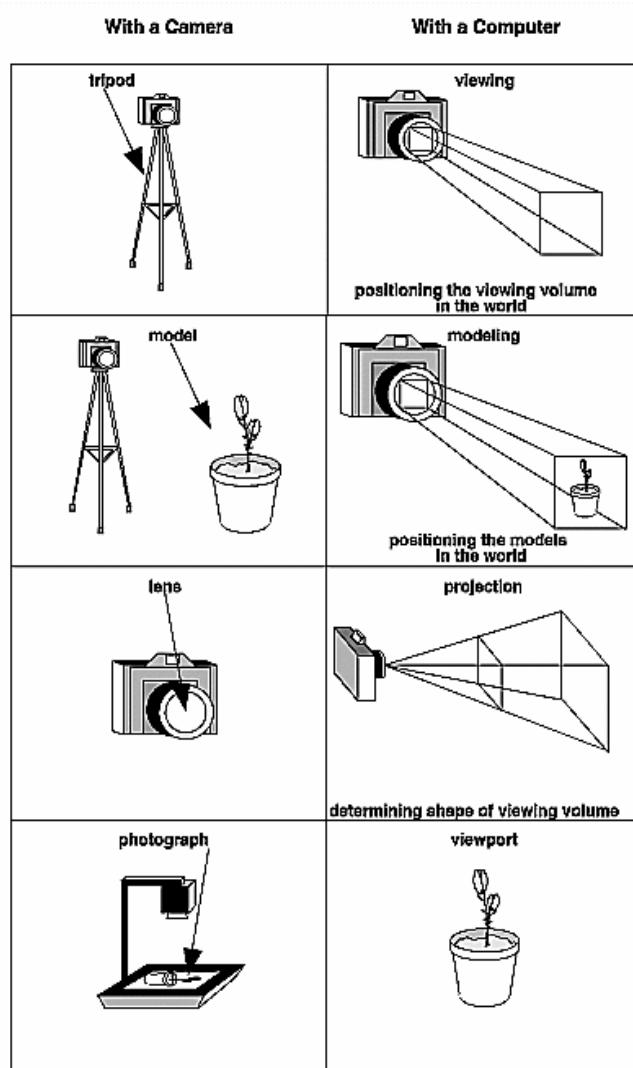


Pipeline graphique 3D

Caméra synthétique



Modèle de la caméra synthétique : analogie entre la visualisation d'une scène 3D à l'écran et la prise d'une photo à l'aide d'une caméra

Caméra synthétique

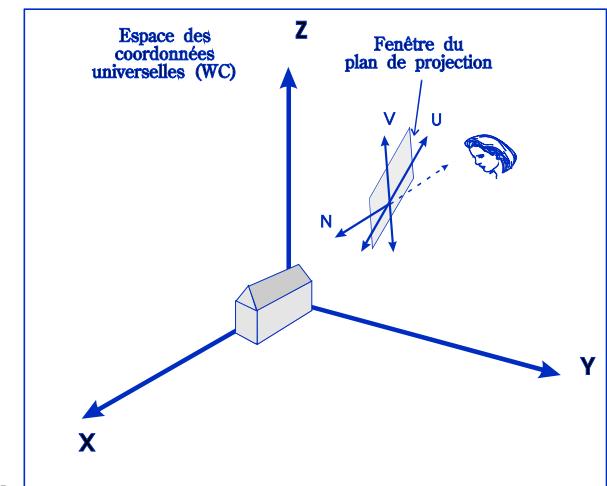
Modèle de la caméra synthétique: analogie entre la visualisation d'une scène 3D à l'écran et la prise d'une photo à l'aide d'une caméra :

- Établissement d'un système de coordonnées de visualisation
 - « placer puis pointer la caméra »
- Application d'une transformation de projection
 - « choisir une focale »
 - spécification du volume de visualisation (et du type de projection)
- Découpage de la scène selon le volume de visualisation
- Conversion au monde d'affichage
 - « choisir les dimensions de la photo »
 - correspondance entre les coordonnées transformées et les pixels à l'écran à l'intérieur de la clôture.

Référentiel de visualisation

Un peu d'historique :

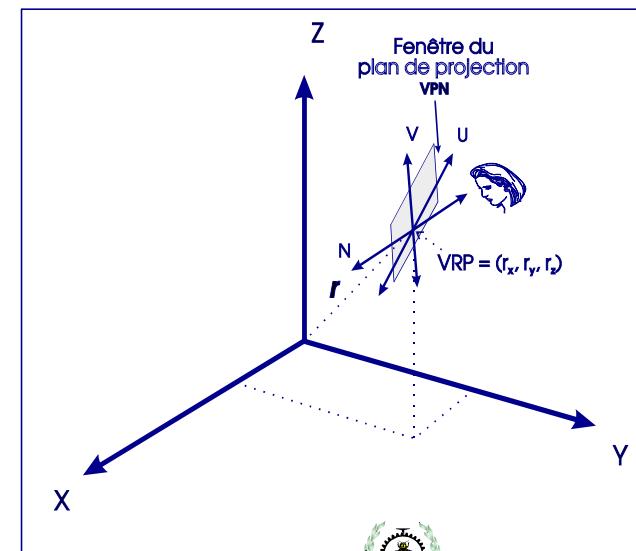
- L'observateur se situe sur l'axe des n et le plan de projection coïncide avec le plan u-v.
- Pour visualiser un objet de différents points de vue, il est nécessaire d'effectuer des rotations et des translations.
- Initialement introduit dans le système CORE, le modèle a été repris dans GKS-3D, PHIGS et IRIX/gl.
 - Ce modèle de caméra synthétique utilisait :
 - un plan de projection délimité par une fenêtre,
 - un système de coordonnées de visualisation (VC),
 - la position de l'observateur définie dans ce système de coordonnées.
 - Ce système de coordonnées est appelé UVN.
 - Le plan de visualisation coïncide avec le plan U-V.



Référentiel de visualisation

Système de coordonnées de la caméra

- La position du plan de projection (visualisation) est établie à l'aide d'un point de référence VRP [view reference point] sur le plan et ses coordonnées $r = (r_x, r_y, r_z)$ sont spécifiées en coordonnées universelles (WC).
- Normale n au plan : La normale au plan de visualisation VPN [viewplane normal] est spécifiée par le vecteur unitaire $n = (n_x, n_y, n_z)$.
- Vecteur v : La direction de l'axe V est spécifiée par un vecteur unitaire v perpendiculaire à n et qui habituellement pointe vers le haut.
- Vecteur u : Enfin, la direction de l'axe U est obtenue par le produit vectoriel $u = n \times v$
- La fenêtre et la position de l'observateur sont ensuite spécifiées en coordonnées de visualisation (VC).



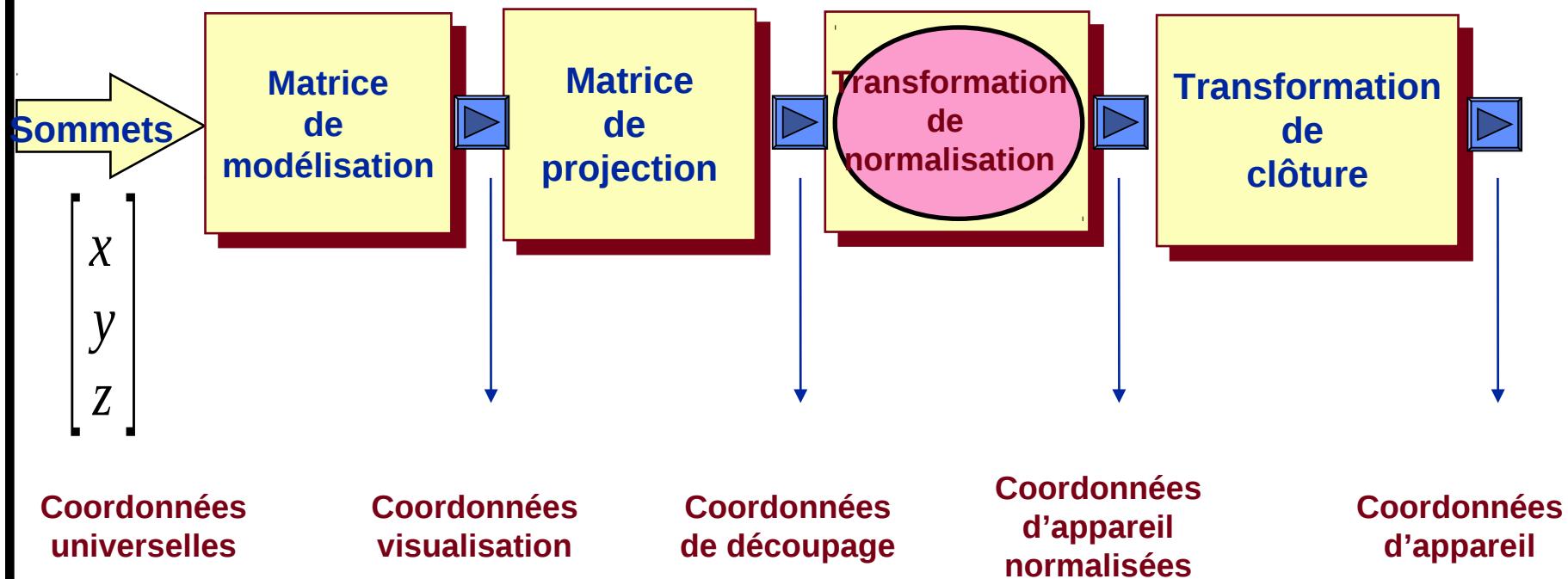
Référentiel de visualisation

Avantages du modèle de la caméra synthétique :

- survol: en changeant la position de l'observateur par l'intermédiaire de la valeur du VRP, on peut ainsi observer l'objet de différents points de vue,
- balayage: changer la direction du plan de visualisationVPN équivaut à pivoter la tête,
- pivotement: changer la direction du vecteur v (vers le haut) équivaut à pivoter la caméra, (certains modèles de caméra virtuelle permettent de spécifier directement l'angle de pivotement [spin]),
- angle d'ouverture [aperture]: peut être spécifié indirectement en changeant les dimensions de la fenêtre de visualisation.
- position de l'observateur: en variant la position de l'observateur, soit sur l'axe des N ou ailleurs, on peut obtenir une projection oblique.

PIPELINE DES TRANSFORMATIONS

OpenGL



Transformations de modélisation

Changement d'échelle

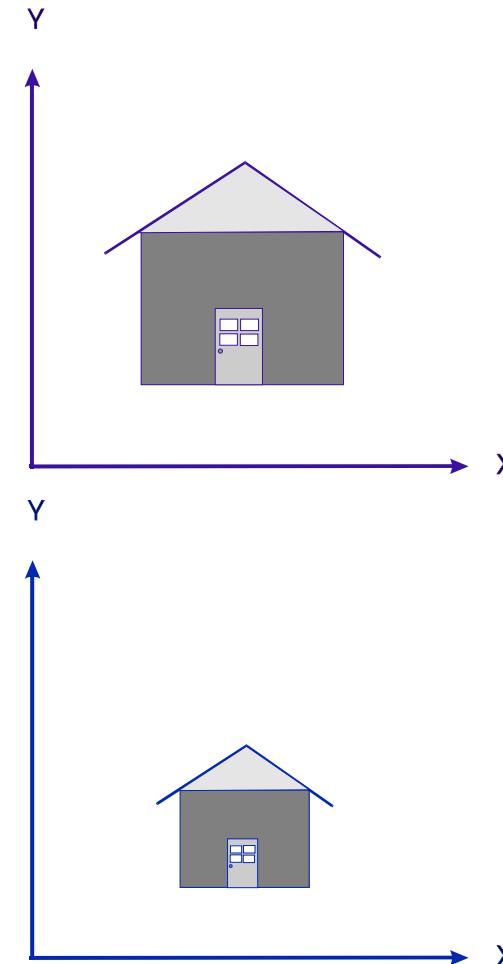
- **Changement d'échelle** : multiplier les coordonnées par un facteur d'échelle :

$$x^* = S_x \cdot x$$

$$y^* = S_y \cdot y$$

sous la forme matricielle suivante:

$$P^* = \begin{bmatrix} x^* \\ y^* \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



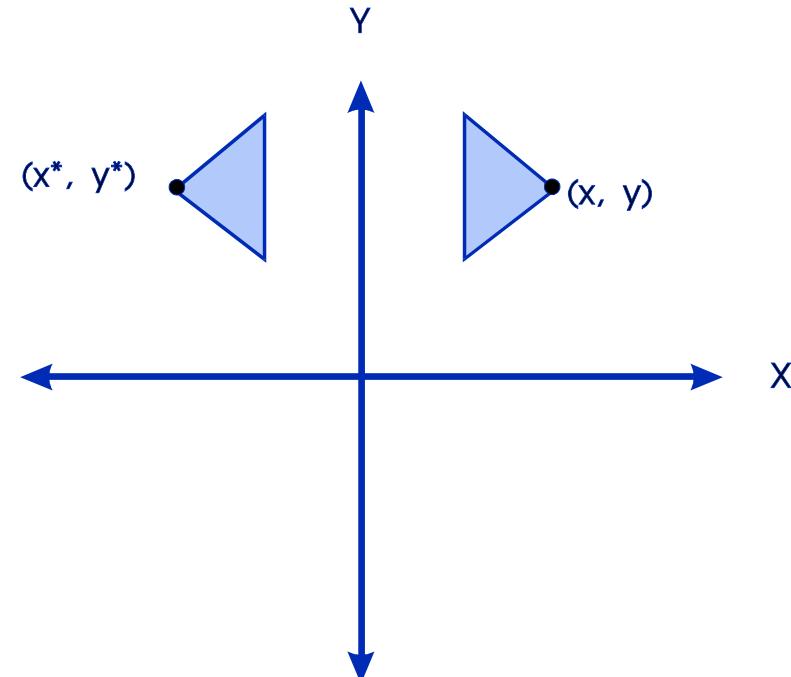
Réflexion

Réflexion d'un point par rapport à un axe: en changeant le signe de l'autre coordonnée.

Par rapport à l'axe des y

$$x^* = -x$$

$$y^* = y$$



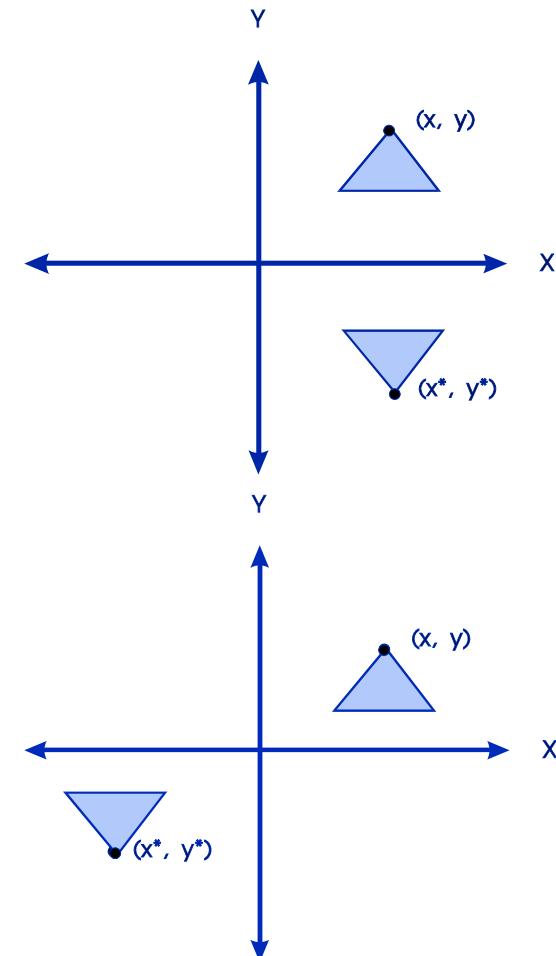
Sous forme matricielle

$$P^* = \begin{bmatrix} x^* \\ y^* \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Réflexion (suite)

- **Réflexion** par rapport à l'axe des x:

$$P^* = \begin{bmatrix} x^* \\ y^* \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



- par rapport à l'origine

$$P^* = \begin{bmatrix} x^* \\ y^* \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

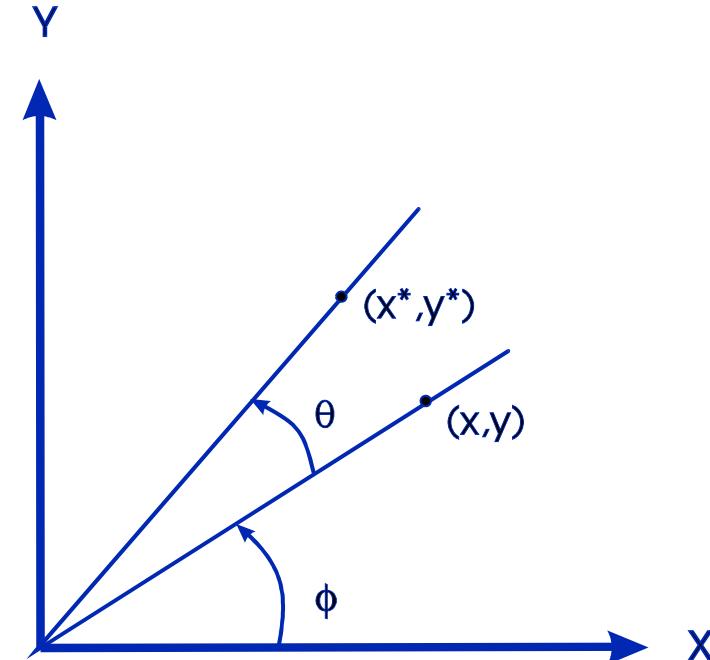
Rotation

- **Rotation** d'un point par rapport à l'origine: le point $P = (x, y)$, peut s'écrire

$$x = r \cos \phi$$

$$y = r \sin \phi$$

où r : le rayon de l'origine au point P .



Rotation (suite)

- le point transformé,

$$\begin{aligned}x^* &= r \cos(\theta + \phi) \\&= r \cos\phi \cos\theta - r \sin\phi \sin\theta \\&= x \cdot \cos\theta - y \cdot \sin\theta \\y^* &= r \sin(\theta + \phi) \\&= r \cos\phi \sin\theta + r \sin\phi \cos\theta \\&= x \cdot \sin\theta + y \cdot \cos\theta\end{aligned}$$

- en notation matricielle,

$$P^* = \begin{bmatrix} x^* \\ y^* \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

TRANSLATION

- Translation: déplacer un point dans une direction donnée.

$$\vec{P} = \vec{P} + \vec{T}$$

où P est le point original,

T = [Tx, Ty]^t est le déplacement

et P* le point transformé.

$$\begin{aligned}x^* &= x + T_x \\y^* &= y + T_y\end{aligned}$$

- C'est une somme au lieu d'un produit. :-(

TRANSLATION (suite)

- On utilise l'artifice suivant

$$P = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

- Alors, le système peut s'exprimer aisément comme un produit

$$\begin{bmatrix} x^* \\ y^* \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Coordonnées homogènes

- Ajoute une composante au point pour donner

$$P = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- On peut l'interpréter comme une coordonnée additionnelle, ce qui place le plan $x-y$ comme un sous-ensemble d'un espace à trois dimensions. Dans ce cas, il faut noter que la troisième dimension n'est pas physique. On multiplie maintenant le point $(x, y, 1)$ par un scalaire $h \neq 0$,

$$P = \begin{bmatrix} hx \\ hy \\ h \end{bmatrix}$$

TRANSFORMATIONS DANS L'ESPACE

- Représentation d'un point

$$P = (x, y, z)^t$$

- On introduit une quatrième composante

$$(x, y, z, 1)^t$$

que l'on interprète comme un point dans un espace à 3+1 dimensions.

- Un objet tridimensionnel, à n sommets, devient ainsi :

$$\begin{bmatrix} x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \\ z_1 & z_2 & \dots & z_n \\ 1 & 1 & \dots & 1 \end{bmatrix}$$

TRANSFORMATIONS DANS L'ESPACE

- La forme générale de la matrice de transformation 3D est :

Matrice de transformation

$$= \begin{bmatrix} a & b & c & l \\ d & e & f & m \\ g & h & i & n \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

matrice de transformation

$$= \begin{bmatrix} 3 \times 3 & 3 \\ \dots & \times \\ \dots & 1 \\ \hline 1 \times 3 & 1 \times 1 \end{bmatrix}$$

TRANSFORMATION DE MODÉLISATION

OpenGL

- Transformations en 3D

$$\begin{bmatrix} x^* \\ y^* \\ z^* \\ 1 \end{bmatrix} = \begin{bmatrix} m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \\ m_4 & m_8 & m_{12} & m_{16} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Transformation en 2D

$$\begin{bmatrix} x^* \\ y^* \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} m_1 & m_5 & 0 & m_{13} \\ m_2 & m_6 & 0 & m_{14} \\ 0 & 0 & 1 & 0 \\ m_4 & m_8 & 0 & m_{16} \end{bmatrix} \bullet \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix}$$

CONCATÉNATION DES TRANSFORMATIONS

```
glMatrixMode ( GL_MODELVIEW);  
glLoadIdentity();  
glTranslatef ( 4.0, 5.0, 6.0);  
glRotatef(45.0, 1.0, 2.0, 3.0);  
glTranslatef ( -4.0, -5.0, -6.0);
```

$$C \leftarrow I$$

$$C \leftarrow C * T(4.0, 5.0, 6.0)$$

$$C \leftarrow C * R(45.0, 1.0, 2.0, 3.0)$$

$$C \leftarrow C * T(-4.0, -5.0, -6.0)$$

$$C = T(4.0, 5.0, 6.0) R(45.0, 1.0, 2.0, 3.0) T(-4.0, -5.0, -6.0)$$

CONCATÉNATION DES TRANSFORMATIONS

- L'enchaînement des transformations est effectué par la multiplication des matrices de transformation.
- Soit p un point et q le point transformé;
 C, B, A des matrices de transformation
- Le produit matriciel étant associatif, on peut écrire

$$q = CBAp$$

est effectué comme suit:

$$q = (C(B(Ap))) = (CBA)p = Mp$$

avec

$$M = CBA$$



TRANSFORMATION DE MODÉLISATION

OpenGL

- Pour appliquer les transformations, OpenGL peut donc concaténer les multiples transformations dans une seule matrice courante [CTM]

Current Transformation = CTM = Matrix

$$\begin{bmatrix} m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \\ m_4 & m_8 & m_{12} & m_{16} \end{bmatrix}$$

(Il y a un matrice pour les transformations de modélisation,
une matrice pour les transformations de projection et
une matrice pour les transformations de texture.)

TRANSLATION

- Calcul des coordonnées d'un point déplacé par une translation

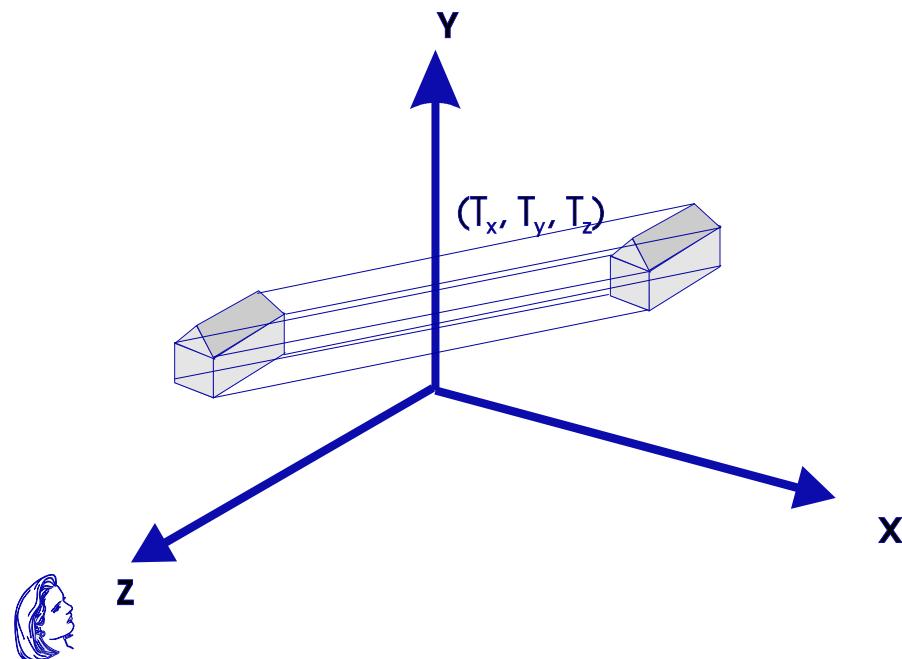
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

ce qui correspond à :

$$x' = x + T_x$$

$$y' = y + T_y$$

$$z' = z + T_z$$

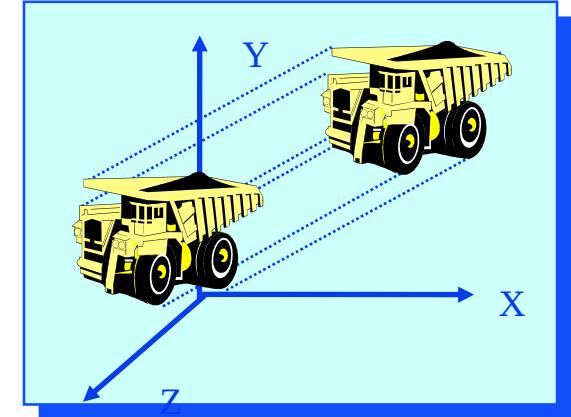


TRANSLATION

OpenGL

- Appliquer une translation

```
void glTranslate{fd}( TYPE Tx, TYPE Ty, TYPE Tz );
```



$$CTM = C \leftarrow C \cdot \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

MISE À L'ÉCHELLE

- Les éléments de la matrice de transformation sur la diagonale produisent des mises à l'échelle sur un objet.

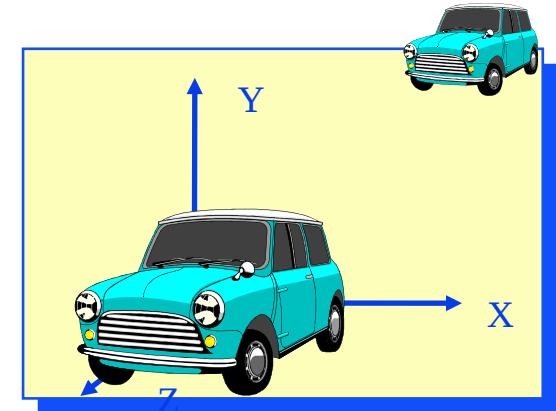


$$\begin{aligned} P' &= (x', y', z', 1)^t = (xS_x, yS_y, zS_z, 1)^t \\ &= \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \\ &= S \bullet P \end{aligned}$$

MISE À L'ÉCHELLE OpenGL

- Applicuer une mise à l'échelle

```
void glScale{fd}(TYPE Sx, TYPE Sy, TYPE Sz);
```



$$CTM = C \leftarrow C \bullet S = C \bullet \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

ROTATION OpenGL

- ➊ Appliquer une rotation

```
void glRotate{fd}( TYPE angle, TYPE x, TYPE y, TYPE z );
```

angle en degrés (p.e. 45°)

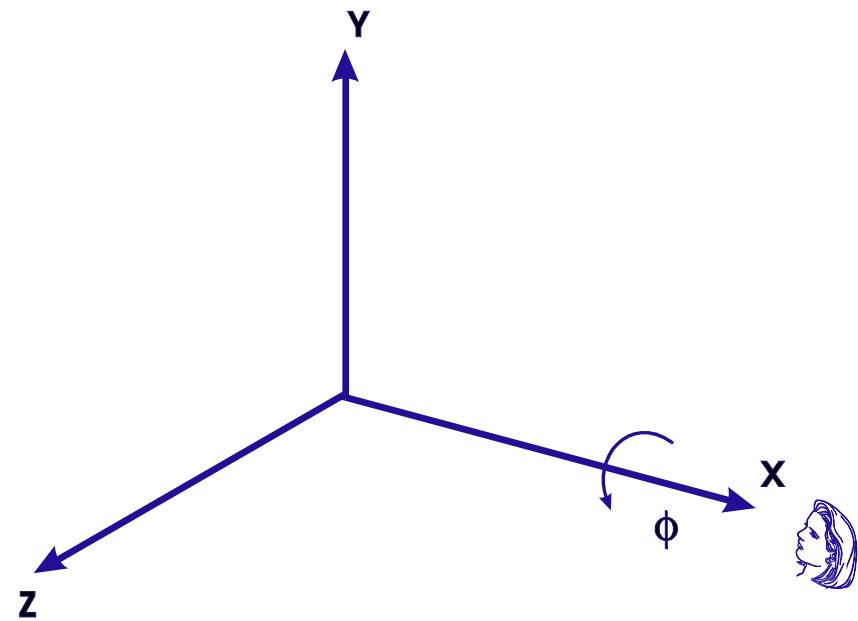
$$CTM = C \leftarrow C \bullet R(\theta)$$

ROTATIONS

- Autour de l'axe x

$$P' = R_x(\phi) P$$

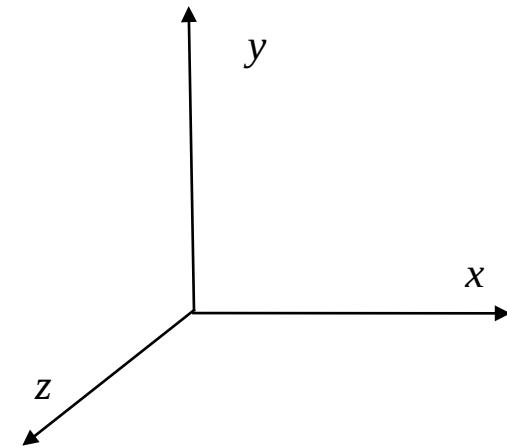
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



ROTATION OpenGL

- Rotation autour de l'axe des x

glRotate{fd} (angle, 1.0, 0.0, 0.0);



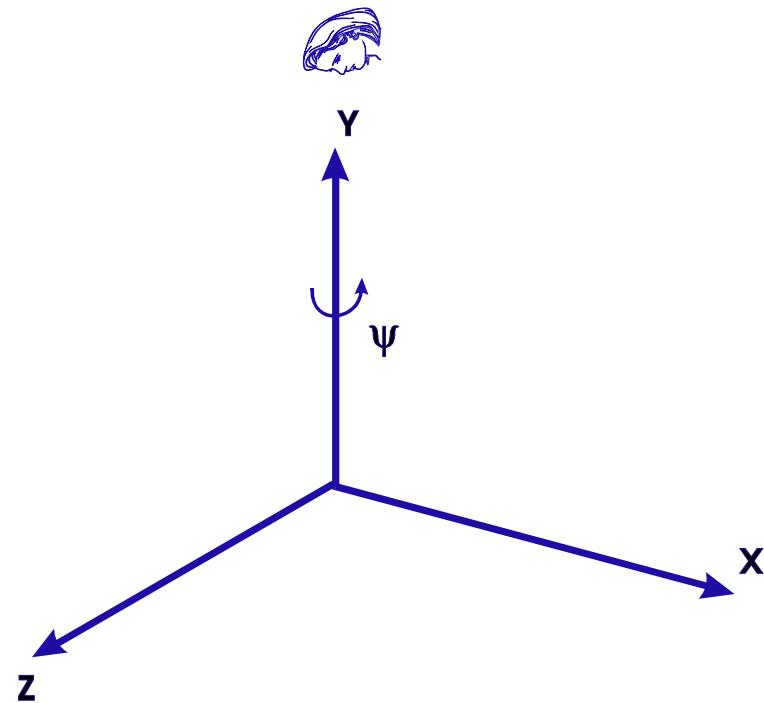
$$CTM = C \leftarrow C \bullet R_x(\theta_x) = C \bullet \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta_x & -\sin \theta_x & 0 \\ 0 & \sin \theta_x & \cos \theta_x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

ROTATIONS

$$P' = R_Y(\psi) P$$

$$R_Y(\psi) = \begin{bmatrix} \cos\psi & 0 & \sin\psi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\psi & 0 & \cos\psi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

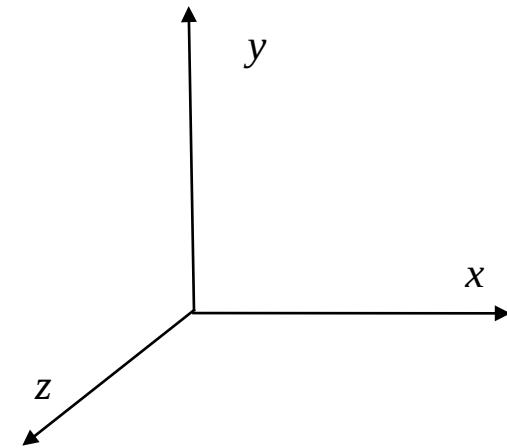
$$P' = R_z(\theta) \bullet R_y(\psi) \bullet R_x(\varphi) \bullet P$$



ROTATION OpenGL

- Rotation autour de l'axe des y

glRotate{fd} (angle, 0.0, 1.0, 0.0);



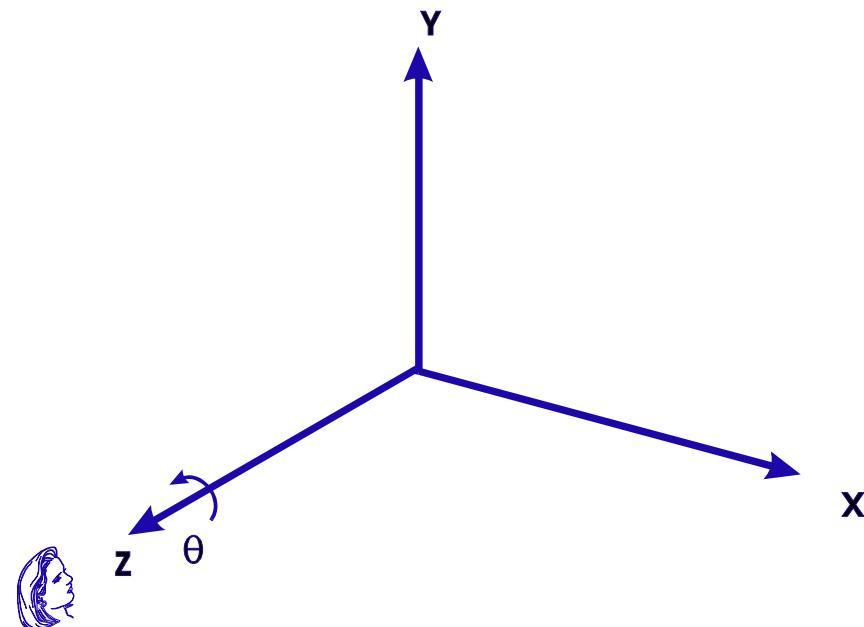
$$CTM = C \leftarrow C \bullet R_y(\theta_y) = C \bullet \begin{bmatrix} \cos \theta_y & 0 & \sin \theta_y & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta_y & 0 & \cos \theta_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

ROTATIONS

- Autour de l'axe z

$$P' = R_Z(\theta) P$$

$$R_Z(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

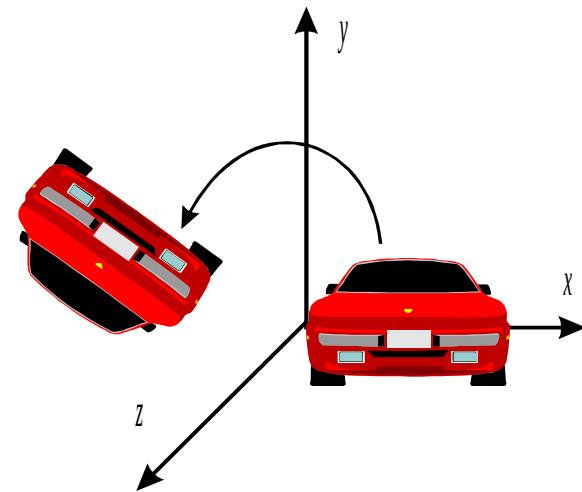


ROTATION

OpenGL

- Rotation autour de l'axe des z

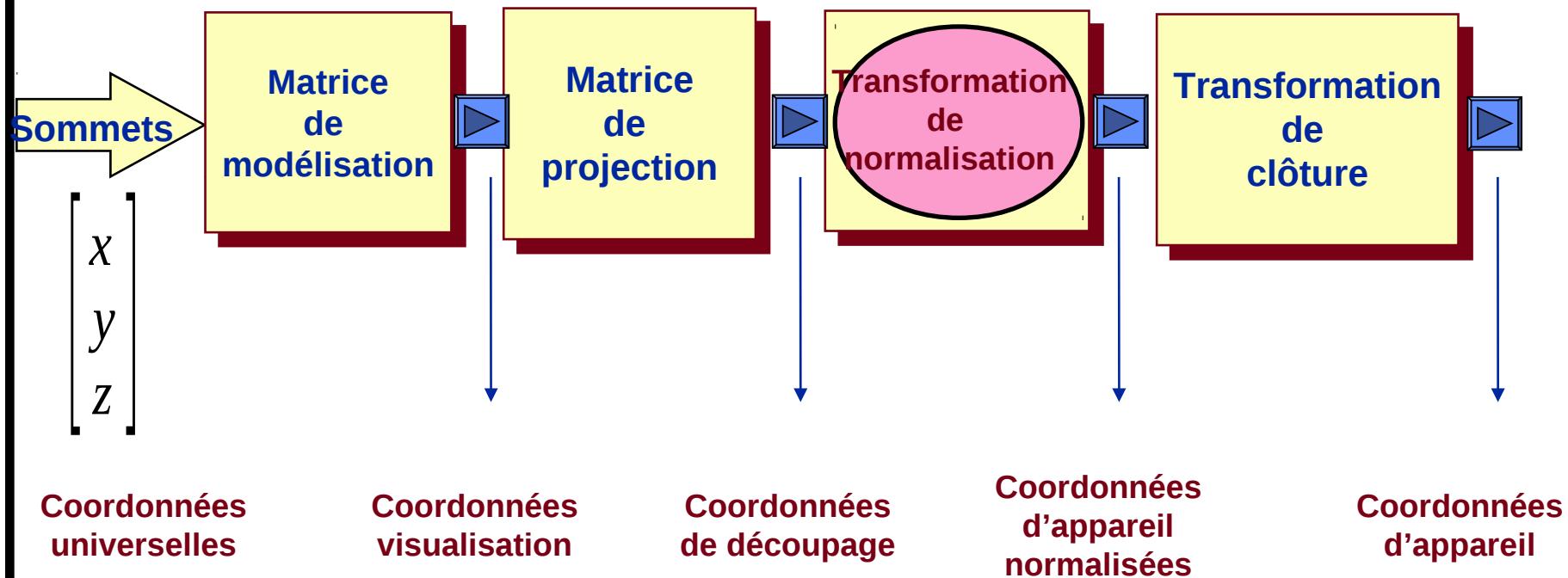
glRotate{fd} (angle, 0.0, 0.0, 1.0);



$$CTM = C \leftarrow C \bullet R_z(\theta_z) = C \bullet \begin{bmatrix} \cos \theta_z & -\sin \theta_z & 0 & 0 \\ \sin \theta_z & \cos \theta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

PIPELINE DES TRANSFORMATIONS

OpenGL



MATRICE DE MODÉLISATION

OpenGL

Il faut indiquer que les transformations précédentes s'appliquent à la matrice de modélisation.

- ➊ Indiquer à quelle matrice s'appliquent les énoncés qui suivent :

*void **glMatrixMode** (GLenum mode);*

mode GL_MODELVIEW

GL_PROJECTION

GL_TEXTURE

- ➋ Charger la matrice identité

*void **glLoadIdentity** ();*

COPIER LA MATRICE COURANTE

OpenGL

- Obtenir une copie de la matrice courante

```
void glGetFloatv( GLenum pname, GLfloat *m );
```

```
void glGetDoublev( GLenum pname, GLdouble *m );
```

pname GL_MODELVIEW_MATRIX
 GL_PROJECTION_MATRIX
 GL_TEXTURE_MATRIX

$m \leftarrow \text{CTM}$

CHARGER UNE MATRICE

OpenGL

- Charger une matrice de transformation

glLoadMatrix{fd} (*const TYPE *m*);

$$CTM \leftarrow \begin{bmatrix} m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \\ m_4 & m_8 & m_{12} & m_{16} \end{bmatrix}$$

MULTIPLIER UNE MATRICE

OpenGL

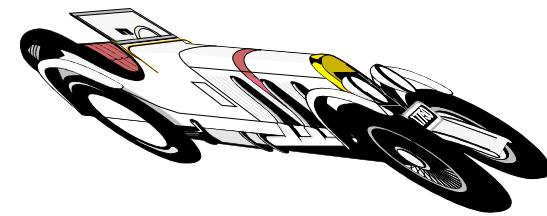
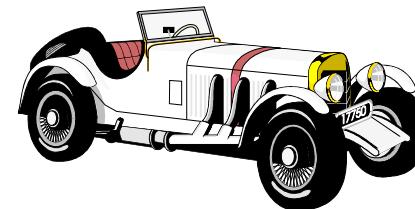
- Concaténer une matrice à la matrice courante de transformation

glMultMatrix{fd} (*const TYPE *m*);

$$CTM \leftarrow CTM \bullet \begin{bmatrix} m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \\ m_4 & m_8 & m_{12} & m_{16} \end{bmatrix}$$

GLISSEMENT

$$G = \begin{bmatrix} 1 & Sh_{yx} & Sh_{zx} & 0 \\ Sh_{xy} & 1 & Sh_{zy} & 0 \\ Sh_{xz} & Sh_{yz} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



- La dépendance de x' est exprimée par l'équation $x' = x + Sh_{yx}Y + Sh_{zx}Z$.
 Sh_{yx} indique de combien la coordonnée y affecte la coordonnée x'
- $P' = GP$

CHARGER UNE MATRICE DE GLISSEMENT

OpenGL

- Charger une matrice de glissement

glLoadMatrix{fd} (*const TYPE *m*);

$$CTM \leftarrow H_{yz}(\theta) = \begin{bmatrix} 1 & \cot \theta & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

MISE SUR LA PILE

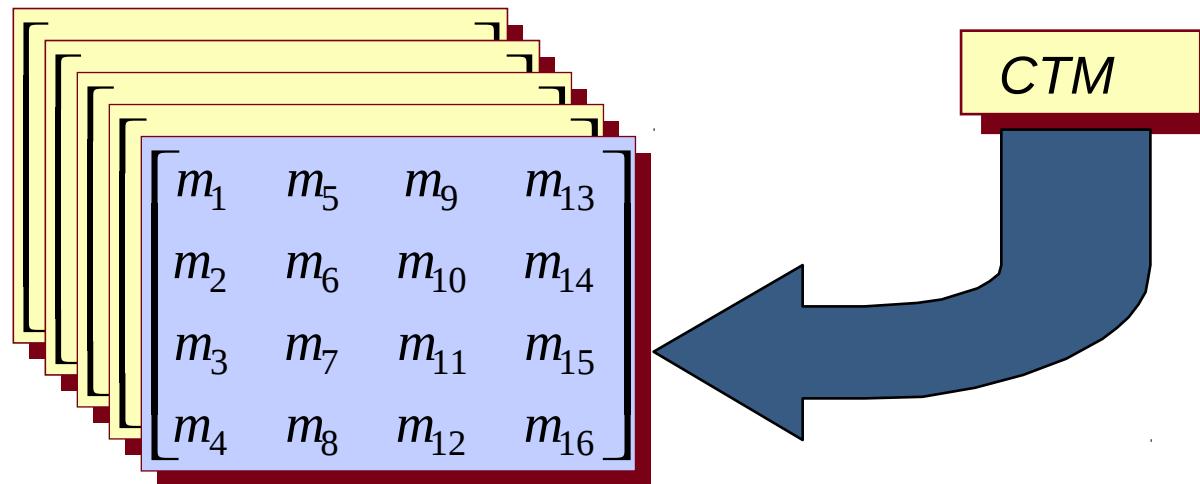
OpenGL

- OpenGL offre une pile de matrices où on peut sauvegarder la matrice courante
- On peut copier la matrice courante sur cette pile pour se souvenir de la position courante du repère afin d'y revenir plus tard.
- On utilise
 - ***glPushMatrix()***
 - ***glPopMatrix()***
 - ***glLoadMatrix()***
- S'applique séparément pour les matrices de modélisation, de projection et de texture. (Trois piles différentes sont utilisées.)

MISE SUR LA PILE OpenGL

- ➊ Déposer une copie de la matrice courante sur la pile

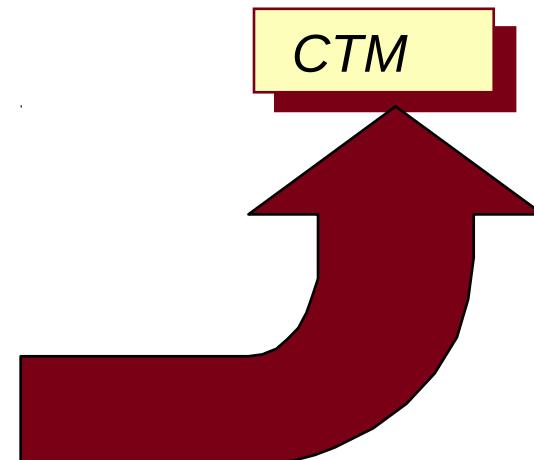
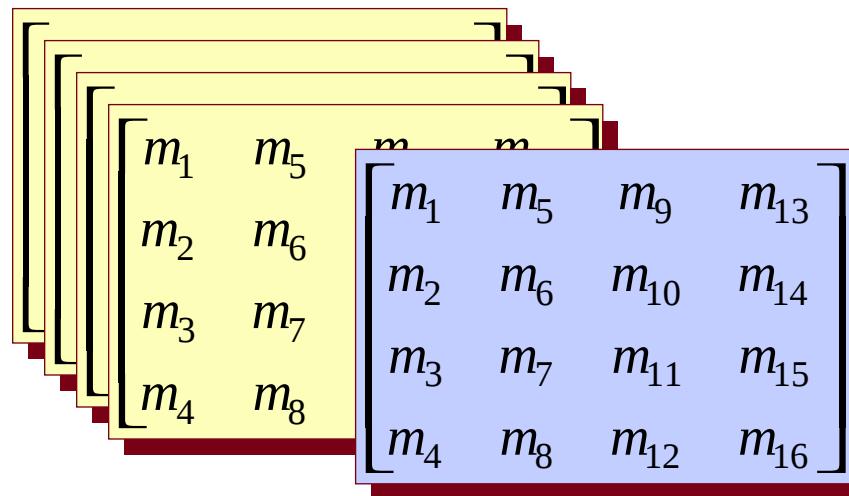
```
void glPushMatrix ( );
```



RETIRER DE LA PILE OpenGL

- Retirer une matrice de la pile et la mettre dans la matrice courante

*void **glPopMatrix** ();*



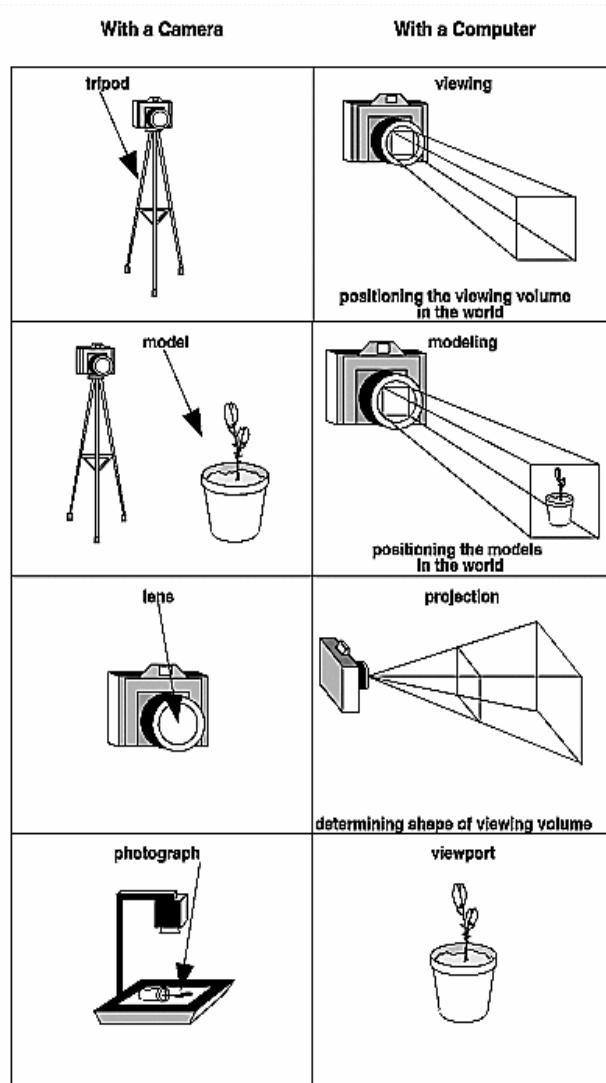
FONCTIONS DE TRANSFORMATION

OpenGL

<i>glMatrixMode</i>	Choisir la matrice
<i>glLoadIdentity</i>	Charger la matrice identité
<i>glLoadMatrix</i>	Charger une matrice comme matrice courante
<i>glMultMatrix</i>	Multiplier une matrice avec la matrice courante
<i>glTranslate</i>	Obtenir une matrice de translation
<i>glRotate</i>	Obtenir une matrice de rotation
<i>glScale</i>	Obtenir une matrice de mise à l'échelle
<i>glPushMatrix</i>	Stocker la matrice courante sur la pile
<i>glPopMatrix</i>	Prendre de la pile la matrice (devient courante)
<i>glGetFloatv</i>	Obtenir copie de la matrice courante

Positionner la caméra (visualisation)

Caméra synthétique



La matrice de modélisation est, en fait, composée de deux parties:

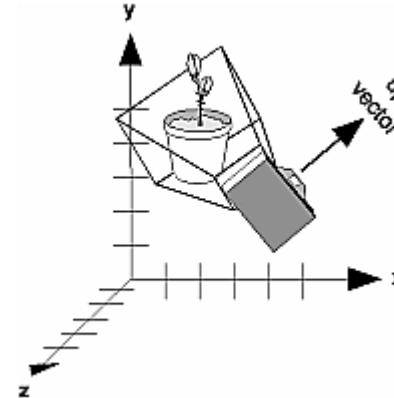
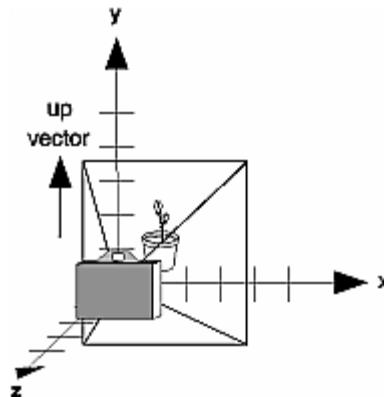
- 1) Le positionnement de la caméra synthétique (visualisation) avec *gluLookAt*
- 2) Les transformations de modélisation: *glTranslate*, *glScale*, *glRotate*

Établir les coordonnées de visualisation

Utilisation de la caméra synthétique :

- Établissement d'un système de coordonnées de visualisation
« placer puis pointer la caméra »
- (Initialement, la caméra est à l'origine et pointe vers les z négatifs)

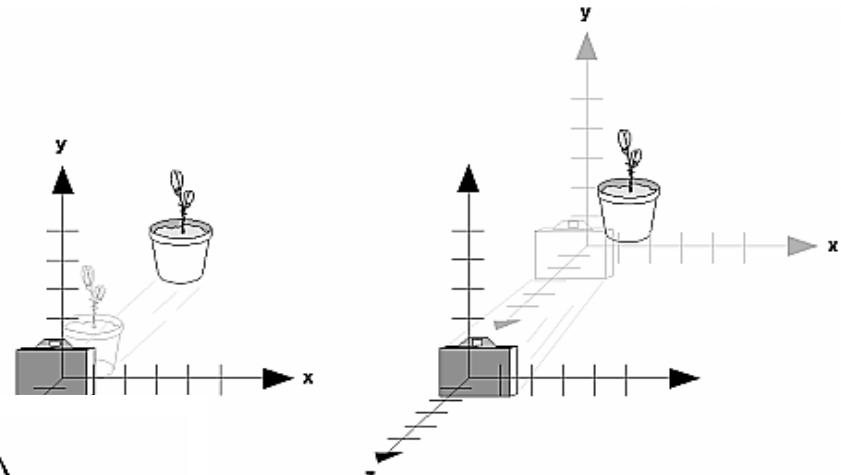
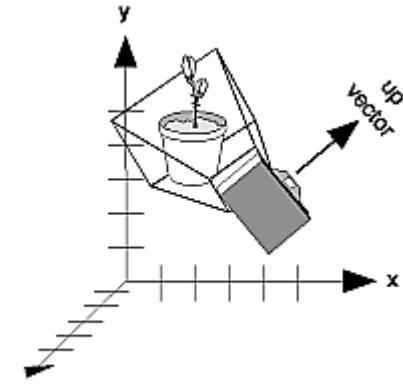
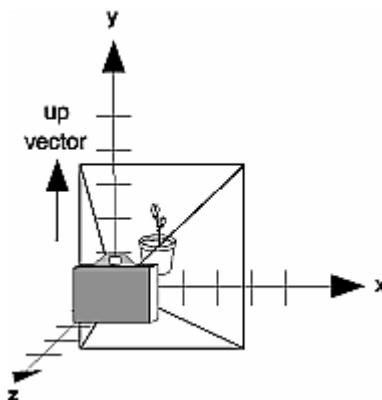
```
gluLookAt( GLdouble obsx,      GLdouble obsy,      GLdouble obsz,  
           GLdouble ptVisex,   GLdouble ptVisey,   GLdouble ptVisez,  
           GLdouble upx,       GLdouble upy,       GLdouble upz );
```



Positionner la caméra

- *gluLookAt()* définit un changement de base de l'espace vectoriel
- On pourrait faire l'équivalent avec une rotation et une translation

$$A_{WV} = \begin{bmatrix} u_x & u_y & u_z & r'_x \\ v_x & v_y & v_z & r'_y \\ n_x & n_y & n_z & r'_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Positionner la caméra

- Quelques exemples équivalents (sans gluLookAt):

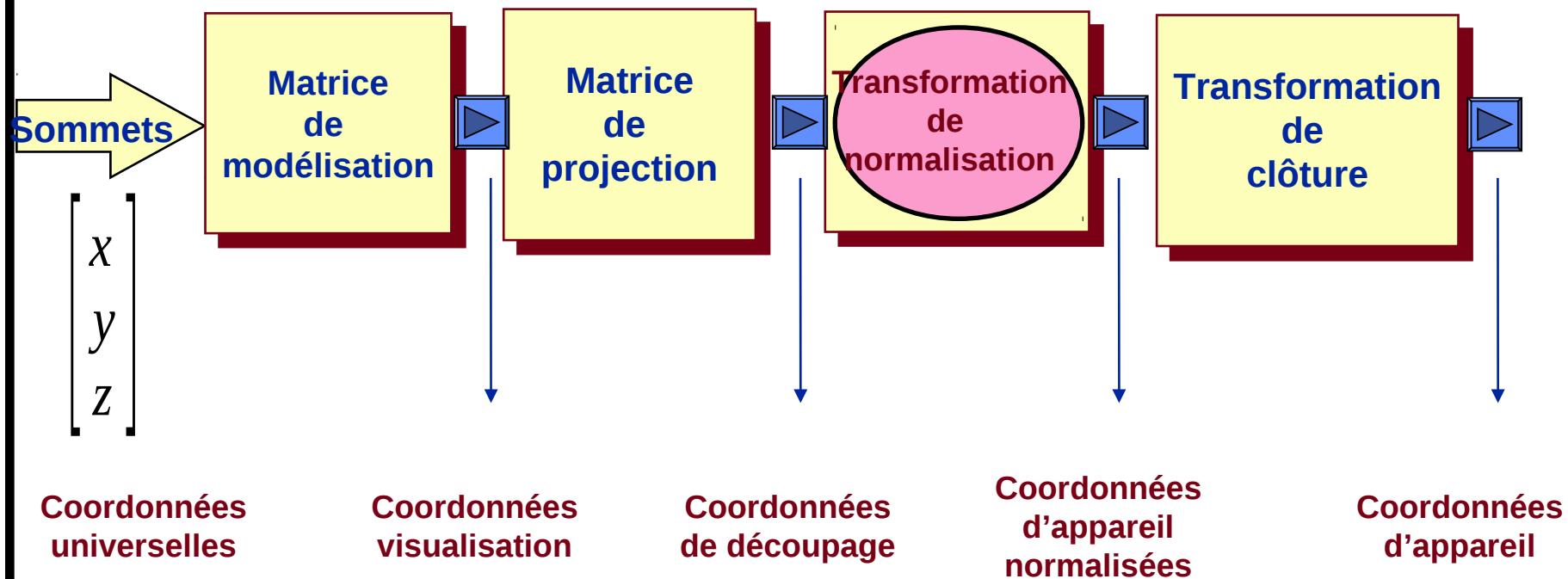
```
void pilotView( GLdouble planex, GLdouble planey, GLdouble planez,
                GLdouble roll, GLdouble pitch, GLdouble heading )
{
    glRotated( roll, 0.0, 0.0, 1.0 );
    glRotated( pitch, 0.0, 1.0, 0.0 );
    glRotated( heading, 1.0, 0.0, 0.0 );
    glTranslated( planex, planey, planez );
}

void polarView( GLdouble distance,
                GLdouble twist, GLdouble elevation, GLdouble azimuth )
{
    glTranslated( 0.0, 0.0, distance );
    glRotated( twist, 0.0, 0.0, 1.0 );
    glRotated( elevation, 1.0, 0.0, 0.0 );
    glRotated( azimuth, 0.0, 0.0, 1.0 );
}
```

Définir la projection

PIPELINE DES TRANSFORMATIONS

OpenGL

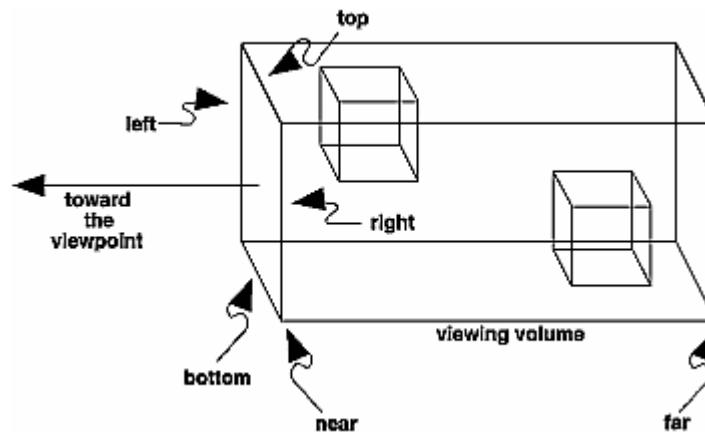


MATRICE DE PROJECTION

OpenGL

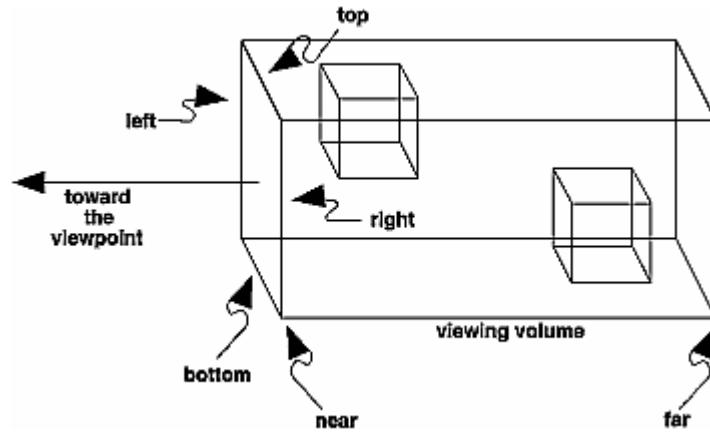
- La transformation de projection permet de définir le volume de visualisation, c'est-à-dire la portion de l'espace 3D qui sera affichée (projétée) à l'écran
 - Après l'application de la transformation de visualisation (et de modélisation), le repère est maintenant positionné à l'oeil de l'observateur
 - On veut alors définir la région 3D qui sera visible à partir de ce point de vue
- Deux types de projection sont possibles :
 - 1) projection orthographique : pour maintenir la taille des objets et les angles (ex. : afficher un plan)
 - 2) projection perspective : pour afficher un rendu réaliste

Projection parallèle (OpenGL)



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

Projection parallèle (OpenGL)



```
void glOrtho( GLdouble left, GLdouble right,  
              GLdouble bottom, GLdouble top,  
              GLdouble near, GLdouble far);  
  
void gluOrtho2D( GLdouble left, GLdouble right,  
                  GLdouble bottom, GLdouble top );  
  
(near = -1, far = 1 )
```

Projections orthogonales: choix du point de vue

- En dessin technique, une projection sur un des plans principaux peut être définie ainsi:
 - Sur $z=0$, (plan x-y)

$$M_z = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Sur $y=0$, (plan x-z)

$$M_y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Sur $x=0$, (plan y-z)

$$M_x = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

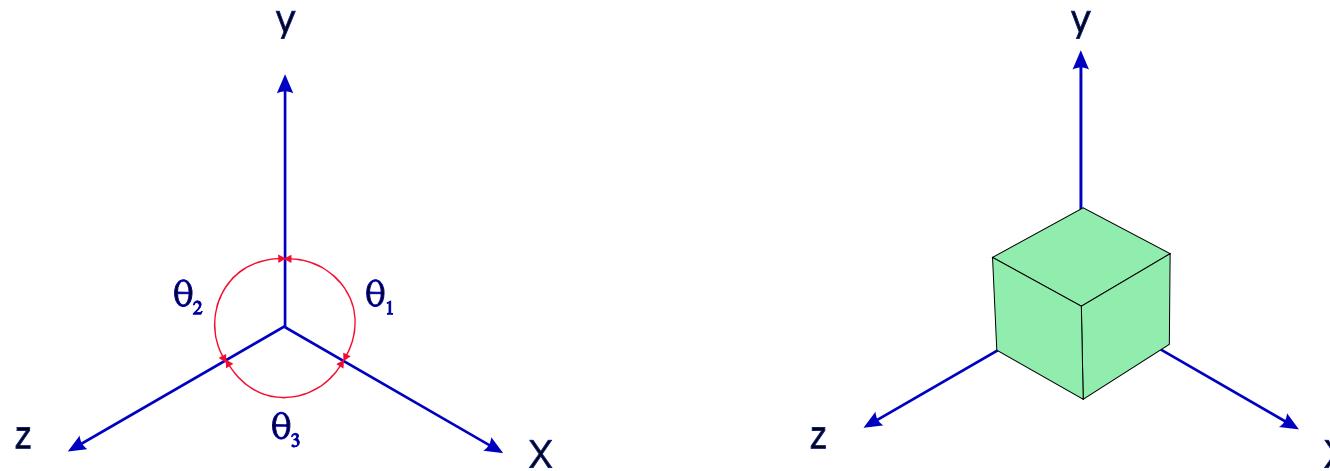
Projection axonométrique: choix du point de vue

- La projection axonométrique est une projection orthogonale dans laquelle la direction de projection (axe de projection) n'est pas parallèle à l'un des axes principaux.
- On procède au préalable par une transformation de l'objet pour en obtenir un bon point de vue (avec gluLookAt). On projette ensuite sur un plan avec une projection orthogonale.
- Une projection axonométrique comprend donc deux parties:
 - une rotation de la caméra
 - suivie d'une projection orthogonale.
- De façon générale, l'observateur se situe sur l'axe des z et on projette sur $z = 0$. Donc les seules rotations nécessaires pour obtenir une vue sont celles autour de x et y.
- Les rotations sont représentées par :

$$M_{\text{axonométrique}} = M_z \cdot R_y(\Psi) \cdot R_x(\phi)$$

Projection axonométrique: choix du point de vue

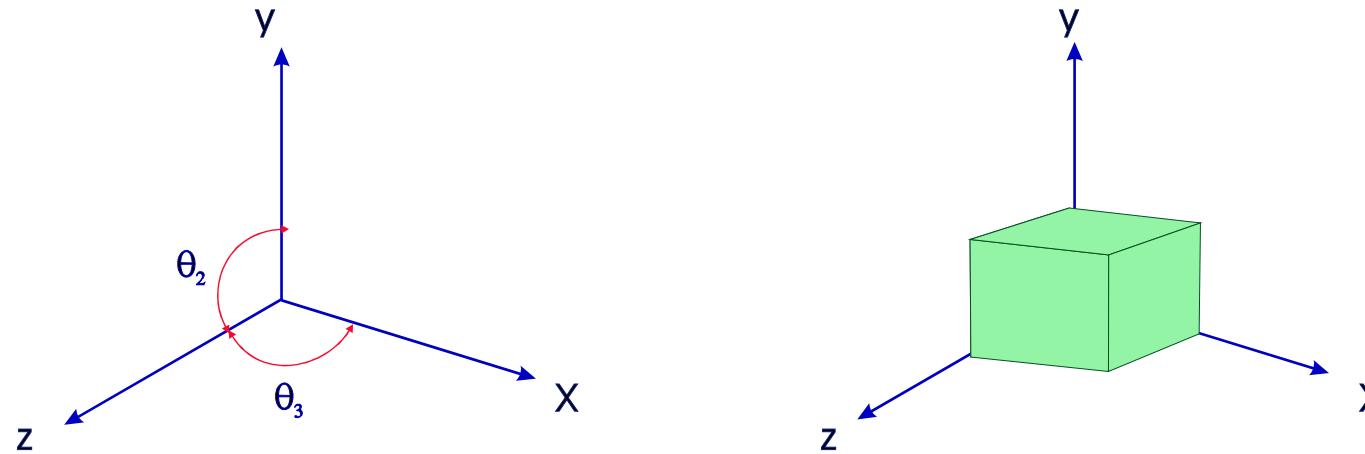
- Projection isométrique: $\phi=35,26439^\circ$ $\psi = - 45^\circ$
La direction de la projection fait un angle égal avec *chacun* des trois axes principaux.



Projection axonométrique: choix du point de vue

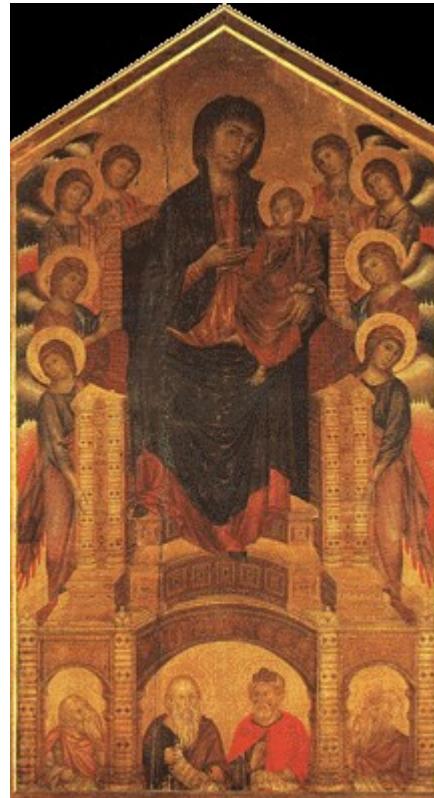
- Projection dimétrique: $\phi = 20,705^\circ$ $\psi = 22,208^\circ$

La direction de la projection fait un angle égal avec *deux* des trois axes principaux.



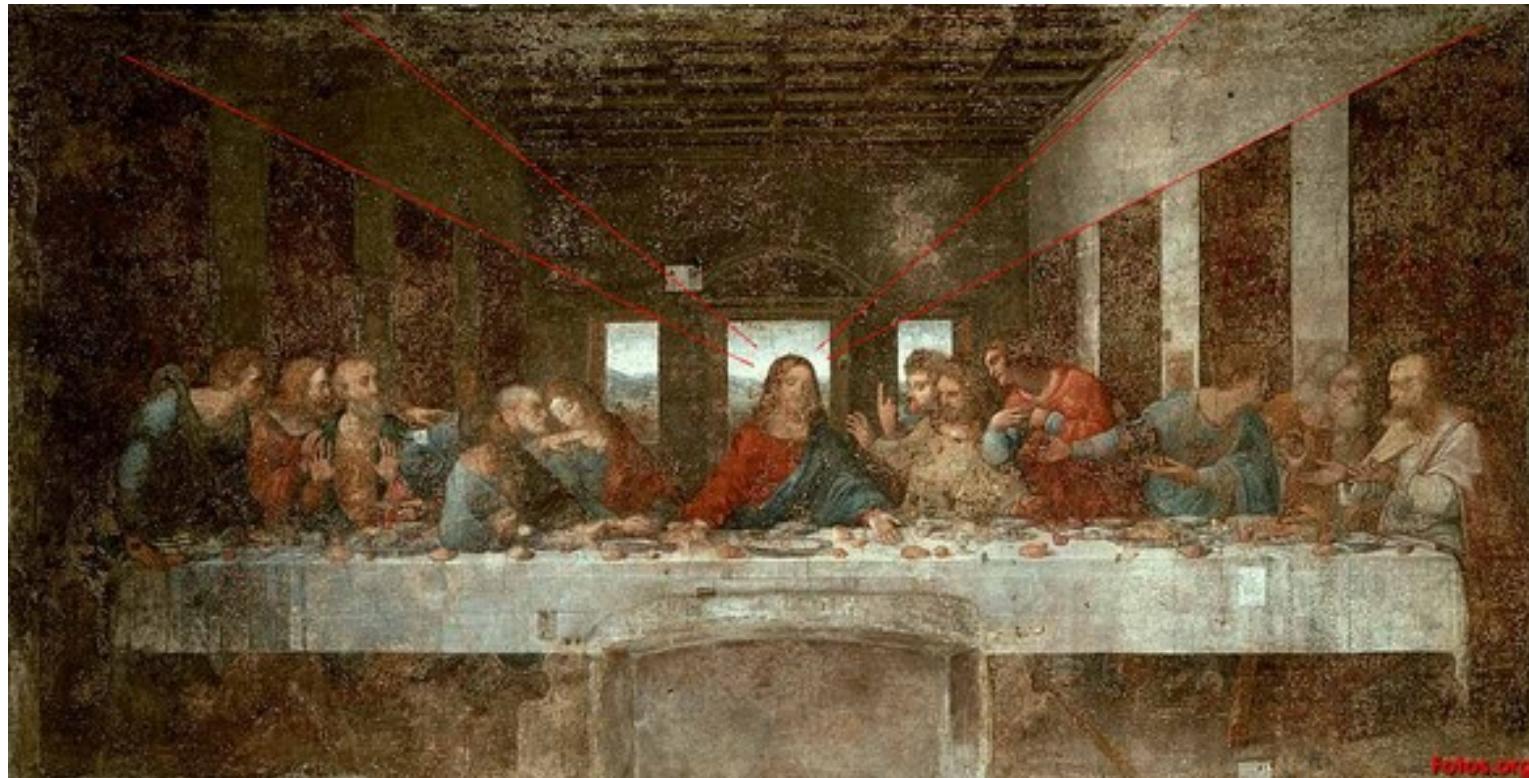
Projection perspective

Au Moyen âge, on ne savait pas reproduire correctement la perspective:



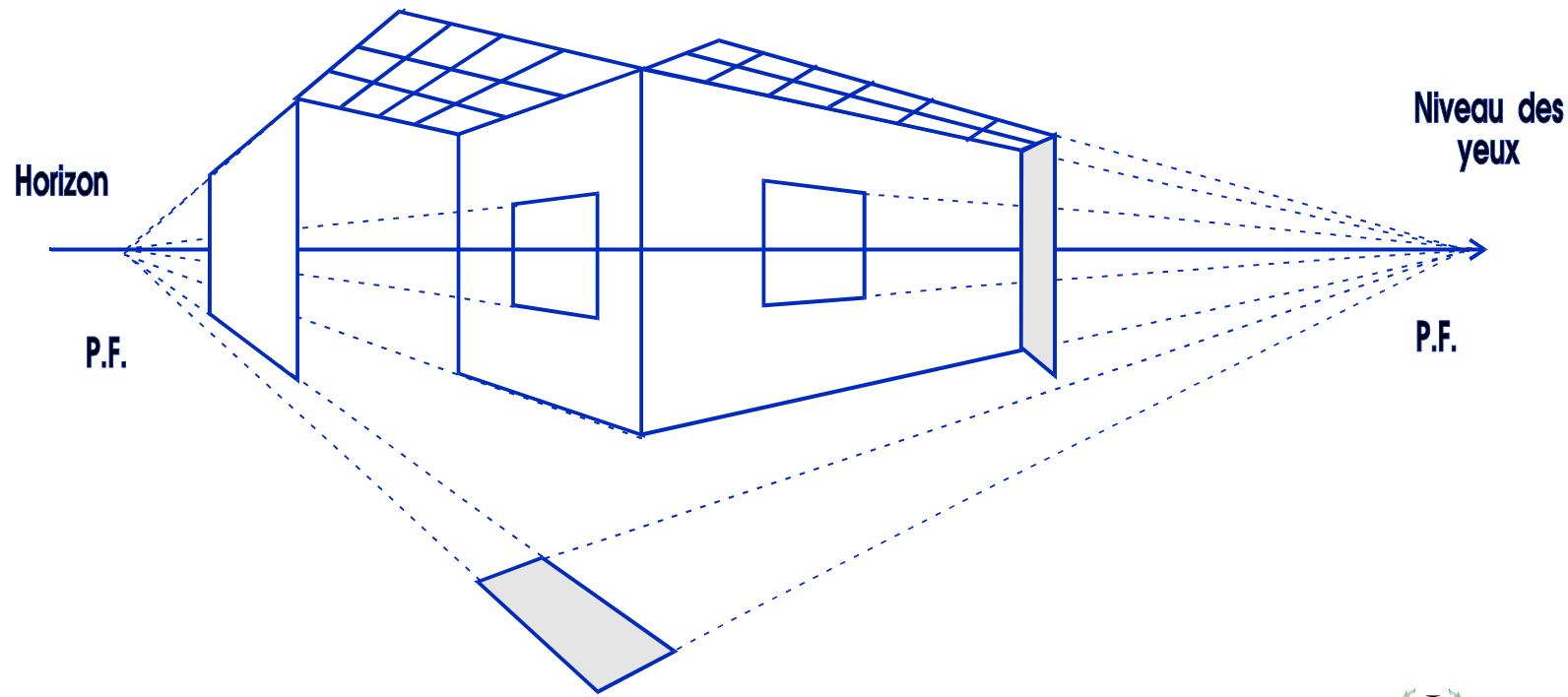
Projection perspective

Léonard de Vinci a compris comment reproduire la perspective et l'utilise pour diriger l'attention sur le point de fuite :



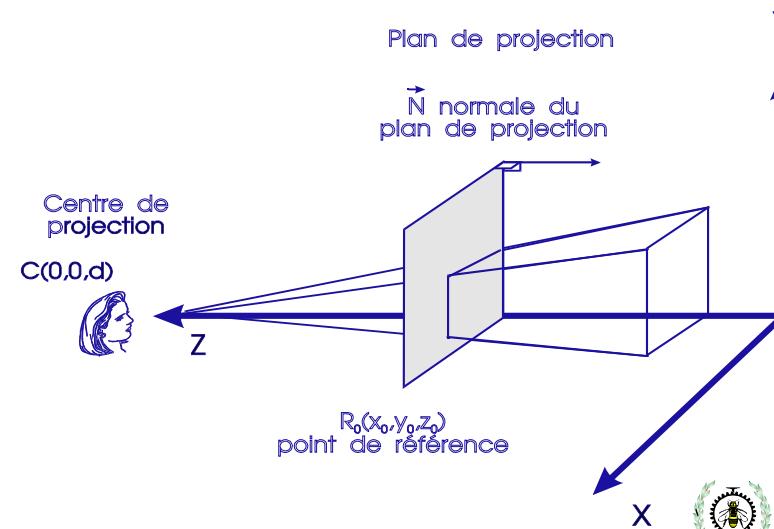
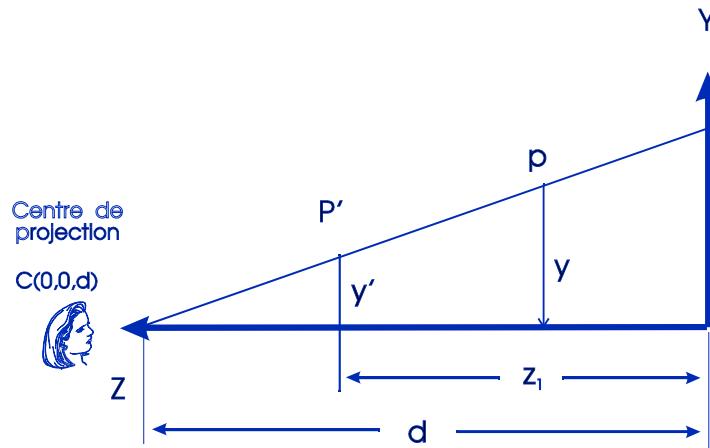
Projection perspective

- Une projection perspective peut contenir plusieurs points de fuite.

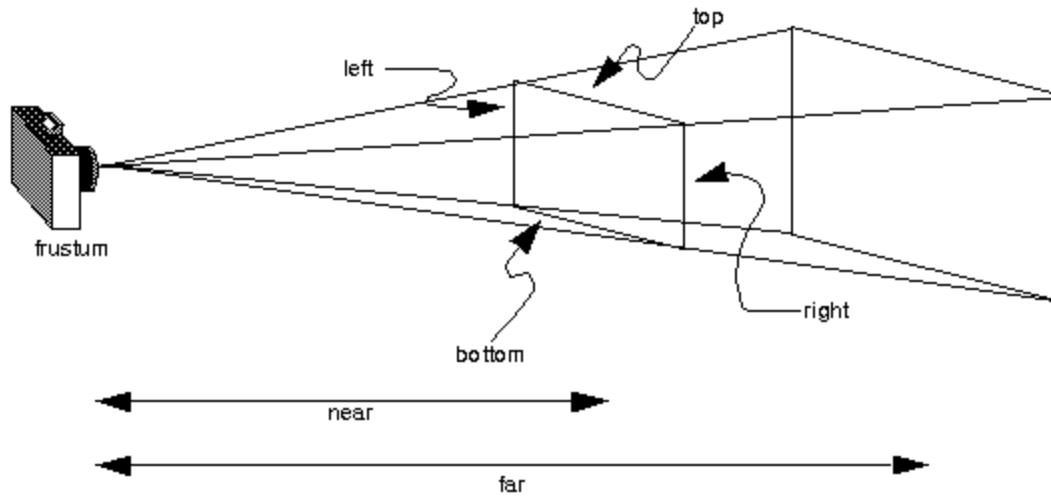


Projection perspective

- But: obtenir un meilleur réalisme
- Projection en fonction de z (distance à l'observateur)
- On trace des lignes de projection de l'observateur à l'objet.
- Les lignes de projection convergent vers l'observateur, en passant par un plan de projection.
- L'intersection de ces lignes avec le plan de projection formeront la projection de l'objet.

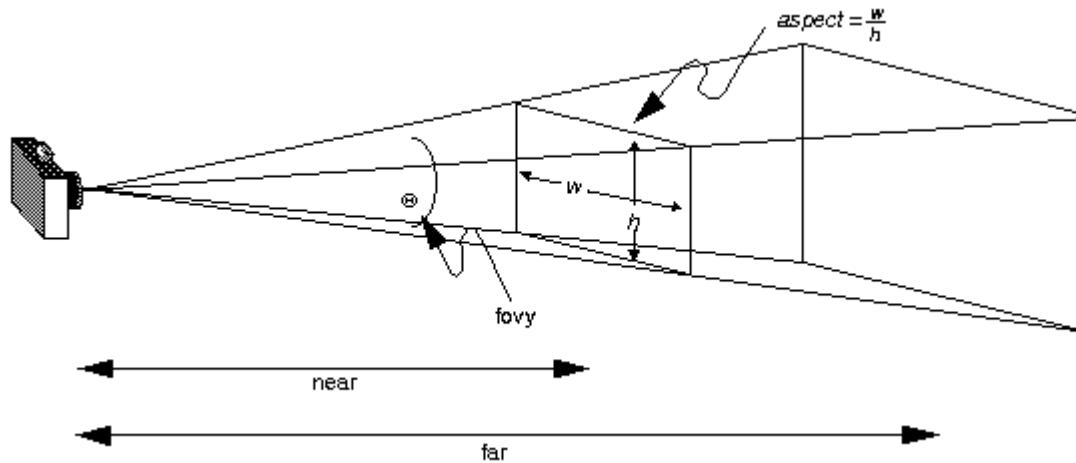


Projection perspective (OpenGL)



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

Projection perspective (OpenGL)



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

Projection perspective (OpenGL)

Les matrices générées par les deux types de projection sont semblables, mais différentes:

glOrtho()

Projection orthographique

$$R = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & \frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & \frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & \frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

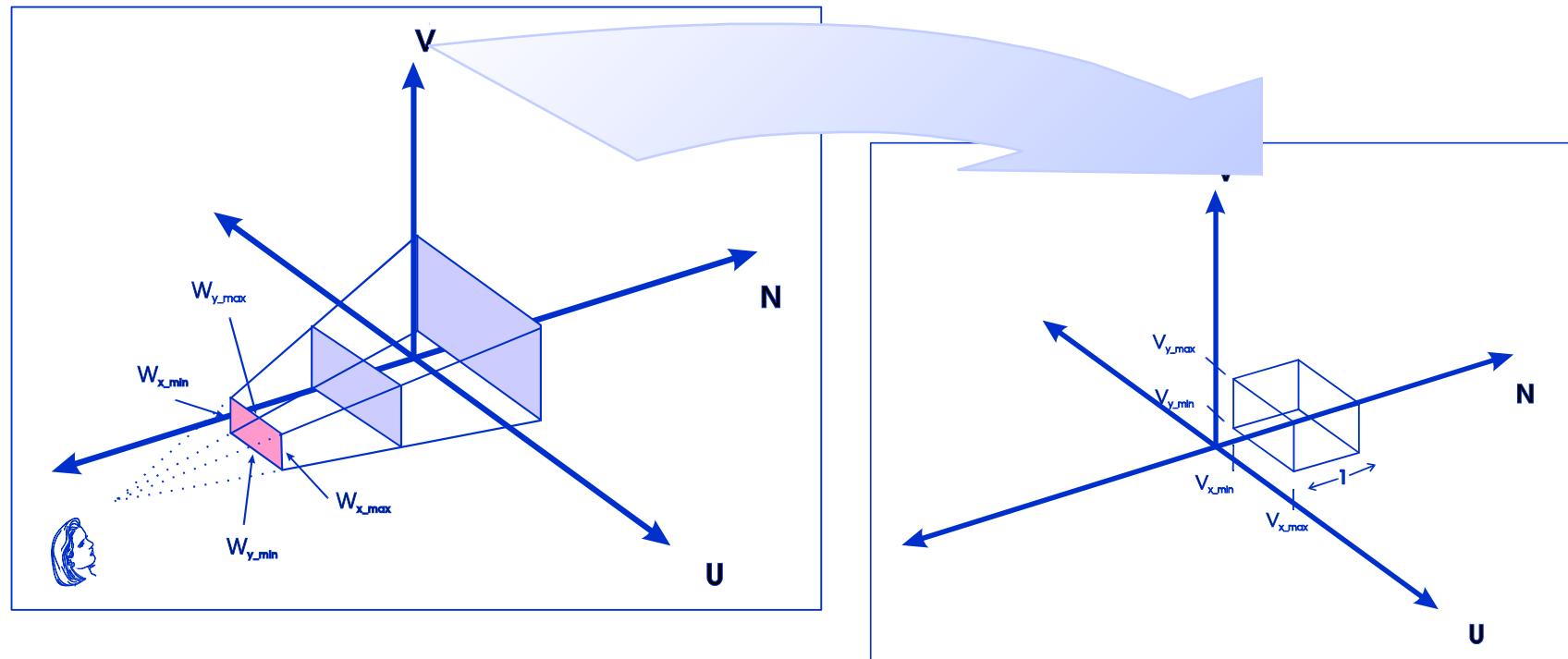
glFrustum()

Projection perspective

$$R = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

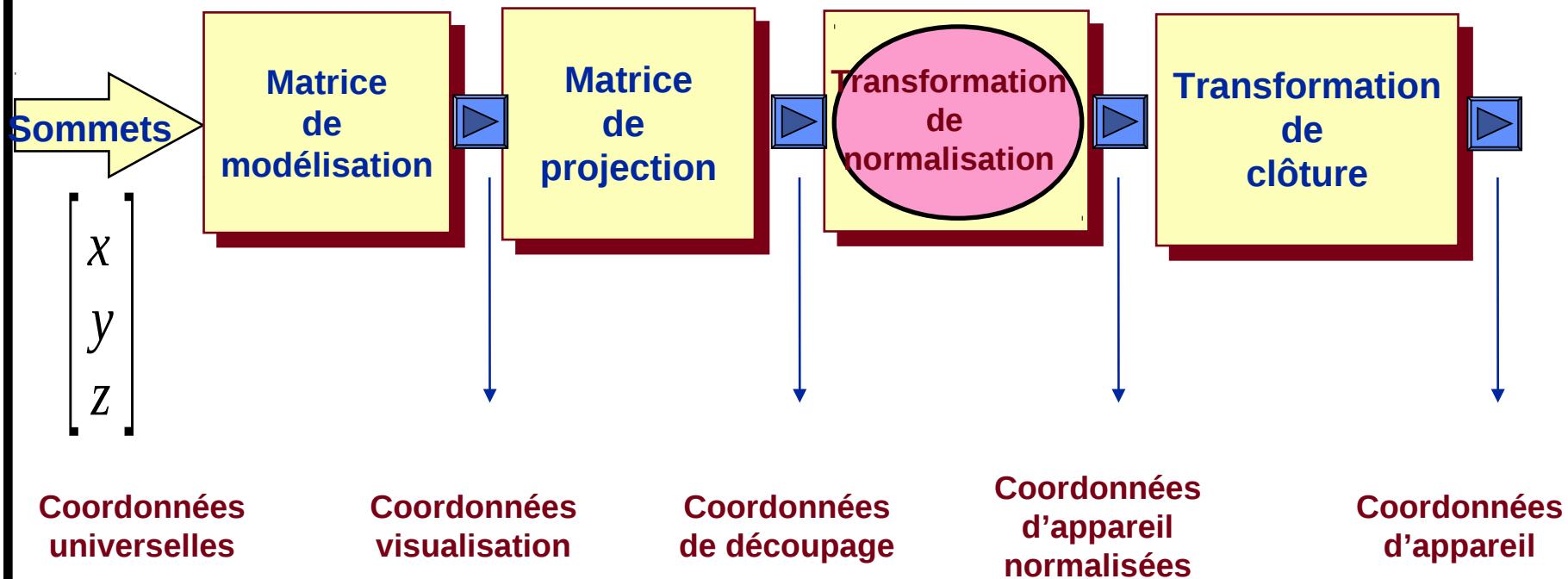
Normalisation du volume de visualisation

- La transformation de projection *normalise* le volume de visualisation vers l'intervalle $[0,1] \times [0,1] \times [0,1]$ (un cube)
- La matrice de projection perspective *déforme* le volume, introduisant ainsi un effet de raccourci



PIPELINE DES TRANSFORMATIONS

OpenGL



Conversion au monde d'affichage

- La dernière transformation du pipeline applique le volume normalisé dans la clôture (en pixels):

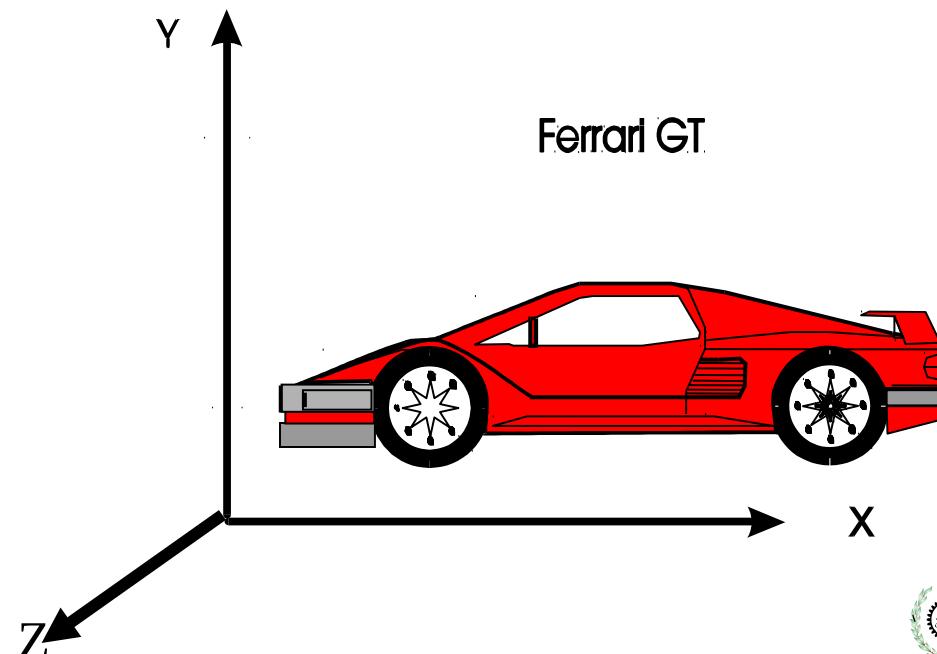
```
glViewport( GLint x, GLint y,  
           GLsizei largeur, GLsizei hauteur);
```

où (x,y) sont les coordonnées du coin inférieur gauche de la clôture.

Fenêtre virtuelle, clôture et rapport d'aspect

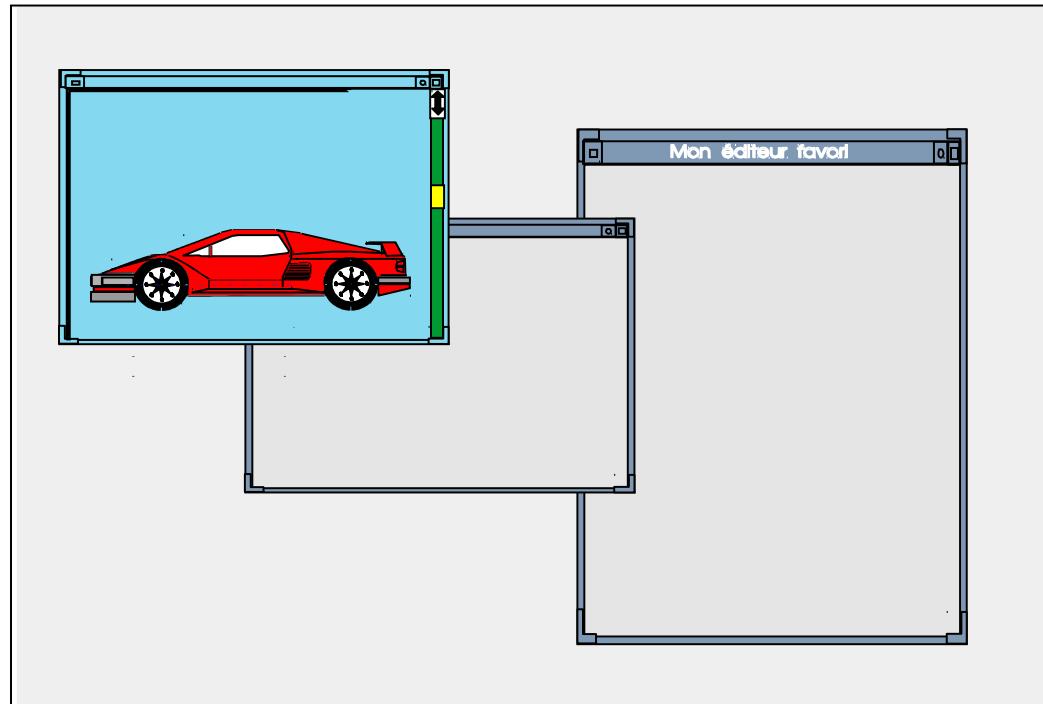
Espaces virtuel et d'affichage

- **objet virtuel** : modélisation partielle de l'objet réel (géométrie, apparence)
- **espace virtuel** : dans lequel s'exprime la codification de l'objet virtuel



Espaces virtuel et d'affichage

- **objet d'affichage** : le rendu graphique de l'objet virtuel à l'écran
- **espace d'affichage** : écran (2D) d'une certaine résolution ou même une portion de cet écran



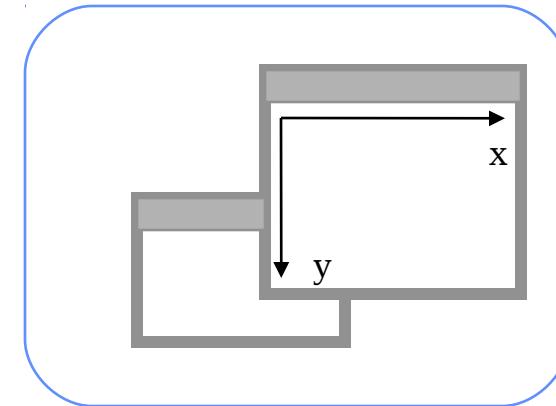
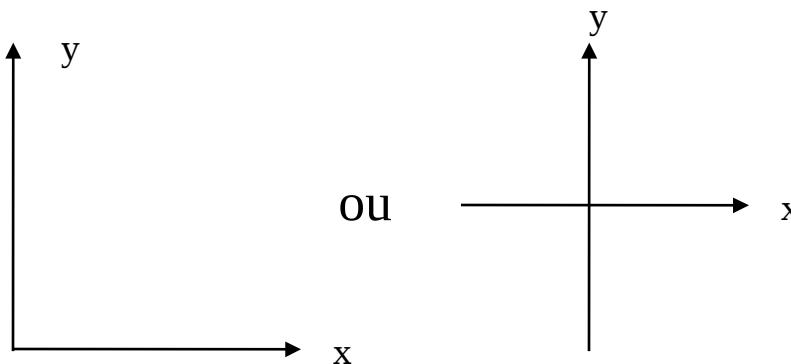
Espaces virtuel et d'affichage

- **espace virtuel**

- dimension arbitraire
- infini
- continu

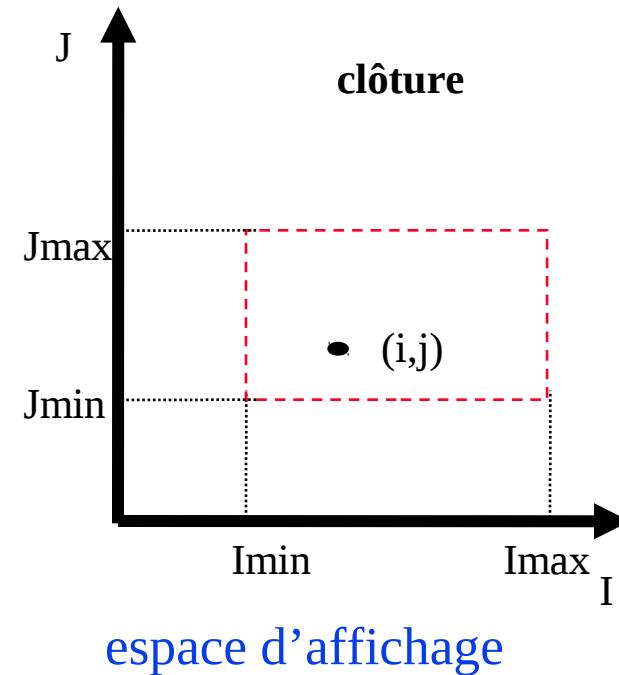
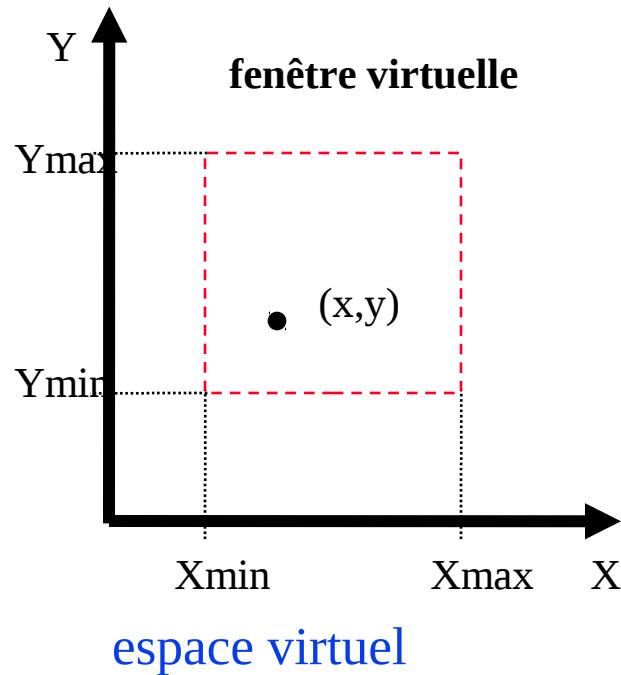
- **espace d'affichage**

- dimension 2
- fini (limité par la résolution)
- discret (modèle pixel)



Réalisation d'une image

- affichage d'un point (x, y)
 - On veut faire correspondre à ce point (x, y) dans l'espace virtuel un pixel (i, j) dans l'espace d'affichage



Réalisation d'une image

- affichage d'un point (x, y) (suite)
 - De part et d'autre, on normalise fenêtre virtuelle et clôture.

$$[X_{\min}, X_{\max}] \times [Y_{\min}, Y_{\max}] \rightarrow [0,1] \times [0,1]$$

$$(x,y) \rightarrow \left[\frac{x - X_{\min}}{X_{\max} - X_{\min}}, \frac{y - Y_{\min}}{Y_{\max} - Y_{\min}} \right]$$

normalisation de la fenêtre virtuelle

$$[I_{\min}, I_{\max}] \times [J_{\min}, J_{\max}] \rightarrow [0,1] \times [0,1]$$

$$(i,j) \rightarrow \left[\frac{i - I_{\min}}{I_{\max} - I_{\min}}, \frac{j - J_{\min}}{J_{\max} - J_{\min}} \right]$$

normalisation de la clôture



Réalisation d'une image

- affichage d'un point (x, y) (suite)
 - Les coordonnées normalisées devraient être identiques.

$$\left[\frac{x - X_{\min}}{X_{\max} - X_{\min}}, \frac{y - Y_{\min}}{Y_{\max} - Y_{\min}} \right] = \left[\frac{i - I_{\min}}{I_{\max} - I_{\min}}, \frac{j - J_{\min}}{J_{\max} - J_{\min}} \right]$$

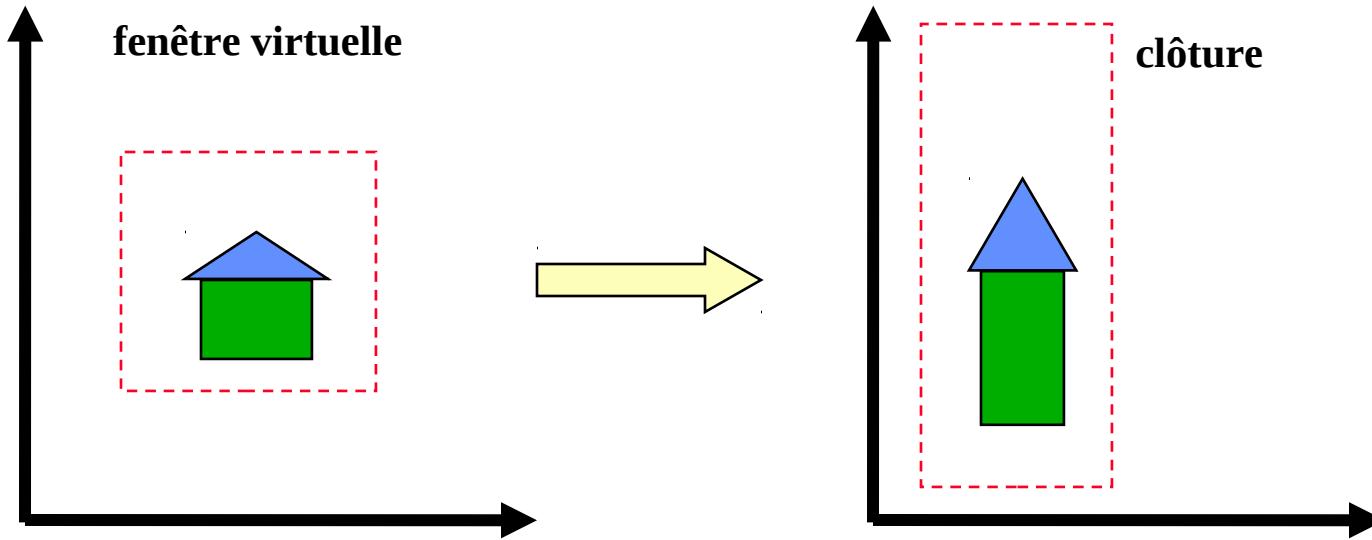
- virtuel \rightarrow affichage

$$i = I_{\min} + \frac{x - X_{\min}}{X_{\max} - X_{\min}} (I_{\max} - I_{\min})$$

$$j = J_{\min} + \frac{y - Y_{\min}}{Y_{\max} - Y_{\min}} (J_{\max} - J_{\min})$$

- affichage \rightarrow virtuel
 - on peut dériver à partir de la même équation

Réalisation d'une image



le rapport d'aspect est respecté si

$$\frac{X_{\max} - X_{\min}}{Y_{\max} - Y_{\min}} = \frac{I_{\max} - I_{\min}}{J_{\max} - J_{\min}}$$

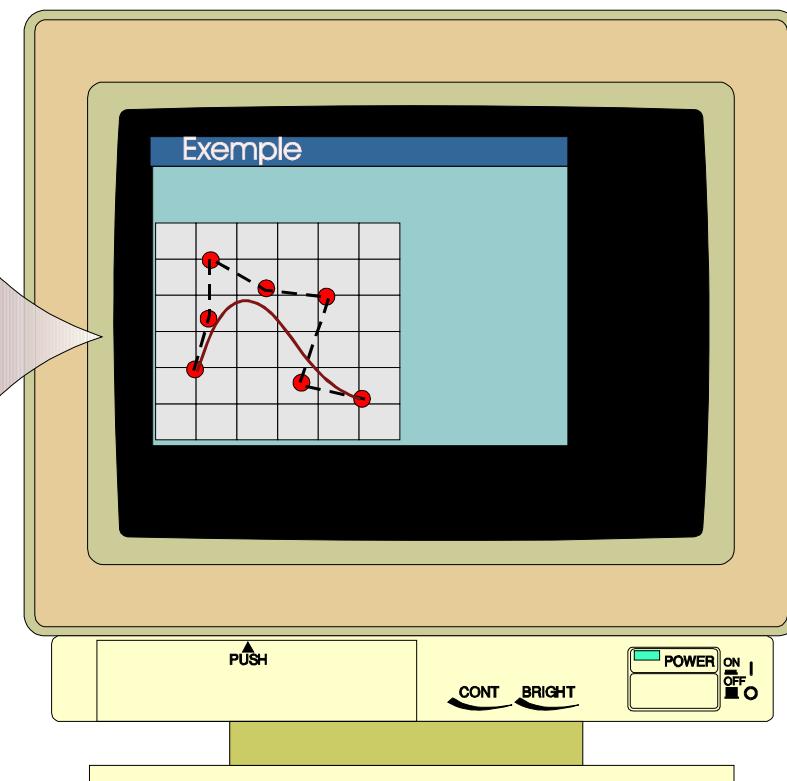
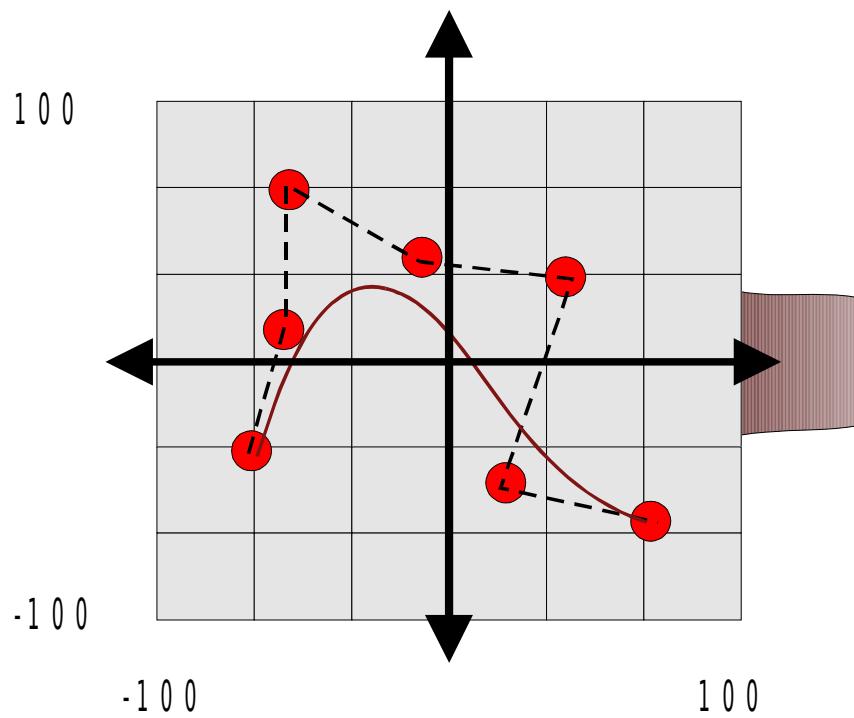
ÉTABLISSEMENT D'UNE FENÊTRE 2D ET D'UNE CLÔTURE

Fonctions OpenGL

```
GLDouble    left = -100.0,  
            right = 100.0,  
            bottom = -100.0,  
            top = 100.0;
```

```
glMatrixMode( GL_PROJECTION ); // pour la matrice de projection  
glLoadIdentity( );  
gluOrtho2D( left, right, bottom, top );  
glMatrixMode( GL_MODELVIEW );  
glViewport( 0, 0, 200, 200 );
```

ÉTABLISSEMENT D'UNE FENÊTRE 2D ET D'UNE CLÔTURE

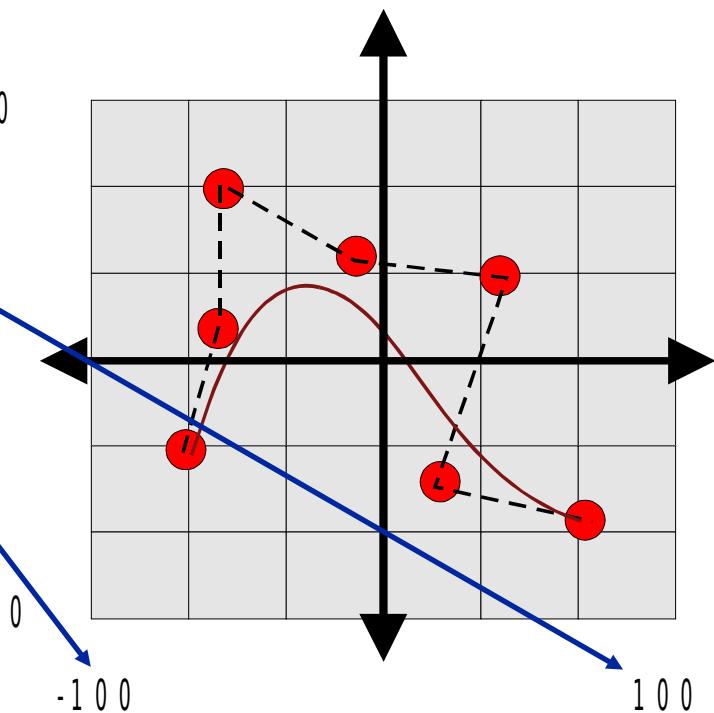


ÉTABLISSEMENT D'UNE FENÊTRE 2D ET D'UNE CLÔTURE

GLDouble

```
left = -100.0,  
right = 100.0,  
bottom = -100.0,  
top = 100.0;
```

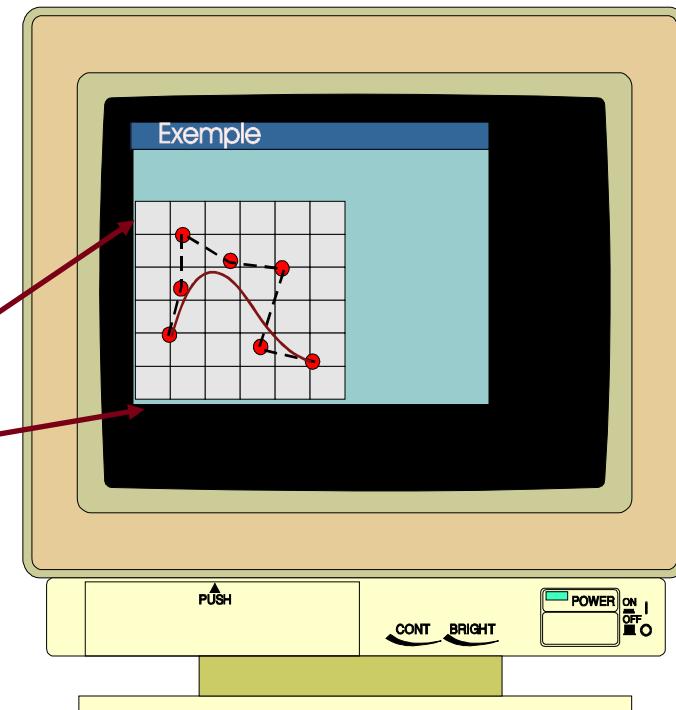
```
glMatrixMode( GL_PROJECTION );  
glLoadIdentity( );  
gluOrtho2D( left, right, bottom, top );  
glMatrixMode( GL_MODELVIEW );  
glViewport( 0, 0, 200, 200 );
```



ÉTABLISSEMENT D'UNE CLÔTURE

```
GLDouble left = -100.0,  
right = 100.0,  
bottom = -100.0,  
top = 100.0;
```

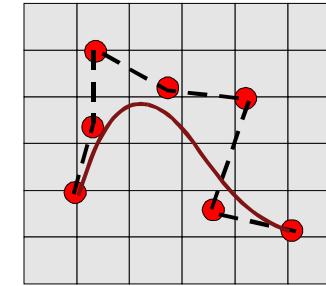
```
glMatrixMode( GL_PROJECTION );  
glLoadIdentity( );  
gluOrtho2D( left, right, bottom, top );  
glMatrixMode( GL_MODELVIEW );  
glViewport( 0, 0, 200, 200 );
```



ÉTABLISSEMENT D'UNE FENÊTRE 2D ET D'UNE CLÔTURE

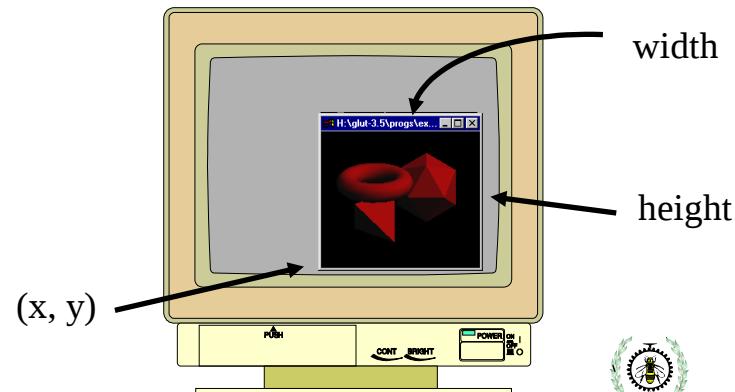
```
void gluOrtho2D( GLdouble left, GLdouble right,  
GLdouble bottom, GLdouble top );
```

(Right, top)



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

(Left, bottom)

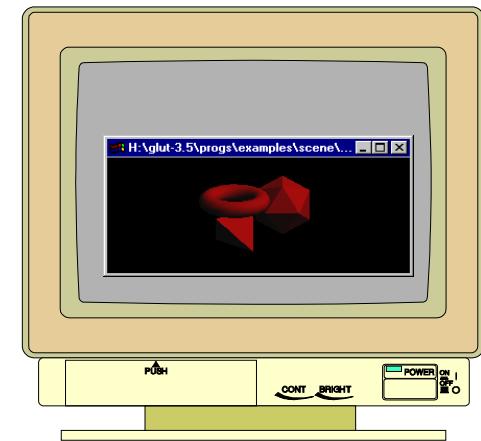


MAINTENIR LE RAPPORT D'ASPECT

Rapport de correction = $\frac{\text{Largeur}}{\text{Hauteur}} = \frac{w}{h}$

- Éviter de déformer l'image:

$$\frac{(x_{\max} - x_{\min})}{(y_{\max} - y_{\min})} = \frac{\Delta x}{\Delta y} = \frac{\Delta I}{\Delta J} = \frac{(I_{\max} - I_{\min})}{(J_{\max} - J_{\min})}$$

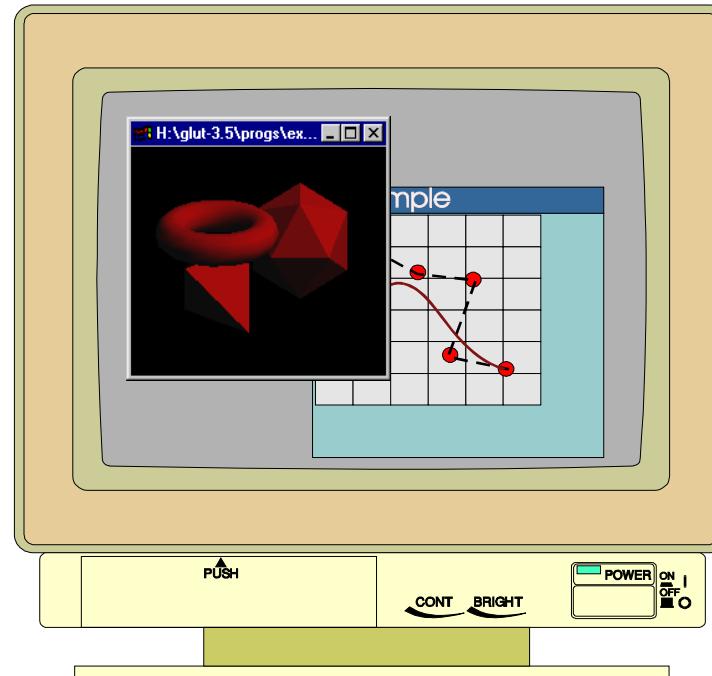


MAINTENIR LE RAPPORT D'ASPECT

- Où établir la fenêtre virtuelle et la clôture?

```
int main( int argc, char **argv )  
{  
    glutInit(&argc, argv);  
    glutInitWindowPosition(500, 500);  
    glutInitWindowSize(500, 500);  
    glutCreateWindow(argv[0]);  
    myinit();  
    glutReshapeFunc(myReshape);  
    glutDisplayFunc(display);  
    glutMainLoop();  
}
```

- Habituellement dans la fonction de rappel pour redimensionner la fenêtre (**glutReshapeFunc(...)**)



MAINTENIR LE RAPPORT D'ASPECT

```
void myReshape(int w, int h)
{
    glViewport( 0, 0, w, h );
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    // utiliser valeurs courantes
    GLdouble dx = Droite-Gauche, dy = Haut-Bas;
    if ( dx/dy < w/h ) // <=> if ( dx*h < dy*w )
    {
        GLdouble ajout = ... ;
        Gauche -= 0.5*ajout;
        Droite += 0.5*ajout;
    }
    else
    {
        GLdouble ajout = ... ;
        Bas -= 0.5*ajout;
        Haut += 0.5*ajout;
    }
    // donner nouvelles valeurs
    gluOrtho2D( Gauche, Droite, Bas, Haut );
    glMatrixMode( GL_MODELVIEW );
}
```

