

Chapitre 1 : Repères et positionnement 3D

Modélisation 3D et Synthèse

Fabrice Aubert
fabrice.aubert@lifl.fr

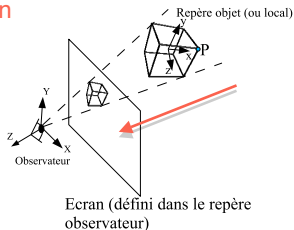


IEEA - Master Info - Parcours IVI

2012-2013

1 Introduction avec OpenGL : librairie pour la programmation 3D

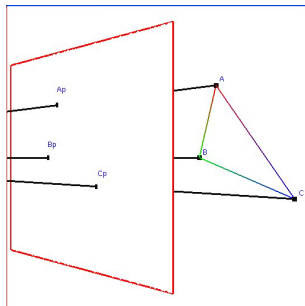
Transformation du repère
par OpenGL sur l'écran
observateur.



- ▶ Affiche uniquement des polygones.
- ▶ Visualisation basée sur un rendu projectif (méthode adoptée par les cartes graphiques).

Phases

- ▶ Le rendu projectif consiste en 2 phases :
 - Une phase géométrique 3D : spécification des coordonnées des sommets dans un repère local (repère objet) et positionnement du repère objet par rapport au repère de la caméra.
 - \Rightarrow calcul des sommets dans le repère caméra
 - \Rightarrow projection sur un plan, définie dans le repère caméra, représentant l'écran (passage des polygones 3D à des polygones 2D).
 - Une phase 2D qui concerne les pixels (phase dite pipeline de rasterization) : remplissage des polygones 2D pixel par pixel (algorithme de balayage).



Exemple de code

- Procédure de tracé d'un carré :

```
void drawSquare() {  
    glBegin(GL_POLYGON); // trace une primitive polygone  
    glVertex3f(1.0,1.0,0.0); // premier sommet  
    glVertex3f(-1.0,1.0,0.0); // second sommet  
    glVertex3f(-1.0,-1.0,0.0); // ...  
    glVertex3f(1.0,-1.0,0.0);  
    glEnd(); // fin de la primitive  
}
```

3f = 3 paramètre -> float

- On déplace (translation, rotation, ...) un repère courant dans lequel seront tracés les sommets (i.e. le repère objet). Initialement le repère courant est l'observateur.

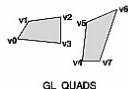
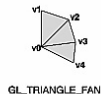
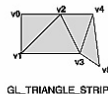
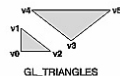
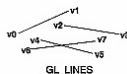
```
void drawScene() {  
    glLoadIdentity(); // initialisation du repère courant = repère caméra  
    glTranslatef(0,0,-5); // déplacer du côté "visible" de l'écran  
    glRotatef(30,0,0,1); // rotation de 30 degrés autour de z  
    glTranslatef(0,2,0); // translation de 2 sur y  
  
    drawSquare(); // tous les sommets rencontrés seront interprétés dans le repère courant  
}
```

Rotation toujours par rapport a un axe

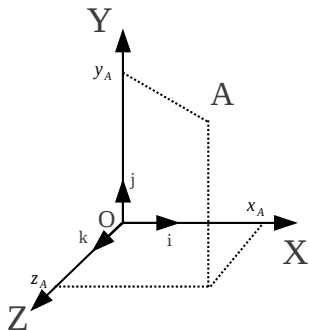
Translate toujours par rapport au repère courant (ici Y a été penché de 30 degrés donc "translate" de biais)

Procédure de tracés

```
glBegin(<primitive>); // <primitive> = GL_POINTS, GL_LINES, etc
glVertex3f(x0,y0,z0); // sommet v0
glVertex3f(x1,y1,z1); // sommet v1
glVertex3f(x2,y2,z2); // sommet v2
...
glEnd();
```



2 Éléments fondamentaux pour la programmation 3D



► Repère noté (O, i, j, k) (origine et vecteurs de base).

► Un point : $A = \overrightarrow{OA} = \begin{pmatrix} A_x \\ A_y \\ A_z \end{pmatrix}$ ou

$$A = (A_x, A_y, A_z).$$

► $A = O + xi + yj + zk.$

► Les repères considérés seront généralement directs :

- Règle de la main **droite** : (Pouce, Index, Majeur) = (X, Y, Z)

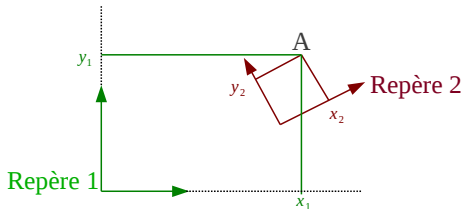
► Un repère est dit orthonormé si :

- Vecteurs (i, j, k) deux à deux orthogonaux.
- (i, j, k) de même normes 1.

Remarques

- ▶ Quand il y aura plusieurs repères : **toujours** préciser le repère en indice :

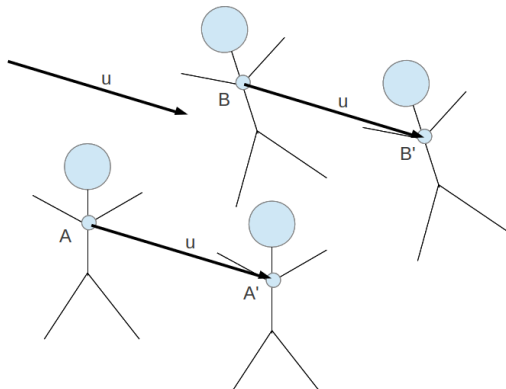
Par exemple, un même point A peut avoir les coordonnées (x_1, y_1, z_1) dans le repère 1 et (x_2, y_2, z_2) dans le repère 2.



⇒ Noter A_1 ou A_2 selon le repère considéré.

Points et directions

- ▶ Différencier les positions (i.e. les points) et les directions (i.e. les vecteurs).
- ▶ Exemple :
 - Le point A est déplacé au point A' par le vecteur u . Le point B est déplacé au point B' par le même vecteur u .
 - Par le calcul : $A' = A + u$ et $B' = B + u$.
 - Remarque sur les notations : pas de "flèche" sur u . Par contre $u = \overrightarrow{AA'} = \overrightarrow{BB'}$. Pour les calculs, on utilisera souvent la relation : $u = A' - A = B' - B$ (ok pour algèbre 3D).



Produit scalaire (dot product)

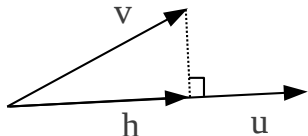
- Apparait pour : équations algébriques (plan), test d'intersection, test de localisation (d'un point par rapport à un plan, ...), éclairement (angle d'incidence de la direction d'éclairement sur un objet), orientation (tests de directions « opposées »), ...

$$\text{Soient } u = \begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix}, v = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} :$$

- Le produit scalaire donne un nombre.
- $u \cdot v = u_x v_x + u_y v_y + u_z v_z$ (calcul valable uniquement dans un repère orthonormé).
- Norme d'un vecteur : $\|u\| = \sqrt{u \cdot u} = \sqrt{u_x u_x + u_y u_y + u_z u_z}$
- Normer un vecteur u signifie le « rendre » de norme 1 (ou unitaire) : $u' = \frac{u}{\|u\|}$
- Autre calcul du produit scalaire : $u \cdot v = \|u\| \|v\| \cos(u, v)$
- Si u et v sont normés, $u \cdot v \in [-1, 1]$

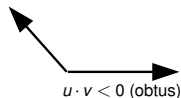
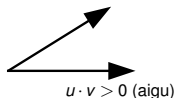
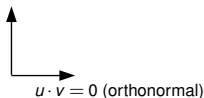
Interprétation du produit scalaire

- ▶ Projection :



- ▶ $u \cdot v = u \cdot h$
- ▶ $\|h\| = \frac{|u \cdot v|}{\|u\|}$
- ▶ Si u est unitaire : $h = (u \cdot v)u$

- ▶ Angle et localisation :

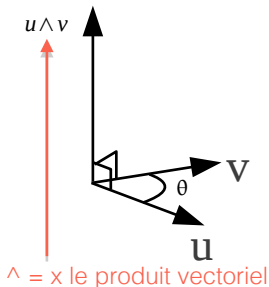


- ▶ Si u et v unitaires alors $u \cdot v = \cos(u, v)$ (donne un nombre dans $[-1, 1]$).
- ▶ $\arccos(u \cdot v)$ donne alors l'angle entre u et v dans $[0, \pi]$.

Produit vectoriel (cross product)

- Apparait pour : détermination de normales (vecteur orthogonal à un plan), construction de vecteurs (repères par exemple), construction d'orientations (définition d'un sens direct-indirect), ...

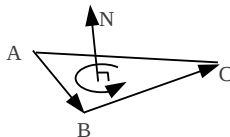
$$\text{Soient } u = \begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix}, v = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} : w = u \times v = \begin{pmatrix} u_y v_z - v_y u_z \\ u_z v_x - v_z u_x \\ u_x v_y - v_x u_y \end{pmatrix}$$



- Si u et v colinéaires (i.e. $u = \lambda v$) alors $u \times v = 0$
- w est orthogonal à u et v . un nb de fois
- w est orienté tel que (u, v, w) est direct (règle de la main droite)
- $\|w\| = \|u\| \|v\| |\sin(\theta)|$

Remarques sur le produit vectoriel

- ▶ Attention le produit vectoriel est non commutatif : $u \times v = -v \times u$ (changement de signe).
- ▶ Un vecteur orthogonal à un polygone est appelé une normale et peut être calculé par produit vectoriel si on connaît les sommets.
- ▶ Exemple d'un triangle (A, B, C) : $n = \overrightarrow{AB} \times \overrightarrow{BC}$ est une normale.
- ▶ La normale d'un triangle (A, B, C) sera dite directe si les sommets A, B et C sont décrits dans le sens trigonométrique par rapport à la normale (en regardant la normale pointée vers soi).
- ▶ sens trigonométrique = sens contraire des aiguilles d'une montre.



Classe Vector3

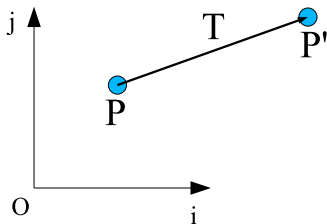
- ▶ Nécessité d'une classe sur les points et les vecteurs.
- ▶ On confond souvent points et vecteurs dans la même classe (à l'utilisateur de savoir ce qu'il manipule).

```
class Vector3 {  
private:  
    double c[3]; // c[0]=x, c[1]=y, c[2]=z  
  
public:  
    /// constructs the vector (x,y,z)  
    Vector3(double x,double y,double z);  
    /// getters  
    double x() const;  
    double y() const;  
    double z() const;  
    /// setters  
    void x(double k);  
    void y(double k);  
    void z(double k);  
  
    /// dot product : returns dot(this,a)  
    double dot(const Vector3 &a) const;  
  
    /// set the cross product a x b to this  
    void cross(const Vector3 &a,const Vector3 &b);  
  
    /// function add : p=p1+p2  
    friend Vector3 operator +(const Vector3 &a,const Vector3 &b);  
    ...  
};
```

3 Transformations

Translation

Translation de vecteur $T = \begin{pmatrix} T_x \\ T_y \\ T_z \end{pmatrix}$.

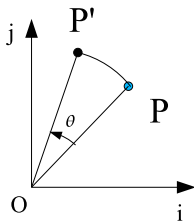


$$\begin{cases} x' &= x + T_x \\ y' &= y + T_y \\ z' &= z + T_z \end{cases}$$

$$P' = P + T$$

Rotation en 2D

Rotation d'angle θ autour de l'origine.



$$\begin{cases} x' &= x \cos(\theta) - y \sin(\theta) \\ y' &= x \sin(\theta) + y \cos(\theta) \end{cases}$$

Comment retrouver ? Soit α l'angle (i, OP) et $r = \|OP'\| = \|OP\|$. Alors

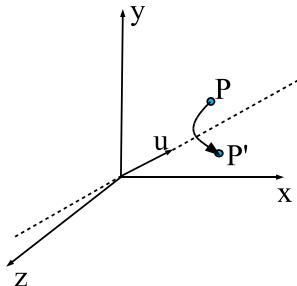
$$\begin{cases} x' &= r \cos(\alpha + \theta) \\ y' &= r \sin(\alpha + \theta) \end{cases}$$

$$\begin{cases} x' &= r \cos(\alpha) \cos(\theta) - r \sin(\alpha) \sin(\theta) \\ y' &= r \cos(\alpha) \sin(\theta) + r \sin(\alpha) \cos(\theta) \end{cases}$$

or $x = r \cos(\alpha)$ et $y = r \sin(\alpha)$

$$P' = RP \text{ avec } R = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

Rotation en 3D



- ▶ La rotation en 3D s'effectue autour **d'un axe** dont on donne un vecteur directeur u (seules rotations considérées : axe passant par l'origine).
- ▶ Le sens de rotation est le sens trigonométrique par rapport à l'axe (« tourne » dans le sens direct quand le vecteur de l'axe pointe vers vous).
- ▶ Exemple : autour de l'axe z (droite de vecteur directeur $(0, 0, 1)$).

$$R_{OZ} = \begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Rotation 3D : autres axes

$$R_{OX} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix}$$

Substituer Y à X et Z à Y

$$R_{OY} = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}$$

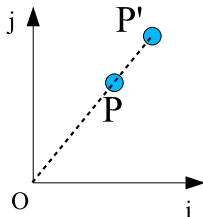
Substituer Z à X et X à Y (attention aux signes)

► La rotation autour d'un axe de vecteur u quelconque est un peu plus laborieuse à exprimer.

Avec $c = \cos(\theta)$, $s = \sin(\theta)$ et $u = (u_x, u_y, u_z)$ **normé**.

$$\begin{pmatrix} u_x^2 + (1 - u_x^2)c & u_x u_y (1 - c) - u_z s & u_x u_z (1 - c) + u_y s \\ u_x u_y (1 - c) + u_z s & u_y^2 + (1 - u_y^2)c & u_y u_z (1 - c) - u_x s \\ u_x u_z (1 - c) - u_y s & u_y u_z (1 - c) + u_x s & u_z^2 + (1 - u_z^2)c \end{pmatrix}$$

Changement d'échelle de rapport k .

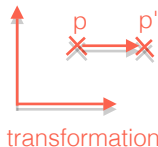


$$\begin{cases} x' = kx \\ y' = ky \\ z' = kz \end{cases} \text{ peut être défini par coordonnées : } \begin{cases} x' = k_x x \\ y' = k_y y \\ z' = k_z z \end{cases}$$

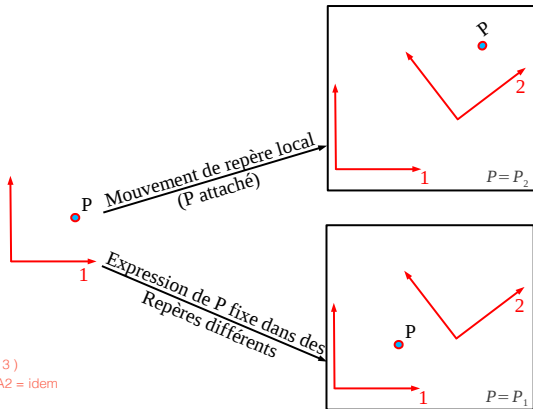
$$P' = SP \text{ avec } S = \begin{pmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & k_z \end{pmatrix}$$

Changements de repères

- ▶ Il s'agit d'une approche généralement adoptée par les librairies graphiques : placer un objet = placer son repère.
- ▶ L'interprétation en **transformation** est différent du raisonnement en **changement de repères**, mais le calcul reste le même :
 - **Transformation** : $P' = \mathcal{T}P$ (2 points dans un même repère) changement de repère correspond mieux
 - **Changement de repères** : $P_1 = \mathcal{T}_{1 \rightarrow 2} P_2$ (un point P dans 2 repères distincts ; $\mathcal{T}_{1 \rightarrow 2}$ indique comment on passe du repère 1 au repère 2).
- ▶ Remarque : $P_1 = \mathcal{T}_{1 \rightarrow 2} P_2$ n'est pour l'instant qu'une notation : elle signifie qu'on applique $\mathcal{T}_{1 \rightarrow 2}$ aux coordonnées de P_2 pour obtenir les coordonnées de P_1 .



Changements de repères : 2 interprétations relatives



Calcule tjrs par rapp au repere 1
 transformation: $A' = T + A = (1,5)(3)$
 chgt repere avec point: $A_1 = T + A_2 = \text{idem}$
 chgt repere sans point:
 $A_1 = T_1 \rightarrow 2A_2$
 $A_1 = T + A_2$
 $A_2 = A_1 - T = T_2 \rightarrow 1A_1$ (avec $T_2 \rightarrow 1 = -T$)

$$\Rightarrow P_1 = T_{1 \rightarrow 2} P_2$$

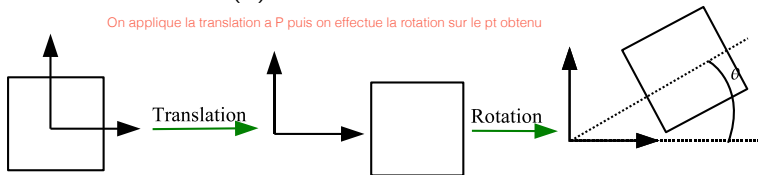
\Rightarrow savoir manipuler et mixer les 3 interprétations (transformations, mouvement du repère local, changement de coordonnées) pour résoudre les problèmes de placements.

Exercice : point $A(0.5, 1.0)$ donné dans un repère 1, donnez le schéma et le calcul des 3 interprétations (calcul de A'_1 ou A_1 ou A_2 selon l'interprétation) pour une translation de vecteur $T = (1, 2)$

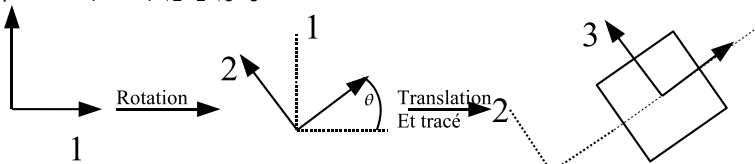
Composition des changements de repères

- ▶ Transformations : la composition de transformations notée $\mathcal{T}_1 \mathcal{T}_2 \dots \mathcal{T}_n$ consiste à appliquer **d'abord** \mathcal{T}_n **puis** \mathcal{T}_{n-1} **puis** ... \mathcal{T}_1 (**lecture de droite à gauche**).
- ▶ Changements de repères : la composition des changements de repères notée $\mathcal{T}_{1 \rightarrow 2} \mathcal{T}_{2 \rightarrow 3} \dots \mathcal{T}_{n-1 \rightarrow n}$ consiste à appliquer **d'abord** $\mathcal{T}_{1 \rightarrow 2}$ **puis** $\mathcal{T}_{2 \rightarrow 3}$ **puis** ... $\mathcal{T}_{n-1 \rightarrow n}$ (**lecture de gauche à droite**).
- ▶ Exemple : RT
 - Transformations : $P' = RT(P)$

On applique la translation à P puis on effectue la rotation sur le pt obtenu



- Repères : $P_1 = R_{1 \rightarrow 2} T_{2 \rightarrow 3} P_3$



- ▶ Avec une librairie graphique on "part" d'un **repère initial** :
 - C'est le repère des points tracés lorsqu'on n'applique aucun changement de repère (par exemple coin bas/gauche de la fenêtre graphique pour une librairie 2D).
- ▶ On conçoit les objets (i.e. les coordonnées de ses points) dans un repère qui permet de le définir le plus aisément possible. Ce repère est dit **repère local** ou **repère objet**.

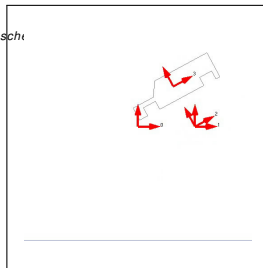


- ▶ Pour placer l'objet où on veut, on déplace alors le repère objet **depuis** le repère initial, c'est à dire qu'on indique $T_{initial \rightarrow objet}$
- ▶ Lorsque l'objet est tracé, la librairie graphique doit calculer les points dans le repère initial :
 - Chaque point de l'objet P_{objet} subit $P_{initial} = T_{initial \rightarrow objet} P_{objet}$
 - La librairie graphique peut alors tracer les points $P_{initial}$.

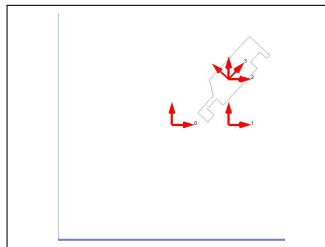
- ▶ On peut décomposer $\mathcal{T}_{initial \rightarrow objet}$ en autant d'étapes que souhaitées. Par exemple :
$$\mathcal{T}_{initial \rightarrow objet} = \mathcal{T}_{initial \rightarrow 1} \mathcal{T}_{1 \rightarrow 2} \mathcal{T}_{2 \rightarrow objet}$$
- ▶ Les bibliothèques graphiques gèrent en général une seule transformation de repère qui traduit le passage du repère initial à un repère courant (i.e. $\mathcal{T}_{initial \rightarrow courant}$).
- ▶ La bibliothèque offre alors les instructions pour modifier à loisir le repère courant en **composant** depuis le repère courant : $\mathcal{T}_{initial \rightarrow courant_{new}} \leftarrow \mathcal{T}_{initial \rightarrow courant_{old}} \mathcal{T}_{courant_{old} \rightarrow courant_{new}}$.
- ▶ Par exemple en OpenGL (2.1) : l'instruction `glTranslatef(5, 0, 5);` modifie le repère courant en le déplaçant du vecteur $T = (5, 0, 5)$.
- ▶ Ces instructions ne **tracent rien** ! Elles modifient le repère courant.

Exemples OpenGL

```
void drawScene(double angle) {  
    // lors de l'appel on se trouve sur un repère courant (noté 0 sur le sché  
    // pour éviter tout effet de bord, on mémorise le repère courant.  
    glPushMatrix();  
  
    glTranslatef(5,0,0);    // passage à 1  
    glRotatef(angle,0,0,1); // passage à 2  
    glTranslatef(0,4,0);    // passage à 3  
  
    drawCar();  
  
    // restituer repère courant (après le pop, le repère courant est à  
    // nouveau le repère 0)  
    glPopMatrix();  
}
```



```
void drawScene(double angle) {  
    // mémoriser le repère courant à l'appel  
    glPushMatrix();  
  
    glTranslatef(5,0,0);    // passage à 1  
    glTranslatef(0,4,0);    // passage à 2  
    glRotatef(angle,0,0,1); // passage à 3  
  
    drawCar();  
  
    // restituer le repère courant à l'appel  
    glPopMatrix();  
}
```



L'animation est obtenue en appelant continuellement `drawScene`. Le paramètre `angle` est incrémenté entre chaque image.

4 Représentation des changement de repères par matrices homogènes

Coordonnées homogènes

- Pour représenter, et calculer, les changements de repères, les bibliothèques de programmation graphique (OpenGL, Java2D/3D, Direct3D, etc) utilisent les matrices homogènes.
- Passer des coordonnées 3D en coordonnées homogènes :

$$P_{3D} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \iff P_H = \begin{pmatrix} xw \\ yw \\ zw \\ w \end{pmatrix} \text{ avec } w \in \mathbb{R}^* (\text{cad } w \neq 0)$$

- Passer des coordonnées homogènes en coordonnées 3D :

$$P_H = \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \iff P_{3D} = \begin{pmatrix} x/w \\ y/w \\ z/w \end{pmatrix}$$

- Remarque : on différencie les points (qui ont une coordonnée $w \neq 0$) et les vecteurs (en posant $w = 0$ et les coordonnées (x, y, z) restent identiques en 3D et en homogènes).
- On peut alors traduire tout changement de repères par une matrice 4×4 , $M_{1 \rightarrow 2}$:
 $P_1 = M_{1 \rightarrow 2} P_2$.

- ▶ Tout changement de repère (translation, rotation, toute composition, ..., projections, ...) peut se traduire par une matrice 4×4 .
- ▶ La composition des changement de repères se traduit par le produit des matrices 4×4 (i.e. $M_{1 \rightarrow 3} = M_{1 \rightarrow 2} M_{2 \rightarrow 3}$).
- ▶ Attention : la somme n'a pas d'interprétation directes en coordonnées homogènes.
Exemple : faire $u = \overrightarrow{AB} = A - B$ n'a aucun sens en coordonnées homogènes.

Exemples

$$T = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{Translation}$$

$$R = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{Rotation}$$

rotation autour de Z

$$S = \begin{pmatrix} k_x & 0 & 0 & 0 \\ 0 & k_y & 0 & 0 \\ 0 & 0 & k_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{Scale}$$

- ▶ Exercice : soient deux translations de vecteurs $T_1(x_1, y_1, z_1)$ et $T_2(x_2, y_2, z_2)$. Appliquer T_1 au point $A(x, y, z)$ par les coordonnées homogènes et retrouver le résultat $A' = A + T_1$. Quelle est la composition de T_1 et T_2 en appliquant le produit ? (retrouver que le résultat est la translation $T_1 + T_2$).
- ▶ Appliquer la composition $T_1 + T_2$ au **vecteur** $u(x, y, z)$.
- ▶ Quelle est la forme générale de la composition TR ? Que constate-t-on quand on applique TR au vecteur u ?

$$A' = \begin{pmatrix} x_1 + x_2 \\ y_1 + y_2 \\ z_1 + z_2 \end{pmatrix} \quad \begin{array}{l} A \rightarrow Ah = (x^*w) \text{ translation: } T_1Ah = (x^*w + tx^*w) \\ (y^*w) \\ (z^*w) \\ (w) \end{array}$$

$$R = \begin{pmatrix} a & b & c & 0 \\ e & f & g & 0 \\ h & i & j & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T_2 = \begin{pmatrix} 1 & 0 & 0 & x_2 \\ 0 & 1 & 0 & y_2 \\ 0 & 0 & 1 & z_2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T_2R = \begin{pmatrix} a & b & c & x_2 \\ e & f & g & y_2 \\ h & i & j & z_2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{array}{lll} T_1 = \begin{pmatrix} 1 & 0 & 0 & x_1 \\ 0 & 1 & 0 & y_1 \\ 0 & 0 & 1 & z_1 \\ 0 & 0 & 0 & 1 \end{pmatrix} & T_2 = \begin{pmatrix} 1 & 0 & 0 & x_2 \\ 0 & 1 & 0 & y_2 \\ 0 & 0 & 1 & z_2 \\ 0 & 0 & 0 & 1 \end{pmatrix} & T_1T_2 = \begin{pmatrix} 1 & 0 & 0 & x_1+x_2 \\ 0 & 1 & 0 & y_1+y_2 \\ 0 & 0 & 1 & z_1+z_2 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

- ▶ Pour le positionnement, OpenGL gère une matrice homogène qui exprime le changement de repère $\mathcal{T}_{Eye \rightarrow Courant}$.
- ▶ Cette matrice est appelée la MODELVIEW.
- ▶ On déplace le repère courant à loisir pour placer les repères objets où on le souhaite en modifiant la valeur de la MODELVIEW.
- ▶ \Rightarrow Tous les sommets rencontrés à l'exécution subissent la matrice MODELVIEW.

Exemple : on appelle `TracerCarre()`

```
void TracerCarre() {  
    glBegin(GL_POLYGON); // trace un polygone dont les sommets suivent  
        glVertex3f(1.0,1.0,0.0); // premier sommet P(1,1,0)  
        glVertex3f(-1.0,1.0,0.0);  
        glVertex3f(-1.0,-1.0,0.0);  
        glVertex3f(1.0,-1.0,0.0);  
    glEnd();  
}
```

\Rightarrow A l'exécution des `glVertex3f(x,y,z)` ; OpenGL applique à $(x,y,z,1) = P_{Object}$ la MODELVIEW :
$$P_{Eye} = MODELVIEW * P_{Object}.$$

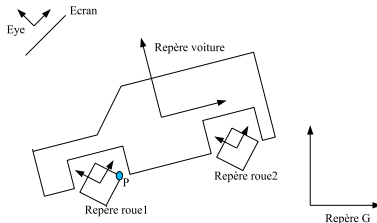
Modifications de la MODELVIEW

- ▶ `glLoadIdentity()` : affecte la matrice courante avec l'identité
- ▶ `glLoadMatrixd(double m[16])` : affecte la matrice avec m (décrite en colonnes).
- ▶ `glTranslatef(tx,ty,tz)` : multiplie à droite la matrice courante avec la matrice de translation T ($\Rightarrow \text{MODELVIEW} \leftarrow \text{MODELVIEW} \times T$).
- ▶ `glRotatef(theta,ux,uy,uz)` : multiplie à droite la matrice courante avec la matrice de rotation R .
- ▶ `glMultMatrixd(double m[16])` : multiplie à droite la matrice courante avec la matrice m .

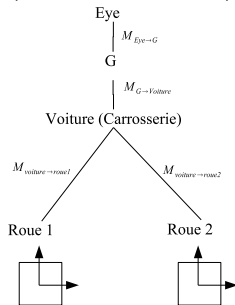
Remarque pour la rotation : on donne l'angle (en degrés !) et le vecteur directeur de l'axe de rotation (l'axe passe toujours par l'origine).

5 Conception hiérarchique

Conception hiérarchique



Le positionnement relatif des repères peut se représenter sous forme d'arbre :



- ▶ Chaque relation se traduit par un changement de repère $M_{pere \rightarrow fils}$
- ▶ Chaque sous-arbre doit pouvoir se concevoir indépendamment de son père (i.e. on peut associer à chaque noeud un `tracerNoeud` sans se préoccuper du noeud père).
- ▶ On remarque que les deux roues sont identiques dans leurs repères locaux (ce sera la même procédure qui tracera les deux roues).

- ▶ On dispose d'une seule matrice (la `MODELVIEW`) qui représente là où se trouve le repère courant.
- ▶ Lorsqu'un noeud a plusieurs fils, on doit mémoriser le repère du père.
- ▶ Eviter de faire des transformations inverses \Rightarrow préférer mémoriser les repères.
- ▶ \Rightarrow utilisation d'une pile pour mémoriser un repère.

- ▶ `glPushMatrix()` empile une copie de la `MODELVIEW`.
- ▶ `glPopMatrix()` recopie le haut de la pile dans la `MODELVIEW` et dépile.
- ▶ Ainsi un `glPopMatrix()` permet de retrouver le repère courant qu'on avait lors du `glPushMatrix()` correspondant.

L'exemple

Appliquer(R1,R2) correspond à une suite d'instructions qui modifie la MODELVIEW (glTranslate, etc).

```
void TracerRoue () {
glPushMatrix ();
    tracerCarre ();
glPopMatrix ();
}

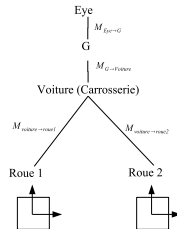
void TracerVoiture () {
glPushMatrix ();

    // MODELVIEW = M_Eye_Voiture (Repere courant = Voiture)
    TracerCarrosserie ();
glPushMatrix (); // Sauvegarde de M_Eye_Voiture
    Appliquer(Voiture,Roue1); // MODELVIEW = M * M_Voiture_Roue1
    // i.e. Repère courant = Roue1

    TracerRoue ();
glPopMatrix (); // depiler la pile MODELVIEW.
// Le haut de pile est donc M_Eye_Voiture
// i.e. Repère courant = Voiture

    Appliquer(Voiture,Roue2); // MODELVIEW = M * M_Voiture_Roue2
    // i.e. Repère courant = Roue2

    TracerRoue ();
glPopMatrix ();
}
```

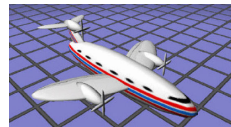
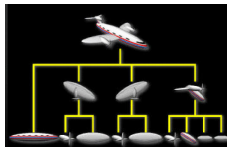
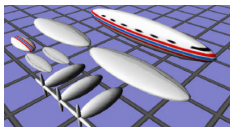


```
void TracerScene () {
glPushMatrix ();
    Appliquer(Eye,G);
    Appliquer(G,Voiture);
    TracerVoiture ();
glPopMatrix ();
}
```

Remarque : la MODELVIEW doit être initialisée par glLoadIdentity() dans une procédure d'initialisation.

Conception hiérarchique : conclusion

- ▶ Concevoir les composants dans des repères locaux les plus simples (ou intuitifs) possibles.
- ▶ Assembler les composants hiérarchiquement (sous forme d'arbres), en les positionnant relativement les uns par rapport aux autres (on place les fils par rapport au père par un changement de repère $M_{père \rightarrow fils}$).



- ▶ La conception hiérarchique (ou plus généralement les graphes de scène) est fondamentale et apparaît dans nombreux formats de scènes 3D (VRML, X3D, ...) et bibliothèques (Java3D, Ogre3D, ...).

6 Manipulations usuelles de changements de repères

Les 3 repères fondamentaux d'une scène 3D

On distingue avant tout 3 repères pour la conception et la visualisation d'une scène :

- ▶ Le repère du monde ou repère de scène (i.e. *World*) : c'est le repère de référence par rapport auquel seront positionnés tous les objets. Dans un graphe de scène, c'est le noeud racine.
- ▶ Le repère de la caméra (i.e. *Eye* ou *Camera*) : c'est le repère du point de vue/de l'observateur (nécessaire pour la visualisation). On place *Eye* par rapport au repère *World*.
- ▶ Le repère objet (i.e. *Object*) : c'est le repère de l'objet (i.e. noeud du graphe de scène) qu'on considère. On place *Object* par rapport à son noeud père (i.e. par rapport à *World* si un seul niveau de hiérarchie).

Autrement dit, en tant que concepteur, on spécifie :

- ▶ $\mathcal{T}_{World \rightarrow Eye}$: placement de la caméra dans le monde
- ▶ $\mathcal{T}_{World \rightarrow Object}$: placement de l'objet dans le monde (placement éventuellement décomposé par hiérarchie).

Exemple : position de la caméra en OpenGL

- ▶ En OpenGL, le repère initial de tracé est le repère Eye .
- ▶ Lors du tracé d'un objet, il faut qu'OpenGL dispose donc de $\mathcal{T}_{Eye \rightarrow Object}$ (passage mémorisé dans la matrice `MODELVIEW`).
- ▶ Comment l'obtenir à partir de $\mathcal{T}_{World \rightarrow Eye}$ et $\mathcal{T}_{World \rightarrow Object}$? (si on conçoit la scène ainsi).
- ▶ $\Rightarrow pastEyeObject = \mathcal{T}_{Eye \rightarrow World} \mathcal{T}_{World \rightarrow Object}$ (décomposer).
- ▶ et $\mathcal{T}_{Eye \rightarrow World}$ peut être déterminé à partir de $\mathcal{T}_{World \rightarrow Eye}$: c'est le passage inverse.

Changements inverses

- ▶ L'inverse de $M_{1 \rightarrow 2}$ est notée $M_{1 \rightarrow 2}^{-1}$ et identique à $M_{2 \rightarrow 1}$
- ▶ Inverse d'un produit de matrices : $(M_1 M_2 M_3)^{-1} = M_3^{-1} M_2^{-1} M_1^{-1}$ (attention à l'ordre !)
- ▶ Inverse d'une composition de changements de repères :
 $(M_{1 \rightarrow 2} M_{2 \rightarrow 3} M_{3 \rightarrow 4})^{-1} = M_{4 \rightarrow 3} M_{3 \rightarrow 2} M_{2 \rightarrow 1}$
- ▶ Dans le cas général l'inversion consiste à résoudre $MM^{-1} = I$ (pivot de gauss, par calcul de déterminant, ...),
- ▶ Mais :
 - La translation T inverse est la translation de vecteur opposé $-T$.
 - La rotation inverse R_θ est la rotation d'angle opposé $R_{-\theta}$
 - Le changement d'échelle $S(k_x, k_y, k_z)$ inverse est $S(\frac{1}{k_x}, \frac{1}{k_y}, \frac{1}{k_z})$.
 - L'inverse de toute matrice orthonormale (matrice 3×3 de rotation par exemple, ou tout passage de repère orthonormé) s'obtient en la transposant :

$$\begin{pmatrix} a_0 & b_0 & c_0 \\ a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{pmatrix}^{-1} = \begin{pmatrix} a_0 & a_1 & a_2 \\ b_0 & b_1 & b_2 \\ c_0 & c_1 & c_2 \end{pmatrix} \text{ si orthonormal}$$

Exemple d'inversion en OpenGL

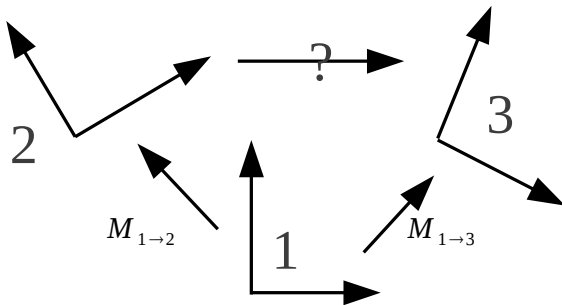
- Si un passage est traduit par :

```
glTranslatef(x,y,z);      // 1→2  
glRotatef(angle,ux,uy,uz); // 2→3
```

- Le code suivant correspond au passage inverse :

```
glRotatef(-angle,ux,uy,uz); // 3→2  
glTranslatef(-x,-y,-z);     // 2→1
```

Autre exemple



- ▶ On connaît $M_{1 \rightarrow 2}$ et $M_{1 \rightarrow 3}$, quelle est la matrice $M_{2 \rightarrow 3}$?
- ▶ Solution : $M_{2 \rightarrow 3} = M_{2 \rightarrow 1} M_{1 \rightarrow 3}$ (il reste à inverser $M_{1 \rightarrow 2}$ pour avoir $M_{2 \rightarrow 1}$).

Remarque sur la matrice TR

- ▶ Tout changement de repères orthonormés directs s'exprime par une matrice de la forme TR .
- ▶ \Rightarrow Il s'agit d'une composition que l'on retrouve très souvent (transformation rigide).

Soit 2 changements de repère T (translation) et R (rotation).
alors

$$TR = \left(\begin{array}{c|c} R_{3D} & T_{3D} \\ \hline 0 & 1 \end{array} \right)$$

- ▶ L'inverse ? Il suffit de transposer R_{3D} (inverse d'une matrice orthonormale) et d'opposer T_{3D} :

$$\begin{aligned} (TR)^{-1} &= R^{-1} T^{-1} = \left(\begin{array}{c|c} R_{3D}^t & 0 \\ \hline 0 & 1 \end{array} \right) \left(\begin{array}{c|c} I & -T_{3D} \\ \hline 0 & 1 \end{array} \right) \\ &= \left(\begin{array}{c|c} R_{3D}^t & -R_{3D}^t T_{3D} \\ \hline 0 & 1 \end{array} \right) \end{aligned}$$

- Soient $1 = (O^{(1)}, i^{(1)}, j^{(1)}, k^{(1)})$ et $2 = (O^{(2)}, i^{(2)}, j^{(2)}, k^{(2)})$, deux repères orthonormés. On connaît les expressions de l'origine et de la base de 2 dans le repère 1.

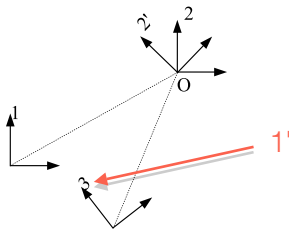
Alors :

$$M_{1 \rightarrow 2} = \left(\begin{array}{ccc|c} i_1^{(2)} & j_1^{(2)} & k_1^{(2)} & O_1^{(2)} \\ \hline 0 & 0 & 0 & 1 \end{array} \right)$$

- L'inverse $M_{2 \rightarrow 1}$? il suffit de faire comme pour $(TR)^{-1}$ (transposer le bloc haut-gauche (c'est une rotation) et d'opposer le bloc haut-droit (c'est une translation) et faire le produit des inverses)

Transformation exprimée dans un autre repère

- ▶ Exemple : rotation du repère 1 autour de O (i.e. rotation par rapport au repère 2) pour obtenir $1'$.
- ▶ Revient à “attacher” le repère 1 au repère 2.



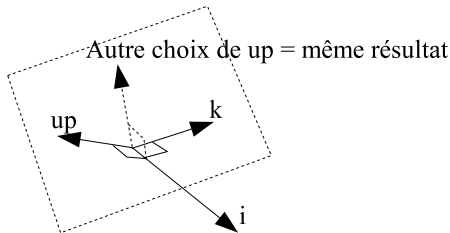
- ▶ $\Rightarrow M_{1 \rightarrow 1'} = M_{1 \rightarrow 2} M_{2 \rightarrow 2'} M_{2' \rightarrow 1'}$
- ▶ avec $M_{2 \rightarrow 2'} = R$ et $M_{2' \rightarrow 1'} = M_{2 \rightarrow 1}$.
- ▶ $\Rightarrow M_{1 \rightarrow 1'} = M_{1 \rightarrow 2} R M_{2 \rightarrow 1}$

LookAt

Placer la caméra par rapport à $World$ en donnant sa position A_{World} , quel point elle regarde (point At_{World}) et son roulis par un vecteur Up_{World} .

► Construire (i, j, k) :

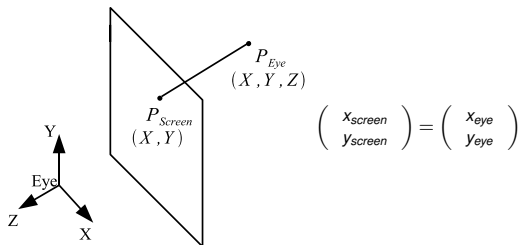
- k est donné par $k = O - At$, et on le normalise.
- i est tel que $i = up \times k$
- enfin j est calculé par $j = k \times i$
- La matrice $M_{World \rightarrow Eye}$ est donnée en mettant en colonne i, j, k et O (repère étant orthonormé).



7 Projection

Projection orthogonale

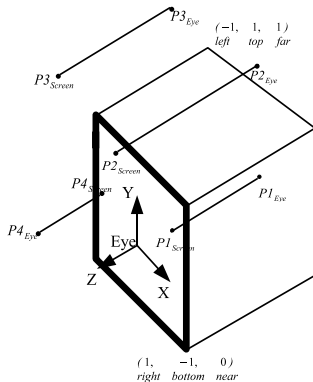
- ▶ En OpenGL tout sommet subit la `MODELVIEW` pour obtenir un point exprimé dans le repère Eye.
- ▶ Comment en déduire les coordonnées à l'écran ?
- ▶ Exemple : projection orthogonale.



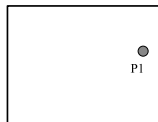
Exercice : quelle est la matrice homogène pour passer de P_{eye} à P_{screen} sachant que l'écran est à une distance `near` de eye ?

Projection orthogonale en OpenGL

- ▶ La projection est mémorisée dans une matrice `PROJECTION`.
- ▶ Tout sommet subit $P_{\text{Projeté}} = \text{PROJECTION} * \text{MODELVIEW} * P_{\text{Object}}$
- ▶ `PROJECTION` est calculée pour s'assurer que toutes les coordonnées (incluant la coordonnée z) soit dans l'intervalle $[-1, 1]$ (coordonnées dites normalisées).
- ▶ \Rightarrow calcul de `PROJECTION` à partir d'un volume de visualisation :



`glOrtho(-1,1,-1,1,0,1)`



Ecran obtenu

Projection orthogonale en OpenGL

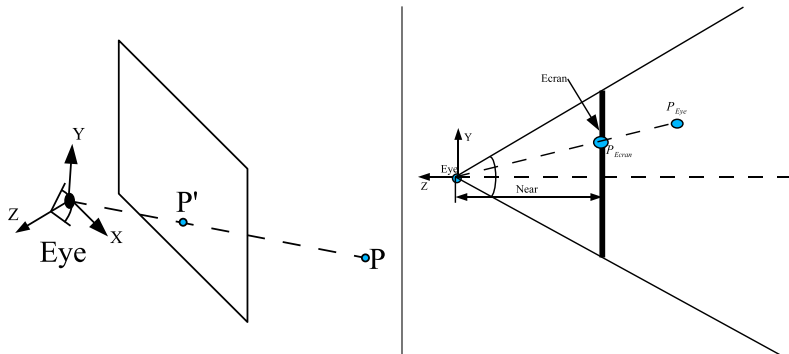
$$P_{\text{Projeté}} = M_{\text{PROJECTION}} P_{\text{Eye}}$$

avec

$$M_{\text{PROJECTION}} = \begin{pmatrix} \frac{2}{\text{right} - \text{left}} & 0 & 0 & -\frac{\text{right} + \text{left}}{\text{right} - \text{left}} \\ 0 & \frac{2}{\text{top} - \text{bottom}} & 0 & -\frac{\text{top} + \text{bottom}}{\text{top} - \text{bottom}} \\ 0 & 0 & -\frac{2}{\text{far} - \text{near}} & -\frac{\text{far} + \text{near}}{\text{far} - \text{near}} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- ▶ C'est la matrice calculée par l'instruction `glOrtho(left, right, bottom, top, near, far)`.
- ▶ résulte de l'application d'une règle de 3 : passer de $x_{\text{eye}} \in [\text{left}, \text{right}]$ à $x_p \in [-1, 1]$.

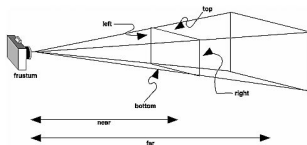
Projection perspective



Exercice : quelle est la matrice de passage $M_{Screen \rightarrow Eye}$?

Projection Perspective

- Définition d'un volume de visualisation pour normaliser les coordonnées (en OpenGL : `glFrustum(left, right, bottom, top, near, far)`).



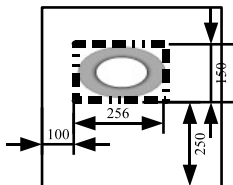
- Attention ! `near` pas "trop proche" de 0, et `far-near` pas "trop grand".

$$M_{\text{PROJECTION}} = \begin{pmatrix} 2 \frac{\text{near}}{\text{right} - \text{left}} & 0 & \frac{\text{right} + \text{left}}{\text{right} - \text{left}} & 0 \\ 0 & 2 \frac{\text{near}}{\text{top} - \text{bottom}} & \frac{\text{top} + \text{bottom}}{\text{top} - \text{bottom}} & 0 \\ 0 & 0 & -\frac{\text{far} + \text{near}}{\text{far} - \text{near}} & -2 \frac{\text{far} * \text{near}}{\text{far} - \text{near}} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

⇒ "complexité" liée à l'élimination des parties cachées (calcul de la profondeur) : sera justifié dans le chapitre correspondant.

► Phase géométrique :

- Tout sommet subit $P_p = M_{\text{PROJECTION}} M_{\text{MODELVIEW}} P_{\text{Object}}$
- P_p est toujours dans le repère caméra et en coordonnées homogènes.
- Les coordonnées normalisées sont obtenues en divisant P_p par sa coordonnée homogène.
- On obtient les coordonnées entières à l'écran de P_p en appliquant un viewport (définition de la fenêtre graphique) :
- Exemple :



`glViewport(100,250,256,150)`

`glViewport(x_min, y_min, width, height)`

$$\Rightarrow \begin{cases} x_{\text{ecran}} = (x + 1) \frac{\text{width}}{2} + x_{\text{min}} \\ y_{\text{ecran}} = (y + 1) \frac{\text{height}}{2} + y_{\text{min}} \end{cases}$$

Spécification de la projection en OpenGL

```
void initGL() {  
    // définition de la matrice de projection (projection perspective ici)  
    glMatrixMode(GL_PROJECTION);  
    glLoadIdentity();  
    glFrustum(-1,1,-1,1,0.1,100);  
  
    // définition de la matrice de transformation (identité = repère courant sur Eye).  
    glMatrixMode(GL_MODELVIEW);  
    glLoadIdentity();  
  
    glViewport(0,0,width,height); // fen\^etre graphique d'OpenGL  
    ...  
}
```