(V2x, V2y, V2z) ;
    glRotatef(cap2, 0, 1, 0) ;
    dessineVelo() ;
  glPopMatrix() ;
  …
  glutSwapBuffer() ;
}
```

Schéma général pour tracer un objet par assemblage

- la transformation globale est définie avant d'appeler la fct de tracé
- elle sera stockée au sommet de la pile **GL_MODELVIEW**
- les T.G. locales sont encadrées par des `Push` et des `Pop`

```
void dessineObjet()
{ glPushMatrix() ; /* partie 1 */
    suite de transformations (scale, rotate, translate)
    dessinePartie1() ;
  glPopMatrix() ;
  glPushMatrix() ; /* partie 2 */
    suite de transformations
    dessinePartie2() ;
  glPopMatrix() ;
  …
}
```

Résumé

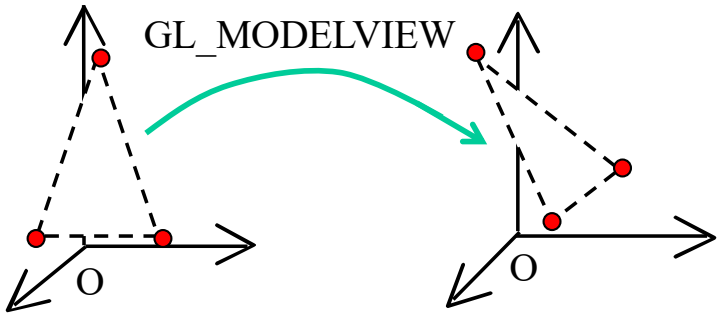
Espace projectif et coordonnées homogènes :

- Ajout d'une 4ème coordonnée **`w`**
- Coordonnées d'un sommet 3D **`(x, y, z, 1)`**
- Coordonnées d'un point à l'infini **`(x, y, z, 0)`**
- Rotations, homothéties et translations se composent sous la forme de produits matriciels

`GL_MODELVIEW`

- Matrice de transformation géométrique 4x4
- appliquée implicitement sur tous les sommets **`S`** au moment du tracé avec **`glVertex`** :

`S'<- GL_MODELVIEW.S`



Résumé

Matrice de modélisation géométrique : `GL_MODELVIEW`

Initialisation : **`glLoadIdentity()`**

Composition des transformations :

- **`GL_MODELVIEW <- GL_MODELVIEW .`** *transformation de base*
- Transformations de base :
  - **`glTranslatef`**(*dx, dy, dz*)
  - **`glScalef`**(k_x, k_y, k_z)
  - **`glRotatef`**(*angle, x, y, z*) *autour du vecteur (x,y,z)*

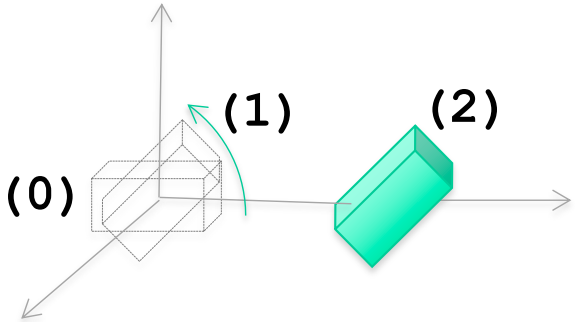
Caméra :

- A l'origine, tournée vers les Z négatifs
- Déplacement avec **`gluLookAt`**(*position, pt visé, verticale*)
  - Simulé en modifiant **`GL_MODELVIEW`**
  - Doit être situé juste après **`glLoadIdentity()`**

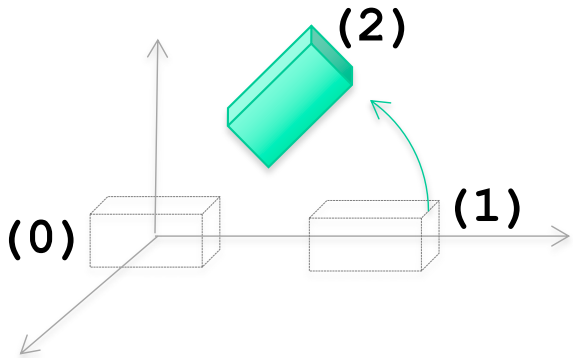
Résumé

Les transformations s'appliquent dans l'ordre inverse de leur écriture :

```
(2) glTranslatef(5, 0, 0);
(1) glRotatef(45, 0, 0, 1);
(0) dessineBoite();
```




```
(2) glRotatef(45, 0, 0, 1);
(1) glTranslatef(5, 0, 0);
(0) dessineBoite();
```



Pile des matrices de modélisation :

- Les transformations de base s'appliquent au sommet de pile
- Dupliquer le sommet : **`glPushMatrix()`**
- Retrouver la transformation ancêtre : **`glPopMatrix()`**

Gestion hiérarchique des objets de la scène

```
void dessineObjet1()
{ glPushMatrix() ;
     suite de transformations et de tracés
  glPopMatrix() ;
}

void dessineObjet2()
{ ... }

void afficheMaScene(void)
{ glClear(GL_COLOR_BUFFER_BIT) ;
  glLoadIdentity() ;        // initialise GL_MODELVIEW avec l'identité
  gluLookAt(position, point visé, verticale) ;  // modifie GL_MODELVIEW
  glPushMatrix() ;
     définir l'orientation et la position de l'objet1 dans la scène
     dessineObjet1() ;
  glPopMatrix() ;
  glPushMatrix() ;
     définir l'orientation et la position de l'objet2 dans la scène
     dessineObjet2() ;
  glPopMatrix() ;
  glutSwapBuffers() ;
}
```

Description d'un objet par assemblage

```
void dessineObjet()
{ glPushMatrix() ; /* partie 1 */
    suite de transformations
    dessinePartie1() ;
  glPopMatrix() ;
  glPushMatrix() ; /* partie 2 */
    suite de transformations
    dessinePartie2() ;
  glPopMatrix() ;
  …
}
```