Visualisation

Ahlem Bougarradh

Introduction

- L'ensemble des transformations de la visualisation peut être comparé à un appareil photo ou à une caméra.
  - Placer la caméra à une position (transformation de visualisation).
  - Construire le décor et le déplacer (transformation de modélisation)
  - Ajuster le zoom (transformation de projection) pour définir quel volume vous prenez en photo.
- Quatre transformations successives utilisées au cours du processus de création d'une image:

Introduction

(1) Transformation de modélisation (Model)

Permet de créer la scène à afficher par création, placement et orientation des objets qui la composent.

(2) Transformation de visualisation (View)

Permet de fixer la position et l'orientation de la caméra de visualisation.

(3) Transformation de projection (Projection)

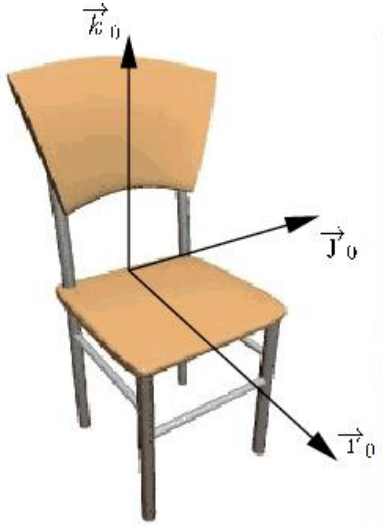
Permet de fixer les caractéristiques optiques de la caméra de visualisation (type de projection, ouverture, ...).

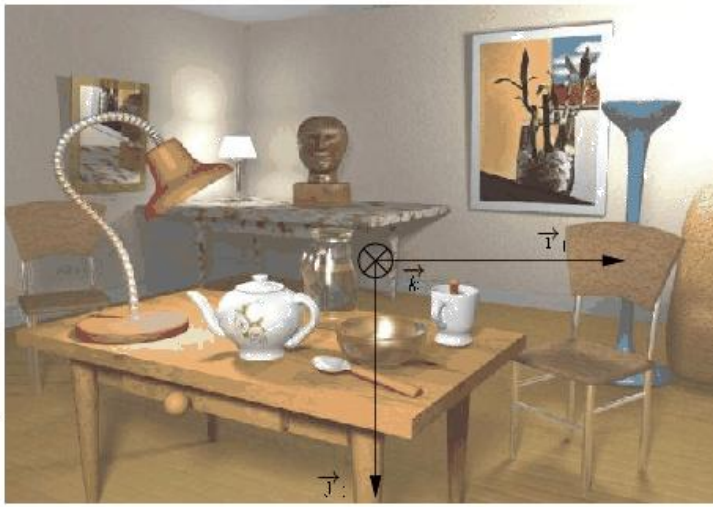
(4) Transformation d'affichage (Viewport)

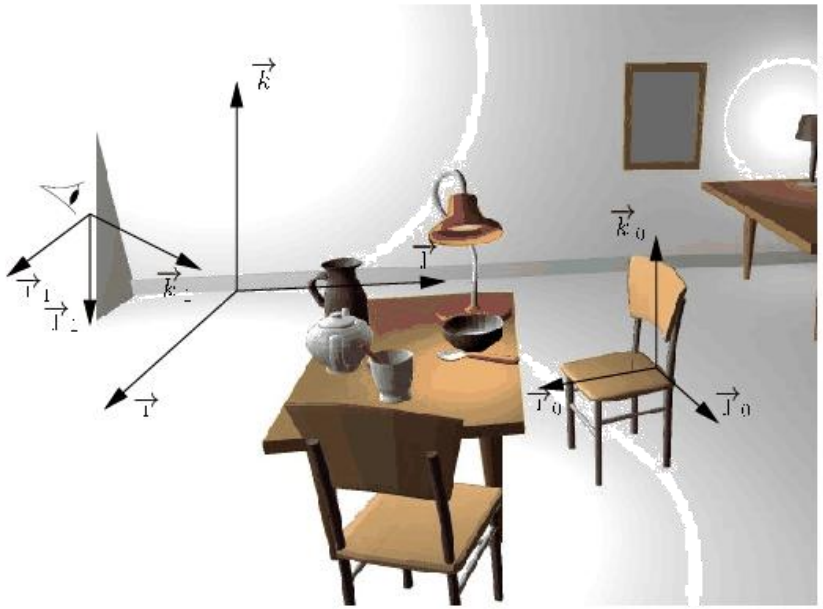
Permet de fixer la taille et la position de l'image sur la fenêtre d'affichage.

Scène 3D: Repères

- Un objet 3d est défini dans l'espace par les coordonnées de ses points dans plusieurs repères :
  - **Le repère global** qui est fixe et qui sert de repère général à la scène (Og,Xg,Yg,Zg).
  - **Le repère objet (ou local)** qui est un repère propre à l'objet grâce auquel sont exprimées ses coordonnées (Oo,Xo,Yo,Zo). En général, l'origine est le centre de l'objet.
  - **Le repère camera** semblable à un repère objet mais pour la caméra (c'est à dire le point à partir duquel on regarde la scène).
  - **Le repère écran** qui représente le repère défini par le moniteur :il correspond à la surface de l'image finale et ne possède que deux coordonnées car il est en deux dimensions (abscisse et ordonnée Xe et Ye). L'origine de ce repère est définie en haut à gauche de l'écran.







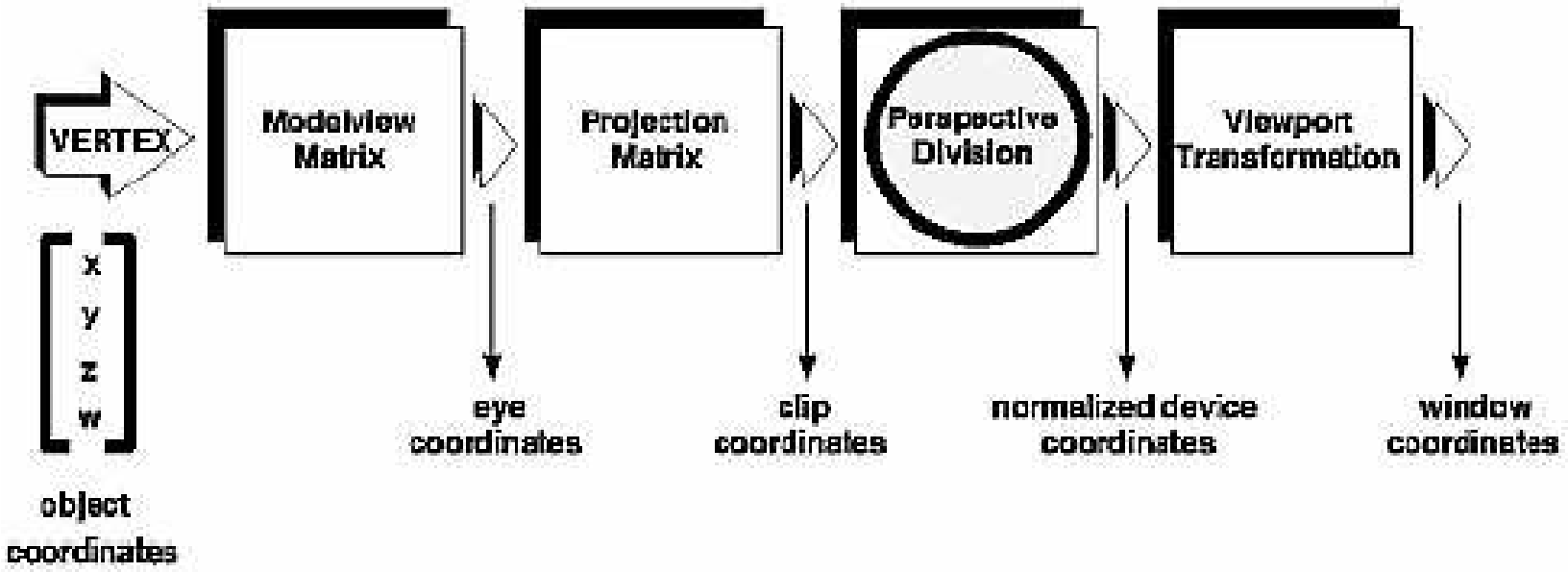
Les étapes de Visualisation

- Avant d'être affiché à l'écran, un objet subit plusieurs transformations:
- La transformation de modélisation : les objets graphiques sont définis dans un repère où ils sont simples à concevoir, puis on transforme ces objets par des rotations, translations et changements d'échelle (affinités) pour les positionner dans le repère du monde.
  - On fait subir ces transformations aux objets au moyen des fonctions glRotatef, glTranslatef et glScalef dans le mode GL_MODELVIEW.
- La transformation de la caméra : une fois dans le repère du monde, tous les objets subissent un changement de repère vers le repère de la caméra.
  - On fait subir cette transformation aux objets au moyen de la fonction gluLookAt, dans le mode GL_MODELVIEW.

Les étapes de Visualisation

- Projection : une fois dans le repère de la caméra, les objets subissent une projection vers une image (dans un buffer OpenGL).
  - Pour régler les paramètres de cette projection, on utilise une fonction de projection dans le mode GL_PROJECTION.
- Transformation d'affichage : une fois générée l'image dans un buffer, elle subit des changement d'échelle sur les axes (affinités orthogonales) pour être ajustée aux dimensions de la fenêtre graphique à l'écran.
  - Cette dernière transformation est paramétrée par la fonction glViewport.

Etapes de visualisation

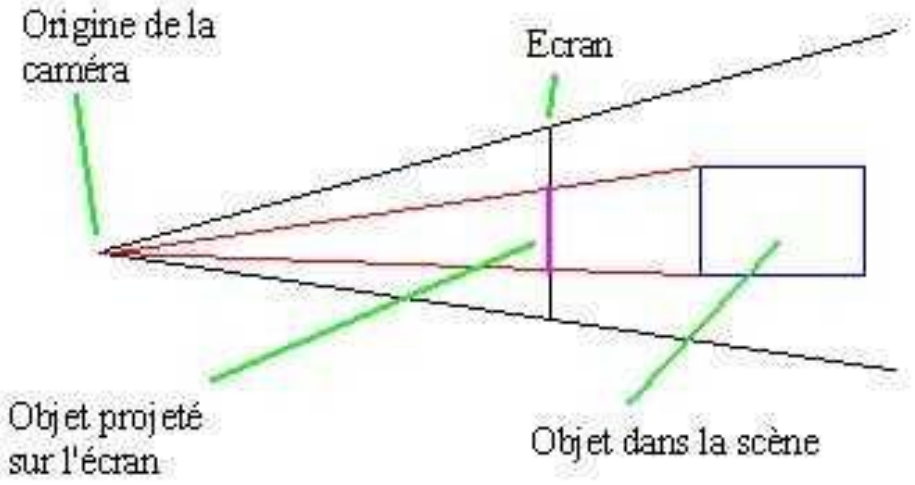


Transformation de visualisation (View)

- L’obtention d’un point de vue différent de la scène s’effectue par des modifications des paramètres de la caméra
- Le point d'origine est défini par l'utilisateur selon l'endroit où il veut placer la caméra dans la scène. Cette position est en un point (x,y,z) défini dans le repère absolu.
- Le vecteur direction indique vers quel endroit porte le regard dans la scène. Le vecteur Vup (up pour haut) permet de connaître la position verticale du repère caméra.

Transformation de visualisation (View)

- La caméra nous indique d'où voir la scène (origine de la caméra), l'angle de vision de la scène. Elle peut être fixe ou en mouvement (lors d'animation).



L'observateur avec glutLookAt

- **void gluLookAt (GLdouble ex, GLdouble ey, GLdouble ez, GLdouble cx, GLdouble cy, GLdouble cz, GLdouble upx, GLdouble upy, GLdouble upz);**

- La fonction gluLookAt permet de passer du système de coordonnées du monde au système de coordonnées de l'observateur (caméra).

- Compose la transformation courante (généralement MODELVIEW) par la transformation donnant un point de vue depuis (ex,ey,ez) avec une direction de visualisation passant par (cx,cy,cz).
- Le vecteur (upx,upy,upz) : doit apparaître vertical dans l'image obtenue par projection, qui devient la direction y (0.0, 1.0, 0.0) dans le repère écran.
- gluLookAt est une fonction de la bibliothèque GLU.

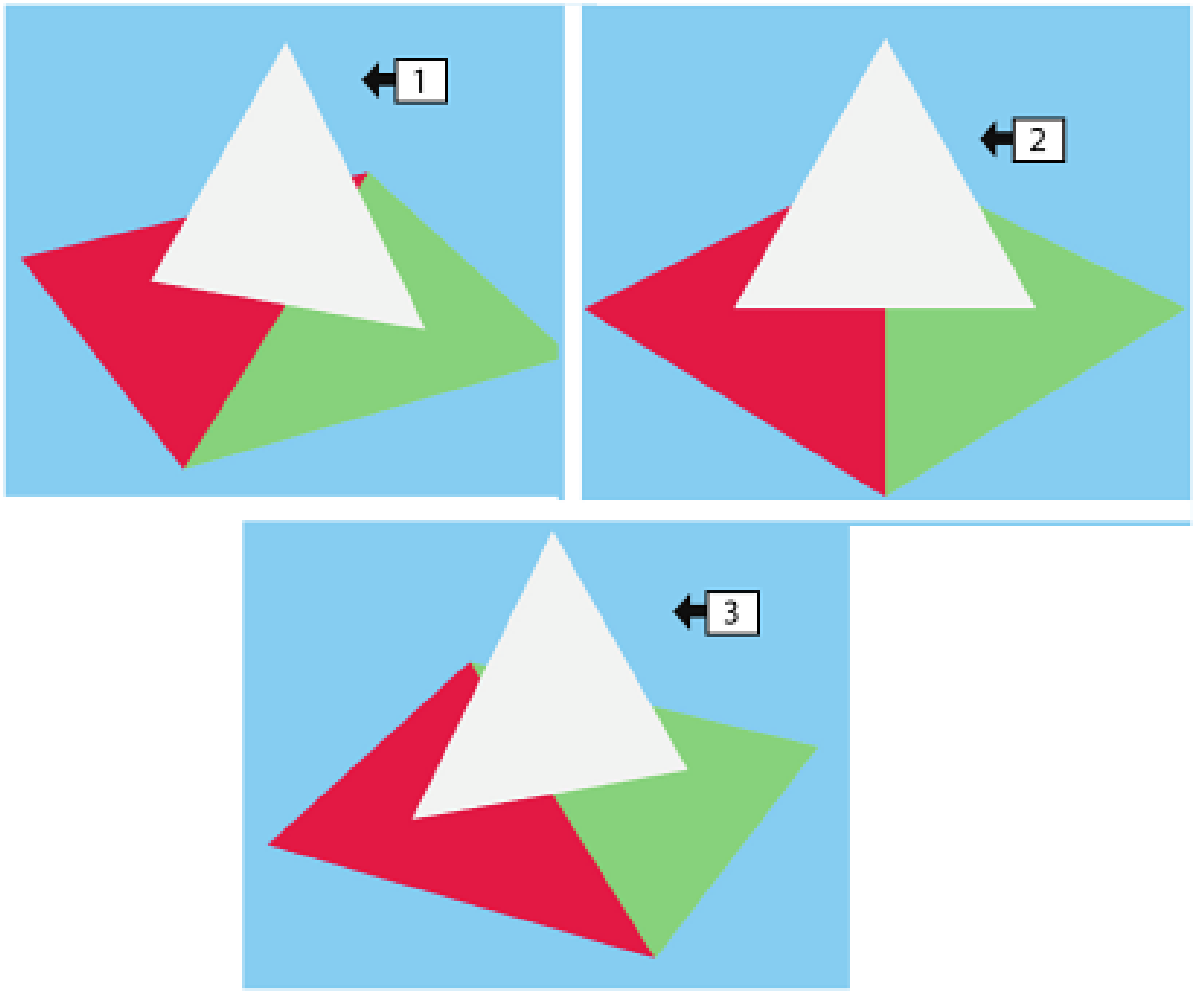
L'observateur avec glutLookAt

- Exemple:

  gluLookAt(10.0,15.0,10.0,3.0,5.0,-2.0,0.0,1.0,0.0): place la caméra en position (10.0,15.0,10.0), l'oriente pour qu'elle vise le point (3.0,5.0,-2.0) et visualisera la direction (0.0,1.0,0.0) de telle manière qu'elle apparaisse verticale dans la fenêtre de visualisation. Ces valeurs sont considérées dans le repère global.

- -> gluLookAt doit être exécuté en mode MODELVIEW et doit être placé avant la création de la scène.

Exemple



Projection

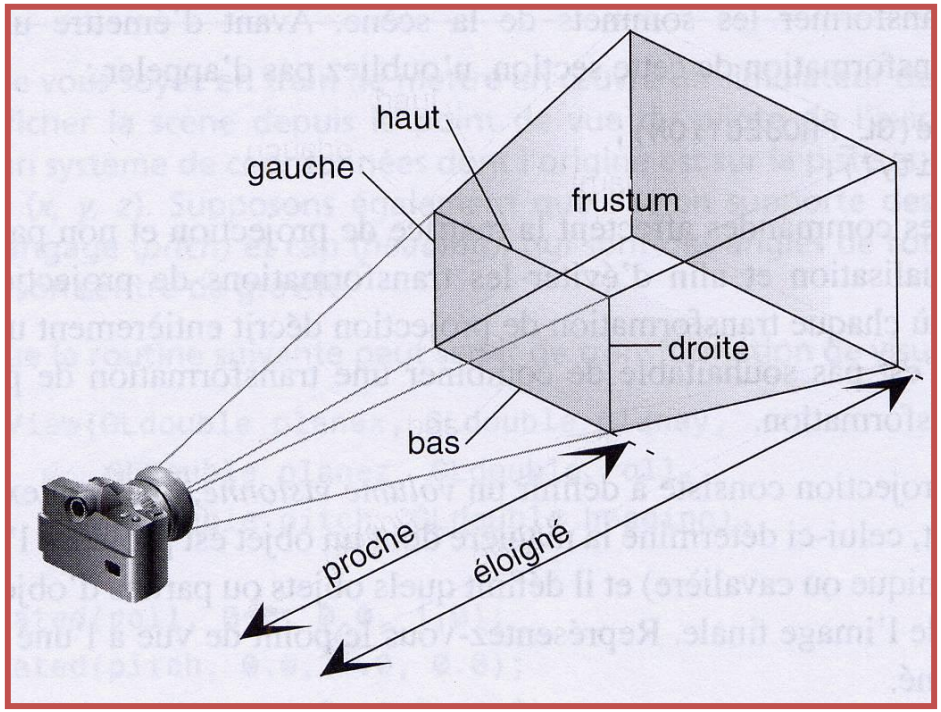
- L’objectif de la projection consiste:
- Définir un volume visionné (Le volume de visualisation) pour définir la manière dont l’objet est projeté à l’écran:
  - Projection en perspective
  - la projection parallèle orthographique
- Définir quel s objets ou parties d’objets sont clippés pour disparaitre de l’image finale.
- Charger la matrice de projection :glMatrixMode(GL_PROJECTION).
- Le rôle d'une matrice de projection est de définir la perspective d'un plan, la façon dont sera projeté une scène du point de vue de la caméra.

Transformation spécifique à la projection(1)

- Projection en perspective: définit un frustum (tronc) de pyramide qui est le champ de vision.
- Plus un objet est éloigné de la caméra, plus il est petit sur l'image finale.
- glFrustum :définit une matrice de projection perspective, à partir du cadre image (gauche, droit, haut, bas), de la focale (plan avant) et du plan arrière.

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

Transformation spécifique à la projection(1)

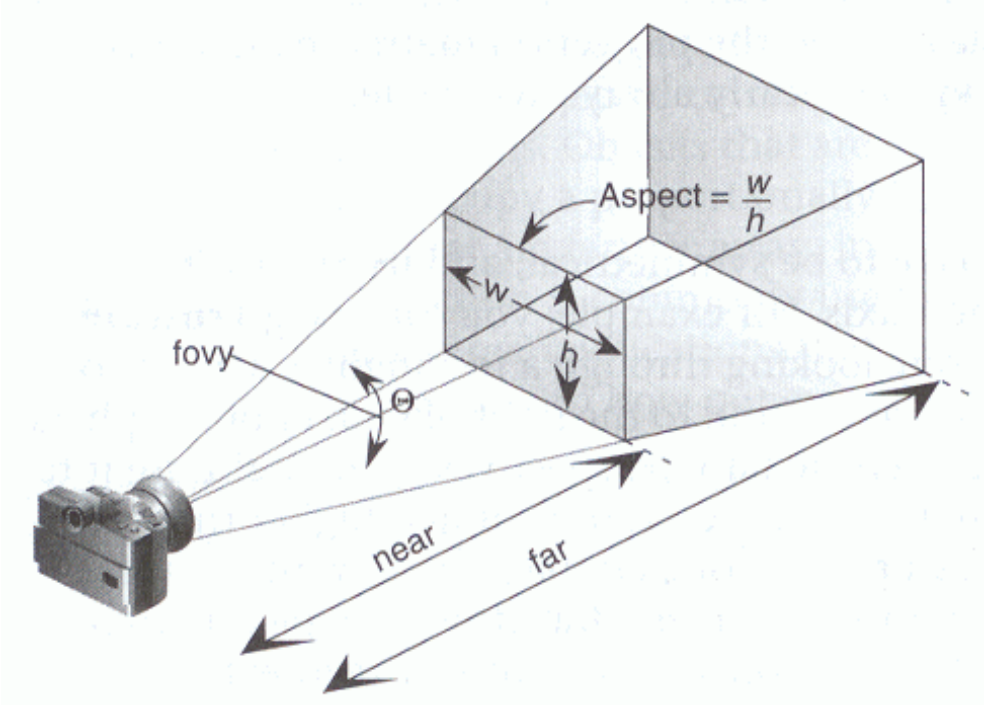


Les 2 plans de découpage sont parallèles. Les distances **proche** et **eloigne** doivent être positives; elles représentent la distance du centre de projection à ces plans.

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(gauche, droite, bas, haut, proche, eloigne);
```

Transformation spécifique à la projection(2)

- Dans plusieurs applications, il est plus naturel d'exiger l'angle de vue.
- gluPerpective a la même fonction, en donnant rapport hauteur/largeur, angle de champ, plans avant et arrière.



Transformation spécifique à la projection(2)

- La fonction :

– **void gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble near, GLdouble far);**

– Compose la transformation courante par la transformation de projection en perspective de volume de visualisation défini par la pyramide tronquée de sommet l'origine O, orientée selon l'axe -z, possédant:

  – **fovy** : désigne l'angle du champ de vision dans le plan yz entre 0.0 et 180.0,

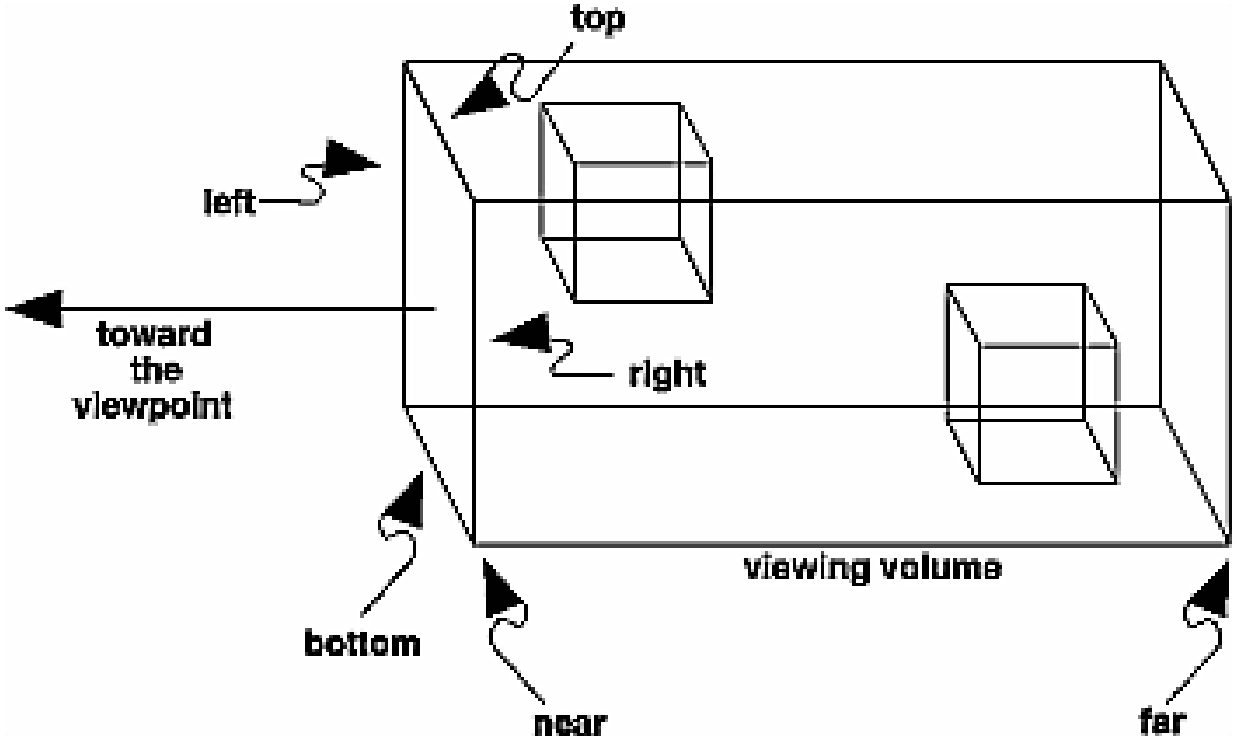
  – **aspect** : désigne le rapport largeur / hauteur de la région considérée du plan de vue,

  – **near** et far :représentent la distance entre le centre de projection et les 2 plans de découpage, le long de l'axe des z négatif. Ces distances doivent toujours être positives.

Transformation spécifique à la projection(2)

- **La** projection orthographique définie un volume de visualisation parallélépipèdique rectangle .
- La taille du volume de visualisation ne change pas d'un bout à l'autre.
- La distance par rapport au point de vue n'affecte pas la taille des objets.
- **void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);**
- **glOrtho:** Compose la transformation courante par la transformation de projection orthographique selon l'axe -z et définie par le volume de visualisation parallélépipèdique (left,right,bottom,top,near,far).
- glOrtho définit une matrice de projection orthogonale

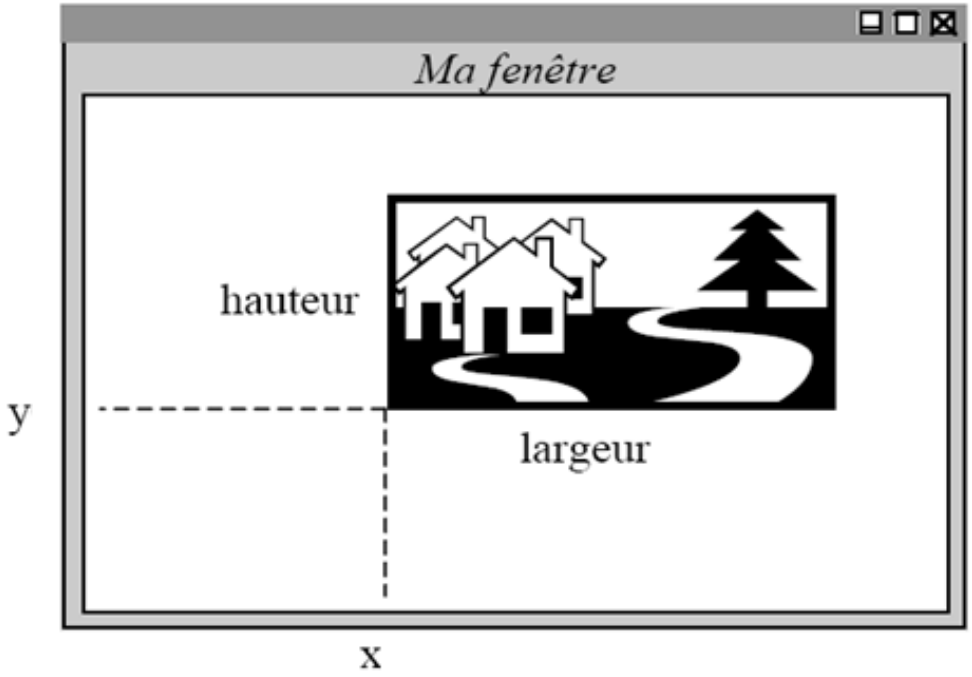
Transformation spécifique à la projection(3)



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

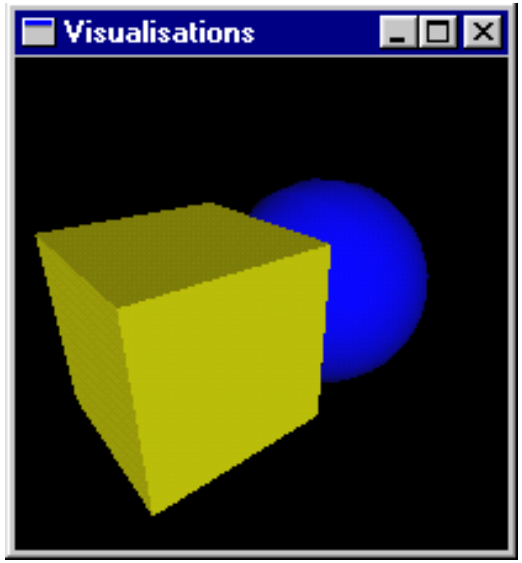
Transformation d'affichage

- Définit le rectangle de pixels dans la fenêtre d'affichage dans lequel l'image calculée sera affichée.
- **void glViewport(GLint x,GLint y,Glsizei l,GLsizei h);**

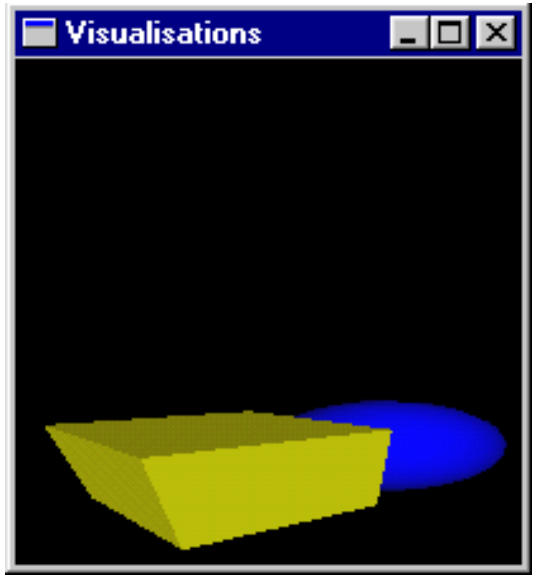


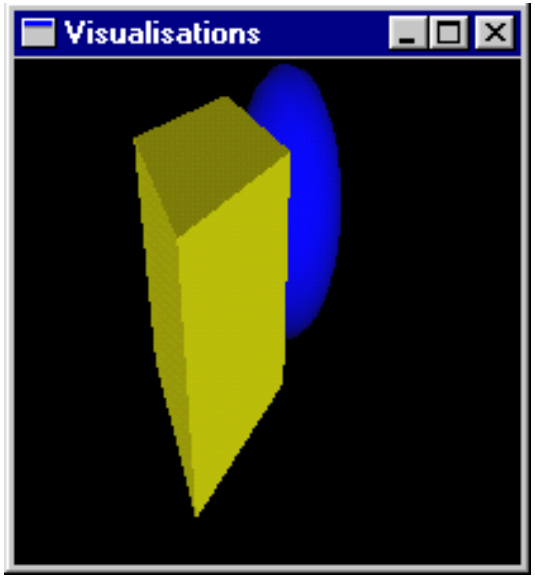
Transformation d'affichage

- **void glViewport(GLint x,GLint y,GLsizei l,GLsizei h);**



Toute la fenêtre





Une partie seulement

Transformation d'affichage

glutReshapeFunc (void (*func)(int w, int h)**)**

➡action à exécuter quand la fenêtre est redimensionnée
Apprtient à la bibliothèque GLUT,
Appelée dans la fonction main

Ex :
glutReshapeFunc (refenetrer);

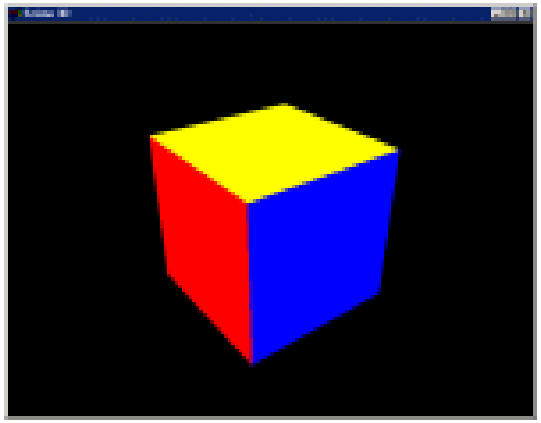
```
void refenetrer (int  w, int h)
{
…
}
```

Transformation d'affichage

```
void refenetrer( i n t  width , i n t  h e i g h t )
 {
g lVi ewp o r t ( 0 , 0 , ( GLint ) width , ( GLint ) h e i g h t) ;
glMat rixMode (GL PROJECTION ) ;
g l L o a d I d e n t i t y ( ) ;
g l u P e r s p e c t i v e ( 6 0 . 0 , width /he i ght , 1 . 0 , 2 0 . 0 ) ;
glMat rixMode (GL MODELVIEW) ;
}
```

Exemple: cube
Mise en place du buffer de profondeur

1. Mettre en place la scène
    - 1.1 Placer les objets
        - 1.1.1 Définir le cube
        - 1.1.2 Attribuer une couleur à chaque face
    - 2. Configurer la vue
    - 3. Placer la caméra
    - 4. Visualiser résultat intermédiaire
5. Mise en place d'un Z-Buffer



Buffer de profondeur(Z Buffer)

- Pour résoudre ce problème de faces cachées, OpenGL propose une technique extrêmement répandue en synthèse d'images : le tampon de profondeur (Z-buffer).
- L'idée principale est de créer en plus de notre image un tampon de même taille.
- A chaque fois qu'on dessine un polygone, pour chaque pixel qui le constitue, on calcule sa profondeur et on la compare avec celle qui est déjà stockée dans le tampon.
- Si la profondeur stockée dans le tampon est supérieure à celle du polygone qu'on est en train de traiter, alors le polygone est plus proche (au point considéré) de la caméra que les objets qui ont déjà été affichés.
- Le point du polygone est affiché sur l'image, et la profondeur du pixel est stockée dans le tampon de profondeur.

Buffer de profondeur(Z Buffer)

- A chaque fois qu'on dessine un polygone, pour chaque pixel qui le constitue, on calcule sa profondeur et on la compare avec celle qui est déjà stockée dans le tampon.
- Si la profondeur stockée dans le tampon est supérieure à celle du polygone qu'on est en train de traiter, alors le polygone est plus proche (au point considéré) de la caméra que les objets qui ont déjà été affichés.
- Le point du polygone est affiché sur l'image, et la profondeur du pixel est stockée dans le tampon de profondeur.
-

Buffer de profondeur(Z Buffer)

- Première chose à faire, il faut modifier l'appel à la fonction glut d'initialisation de l'affichage pour lui indiquer qu'il faut allouer de l'espace pour le tampon de profondeur :
  - **glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH);**

- L'utilisation ou non du tampon de profondeur est gérée par une variable d'état OpenGL : **GL_DEPTH_TEST.**

- L'activation du tampon de profondeur se fait en plaçant la ligne suivante dans la phase d'initialisation du programme :
  - **glEnable(GL_DEPTH_TEST);**

Buffer de profondeur(Z Buffer)

- Tout comme le tampon image, il faut effacer le tampon de profondeur à chaque fois que la scène est redessinée.

- Il suffit de modifier l'appel glClear() de la fonction d'affichage :

  - **glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);**

Le Double Buffer

- Le double buffer est une technique ayant pour but l'amélioration de la qualité et de la vitesse d'affichage lors de la réalisation animations.
- En l'absence du double-buffering, une image en cours de calcul peut être affichée au cours d'une animation faisant alors clairement apparaître la génération progressive du résultat final.

- Avec le double-buffering, l'image est calculée dans un premier tampon mémoire, tandis qu'un second tampon est envoyé à l'écran.
- Lorsque l'image est terminée, les deux tampons sont inversés de manière que l'image finie soit envoyée à l'écran et que l'ancien tampon écran puisse être utilisé pour calculer l'image suivante de l'animation.

Le Double Buffer

- Glut permet de gérer un double tampon de façon extrêmement simple.
- Tout d'abord, comme pour le tampon de profondeur, il faut indiquer au système que l'on souhaite utiliser un double tampon d'image. Nous modifions donc à nouveau l'appel à glutInitDisplayMode() :

- **glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);**

- Il ne reste plus qu'à indiquer à glut le moment opportun pour l'échange des buffers : la fin du dessin de la scène. On place à cet effet un appel à **glutSwapBuffers()** à la fin de notre fonction d'affichage.

Le Double Buffer