Cours synthèse d'images en temps réel

Synthèse d'images animées

- Audiovisuel
- Effets spéciaux
- Jeux vidéos
- Etudes d'impact
- Simulateurs
- Visualisation scientifique

Synthèse d'images animées

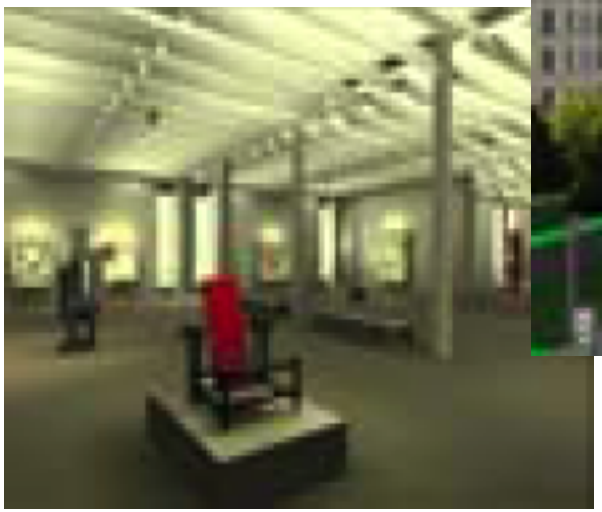
- Audiovisuel
- Effets spéciaux
- Jeux vidéos
- Etudes d'impact
- Simulateurs
- Visualisation scientifique

Synthèse d'images animées

- Audiovisuel
- Effets spéciaux
- Jeux vidéos
- Etudes d'impact
- Simulateurs
- Visualisation scientifique

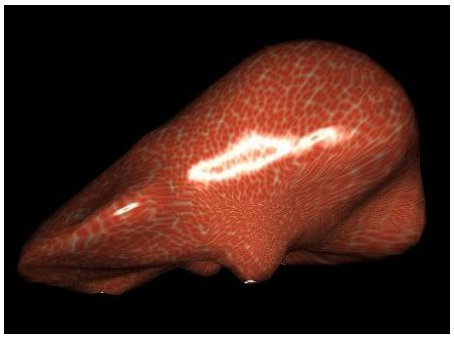
Synthèse d'images animées

- Audiovisuel
- Effets spéciaux
- Jeux vidéos
- Etudes d'impact
- Simulateurs
- Visualisation scientifique



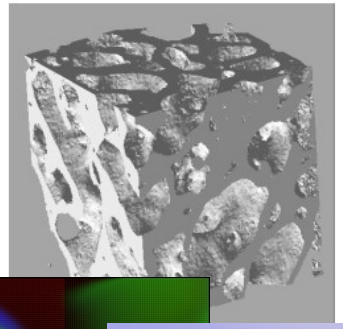
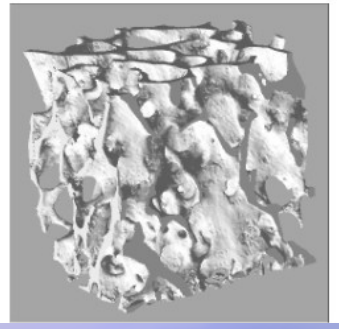
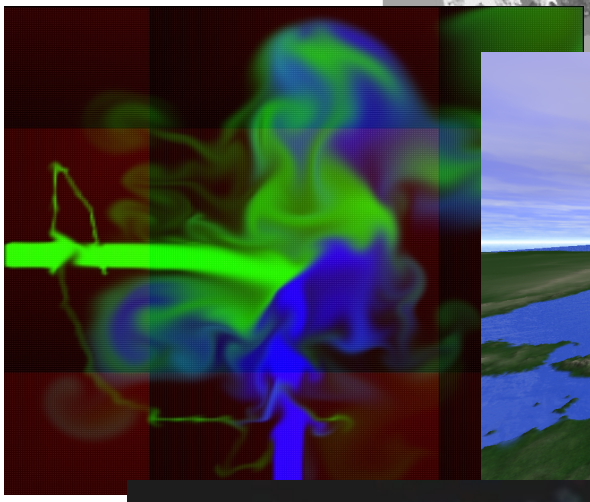
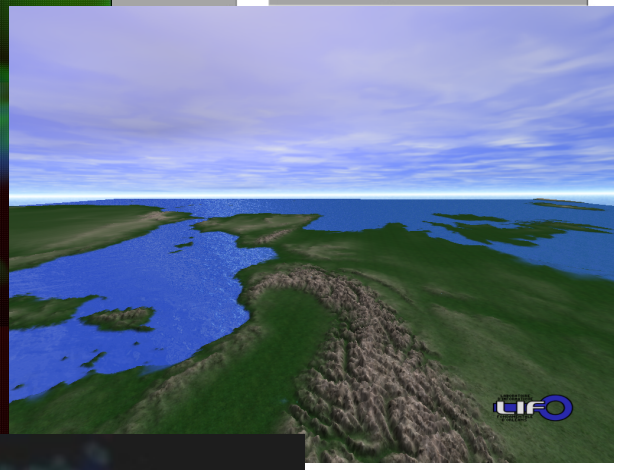
Synthèse d'images animées

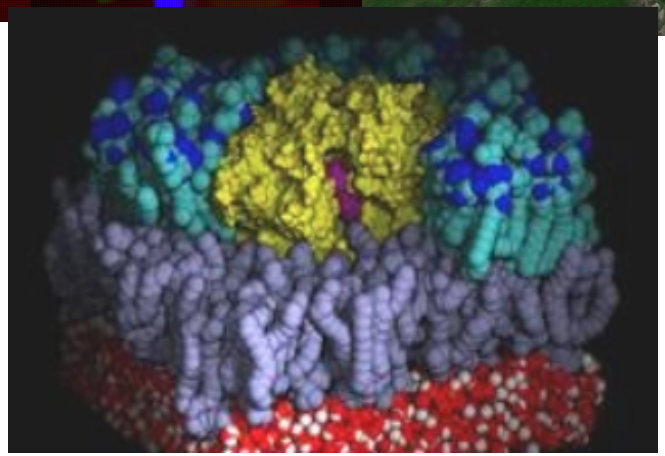
- Audiovisuel
- Effets spéciaux
- Jeux vidéos
- Etudes d'impact
- Simulateurs
- Visualisation scientifique



Synthèse d'images animées

- Audiovisuel
- Effets spéciaux
- Jeux vidéos
- Etudes d'impact
- Simulateurs
- Visualisation scientifique



Synthèse d'images animées

- Audiovisuel
- Effets spéciaux
- Jeux vidéos
- Études d'impact
- Simulateurs
- Visualisation scientifique

Réalisme

Temps réel

Synthèse d'images animées

- Modélisation
  - des objets, d'une scène
- Rendu d'une image
  - à partir des objets, matières,éclairages, caméras…
- Animation
  - spécifier ou calculer
  - mouvements et déformations
- La bibliothèque OpenGL

OpenGL

Définition

- **GL** et **OpenGL** sont des librairies graphiques d'affichage trois dimensions.
- **GL**: Librairie graphique standard de Silicon Graphics (orientée visualisation). Utilisée sur les stations **SGI** ainsi que sur les stations d'autres constructeurs (sous licence).
- **OpenGL**: Sous-ensemble de GL ne nécessitant pas une licence SGI (orienté visualisation)
- Sous Linux: MESA
- API graphique multi plate-forme.
- Depuis 1992, OpenGL est géré par l'« OpenGL Architecture Review Board » (**ARB**).

L'ARB

- Constitué des majors de l'industrie de l'informatique et de l'informatique graphique:
  - ATI, Compaq, Evans & Sutherland, HP, IBM, nVidia, Intergraph, Microsoft, SGI…
  - Microsoft, l'un des membres fondateurs, s'est retiré en mars 2003.
- L'ARB a pour rôle de maintenir et de faire évoluer les spécifications d'OpenGL. Il sélectionne et intégre les nouvelles extensions de l'API.

L'ARB

- En 2006, l'ARB a transféré le contrôle de la spécification OpenGL au Khronos Group, qui s'occupait déjà de différentes spécifications OpenGL pour les systèmes embarqués et les consoles de jeux vidéo
- Les spécifications d'OpenGL sont relativement stables. Aujourd'hui nous sommes à la version 4.1. OpenGL 2.1, OpenGL 3.0 et OpenGL 4.0 cohabitent.

- Interface logicielle avec les adaptateurs graphiques 3D
- Disponible pour un nombre important de plateformes et de langages
- Environ 120 instructions distinctes
- Spécification d'objets graphiques et production d'applications graphiques 3D interactives
- Conçu initialement pour fonctionner en environnement Unix
- -> exploitation de X-Windows-> exploitation des réseaux
- Standard de la programmation d'applications graphiques en CAO
- Claire évolution vers l'industrie du jeu
- Indépendant de la machine hôte
- Existence des versions 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 2.0, 2.1, 3.0, 3.1, 4.0 et 4.1

- OpenGL est indépendante de la plate-forme matérielle, et du système d'exploitation.
- Ouverture
- Souplesse d'utilisation
- Disponibilité sur des plate-formes variées,
- OpenGL est utilisée dans de nombreuses applications scientifiques, industrielles ou artistiques 3D et certaines applications 2D vectorielles.
- Cette bibliothèque est également populaire dans l'industrie du jeu vidéo où elle est en rivalité avec Direct3D (sous Microsoft Windows).
- Une version nommée OpenGL ES a été conçue spécifiquement pour les applications embarquées (téléphones portables, agenda de poche...).

L'architecture d'OpenGL

- OpenGL est basé sur le principe d'une machine à état.
  - `glEnable` / `glDisable` :
    - Permet d'activer ou non un état de la machine
    - Exemple: glEnable(GL_LIGHTING) active le calcul de l'éclairage de la scène.
- Par le biais de l'API, il est possible de spécifier différents aspects de la machine à état:
  - Couleur,
  - Illumination,
  - Blending…

L'architecture d'OpenGL

- Exemple: Pour dessiner un sommet rouge:
  - on positionne d'abord l'état correspondant à la couleur courante à la valeur rouge,
  - ensuite on demande le dessin d'un sommet.
    - Tous les sommets qui seront ensuite dessinés seront rouges, tant que l'on n'a pas modifié la couleur courante.
- Et ce principe que nous avons illustré à partir de l'état "couleur" s'applique à tous les états, comme l'épaisseur des traits, la transformation courante, l'éclairage, etc.
- Pour interroger la valeur d'un état, on utilise par exemple la commande `glGet*(...)` sous une de ses variantes.

Compléments d'OpenGL: La librairie GLU

- OpenGL Utility ajoute des fonctions de plus haut niveau:
  - 2D image scaling
  - Rendering 3D objects (sphere, cylindres,…)
  - Génération automatique des textures de Mip-mapping
  - Support pour les courbes et surfaces NURBS(Non-Uniform Rational Basis Splines)
  - Gérer les matrices de transformation et de projection,
  - la facettisation des polygones et le rendu de surface.

Compléments d'OpenGL

- OpenGL ne gère pas non plus le fenêtrage, et les entrées/sorties avec le clavier ou la souris.
- Pour celà, on utilise une bibliothèque complémentaire :
  - GLX pour X Windows (fonctions ayant pour préfixe **glx**)
  - WGL pour Microsoft Windows (fonctions ayant pour préfixe **wgl**)
  - GLUT: OpenGL Utility Toolkit, une boite à outils indépendante du système de fenêtrage, écrite par Mark Kilgard pour simplifier la tâche d'utiliser des systèmes différents (fonctions ayant pour préfixe **glut**).
- D'autres toolkits complets existent, comme Qt

La librairie GLUT

- OpenGL Utility Toolkit est une librairie développée pour la plupart des plates-formes.
- Elle gère de manière transparente le fenêtrage, les menus, la gestion du clavier et de la souris…
- Elle permet de développer un code multi plates-formes intégrant les gestions spécifiques des différents systèmes d'exploitation.

Principe: boucle evénementielle

- GLUT est basé sur une boucle infinie `glutMainLoop()` de lecture des événements qui se produisent sur les fenêtres qu'il gère.
- La réaction aux événements est programmée au travers de fonctions dit de « callback ».
- L'association entre un événement et son callback est effectuée par l'appel à la fonction `glutNomEv(&NomFonction)`.
- Un certain nombre d'initialisations sont nécessaires au paramétrage de la fenêtre.

Exemple

```
void DrawGLScene()
{ Affichage de la sc`ene}
void keyPressed(unsigned char key, int x, int y)
{ traitement des touches clavier }
int main(int argc, char *argv[]) {
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_RGBA|GLUT_DOUBLE);
glutInitWindowSize(640, 480);
glutInitWindowPosition(0, 0);
window = glutCreateWindow("Mon appli");
glutDisplayFunc(&DrawGLScene);
glutKeyboardFunc(&keyPressed);
glutMainLoop();
return 1;
}
```

GLUT: les fonctions de gestion d'une fenêtre

- `glutInit(int * argc, char **argv)` initialise GLUT et traite les arguments de la ligne de commande.
- `glutInitDisplayMode(unsigned int mode)` permet de choisir le mode d'affichage (couleurs RVB vs couleurs indexées , déclaration des buffers utilisés).
- `glutInitWindowPosition(int x, int y)` spécifie la localisation dans l'écran du coin haut gauche de la fenêtre.
- `glutInitWindowSize(ind width, int size)` spécifie la taille en pixels de la fenêtre.
- `int glutCreateWindow(char * string)` crée une fenêtre contenant un contexte OpenGL et renvoie l'identificateur de la fenêtre. La fenêtre ne sera affichée que lors du premier `glutMainLoop()`

GLUT: la fonction d'affichage

- `glutDisplayFunc(void (*func)(void))` spécifie la fonction à appeler pour l'affichage
- `glutPostRedisplay(void)` permet de forcer le réaffichage de la fenêtre.

GLUT: Lancement de la boucle d'événements

- 1 Attente d'un événement
- 2 Traitement de l'événement reçu
- 3 Retour en 1.

`glutMainLoop(void)` lance la boucle principale qui tourne pendant tout le programme, attend et appelle les fonctions spécifiques pour traiter les événements.

GLUT: Traitement des événements

- `glutReshapeFunc(void (*func)(int w, int h))` spécifie la fonction à appeler lorsque la fenêtre est retaillée.
- `glutKeyboardFunc(void (*func)(unsigned char key, int x, int y))` spécifie la fonction à appeler lorsqu'une touche est pressée
- `glutSpecialFunc(void (*func)(int key, int x, int y))` spécifie la fonction à appeler lorsqu'une touche spéciale (caractère non-ASCII) est pressée : les flèches, les touches de fonction, etc...
- `glutMouseFunc(void (*func)(int button, int state, int x, int y))` spécifie la fonction à appeler lorsqu'un bouton de la souris est pressé.
- `glutMotionFunc(void (*func)(int x, int y))` spécifie la fonction à appeler lorsque la souris est déplacée tout en gardant un bouton appuyé.

GLUT: Exécution en tâche de fond

- `glutIdleFunc(void (*func)(void))` spécifie la fonction à appeler lorsqu'il n'y a pas d'autre événement.

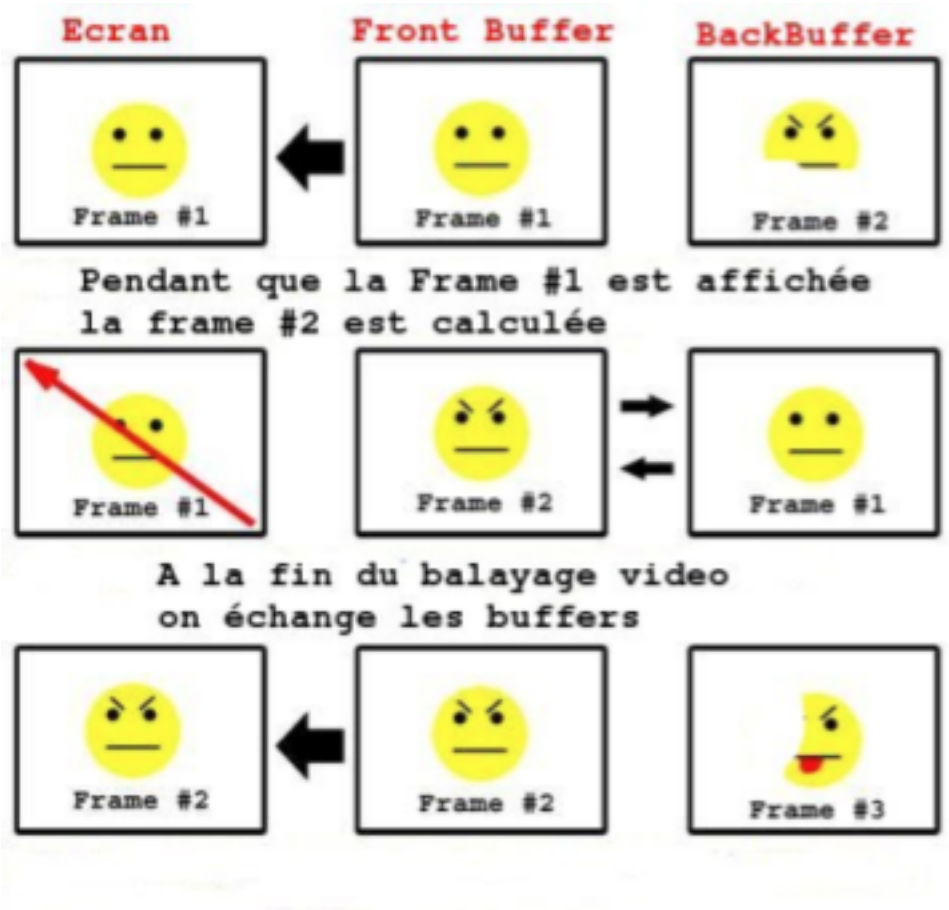
Exemple : un `glutPostRedisplay()`dedans permet de forcer le ré-affichage de la fenêtre dès la fin des événements.

GLUT: Dessin d'objets Tridimensionnels

- GLUT possède des routines pour afficher les objets suivants :
  - cone
  - cube
  - tetraèdre
  - octaèdre
  - icosaèdre
  - dodécaèdre
  - sphère
  - tore
  - théière (!)

GLUT: Animation

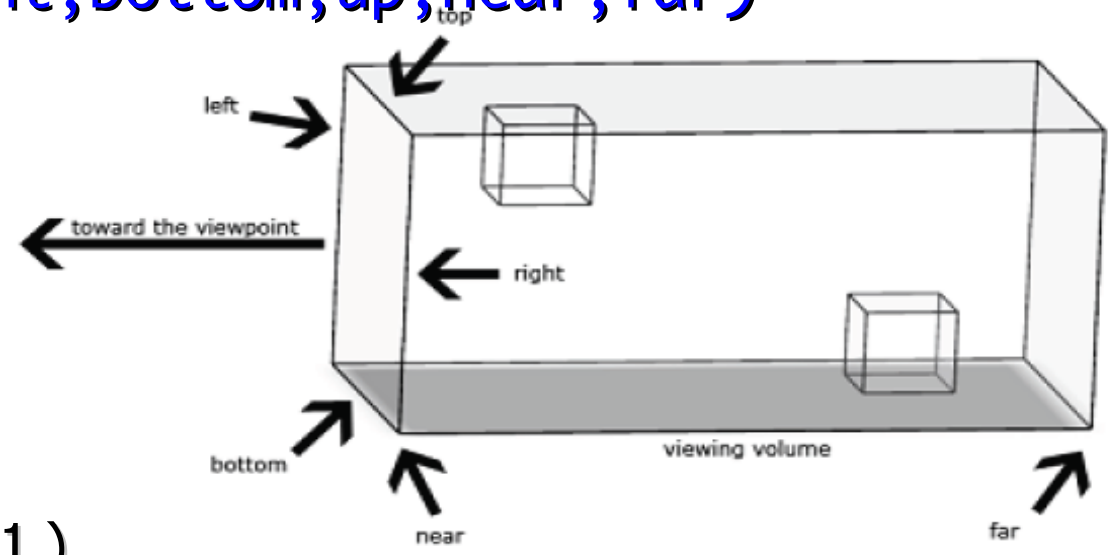
- Une animation peut se réaliser simplement en utilisant la technique du double buffer :
  - montrer à l'écran une image correspondant à une première zone mémoire (buffer)
  - dessiner les objets dans une deuxième zone mémoire qui n'est pas encore à l'écran,
  - elle sera affiché lorsque la scène entière y sera calculée, et à une fréquence régulière.
  - L'échange des buffers peut se faire par la commande `glutSwapBuffers(void);`



Avant de commencer le dessin...

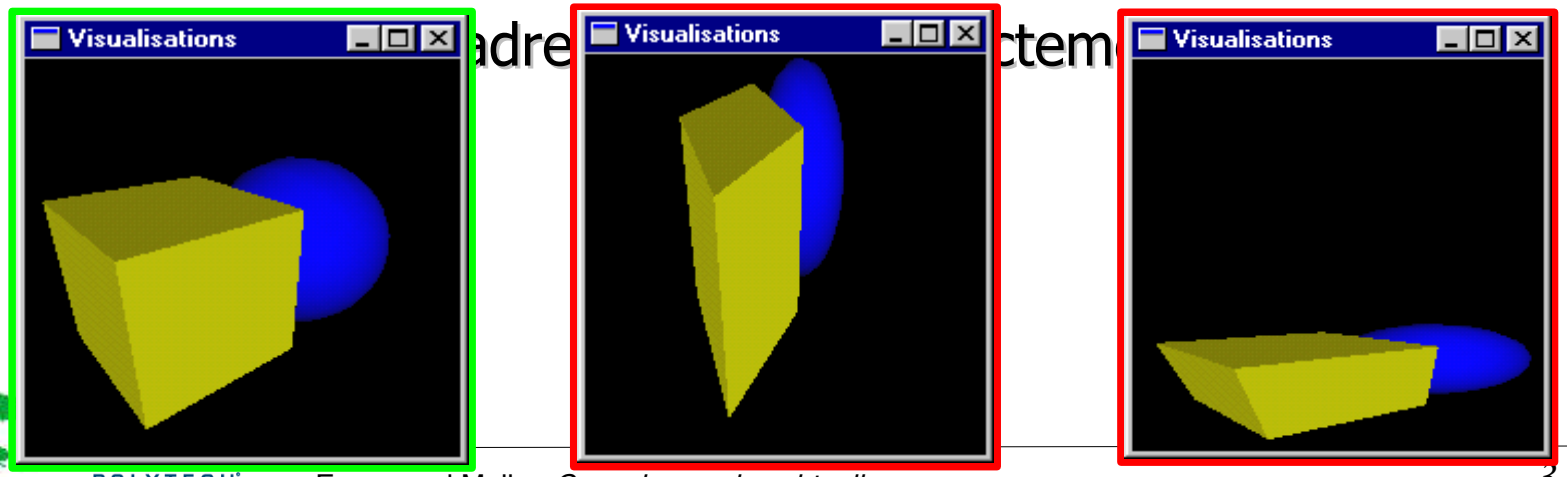
- Volume de visualisation
  - La scène est définie dans un repère absolu
  - Volume de visualisation: partie de la scène prise en considération
  - Dans un premier temps :
    `glOrtho(left,right,bottom,up,near,far)`
  - Seuls les objets a l'intérieur du volume de visualisation sont dessinés
  - Par défaut:

    glOrtho(-1.,1.-1.,1.,-1.,1.)



Projection sur la fenêtre

- Projection de la scène
  - La scène incluse dans le volume de visualisation est projetée orthogonalement au plan near
  - Le cadrage de cette projection par rapport à la fenêtre d'affichage est défini par `glViewPort(x,y,width,heigth)`
    - `x,y` indiquent l'angle inférieur gauche du cadrage dans la fenêtre
    - `width,heigth` indiquent la largeur et la hauteur du cadre



Syntaxe des commandes OpenGL

- Les commandes OpenGL commencent par le préfixe `gl`, et la première lettre de chaque mot est en majuscules.
- Le suffixe est composé d'un chiffre qui indique le nombre d'arguments, et d'une lettre qui indique le type des arguments.
- De plus, certaines commandes peuvent se terminer par la lettre `v`, pour indiquer l'emploi d'un tableau en paramètre.

Principe des fonctions OpenGL

- `glVertexX` :
  - Le X signifie qu'il existe plusieurs fonctions `glVertex` avec des paramètres différents:
    - 3i : **`glVertex3i`** attend 3 entiers
    - 3f : **`glVertex3f`** attend 3 flottants

-	*Type*	*Type C*	*Type OpenGL*
b	entier 8 bits	signed char	GLbyte
s	entier 16 bits	short	GLshort
i	entier 32 bits	long, int	GLint, GLsizei
f	réel 32 bits	float	GLfloat, GLclampf
d	réel 64 bits	double	GLdouble, GLclampd
ub	entier non signé8 bits	unsigned char	GLubyte, GLboolean
us	entier non signé16 bits	unsigned short	GLushort
ui	entier non signé32 bits	unsigned long	GLuint, GLenum,GLbitfield

Exemple

```
glVertex2i(1,3);   <=>   glVertex2d(1.0,3.0);
```

Une lettre terminale v supplémentaire indique que la fonction prend comme paramètre un pointeur sur un tableau.

```
glColor3i(1,0,1); <=>   int c[3]={1,0,1};

                        glColor3iv(c);
```

- Effacement de la fenêtre de dessin
- Par effacement de la fenêtre de dessin, on entend effacement des zones de mémoire suivantes (définies au sein de l'environnement OpenGL):
  - le tampon couleur -> stockage de la couleur des pixels de l'image (le tampon couleur doit être vidé avant le calcul d'une image),
  - le tampon profondeur -> stockage, pour chaque pixel de l'image, d'une information de profondeur utilisée lors de l'élimination des parties cachées (si l'élimination des parties cachées est activée, le tampon profondeur doit être vidé avant le calcul d'une image),
  - le tampon accumulation -> accumulation d'images les unes sur les autres,
  - le tampon stencil-> restreindre le rendu à certaines portions de l'écran
- L'effacement est réalisé par:

```
void glClear(GLbitfield mask);
```

mask: choix du buffer à effacer

mask
GL_COLOR_BUFFER_BIT Effacement des pixels de la fenêtre
GL_DEPTH_BUFFER_BIT Effacement de l'information de profondeur associée à chaque pixel de la fenêtre (information utilisée pour l'élimination des parties cachées)
GL_ACCUM_BUFFER_BIT Effacement du tampon accumulation utilisé pour composer des images
GL_STENCIL_BUFFER_BIT Non renseigné

Les valeurs de masque sont composables par "ou" pour effectuer plusieurs opérations d'effacement en une seule instruction `glClear` et rendre possible l'optimisation de ces opérations.

Les couleurs sont définies en OpenGL de deux manières :

- Couleurs indexées : 256 couleurs sont choisies, et on se réfère au numéro de la couleur (son index). C'est un mode qui était intéressant lorsque les écrans d'ordinateurs ne savaient afficher que 256 couleurs simultannées.
- Couleurs RVBA : une couleur est définie par son intensité sur 3 composantes Rouge, Vert, Bleu. La quatrième composante est appelée canal Alpha, et code l'opacité.

- Choix de la couleur d'éffacement du tampon couleur:
  - `void glClearColor(GLclampf r,GLclampf v,GLclampf b, GLclampf alpha) ;`

r, v, b, alpha: couleurs de remplissage du tampon couleur lors d'un effacement

- Choix de la profondeur d'initialisation du tampon profondeur:
  - `void glClearDepth(GLclampf depth) ;`

depth: valeur de remplissage du tampon profondeur lors d'un effacement

- Choix de la couleur de tracé
  - `void glColor3{b s i f d ub us ui} (TYPE r,TYPE v,TYPE b);`
  - `void glColor4{b s i f d ub us ui} (TYPE r,TYPE v,TYPE b,TYPE alpha);`

- OpenGL dessine principalement des points, des lignes et des polygones:
  - L' élément de base de ces objets est le sommet ou vertex
  - Un sommet est un ensemble de 3 coordonnées (+ 1 pour avoir

des coordonnées homogènes)

- OpenGL travaille en coordonnées homogènes afin de linéariser des opérations sur les matrices qui sont affines en 3d.
  - Principe:

$$\textit{Une matrice de rotation en } 3\text{d} \textit{ est de taille } 3*3 : A = \begin{bmatrix} A_{1\,1} & A_{1\,2} & A_{1\,3} \\ A_{2\,1} & A_{2\,2} & A_{2\,3} \\ A_{3\,1} & A_{3\,2} & A_{3\,3} \end{bmatrix}$$

$$\textit{alors qu ' une translation } T \textit{ s ' écrit sous la forme d ' un vecteur } 3 : T = \begin{bmatrix} T_1 \\ T_2 \\ T_3 \end{bmatrix}$$

$$\textit{donc quand on veut calculer la nouvelle position}$$
$$\textit{d ' un point } V(x, y, z) \textit{ il faut faire} : V_{nouveau} = AV + T$$

En coordonnées homogènes...

$$\text{Une matrice de rotation en } 3\text{d est de taille } 4*4 : M_{rot} = \begin{bmatrix} A_{11} & A_{12} & A_{13} & 0 \\ A_{21} & A_{22} & A_{23} & 0 \\ A_{31} & A_{32} & A_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{alors une translation } T \text{ s'écrit sous la forme aussi d'une matrice } 4*4 :$$

$$M_{trans} = \begin{bmatrix} 1 & 0 & 0 & V_1 \\ 0 & 1 & 0 & V_2 \\ 0 & 0 & 1 & V_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{donc quand on veut calculer la nouvelle position}$$
$$\text{d'un point } V(x, y, z, 1) \text{ il faut faire : } V_{nouveau} = M_{rot} * M_{trans} * V$$

- Par ailleurs cela permet de définir des vecteur infinis: (x,y,z,0)
- Et de définir des classes d'équivalences constituées de vecteurs colinéaires: si c est un scalaire alors (cx,cy,cz,cw)≃(x,y,z,w)

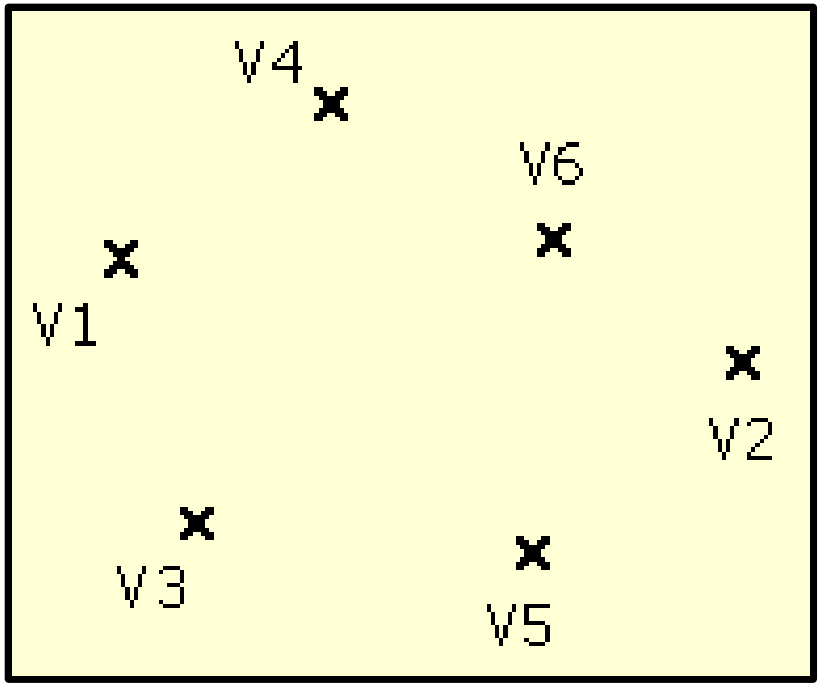
Sommets, lignes et polygones

- Sommet: Ensemble de trois coordonnées représentant une position dans l'espace
- Ligne: Segment rectiligne entre deux sommets
- Polygone: Surface délimitée par une boucle fermée de lignes
  - Par définition, un polygone OpenGL ne se recoupe pas, n'a pas de trou et est de bord convexe.
    - Si on veut représenter un polygone ne vérifiant pas ces conditions, on devra le subdiviser (tessélation) en un ensemble de polygones convenables.
    - GLU propose des fonctions pour gérer les polygones spéciaux.
  - Les polygones OpenGL ne doivent pas forcément être plans.
    - En revanche, le résultat à l'affichage n'est pas déterministe en cas de non planéité

- Déclaration d'une position
  - `glVertex{234}{sifd}[v](TYPE coords) ;`
  - coords: coordonnées de la position
  - Cette fonction sert uniquement à la déclaration de sommets au sein d'une primitive graphique.
- Déclaration d'une primitive graphique
  - Une primitive graphique est constituée d'un ensemble de positions.
  - Elle est créée par la réalisation successive des instructions glVertex permettant la définition des positions de ses sommets.
  - La primitive est délimitée au début et à la fin par
- `void glBegin(GLenum mode) ;`
- mode: GL_POINTS, GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP, GL_POLYGON, GL_QUADS, GL_QUAD_STRIP, GL_TRIANGLES, GL_TRIANGLE_STRIP ou GL_TRIANGLE_FAN

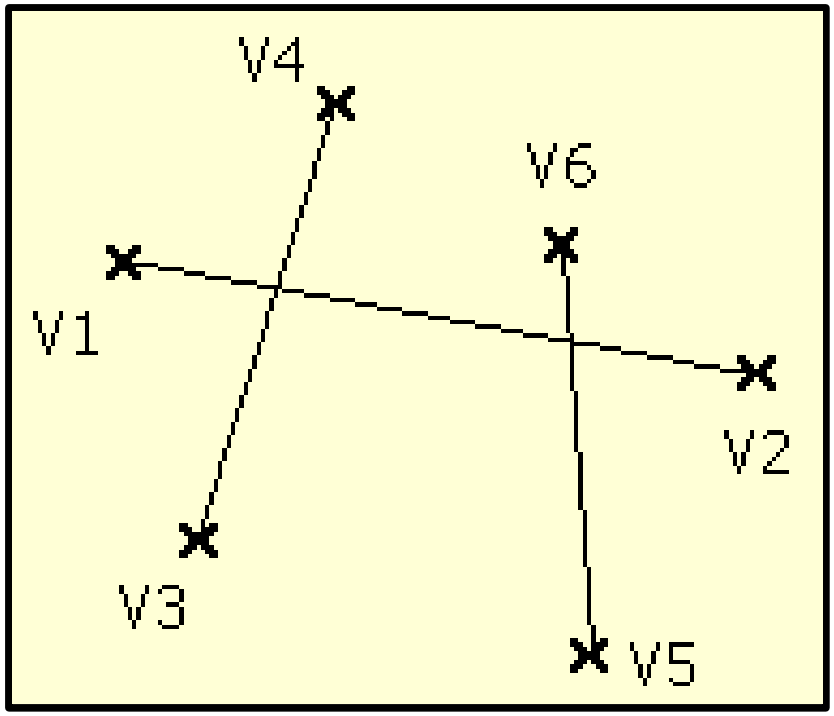
Exemple

```
glBegin(GL_POINTS);
glVertex3f(x1,y1,z1)
;
glVertex3f(x2,y2,z2)
;
glVertex3f(x3,y3,z3)
;
glVertex3f(x4,y4,z4)
;
glVertex3f(x5,y5,z5)
;
glVertex3f(x6,y6,z6)
;
glEnd();
```



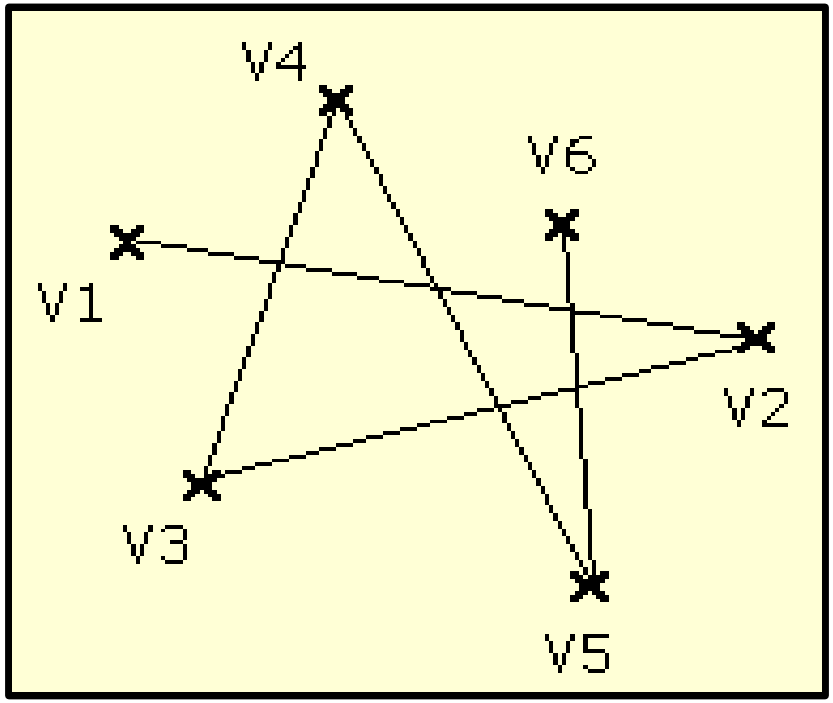
Exemple

```
glBegin(GL_LINES);
glVertex3f(x1,y1,z1)
;
glVertex3f(x2,y2,z2)
;
glVertex3f(x3,y3,z3)
;
glVertex3f(x4,y4,z4)
;
glVertex3f(x5,y5,z5)
;
glVertex3f(x6,y6,z6)
;
glEnd();
```



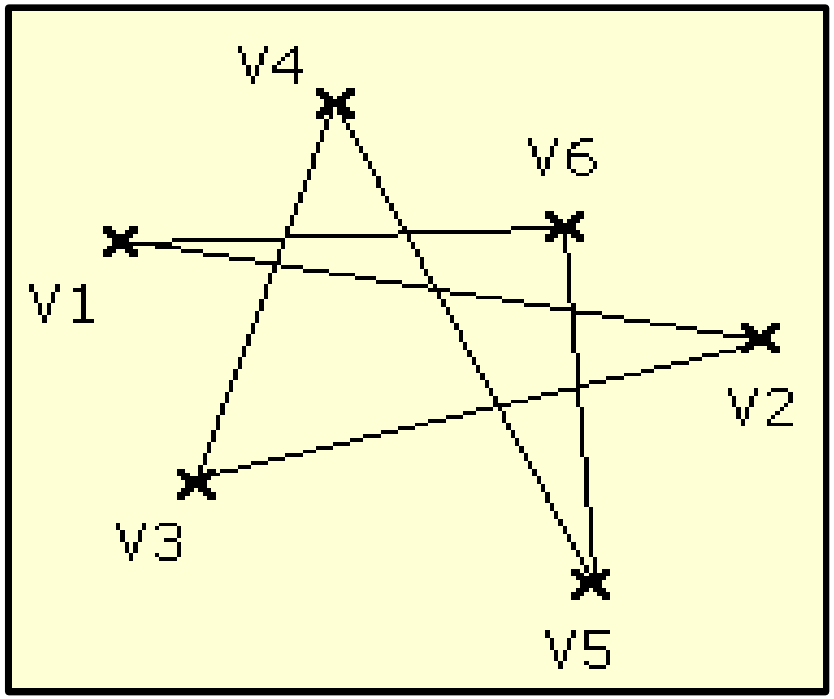
Exemple

```
glBegin(GL_LINE_STRIPS)
;
glVertex3f(x1,y1,z1)
;
glVertex3f(x2,y2,z2)
;
glVertex3f(x3,y3,z3)
;
glVertex3f(x4,y4,z4)
;
glVertex3f(x5,y5,z5)
;
glVertex3f(x6,y6,z6)
;
glEnd();
```



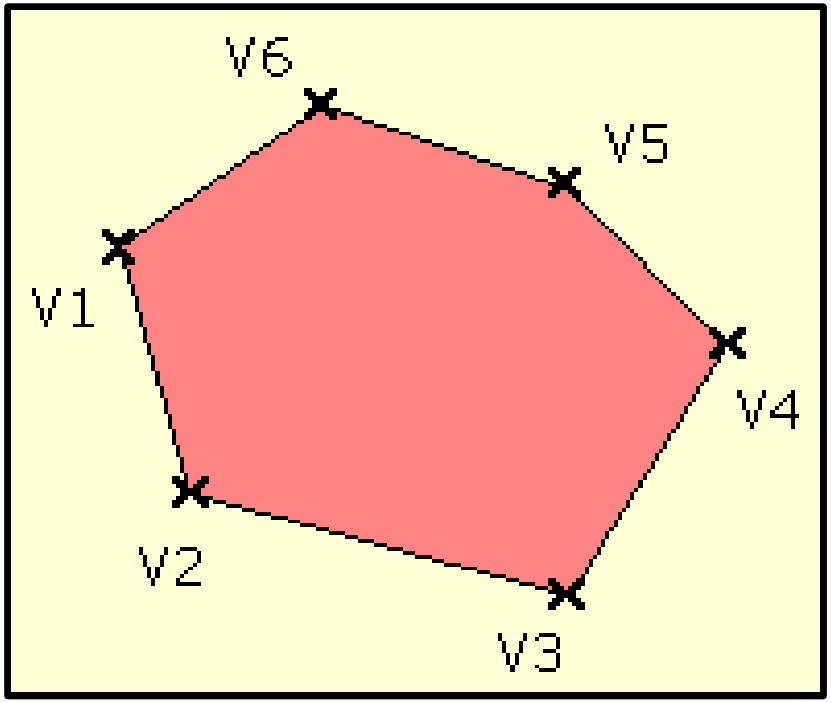
Exemple

```
glBegin(GL_LINE_LOOP
);
glVertex3f(x1,y1,z1)
;
glVertex3f(x2,y2,z2)
;
glVertex3f(x3,y3,z3)
;
glVertex3f(x4,y4,z4)
;
glVertex3f(x5,y5,z5)
;
glVertex3f(x6,y6,z6)
;
```



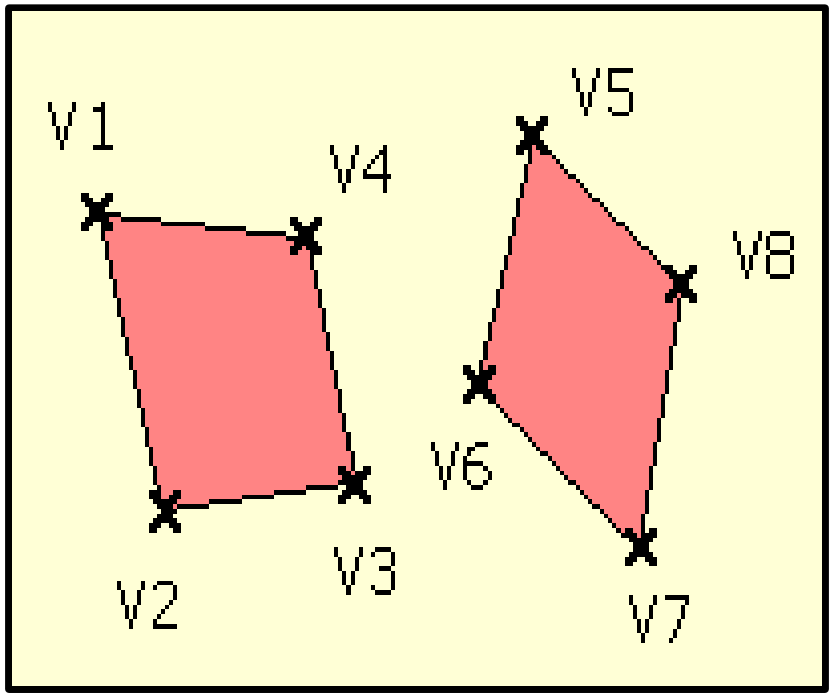
Exemple

```
glBegin(GL_POLYGON);
glVertex3f(x1,y1,z1)
;
glVertex3f(x2,y2,z2)
;
glVertex3f(x3,y3,z3)
;
glVertex3f(x4,y4,z4)
;
glVertex3f(x5,y5,z5)
;
glVertex3f(x6,y6,z6)
;
glEnd();
```



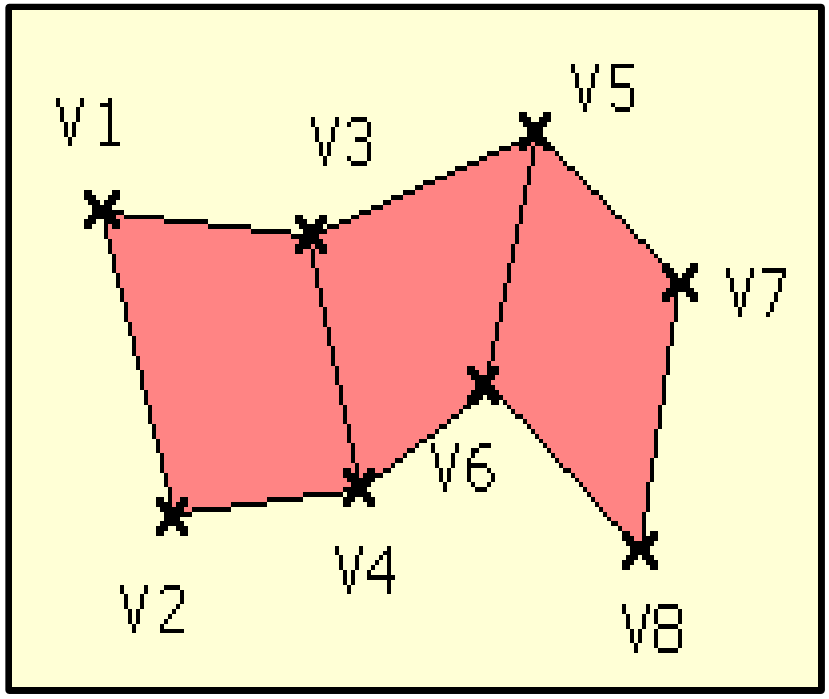
Exemple

```
glBegin(GL_QUADS);
glVertex3f(x1,y1,z1)
;
glVertex3f(x2,y2,z2)
;
glVertex3f(x3,y3,z3)
;
glVertex3f(x4,y4,z4)
;
glVertex3f(x5,y5,z5)
;
glVertex3f(x6,y6,z6)
;
glEnd();
```



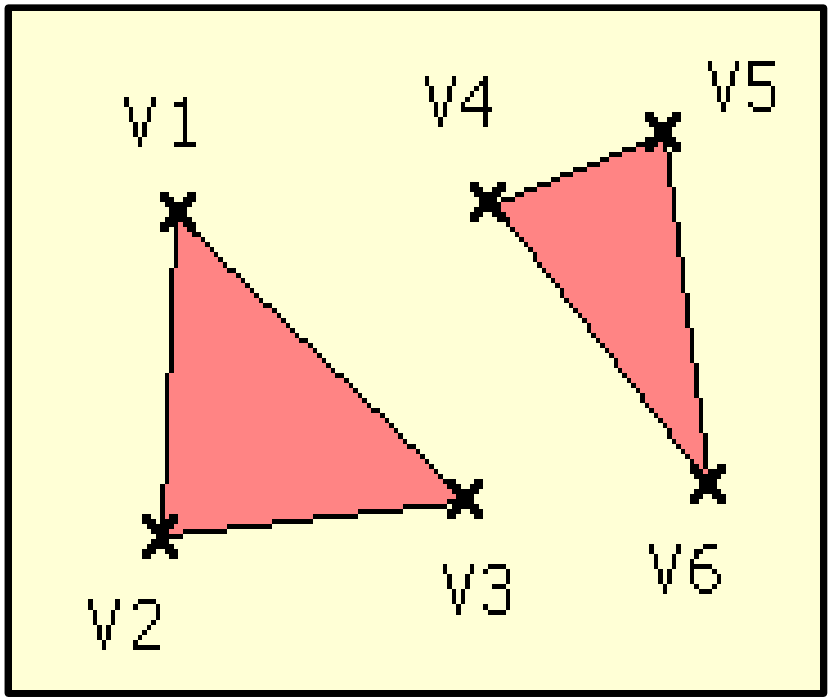
Exemple

```
glBegin(GL_QUAD_STRIP);
glVertex3f(x1,y1,z1);
glVertex3f(x2,y2,z2);
glVertex3f(x3,y3,z3);
glVertex3f(x4,y4,z4);
glVertex3f(x5,y5,z5);
glVertex3f(x6,y6,z6);
```



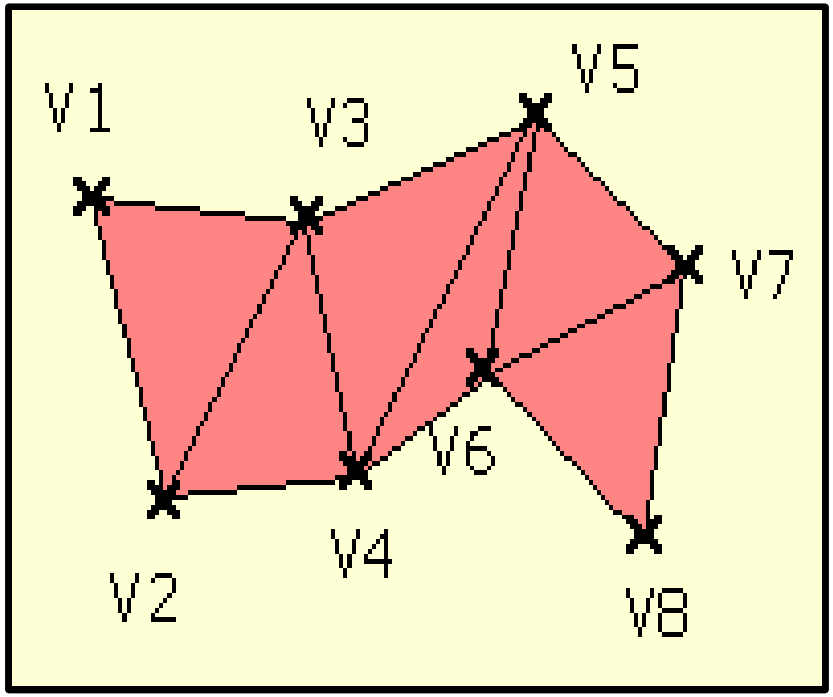
Exemple

```
glBegin(GL_TRIANGLES
);
glVertex3f(x1,y1,z1)
;
glVertex3f(x2,y2,z2)
;
glVertex3f(x3,y3,z3)
;
glVertex3f(x4,y4,z4)
;
glVertex3f(x5,y5,z5)
;
glVertex3f(x6,y6,z6)
;
```



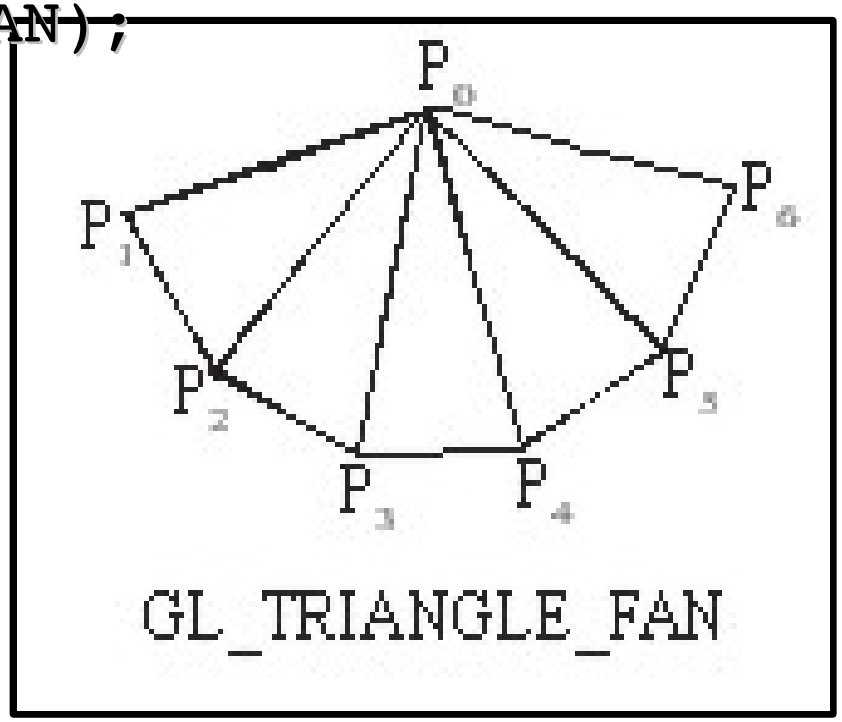
Exemple

```
glBegin(GL_TRIANGLES_STRIP);
glVertex3f(x1,y1,z1);
glVertex3f(x2,y2,z2);
glVertex3f(x3,y3,z3);
glVertex3f(x4,y4,z4);
glVertex3f(x5,y5,z5);
glVertex3f(x6,y6,z6);
glEnd();
```



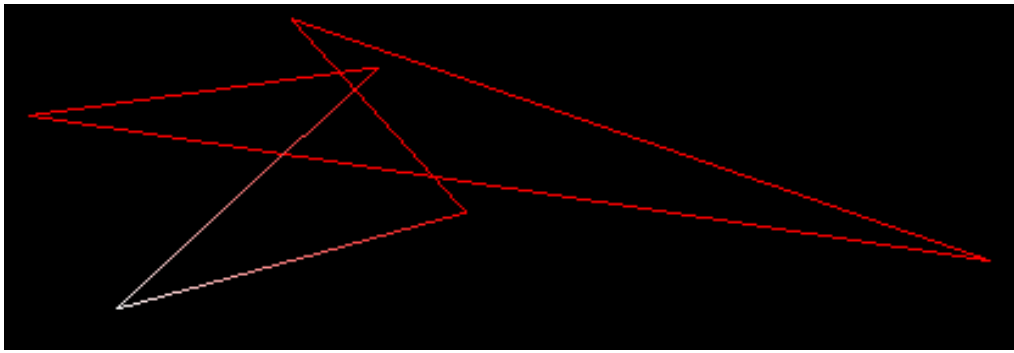
Exemple

```
glBegin(GL_TRIANGLE_FAN);
glVertex3f(x1,y1,z1);
glVertex3f(x2,y2,z2);
glVertex3f(x3,y3,z3);
glVertex3f(x4,y4,z4);
glVertex3f(x5,y5,z5);
glVertex3f(x6,y6,z6);
glEnd();
```



Exemple: Création d'un polygone

```
glBegin(GL_POLYGON);
glColor3f(1.0F,1.0F,1.0F);
glVertex2f(0.0F,0.0F) ;
glColor3f(1.0F,0.0F,0.0F);
glVertex2f(2.0F,1.0F) ;
glVertex2f(1.0F,3.0F) ;
glVertex2f(5.0F,0.5F) ;
glVertex2f(-.5F,2.0F) ;
glVertex2f(1.5F,2.5F) ;
glEnd() ;
```



- La définition d'une primitive peut faire appel à d'autres informations que ses seuls points. Celles-ci sont aussi renseignées via des appels à des fonctions particulières.
- Parmi ces fonctions, l'une des plus importantes est:
  - `void glNormal3{bsidf}(TYPE x,TYPE y,TYPE z);`
  - `void glNormal3{bsidf}v(const TYPE *v);`
    - Configuration de la normale courante (utilisée pour les calculs d'éclairage) avec la valeur passée en paramètre (valeur par défaut : (0.0,0.0,1.0)).

Exemple:

```
glBegin(GL_POLYGON);
glColor3f(1.0F,0.0F,0.0F) ;
glVertex2f(0.0F,0.0F) ;
glNormal3f(-1.0F,0.0F,0.0F) ;
glVertex2f(2.0F,1.0F) ;
glColor3f(0.0F,1.0F,0.0F) ;
glVertex2f(1.0F,3.0F) ;
glEnd() ;
```

Définit un polygone à trois sommets. Donne la couleur rouge aux premier et deuxième sommets, et la couleur verte au troisième. Donne la normale (-1.0,0.0,0.0) aux deuxième et troisième sommets. Le premier adopte comme normale la valeur courante définie avant le glBegin.

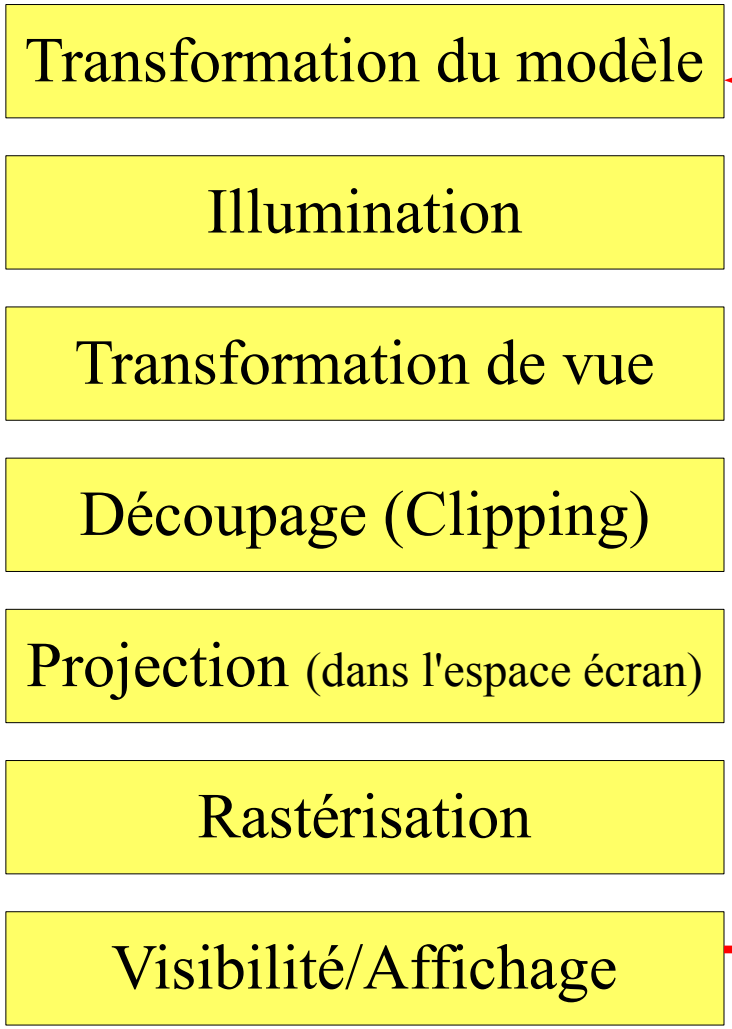
Terminaison du tracé d'une image

Utilisé pour forcer l'exécution entière du tracé d'une image (vidage d'un pipeline d'affichage sur une station graphique, exécution des ordres de tracé sur un réseau).

- `void glFlush(void);`
- `void glFinish(void);`

glFinish se met en attente d'une notification d'exécution de la part du dispositif graphique tandis que glFlush n'est pas bloquant.

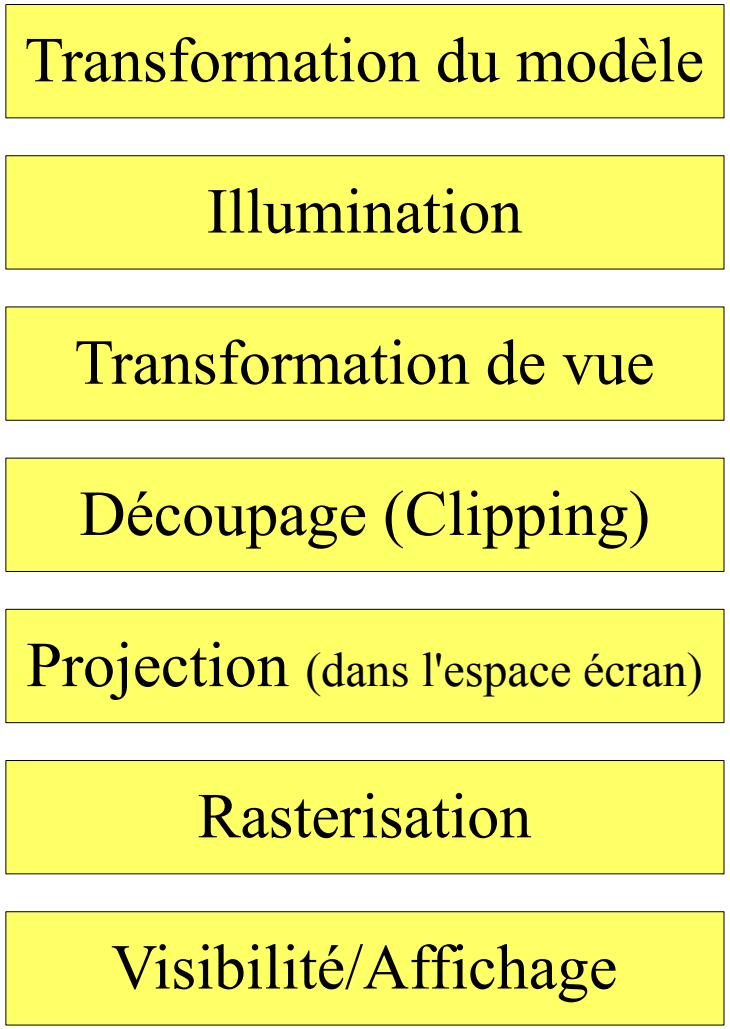
Le Pipeline Graphique



- Modèle géométrique: objets, surfaces et sources de lumière
- Modèle d'illumination: calcul de la lumière
- Caméra: point de vue, cône de vision
- Fenêtre: là où l'image est construite

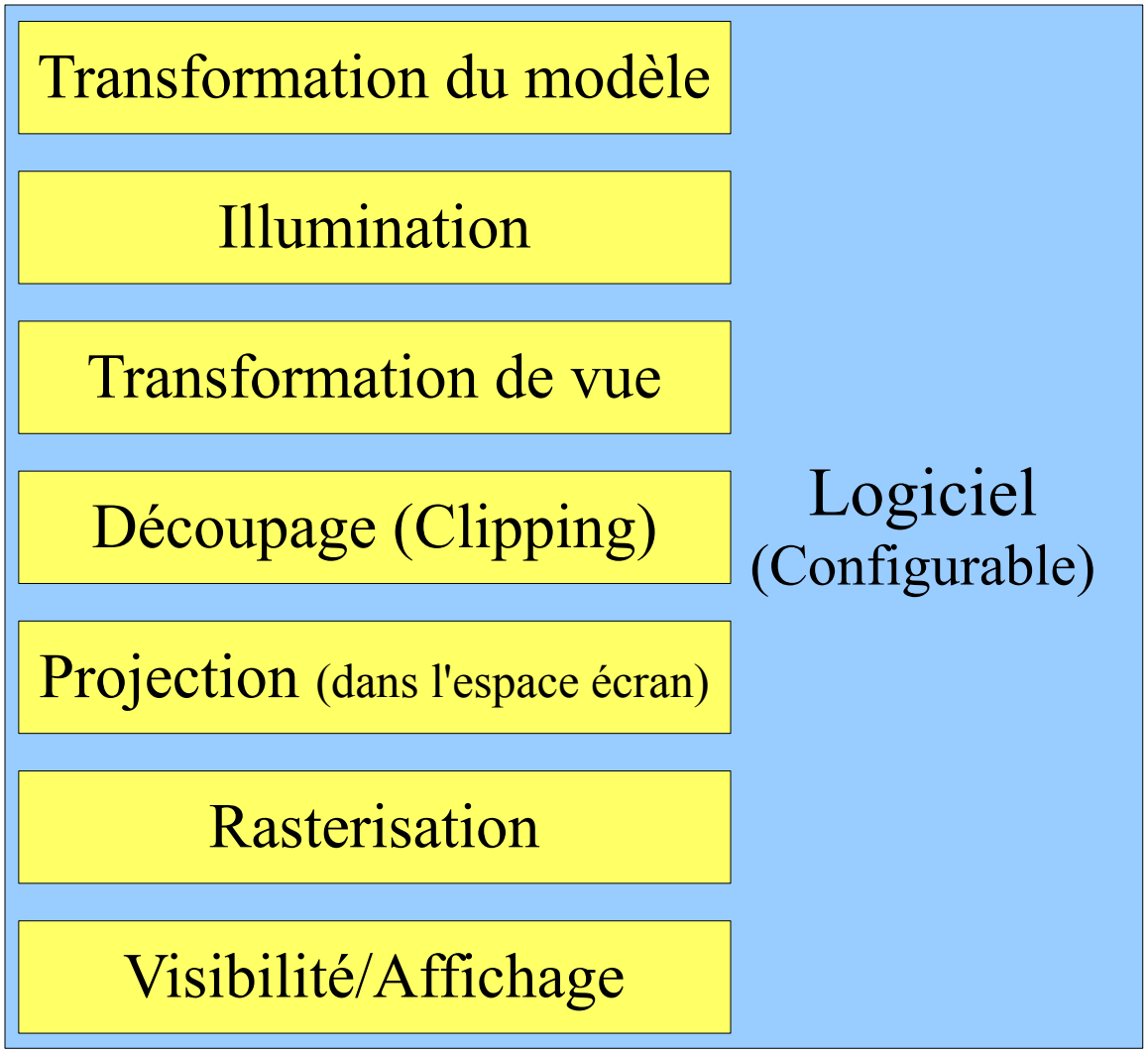
- pixels, couleurs et intensités portés par un signal analogique ou numérique adapté à l'écran/projecteur

Le Pipeline Graphique



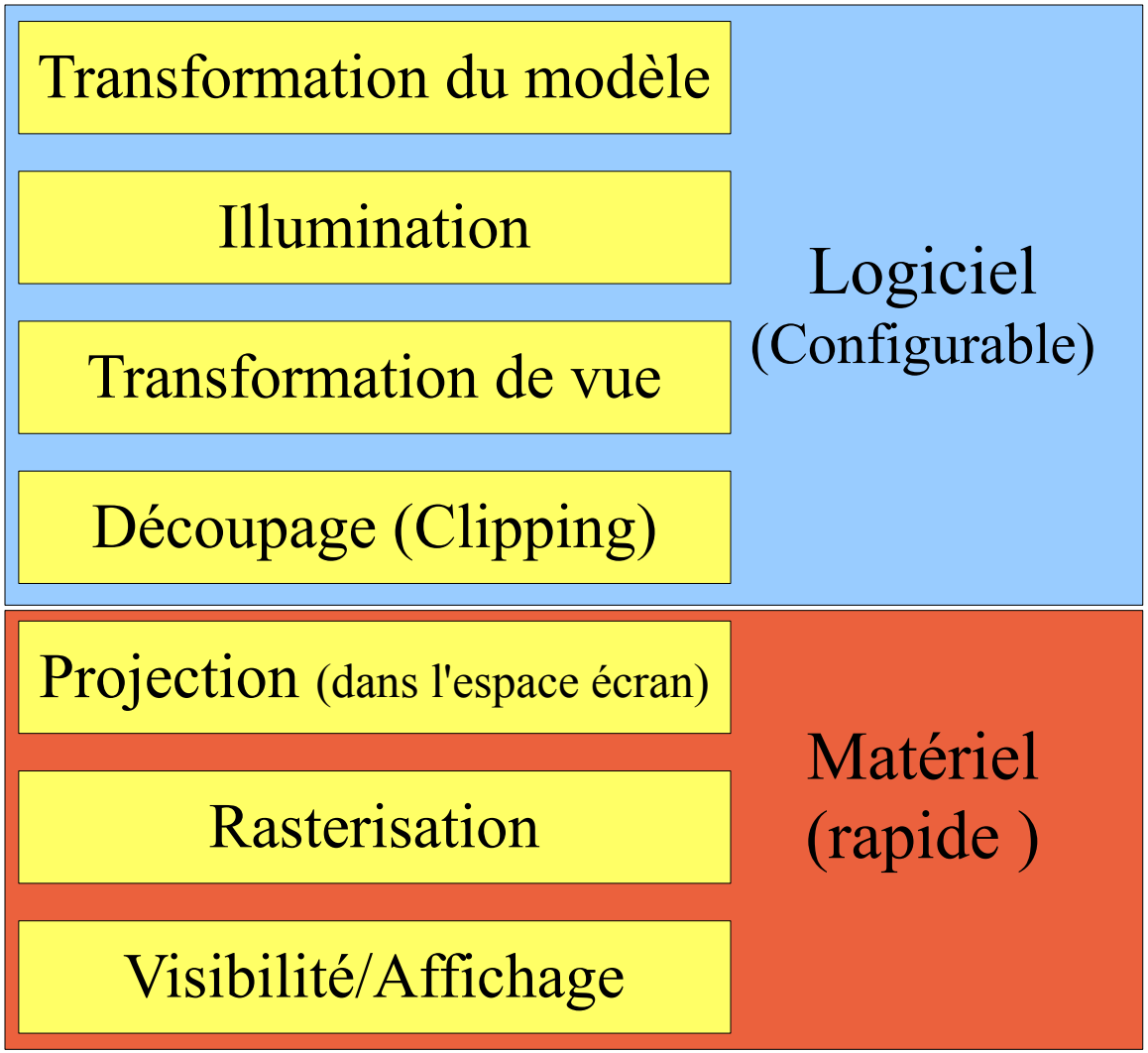
- Chaque primitive passe les étapes dans l'ordre
- Le pipeline peut être indifféremment implanté matériellement ou logiquement
- On peut ajouter à certaines étapes des moyen d'expression de traitement libres (vertex/pixel program)

Le Pipeline Graphique



Sans carte graphique 3D

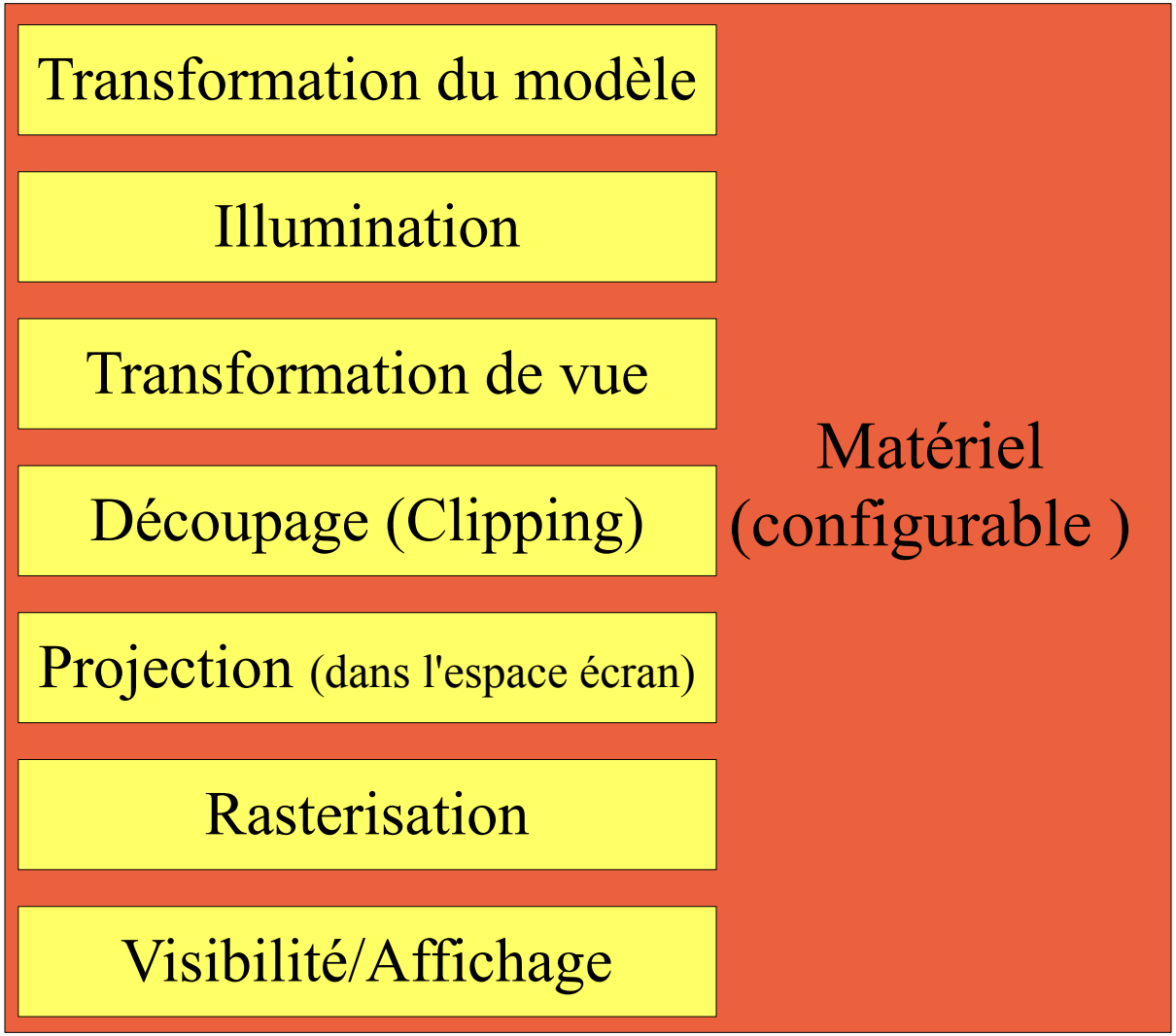
Le Pipeline Graphique



Cartes graphiques 3D

1ère génération

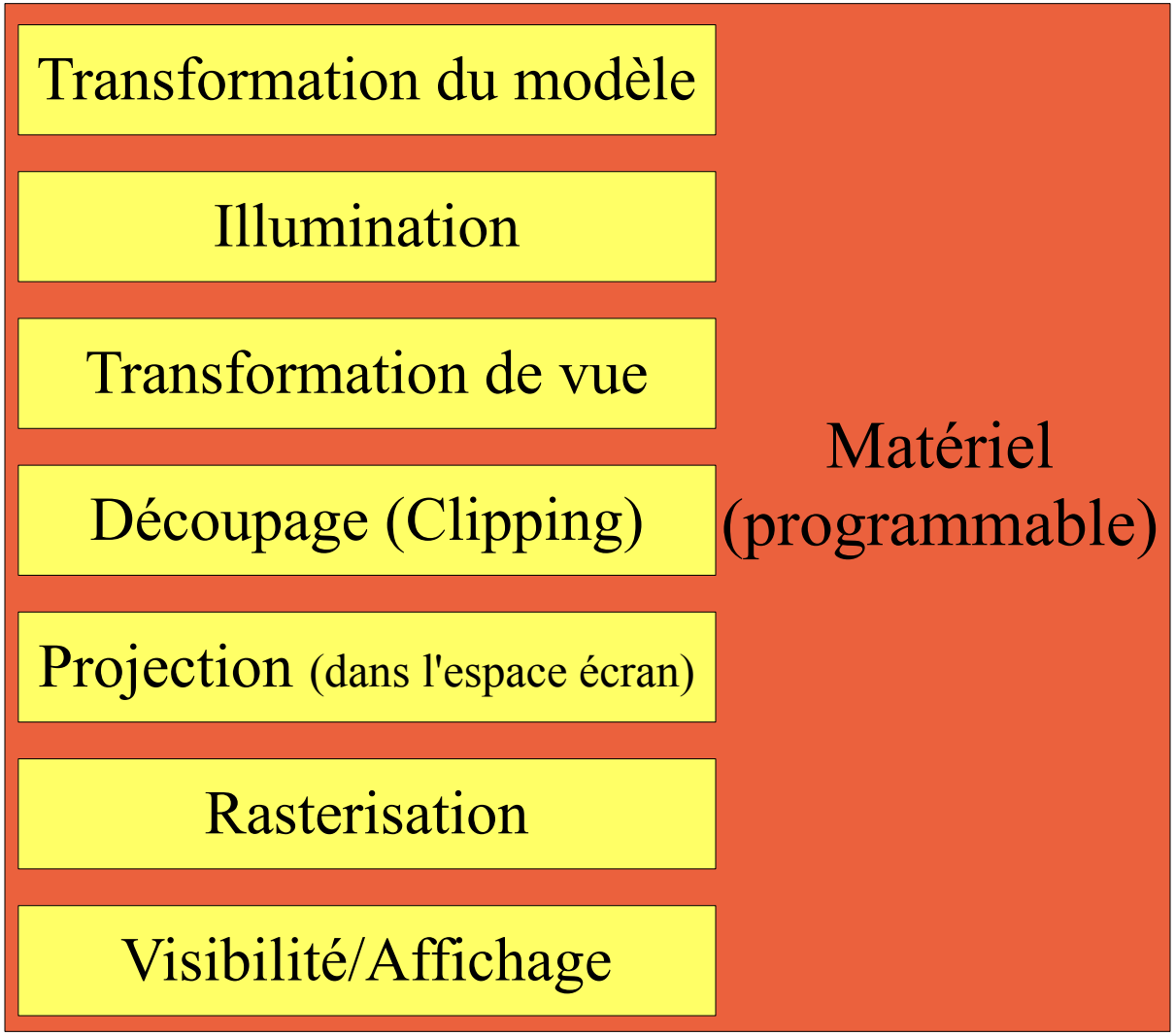
Le Pipeline Graphique



Cartes graphiques 3D

2ème génération

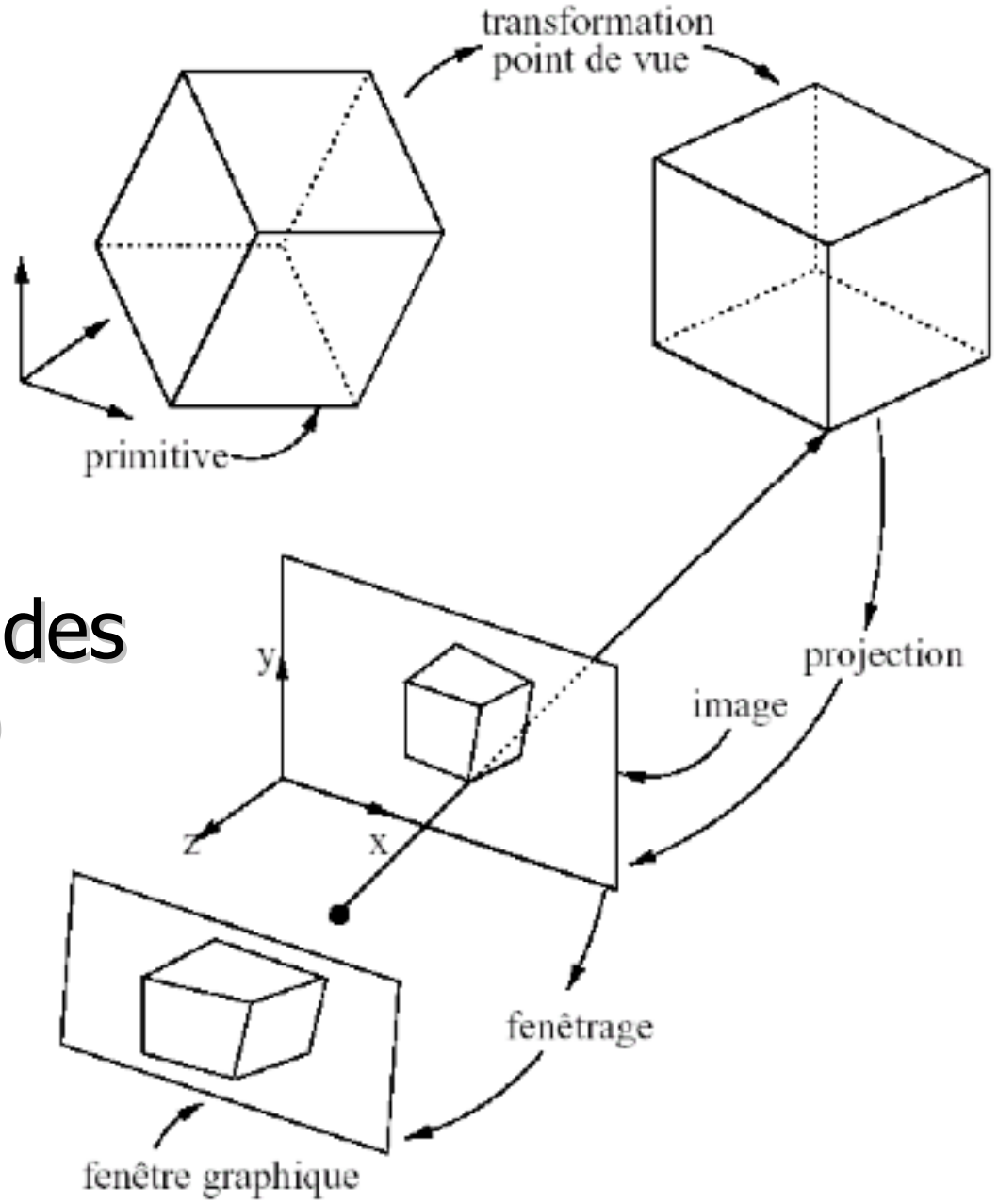
Le Pipeline Graphique



Cartes graphiques 3D

3ème génération

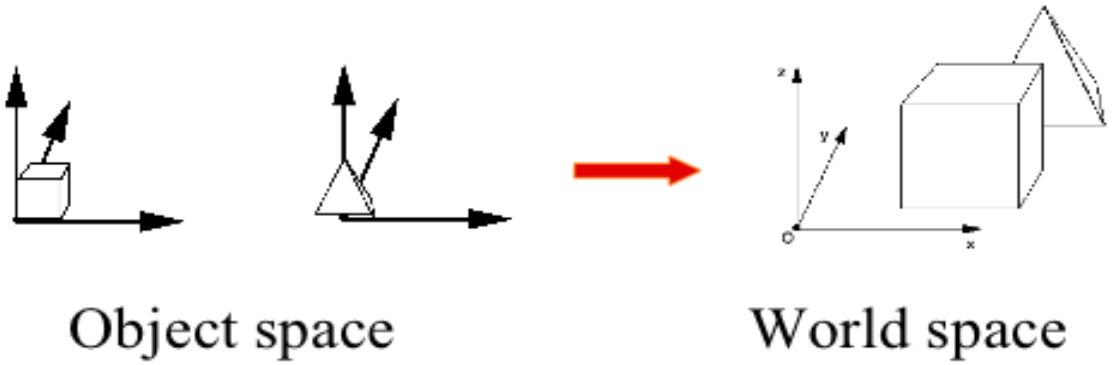
Transformations des primitives 3D



Transformation du modèle

- Transformation du modèle
- Illumination
- Transformation de vue
- Découpage (Clipping)
- Projection (dans l'espace écran)
- Rastérisation
- Visibilité/Affichage

Passage du système de coordonnées local vers des coordonnées dans un repère global

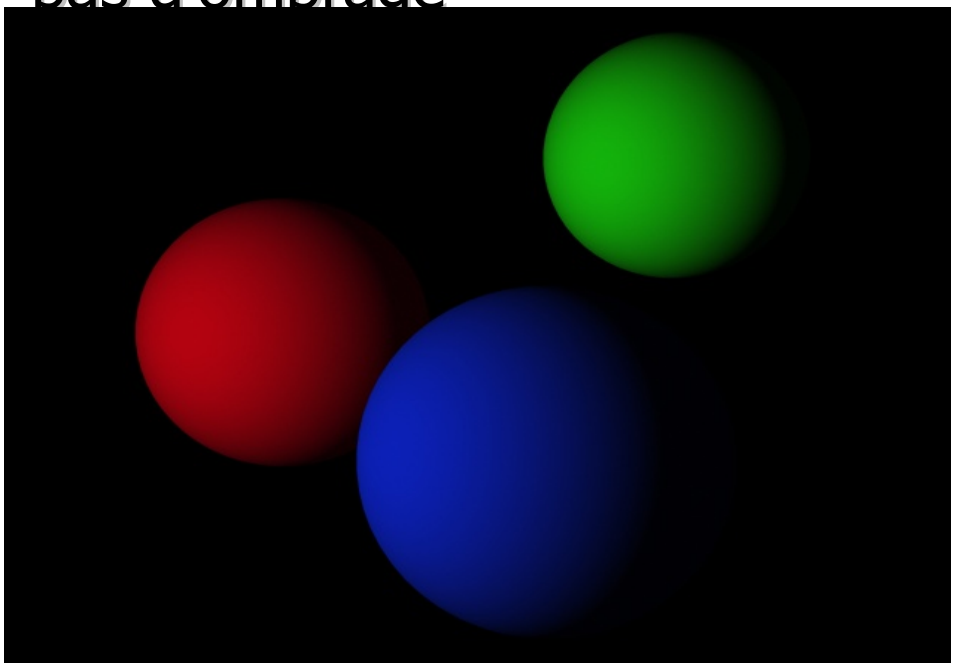


Illumination (shading)

- Transformation du modèle
- **Illumination** ←
- Transformation de vue
- Découpage (Clipping)
- Projection (dans l'espace écran)
- Rastérisation
- Visibilité/Affichage

Les primitives sont éclairées selon leur matériau, leur surface, et les sources de lumières

Le calcul est local aux primitives: pas d'ombrage



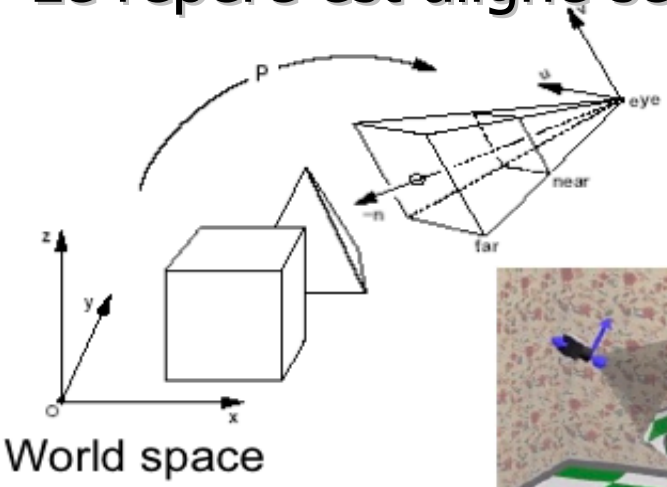
Rendu facette Gouraud Phong

Transformation de vue

- Transformation du modèle
- Illumination
- Transformation de vue
- Découpage (Clipping)
- Projection (dans l'espace écran)
- Rastérisation
- Visibilité/Affichage

On passe des coordonnées du monde à celles du point de vue (espace caméra). Les sommets sont transformés par la matrice de modèle / vue (modelview chez OpenGL)

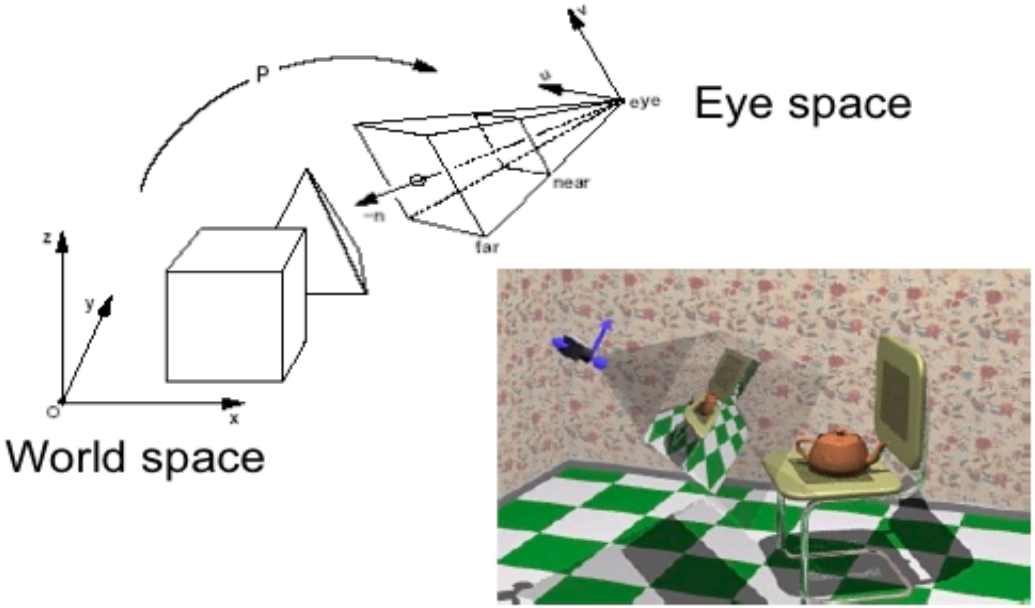
Le repère est aligné selon Z



Transformation de vue

- Transformation du modèle
- Illumination
- Transformation de vue
- Découpage (Clipping)
- Projection (dans l'espace écran)
- Rastérisation
- Visibilité/Affichage

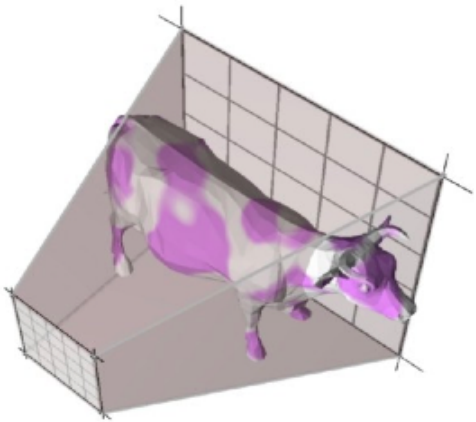
- Les coordonnées sont transformées par la matrice de projection afin de leur faire subir l'effet de perspective.



Découpage

- Transformation du modèle
- Illumination
- Transformation de vue
- Découpage (Clipping) ←
- Projection (dans l'espace écran)
- Rastérisation
- Visibilité/Affichage

Suppression de la géométrie en dehors du cone de vision (frustrum culling)



Découpage

Transformation du modèle

Illumination

Transformation de vue

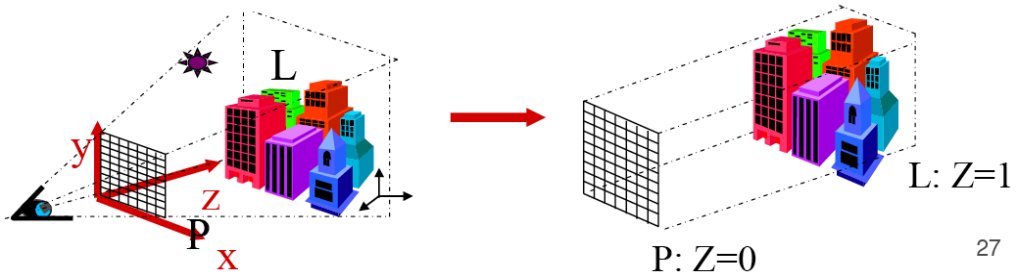
Découpage (Clipping)

Projection (dans l'espace écran)

Rastérisation

Visibilité/Affichage

- Les coordonnées (X, Y, Z) obtenues précédemment sont divisées par la composante W afin d'être normalisées (NDC: normalized device coordinates). Elles se retrouvent donc entre -1 et 1.



27

Projection

- Les coordonnées normalisées sont projetées sur l'image 2D. (-> window coordinates)

Transformation du modèle

Illumination

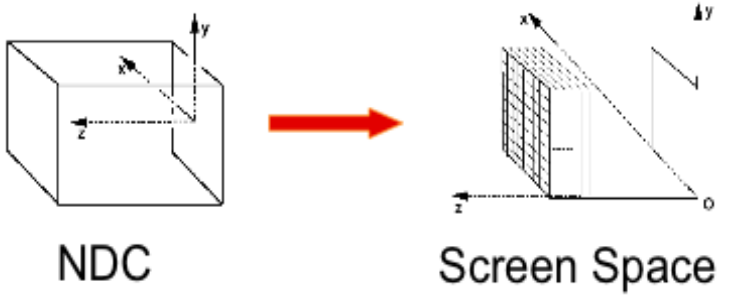
Transformation de vue

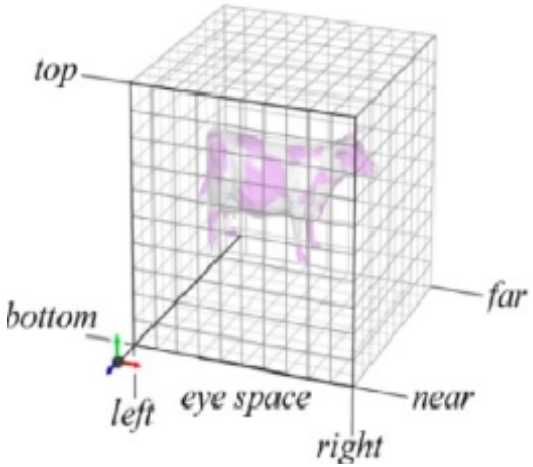
Découpage (Clipping)

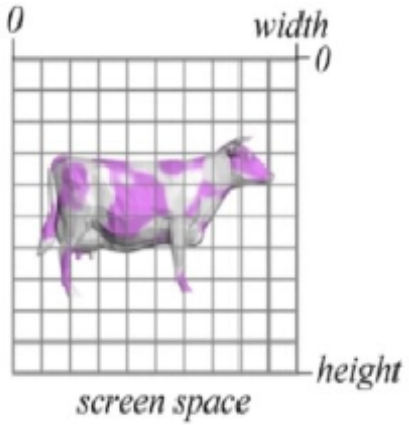
Projection (dans l'espace écran)

Rastérisation

Visibilité/Affichage







Processus de visualisation OpenGL

- Quatre transformations successives utilisées au cours du processus de création d'une image:
  - Transformation de modélisation (Model)
    - Permet de créer la scène à afficher par création, placement et orientation des objets qui la composent.
  - Transformation de visualisation (View)
    - Permet de fixer la position et l'orientation de la caméra de visualisation.
  - Transformation de projection (Projection)
    - Permet de fixer les caractéristiques optiques de la caméra de visualisation (type de projection, ouverture, ...).
  - Transformation d'affichage (Viewport)
    - Permet de fixer la taille et la position de l'image sur la fenêtre d'affichage.

Processus de visualisation OpenGL

- Les transformations de visualisation et de modélisation n'en forment qu'une pour OpenGL (transformation **modelview**). Cette transformation fait partie de l'environnement OpenGL.
- La transformation de projection existe en tant que telle dans OpenGL, et fait elle aussi partie de l'environnement OpenGL.
- Chacune de ces deux transformations peut être modifiée indépendamment de l'autre ce qui permet d'obtenir une indépendance des scènes modélisées vis à vis des caractéristiques de la "caméra" de visualisation.
- La transformation d'affichage est elle aussi paramétrable en OpenGL.

Processus de visualisation OpenGL

- `void glMatrixMode(GLenum mode);`
  - mode: transformation sur laquelle (GL_MODELVIEW ou GL_PROJECTION) les transformations géométriques à venir vont être composées de manière incrémentale.
  - Pour réaliser un affichage, glMatrixMode est généralement appelé successivement une fois sur chacun des deux paramètres de manière à établir les matrices **modelview** et **projection**.
  - Ces appels sont habituellement réalisés au sein de la fonction reshape si la librairie GLUT est utilisée.
  - Dans la fonction display de GLUT, l'habitude veut que l'on ne travaille qu'en modelview.

Processus de visualisation OpenGL

- Exemple

```
void reshape (int w, int h)
{glViewport (0, 0, (GLsizei) w, (GLsizei) h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-2.0, 3.0, -2.0, 3.0, -5.0, 5.0); //Système de coordonnées
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();}
```

Transformations géométriques

- `void glLoadIdentity(void);`
  - Affecte la transformation courante avec la transformation identité.
- `void glLoadMatrix{f d}(const TYPE *m);`
  - Affecte la transformation courante avec la transformation caractérisée mathématiquement par la matrice m (16 valeurs en coordonnées homogènes).
- `void glMultMatrix{f d}(const TYPE *m);`
  - Compose la transformation courante par la transformation de matrice m (16 valeurs en coordonnées homogènes).

Transformations géométriques

- `void glTranslate{f d}(TYPE x,TYPE y,TYPE z);`
  - Compose la transformation courante par la translation de vecteur (x,y,z). Très utilisé en modélisation.
- `void glRotate{f d}(TYPE a,TYPE dx,TYPE dy,TYPE dz);`
  - Compose la transformation courante par la rotation d'angle **a** degrés autour de l'axe (**dx**,**dy**,**dz**) passant par l'origine. Très utilisé en modélisation.
- `void glScale{f d}(TYPE rx,TYPE ry,TYPE rz);`
  - Compose la matrice courante par la transformation composition des affinités d'axe **x**, **y** et **z**, de rapports respectifs **rx**, **ry** et **rz** selon ces axes. Très utilisé en modélisation.

Transformations géométriques

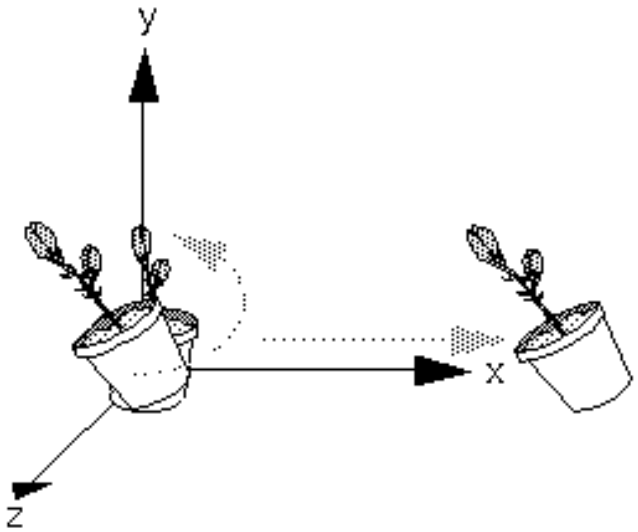
- Exemple:

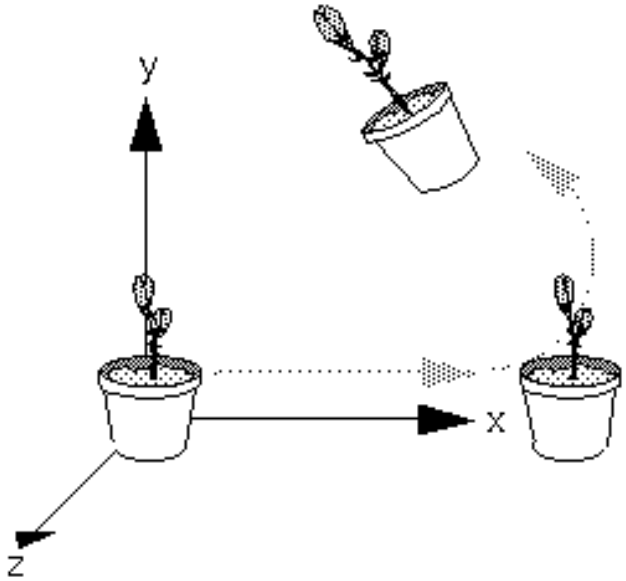
```
glLoadIdentity(); // C = I
glMultMatrixf(N); // C = N
glMultMatrixf(M); // C = NM
glMultMatrixf(L); // C = NML
glBegin(GL_POINTS);
glVertex3f(v);          // N(M(Lv))
glEnd();
```

- Ici L est appliqué en premier sur le point et N en dernier.

Transformations géométriques

- L'ordre est important
- Exemple: une rotation de 45° et une translation



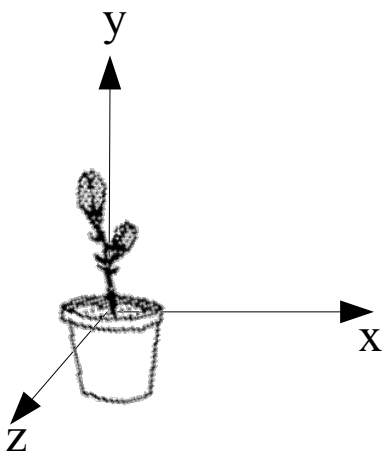



```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glMultMatrixf(T);  // trans
glMultMatrixf(R);  // rot
draw_the_object();
```

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glMultMatrixf(R);  // rot
glMultMatrixf(T);  // trans
draw_the_object();
```

Transformations géométriques

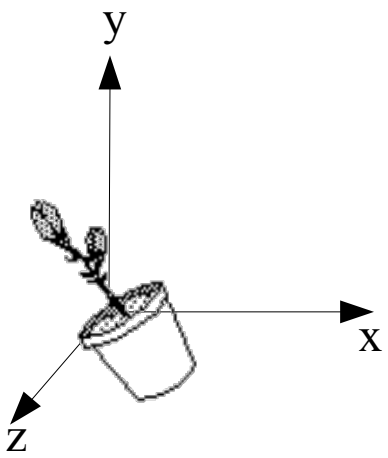
- Deux intuitions cohabitent
- Exemple: une rotation de 45° et une translation




```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glMultMatrixf(T);      // trans
glMultMatrixf(R);      // rot
draw_the_object();
```

Transformations géométriques

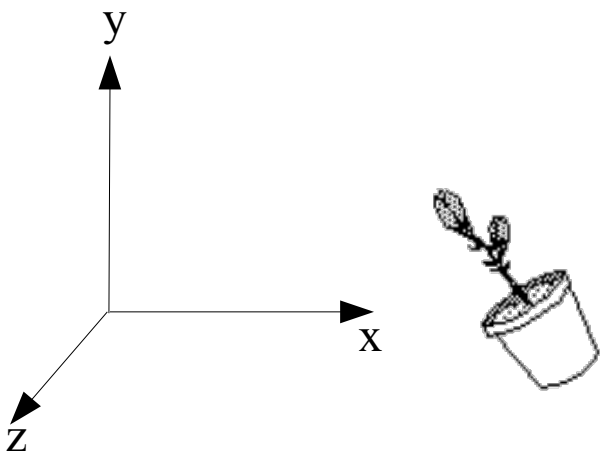
- Le repère est fixe
- Lecture du code à l'envers!




```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glMultMatrixf(T);       // trans
glMultMatrixf(R);       // rot
draw_the_object();
```

Transformations géométriques

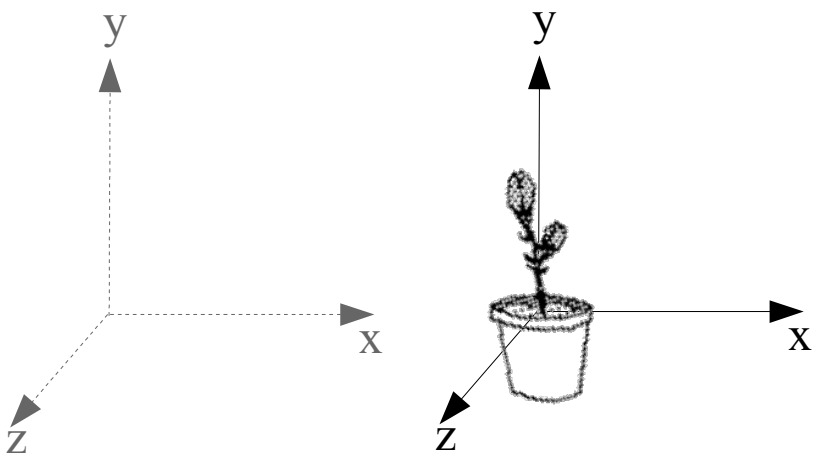
- Le repère est fixe
- Lecture du code à l'envers!




```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glMultMatrixf(T);       // trans
glMultMatrixf(R);       // rot
draw_the_object();
```

Transformations géométriques

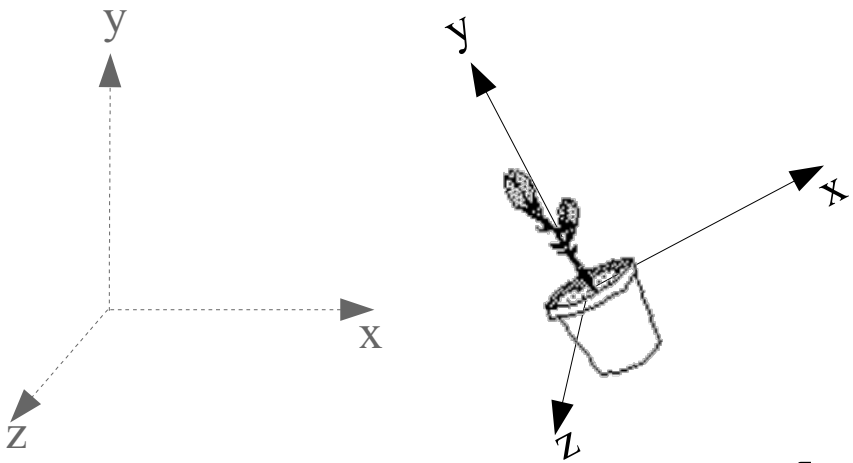
- Le repère bouge avec les transformations
- Lecture du code classique




```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glMultMatrixf(T);       // trans
glMultMatrixf(R);       // rot
draw_the_object();
```

Transformations géométriques

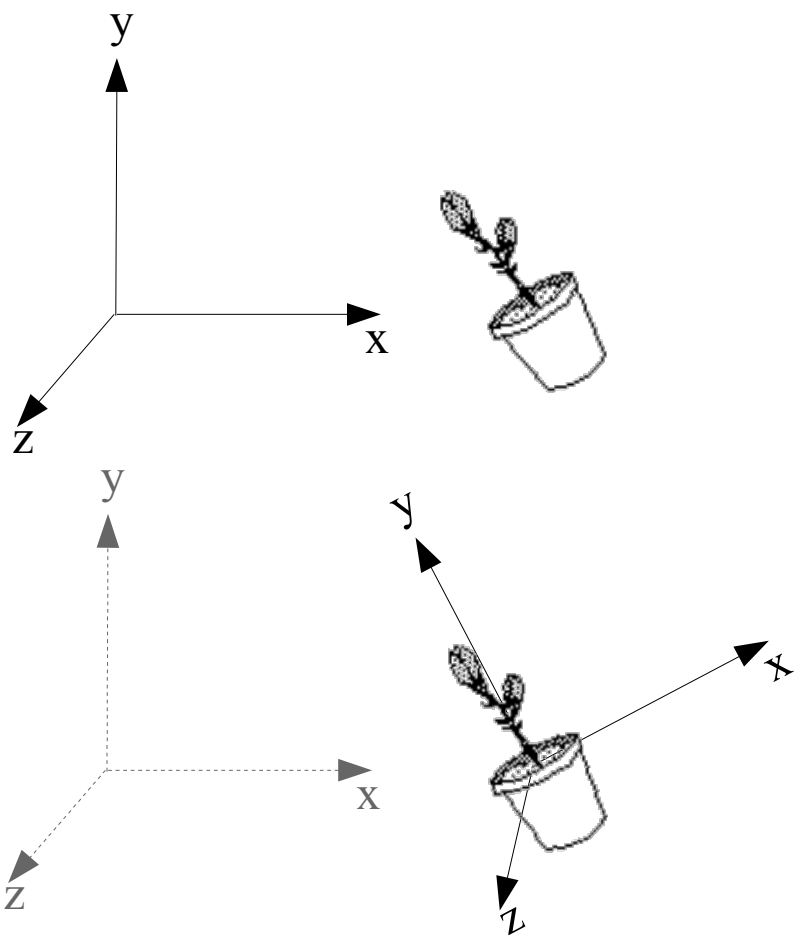
- Le repère bouge avec les transformations
- Lecture du code classique




```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glMultMatrixf(T);       // trans
glMultMatrixf(R);       // rot
draw_the_object();
```

Transformations géométriques

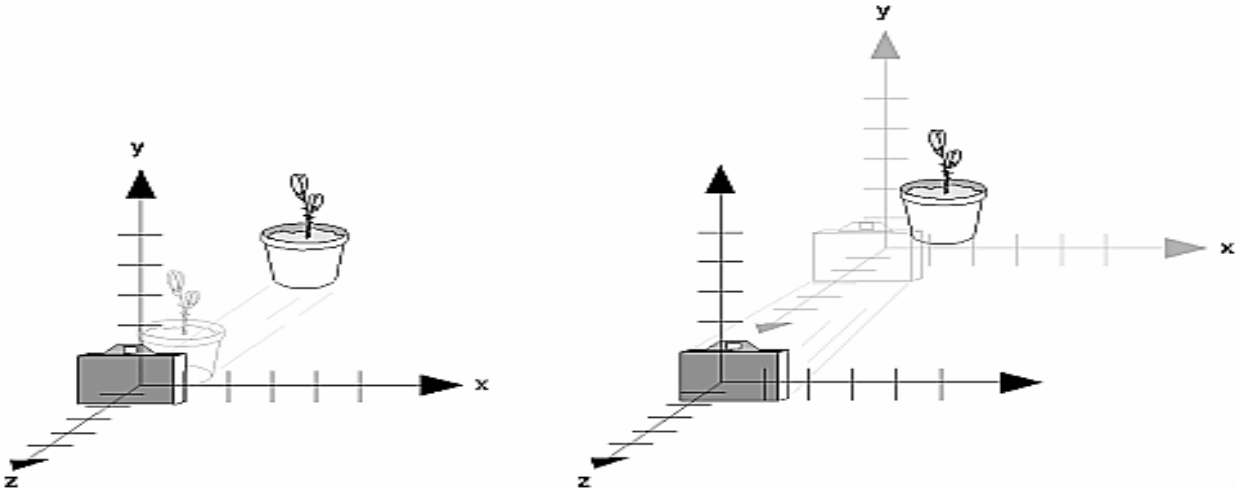
- Dans les deux cas le résultat est le même!




```
glMatrixMode(GL_MODELVIEW
  );
glLoadIdentity();
glMultMatrixf(T);
glMultMatrixf(R);
draw_the_object();
```

Dualité de la modélisation et de la visualisation

- Les effets sont relatifs
  - Translater un objet de (0,0,-5) a le même effet que de translater la caméra de (0,0,5)



➔ Transformations de modélisation+Transformations de visualisation = Transformations ModelView

Transformation spécifique à la visualisation

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

- 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). (upx,upy,upz) indique quelle est la direction du repère courant (fréquemment le repère global) qui devient la direction y (0.0, 1.0, 0.0) dans le repère écran. gluLookAt est une fonction de la bibliothèque GLU.

Exemple:

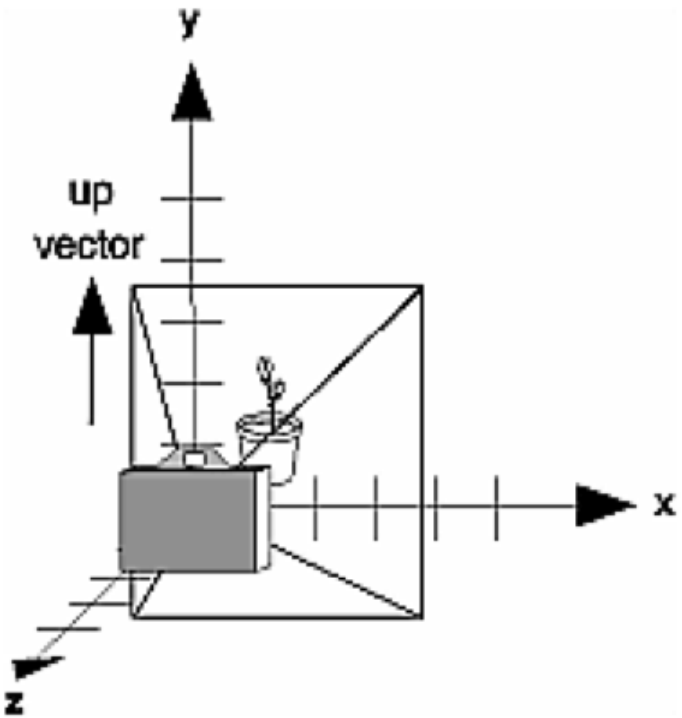
```
gluLookAt(10.0,15.0,10.0,3.0,5.0,-
2.0,0.0,1.0,0.0)
```

- 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

```
gluLookat (0.0, 0.0, 0.0, 0.0, 0.0, -100.0,
0.0, 1.0, 0.0);
```

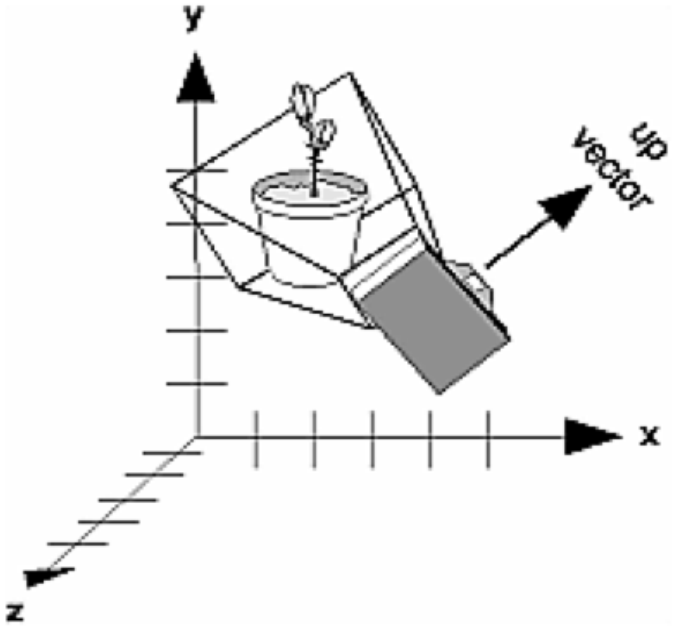
- La Valeur z du point de référence est -100.0
- Mais n'importe quel négatif eu suffit.
- C'est la position par défaut
- L'appel est superflu.



gluLookAt

```
gluLookAt(4.0, 2.0, 1.0, 2.0, 4.0, -3.0, 2.0, 2.0, -1.0);
```

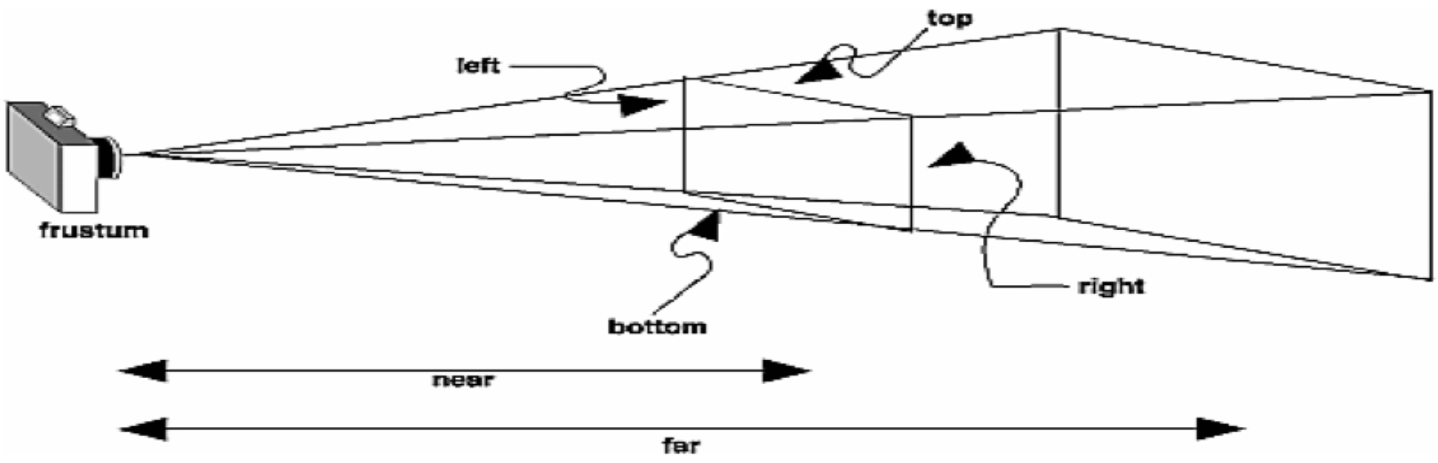
- Position de la caméra :(4, 2, 1)
- Point de référence en (2, 4, -3)
- Vecteur d'orientation: (2, 2, -1) ce qui donne une rotation du point de vue c



Transformations de projection

```
void glFrustum(GLdouble g, GLdouble d, GLdouble
b, GLdouble h, GLdouble cmin, GLdouble cmax);
```

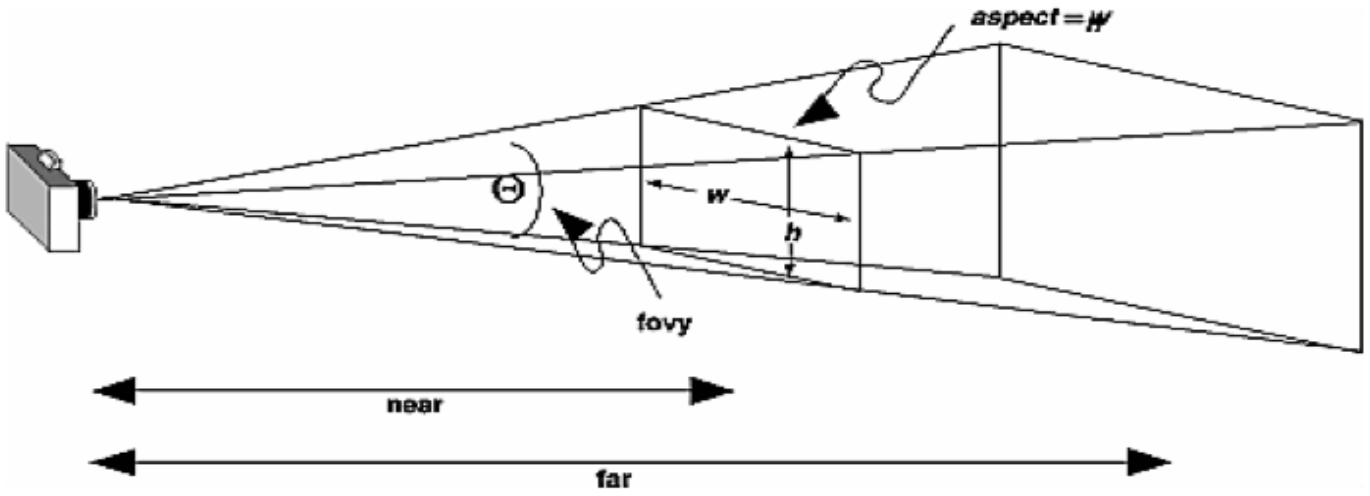
- Compose la transformation courante par la transformation de projection en perspective de volume de visualisation défini par la pyramide tronquée de sommet l'origine O, orientée selon l'axe -z et de plan supérieur défini par la diagonale (g,b,-cmin) (d,h,-cmin).
- cmin et cmax sont les distances entre l'origine et les plans de clipping proche (-cmin en z) et éloignés (-cmax en z). cmin et cmax doi
  cmax car



Transformations de projection

```
void gluPerspective(GLdouble foc, GLdouble
ratio, GLdouble cmin, GLdouble cmax);
```

- 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 l'angle foc comme angle d'ouverture verticale, l'apect-ratio ratio (rapport largeur/hauteur) et les plans de clipping proche et éloignés -cmin et -cmax.cmin et cmax doivent avoir une valeur positive et respecter cmin < cn



Transformations de projection

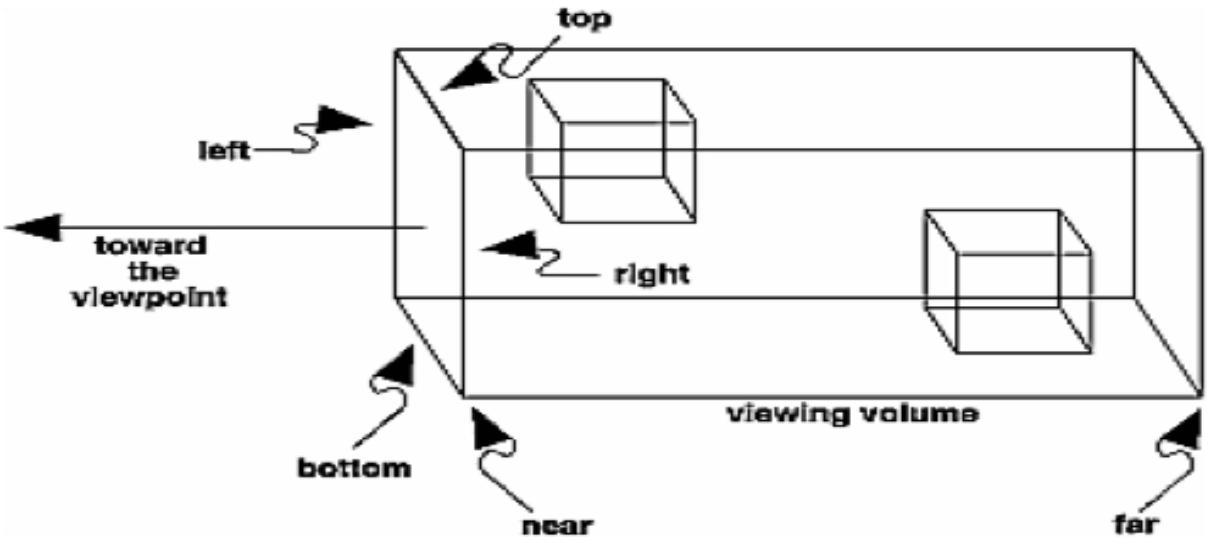
- Exemple:
- `gluPerspective(40.0,1.5,50.0,100.0)`
  - configure une caméra de visualisation en perspective placée à l'origine (c'est obligatoire) et orientée selon l'axe -z (c'est obligatoire) avec:
  - un angle d'ouverture verticale de 40.0°,
  - un angle d'ouverture horizontale de 40.0*1.5 = 60.0°,
  - un plan de clipping qui élimine tous les objets ou morceaux d'objet situés en z > -50.0
  - et un plan de clipping qui élimine tous les objets ou morceaux d'objets situés en z < -100.0.
  - Ces valeurs sont considérées dans le repère global.

Transformations de projection

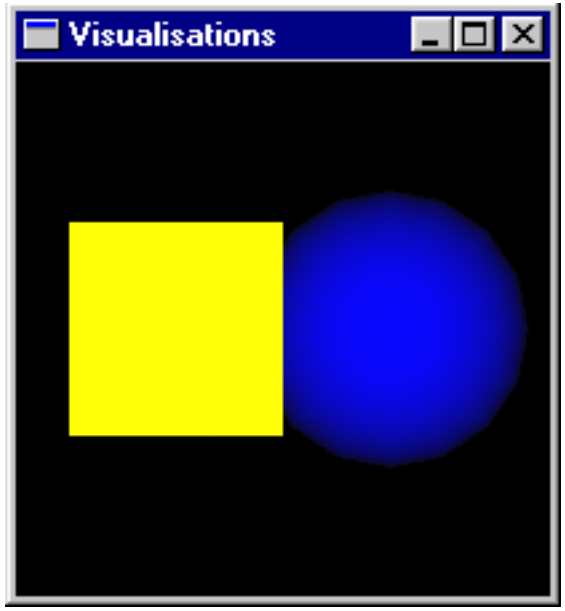
```
void glOrtho(GLdouble g, GLdouble d, GLdouble
b, GLdouble h, GLdouble cmin, GLdouble cmax);
```

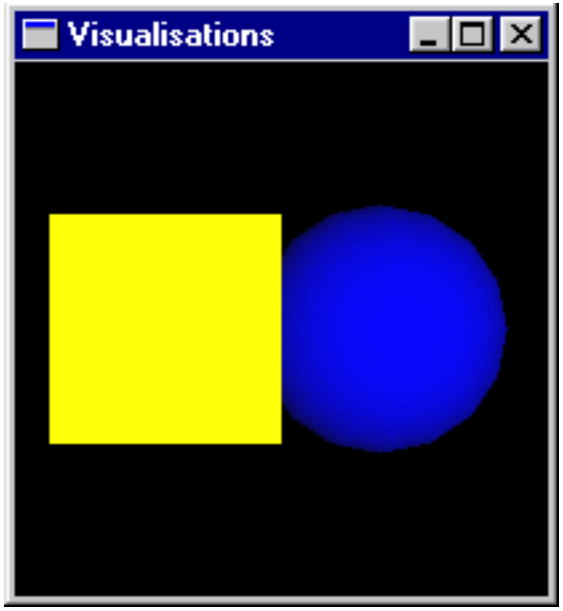
Compose la transformation courante par la transformation de projection orthographique selon l'axe -z et définie par le volume de visualisation parallélépipèdique (g,d,b,h,-cmin,-cmax).

cmin et cmax peuvent être positifs ou négatifs et mais doivent respecter cmin < cmax.

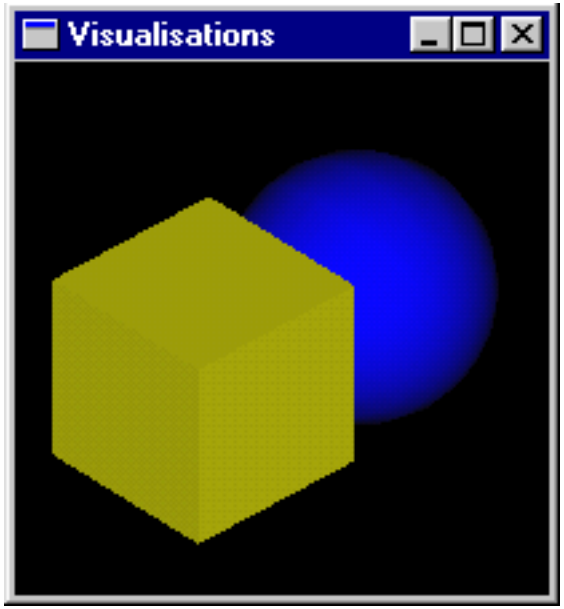


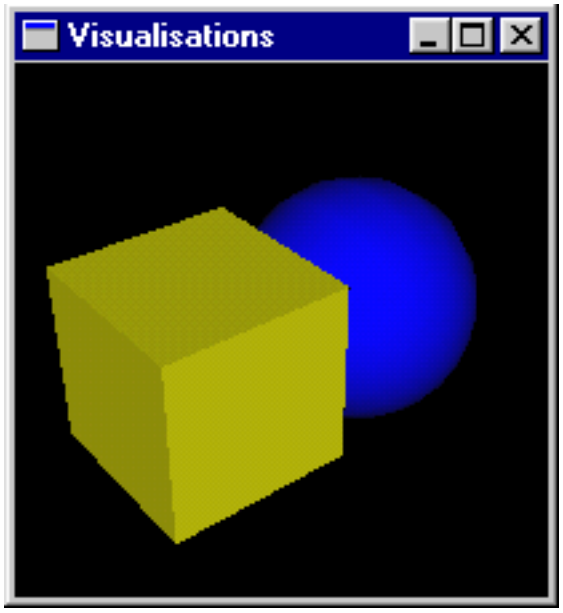
Visualisation en projection parallèle et projection en perspective



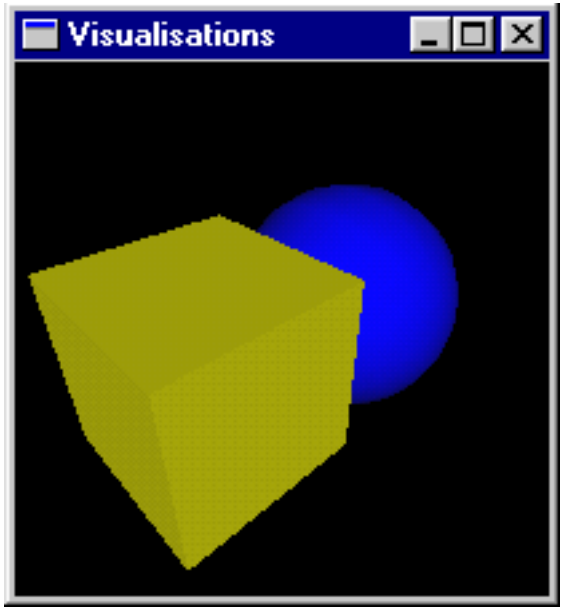


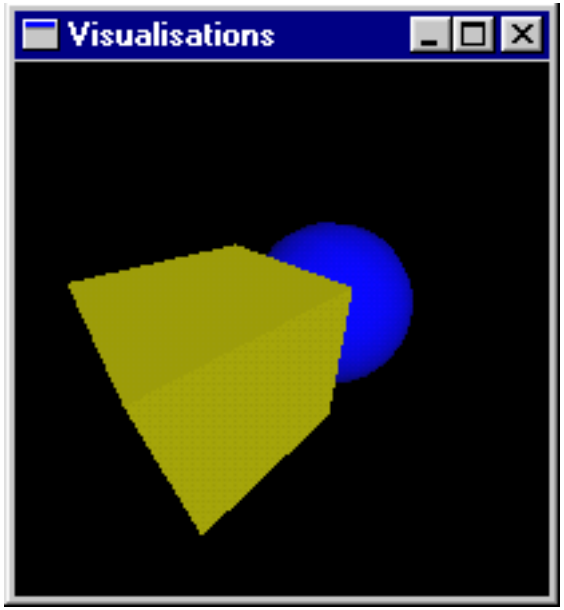
Visualisation en projection parallèle et projection en perspective



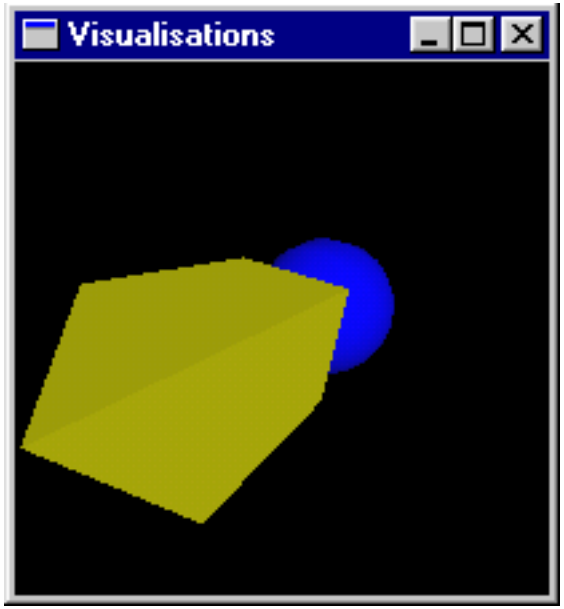


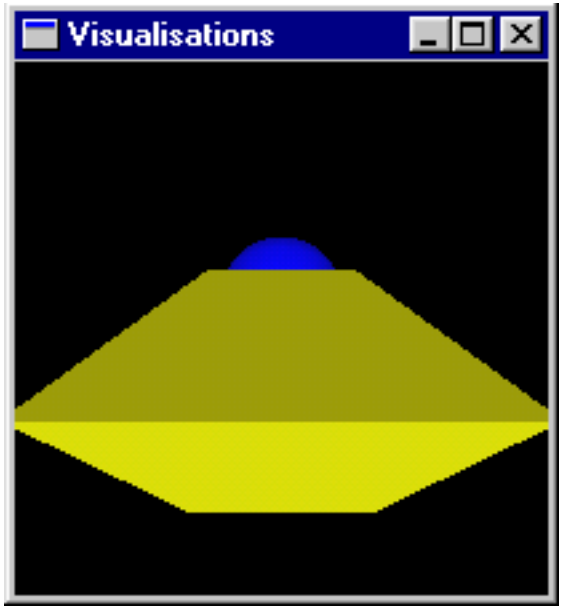
Visualisation en projection en perspective

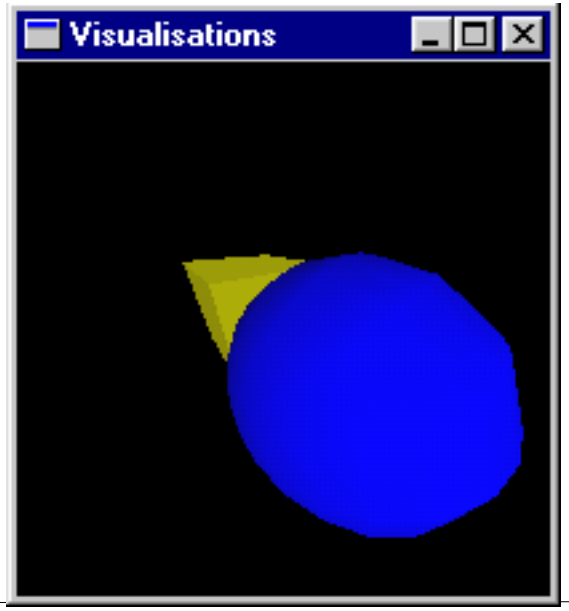




Visualisation en projection en perspective
Avec rapprochement de scène



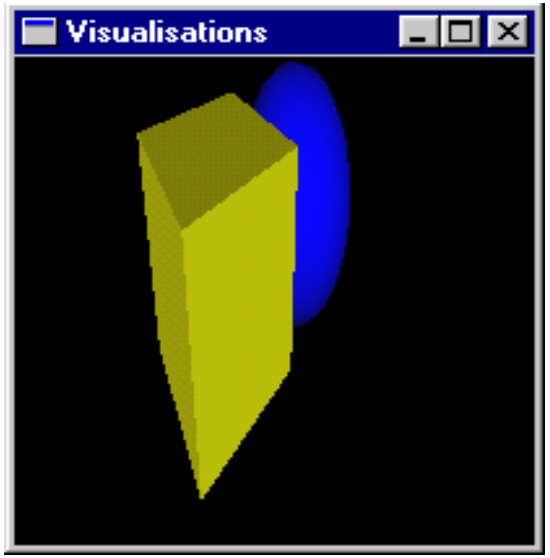


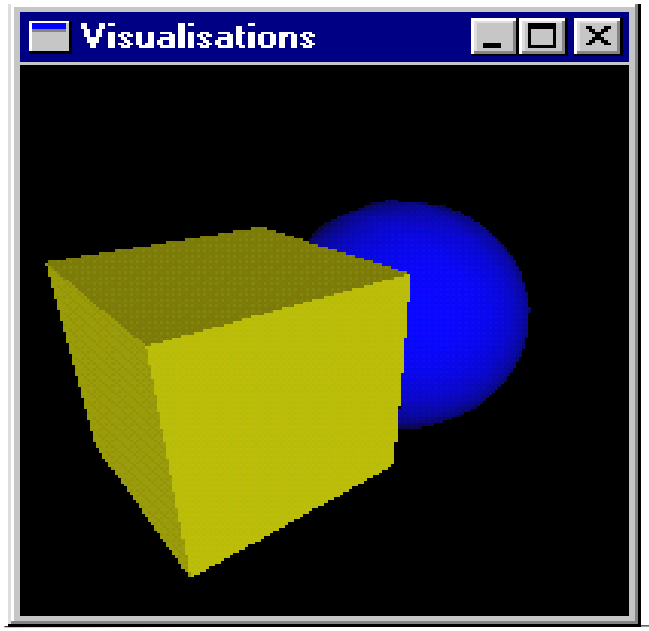


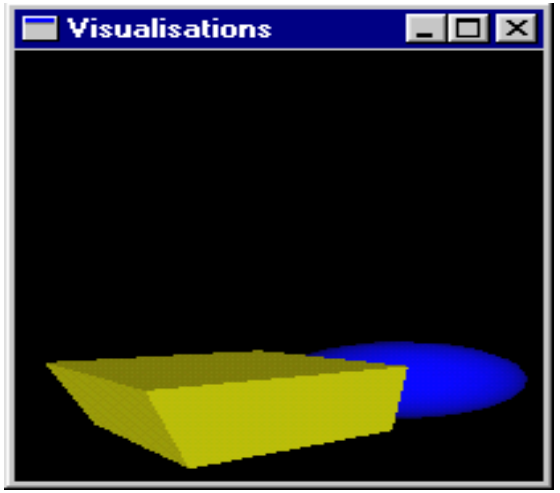
Transformation d'affichage

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

Définit le rectangle de pixels dans la fenêtre d'affichage dans lequel l'image calculée sera affichée. (position et taille dans la fenêtre)

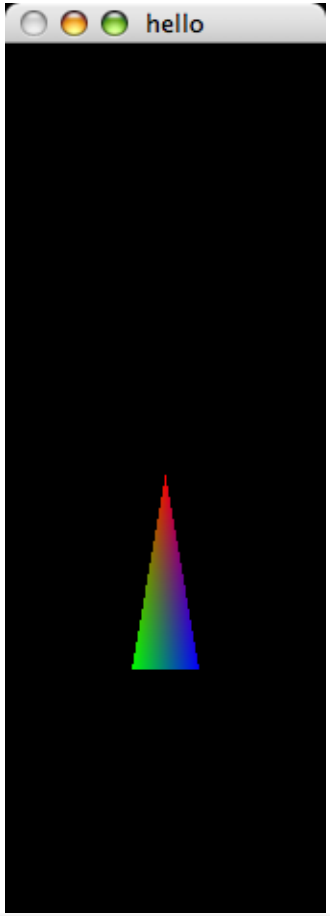


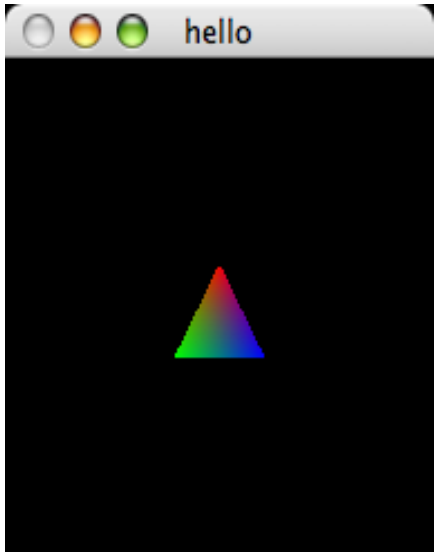


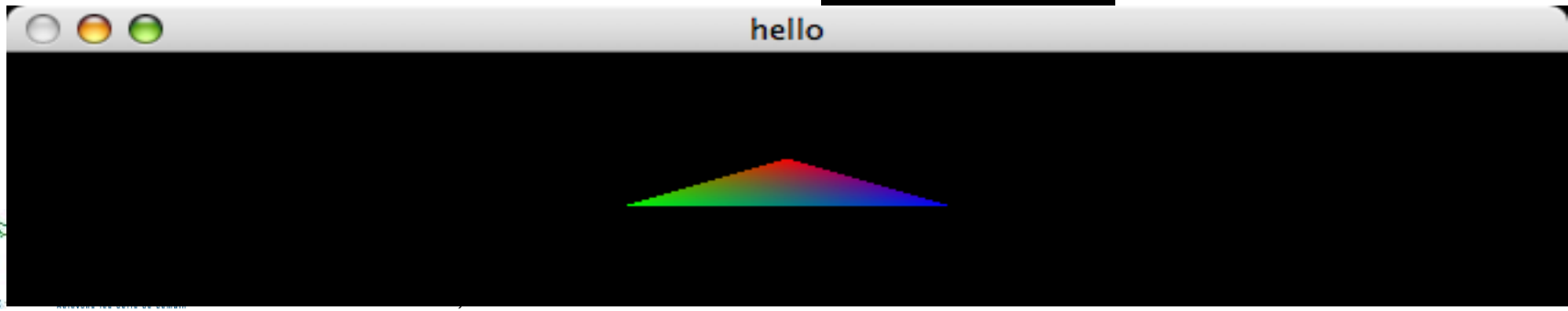


Transformation d'affichage

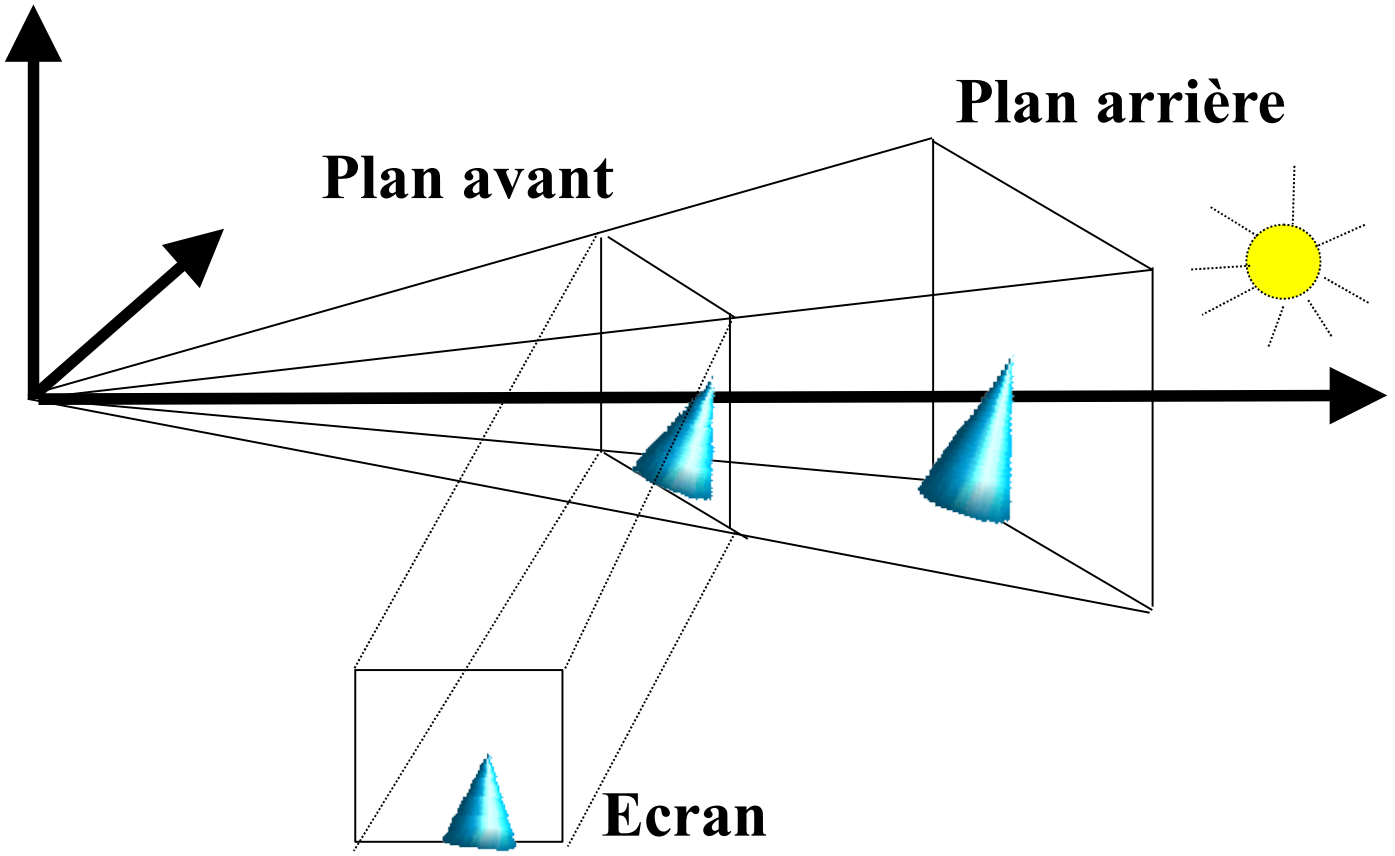
- Si le rapport du volume visualisé n'est pas égal à celui du volume de visualisation une distorsion apparait (les carrés deviennent des rectangles et les cercles des ellipses).
- Exemple distordu:
  - gluPerspective(fovy, 1.0, near, far);
  - glViewport(0,0,800,400) ;
- Exemple non distordu:
  - gluPerspective(fovy, 1.0, near, far);
  - glViewport(0,0,400,400) ;







Résumé



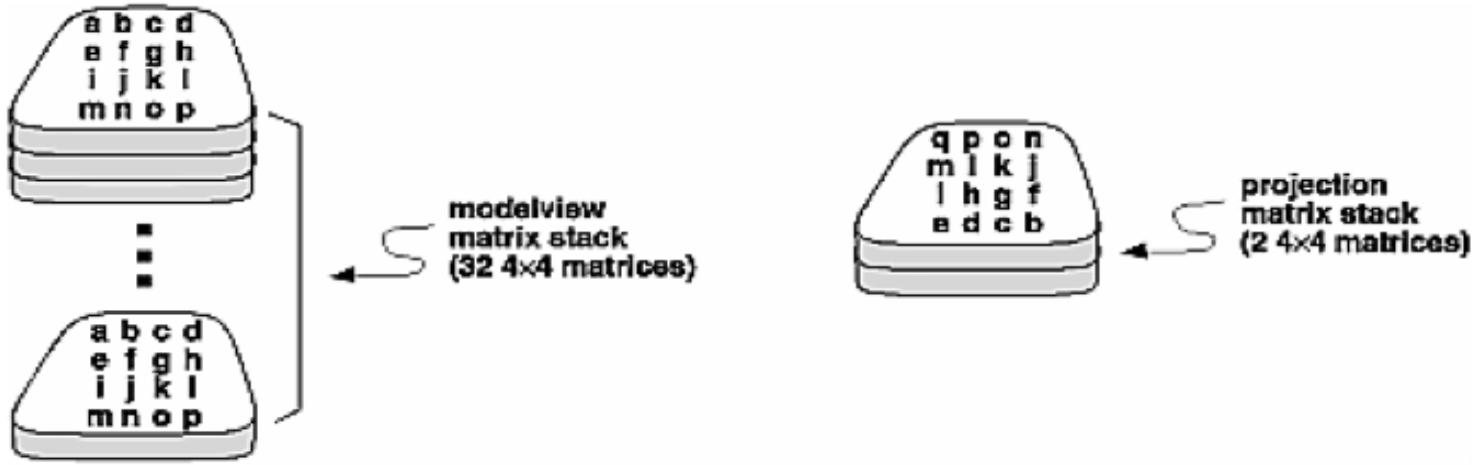
Pile de matrices

- `void glPushMatrix(void);`

Empile la matrice courante dans la pile de matrices.

- `void glPopMatrix(void);`

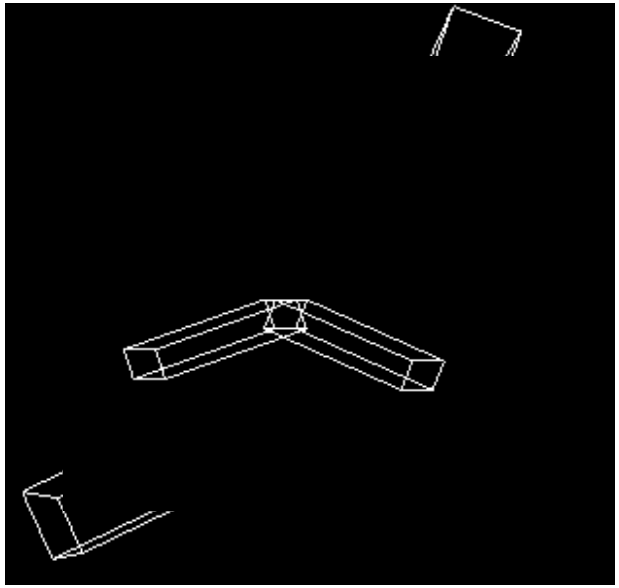
Dépile la matrice en haut de pile et remplace la matrice courante par celle-ci.



Pile de matrices

```
glClear (GL_COLOR_BUFFER_BIT);
   glPushMatrix();
   glTranslatef (-1.0, 0.0, 0.0);
   glRotatef ((GLfloat) shoulder,
0.0, 0.0, 1.0);
   glTranslatef (1.0, 0.0, 0.0);
   glPushMatrix();
   glScalef (2.0, 0.4, 1.0);
   glutWireCube (1.0);
   glPopMatrix();
   glTranslatef (1.0, 0.0, 0.0);
   glRotatef ((GLfloat) elbow,
0.0, 0.0, 1.0);
   glTranslatef (1.0, 0.0, 0.0);
```

```
glPushMatrix();
 glScalef (2.0, 0.4,
1.0);
 glutWireCube (1.0);
 glPopMatrix();
 glPopMatrix();
 glutSwapBuffers();
```



Plans de clipping

- En plus des six plans de clipping il est possible d'en créer 6 autres:

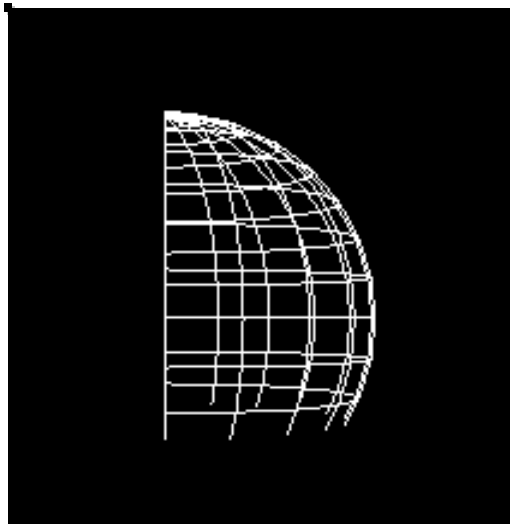
```
void glClipPlane( GLenum plane,
                  const GLdouble *equation )
```

- plane : Nom du plan ex: GL_CLIP_PLANEi
- equation: `GLdouble eqn[4] = {a, b, c, d};`

qui s'interprète comme l'équation d'un plan.

- On coupe tout ce qui vérifie:

ax+by+cz+d<0



```
GLdouble eqn[4] = {0.0, 1.0, 0.0,0.5};
GLdouble eqn2[4] = {1.0, 0.0,0.0,0.0};
```