Utilisation de la matrice MODELVIEW sous OpenGL

rixed@free.fr

August 27, 2002

Abstract

Avec OpenGL, peut on déplacer un objet indépendamment des autres ? Comment déplacer la caméra ? OpenGL a t-il ou pas une "caméra" ? Dans quel ordre multiplier les matrices pour obtenir tel effet ? Toutes ces questions sont quotidiennes sur les forums OpenGL (cf http://www.opengl.org). Apparemment, beaucoup d'utilisateurs d'OpenGL ne comprennent pas comment utiliser efficacement la pile de matrices MODELVIEW. Il s'agit en fait d'un fonctionnement tellement simple et naturel qu'OpenGL, contrairement à d'autres API graphiques, ne cherche pas à le masquer derrière une batterie de fonctions de maniement de la caméra. Cet article a pour but d'aider les programmeurs peut habitués aux mathématiques utilisées en 3D à utiliser OpenGL efficacement.

La première partie de l'article, purement mathématiques, consiste en un rappel de notions de géométrie et d'algèbre linéaire. Les notations employées sont un compromis entre les notations usuelles et ce que LaTeXm'a laissé faire.

La seconde partie est l'application de ces notions aux applications OpenGL les plus typiques.

Avertissement : Pour des raisons évidentes de simplicité, les formules de maths s'écrivent. Mais pour bien faire, elles devraient se dessiner. Je vous invite donc fortement à lire cet article avec une feuille blanche et un crayon afin de vous représenter graphiquement ce qui se cache derrière chaque formule au fur et à mesure que vous lisez. C'est à ce prix (modique ; à défaut de feuille on peut même écrire directement sur sa table, c'est très joli) que l'on comprend les maths.

Le lecteur pressé pourra sauter les sections écrite dans une fonte plus petite ; ces sections traitent des repères non orthonormés. La plupart des programmes OpenGL ne travaillant qu'avec des repères orthonormés, cette lecture n'est donc pas utile, ni la distinction entre coordonnées contra- et co-variantes.

Contents

List of Figures

1 Notions de géométrie

1.1 Éclaircissements sur les vecteurs

Un vecteur, c'est le déplacement d'un point à un autre, par exemple le vecteur $\overrightarrow{AB}$ est le déplacement qui conduit de A à B.

Pour pouvoir repérer n'importe quel point dans un espace à n dimensions (On utilisera dans cet article des espaces à deux ou à trois dimensions), il suffit de se donner n vecteurs non colinéaires[1] et une origine. N'importe quel point de l'espace est alors une combinaison unique de ces vecteurs : *tant* de fois le premier vecteur plus *tant* de fois le second, etc... Les valeurs des "*tant*" utilisés ici permettent donc de donner une position à tous les points de l'espace ; on les appelle *coordonnées* du point.

L'ensemble de ces vecteurs et l'origine que l'on s'est donné forment une base, ou un *repère*, vis à vis duquel chaque point n'a qu'un seul jeux de coordonnées possible.

Par exemple : si les vecteurs $\overrightarrow{OA}$, $\overrightarrow{OB}$ et $\overrightarrow{OC}$ forment un repère dans un espace à 3 dimensions, alors à chaque point P de cet espace on peut trouver des nombres x, y et z tels que $\overrightarrow{OP} = x \times \overrightarrow{OA} + y \times \overrightarrow{OB} + z \times \overrightarrow{OC}$. Le trio (x, y, z) est unique et est

[1] deux vecteurs sont colinéaires, ou proportionnels, si l'un est un multiple de l'autre. Deux vecteurs identiques sont un cas particulier de vecteurs colinéaires ou le facteur de proportionnalité vaut 1

appelé coordonnées contra-variantes de P dans le repère $\mathcal{R} = (O, \overrightarrow{OA}, \overrightarrow{OB}, \overrightarrow{OC})$. x est la coordonnée de P dans R le long de $\overrightarrow{OA}$, etc...

Lorsque les vecteurs d'un repère sont tous orthogonaux deux à deux, le repère est dit orthogonal. Si la longueur de tous les axes est l'unité, le repère est alors dit normé[2].

On se choisi souvent un repère construit avec un tel jeux de vecteurs pour jouer le rôle de repère absolu. Ce repère est une pure commodité : lorsqu'on parlera de coordonnées absolues, on voudra dire : coordonnées dans ce repère.

Un repère à la fois orthogonal et normal est dit *orthonormé.*

Lorsqu'on veut parler des coordonnées contra-variantes d'un point dans un repère particulier $\mathcal{R}' = (O, \overrightarrow{e0}, \overrightarrow{e1}, \overrightarrow{e2})$, on note le repère en indice supérieur, par exemple : $\overrightarrow{AB^{\mathcal{R}'}}$. Les coordonnées de $\overrightarrow{AB^{\mathcal{R}'}}$ et de $\overrightarrow{AB^{\mathcal{R}}}$ ne seront donc pas les mêmes !

On aura remarqué que les vecteurs s'ajoutent et se multiplient comme des déplacements :

(a, b, c) et (d, e, f) étant les coordonnées de deux vecteurs (dans un même repère), et n un nombre, on a :

$$n \times (a, b, c) = (n \times a, n \times b, n \times c)$$

$$(a, b, c) + (d, e, f) = (a + d, b + e, c + f)$$

La définition des coordonnées données ci-dessus signifie donc ceci : le déplacement qui conduit de O à A est donnée dans le repère $\mathcal{R} = (O, \overrightarrow{e0}, \overrightarrow{e1}, \overrightarrow{e2})$ par x fois le déplacement $\overrightarrow{e0}$, plus y fois le déplacement $\overrightarrow{e1}$, etc... où (x, y, z) sont les coordonnées de $\overrightarrow{OA^{\mathcal{R}}}$. Les déplacements le long des trois axes $\overrightarrow{e0}$, $\overrightarrow{e1}$ et $\overrightarrow{e2}$ pouvant bien sur être effectués dans n'importe quel ordre.

Géométriquement, voici ce que ça donne, en 2D (j'ai fait exprès de prendre des vecteurs du repères $(O, \overrightarrow{OA}, \overrightarrow{OA})$ complètement quelconques) :

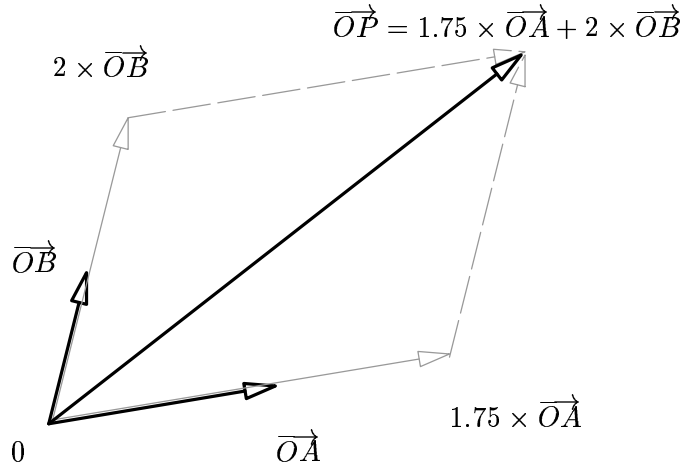

Figure 1: Coordonnées contra-variantes

[2] cela suppose que l'espace soit doté d'une mesure "absolue" de distance et d'angle

1.2 Produit scalaire

Nous allons voir maintenant une autre façon de se donner des coordonnées à partir d'un repère, coordonnées dites covariantes. Nous allons le faire dans un espace à deux dimensions car on voit mieux ce qui se passe et les calculs sont plus simples. Les résultats auxquels nous allons aboutir restent vrais en trois dimensions.

Le produit scalaire entre deux vecteurs $\overrightarrow{OP}$ et $\overrightarrow{OQ}$ donne un nombre, noté

$$\overrightarrow{OP} \bullet \overrightarrow{OQ}$$

Une définition géométrique du produit scalaire pourrait être celle-ci : le produit scalaire entre le vecteur $\overrightarrow{OP}$ et le vecteur $\overrightarrow{OQ}$ est égal à la longueur de $\overrightarrow{OP}$ projeté sur $\overrightarrow{OQ}$ fois la longueur de $\overrightarrow{OQ}$. Ceci donne la formule suivante :

$$\overrightarrow{OP} \bullet \overrightarrow{OQ} = \|\overrightarrow{OP}\| \times \|\overrightarrow{OQ}\| \times \cos\gamma$$

où γ est l'angle entre $\overrightarrow{OP}$ et $\overrightarrow{OQ}$, et où $\|\overrightarrow{OP}\|$ désigne la norme[3] de $\overrightarrow{OP}$, etc.

On voit que $\overrightarrow{OP} \bullet \overrightarrow{0Q} = 0$ seulement si la longueur de $\overrightarrow{0P}$ ou de $\overrightarrow{OQ}$ est nulle, ou-bien si $\overrightarrow{OP}$ et $\overrightarrow{OQ}$ sont perpendiculaires.

Ceci nous amène à un nouveau jeux de coordonnées, appelé *coordonnées covariantes* (Cf figure 2) : on peut en effet repérer la position d'un point grâce aux produits scalaires entre le vecteur position de ce point[4] et chacun des vecteurs du repère. Dans $\mathcal{R} = (O, \overrightarrow{OA}, \overrightarrow{OB})$, les produits scalaires d'un vecteur $\overrightarrow{OP}$ par les vecteur $\overrightarrow{OA}$ et $\overrightarrow{OB}$ donnent en effet deux nombres qui permettent de repérer sans ambiguïté la position du point dans le repère[5].

On notera désormais les coordonnées covariantes avec un indice en bas et les coordonnées contra-variantes avec un indice en haut.

La relation entre les coordonnées covariantes et contra-variantes dépend du produit scalaire des vecteurs du repère entre eux. Dans le cas d'un repère orthonormé, ces coordonnées sont égales, ce qui est très pratique.

On peut trouver une formule simple pour calculer $\overrightarrow{OP} \bullet \overrightarrow{OQ}$ dans le repère $\mathcal{R} = (O, \overrightarrow{OA}, \overrightarrow{OB})$ *si* $\mathcal{R}$ *est orthonormé* :

$$\overrightarrow{OP} \bullet \overrightarrow{OQ} = \|\overrightarrow{OP}\| \times \|\overrightarrow{OQ}\| \times \cos(\alpha - \beta)$$

où α est l'angle entre les vecteurs $\overrightarrow{OP}$ et $\overrightarrow{OA}$ et β l'angle entre $\overrightarrow{OQ}$ et $\overrightarrow{OA}$.

$$\overrightarrow{OP} \bullet \overrightarrow{OQ} = \|\overrightarrow{OP}\| \times \|\overrightarrow{OQ}\| \times (\cos\alpha \times \cos\beta + \sin\alpha \times \sin\beta)$$

Or, $\|\overrightarrow{OP}\| \times \cos\alpha$ et $\|\overrightarrow{OP}\| \times \sin\alpha$ sont les coordonnées covariantes de $\overrightarrow{OP}$ dans le repère $\mathcal{R}$ et $\|\overrightarrow{B}\| \times \cos\beta$, $\|\overrightarrow{B}\| \times \sin\beta$ les coordonnées covariantes de $\overrightarrow{B}$ dans ce même repère (on utilise ici l'hypothèse que $\mathcal{R}$ est orthonormé). Notons (P^x, P^y) et (Q^x, Q^y) ces coordonnées (contra-variantes).

[3] norme = longueur

[4] vecteur position = vecteur allant de l'origine du repère considéré au point

[5] Si les axes du repère ne sont pas normés, comme c'est la cas dans la figure 2, on représente non pas $\overrightarrow{OP} \bullet \overrightarrow{OA} \times \overrightarrow{OA}$ mais plutôt $\frac{\overrightarrow{OP} \bullet \overrightarrow{OA}}{\|\overrightarrow{OA}\|^2} \times \overrightarrow{OA}$ afin de faire apparaitre l'angle droit de la projection de $\overrightarrow{OP}$ sur $\overrightarrow{OA}$ (de même avec $\overrightarrow{OB}$). Cela reviens à travailler avec les versions normées de ces deux vecteurs

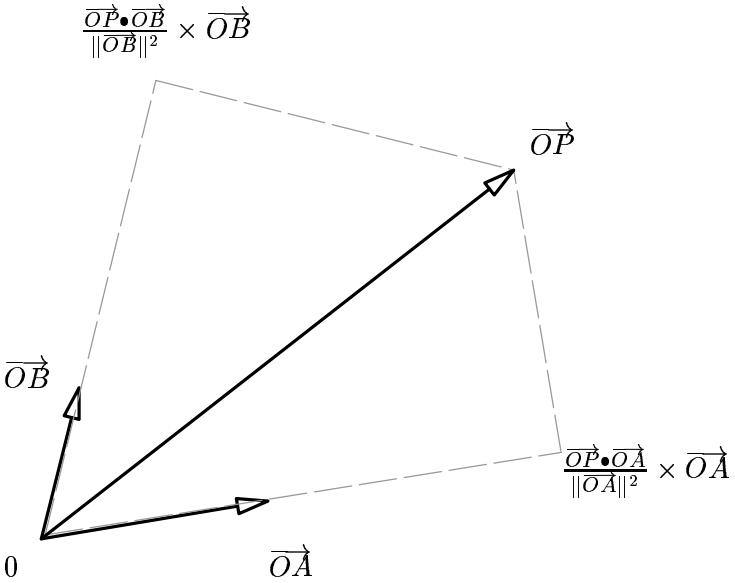

Figure 2: Coordonnées co-variantes

On a alors :

$$\overrightarrow{OP} \bullet \overrightarrow{OQ} = P^x \times Q^x + P^y \times Q^y$$

C'est cette relation que l'on utilise souvent pour calculer un produit scalaire.

Dans le cas général où $\mathcal{R}$ est quelconque, on a :

$$\overrightarrow{OP} \bullet \overrightarrow{OQ} = (P^x \times \overrightarrow{OA} + P^y \times \overrightarrow{OB}) \bullet (Q^x \times \overrightarrow{OA} + Q^y \times \overrightarrow{OB})$$

On peut alors développer le produit scalaire (le produit scalaire est distributif et commutatif) :

$$\overrightarrow{OP} \bullet \overrightarrow{OQ} = P^x \times Q^x \times \overrightarrow{OA}^2 + P^x \times Q^y \times 2 \times \overrightarrow{OA} \bullet \overrightarrow{OB} + P^y \times Q^y \times \overrightarrow{OB}^2$$

On retrouve donc la formule précédente si $\mathcal{R}$ est orthonormé.

1.3 Changement de repère

Mettons qu'on ait un repère $\mathcal{R} = (O, \overrightarrow{A}, \overrightarrow{B})$, et les vecteurs $\overrightarrow{V}$, $\overrightarrow{C}$ et $\overrightarrow{D}$ (dorénavant, on note simplement $\overrightarrow{V}$ le vecteur position $\overrightarrow{OV}$).

Si les points C et D ne sont pas alignés on peut former un repère $\mathcal{R}' = (O, \overrightarrow{C}, \overrightarrow{D})$. Supposons que l'on connaisse les coordonnées contra-variantes (V'^x, V'^y) de $\overrightarrow{V}$ (on note donc $\overrightarrow{V^{\mathcal{R}'}}$), et les coordonnées contra-variantes (C^x, C^y) et (D^x, D^y) de $\overrightarrow{C}$ et $\overrightarrow{D}$ dans $\mathcal{R}$ (notées donc respectivement $\overrightarrow{C^{\mathcal{R}}}$ et $\overrightarrow{D^{\mathcal{R}}}$).

Cherchons les coordonnées (V^x, V^y) de $\overrightarrow{V}$ dans $\mathcal{R}$ ($\overrightarrow{V^{\mathcal{R}}}$) :

Par définition :

$$\overrightarrow{V} = V'^x \times \overrightarrow{C} + V'^y \times \overrightarrow{D}$$

Explicitons $\overrightarrow{C}$ et $\overrightarrow{D}$ dans $\mathcal{R}$:

$$\overrightarrow{V} = V'^x \times (C^x \times \overrightarrow{A} + C^y \times \overrightarrow{B}) + V'^y \times (D^x \times \overrightarrow{A} + D^y \times \overrightarrow{B})$$

$$\overrightarrow{V} = (V'^x \times C^x + V'^y \times D^x) \times \overrightarrow{A} + (V'^x \times C^y + V'^y \times D^y) \times \overrightarrow{B}$$

Ou encore :

$$(V^x, V^y) = (V'^x \times C^x + V'^y \times D^x, V'^x \times C^y + V'^y \times D^y)$$

Cette formule est assez intuitive.

En passant, on peut trouver des formules supplémentaires en utilisant les coordonnées covariantes (notées avec l'indice en bas) :

Par définition :

$$(V_x, V_y) = (\overrightarrow{V} \bullet \overrightarrow{A}, \overrightarrow{V} \bullet \overrightarrow{B})$$

Développons :

$$(V_x, V_y) = ((V'^x \times \overrightarrow{C} + V'^y \times \overrightarrow{D}) \bullet \overrightarrow{A}, (V'^x \times \overrightarrow{C} + V'^y \times \overrightarrow{D}) \bullet \overrightarrow{B})$$

$$(V_x, V_y) = (V'^x \times \overrightarrow{C} \bullet \overrightarrow{A} + V'^y \times \overrightarrow{D} \bullet \overrightarrow{A}, V'^x \times \overrightarrow{C} \bullet \overrightarrow{B} + V'^y \times \overrightarrow{D} \bullet \overrightarrow{B})$$

$$(V_x, V_y) = (V'^x \times C_x + V'^y \times D_x, V'^x \times C_y + V'^y \times D_y)$$

On peut le faire également dans l'autre sens :

$$(V'_x, V'_y) = (\overrightarrow{V} \bullet \overrightarrow{C}, \overrightarrow{V} \bullet \overrightarrow{D})$$

$$(V'_x, V'_y) = (\overrightarrow{V} \bullet (C^x \times \overrightarrow{A} + C^y \times \overrightarrow{B}), \overrightarrow{V} \bullet (D^x \times \overrightarrow{A} + D^y \times \overrightarrow{B}))$$

$$(V'_x, V'_y) = (C^x \times \overrightarrow{V} \bullet \overrightarrow{A} + C^y \times \overrightarrow{V} \bullet \overrightarrow{B}, D^x \times \overrightarrow{V} \bullet \overrightarrow{A} + D^y \times \overrightarrow{V} \bullet \overrightarrow{B})$$

$$(V'_x, V'_y) = (C^x \times V_x + C^y \times V_y, D^x \times V_x + D^y \times V_y)$$

Remarquez bien que si $\mathcal{R}$ et $\mathcal{R}'$ sont tous deux orthonormés, les coordonnées contra-variantes sont égales aux coordonnées covariantes et on obtient bien 3 fois la même formule.

1.4 Notation matricielle

En reprenant les repères $\mathcal{R}$ et $\mathcal{R}'$ du paragraphe précédent, je note la matrice $M^{\mathcal{R} \leftarrow \mathcal{R}'}$ le tableau suivant :

$$\begin{array}{c} \\ \overrightarrow{A} \\ \overrightarrow{B} \end{array} \begin{array}{c} \begin{array}{cc} \overrightarrow{C} & \overrightarrow{D} \end{array} \\ \begin{pmatrix} \mathrm{C}^x & \mathrm{D}^x \\ \mathrm{C}^y & \mathrm{D}^y \end{pmatrix} \end{array}$$

qui se lit comme ceci :

$$\begin{cases} \overrightarrow{C} = C^x \times \overrightarrow{A} + C^y \times \overrightarrow{B} \\ \overrightarrow{D} = D^x \times \overrightarrow{A} + D^y \times \overrightarrow{B} \end{cases}$$

Pourquoi cette notation loufoque ? J'ai noté en colonne les coordonnées contra-variantes des vecteurs écrits en haut par rapport aux vecteurs écris à gauche. Lorsqu'on multiplie ces tableaux (ou matrices) par un vecteur, il faut prendre les coordonnées du vecteur par rapport aux axes notés en haut, et on obtient comme résultat les coordonnées du vecteurs par rapport aux axes notés à gauche.

Cette multiplication d'une matrice par un vecteur se déroule et se note comme ceci :

$$\begin{array}{ccccc} & multiplication(\star) & \longrightarrow & & \overrightarrow{V^{\mathcal{R}'}} \\ & \downarrow & & & \begin{pmatrix} \mathrm{V'}^x \\ \mathrm{V'}^y \end{pmatrix} \\ & \overrightarrow{C} \quad \overrightarrow{D} & & & \overrightarrow{V^{\mathcal{R}}} \\ \overrightarrow{A} & \begin{pmatrix} \mathrm{C}^x & \mathrm{D}^x \end{pmatrix} & = & \begin{pmatrix} \mathrm{V}_x = C^x \times V'^x + D^x \times V'^y \end{pmatrix} \\ \overrightarrow{B} & \begin{pmatrix} \mathrm{C}^y & \mathrm{D}^y \end{pmatrix} & & \begin{pmatrix} \mathrm{V}_y = C^y \times V'^x + D^y \times V'^y \end{pmatrix} \end{array}$$

Ou encore :

$$M^{\mathcal{R}\leftarrow\mathcal{R}'} \star \overrightarrow{V^{\mathcal{R}'}} = \overrightarrow{V^{\mathcal{R}}}$$

Comme on le voit, cette notation ramasse beaucoup de significations.

Puisque le produit scalaire est commutatif, les coordonnées covariantes de $\overrightarrow{A}$ et de $\overrightarrow{B}$ dans le repère $\mathcal{R}'$ s'obtiennent simplement à partir des coordonnées covariantes de $\overrightarrow{C}$ et $\overrightarrow{D}$ dans $\mathcal{R}$. Dans le cas où les repères $\mathcal{R}$ et $\mathcal{R}'$ sont orthonormés, les coordonnées contra-variantes sont égales aux coordonnées covariantes. $M^{\mathcal{R}'\leftarrow\mathcal{R}}$ s'obtient alors facilement à partir de $M^{\mathcal{R}\leftarrow\mathcal{R}'}$:

$$\begin{array}{ccc} & \overrightarrow{A} & \overrightarrow{B} \\ \overrightarrow{C} & \begin{pmatrix} \mathrm{C}_x = C^x & \mathrm{C}_y = C^y \end{pmatrix} \\ \overrightarrow{D} & \begin{pmatrix} \mathrm{D}_x = D^x & \mathrm{D}_y = D^y \end{pmatrix} \end{array}$$

On a simplement inversé lignes et colonnes. Deux matrices que l'on déduit l'une de l'autre en inversant lignes et colonnes sont dites transposées l'une de l'autre. La transposée de M est notée $transp(M)$.

On peut donc rajouter :

$$M^{\mathcal{R}'\leftarrow\mathcal{R}} \star \overrightarrow{V^{\mathcal{R}}} = transp(M^{\mathcal{R}\leftarrow\mathcal{R}'}) \star \overrightarrow{V^{\mathcal{R}}} = \overrightarrow{V^{\mathcal{R}'}}$$

qui est la relation inverse de celle vue plus haut.

Obtenir les coordonnées d'un vecteur dans un repère orthonormé donné quand on les connaît dans un autre repère orthonormé consiste à chercher la matrice de passage du premier repère vers le second, c'est à dire à connaître soit les coordonnées des axes du premier repère dans le second repère, soit au contraire les axes du second dans le premier. La notation est alors un bon moyen mnémotechnique de retrouver quelles opérations effectuer pour trouver le résultat. Et c'est pour ça qu'on l'a inventé.

1.5 Résumons en 3 dimensions

Les coordonnées d'un vecteur $\overrightarrow{V}$ sont toujours définies dans un repère $\mathcal{R}$ formé par une origine O et trois vecteurs $\overrightarrow{A}$, $\overrightarrow{B}$ et $\overrightarrow{C}$: on note $\overrightarrow{V^{\mathcal{R}}}$ et on l'écrit en notation matricielle :

$$\begin{array}{cc} & \overrightarrow{V} \\ \overrightarrow{A} & \\ \overrightarrow{B} & \left(\begin{array}{c} \mathrm{V}^x \\ \mathrm{V}^y \\ \mathrm{V}^z \end{array} \right) \\ \overrightarrow{C} & \end{array}$$

On note en colonne les coordonnées et à côté des coordonnées les vecteurs par rapport auxquels ces coordonnées se rapportent.

Si le repère $\mathcal{R}$ est orthonormé, V_x, V_y et V_z sont les produits scalaires entre $\overrightarrow{V}$ et $\overrightarrow{A}$, $\overrightarrow{B}$ et $\overrightarrow{C}$.

Si O, $\overrightarrow{D}$, $\overrightarrow{E}$ et $\overrightarrow{F}$ forment un second repère orthonormé appelé $\mathcal{R}'$, la matrice formée par les coordonnées de $\overrightarrow{A}$, $\overrightarrow{B}$ et $\overrightarrow{D}$ par rapport à $\overrightarrow{D}$, $\overrightarrow{E}$ et $\overrightarrow{F}$, multipliée ($\star$) par un vecteur exprimé dans $\mathcal{R}$ donne le même vecteur exprimé dans $\mathcal{R}'$.

La matrice transposée multipliée par un vecteur exprimé dans $\mathcal{R}'$ donne le même vecteur exprimé dans $\mathcal{R}$.

1.6 Changement de repère multiple

Prenons maintenant 3 repères d'un espace à deux dimensions : $\mathcal{R}$ formé par O, $\overrightarrow{A}$ et $\overrightarrow{B}$, $\mathcal{R}'$ formé par O, $\overrightarrow{C}$ et $\overrightarrow{D}$ et $\mathcal{R}''$ formé par O, $\overrightarrow{E}$ et $\overrightarrow{F}$. On a un vecteur $\overrightarrow{V^{\mathcal{R}''}}$ défini dans $\mathcal{R}''$, on cherche ses coordonnées dans $\mathcal{R}$.

Supposons que l'on connaisse $\overrightarrow{E^{\mathcal{R}}}$ et $\overrightarrow{F^{\mathcal{R}}}$: on multiplierait $\overrightarrow{V^{\mathcal{R}''}}$ par la matrice formée par les coordonnées de $\overrightarrow{E^{\mathcal{R}}}$ et $\overrightarrow{F^{\mathcal{R}}}$ (i.e. exprimés dans $\mathcal{R}$) : on passera alors de $\mathcal{R}''$ à $\mathcal{R}$.

Le cas intéressant est celui où on ne connaît $\overrightarrow{E}$ et $\overrightarrow{F}$ que dans $\mathcal{R}'$, et où on connaît en plus $\overrightarrow{C}$ et $\overrightarrow{D}$ dans $\mathcal{R}$.

Il faut alors trouver $\overrightarrow{E^{\mathcal{R}}}$ et $\overrightarrow{F^{\mathcal{R}}}$.

Pour les trouver, il faut évidement multiplier $\overrightarrow{E^{\mathcal{R}'}}$ et $\overrightarrow{F^{\mathcal{R}'}}$ par la matrice formée par les vecteurs $\overrightarrow{C}$ et $\overrightarrow{D}$ exprimés dans $\mathcal{R}$: En notation matricielle, multiplier tous les vecteurs formant un repère $\mathcal{R}''$, exprimés dans $\mathcal{R}'$, par la matrice de passage de $\mathcal{R}'$ vers $\mathcal{R}$, se note comme ceci (en appelant (E'^x, E'^y) et (F'^x, F'^y) les coordonnées de $\overrightarrow{E^{\mathcal{R}'}}$ et $\overrightarrow{F^{\mathcal{R}'}}$ et (C^x, C^y), (D^x, D^y) les coordonnées de $\overrightarrow{C^{\mathcal{R}}}$ et $\overrightarrow{D^{\mathcal{R}}}$) :

$$\begin{array}{ccccc} & & & & \overrightarrow{E^{\mathcal{R}'}} \quad \overrightarrow{F^{\mathcal{R}'}} \\ & \overset{\star}{\downarrow} & \longrightarrow & \overrightarrow{C} & \left(\begin{array}{cc} \mathrm{E'}^x & \mathrm{F'}^x \\ \mathrm{E'}^y & \mathrm{F'}^y \end{array} \right) \\ & & & \overrightarrow{D} & \\ \overrightarrow{C} \quad \overrightarrow{D} & & & & \overrightarrow{E^{\mathcal{R}}} \quad \overrightarrow{F^{\mathcal{R}}} \\ \overrightarrow{A} \quad \left(\begin{array}{cc} \mathrm{C}^x & \mathrm{D}^x \\ \mathrm{C}^y & \mathrm{D}^y \end{array} \right) & = & & \left(\begin{array}{cc} \mathrm{E'}^x \times C^x + E'^y \times D^x & \mathrm{F'}^x \times C^x + F'^y \times D^x \\ \mathrm{E'}^x \times C^y + E'^y \times D^y & \mathrm{F'}^x \times C^y + F'^y \times D^y \end{array} \right) \\ \overrightarrow{B} & & & & \end{array}$$

On appelle composition de matrice cette multiplication, et on la note également $\star$. On obtient en bas à droite du tableau les coordonnées de $\overrightarrow{E}$ et $\overrightarrow{F}$ par rapport à $\overrightarrow{A}$ et $\overrightarrow{B}$, c'est à dire la matrice de passage de $\mathcal{R}''$ vers $\mathcal{R}$. On note en abrégé :

$$M^{\mathcal{R}\leftarrow\mathcal{R}''} = M^{\mathcal{R}\leftarrow\mathcal{R}'} \star M^{\mathcal{R}'\leftarrow\mathcal{R}''}$$

On peut ainsi composer les matrices à volonté, en prenant toujours soin de faire correspondre les vecteurs comme il faut (les vecteurs de la matrice en haut à droite doivent être définis par rapport aux vecteurs de la matrice en bas à gauche, et le résultat donnera les vecteurs de la matrice de droite par rapport aux mêmes vecteurs que les vecteurs de la matrice de gauche).

Dans le cas où on manipule des repères orthonormés, on pourra également utiliser les matrices transposées pour passer un vecteur dans les deux sens (du repère $\mathcal{R}_1$ vers le repère $\mathcal{R}_2$ pour les vecteurs exprimés dans $\mathcal{R}_1$ que l'on veut connaître dans $\mathcal{R}_2$, ou dans l'autre sens pour les vecteurs exprimés dans $\mathcal{R}_2$ et que l'on veut connaître dans $\mathcal{R}_1$).

1.7 Matrices 4x4

Jusqu'ici, tous nos repères étaient définis par rapport à la même origine, le point O. Que se passe t-il si on prend deux repères qui n'ont pas la même origine ? Prenons par exemple le repère $\mathcal{R}$ formé de l'origine O et des axes $\overrightarrow{A}$, $\overrightarrow{B}$ et $\overrightarrow{C}$ et le repère $\mathcal{R}'$ formé de l'origine P et des axes $\overrightarrow{D}$, $\overrightarrow{E}$ et $\overrightarrow{F}$. On a $\overrightarrow{V^{\mathcal{R}'}}$, on cherche $\overrightarrow{V^{\mathcal{R}}}$.

On sait déjà comment avoir $\overrightarrow{V^{\mathcal{R}''}}$, où $\mathcal{R}''$ est le repère formé par P, $\overrightarrow{A}$, $\overrightarrow{B}$ et $\overrightarrow{C}$. Géométriquement (Cf figure 3), on voit que $\overrightarrow{V^{\mathcal{R}}} = \overrightarrow{V^{\mathcal{R}''}} + \overrightarrow{OP^{\mathcal{R}}}$.

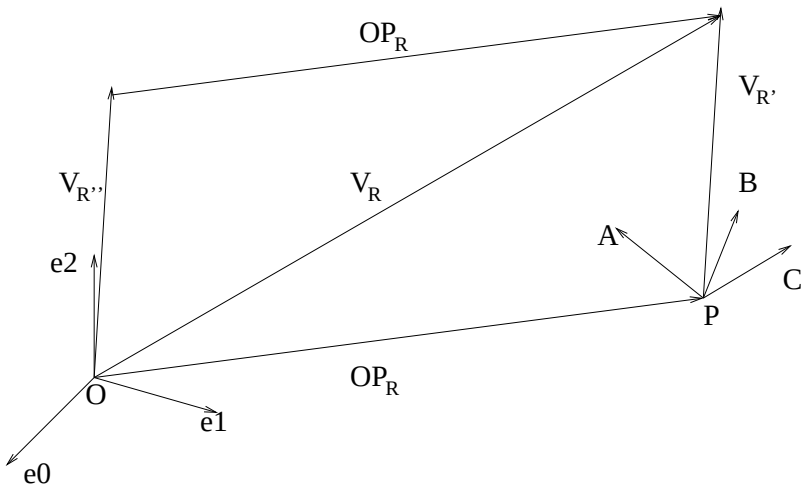

Figure 3: Changement de repère lorsque les origines ne coïncident pas

L'idée des matrices 4x4 est de réaliser ce changement de repère et cette translation supplémentaire dans un seul calcul matriciel. Pour rajouter cette opération, on rajoute une colonne à la matrice habituelle, colonne contenant les coordonnées du déplacement à ajouter au nouveau vecteur. Pour des raisons (essentiellement) de commodité, on rajoute également une ligne afin d'avoir le même nombre de lignes que de colonnes.

Prenons la matrice 4x4 suivante :

$$\begin{array}{c} \\ \overrightarrow{A} \\ \overrightarrow{B} \\ \overrightarrow{C} \\ \overrightarrow{?} \end{array} \begin{array}{c} \begin{array}{cccc} \overrightarrow{D} & \overrightarrow{E} & \overrightarrow{F} & \overrightarrow{T} \end{array} \\ \begin{pmatrix} \mathrm{D}^x & \mathrm{E}^x & \mathrm{F}^x & \mathrm{T}^x \\ \mathrm{D}^y & \mathrm{E}^y & \mathrm{F}^y & \mathrm{T}^y \\ \mathrm{D}^z & \mathrm{E}^z & \mathrm{F}^z & \mathrm{T}^z \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

(Je note '$\overrightarrow{?}$' un vecteur qui n'a pas de signification évidente, et dont la ligne vaudra toujours (0,0,0,1) dans cette article).

Multiplions cette matrice par le vecteur exprimé dans $\mathcal{R}'$ avec une coordonnée supplémentaire vallant toujours 1 :

$$\begin{array}{c} \\ \overrightarrow{D} \\ \overrightarrow{E} \\ \overrightarrow{F} \\ \overrightarrow{?} \end{array} \begin{array}{c} \begin{array}{c} \overrightarrow{V^{\mathcal{R}'}} \end{array} \\ \begin{pmatrix} \mathrm{V}^x \\ \mathrm{V}^y \\ \mathrm{V}^z \\ 1 \end{pmatrix} \end{array}$$

On obtiendra :

$$\begin{array}{c} \\ \overrightarrow{A} \\ \overrightarrow{B} \\ \overrightarrow{C} \\ \overrightarrow{?} \end{array} \begin{array}{c} \begin{array}{c} \overrightarrow{V^{\mathcal{R}}} \end{array} \\ \begin{pmatrix} \mathrm{V}^x \times D^x + V^y \times E^x + V^z \times F^x + T^x \\ \mathrm{V}^x \times D^y + V^y \times E^y + V^z \times F^y + T^y \\ \mathrm{V}^x \times D^z + V^y \times E^z + V^z \times F^z + T^z \\ 1 \end{pmatrix} \end{array}$$

On a donc le même $\overrightarrow{V^{\mathcal{R}}}$ que l'on aurait obtenu avec une matrice 3×3 normale, plus le vecteur $\overrightarrow{T^{\mathcal{R}}}$

On voit donc que pour pouvoir ajouter un vecteur constant au vecteur $\overrightarrow{V^{\mathcal{R}}}$ au cours de la multiplication du vecteur $\overrightarrow{V^{\mathcal{R}'}}$ par $M^{\mathcal{R}\leftarrow\mathcal{R}'}$, il suffit de rajouter $\overrightarrow{T^{\mathcal{R}}}$ dans la dernière colonne de la matrice, et 1 dans les coordonnées de $\overrightarrow{V}$.

Il faut donc, pour passer de $\mathcal{R}$ à $\mathcal{R}'$, que $\overrightarrow{T}$ vaille $\overrightarrow{OP^{\mathcal{R}}}$, c'est à dire la position de l'origine de $\mathcal{R}'$ exprimée dans $\mathcal{R}$.

Avec les matrices 4x4, on peut donc passer de repère à repère même s'ils n'ont pas d'origine commune en une seule opération.

1.8 Composition de matrice 4x4

La composition de matrices 4x4 en cas de repères imbriqués se fait sans difficultés. Quelque soit les matrices, leur taille et leur signification, multiplier successivement un vecteur par deux matrices est équivalent à multiplier le vecteur par la composées des matrices. Le vecteur $\overrightarrow{T}$ de la matrice composée, sera la vecteur $\overrightarrow{T}$ de la matrice de droite, multiplié par la matrice de gauche (tout comme n'importe quel vecteur de la matrice de droite).

1.9 Inversion de matrice 4x4

L'inversion de matrice 4x4, par contre, ne se fait pas aussi naturellement que pour les matrices 3x3. Lorsque l'origine des repères est la même, on a vu que la matrice inverse de $M_{3\times3}^{\mathcal{R}\leftarrow\mathcal{R}'}$, c'est à dire $M_{3\times3}^{\mathcal{R}'\leftarrow\mathcal{R}}$, était simplement la transposée de $M_{3\times3}^{\mathcal{R}\leftarrow\mathcal{R}'}$. Lorsque $\overrightarrow{T}$ figure dans la matrice, la transposée n'a plus du tout de signification géométrique. L'inverse de $M_{4\times4}^{\mathcal{R}\leftarrow\mathcal{R}'}$ doit en effet toujours avoir un vecteur $\overrightarrow{T}$ dans la dernière colonne, et si on prenait la transposée de $M_{4\times4}^{\mathcal{R}\leftarrow\mathcal{R}'}$ on se retrouverait toujours avec le vecteur $(0,0,0,1)$ dans le dernière colonne, ce qui ne mènerait à rien.

Si on peut toujours transposer la partie "rotation" de la matrice (les 9 valeurs en haut à gauche), il faut donc faire quelque-chose de spécial pour $\overrightarrow{T}$.

On voit géométriquement que le vecteur $\overrightarrow{T}$ de $M_{4\times4}^{\mathcal{R}'\leftarrow\mathcal{R}}$ doit valoir, non plus $\overrightarrow{OP^{\mathcal{R}}}$, mais $\overrightarrow{PO^{\mathcal{R}'}}$. Si on ne connaît pas ce vecteur, on peut le calculer par la méthode habituelle ($\overrightarrow{PO^{\mathcal{R}'}} = M_{3\times3}^{\mathcal{R}'\leftarrow\mathcal{R}} \star \overrightarrow{PO^{\mathcal{R}}}$, sachant bien sur que $\overrightarrow{PO^{\mathcal{R}}} = -\overrightarrow{OP^{\mathcal{R}}}$).

2 Mise en pratique

2.1 Fonctions OpenGL et pile de matrices

OpenGL fonctionne avec des matrices 4x4, et considère que les vecteurs sont des vecteur à 4 dimensions (dont la quatrième coordonnée vaut 1). Il met à disposition des piles de matrices que l'on utilise grâce aux fonctions GLLOADMATRIX, GLPOPMATRIX et GLPUSHMATRIX. Chaque pile de matrice possède un nom, et celle qu'openGL utilise pour transformer les positions des points s'appelle MODELVIEW.

GLLOADMATRIX remplace la matrice au sommet de la pile par la matrice donnée en argument.

GLPOPMATRIX incrémente le pointeur sur la pile (en se représentant la pile croissant vers les adresses inférieures).

GLPUSHMATRIX copie la matrice qui se trouve au sommet de la pile vers le sommet de la pile, et décrémente le pointeur sur la pile, ce qui a pour effet de faire une copie de la matrice sur la pile.

Lorsqu'on donne un vecteur à openGL, il le multiplie par la matrice qui se trouve au sommet de la pile MODELVIEW pour obtenir les coordonnées du vecteur "par rapport à la caméra", c'est à dire juste avant projection.

La bonne façon d'utiliser cette pile est donc la suivante :

Soit $\mathcal{C}$ le repère de la caméra, formé par l'origine C et les axes $\overrightarrow{C_X}$, $\overrightarrow{C_Y}$ et $\overrightarrow{C_Z}$. Le point C est la position de la caméra, $\overrightarrow{C_X}$, $\overrightarrow{C_Y}$ et $\overrightarrow{C_Z}$ sont les axes de la caméra (je vous rappelle que par défaut $\overrightarrow{C_X}$ pointe vers la droite, $\overrightarrow{C_Y}$ vers le haut, et $\overrightarrow{C_Z}$ vers l'arrière). Ces vecteurs sont connus dans le repère absolu de l'espace.

Soit $\mathcal{O}$ le repère d'un objet, formé de l'origine O et des vecteurs $\overrightarrow{X}$, $\overrightarrow{Y}$ et $\overrightarrow{Z}$, où O est la position de l'objet et $\overrightarrow{X}$, $\overrightarrow{Y}$ et $\overrightarrow{Z}$ les axes du repère de l'objet. Ces vecteurs sont eux aussi connus dans le repère absolu. Les points de l'objet sont, quant à eux, connus dans le repère $\mathcal{O}$.

Appelons $\mathcal{R}$ le repère absolu par rapport auquel les repères $\mathcal{C}$ et $\mathcal{O}$ sont définis (c'est le repère du "monde").

Les points de l'objet sont les vecteurs $\overrightarrow{V^{\mathcal{O}}}$, exprimés dans $\mathcal{O}$.

On cherche à avoir les vecteurs $\overrightarrow{V}$ dans le repère de la caméra. Donc, on doit calculer les $\overrightarrow{V^{\mathcal{C}}} = M_{4\times4}^{\mathcal{C}\leftarrow\mathcal{O}} \star \overrightarrow{V^{\mathcal{O}}}$.

Que l'on peut décomposer comme ceci :

$$\overrightarrow{V^{\mathcal{C}}} = M_{4\times4}^{\mathcal{C}\leftarrow\mathcal{R}} \star M_{4\times4}^{\mathcal{R}\leftarrow\mathcal{O}} \star \overrightarrow{V^{\mathcal{O}}}$$

pour n'utiliser que des matrices connues. En effet :

$M_{4\times4}^{\mathcal{R}\leftarrow\mathcal{O}}$ est donnée par la position de l'objet dans $\mathcal{R}$.

$M_{4\times4}^{\mathcal{C}\leftarrow\mathcal{R}}$ est l'inverse (la transposée) de la position de la caméra dans $\mathcal{R}$ (c'est à dire la position de $\mathcal{R}$ dans le repère de la caméra $\mathcal{C}$)

On commence donc par empiler $M_{4\times4}^{\mathcal{C}\leftarrow\mathcal{R}}$ dans MODELVIEW, puis on multiplie par $M_{4\times4}^{\mathcal{R}\leftarrow\mathcal{O}}$, et ensuite on envoi les $\overrightarrow{V^{\mathcal{O}}}$.

Dans le cas d'objets hiérarchisés, c'est à dire lorsqu'on a par exemple $\mathcal{O}_2$, $\mathcal{O}_3$ et $\mathcal{O}_4$ tous les trois définis dans $\mathcal{O}_1$, lui même défini dans $\mathcal{R}$, on fait :

1. GLLOAD $M_{4\times4}^{\mathcal{C}\leftarrow\mathcal{R}}$,
2. GLMULTIPLIE $M_{4\times4}^{\mathcal{R}\leftarrow\mathcal{O}_1}$,
3. GLPUSH pour sauver le résultat,
4. GLMULTIPLIE $M_{4\times4}^{\mathcal{O}_1\leftarrow\mathcal{O}_2}$,
5. Envoi des $\overrightarrow{V^{\mathcal{O}_2}}$,
6. GLPOP pour récupérer le résultat de $M_{4\times4}^{\mathcal{C}\leftarrow\mathcal{R}} \star M_{4\times4}^{\mathcal{R}\leftarrow\mathcal{O}_1}$,
7. GLPUSH pour sauvegarder à nouveau ce résultat,
8. GLMULTIPLIE $M_{4\times4}^{\mathcal{O}_1\leftarrow\mathcal{O}_3}$,
9. Envoi des $\overrightarrow{V^{\mathcal{O}_3}}$,
10. GLPOP pour récupérer le résultat de $M_{4\times4}^{\mathcal{C}\leftarrow\mathcal{R}} \star M_{4\times4}^{\mathcal{R}\leftarrow\mathcal{O}_1}$,
11. GLMULTIPLIE $M_{4\times4}^{\mathcal{O}_1\leftarrow\mathcal{O}_4}$,
12. Envoi des $\overrightarrow{V^{\mathcal{O}_4}}$.

On peut ainsi facilement gérer les objets organisés hiérarchiquement dans un arbre.

2.2 Structures de données

Pour utiliser la pile MODELVIEW de cette manière, je vous propose d'imaginer cette organisation des données. Ces structures seront reprises et employées pour résoudre les problèmes que l'on se posera dans la suite de ce document.

```
// structures

typedef struct {
  union {
    GLfloat c[4];
    struct {
      GLfloat x,y,z,u;
    } n;
  };
} Vecteur;

typedef struct {
  union {
    GLfloat c[16];
    struct {
      Vecteur x,y,z,t;
    } n;
  };
} Matrice;

typedef struct {
  Matrice position;
  Objet *pere;
  void *(affiche(void));
} Objet;

// 10 objets et la caméra

#define NBOBJETS 10
Objet objet[NBOBJETS];
Objet camera;
```

Où les coordonnées des vecteurs ou les matrices peuvent être accédées directement par le tableau `c` ou-bien nommément grâce aux structures `n`. Ainsi, par exemple, `matrice.n.z.n.y` désigne le même élément que `matrice.c[9]` ou `matrice.n.z.c[1]`. Attention : j'utilise ici des union non nommées, qui ne sont normalement pas valides en C même si certains compilateurs C les acceptent. Si vous voulez utiliser ces structures, compilez en C++, qui accepte les unions non nommées.

L'objet contient l'élément `pere` qui pointe sur l'objet parent de celui considéré, c'est à dire l'objet par rapport auquel la position de celui considéré est exprimée. Si la position de l'objet est relative au repère absolu, ce pointeur vaudra `NULL`.

La fonction pointée par `affiche` affiche l'objet lui même. Par exemple, cette fonction peut faire l'affaire si l'objet est un cube :

```
void afficheCube(void) {
```

```
  glBegin(GL_QUAD);
  glVectorf(-1,-1,-1);
  glVectorf(-1,-1, 1);
  glVectorf( 1,-1, 1);
  glVectorf( 1,-1,-1);
  glVectorf(-1, 1,-1);
  glVectorf(-1, 1, 1);
  glVectorf(-1,-1, 1);
  glVectorf(-1,-1,-1);
  ... etc ...
  glEnd();
}
```

Nous allons considérer dans la suite que l'objet caméra a sa position déjà inversée, c'est à dire que la partie rotation de la position de la caméra est transposée, et que le vecteur $\overrightarrow{T}$ de la position, plutôt que d'être la translation de C dans le repère absolu $\mathcal{R}$, est la position de l'origine de $\mathcal{R}$ dans le repère de la caméra $\mathcal{C}$. Le déplacement de la caméra se fait aussi aisément, parfois même plus aisément que si cette position n'était pas inversée, et cela élimine le besoin de transposer cette matrice lors de l'affichage.

D'autre part, la position de la caméra sera considérée pour l'instant comme exprimée par rapport au repère absolu (son père vaut `NULL`).

Nous aurons également besoin d'un certain nombre de fonctions, dont voici les déclarations (je ne donne pas toutes les implémentations) :

```
// copy v2 dans v1
void copy(Vecteur *v1, Vecteur *v2);
// renvoit m1*m2
void compose(Matrice *m1, Matrice *m2, Matrice *resultat);
// renvoit m*v
void multiplie(Matrice *m, Vecteur *v, Vecteur *resultat);
// renvoit v1.v2 (produit scalaire)
double scalaire(Vecteur *v1, Vecteur *v2) {
  return v1->c[0]*v2->c[0]+v1->c[1]*v2->c[1]+v1->c[2]*v2->c[2];
}
// renvoit la norme d'un vecteur
double norme(Vecteur *v) {
  return sqrt(scalaire(v,v));
}
// renorme un vecteur
void renorme(Vecteur *v) {
  double n=norme(v);
  if (0!=n) {
    n = 1/n;
    v->c[0] *= n;
    v->c[1] *= n;
    v->c[2] *= n;
  }
```

```
}
// renvoit le produit vectoriel
void vectoriel(Vecteur *v1, Vecteur *v2, Vecteur *resultat);
```

2.3 Problèmes

2.3.1 Rendre une liste d'objets sans architecture

Revenons sur ce problème déjà traité. Voici, avec nos nouvelles structures, une fonction qui affiche tous les objets si tous les objets sont relatifs au repère absolu.

```
void afficheTout() {
  int o;
  glMatrixMode(GL_MODELVIEW);
  glLoadMatrixf(camera.position.c);
  for (o=0; o<NBOBJETS; o++) {
    glPushMatrix();
    // Soit ce push et le pop qui suit,
    // Soit déplacer le LoadMatrix de dessus ici.
    // je préfère comme ceci car ca peut
    // réduire les transferts RAM->VRAM
    glMultMatrixf(objet[o].position.c);
    objet[o].affiche();
    glPopMatrix();
  }
}
```

2.3.2 Rendre une liste d'objets architecturés

Il s'agit du problème que nous avons traité dans l'introduction de cette section.

Si les objets sont susceptibles d'être relatifs à d'autres objets, on peut toujours les afficher très simplement. Restriction : Les relations de dépendance entre les objets doivent former un arbre, et la liste doit contenir ces objets dans l'ordre de parcours en profondeur de l'arbre (Cf figure 4)

La routine afficheTout deviens :

```
void afficheTout() {
  int o;
  glMatrixMode(GL_MODELVIEW);
  glLoadMatrixf(camera.position.c);
  initPile();
  pushPile(NULL);       // pour la camera
  for (o=0; o<NBOBJETS; o++) {
    while (topPile()!=objet[o].pere) {
      glPopMatrix();
```

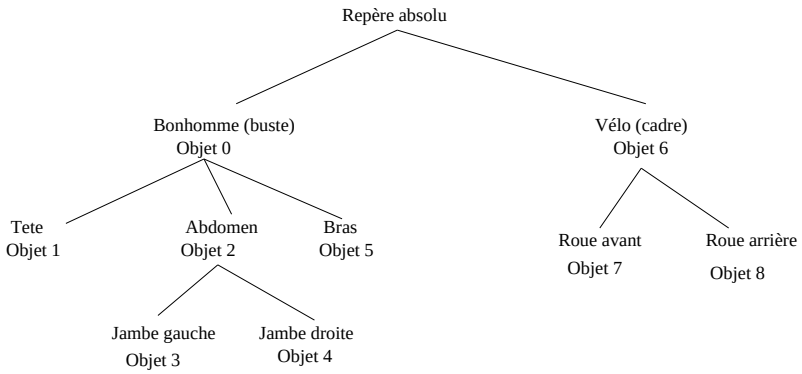

Figure 4: Parcours en profondeur

```
      popPile();
    }
    glPushMatrix();
    pushPile(objet+o);
    glMultMatrixf(objet[o].position.c);
    objet[o].affiche();
  }
}
```

Où les fonctions `initPile`, `pushPile`, `popPile` et `topPile` gèrent une pile de pointeurs sur objets, qui mémorise les références des objets dont la position dans le repère de la caméra est au sommet de la pile MODELVIEW.

Une implémentation possible serait :

```
Objet *(pile[NBOBJETS]);
int idxPile;

void initPile() {
  idxPile=0;
}
void pushPile(Objet *obj) {
  pile[idxPile++]=obj;
}
void popPile() {
  idxPile--;
}
Objet* topPile() {
  return pile[--idxPile];
}
```

2.3.3 Rotation d'un objet autour d'un axe de son repère

Généralement, les objets que l'on affiche doivent se déplacer d'une image à l'autre. Gérer les objets hiérarchisé facilite grandement le déplacement des objets. Dans la plupart des phénomènes, en effet, les mouvements des objets peuvent être réduit à des mouvements simples si on "accroche" les objets entres eux de la bonne façon (par exemple, dans le repère lié au cadre du vélo, la position de la roue se réduit à une simple rotation autour de son centre).

Un cas intéressant de déplacement est la rotation autour d'un axe. Si cet axe est constant dans le repère de l'objet, on peut ramener la rotation à une rotation autour d'un des vecteurs formant le repère. Prenons par exemple le vecteur $\overrightarrow{Z}$ du repère de l'objet (le troisième, les autres étant appelés $\overrightarrow{X}$ et $\overrightarrow{Y}$). Appelons $\mathcal{O}$ ce repère. Appelons $\mathcal{O}'$ ce même repère tourné d'un angle α autour de l'axe $\overrightarrow{Z}$ de $\mathcal{O}$. La trigonométrie nous donne les coordonnées par rapport à $\mathcal{O}$ des vecteurs $\overrightarrow{X'}$, $\overrightarrow{Y'}$ et $\overrightarrow{Z'}$ qui forment $\mathcal{O}'$:

$$\overrightarrow{X'^{\mathcal{O}}} = (\cos\alpha, \sin\alpha, 0)$$

$$\overrightarrow{Y'^{\mathcal{O}}} = (-\sin\alpha, \cos\alpha, 0)$$

$$\overrightarrow{Z'^{\mathcal{O}}} = (0, 0, 1)$$

Faire tourner les points de $\mathcal{O}$ autour de $\overrightarrow{Z}$ peut donc facilement s'obtenir en supposant que les points sont en fait définis dans $\mathcal{O}'$, lui même exprimé dans $\mathcal{O}$. Cela reviens, dans notre exemple, à dire que l'objet o' est relatif à l'objet o, et à donner à l'objet o la position initiale de l'objet que l'on veut faire tourner tandis que l'objet o' a comme position initiale l'identité, c'est à dire la matrice formée par des 0 partout et des 1 sur la diagonale, et plus tard lorsque α deviens non nul par la matrice trigonométrique exprimée tout à l'heure.

2.3.4 Translation d'un objet le long d'un vecteur constant

La translation le long d'un vecteur constant ne pose aucune difficulté. On fera attention au repère par rapport auquel est défini le vecteur de translation : il faut évidement exprimer ce vecteur dans le repère par rapport auquel est défini la position de l'objet que l'on veut translater avant d'ajouter ce vecteur à la colonne $\overrightarrow{T}$ de la position de l'objet.

2.3.5 Trouver un Vecteur perpendiculaire

Souvent, notamment pour construire un repère comme on le verra dans le paragraphe suivant, on a besoin de calculer un vecteur perpendiculaire à un vecteur donné $\overrightarrow{V}$.

Une méthode simple consiste à prendre pour commencer un vecteur $\overrightarrow{W}$ dont on est sûr qu'il n'est pas aligné avec $\overrightarrow{V}$, puis à lui retirer sa projection le long de $\overrightarrow{V}$. C'est à dire :

$$\overrightarrow{W} = vecteurpresquequelconque$$

$$\overrightarrow{W} -= (\overrightarrow{V} \bullet \overrightarrow{W}) \times \overrightarrow{V}$$

Ce vecteur $\overrightarrow{W}$ sera ainsi perpendiculaire à $\overrightarrow{V}$.

Voici géométriquement ce que cela donne :

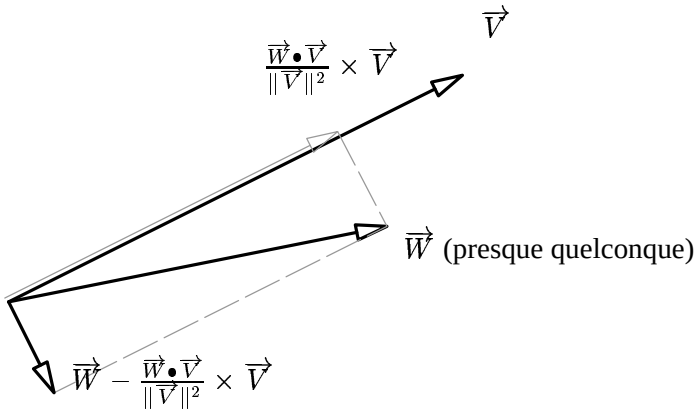

Figure 5: Construire un vecteur perpendiculaire

Dans le cas usuel $\overrightarrow{V}$ est normé, ce qui facilite encore le calcul.
Une implémentation possible pourrait être :

```
void perpendicularise(Vecteur *v, Vecteur *w) {
  // rend w perpendiculaire à v
  double s;
  int i;
  s = scalaire(w, v);  // Diviser ici par scalaire(v, v) si
  for (i=0; i<3; i++)  // on n'est pas certain que v soit normé.
    w->c[i] -= s*v->c[i];
}
```

2.3.6 Trouver deux vecteurs pour former un repère

Parfois, pour les besoins d'un calcul, on a besoin d'un repère orthonormé dont le vecteur $\overrightarrow{Z}$, par exemple, doit être aligné avec un vecteur $\overrightarrow{V}$ donné.

On commence par dire que $\overrightarrow{Z} = \overrightarrow{V}$. On peut ensuite obtenir un vecteur $\overrightarrow{X}$ perpendiculaire à $\overrightarrow{Z}$ avec la méthode ci-dessus. Ensuite, on doit renormer $\overrightarrow{X}$ et $\overrightarrow{Z}$. Puis, on obtient $\overrightarrow{Y}$ avec un produit vectoriel entre $\overrightarrow{Z}$ et $\overrightarrow{X}$.

Le produit vectoriel de deux vecteurs $\overrightarrow{V}$ et $\overrightarrow{W}$ se note $\overrightarrow{U} = \overrightarrow{V} \wedge \overrightarrow{W}$. Les coordonnées de $\overrightarrow{U}$ (U^x, U^y, U^z) s'obtiennent par les formules suivantes :

$$U^x = V^y \times W^z - V^z \times W^y$$
$$U^y = V^z \times W^x - V^x \times W^z$$
$$U^z = V^x \times W^y - V^y \times W^x$$

Les propriétés du produit vectoriel entre $\overrightarrow{V}$ et $\overrightarrow{W}$ sont de donner un vecteur perpendiculaire à la fois à $\overrightarrow{V}$ et à $\overrightarrow{W}$, et dont la norme est égale à la surface du triangle dont les côtés sont formés par $\overrightarrow{V}$ et $\overrightarrow{W}$ (c'est à dire que $\|\overrightarrow{U}\| = 0.5 \times \|\overrightarrow{V}\| \times \|\overrightarrow{W}\| \times \sin\alpha$ où α est l'angle entre $\overrightarrow{V}$ et $\overrightarrow{W}$). Le produit vectoriel est donc souvent employé pour trouver des vecteurs perpendiculaires à d'autres vecteurs.

Une implémentation possible :

```
void repere(Vecteur *x, Vecteur *y, Vecteur *z) {
  // Z contient le vecteur Z pas forcément normé
  Vecteur a = {{ 1,0,0 }};
  Vecteur b = {{ 0,1,0 }};
  // trouve un x non aligné avec z
  if (z->c[1]==0 && z->c[2]==0)
    copy(x, &b);
  else
    copy(x, &a);
  perpendicularise(z,x);
  renorme(x);
  renorme(z);
  vectoriel(z, x, y);
}
```

2.3.7 Déplacement de la caméra avec la souris

Considérons maintenant le déplacement de la caméra. Essayons de déplacer la caméra uniquement avec la souris, de la façon suivante : Lorsqu'on déplace la souris, le caméra se tourne vers la droite, la gauche, le haut ou le bas en fonction de la position du curseur, et lorsqu'on appuis sur un bouton la caméra avance droit devant elle.

Appelons `depX` et `depY` les déplacements en X et en Y de la souris. Le déplacement en X induit une rotation de $\mathcal{C}$ autour de son axe $\overrightarrow{C_Y}$, et le déplacement en Y une rotation de $\mathcal{C}$ autour de son axe $\overrightarrow{C_X}$.

On a vu que la position de la caméra serait toujours exprimée par rapport au repère absolu (ce n'est évidement pas une nécessité pour OpenGL que la première matrice poussée sur la pile soit celle de la caméra ; pour openGL, seule la matrice au sommet de la pile a une signification). Cette contrainte que nous nous sommes nous même posé nous empêche de résoudre cette difficulté en intercalant des objets entre la caméra et le repère absolu comme nous l'avons fait pour faire tourner un objet.

Nous devons donc nous même calculer la nouvelle position de la caméra avant de la pousser sur la pile ModelView. Il est de toute façon plus pratique de connaître la position de la caméra par rapport au repère absolu, et poser des positions intermédiaires entre les deux nous aurait compliqué la vie dans la suite.

Pour faire tourner la caméra autour de son axe $\overrightarrow{CX}$, la trigonométrie nous indique qu'il faut multiplier la position de la caméra par la matrice 4x4 suivante (qui ne s'attaquera qu'à la partie rotation de la position), l'angle α étant proportionel à `depY` :

$$\begin{array}{c} \\ \overrightarrow{C_X} \\ \overrightarrow{C_Y} \\ \overrightarrow{C_Z} \\ \overrightarrow{?} \end{array} \begin{array}{c} \begin{array}{cccc} \overrightarrow{C_{X'}} & \overrightarrow{C_{Y'}} & \overrightarrow{C_{Z'}} & \overrightarrow{T} \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

Mais attention : nous ne stoquons pas dans caméra la position absolue de la caméra, mais la transposée de cette position, c'est à dire la position de l'origine du repère absolu dans le repère de la caméra.

Le problème se pose une fois de plus pour la partie translation de la position (pour la partie rotation, il suffit de transposer). Si la caméra tourne sur elle même, la position de l'origine du repère de la caméra dans le repère absolu ne change pas, mais il est clair que la position de l'origine du repère absolu dans le repère de la caméra, elle, change. Nous somme dans un cas où il est plus difficile de stoquer l'inverse de la position de la caméra que sa position.

Si les vecteurs formant le repère de la caméra tournent d'un angle α autour du vecteur $\overrightarrow{C_X}$ de $\mathcal{C}$ dans un sens, alors l'origine W du repère absolu $\mathcal{R}$ tourne d'un angle α dans l'autre sens. Il faut donc multiplier la position $\overrightarrow{T}$ de ce point, présente dans la quatrième colonne de la matrice représentant le repère $\mathcal{C}$, par la matrice donnée à l'instant avec un angle α opposé.

```
void tourneX(Matrice *m, double angle) {
  double c = cos(angle);
  double s = sin(angle);
  // les 6 coeficient de la matrice
  // qui vont changer
  double coef[6];
  coef[0] = c*m->c[4] -s*m->c[8];
  coef[1] = c*m->c[5] -s*m->c[9];
  coef[2] = c*m->c[6] -s*m->c[10];
  coef[3] = s*m->c[4] +c*m->c[8];
  coef[4] = s*m->c[5] +c*m->c[9];
  coef[5] = s*m->c[6] +c*m->c[10];
  m->c[4] = coef[0];
  m->c[5] = coef[1];
  m->c[6] = coef[2];
  m->c[8] = coef[3];
  m->c[9] = coef[4];
  m->c[10] = coef[5];
  // maintenant la translation
  // (seuls x et y changent)
  coef[0] =-c*m->c[13] -s*m->c[14];
  coef[1] = s*m->c[13] -c*m->c[14];
  m->c[13] = coef[0];
```

```
  m->c[14] = coef[1];
}
```

Après ces calculs, la caméra aura donc tournée autours de son vecteur $\overrightarrow{C_X}$ pointant vers la droite d'un angle proportionnel à `depY`. Il faut faire la même oppération avec un angle β (fonction de `depX`) autours de $\overrightarrow{C_Y}$ qui pointe vers le haut.

Mais attention : tous ces calculs auront pu propager des erreur d'arrondis dans la position, et à force de les exécuter, il se peut qu'au bout d'un certain temps (très long dans la pratique), la matrice représentant $\mathcal{C}$ se déforme de façon visible. Il faut donc, de temps en temps, la renormaliser avant que ces déformations ne soient visibles.

Renormaliser la matrice donnant la position de la caméra consiste ici à s'assurer que les vecteurs $\overrightarrow{C_X}$, $\overrightarrow{C_Y}$ et $\overrightarrow{C_Z}$ sont bien de norme 1 et orthogonaux entre eux. Pour se faire, on peut procéder comme ceci : d'abord, renormer $\overrightarrow{C_X}$. Ensuite, réorthogonaliser $\overrightarrow{C_Y}$ et le renormer. Finalement, recalculer $\overrightarrow{C_Z}$ avec un produit vectoriel : $\overrightarrow{C_Z} = \overrightarrow{C_X} \wedge \overrightarrow{C_Y}$.

Réorthogonaliser $\overrightarrow{C_Y}$ consiste à retirer de $\overrightarrow{C_Y}$ sa composante suivant $\overrightarrow{C_X}$, qui doit être nulle puisque $\overrightarrow{C_X}$ et $\overrightarrow{C_Y}$ sont censés être perpendiculaires. On a déjà vu comment faire : $\overrightarrow{C_Y} -= (\overrightarrow{C_Y} \bullet \overrightarrow{C_X}) \times \overrightarrow{C_X}$. Ne pas oublier de renormer $\overrightarrow{C_Y}$ après cette opération.

On aurrait pu effectuer ces opérations dans un autre sens mais il est bon que la plus grande opération soit effectuée sur $\overrightarrow{C_Z}$, puisque c'est suivant Z que les changements sont le moins visibles (et ensuite en général, le long de $\overrightarrow{C_Y}$, car il y a souvent moins de résolution en Y qu'en X dans l'image générée).

```
void renormalise(Matrice *m) {
  renorme(&m->n.x);
  perpendicularise(&m->n.x, &m->n.y);
  renorme(&m->n.y);
  vectoriel(&m->n.x, &m->n.y, &m->n.z);
}
```

Ensuite, occupons nous de faire avancer la caméra lorsqu'un bouton de la souris est pressé.

Pour faire avancer un objet le long d'un de ses axes, on a vu qu'il suffit d'ajouter un vecteur constant, ici l'axe voulu du repère de l'objet considéré, au vecteur $\overrightarrow{T}$ de sa position. Pour la caméra, dont on ne connait que l'inverse de la position, les choses sont différentes, et encore plus simple.

Faire avancer la caméra droit devant consiste à réduire la coordonnée le long de $\overrightarrow{C_Z}$ de l'origine de $\mathcal{R}$. Il suffit donc de retirer à la coordonnées z de $\overrightarrow{C_T}$ une valeur proportionnelle à la vitesse du déplacement de la caméra :

```
void avance(double vitesse) {
  camera.position.c[14] -= vitesse;
}
```

2.3.8 Calcul de la position du centre d'un objet dans le repère de la caméra

On a vu jusqu'à maintenant comment envoyer les points d'un objet à openGL pour que leur positions soient correctement calculée par l'API. Mais souvent, indépendamant de l'affichage, on a besoin de connaitre la position du centre d'un objet dans le repère de la caméra, notament pour déterminer avant de l'afficher s'il est visible ou pas.

Mettons que le centre de l'objet ne soit pas à l'origine du repère de l'objet (par exemple, si l'objet est articulé autours d'un point précis, il est plus utile de mettre le centre du repère sur ce point d'articulation plutôt que su le centre de l'objet pour faciliter son mouvement). On se donne donc le vecteur `centre` dans la structure `Objet` qui donne la position du centre de l'objet dans le repère local de l'objet (ce vecteur est donc constant, calculé par exemple une fois pour toute au début du programme). On rajoute aussi le rayon de la sphère englobante, dont le centre est le centre de l'objet, qui sera utile par la suite.

```
typedef struct {
  Matrice position;
  Vecteur centre;
  double rayon;
  Objet *pere;
  void *(affiche(void));
} Objet;
```

Question : où ce trouve le centre dans le repère de la caméra ?

Soit $\mathcal{O}_2$ le repère de l'objet considéré, lui même défini dans $\mathcal{O}_1$, lui même défini dans le repère absolu $\mathcal{R}$, et $\mathcal{C}$ le repère de la caméra défini dans $\mathcal{R}$. Le vecteur $\overrightarrow{Centre}$ est donc connu dans $\prime_2$. Le calcul à effectuer, en n'utilisant que des matrices connues, est donc :

$$M^{\mathcal{C}\leftarrow\mathcal{R}} \star M^{\mathcal{R}\leftarrow\mathcal{O}_1} \star M^{\prime_1\leftarrow\mathcal{O}_2} \star \overrightarrow{Centre^{\mathcal{O}_2}} = \overrightarrow{Centre^{\mathcal{C}}}$$

On peut généralement calculer ce centre juste avant l'affichage de l'objet, pour tous les objets. Dans ce cas, on n'est pas obligé de calculer soit même les compositions de matrices puisqu'elles sont faites dans MODELVIEW.

Deux solutions donc : la plus simple est de lire la matrice MODELVIEW et de la multiplier par le vecteur $\overrightarrow{Centre^{\mathcal{O}_2}}$. L'autre méthode consiste à calculer soit meme toutes les compositions de matrices et à donner les résultats à OpenGL (ne plus faire alors que des `glLoadMatrixf` et plus de `glPushMatrix`, `glMultMatrixf`, etc...

En fonction des cas et des implémentations d'OpenGL, le plus rapide peut être soit l'un soit l'autre.

Nous allons écrire ici la première méthode.

donc dans la fonction `afficheTout` :

```
void afficheTout() {
  int o;
```

```
  Matrice temp;
  Vecteur centreOdansC;
  glMatrixMode(GL_MODELVIEW);
  glLoadMatrixf(camera.position.c);
  initPile();
  pushPile(NULL);       // pour la camera
  for (o=0; o<NBOBJETS; o++) {
    while (topPile()!=objet[o].pere) {
      glPopMatrix();
      popPile();
    }
    glPushMatrix();
    pushPile(objet+o);
    glMultMatrixf(objet[o].position.c);
    glGetFloatv(GL_MODELVIEW_MATRIX,temp.c);
    multiplie(&temp, &objet[o].centre, &centreOdansC);
    /* Ici on a le centre de l'objet exprimé dans le
     * repère de la caméra. On peut par exemple s'en
     * servir comme ca pour éliminer les objets qui
     * sont entièrement dans le dos de la caméra :
    */
    if (centre0dansC.n.z<objet[o].rayon) {
      objet[o].affiche();
    }
  }
}
```

En fait, dans la plupart des cas il vaut mieux se garder de récuperer la matrice MODELVIEW aussi souvent ; En effet, sur les cartes vidéos qui font elles mêmes les transformations des points, cette lecture nécessite de ramener les données de la carte elle même et interrompt son travail le temps du transfert. Le temps ainsi perdu peut être suppérieur au temps qu'il aurait fallu pour calculer soit même la matrice.

En fait, dans le cas le plus général, on n'a pas nécéssairement besoin de connaitre le centre de tous les objets : par exemple, on se contentera de déterminer la visibilité du vélo avant d'afficher tous les objets composants le vélos dans leur repères propres ; on a donc deux arborescences de repères : des repères à calculer soit même dont on a besoin de connaitre la position (dans le repère absolu et/ou dans le repère de la caméra, en fonction des besoins), et les repères de MODELVIEW utiles à l'affichages, plus nombreux.

2.3.9 Frustum culling d'une sphère englobante

Nous allons utiliser maintenant la sphère englobante d'un objet et les paramètres de projection pour déterminer simplement si un objet est au moins partielement inclus dans la zone de vision ou pas (auquel cas il n'est pas nécessaire d'envoyer la géométrie de l'objet à openGL).

La zone de vision est définie avec `glFrustum(x1,y1, x2,y2, z1,z2)` en

mode `GL_PROJECTION`. Les paramètres sont tels que les coordonnées $(x1, y1, -z1)$ seront projetées dans le coin bas-gauche du viewport et $(x2, y2, -z1)$ dans le coint haut-droit. Quant à $(z1, z2)$ ils fournissent les éloignements minimum et maximum qui seront visibles. Ces six paramètres définissent donc une pyramide tronquée qui est la zone de vision.

Connaissant la position du centre d'une sphère englobante et son rayon, et les paramètres du frustum, on peut déterminer si une sphère est complètement en dehors de cette pyramide tronquée. Soit le vecteur `pos` la position du centre et `rayon` le rayon de la sphère. Les deux premières conditions à remplir pour être au moins en partie dans le frustrum sont : `pos.z-rayon>=-frustumZ1` et `pos.z+rayon<=-frustumZ2` ; sinon, c'est que l'objet est complètement dans le dos de la caméra, oubien trop loin hors du z-buffer.

Les conditions sur les bords sont plus pointues et nécessitent un petit dessin (Cf figure 6).

On a représenté sur ce dessin une vue de haut du repère de la caméra et du bord gauche du frustum, ainsi que la sphère englobante d'un objet dont le centre se situe en $(posX, posY)$.

En n'utilisant que des distances algébriques, ce qui implique d'utiliser $-frustumZ1$ au lieu de $frustumZ1$ pour replacer cette longeur dans le repère de la caméra, on voit sur ce dessin que :

1. d, la distance en X entre le bord gauche du frustum et le centre de la sphère englobante, vaut $posX - X$ où X est la demi-largeur du frustum pour ce Z, c'est à dire
$$X = posZ \times \frac{frustumX1}{-frustumZ1}$$
2. h, la distance entre la sphère englobante et le bord gauche du frustum, vaut, par similarité avec le triangle formé par $\overrightarrow{frustumX1}$ et $\overrightarrow{-frustumZ1}$:
$$\frac{-d}{\sqrt{1 + \frac{frustumX1^2}{frustumZ1^2}}}$$

On a donc
$$h = -\frac{posX + posZ \times \frac{frustumX1}{frustumZ1}}{\sqrt{1 + \frac{frustumX1^2}{frustumZ1^2}}}$$

Il faut comparer cette distance h avec le rayon de la sphère englobante pour savoir si la sphère est intégralement à gauche ou non du frustum.

Refaire des tests similaires avec les bords droit, haut et bas du frustum.

Pour calculer ces distances, on fera bien de précalculer, à chaque intitialisation du frustum, les constantes
$$K1_frustumGauche = \frac{frustumX1}{frustumZ1}$$
et
$$K2_frustumGauche = \sqrt{1 + Kfrustum Gauche^2}$$
pour le bord gauche, etc...

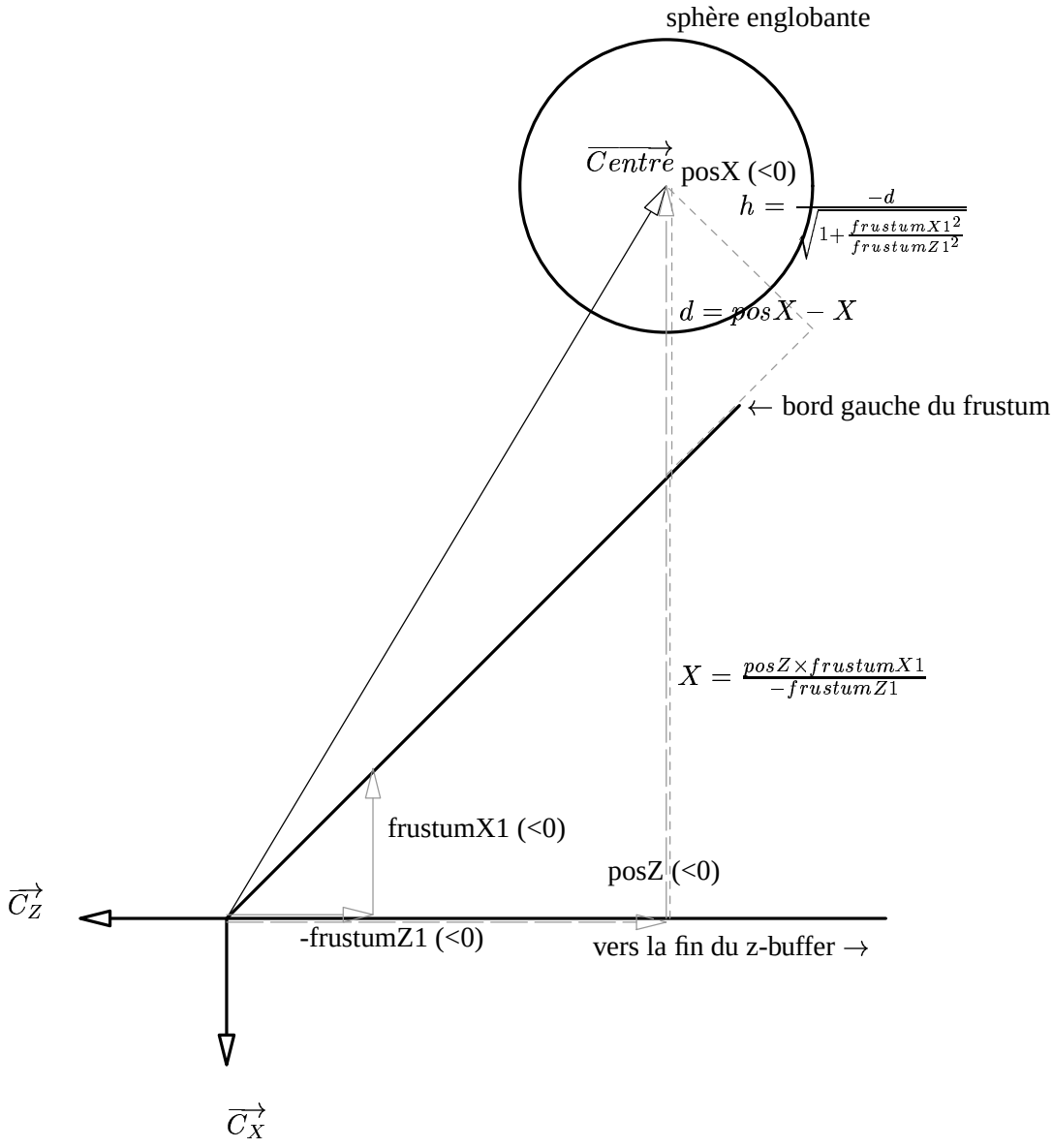

Figure 6: Distance entre une sphère et le frustum