Chapitre 3

La visualisation

3.1 Introduction

Nous présentons dans ce chapitre les opérations mises en œuvre pour obtenir une représentation 2D d'un modèle 3D. Cet exposé n'est pas un cours détaillé mais à pour but d'expliquer les outils utilisés par la librairie OpenGL lors du calcul de la représentation des scènes. Pour un exposé plus général, le lecteur est invité à se reporter, par exemple, à [LR88].

OpenGL représente chaque sommet du monde virtuel dans un espace à quatre dimensions : $s = (x, y, z, w)$. Afin d'obtenir les coordonnées «écran» de ce sommet, les opérations suivantes sont effectuées :

$\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$	$\longrightarrow$	Multiplication (à gauche) par la matrice modèle/vue	Positionne l'objet par rapport aux autres objets et au point de vue de l'observateur
	$\longrightarrow$	Multiplication (à gauche) par la matrice de projection	Transformation permettant un rendu réaliste, correspond à la projection de l'objet sur un plan
	$\longrightarrow$	Division par w	Permet d'obtenir les coordonnées du point dans l'espace à trois dimensions
	$\longrightarrow$	Transformation associée au périphérique de vue	Permet d'obtenir les coordonnées écran de l'objet

3.2 La matrice modèle/vue

C'est une matrice 4×4 de transformation M_{MV} appliquée à chaque point de l'espace. Elle permet de placer l'objet par rapport aux autres (matrice du modèle) et par rapport à l'observateur (matrice de vue).

$$s \longrightarrow M_{MV}.s$$

La matrice M_{MV} est donc la matrice résultant de différentes transformations.

En général, cette matrice est le résultat de la composition de translations, rotations et mises à l'échelle.

3.3 La matrice de projection

Permet de spécifier la projection utilisée pour représenter la scène en deux dimensions.

Nous présentons ici deux types de projections :
- La projection orthogonale
- La projection en perspective

Il peut sembler réducteur de se limiter à ces seules projections, cependant la plupart des autres projections permettant le rendu de scènes en 3D n'ont été introduites que pour faciliter la tâche du dessinateur, comme nous nous servons d'une machine pour effectuer les calculs, ce type de considérations n'est plus à prendre en compte.

3.3.1 La projection orthogonale

Nous nous plaçons dans le cas d'une projection parallèle à l'axe $[Oz)$ sur un plan d'équation $z = 0$ (Fig. 3.1). Le lecteur pourra aisément en déduire la matrice résultant d'une projection parallèle quelconque.

Dans ce cas, l'image du point de coordonnées (homogènes) $S = \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$ est donc $P = \begin{pmatrix} x \\ y \\ 0 \\ w \end{pmatrix}$.

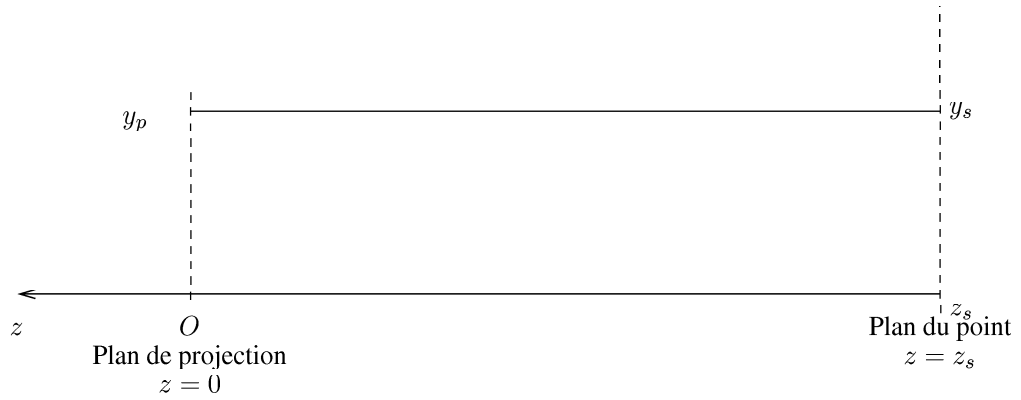

FIG. 3.1 – *Projection orthogonale*

La matrice de projection est donc

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Cependant, afin de conserver une pseudo-profondeur pour pouvoir discerner deux points de même projection (ce qui peut être utile dans la gestion des faces cachées, par exemple), on utilise plutôt la matrice :

$$\mathcal{M}_{ortho} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \mathcal{I}$$

Ce type de projection, qui à l'avantage d'être extrêmement simple (en fait on n'effectue aucun calcul), à l'inconvénient de ne pas refléter la scène telle qu'un humain la verrait.

3.3.2 La projection perspective

Elle permet un rendu plus réaliste que la projection orthogonale : plus un objet est loin, plus il est petit. Il faut ici définir la position de l'observateur en plus du plan de projection. Nous considérerons par la suite que l'observateur est placé en $(0, 0, 0)$ et que la projection s'effectue sur un plan parallèle au plan (O, x, y) et passant par le point $z = -near$ (ceci afin de prendre les mêmes conventions qu'en OpenGL).

Pour obtenir la projection s_p d'un point $s = (x_s, y_s, z_s)$ on calcule l'intersection entre la droite (O, s) et le plan d'équation $z = -near$ (Fig. 3.2).

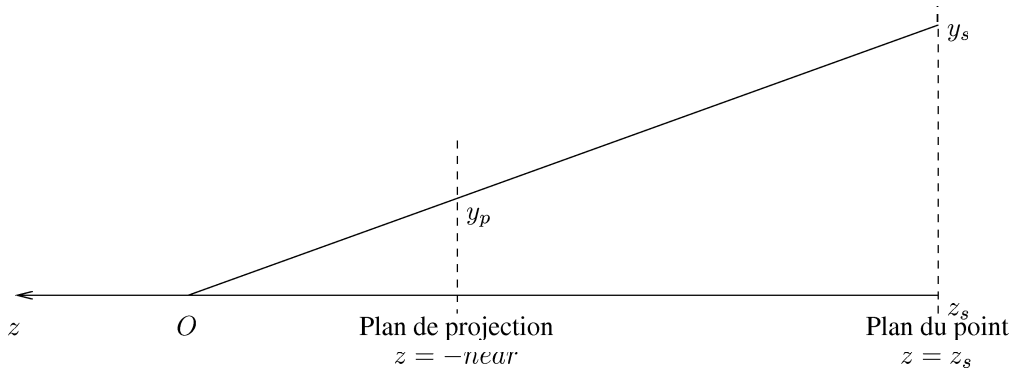

FIG. 3.2 – *Projection en perspective*

On a

$$\frac{y_p}{y_s} = \frac{near}{-z_s}$$

d'où

$$y_p = \frac{near}{-z_s} y_s$$

De même,

$$x_p = \frac{near}{-z_s} x_s \text{ et } z_p = -near.$$

On vérifie alors que la matrice de projection $\mathcal{M}$ est

$$\mathcal{M} = \begin{pmatrix} near & 0 & 0 & 0 \\ 0 & near & 0 & 0 \\ 0 & 0 & near & 0 \\ 0 & 0 & -1 & 0 \end{pmatrix}.$$

En effet, on a, en coordonnées homogènes :

$$\mathcal{M}.\begin{pmatrix} wx_s \\ wy_s \\ wz_s \\ w \end{pmatrix} = \begin{pmatrix} near.wx_s \\ near.wy_s \\ near.wz_s \\ -wz_s \end{pmatrix}$$

et donc

$$x_p = \frac{near.wx_s}{-wz_s}, y_p = \frac{near.wy_s}{-wz_s} \text{ et } z_p = \frac{near.wz_s}{-wz_s}$$

d'où

$$x_p = -\frac{near.x_s}{z_s}, y_p = -\frac{near.y_s}{z_s} \text{ et } z_p = -near.$$

Cependant, comme pour la projection orthogonale, on souhaite conserver une pseudo-profondeur. On cherche donc à obtenir z_p sous une forme similaire à celle de x_p et y_p :

$$z_p = -\frac{az_s + b}{z_s}.$$

Afin de fixer a et b, on choisit $z_p = -1$ pour $z_s = -near$ et $z_p = 1$ pour $z_s = -far$ (où far est telle que $near < far$, il correspondra à la face arrière du volume de vue). On a alors

$$\begin{cases} \frac{-a.near+b}{near} = -1 \\ \frac{-a.far+b}{far} = 1 \end{cases} \iff \begin{cases} a = \frac{near+far}{near-far} \\ b = \frac{2near.far}{near-far} \end{cases}$$

La matrice de projection devient alors :

$$\mathcal{M}_{persp} = \begin{pmatrix} near & 0 & 0 & 0 \\ 0 & near & 0 & 0 \\ 0 & 0 & \frac{near+far}{near-far} & \frac{2near.far}{near-far} \\ 0 & 0 & -1 & 0 \end{pmatrix}.$$

Remarque 3.3.1 On a effectué les calculs en considérant $w = 1$, on peut vérifier que la matrice donne le résultat souhaité pour $\frac{z_s}{w} = -near$ et $\frac{z_s}{w} = -far$.

3.3.3 Volume de vue

L'affichage de la scène se fait sur un périphérique de dimensions finies. On doit donc définir quelle portion du plan de projection sera à afficher. Nous choisirons ici une surface rectangulaire. De plus on impose que seuls les objets dont la profondeur est comprise dans un certain intervalle seront représentés, on obtient alors un volume contenant l'ensemble des objets à représenter : c'est le **volume de vue**.

Volume de vue pour la projection orthogonale

Le volume est un quadrilatère (Fig. 3.3), il est défini par six plans d'équations $z = -near$, $z = -far$, $x = left$, $x = right$, $y = bottom$ et $y = top$.

Volume de vue pour la projection perspective

Le volume est un tétraèdre tronqué (Fig. 3.4), il est défini par six plans d'équations $z = -near$ (le plan de projection), $z = -far$, $x = -\frac{left}{near}z$, $x = -\frac{right}{near}z$, $y = -\frac{bottom}{near}z$ et $y = -\frac{top}{near}z$.

3.3.4 Normalisation

Afin de simplifier les algorithmes (découpage, élimination des faces cachées, traçage, remplissage), on normalise le volume de vue, c'est-à-dire qu'on effectue une mise à l'échelle de façon à obtenir $x, y, z \in [-1, 1]$ pour tout point à l'intérieur du volume de vue. Cette normalisation s'effectue après la projection (nous avons vue que les matrices utilisées permettent de conserver une pseudo-profondeur).

3.3.5 Normalisation pour la vue orthogonale

On rappelle que $\mathcal{M}_{ortho} = \mathcal{I}$ et que le volume de vue est délimité par les plans d'équations $z = -near$, $z = -far$, $x = left$, $x = right$, $y = bottom$ et $y = top$.

On effectue tout d'abord une translation afin de centrer le volume autour du point O :

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

Les plans délimitant le volume de vue sont alors $z = -\frac{near-far}{2}$, $z = \frac{near-far}{2}$, $x = -\frac{right-left}{2}$, $x = \frac{right-left}{2}$, $y = -\frac{top-bottom}{2}$ et $y = \frac{top-bottom}{2}$.

On effectue ensuite la mise à l'échelle :

$$\begin{pmatrix} \frac{2}{right-left} & 0 & 0 & 0 \\ 0 & \frac{2}{top-bottom} & 0 & 0 \\ 0 & 0 & \frac{2}{near-far} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

On obtient donc

$$\mathcal{M}_{ortho,norm} = \begin{pmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & \frac{2}{near-far} & \frac{near+far}{near-far} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Le point $(left, bottom, -near, 1)$ est donc projeté en $(-1, -1, -1, 1)$ et le point $(right, top, -far, 1)$ est projeté en $(1, 1, 1, 1)$.

Remarque 3.3.2 Les spécifications OpenGL 1.1 sont fausses (rectifiées dans la 1.3) !!! En effet il manque des signes «-» dans la matrice de projection associée à une projection orthogonale.

3.3.6 Normalisation pour la vue perspective

On rappelle que

$$\mathcal{M}_{persp} = \begin{pmatrix} near & 0 & 0 & 0 \\ 0 & near & 0 & 0 \\ 0 & 0 & \frac{near+far}{near-far} & \frac{2near.far}{near-far} \\ 0 & 0 & -1 & 0 \end{pmatrix}.$$

Lors de cette projection, on a déjà $z \in [-1, 1]$. Par contre, en coordonnées cartésiennes, on a $x \in [left, right]$ et $y \in [bottom, top]$. Pour se ramener autour du point O, il faut donc effectuer la translation :

$$\begin{pmatrix} 1 & 0 & 0 & -\frac{right+left}{2} \\ 0 & 1 & 0 & -\frac{top+bottom}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

On normalise ensuite les abscisses et les ordonnées :

$$\begin{pmatrix} \frac{2}{right-left} & 0 & 0 & 0 \\ 0 & \frac{2}{top-bottom} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

On obtient donc :

$$\mathcal{M}_{persp,norm} = \begin{pmatrix} \frac{2near}{right-left} & 0 & \frac{right+left}{right-left} & 0 \\ 0 & \frac{2near}{top-bottom} & \frac{top+bottom}{top-bottom} & 0 \\ 0 & 0 & \frac{near+far}{near-far} & \frac{2near.far}{near-far} \\ 0 & 0 & -1 & 0 \end{pmatrix}.$$

Le point $(left, bottom, -near, 1)$ est donc projeté en $(-near, -near, -near, near)$, soit $(-1, -1, -1)$ en coordonnées cartésiennes.

Nous laissons au lecteur la détermination du point qui est projeté en $(1, 1, 1)$.

3.4 Le périphérique de vue

Une fois la projection effectuée, il faut transformer les coordonnées de chaque point en coordonnées «fenêtre».

Transformation :

$$\begin{array}{ccc} \text{Fenêtre virtuelle} & \longrightarrow & \text{Fenêtre écran} \\ [left, right] \times [bottom, top] & & [x_{min}, x_{max}] \times [y_{min}, y_{max}] \end{array}$$

– Normalisation.

On effectue une transformation sur chacunes des fenêtres afin d'obtenir un même système de coordonnées :

$$(x, y) \mapsto \left(\frac{x - left}{right - left}, \frac{y - bottom}{top - bottom} \right)$$

$$(i, j) \mapsto \left(\frac{i - x_{min}}{x_{max} - x_{min}}, \frac{j - y_{min}}{y_{max} - y_{min}} \right)$$

On se ramène ainsi à des coordonnées comprises entre 0 et 1.

– Affichage.
Les coordonnées normalisées doivent être identiques, il faut donc

$$i = x_{min} + \frac{x - left}{right - left}(x_{max} - x_{min})$$

$$j = y_{min} + \frac{y - left}{top - bottom}(y_{max} - y_{min})$$

Ainsi, si on est parti d'un volume de vue normalisé, on obtient :

$$i = x_{min} + \frac{x + 1}{2}(x_{max} - x_{min})$$

$$j = y_{min} + \frac{y + 1}{2}(y_{max} - y_{min})$$

Le rapport d'aspect est respecté (un cercle ressemblera à un cercle et non un ovale) si on a :

$$\frac{x_{max} - x_{min}}{y_{max} - y_{min}} = \frac{right - left}{top - bottom}.$$

3.5 OpenGL

3.5.1 Modèle - Vue

Lorsqu'on veut travailler sur la matrice modèle/vue on doit tout d'abord spécifier que les transformations qui vont suivre s'appliquent à celle-ci :

```
void glMatrixMode(GL_MODELVIEW);
```

On peut ensuite initialiser la matrice :

```
void glLoadIdentity();
```

Enfin, on peut spécifier les transformations, il faut savoir qu'OpenGL multiplie la matrice de transformation à droite de la matrice modèle/vue. Ainsi, si on veut faire subir à un objet une translation selon la direction (Ox) puis une rotation autour de (Oy) on écrira par exemple :

```
glRotatei(45, 0, 1, 0);
glTranslatei(1, 0, 0);
```

3.5.2 Projections

Avant de définir le type de projection, il faut activer la matrice de projection :

```
void glMatrixMode(GL_PROJECTION);
```

OpenGL permet de définir les projections en perspectives ou orthogonales :

```
void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble
top, GLdouble near, GLdouble far);
```

Pour la projection orthogonale, les paramètres sont les mêmes que ceux vus en cours.

```
void glFrustum(GLdouble left, GLdouble right, GLdouble bottom, GLdouble
top, GLdouble near, GLdouble far);
```

Pour la projection perspective.

La GLU fournit une autre fonction pour définir une vue en perspective :

```
void gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble near,
GLouble far);
```

où `fovy` est l'angle entre le plan bas du plan haut. `aspect` est le rapport largeur/hauteur de la fenêtre de projection (Fig 3.5).

Par défaut, l'observateur est placé en O et regarde vers les z négatifs. Cependant, la GLU permet de déplacer cet observateur grâce à la fonction (Fig. 3.6) :

```
void gluLookAt(GLdouble eyex, GLdouble eyey, GLdouble eyez, GLdouble
centerx, GLdouble centery, GLdouble centerz, GLdouble upx, GLdouble
upy, GLdouble upz);
```

$(eyex, eyey, eyez)$ indique la position de l'observateur.
$(centerx, centery, centerz)$ est un point se trouvant sur la ligne de vue.
(upx, upy, upz) indique la direction bas-haut du volume de vue.

Les projections étant toutes par rapport à un observateur en O, cette fonction définit en fait les translations et rotations nécessaires pour placer les objets à la bonne position par rapport à l'observateur. Elle doit donc s'appliquer à la matrice modèle/vue :

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(eyex, eyey, eyez, centerx, centery, centerz, upx, upy, upz);
```

3.5.3 Écran de visualisation

On peut définir la portion de la fenêtre dans laquelle sera dessinée la scène :

```
void glViewport(GLint x, GLint y, GLsizei width, GLsizei height);
```

Définit un rectangle de la fenêtre dans lequel sera représenté la scène (Fig. 3.7). `x`,`y` sont les coordonnées du pixel correspondant au bord inférieur gauche du rectangle (le pixel de coordonnées 0,0 est celui en bas, à gauche de la fenêtre). `width` et `height` correspondant quant-à eux à la largeur et la hauteur du rectangle.

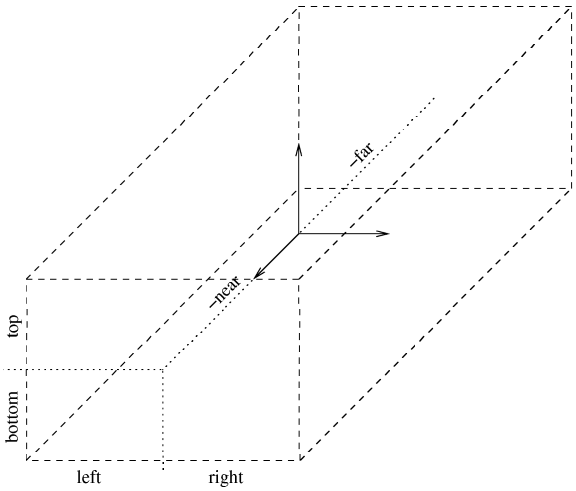

FIG. 3.3 – *Projection orthogonale*

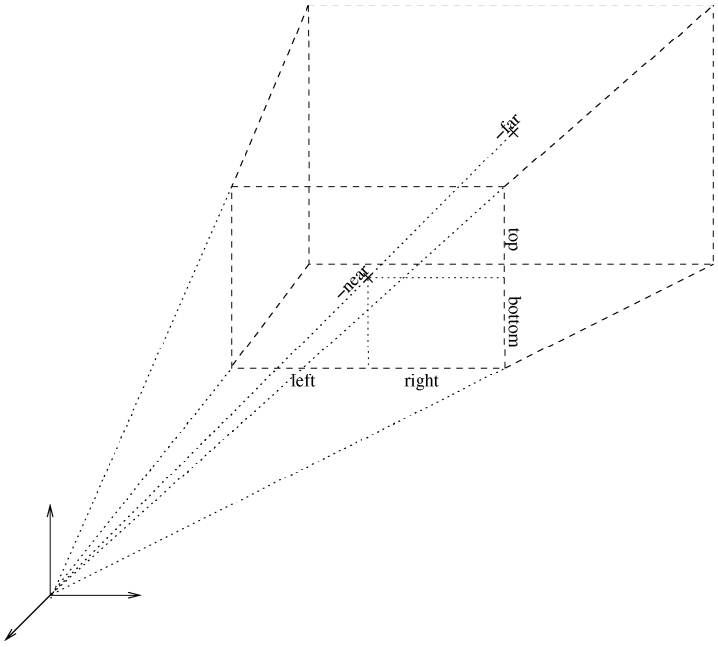

FIG. 3.4 – *Projection perspective*

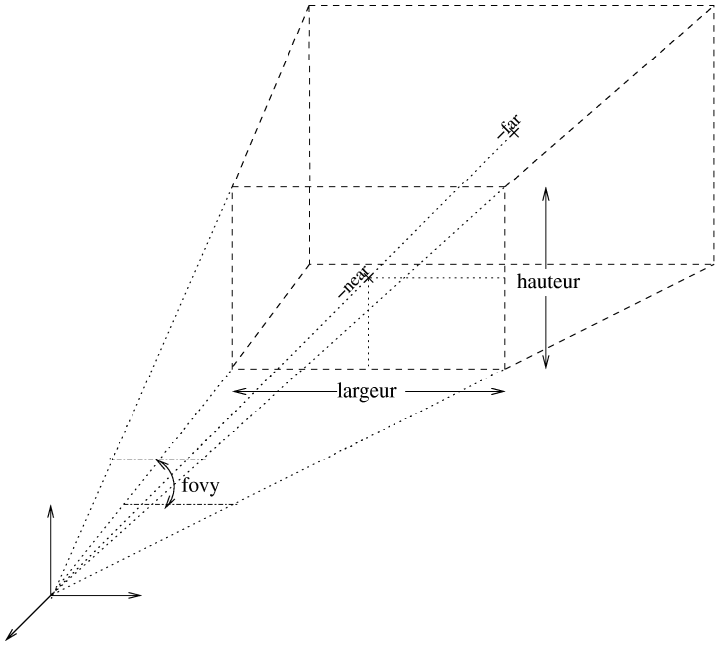

FIG. 3.5 – *La fonction gluPerspective*

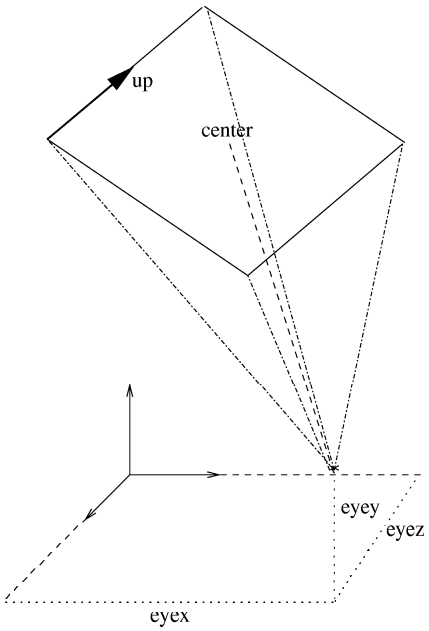

FIG. 3.6 – *La fonction gluLookAt*

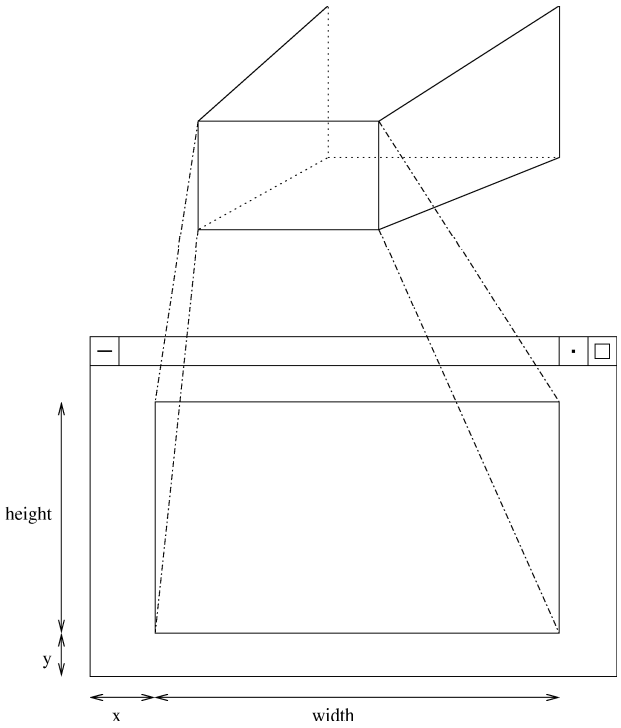

FIG. 3.7 – *Définition du viewport*