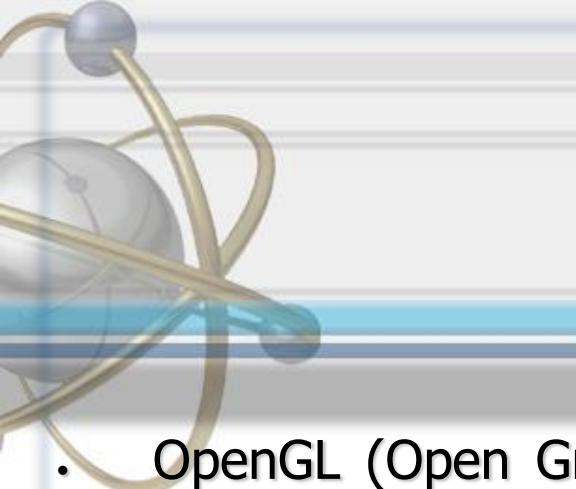
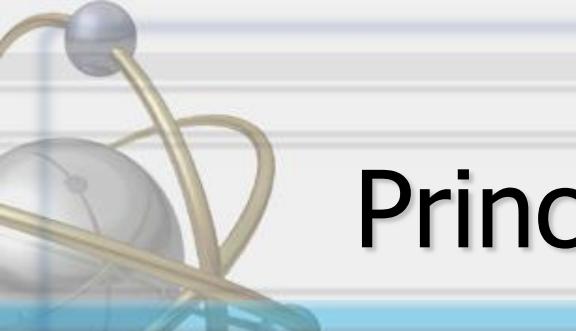


M. AMMI  
[ammi@limsi.fr](mailto:ammi@limsi.fr)



# Histoire

- OpenGL (Open Graphics Library) est une spécification qui définit une API multi plate-forme pour la conception d'applications générant des images 3D (mais également 2D).
- La spécification s'est traduite en une librairie graphique 3D/2D.
  1. On lui donne des ordres de tracé de primitives graphiques (facettes, etc.) directement en 3D, une position de caméra, des lumières, des textures à plaquer sur les surfaces, etc.
  2. OpenGL se charge ensuite de faire les changements de repère, la projection en perspective à l'écran, le clipping, l'élimination des parties cachées, d'interpoler les couleurs, et de rasteriser (tracer ligne à ligne) les faces pour en faire des pixels (image finale)



# Principales fonctionnalités

- Spécification en cours : la version 2.1 (2006)

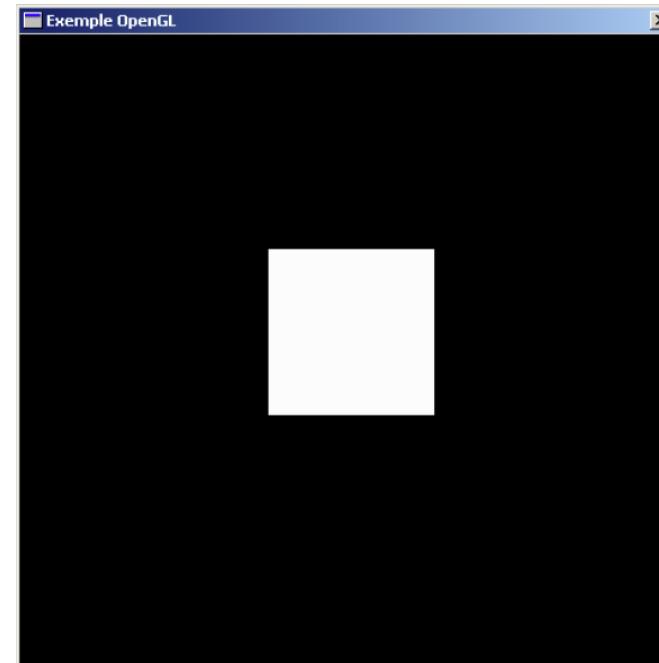
- Environ 250 instructions distinctes

- **Primitives Géométriques** Vous permettent de construire des descriptions mathématiques d'objets à partir de points, lignes, polygones, images et bitmaps.
- **Codage de couleur** en RGBA (Rouge-Verte-Bleue-Alpha) ou en mode index de couleur.
- **Visualisation et Modelage** permettent d'arranger les objets dans une scène tri-dimensionnelle, de bouger les caméras dans l'espace et de choisir le point de vue d'où sera visualisé la scène.
- **Projection de Texture** apporte du réalisme au modèle en donnant un aspect réaliste aux surfaces des polygones.
- **Eclairage des Matériaux** est une partie indispensable de tout infographie 3D. OpenGL fournit les commandes pour calculer la couleur de n'importe quel point à partir des propriétés des matériaux et des sources de lumières dans la pièce.
- **Double Tampon** élimine les clignotements dans les animations. Chaque image de l'animation est construite dans une mémoire tampon séparée et affichée quand le tracé est terminé.
- **Anti-repliement** réduit les lignes en zig-zag sur les écrans. Elles apparaissent surtout en basse résolution. L'anti-repliement est une technique classique qui consiste à modifier la couleur et l'intensité des pixels en bordure de ligne pour atténuer l'effet de zig-zag.
- **Ombrage de Gouraud** est une technique qui applique des ombrages réguliers aux objets 3D et donne de subtiles différences de couleurs sur la surface.
- **Tampon-Z** mémorise la coordonnée Z d'un objet 3D. Le tampon Z sert à mémoriser la proximité d'un objet par rapport à l'observateur. Sa fonction est aussi cruciale pour la suppression des parties cachées.
- **Effets Atmosphériques** comme le brouillard, la fumée et le voilage rendent les images numériques plus réalistes. Sans ces effets, les images apparaissent parfois trop piquées et trop bien définies. *Fog* est un terme qui décrit un algorithme qui simule la brume, le crachin, la fumée, la pollution ou simplement l'air en ajoutant de la profondeur à l'image.
- **Mélange Alpha** utilise la valeur Alpha (valeur de diffusion du matériau) du code RGBA afin de combiner la couleur d'un fragment en cours de traitement avec celle d'un pixel déjà stocké dans le tampon d'image. Imaginez par exemple devoir tracer une fenêtre transparente bleue devant une boîte rouge. Le mélange Alpha permet de simuler la transparence de la fenêtre de telle sorte que la boîte vue à travers la vitre soit violette.
- **Plans masqués** restreint le tracé à certaines portions de l'écran.
- **Listes d'affichage** permettent le stockage de commandes de tracé dans une liste pour un affichage ultérieur. Avec une utilisation correcte, les listes d'affichages peuvent améliorer nettement les performances.
- **Evaluateurs Polynomiaux** supportent les Splines non uniformes rationnelles-B. Cela permet de tracer des courbes régulières avec quelques points ce qui évite d'en stocker de nombreux intermédiaires.
- **Retour, Sélection et Choix** sont des possibilités qui permettent aux applications d'autoriser l'utilisateur à sélectionner une région de l'écran ou de choisir un objet dessiné à l'écran. Le mode Retour, permet au développeur d'obtenir les résultats des calculs d'affichage.
- **Primitives d'Affichage** (Octets de l'affichage et rectangles de pixels)
- **Opérations sur les Pixels**
- **Transformations:** rotation, échelles, translations, perspectives en 3D, etc.

# Exemple

```
#include <gl.h>
...
void main() {
...
    glClearColor(0.0,0.0,0.0,0.0) ;
    glClear(GL_COLOR_BUFFER_BIT) ;
    glOrtho(-1.0,1.0,-1.0,1.0,-1.0,1.0) ;
    glColor3f(1.0,1.0,1.0) ;
    glBegin(GL_POLYGON) ;
    glVertex2f(-0.5,-0.5) ;
    glVertex2f(-0.5,0.5) ;
    glVertex2f(0.5,0.5) ;
    glVertex2f(0.5,-0.5) ;
    glEnd() ;

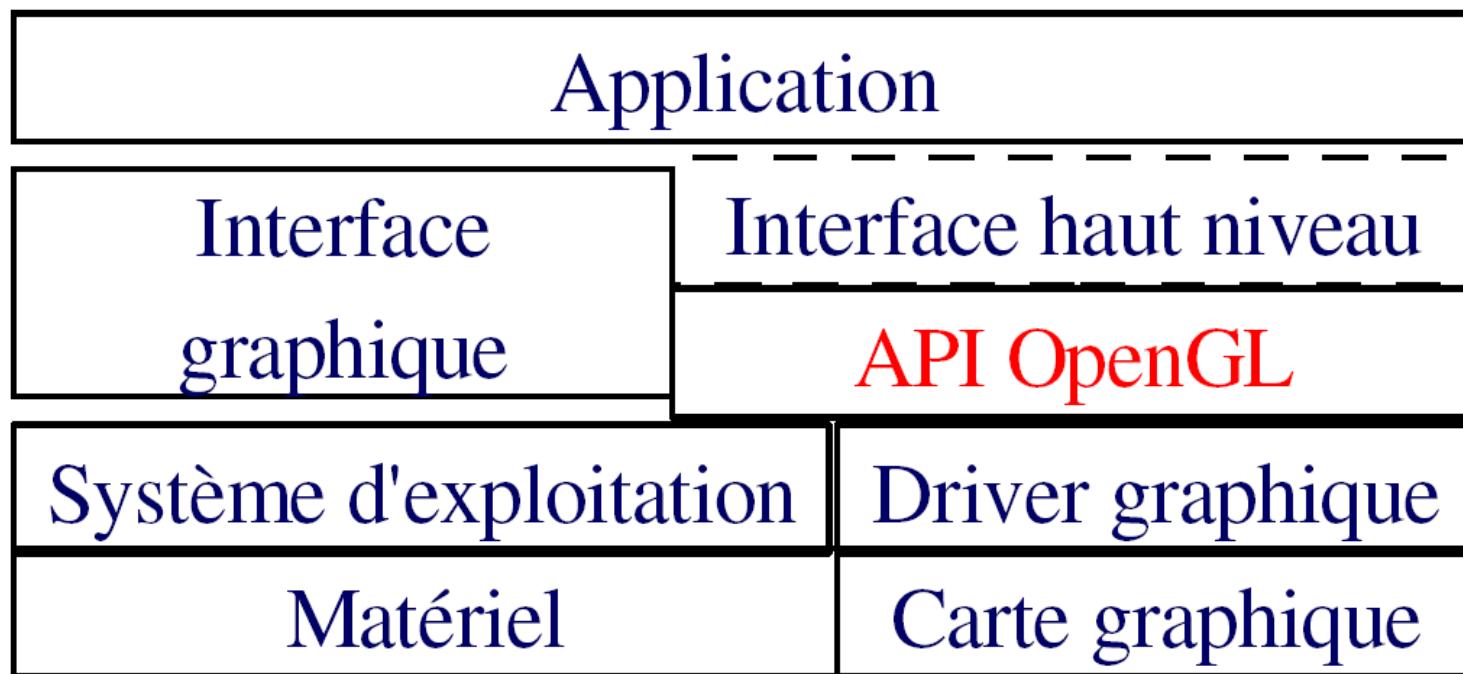
    glFlush() ;
...
}
```



Ce programme dessine un carré blanc au centre d'une fenêtre de fond noir



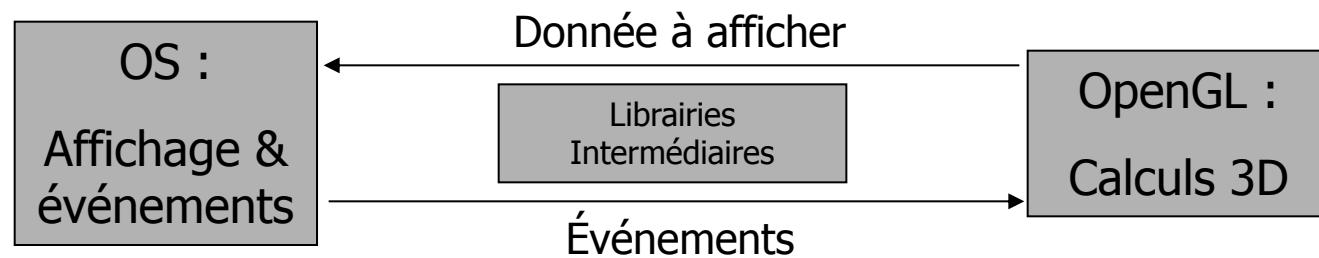
# OpenGL & le système d'exploitation



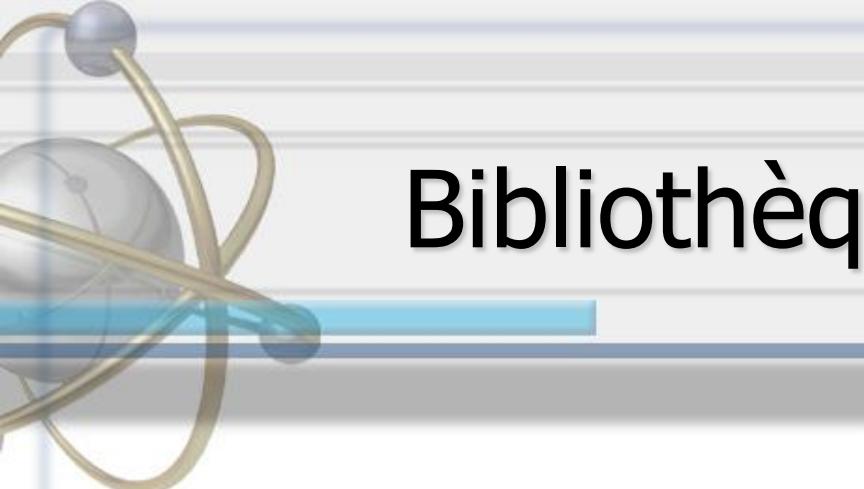
# OpenGL & le système d'exploitation

- Des bibliothèques construites au dessus d'OpenGL fournissent des fonctions de plus haut niveau :

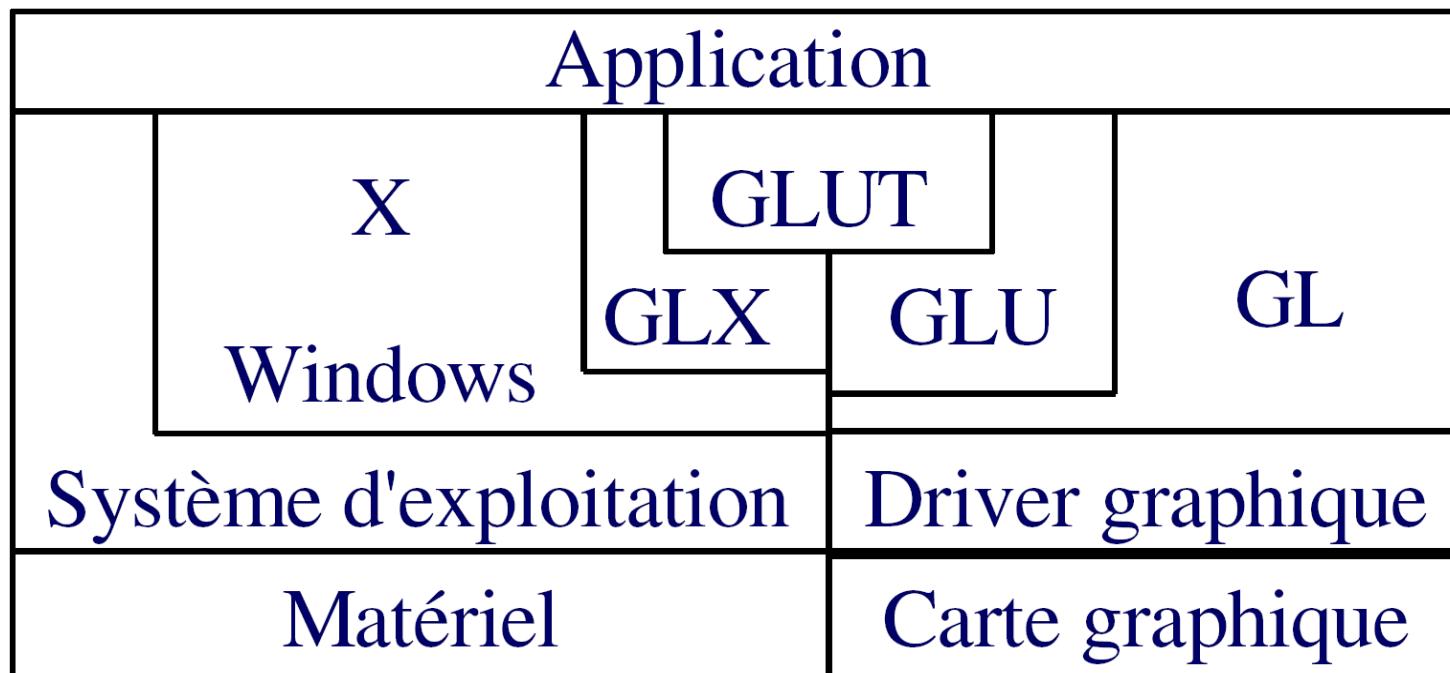
- GLUT
- GLX
- AUX
- etc.



- Programmation simplifiée
- Indépendants de l'OS
- Ne font pas officiellement partie d'OpenGL
- Gestion des fenêtres, clavier et souris
- Primitives avancées : Cube, sphères, etc.



# Bibliothèques d'OpenGL





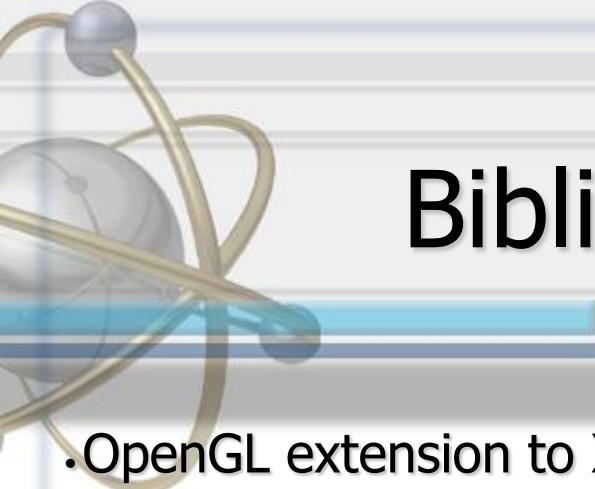
# Bibliothèques d'OpenGL

## • OpenGL Library (GL)

- Librairie standard proposant les fonctions de base pour l'affichage en OpenGL
- Pas de fonction pour la construction d'une interface utilisateur (fenêtres, souris, clavier, ...)
- Préfixe des fonctions: gl (glVertex3f)

## • OpenGL Utility Library (GLU)

- Librairie standard proposant des commandes bas-niveau écrites en GL:
  - certaines transformations géométriques
  - triangulation des polygones
  - rendu des surfaces paramétriques et quadriques
  - ...
- Préfixe des fonctions: glu (gluPerspective)



# Bibliothèques d'OpenGL

## .OpenGL extension to X-Windows (GLX)

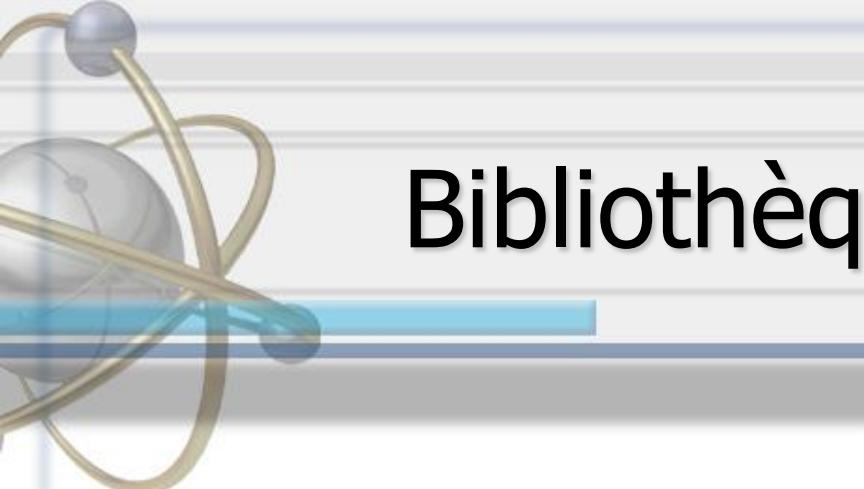
- Utilisation d'OpenGL en association avec X Windows pour l'interface utilisateur
- Préfixe des fonctions: glX (glXCreateWindow)

## .Auxiliary Library (Aux)

- Bibliothèque écrite pour rendre simple la programmation de petites applications dans le cadre d'interfaces graphiques interactives simples:

- gestion d'une fenêtre d'affichage
- gestion de la souris
- gestion du clavier
- ...

- Préfixe des fonctions: aux (auxInitWindow)

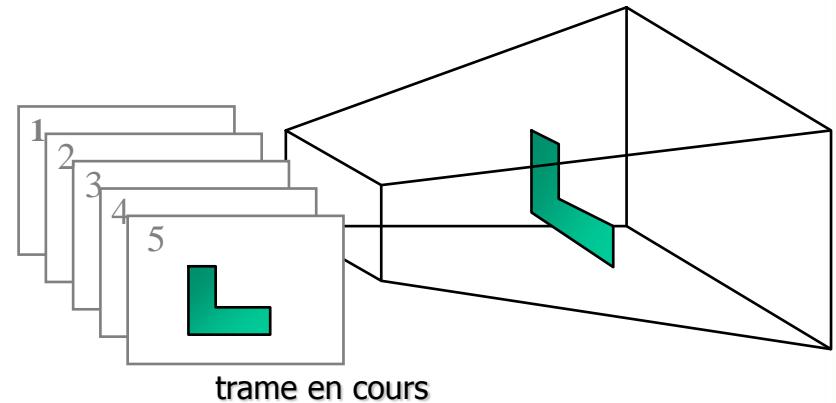


# Bibliothèques d'OpenGL

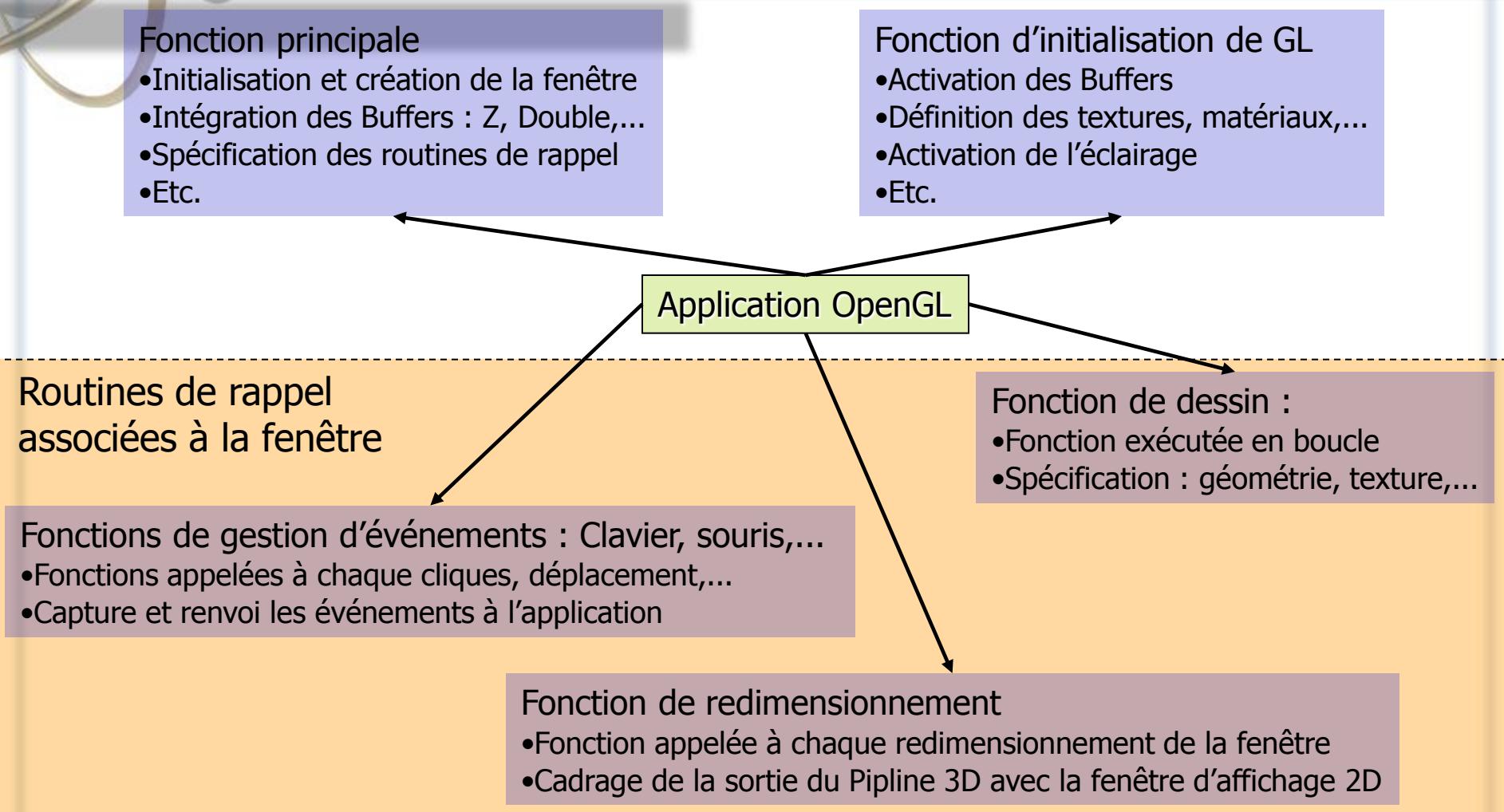
- OpenGL Utility Toolkit (GLUT)
  - Équivalent à Aux
  - Interface utilisateur programmable plus élaborée (multifenêtre, menus popup) -> plus grande complexité
  - Préfixe des fonctions: glut (glutInitWindowSize)
  - La plus utilisée en association avec OpenGL

# Organisation d'une application OpenGL

```
main() {  
    // Initialisation de l'application:  
    // - Ouverture d'une fenêtre  
    // - Chargement & définition des textures, matériaux, etc.  
    while(pas fini) {  
        // Spécification du code de la trame en cours:  
        // - Effacer l'écran  
        // - Spécification de la position & propriétés de la caméra  
        // - Spécification de l'éclairage (activation, etc)  
        for(chaque objet) {  
            // Spécification du code correspondant à l'objet  
            // - Spécification de la position & orientation (matrix) de l'objet  
            for(chaque matériau) {  
                // Spécification des propriétés des matériaux  
                // - propriétés d'éclairage  
                // - propriétés des textures  
                for(chaque primitive) {  
                    // Rendre la primitive (glBegin, glVertex3f...)  
                }  
            }  
        }  
        // Finalisation du code de la trame en cours (glSwapbuffers())  
    }  
    // Code de sortie  
}
```



# Organisation d'une application OpenGL





# Organisation d'une application OpenGL

```
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitWindowSize(largeur, hauteur);
    glutInitWindowPosition(10, 100);
    glutInitDisplayMode(GLUT_RGBA | GLUT_DEPTH | GLUT_DOUBLE);
    glutCreateWindow(argv[0]);

    /* Routines de rappel */
    glutReshapeFunc(reshape);
    glutDisplayFunc(display);
    glutKeyboardFunc(keyboard);
    glutMouseFunc(mouse);

    /* Autres initialisation */
    initGL();

    /* Lancement de la boucle principale */
    glutMainLoop();
    return 0;
}
```



# Organisation d'une application OpenGL

```
void initGL(void)
{
    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LESS);
    glEnable(GL_CULL_FACE);

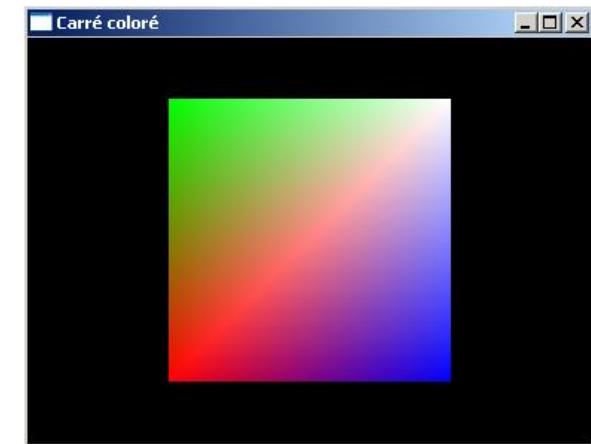
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);

    glLightModelfv(GL_AMBIENT, LAmbientValue);
    glLightfv(GL_LIGHT0, GL_POSITION, Lposition1);
    glLightfv(GL_LIGHT0, GL_AMBIENT, LAmbientValue);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, LDiffuseValue);
    glLightfv(GL_LIGHT0, GL_SPECULAR, LSpecularValue);

    glShadeModel(GL_SMOOTH);
}
```

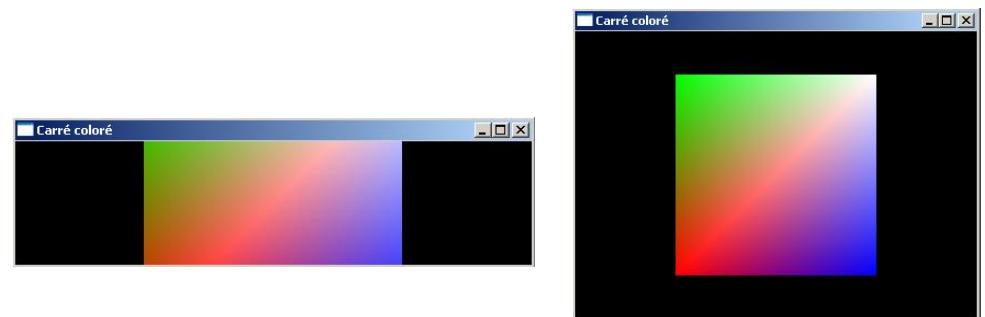
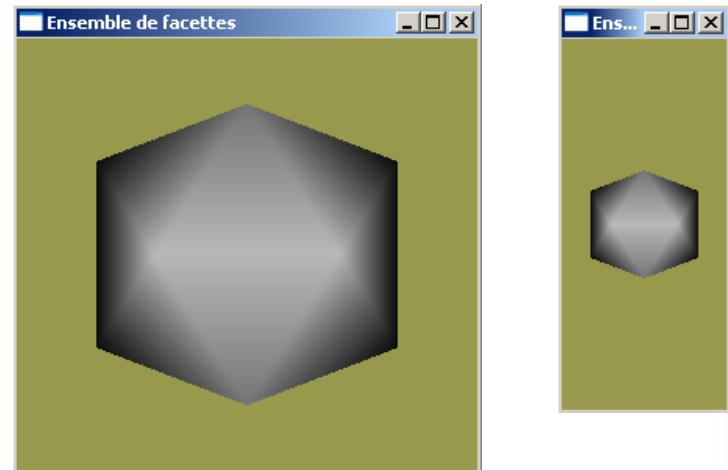
# Organisation d'une application OpenGL

```
void display(void) {  
    glClear(GL_COLOR_BUFFER_BIT);  
  
    glBegin(GL_POLYGON) ;  
    glColor4fv(couleurRouge()) ;  
    glVertex2f(-0.5F,-0.5F) ;  
    glColor4fv(couleurVert()) ;  
    glVertex2f(-0.5F,0.5F) ;  
    glColor4fv(couleurBlanc()) ;  
    glVertex2f(0.5F,0.5F) ;  
    glColor4fv(couleurBleu()) ;  
    glVertex2f(0.5F,-0.5F) ;  
    glEnd() ;  
  
    glFlush();  
  
    glutSwapBuffers();  
}
```



# Organisation d'une application OpenGL

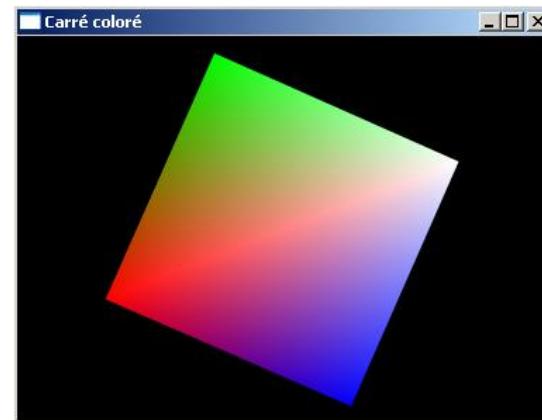
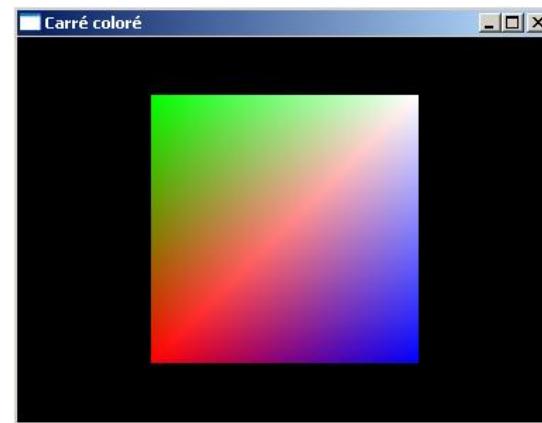
```
void reshape(int large, int haut)
{
    glViewport(0, 0, large, haut);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60.0, (GLfloat)large/(GLfloat)haut, 0.1, 128.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
```



# Organisation d'une application OpenGL

```
void Keyboard(unsigned char key, int x, int y)
{
    printf("vous avez appuyé sur %c ",key);
    if (key==27) exit(0);
}
```

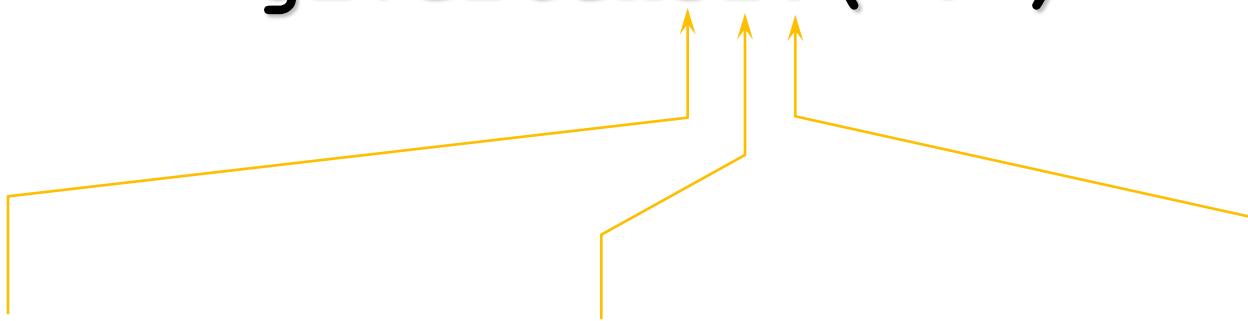
```
void Keyboard(unsigned char key, int x, int y)
{
    Switch (key)
    {
        Case : "A"
            view_rotx=+0.1;
            break();
        Case : "B"
            view_roty=+0.1;
            break();
    }
    glRotatef(view_rotx,1.0,0.0,0.0);
    glRotatef(view_roty,0.0,1.0,0.0);
}
```



# Format des instructions

## • Format des instructions

**glVertex3fv( v )**



**Nombre de données**

2 - (x,y)  
3 - (x,y,z)  
4 - (x,y,z,w)

**Type de donnée**

b - byte  
ub - unsigned byte  
s - short  
us - unsigned short  
i - int  
ui - unsigned int  
f - float  
d - double

**Vecteur**

Ne pas mettre "v"  
pour la version  
scalaire

**glVertex2f( x, y )**



Ex:

`void glVertex2d( GLdouble x, GLdouble y )`

`void glVertex3f( GLfloat x, GLfloat y, GLfloat z )`

`void glVertex4i( GLint x, GLint y, GLint z, GLint w )`

`void glVertex4sv( const GLshort *v )`



# Syntaxe

Type OpenGL	Dans une fonction	Representation	Type de donnée
GLbyte	b	8-bit integer	signed char
GLshort	s	16-bit integer	short
GLint, GLsizei	i	32-bit integer	long
GLfloat	f	32-bit float	float
GLdouble	d	64-bit float	double
GLubyte, GLboolean	ub	8-bit unsigned integer	unsigned char
GLushort	us	16-bit unsigned short	unsigned short
GLuint, GLenum, GLbitfield	ui	32-bit unsigned integer	unsigned long



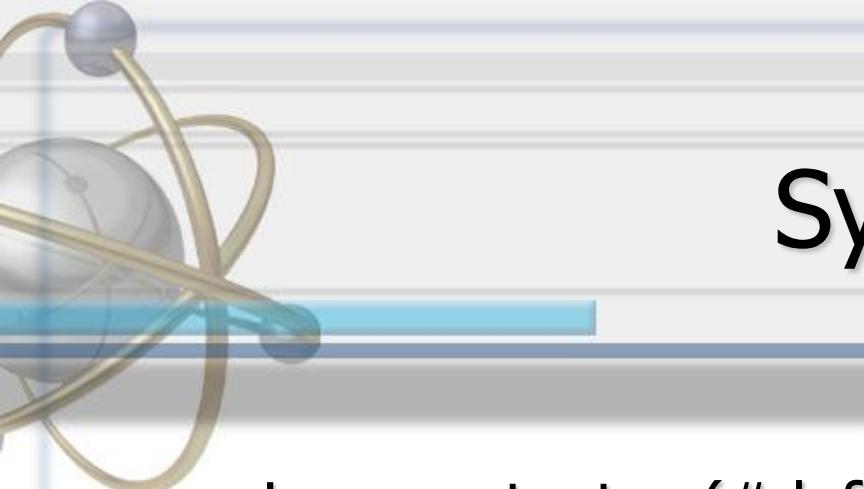
# Syntaxe

EX:

```
GLfloat R,G,B;  
R = 0.0; G = 0.0; B = 1.0;  
glColor3f(R,G,B);
```

Ou encore

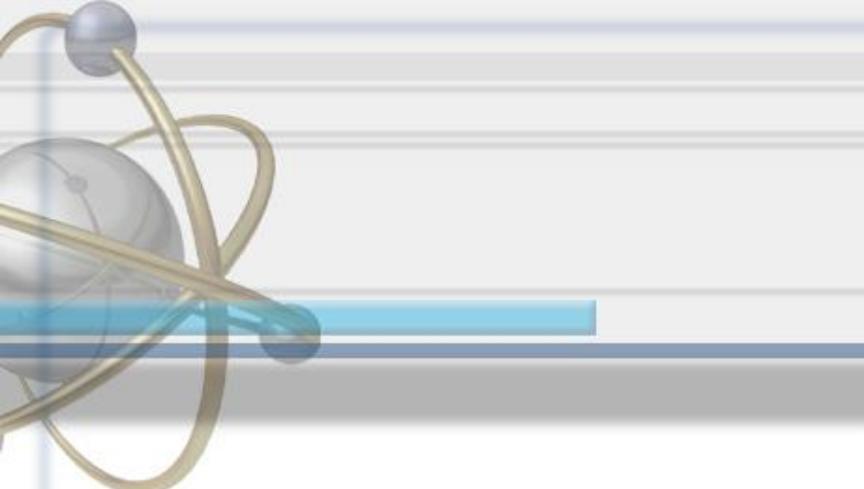
```
GLfloat blue[] = {0.0, 0.0, 1.0};  
glColor3fv(blue);
```



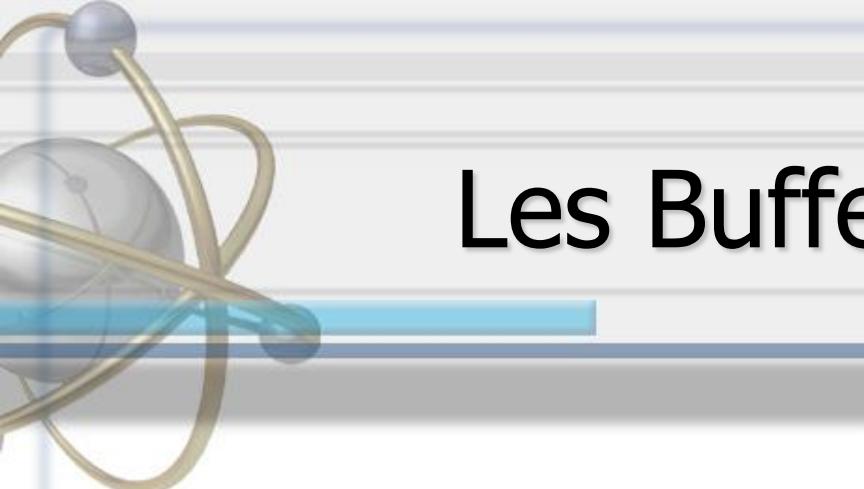
# Syntaxe

- Les constantes (`#define` ou `enum`) sont données en majuscule et commencent par le préfixe `GL_` :
  - `GL_COLOR_BUFFER_BIT` : `0x00004000`
  - `GL_NO_ERROR` : `0x0`
  - `GL_TEXTURE_2D` : `0x0DE1`
  - etc.
- Ex :

```
glClearColor (1.0, 1.0, 1.0, 1.0);
glClear(GL_COLOR_BUFFER_BIT);
```



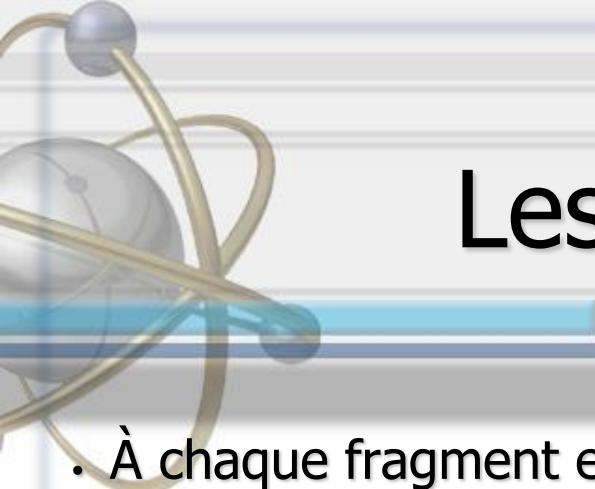
# Les Buffers d'OpenGL



# Les Buffers d'OpenGL

## Framebuffer

- Le framebuffer est une zone de la mémoire vidéo dans laquelle l'image est écrite (sous forme d'un bitmap) avant d'être envoyée vers le moniteur

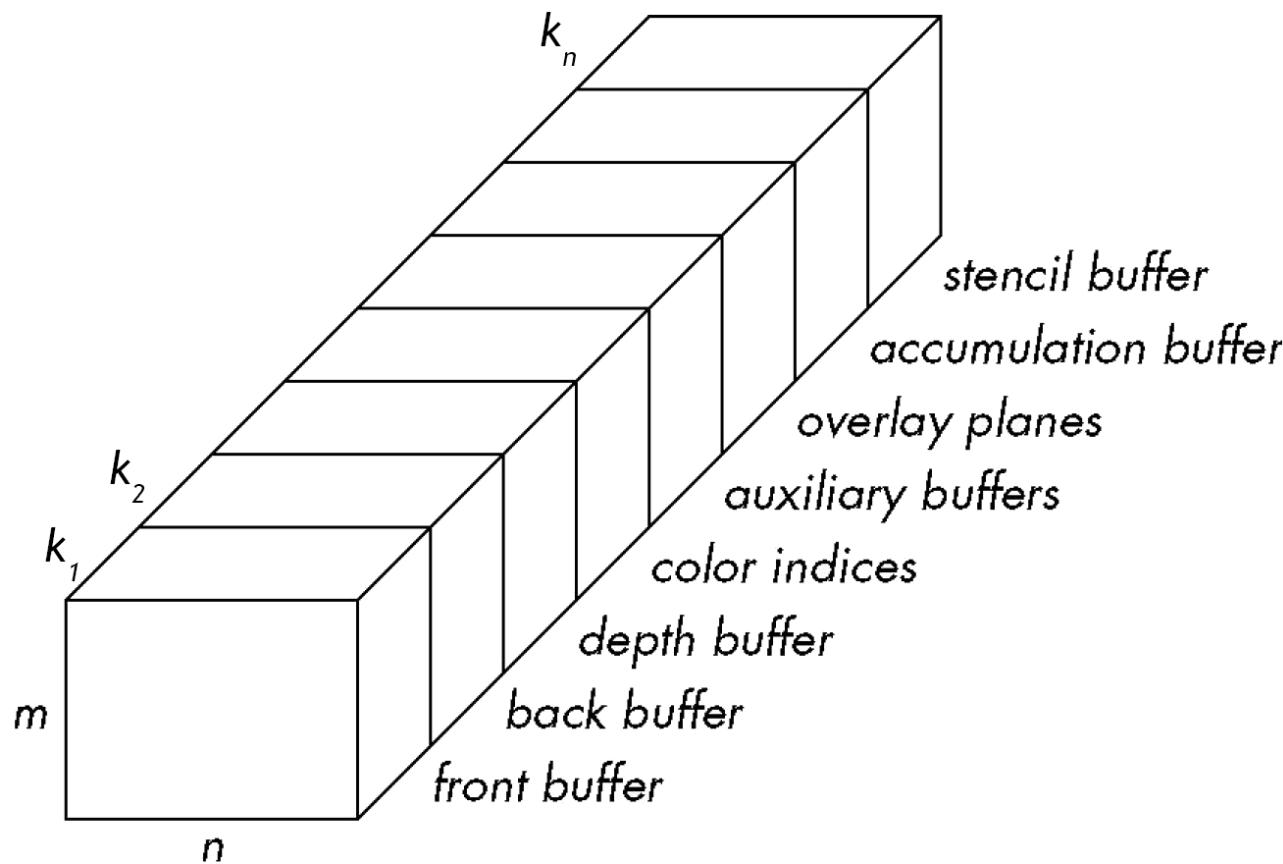


# Les Buffers d'OpenGL

- À chaque fragment est associé des coordonnées qui correspondent à un pixel, une couleur, profondeur, valeur alpha, etc.
- Ces informations sont interpolées à partir des valeurs aux sommets
- Le stockage de ces diverses informations est faite dans des buffers :
  - color buffer
  - depth buffer
  - stencil buffer
  - accumulation buffer
  - ...
- Ces buffers servent aux support d'opérations spéciales avant que les pixels soient finalement écrits dans le color-buffer visible

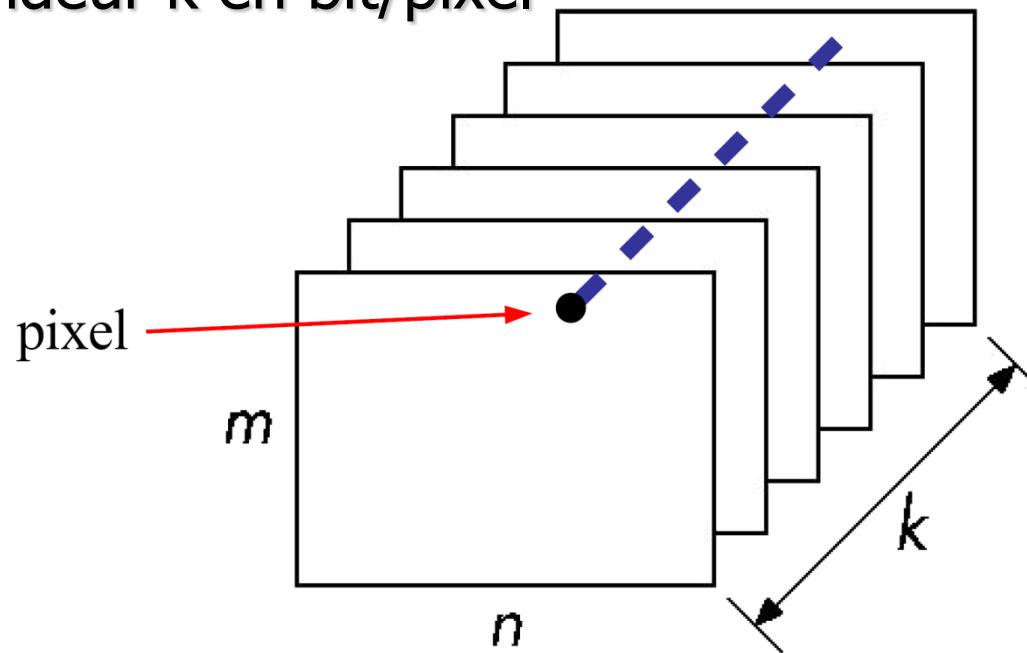
# Les Buffers d'OpenGL

Frame buffer = color buffer + depth buffer + stencil buffer + ...



# Les Buffers d'OpenGL

- Chaque buffer est défini par sa résolution spatiale ( $n \times m$ ) et sa profondeur  $k$  en bit/pixel



- La quantité de mémoire vidéo limite la résolution maximale et le nombre de couleurs des images

# Les Buffers d'OpenGL

X	Y	bpp	nbCoul	Mémoire	Carte
320	200	1	2	7,8Ko	8Ko
320	200	8	256	64Ko	64Ko
640	480	4	16	150Ko	256Ko
800	600	4	16	235Ko	512Ko
640	480	8	256	300Ko	512Ko
1024	768	4	16	384Ko	512Ko
800	600	8	256	468,7Ko	512Ko
640	480	16	65536	600Ko	1Mo
1280	1024	4	16	640Ko	1Mo
1024	768	8	256	768Ko	1Mo
640	480	24	16M	900Ko	1Mo
800	600	16	65536	937Ko	1Mo
1280	1024	8	256	1,25Mo	2Mo
800	600	24	16M	1,3Mo	2Mo
1024	768	16	65536	1,5Mo	2Mo
1024	768	24	16M	2,25Mo	3Mo

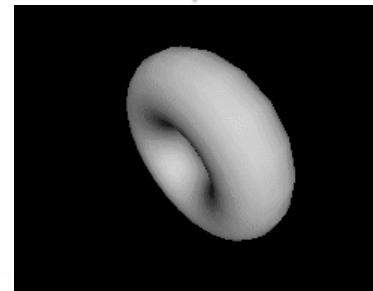
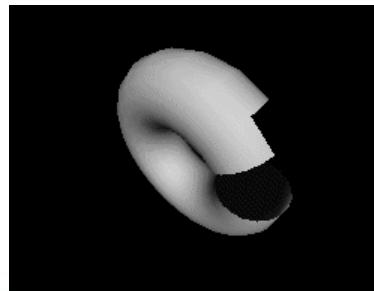
Les capacités sont à doubler si on emploie la technique dite du double-buffer.



# Les Buffers d'OpenGL

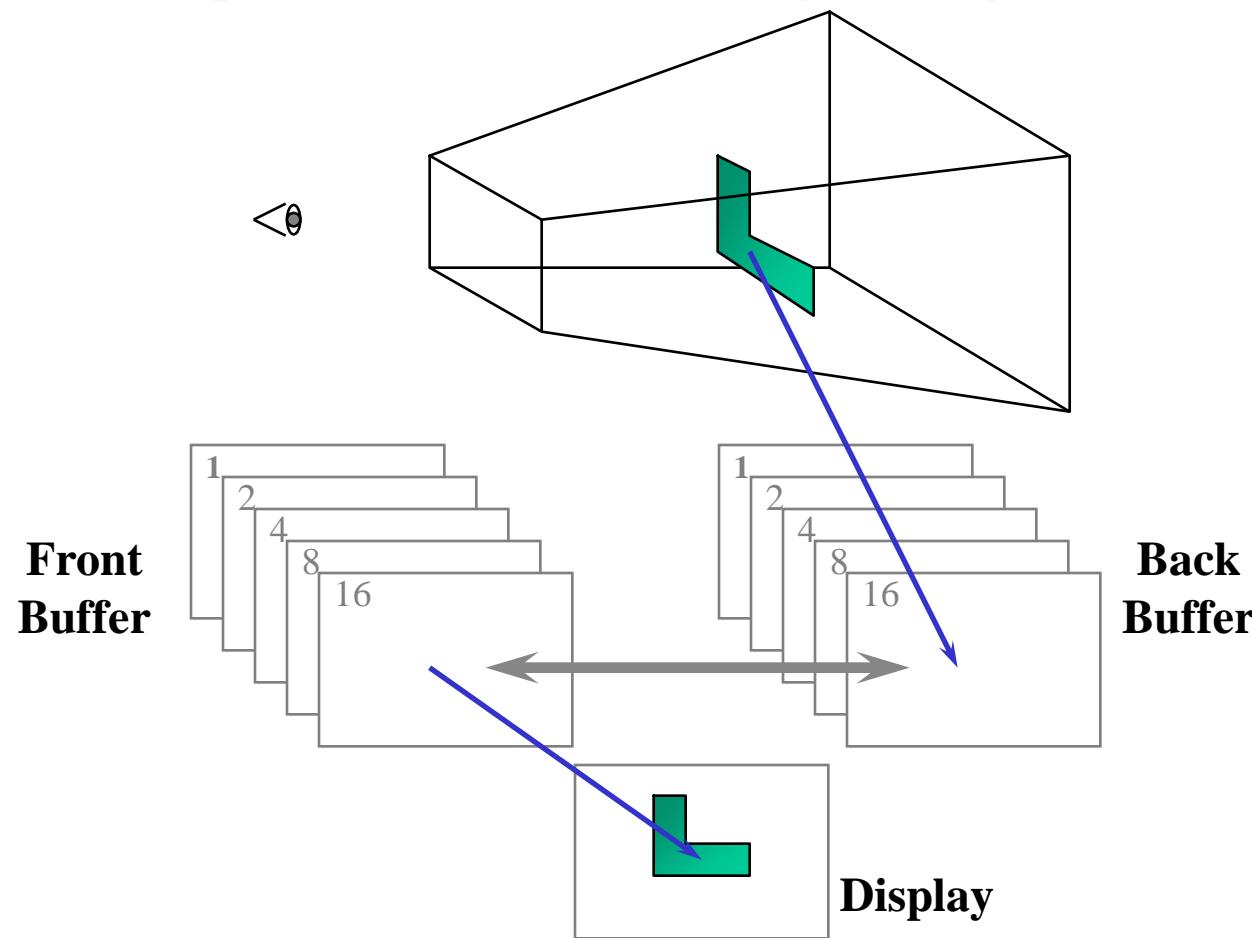
## Color buffers

- Il y en a plusieurs
  - front-left, front-right, back-left, back-right et quelque buffers auxiliaires
  - les buffers left, right et auxiliaires sont optionnels
- Ce sont les buffers dans lesquels on dessine
  - ils contiennent l'information de couleur pour chaque pixel
  - soit sous la forme d'un triplet (RGB)
  - soit d'un quadruplet (RGBA). A pour Alpha : l'opacité
- En openGL on a moyen de jouer avec 2 Color buffers, celui qui est actuellement afficher, pendant qu'on calcule dans le suivant ce qui sera affichée à l'image suivante.



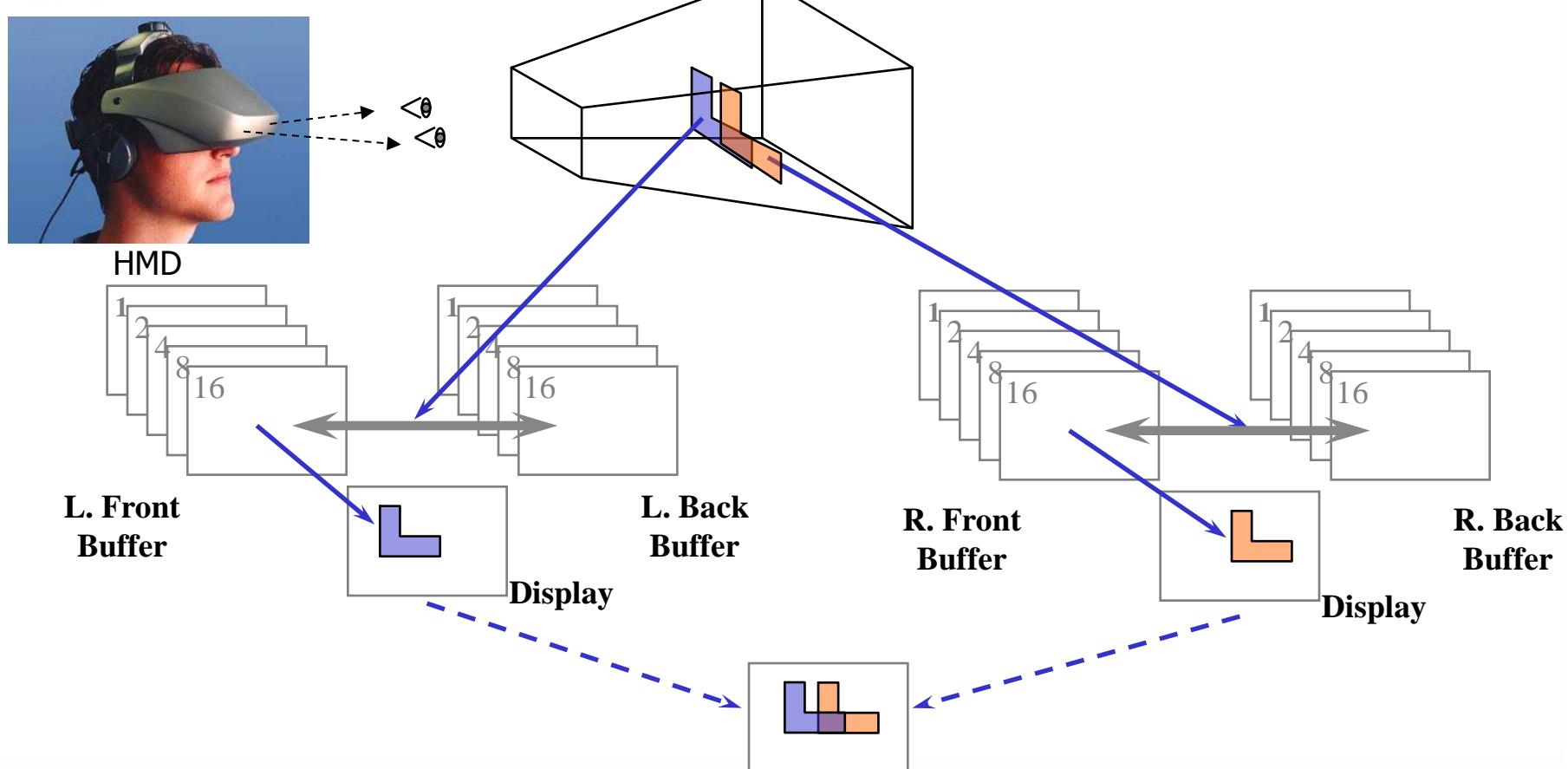
# Les Buffers d'OpenGL

- Le double-buffering nécessite un front (visible) et un back buffer



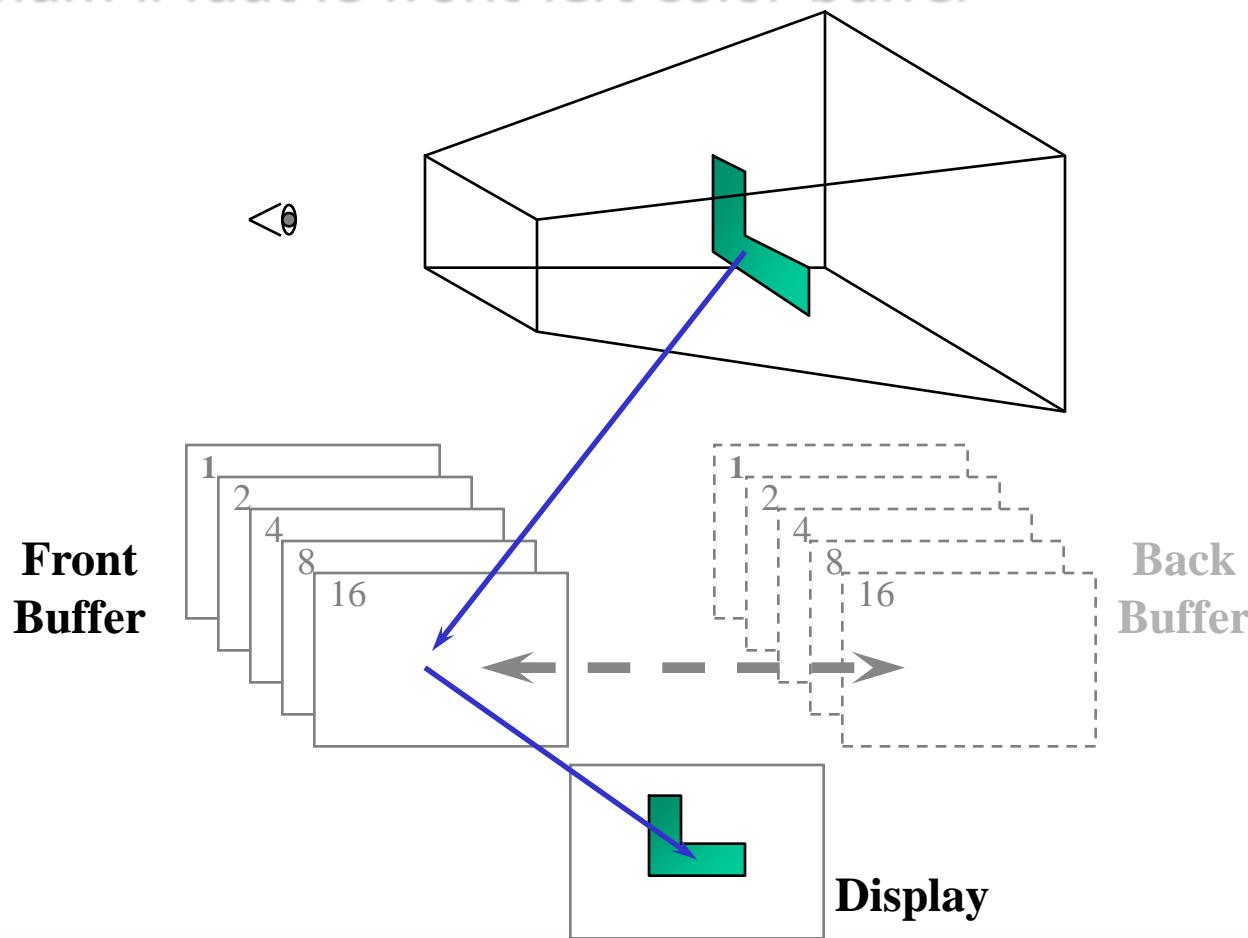
# Les Buffers d'OpenGL

- Pour l'affichage stéréoscopique on a besoin de front-right, front-left, back-right, back-left buffers pour les images droites et gauches



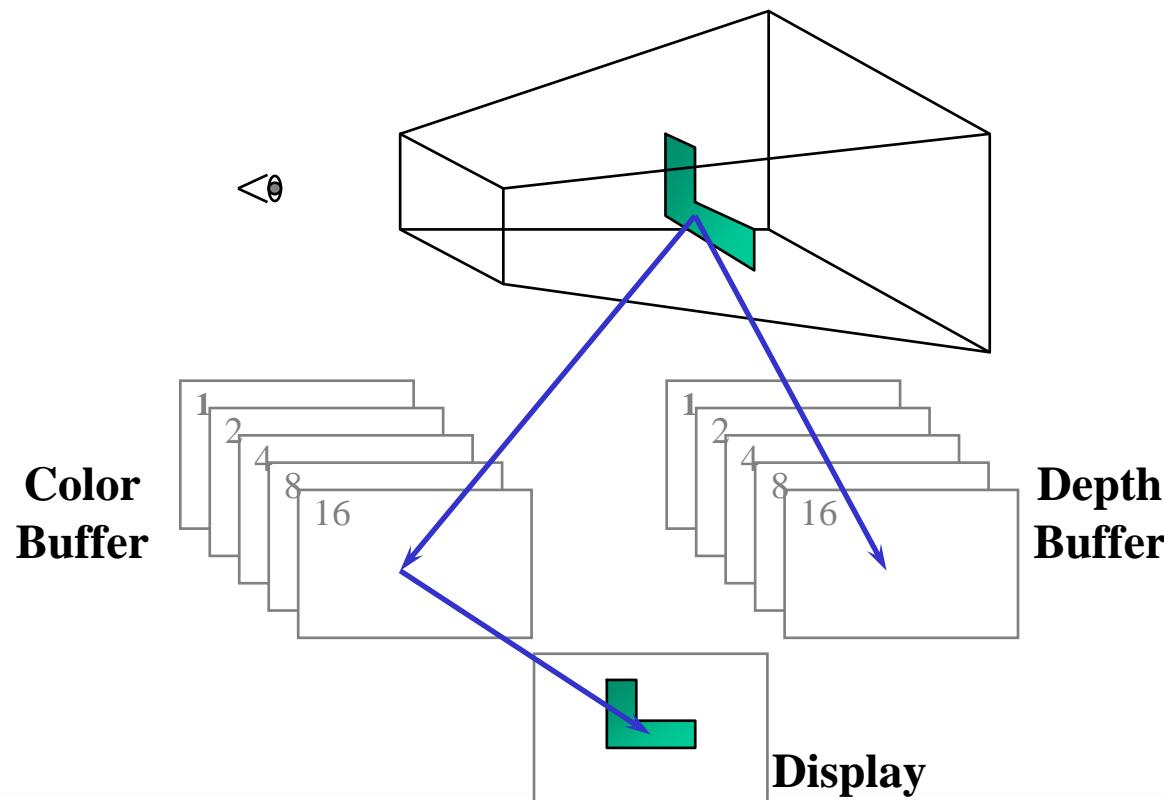
# Les Buffers d'OpenGL

- Au minimum il faut le front-left color buffer



# Les Buffers d'OpenGL

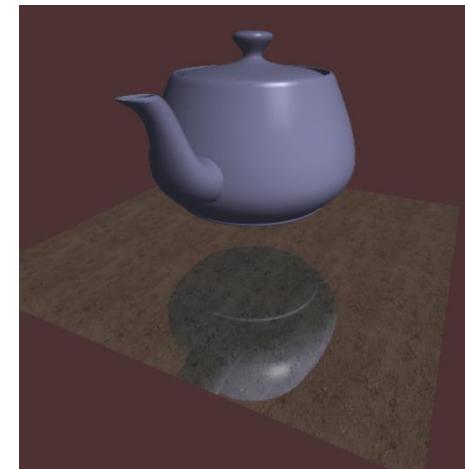
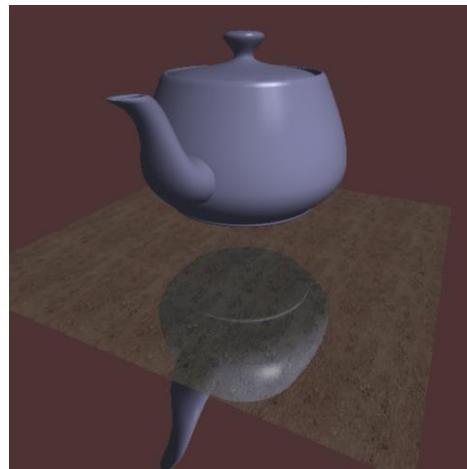
- Depth buffer (ou Z-buffer)
  - stocke une valeur de profondeur pour chaque pixel
  - utiliser pour la suppression des parties cachées





# Les Buffers d'OpenGL

- Stencil buffer
  - Permet de faire du clipping non rectangulaire en utilisant un buffer additionnel
  - Stocke des informations pour restreindre le dessin à certaines parties de l'écran (masquage de régions)
  - Application : ombres simples et reflets sur les miroirs.



- Mettre les pixels correspondants à la surface de réflexion à 1 dans le stencil buffer
- Copier dans le Color buffer uniquement les pixels pour lesquels le stencil buffer est égale à 1



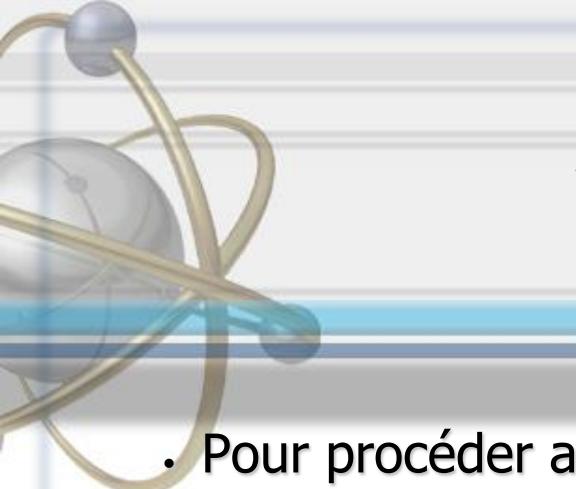
# Vider les Buffers

- Selon la taille du tampon, le vider peut constituer une opérations lourde en temps d'exécution.
- Certaines implémentations d'OpenGL permettent de vider plusieurs tampons en même temps.
- Pour en bénéficier, il faut :
  1. Spécifier les valeurs à écrire dans chaque tampon à vider
  2. Effectuer le vidage



# Vider les Buffers

- Pour définir les valeurs de vidage des tampons, utiliser les commandes suivantes :
  - `void glClearColor( GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha );`
  - `void glClearIndex( GLfloat index );`
  - `void glClearDepth( GLclampd depth );`
  - `void glClearStencil( GLint s);`
  - `void glClearAccum( GLfloat red, GLfloat green, GLfloat blue, GLfloat alpha );`
- Les paramètres correspondent aux valeurs de vidage

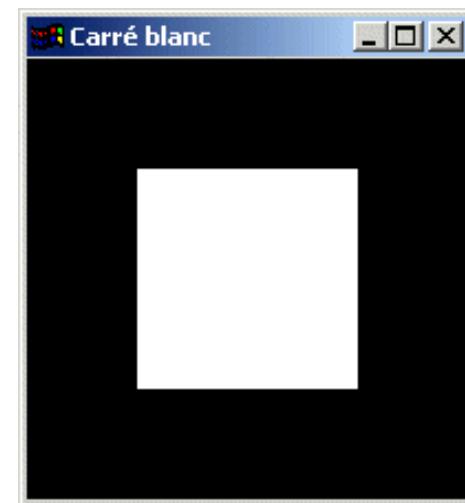


# Vider les Buffers

- Pour procéder au vidage, utiliser :  
`glClear(GLbitfield mask)`
- avec comme valeur de « mask », un OU logique parmi :
  - `GL_COLOR_BUFFER_BIT`
  - `GL_DEPTH_BUFFER_BIT`
  - `GL_STENCIL_BUFFER_BIT`
  - `GL_ACCUM_BUFFER_BIT`
- Lors du vidage du tampon chromatique, tous les tampons chromatiques activés en écriture sont vidés.

# Vider les Buffers

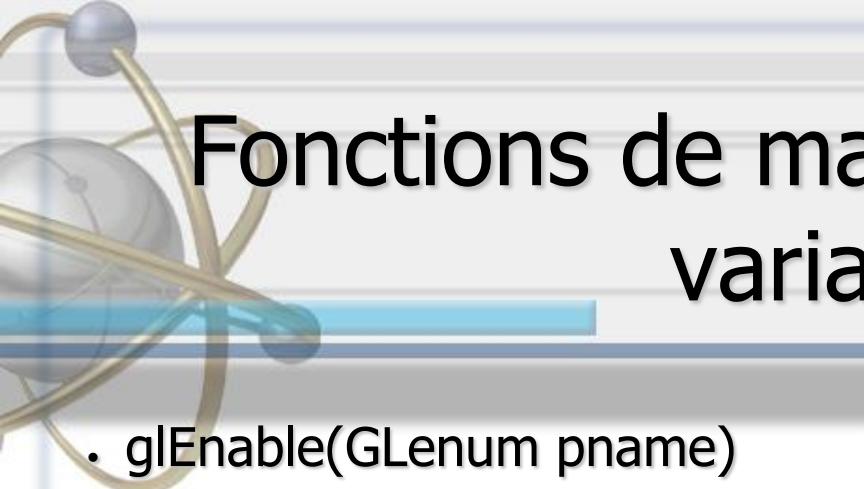
```
void display(void) {  
    // Initialiser le buffer de couleur avec la couleur noir  
    glClearColor(0.0F,0.0F,0.0F,0.0F);  
    glClear(GL_COLOR_BUFFER_BIT);  
  
    glColor3f(1.0F,1.0F,1.0F);  
  
    // Dessin du carre  
    glBegin(GL_POLYGON);  
    glVertex2f(-0.5F,-0.5F);  
    glVertex2f(-0.5F,0.5F);  
    glVertex2f(0.5F,0.5F);  
    glVertex2f(0.5F,-0.5F);  
    glEnd();  
  
    glFlush();  
}
```





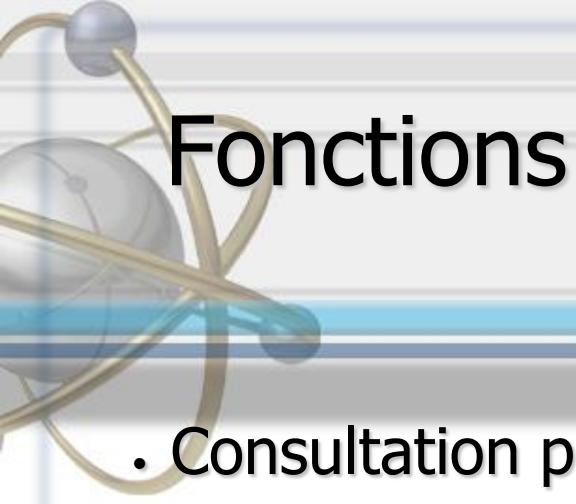
# Variables d'état

- OpenGL est une machine d'états : state-machine.
- OpenGL possède un ensemble de valeurs (ou états) qui définissent un ensemble de comportements.
  - Ex : la couleur (que ce soit la couleur de 'fond' que la couleur de la primitive dessinée), la gestion de l'éclairage (activée ou désactivée, et combien de lumière sont activés), la gestion de la transparence (quelles fonctions pour calculer la transparence), les transformations de projection (perspective ou cavalière), la matrice active (projection, modélisation-visualisation ou texture),
  - Tous ces états peuvent être changés à n'importe quel moment.
- Des instructions existent pour placer le système dans certains états.
  - Ceux-ci resteront actifs (utilisés à cette valeur) jusqu'à être changés (couleurs de tracé, matrices de transformation, activation du de l'élimination des parties cachées, ...).
- Les variables d'état possède des valeurs par défaut
  - Ex : la couleur de vidage du buffer de couleur est par défaut le noir (`glClearColor(0.0, 0.0, 0.0, 0.0)`) ou encore la lumière est par défaut désactivé (`glDisable(GL_LIGHTING)`).
- Chaque fois que le dessin d'un objet est demandé, l'ensemble de ces variables définit la manière avec laquelle l'objet est dessiné.
- L'ensemble de ces variables d'états constitue l'environnement OpenGL.



# Fonctions de manipulation directe des variables d'état

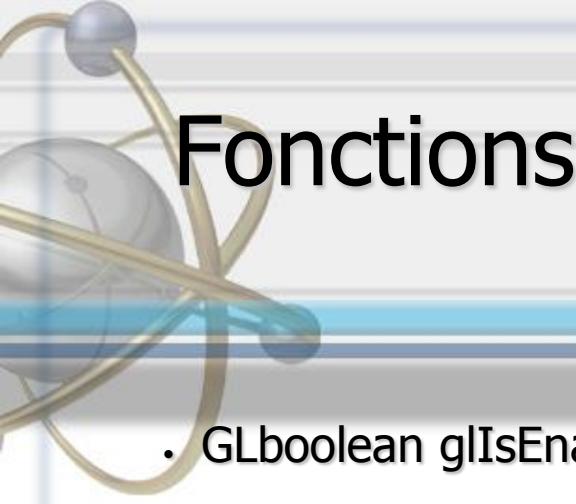
- glEnable(GLenum pname)
  - Activation d'une variable d'état booléenne
  - pname: variable d'état à activer
  - Exemples :
    - glEnable(GL\_LIGHTING) -> activation de la gestion des lumières et des matériaux
    - glEnable(GL\_DEPTH\_TEST) -> activation de l'élimination des parties cachées
- glDisable(GLenum pname)
  - Inhibition d'une variable d'état booléenne
  - pname: variable d'état à désactiver



# Fonctions de manipulation directe des variables d'état

- Consultation partielle des variables d'états

- `glGetBooleanv(GLenum pname,GLboolean *param)`
- `glGetIntegerv(GLenum pname,GLinteger *param)`
- `glGetFloatv(GLenum pname,GLfloat *param)`
- `glGetDoublev(GLenum pname,GLdouble *param)`
  - `pname`: variable d'état consultée
  - `param`: résultat (pointeur si une seule valeur, tableau si plusieurs valeurs)



# Fonctions de manipulation directe des variables d'état

- `GLboolean glIsEnabled(GLenum pname)`
  - Consultation de variables d'états booléennes
  - `pname`: variable d'état consultée
- `glPushAttrib(GLbitfield mask)`
  - Sauvegarde de variables d'état par empilement dans une pile
  - `mask`: variables d'état à sauvegarder
- `glPopAttrib()`
  - Restauration de variables d'état par dépilement



# Pile d'Attributs (stack attribute)

## Mask

- GL\_ACCUM\_BUFFER\_BIT
- GL\_ALL\_ATTRIB\_BITS
- GL\_COLOR\_BUFFER\_BIT
- GL\_CURRENT\_BIT
- GL\_DEPTH\_BUFFER\_BIT
- GL\_ENABLE\_BIT
- GL\_EVAL\_BIT
- GL\_FOG\_BIT
- GL\_HINT\_BIT
- GL\_LIGHTING\_BIT
- GL\_LINE\_BIT
- GL\_LIST\_BIT
- GL\_PIXEL\_MODE\_BIT
- GL\_POINT\_BIT
- GL\_POLYGON\_BIT
- GL\_POLYGON\_STIPPLE\_BIT

## Attribute Group

- accum-buffer
- color-buffer
- current
- depth-buffer
- enable
- eval
- fog
- hint
- lighting
- line
- list
- pixel
- point
- polygon
- polygon-stipple



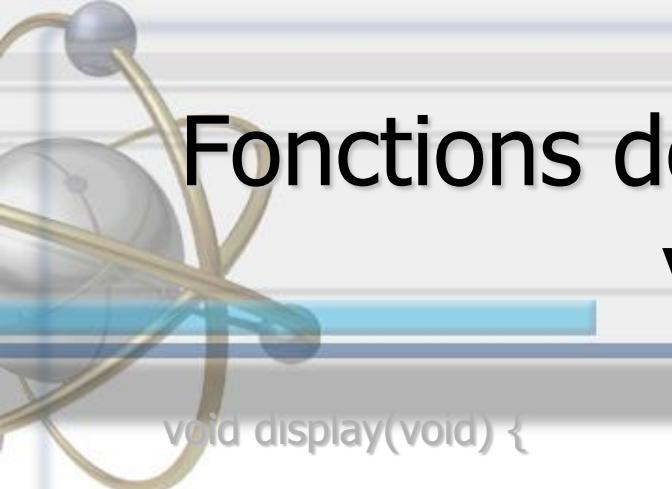
# Fonctions de manipulation directe des variables d'état

```
void initGL(void)
{
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);

    glLightModelfv(GL_AMBIENT, LAmbientValue);
    glLightfv(GL_LIGHT0, GL_POSITION, Lposition1);
    glLightfv(GL_LIGHT0, GL_AMBIENT, LAmbientValue);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, LDiffuseValue);
    glLightfv(GL_LIGHT0, GL_SPECULAR, LSpecularValue);

    glShadeModel(GL_SMOOTH);
    glDepthFunc(GL_LEQUAL);

    glEnable(GL_CULL_FACE);
}
```



# Fonctions de manipulation directe des variables d'état

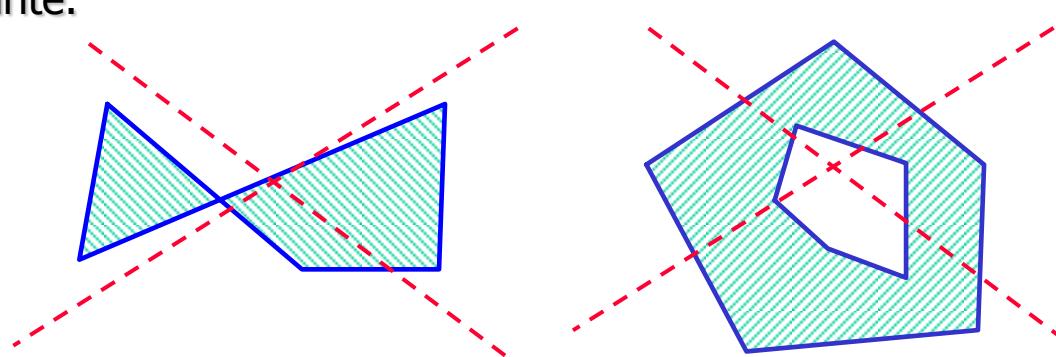
```
void display(void) {  
  
    glClearColor(0.0F,0.0F,0.0F,0.0F) ;  
    glClear(GL_COLOR_BUFFER_BIT) ;  
  
    glColor3f(1.0F,0.0F,0.0F) ;  
    // Primitives 1  
        // Sommet 1  
        // Sommet 2  
        // ...  
    // Primitives 2  
    ...  
    // Primitives 3  
    ...  
  
    glColor3f(0.0F,0.0F,1.0F) ;  
    // Primitives 4  
    ...  
    glFlush();  
}
```

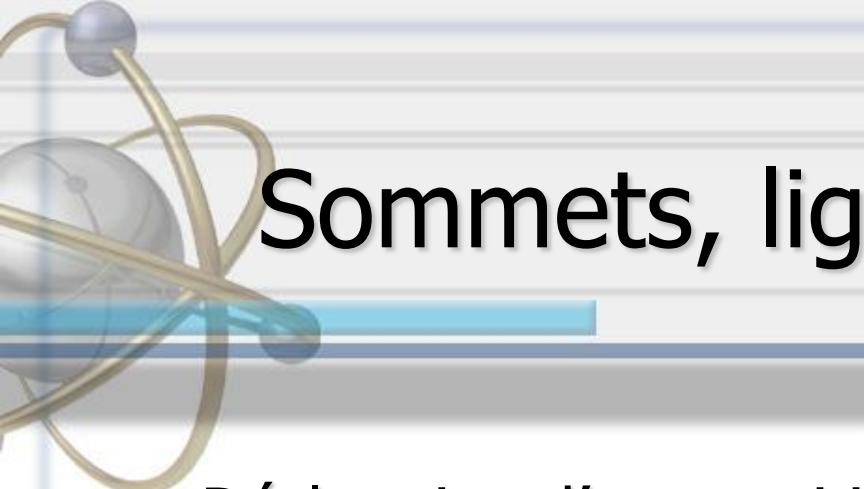


# Instructions de base d'OpenGL

# Sommets, lignes et polygones

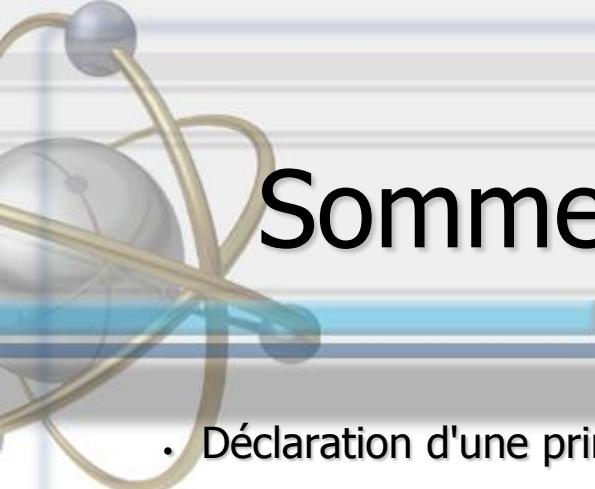
- Sommet : Ensemble de trois coordonnées représentant une position dans l'espace
  - OpenGL travaille en coordonnées homogènes.
- Ligne : Segment rectiligne entre deux sommets
- Polygone : Surface délimitée par une boucle fermée de lignes
  - Par définition, un polygone OpenGL ne se recoupe pas et n'a pas de trou.
  - Si on veut représenter un polygone ne vérifiant pas ces conditions, on devra le subdiviser 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 planarité.





# Sommets, lignes et polygones

- 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.



# Sommets, lignes et polygones

- Déclaration d'une primitive graphique
  - Une primitive graphique est constituée d'un ensemble de sommets.
  - 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`

et

```
void glEnd(void) ;
```

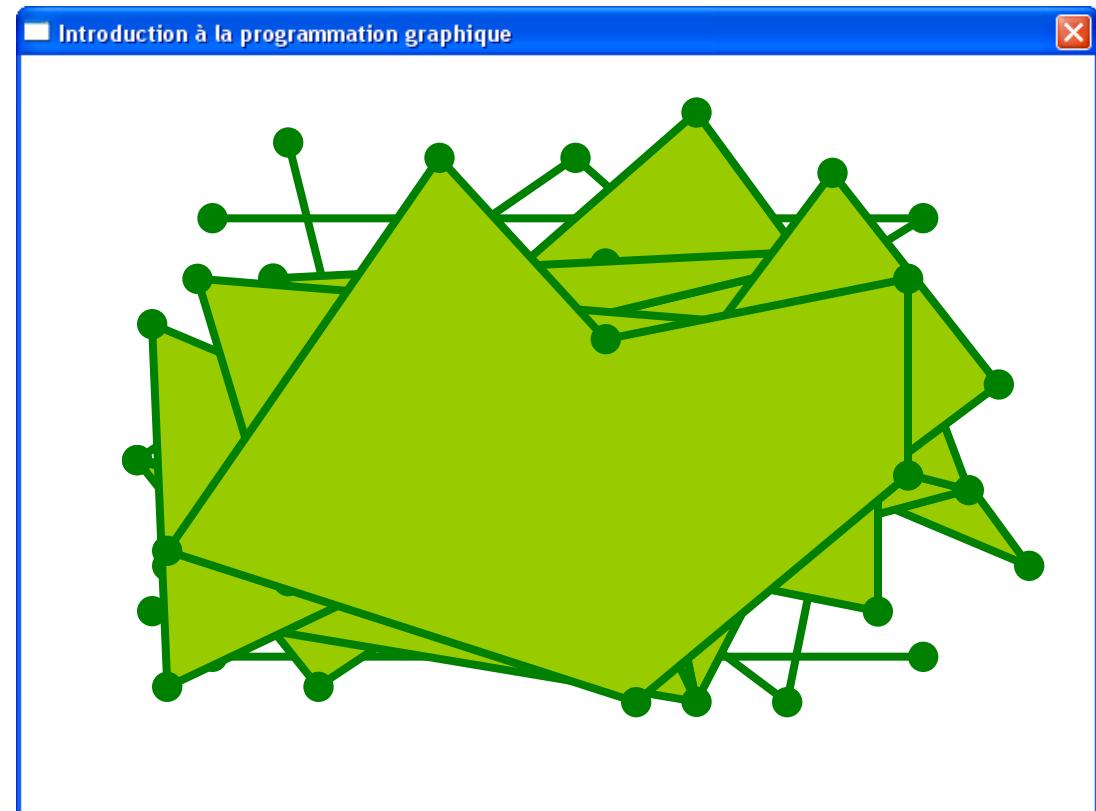
# Primitives géométriques

- **glBegin (GLenum mode)**

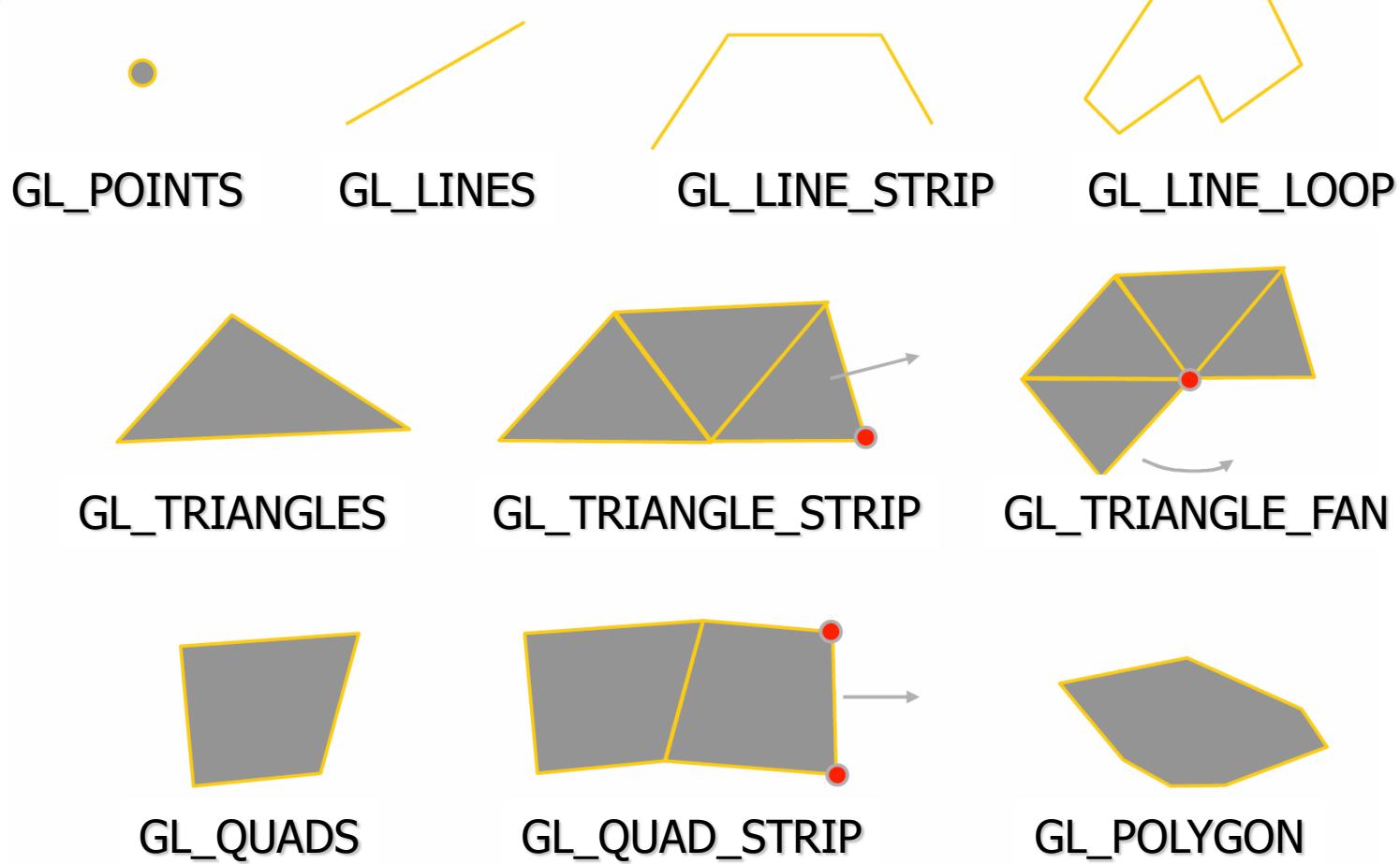
Annonce le début d'une série de sommets 3D.

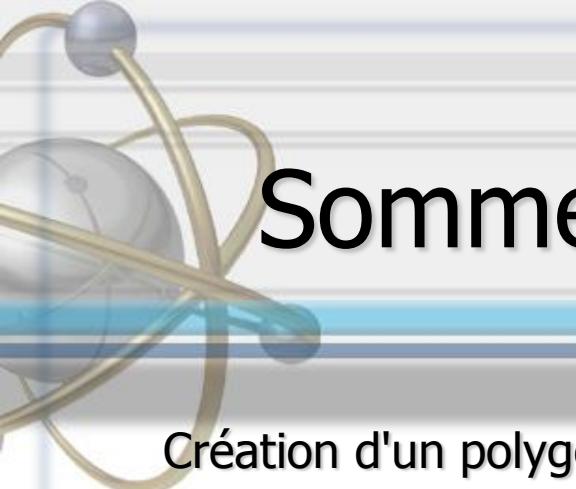
La nature des primitives générées varie en fonction du **mode** choisi :

- **.GL\_POINTS**
- **.GL\_LINES**
- **.GL\_LINE\_STRIP**
- **.GL\_LINE\_LOOP**
- **.GL\_TRIANGLES**
- **.GL\_TRIANGLE\_STRIP**
- **.GL\_TRIANGLE\_FAN**
- **.GL\_QUADS**
- **.GL\_QUAD\_STRIP**
- **.GL\_POLYGON**
- **glEnd ()**



# Type de primitive graphique

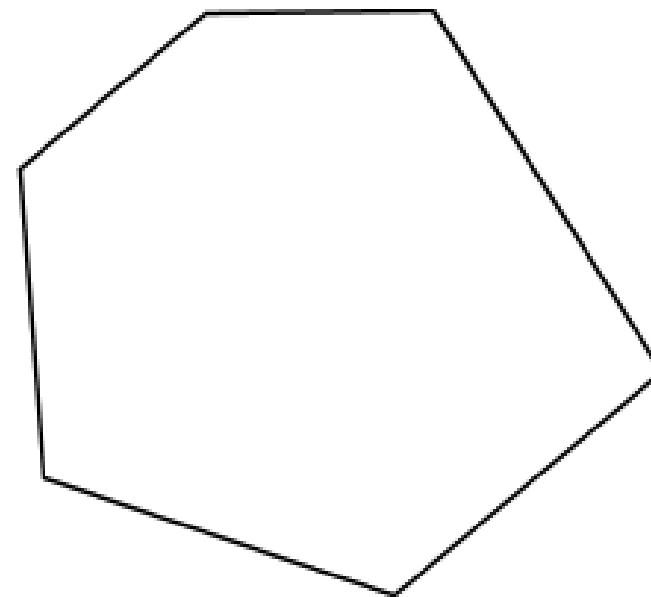




# Sommets, lignes et polygones

Création d'un polygone

```
glBegin(GL_POLYGON);
    glVertex2f(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() ;
```

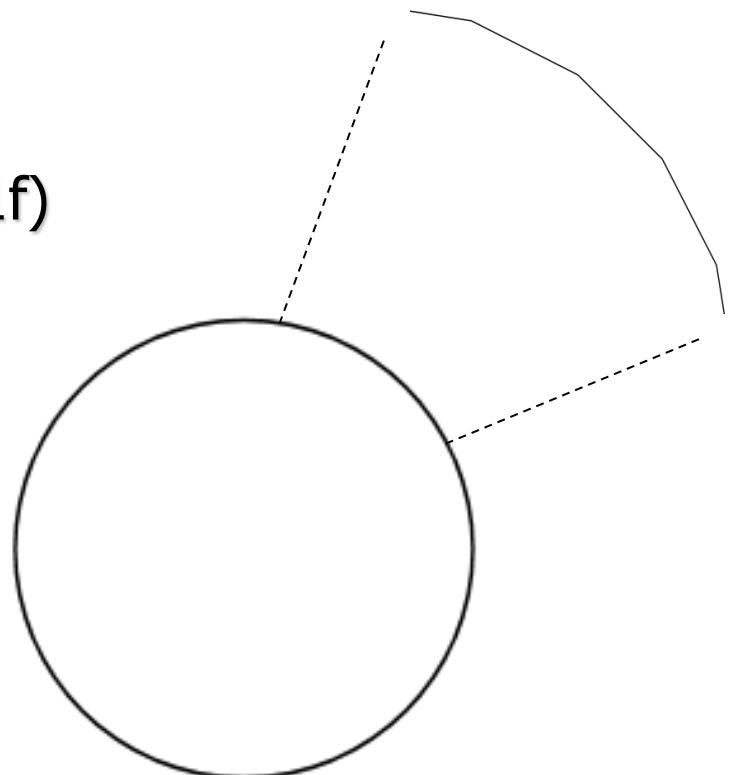


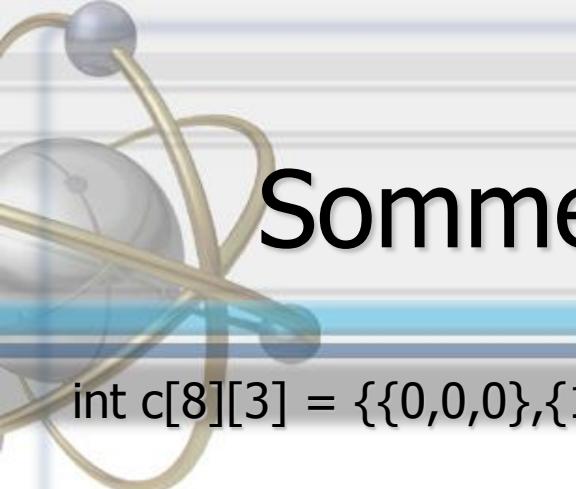


# Sommets, lignes et polygones

- Un cercle :

```
glBegin(GL_LINE_LOOP);
for(float a=0 ; a<2*PI ; a+=0.1f)
    glVertex2f( cosf(a), sinf(a) );
glEnd();
```



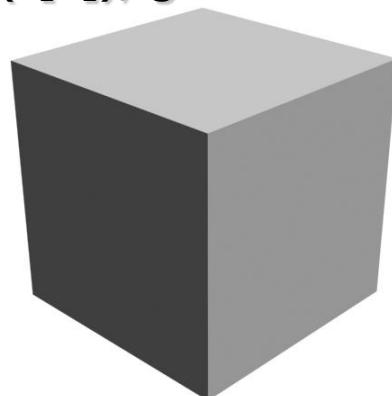


# Sommets, lignes et polygones

```
int c[8][3] = {{0,0,0},{1,0,0},{1,1,0},{0,1,0},{0,0,0},{1,0,0},{1,1,0},{0,1,0}};
```

```
glBegin(GL_QUAD_STRIP);
    glVertex3iv(c[0]); glVertex3iv(c[4]); glVertex3iv(c[1]); glVertex3iv(c[5]);
    glVertex3iv(c[2]); glVertex3iv(c[6]); glVertex3iv(c[3]); glVertex3iv(c[7]);
    glVertex3iv(c[0]); glVertex3iv(c[4]);
glEnd();
```

```
glBegin(GL_QUADS);
    glVertex3iv(c[0]); glVertex3iv(c[1]); glVertex3iv(c[2]); glVertex3iv(c[3]);
    glVertex3iv(c[7]); glVertex3iv(c[6]); glVertex3iv(c[5]); glVertex3iv(c[4]);
glEnd();
```

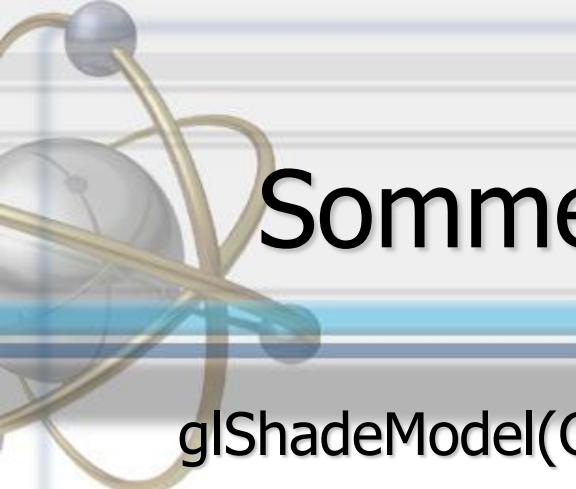




# Sommets, lignes et polygones

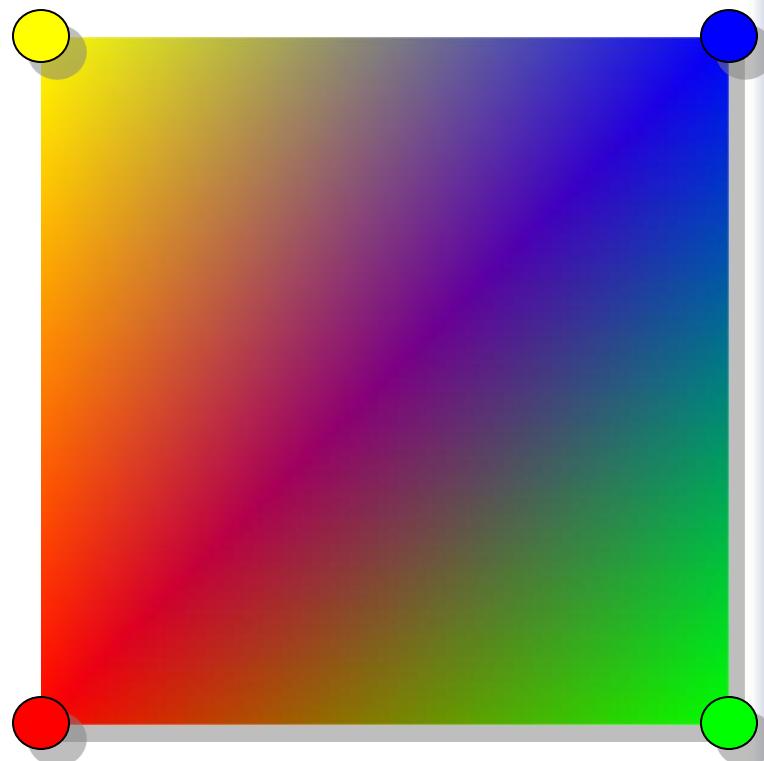
- La définition d'une primitive peut faire appel à d'autres informations que ses seuls points. Celles-ci sont aussi renseignées via des appels à des fonctions particulières.

glVertex*()	Coordonnées du sommet
glColor*()	Couleur associée au sommet
glIndex*()	Indexe de la couleur
glNormal*()	Vecteur normal au sommet
glEvalCoord*()	Génération de coordonnées
glCallList()	Coordonnées textures
glTexCoord*()	Textura koordináták
glEdgeFlag*()	Contrôle du tracé d'un coté
glMaterial*()	Matériaux associés au sommet



# Sommets, lignes et polygones

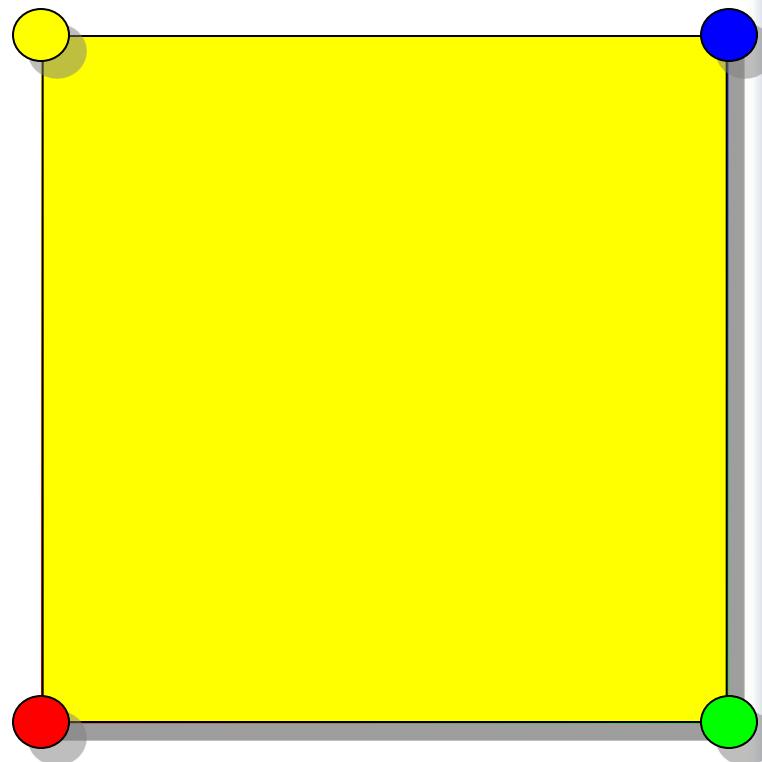
```
glShadeModel(GL_SMOOTH);
glBegin(GL_QUADS);
    glColor3f (1.0,0.0,0.0);
    glVertex3f(0.0,0.0,0.0);
    glColor3f (0.0,1.0,0.0);
    glVertex3f(1.0,0.0,0.0);
    glColor3f (0.0,0.0,1.0);
    glVertex3f(1.0,1.0,0.0);
    glColor3f (1.0,1.0,0.0);
    glVertex3f(0.0,1.0,0.0);
glEnd();
```

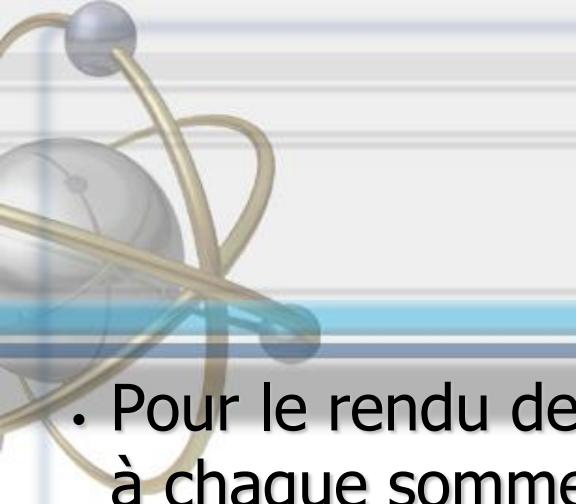




# Sommets, lignes et polygones

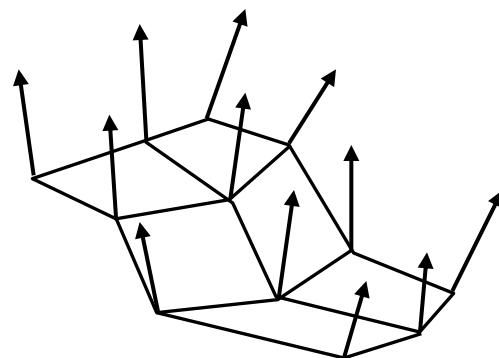
```
glShadeModel(GL_FLAT);
glBegin(GL_QUADS);
    glColor3f (1.0,0.0,0.0);
    glVertex3f(0.0,0.0,0.0);
    glColor3f (0.0,1.0,0.0);
    glVertex3f(1.0,0.0,0.0);
    glColor3f (0.0,0.0,1.0);
    glVertex3f(1.0,1.0,0.0);
    glColor3f (1.0,1.0,0.0);
    glVertex3f(0.0,1.0,0.0);
glEnd();
```



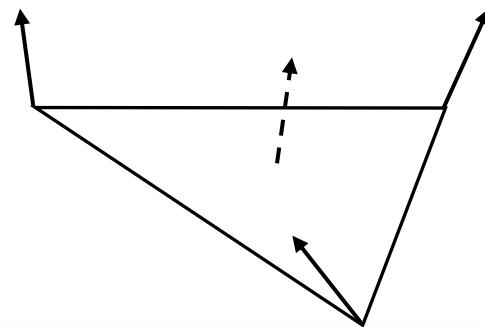


# Normales

- Pour le rendu des surfaces, une direction normale est associée à chaque sommet.



- La normale est propre aux sommets et pas aux polygones

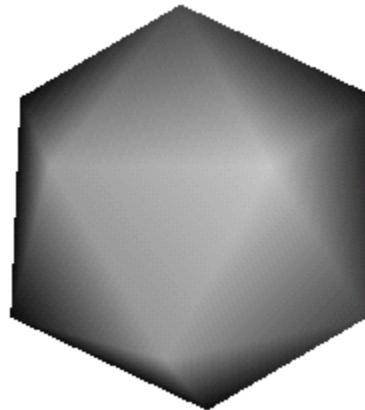


# Déclaration d'un ensemble de facettes

```
#define X .525731112119133606  
#define Z .850650808352039932
```

```
static GLfloat vdata[12][3] = {  
    {-X,0,Z},{X,0,Z}, {-X,0,-Z},  
    {X,0,-Z},{0,Z,X},{0,Z,-X},  
    {0,-Z,X},{0,-Z,-X},{Z,X,0},  
    {-Z,X,0},{Z,-X,0},{-Z,-X,0} };
```

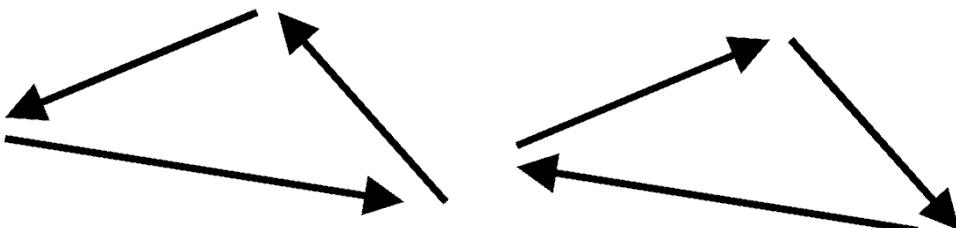
```
static GLint t[20][3] = {  
    {0,4,1},{0,9,4},{9,5,4},{4,5,8},  
    {4,8,1},{8,10,1},{8,3,10},{5,3,8},  
    {5,2,3},{2,7,3},{7,10,3},{7,6,10},  
    {7,11,6},{9,2,5},{0,1,6},{6,1,10},  
    {9,0,11},{9,11,2},{11,0,6},  
    {7,2,11}};
```



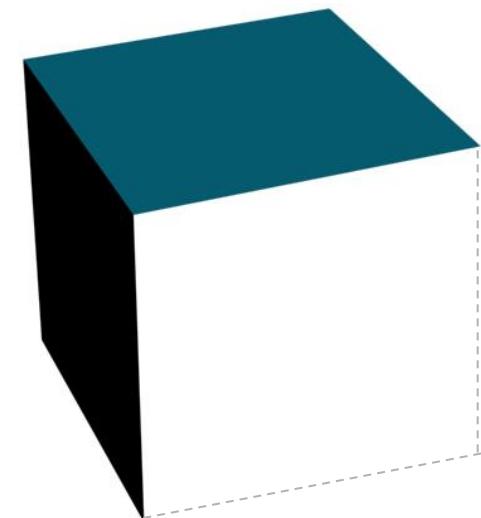
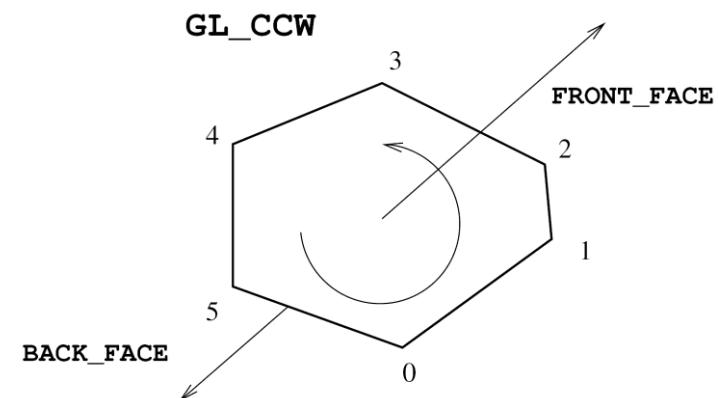
```
for ( i = 0 ; i < 20 ; i++ ) {  
  
    glBegin(GL_TRIANGLES) ;  
  
    glNormal3fv(&vdata[t[i][0]][0]);  
    glVertex3fv(&vdata[t[i][0]][0]);  
  
    glNormal3fv(&vdata[t[i][1]][0]);  
    glVertex3fv(&vdata[t[i][1]][0]);  
  
    glNormal3fv(&vdata[t[i][2]][0]);  
    glVertex3fv(&vdata[t[i][2]][0]);  
  
    glEnd() ; }
```

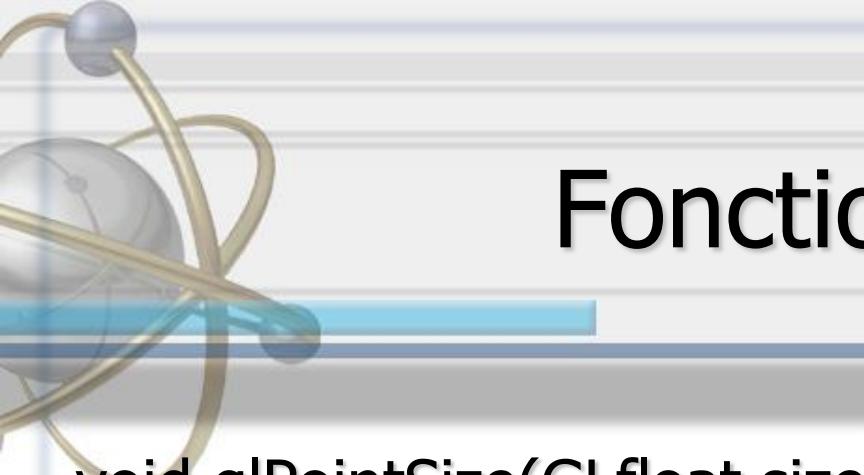
# Sommets, lignes et polygones

- void glFrontFace (GLenum mode);
  - Défini l'orientation des polygones.
  - mode : GL\_CCW (default), GL\_CW



- void glCullFace (GLenum mode);
  - Défini le culling.
  - mode : GL\_BACK, GL\_FRONT, GL\_FRONT\_AND\_BACK
- void glEnable / glDisable (GL\_CULL\_FACE);
  - Demande à OpenGL de (dés)activer le culling : élimination les faces.





# Fonction diverses

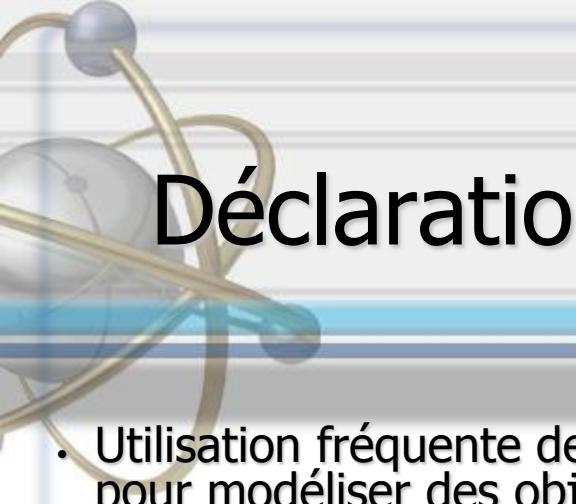
- `.void glPointSize(GLfloat size) ;`

- Configure la taille de tracé (par défaut 1) des sommets.

- `.void glLineWidth(GLfloat width) ;`

- Configure la largeur de tracé (par défaut 1) des segments de ligne.

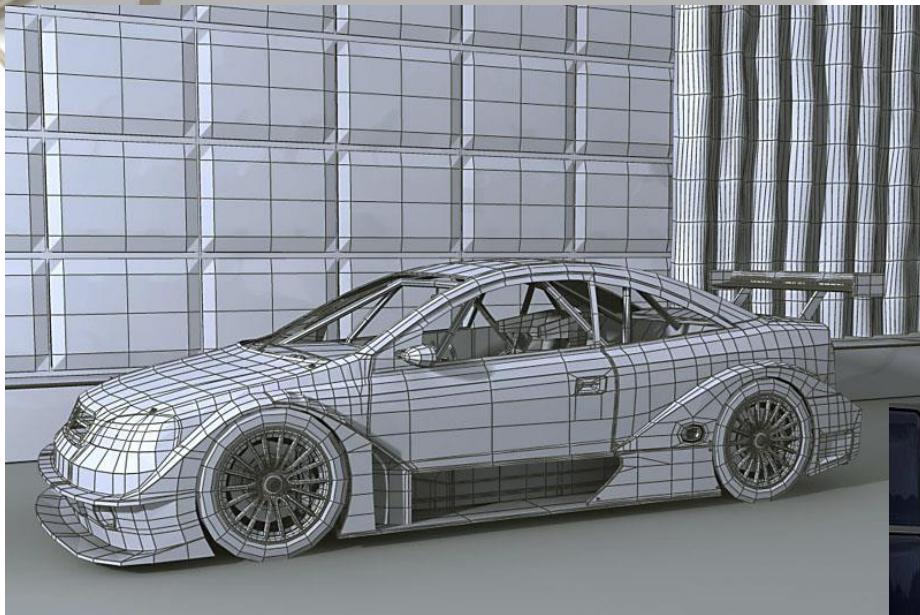


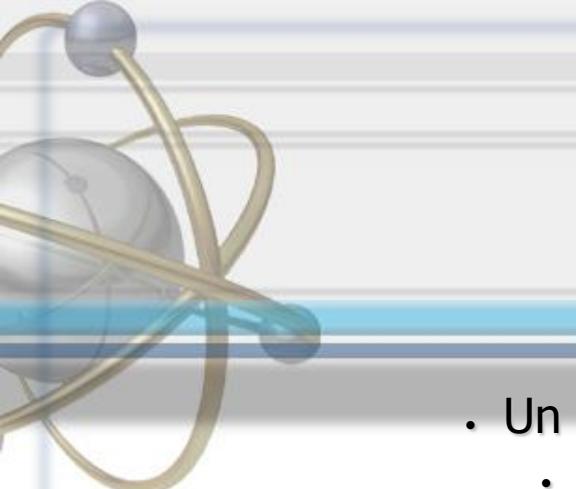


# Déclaration d'un ensemble de facettes

- Utilisation fréquente de surfaces polygonales (ensembles de facettes adjacentes) pour modéliser des objets complexes.
- Recommandations
  - Conserver constante l'orientation des polygones
  - Eviter les facettes non triangulaires
  - Trouver un bon équilibre entre le nombre de polygones et la vitesse et la qualité d'affichage
  - Multiplier le nombre de polygones sur la silhouette des objets
  - Éviter des intersections en T

# Modèles complexes





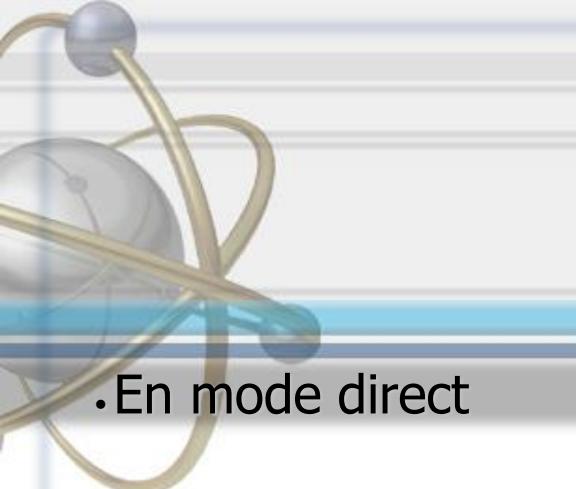
# Mode direct

- Un maillage c'est:
  - Une liste de sommets attribués
  - Une liste de faces (triangulaires)
  - un triangle = indices des trois sommets

```
typedef struct sommet_s
{
    GLfloat position [3];
    GLfloat normale [3];
    GLubyte couleur [4];
    GLfloat texcoord [2];
}sommet_t;
sommet_t sommets[nbrSommets];
```

- Une liste de faces (triangulaires)
  - un triangle = indices des trois sommets

```
GLuint faces[3][nbrFaces];
```

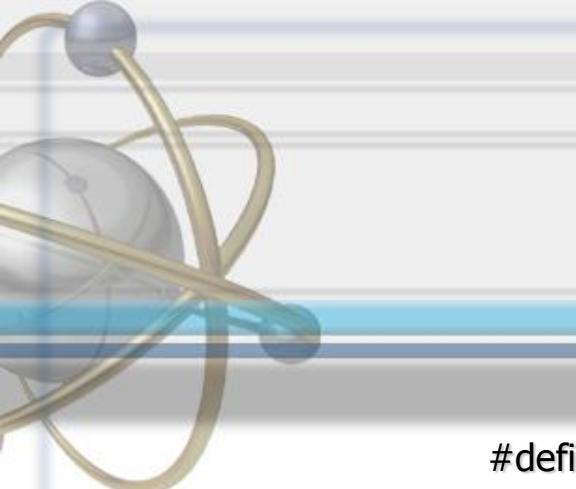


# Mode direct

- En mode direct

```
glBegin(GL_TRIANGLES);
for(int i=0; i<nbrFaces; ++i)
    for(int j=0; j<3; ++j)
    {
        glNormal3fv (sommets[faces[j][i]].normale);
        glColor3ubv (sommets[faces[j][i]].couleur);
        glTexCoord2fv(sommets[faces[j][i]].texcoord);
        glVertex3fv (sommets[faces[j][i]].position);
    }
 glEnd;
```

- En pratique le mode direct n'est jamais utilisé (et va être supprimé)!
- préférer les vertex array



# Vertex array

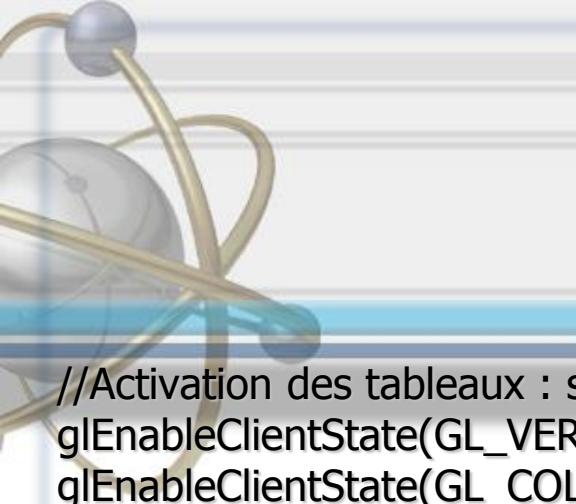
```
#define Num 12500 // Nombre de sommets

//Création des tableaux : sommets, couleurs, coordonnées de texture

static GLint sommets[]={
25,56,65,
...
87,98, 25};

static GLfloat couleurs[]={
1.0, 0.1,0.5,
...
,0.4,0.4,0.8};

static GLfloat texture[]={
0.1,0.8,
...
0.5,0.9
}
```



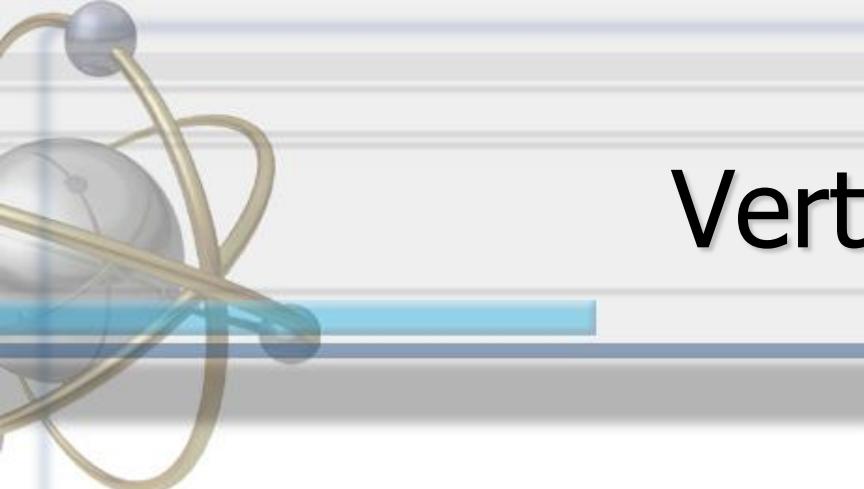
# Vertex array

```
//Activation des tableaux : sommets, couleurs, coordonnées de texture
glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_COLOR_ARRAY);
glEnableClientState(GL_TEXTURE_COORD_ARRAY);

//Déclarer à OpenGL les tableaux où sont stockés les sommets, couleurs, coordonnées de texture
glVertexPointer(3, GL_INT, 0, sommets);
glColorPointer(3, GL_FLOAT, 0, couleurs);
glTexCoordPointer(2, GL_FLOAT, 0, texture);

//Construit une séquence de primitives géométriques contenant les éléments des tableaux activés
glDrawArrays(GL_TRIANGLES, 0, Num);

//Désactivation des tableaux
glDisableClientState(GL_VERTEX_ARRAY);
glDisableClientState(GL_COLOR_ARRAY);
glDisableClientState(GL_TEXTURE_COORD_ARRAY);
```



# Vertex array

- `glEnableClientState(GLenum mode)`  
Active un tableau parmi les 6 possibles :
  - `GL_VERTEX_ARRAY`
  - `GL_NORMAL_ARRAY`
  - `GL_COLOR_ARRAY / GL_INDEX_ARRAY`
  - `GL_TEXTURE_COORD_ARRAY`
  - `GL_EDGE_FLAG_ARRAY`
- `glDisableClientState(GLenum mode)`  
Désactive le tableau indiqué par « mode »



# Vertex array

- `glVertexPointer (size, type, stride, ptr)`

size : nombre de coordonnées

type : types de données

stride : nbre d'octets entre le début de deux données

ptr : pointeur sur le tableau de données

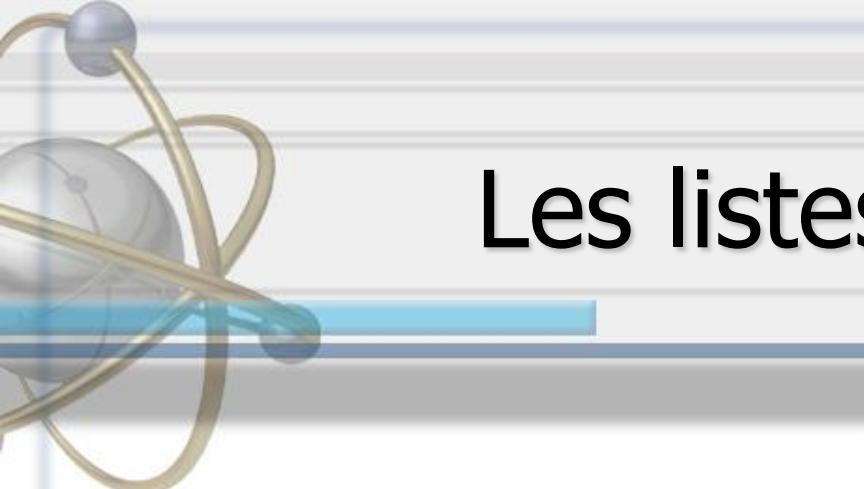
Déclare le tableau où sont stockés les sommets :

- `glNormalPointer`
- `glColorPointer`
- `glIndexPointer`
- `glTexCoordPointer`
- `glEdgeFlagPointer`

- `glDrawArrays(GLenum mode, GLint premier, GLsizei nombre)`

construit une séquence de primitives géométriques contenant les éléments des tableaux activés, de « premier » à « premier + nombre – 1 ».

Le type de primitive géométrique est indiqué par mode de la même manière que dans `glBegin()`



# Les listes d'affichage

- Une liste d'affichage est une suite de commandes OpenGL stockées pour une utilisation future.
- Les commandes d'une liste d'affichage sont les mêmes que les commandes d'affichage immédiat (mode direct).
- Quand une liste d'affichage est invoquée, les commandes qu'elle contient sont exécutées dans l'ordre où elles ont été stockées.
- Les commandes sont non seulement compilées dans la liste d'affichage, mais peuvent aussi être exécutées immédiatement pour obtenir un affichage
- Les listes d'affichages sont des macros et, à ce titre, ne peuvent pas être modifiées ou paramétrées à part en la détruisant et en la redéfinissant.

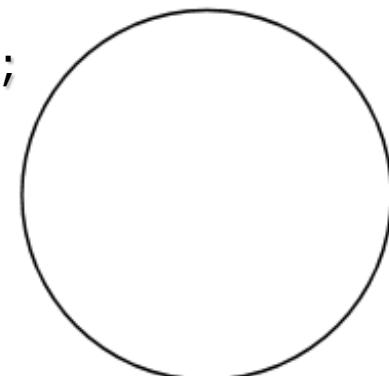


# Les listes d'affichage

```
void cercle() {  
    GLint i ;  
    GLfloat cosinus,sinus ;  
    glBegin(GL_POLYGON) ;  
    for ( i = 0 ; i < 100 ; i++ ) {  
        cosinus = cos(i*2*PI/100.0);  
        sinus = sin(i*2*PI/100.0);  
        glVertex2f(cosinus,sinus) ; }  
    glEnd() ;  
}
```

```
#define CERCLE 1
```

```
void cercle() {  
    GLint i ;  
    GLfloat cosinus,sinus ;  
    glNewList(CERCLE,GL_COMPILE) ;  
    glBegin(GL_POLYGON) ;  
    for ( i = 0 ; i < 100 ; i++ ) {  
        cosinus = cos(i*2*PI/100.0);  
        sinus = sin(i*2*PI/100.0);  
        glVertex2f(cosinus,sinus) ; }  
    glEnd() ;  
    glEndList() ;  
  
    glCallList(CERCLE) ;
```





# Les listes d'affichage

.glNewList(GLuint list, GLenum mode);

.Début d'une liste

- . list : index de la liste.
  - . variable de type entier non-signé qui contient l'indice de la liste précédemment récupérer avec la commande glGenList(...).
- . mode : mode de compilation.
  - . GL\_COMPILE : la liste d'affichage est compilé et prêt à être utilisée. les instructions ne sont pas exécutées lors de l'enregistrement.
  - . GL\_COMPILE\_AND\_EXECUTE : les instructions que vous entrez dans la liste sont enregistrées et exécutées immédiatement.

.glEndList();

.Fin de la liste



# Les listes d'affichage

- **glGenLists (Glsizei range) ;**
  - générer une suite de nouveaux numéros de liste
- **glIsList (GLuint list) ;**
  - retourne GL\_TRUE si le numéro est déjà utilisé
- **glCallList(listName)**
  - Appel et exécute une liste d'affichage
- **glDeleteLists (GLuint list, GLsizei range) ;**
  - Efface une suite de listes



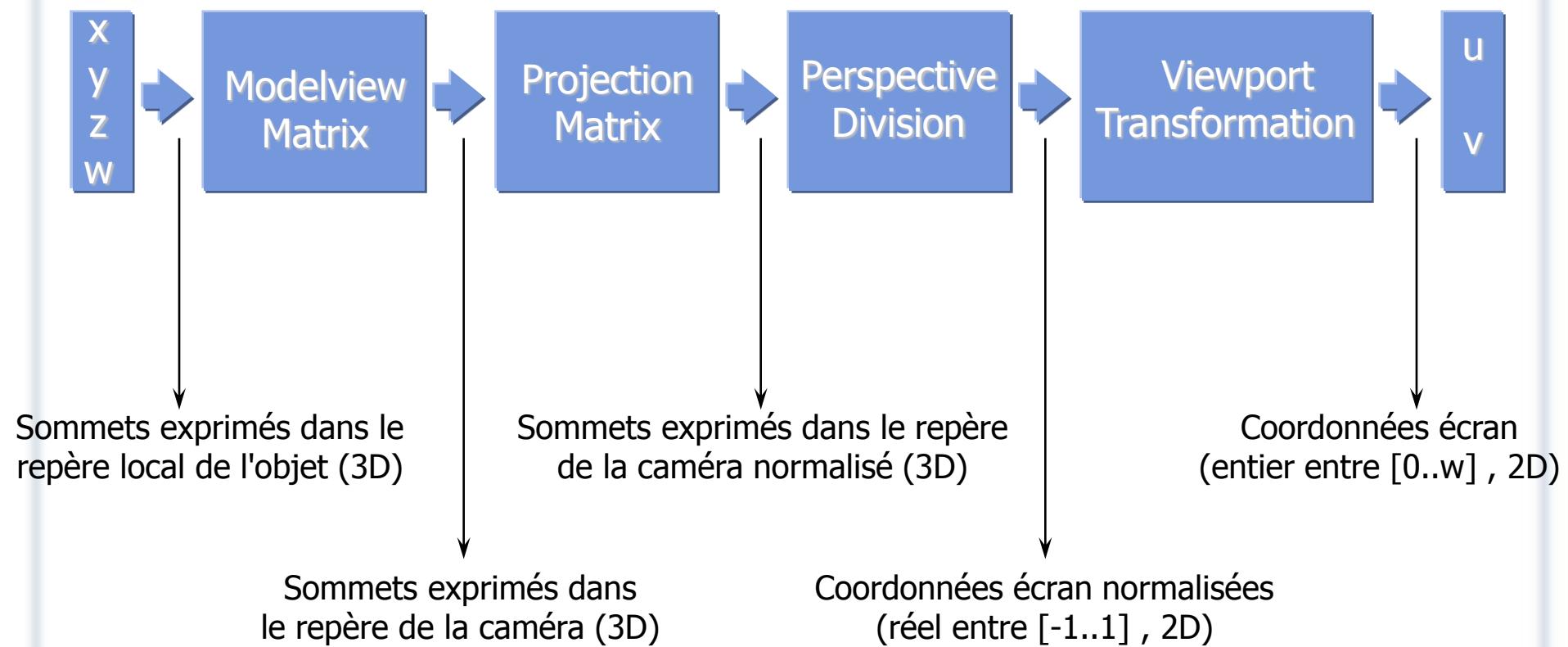
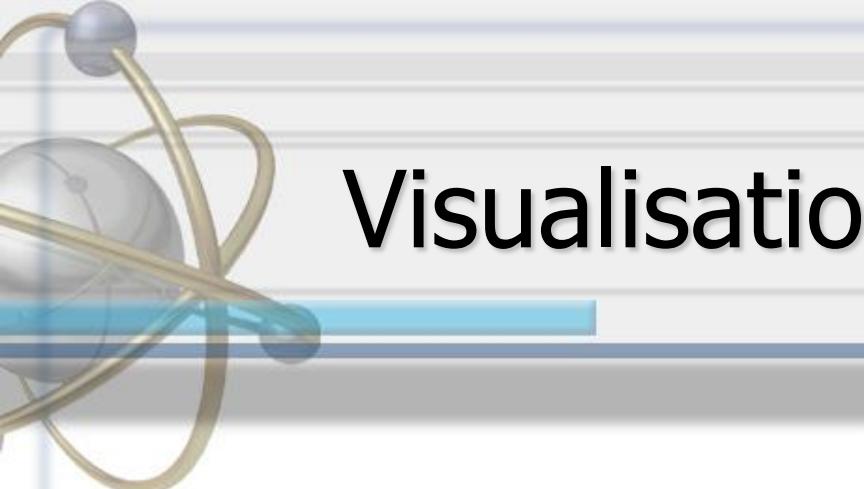
# Les listes d'affichage

- L'utilisation d'une liste d'affichage permet :
  - Une bonne organisation des données
  - une amélioration de la performance :
    - définition d'une géométrie qui a besoin d'être affichée à plusieurs reprises dans un programme
    - définition d'une série de transformations (déplacements, mise à l'échelle, etc...) qui ont besoin d'être appliquées à plusieurs reprises
    - source lumineuse, propriétés des matériaux et modèles d'illumination : définition de matériaux et de conditions d'illumination dans des listes d'affichage pour accélérer le rendu d'une scène
    - dans un modèle client-serveur les listes d'affichage sont gérées par le serveur d'où une réduction du coût de transmission des données à travers le réseau
    - une liste d'affichage peut être mémorisée dans une mémoire dédiée ou sous forme optimisée plus compatible avec le matériel graphique.
    - texture et patrons de remplissage: l'image de la texture est placée dans une mémoire dédiée alors la conversion de format est effectuée lors de la compilation de la liste d'affichage.
    - formats de données: le format spécifié n'est pas toujours compatible avec le matériel graphique. Lors de la compilation d'une liste d'affichage OpenGL transforme les données dans une représentation compatible avec le matériel graphique d'où un gain significatif dans la vitesse d'affichage.
- L'inconvénient principal des listes d'affichage est qu'elles peuvent occuper beaucoup de place en mémoire.



# Visualisation sous OpenGL

# Visualisation sous OpenGL



# Visualisation sous OpenGL

Transformations de modélisation

Illumination (Shading)

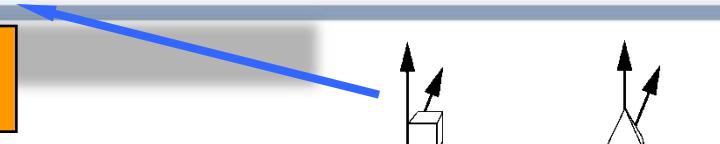
Transformations d'affichage

Clipping

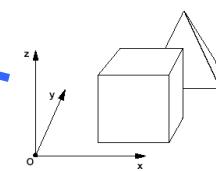
Transformation écran (Projection)

Pixelisation (Rasterization)

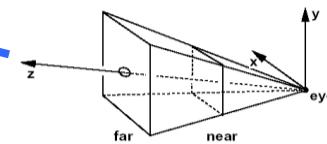
Visibilité / Affichage



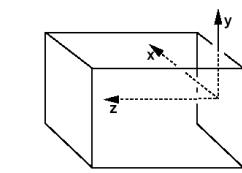
Repère objet



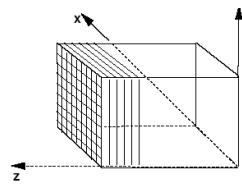
Repère scène



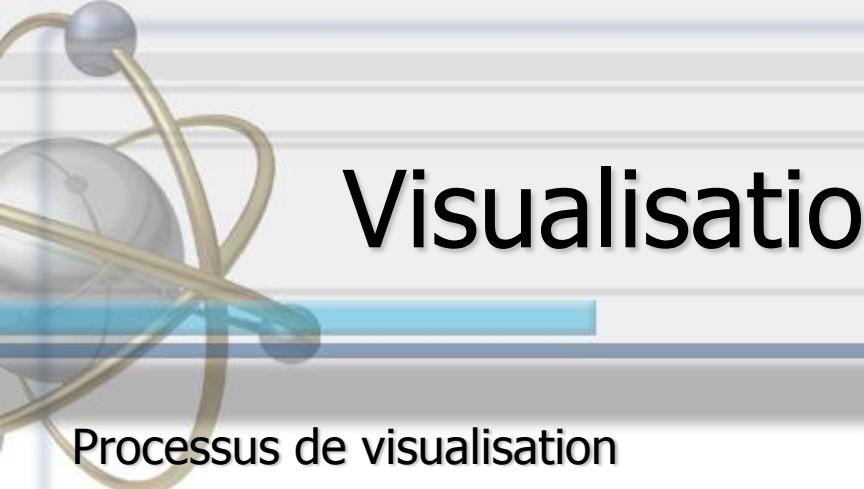
Repère caméra



Repère caméra normalisé (NDC)



Espace écran



# Visualisation sous OpenGL

## Processus de visualisation

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.



# Les matrices sous OpenGL

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Forme théorique

$$[a_{11} \quad a_{21} \quad a_{31} \quad 0 \quad a_{12} \quad \dots \quad 0 \quad a_{11} \quad a_{21} \quad a_{31} \quad 1]$$

Sous OpenGL



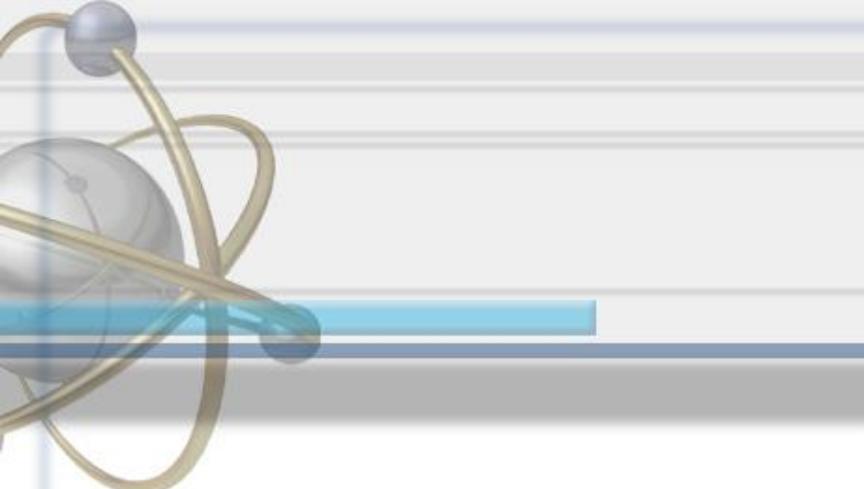
# Choix de la transformation de travail

- void glMatrixMode(GLenum mode);
  - mode : transformation sur laquelle les transformations géométriques à venir vont être composées de manière incrémentale :
    - GL\_MODELVIEW : Positionne et oriente les objets ou la caméra
    - GL\_PROJECTION : Projette les objets de la scène sur le plan image.
  - 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.



# Transformations d'utilité générale

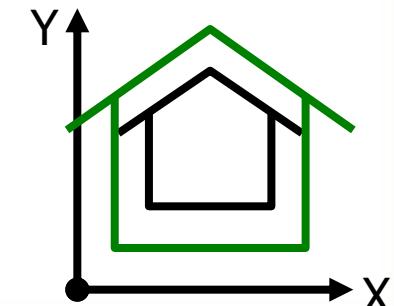
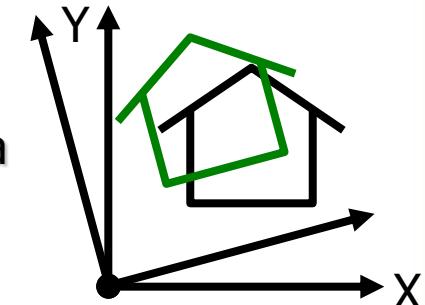
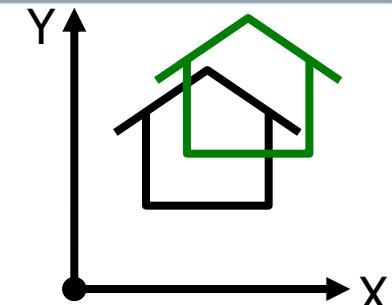
- `void glLoadIdentity(void);`
  - Affecte la transformation courante avec la transformation identité.
- `void glLoadMatrix{f d}(const TYPE *m);`
  - Affecte la transformation courante avec la transformation caractérisée mathématiquement par la matrice m (16 valeurs en coordonnées homogènes).
- `void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



# Transformation de modélisation

# Transformation de modélisation

- `void glTranslate{f d}(TYPE x,TYPE y,TYPE z);`
  - Compose la transformation courante par la translation de vecteur  $(x,y,z)$ .
- `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.
- `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.



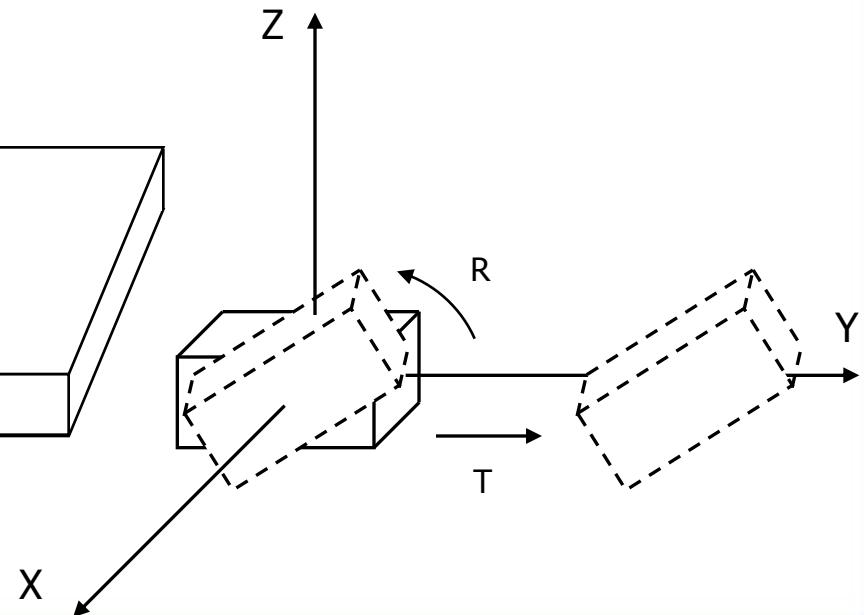
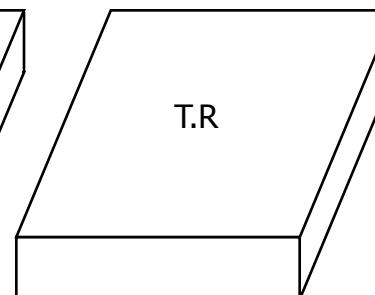
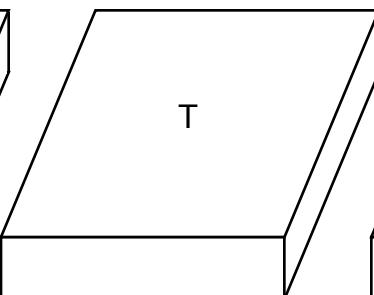
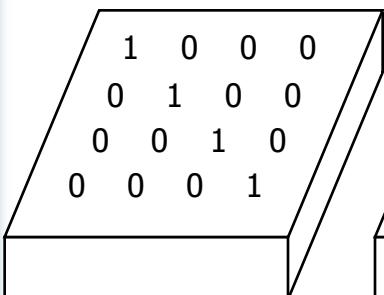


# Transformation de modélisation

- Important :
  - La multiplication n'est pas commutative
  - il faut lire les opérations effectuées sur l'objet dans l'ordre inverse de leur apparition dans le code (propriétés du produit matriciel)

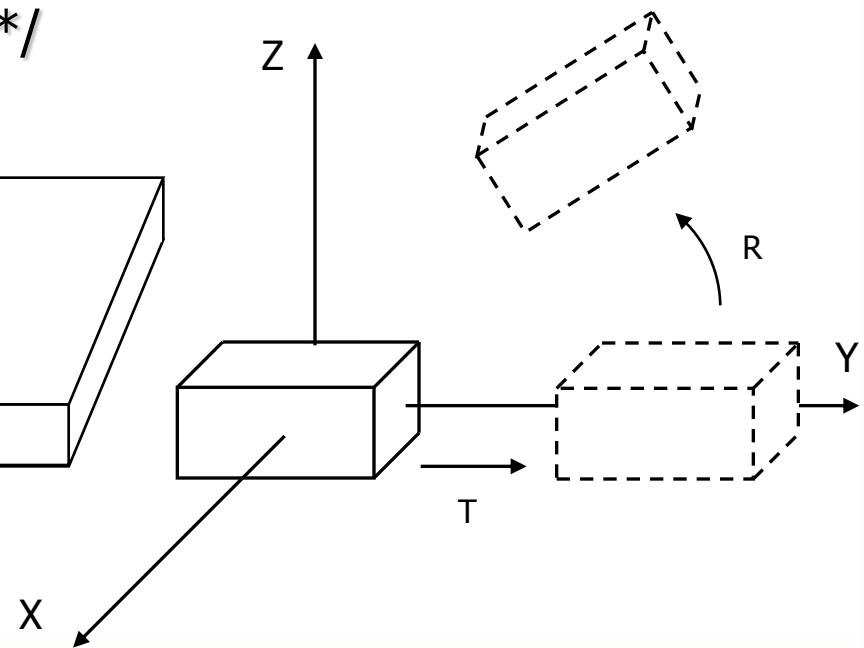
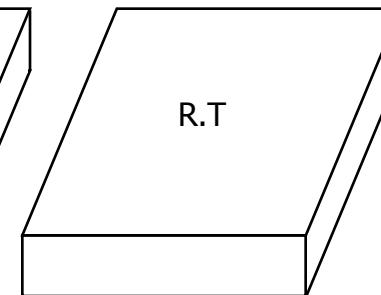
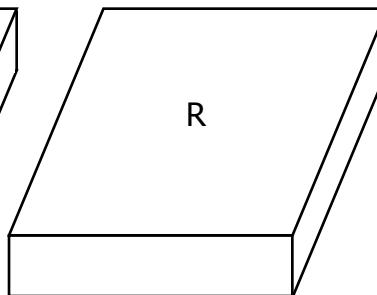
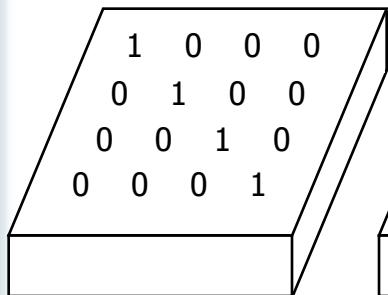
# Transformation de modélisation

```
/* matrice de transformation */  
glMatrixMode (GL_MODELVIEW);  
glLoadIdentity();  
glTranslatef(0, 5, 0);  
glRotatef(45, 1, 0, 0);  
/* objet qui subira la transformation*/  
dessineBoite();
```

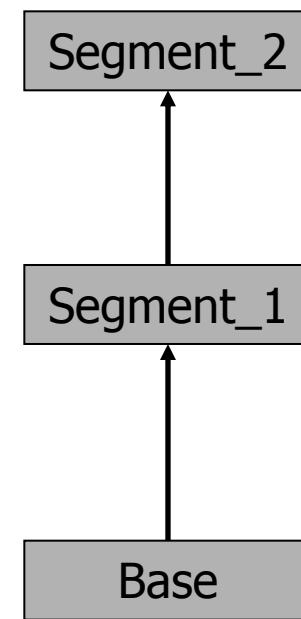
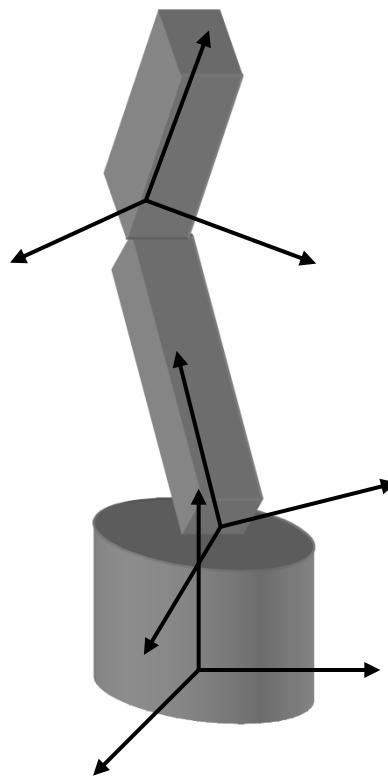


# Transformation de modélisation

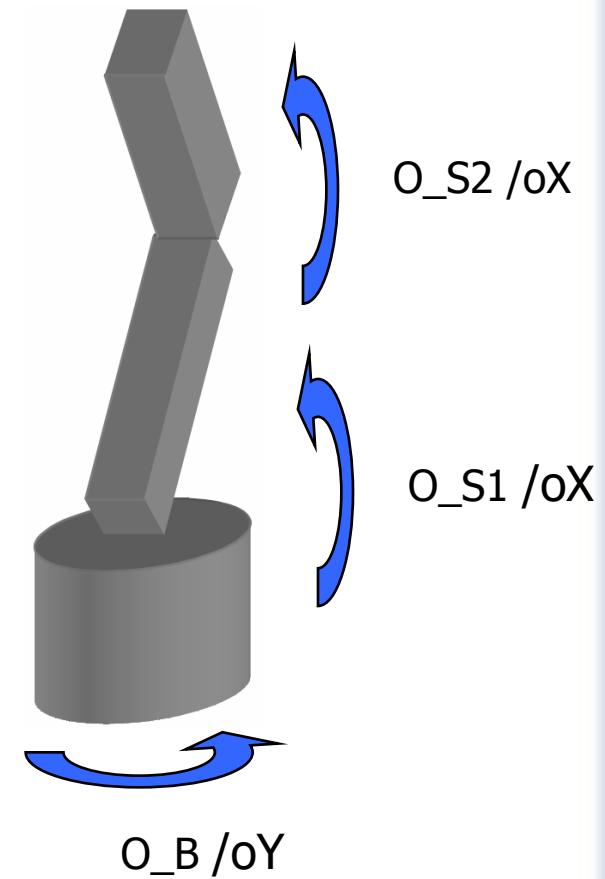
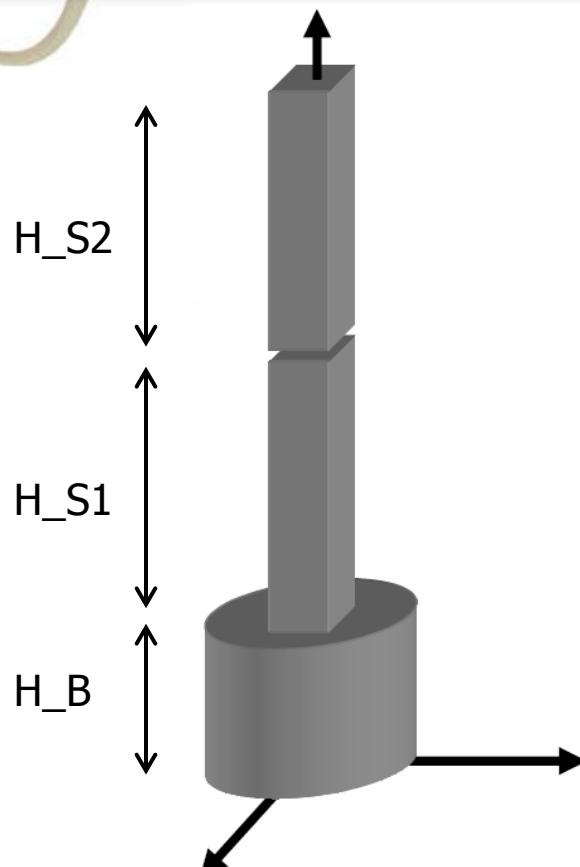
```
/* matrice de transformation */  
glMatrixMode(GL_MODELVIEW);  
glLoadIdentity();  
glRotatef(45, 1, 0, 0);  
glTranslatef(0, 5, 0);  
/* objet qui subira la transformation*/  
dessineBoite();
```



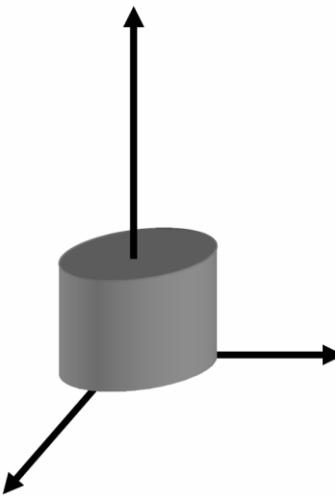
# Description hiérarchique d'un objet



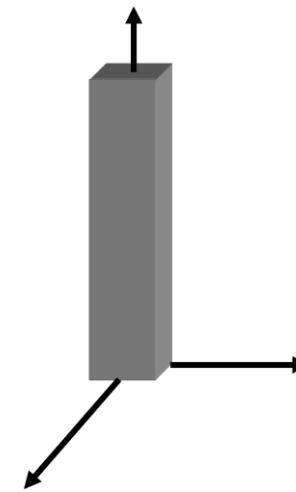
# Description hiérarchique d'un objet



# Description hiérarchique d'un objet



```
base()  
{  
    glBegin(GL_TRIANGLES);  
    ...  
    glEnd;  
}
```

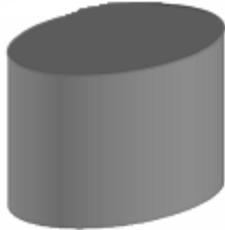


```
Segement_1&2()  
{  
    glBegin(GL_TRIANGLES);  
    ...  
    glEnd;  
}
```



# Description hiérarchique d'un objet

```
glMatrixMode(GL_MODELVIEW);  
glLoadIdentity() ;  
glRotate(O_B, 0.0, 1.0, 0.0);  
base();
```





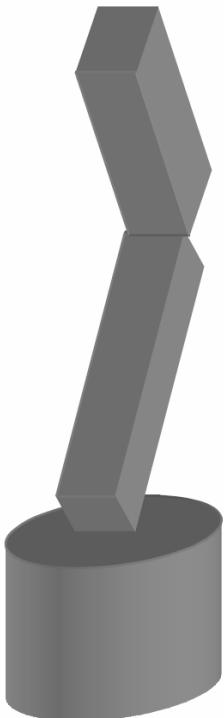
# Description hiérarchique d'un objet



```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glRotate(O_B, 0.0, 1.0, 0.0);
base();
glTranslate(0.0, H_B, 0.0);
glRotate(O_S1, 1.0, 0.0, 0.0);
Segment_1();
```

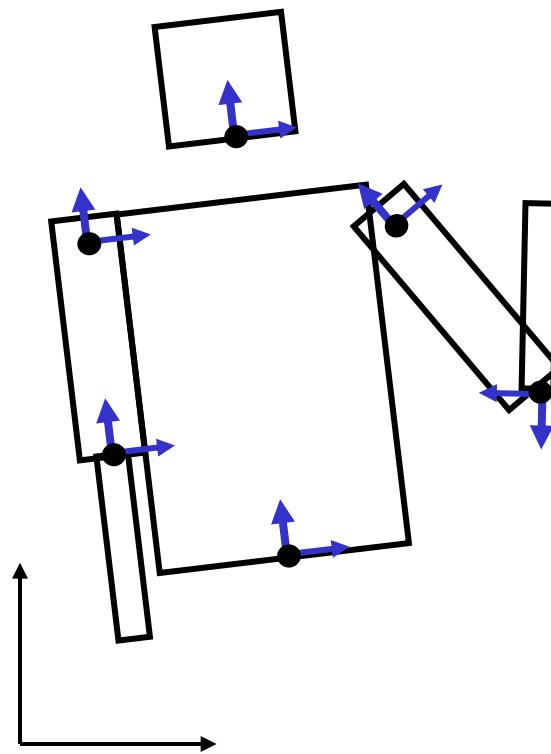


# Description hiérarchique d'un objet



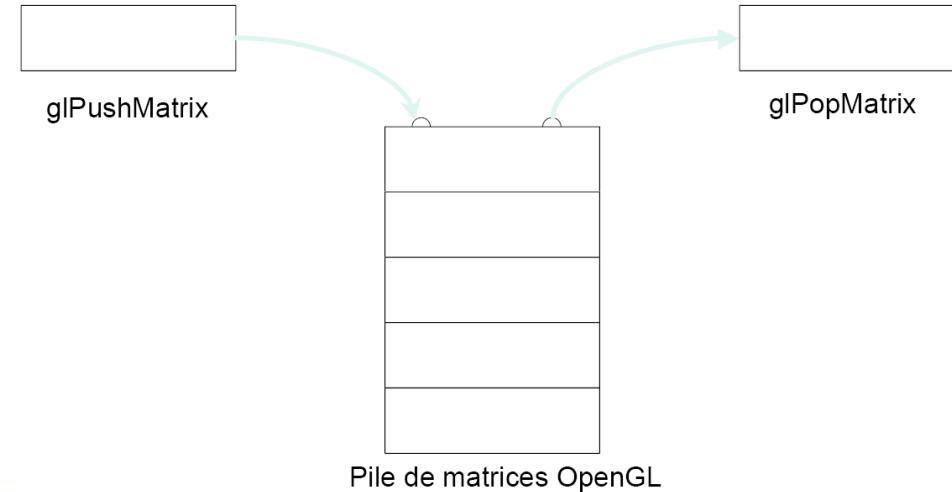
```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glRotate(O_B, 0.0, 1.0, 0.0);
base();
glTranslate(0.0, H_B, 0.0);
glRotate(O_S1, 1.0, 0.0, 0.0);
Segment_1();
glTranslate(0.0, H_S1, 0.0);
glRotate(O_S2, 1.0, 0.0, 0.0);
Segment_2();
```

# Description hiérarchique d'un objet

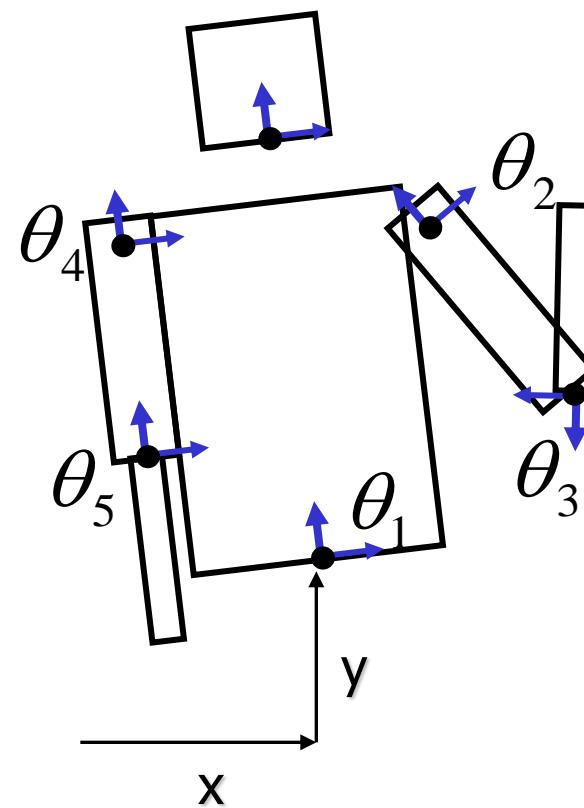
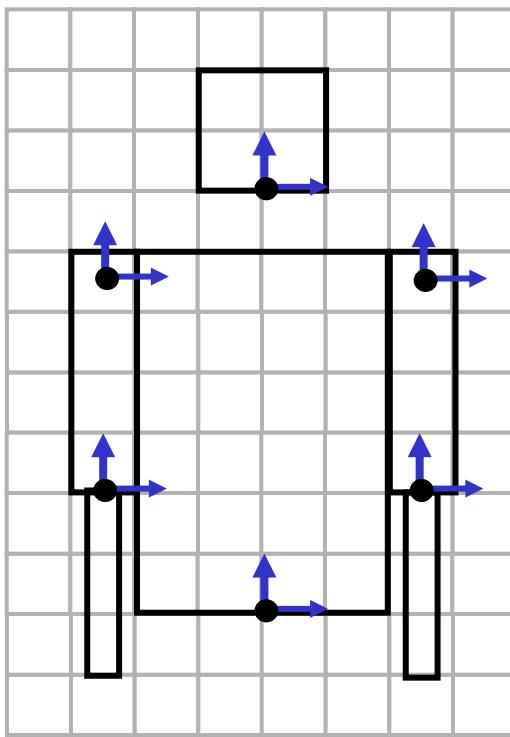


# Description hiérarchique d'un objet

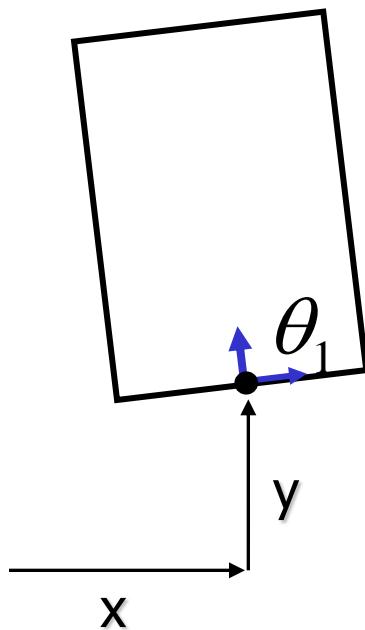
- OpenGL dispose d'une pile de matrices de modélisation
  - Description hiérarchique d'un objet complexe
  - Permet d'appliquer la même transformation à un assemblage
- **glPushMatrix ()**  
Enregistre la matrice actuelle (du mode actuel)  
et prépare un nouveau contexte (avec la même matrice).
- **glPopMatrix ()**  
Restaure le contexte précédent.



# Description hiérarchique d'un objet

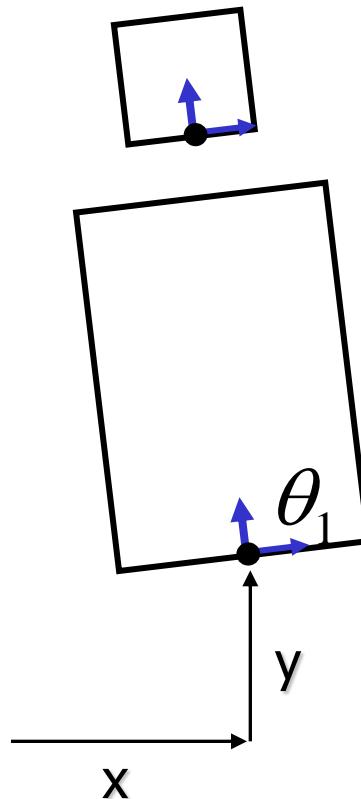


# Description hiérarchique d'un objet



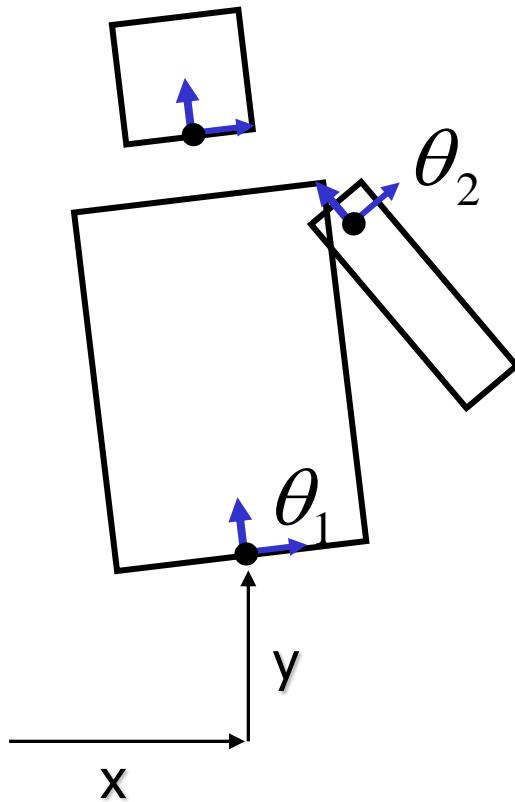
```
glMatrixMode (GL_MODELVIEW);  
glLoadIdentity() ;  
glTranslatef(x,y,0);  
glRotatef(theta1,0,0,1);  
DrawBody();
```

# Description hiérarchique d'un objet



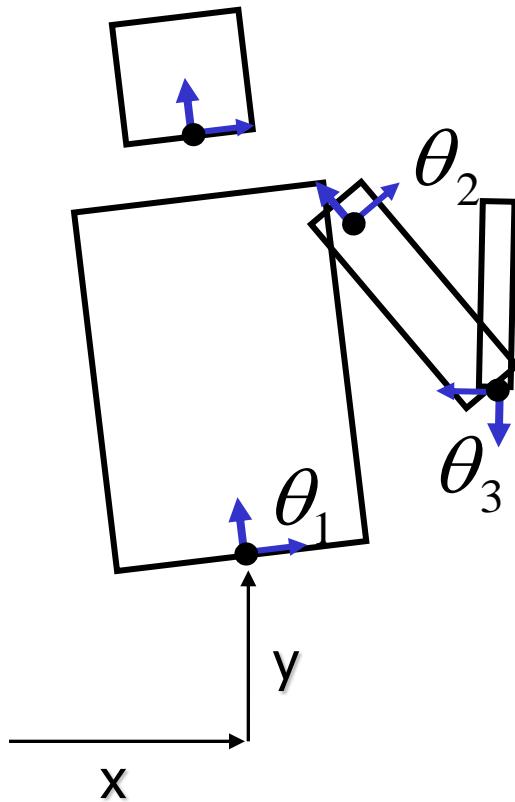
```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(x,y,0);
glRotatef(theta1,0,0,1);
DrawBody();
glPushMatrix();
    glTranslatef(0,7,0);
    DrawHead();
glPopMatrix();
```

# Description hiérarchique d'un objet



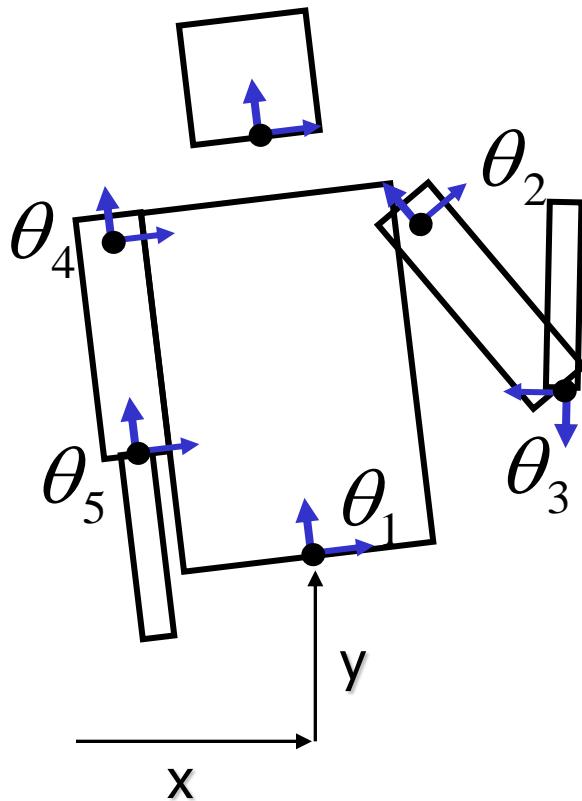
```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslate3f(x,y,0);
glRotatef(theta1,0,0,1);
DrawBody();
glPushMatrix();
    glTranslate3f(0,7,0);
    DrawHead();
glPopMatrix();
glPushMatrix();
    glTranslate(2.5,5.5,0);
    glRotatef(theta2,0,0,1);
    DrawUArm();
glPopMatrix();
```

# Description hiérarchique d'un objet



```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslate3f(x,y,0);
glRotatef(theta1,0,0,1);
DrawBody();
glPushMatrix();
    glTranslate3f(0,7,0);
    DrawHead();
glPopMatrix();
glPushMatrix();
    glTranslate(2.5,5.5,0);
    glRotatef(theta2,0,0,1);
    DrawUArm();
    glTranslate(0,-3.5,0);
    glRotatef(theta3,0,0,1);
    DrawLArm();
glPopMatrix();
```

# Description hiérarchique d'un objet



```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslate3f(x,y,0);
glRotatef(theta1,0,0,1);
DrawBody();
glPushMatrix();
    glTranslate3f(0,7,0);
    DrawHead();
glPopMatrix();
glPushMatrix();
    glTranslate(2.5,5.5,0);
    glRotatef(theta2,0,0,1);
    DrawUArm();
    glTranslate(0,-3.5,0);
    glRotatef(theta3,0,0,1);
    DrawLArm();
glPopMatrix();
... (draw other arm)
```

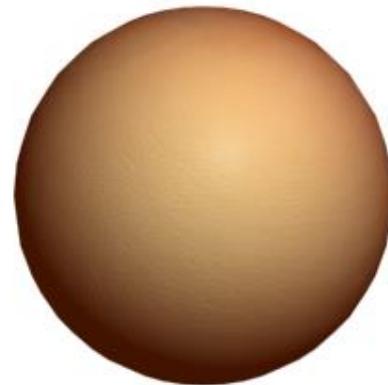
# Description hiérarchique d'un objet

Soleil

|-  
+- Terre

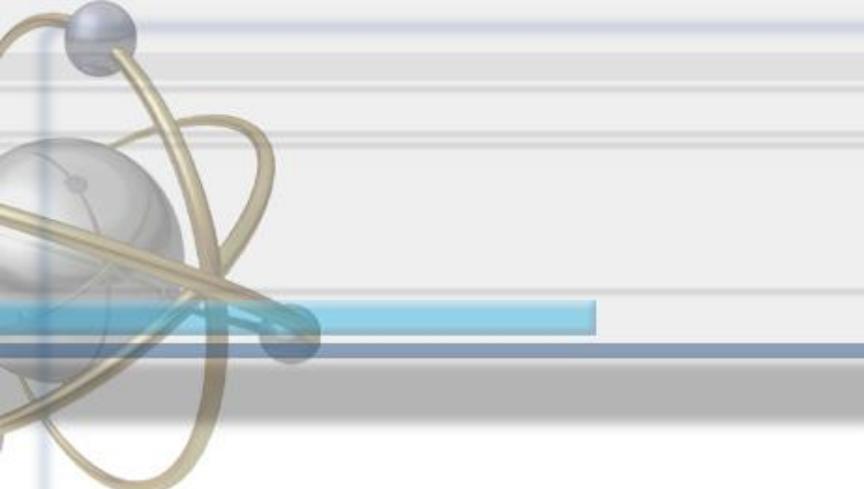
|-   |  
|-   +- Lune

|-  
+- Mars



# Description hiérarchique d'un objet

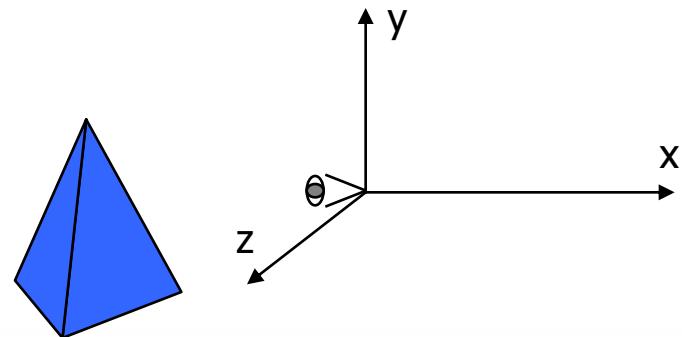
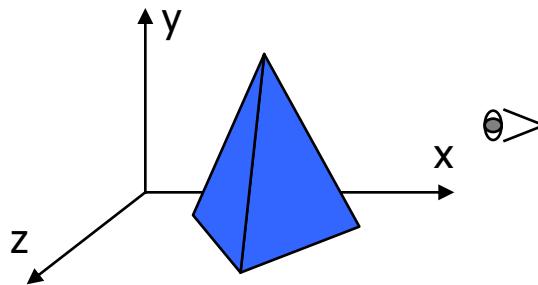
// Code	// pile	// matrice cour.
glPushMatrix();	. (-)	. (-)
drawSun();	. (-)	. (-)
glPushMatrix();	. (-) (S)	. (-) * (S)
glRotatef( angleEarth, 0, 1, 0 );	. (-) (S)	. (-) * (S) * (E)
glTranslatef( 30, 0, 0 );	. (-) (S)	. (-) * (S) * (E)
drawEarth();	. (-) (S)	. (-) * (S) * (E)
glPushMatrix();	. (-) (S) (E)	. (-) * (S) * (E)
glRotatef( angleMoon, 0, 1, 0 );	. (-) (S) (E)	. (-) * (S) * (E) * (M)
glTranslatef( 7, 0, 0 );	. (-) (S) (E)	. (-) * (S) * (E) * (M)
drawMoon();	. (-) (S) (E)	. (-) * (S) * (E) * (M)
glPopMatrix();	. (-) (S)	. (-) * (S) * (E)
glPopMatrix();	. (-)	. (-) * (S)
glPushMatrix();	. (-) ()	. (-) * (S)
glRotatef( angleMars, 0, 1, 0 );	. (-) ()	. (-) * (S) * (Ms)
glTranslatef( 50, 0, 0 );	. (-) ()	. (-) * (S) * (Ms)
drawMars();	. (-) ()	. (-) * (S) * (Ms)
glPopMatrix();	. (-)	. (-) * (S)
glPopMatrix();	. ()	. (-)



# Transformation de visualisation

# Transformation de visualisation

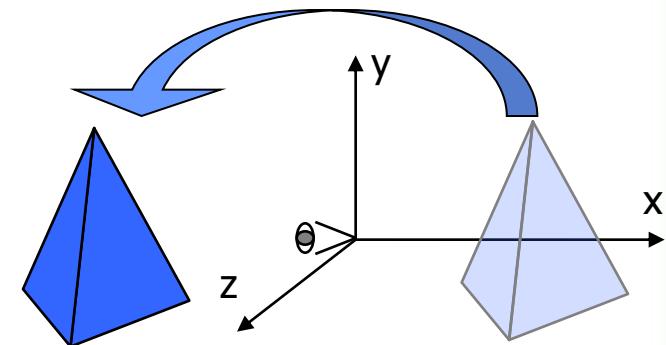
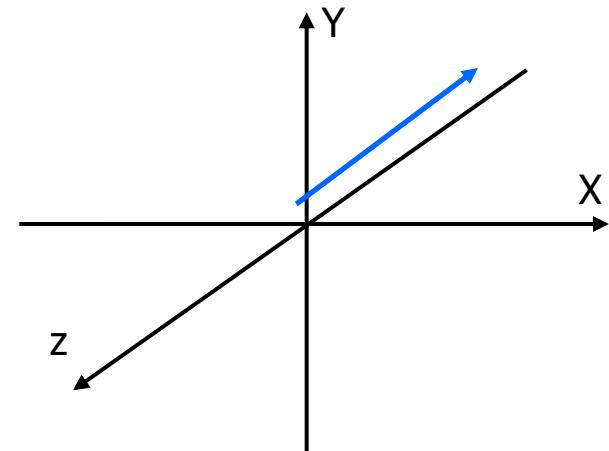
- `glMatrixMode(GL_MODELVIEW)`
  - la matrice active doit être la matrice de modélisation
    - `GL_MODELVIEW` : modélisation & vision
  - Position relative de la caméra par rapport à la scène
  - Les transformations de visualisation et de modélisation n'en forment qu'une pour OpenGL (transformation modelview).
  - Multiplier la matrice courante par une matrice de transformation du repère scène vers le repère caméra



# Transformation de visualisation

- Configuration initiale de la caméra
  - Position : Origine du repère global
  - Orientation : -z

```
glMatrixMode(GL_MODELVIEW);  
glLoadIdentity();  
glTranslatef(-50.0, 0.0, -100.0);  
glRotatef(-90.0, 0.0, 1.0, 0.0);  
Scene();
```



# Transformation de visualisation

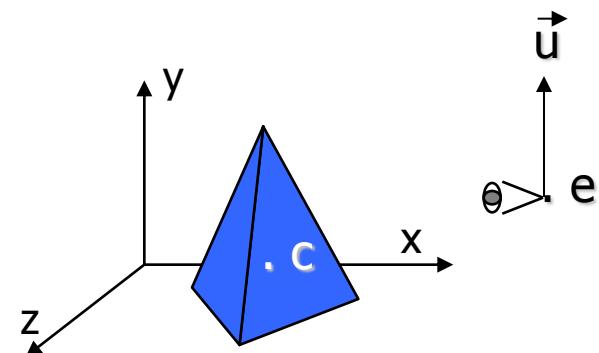
Positionner et orienter la caméra

```
void gluLookAt(  
    GLdouble eX, GLdouble eY, GLdouble eZ,  
    GLdouble cX, GLdouble cY, GLdouble cZ,  
    GLdouble uX, GLdouble uY, GLdouble uZ);
```

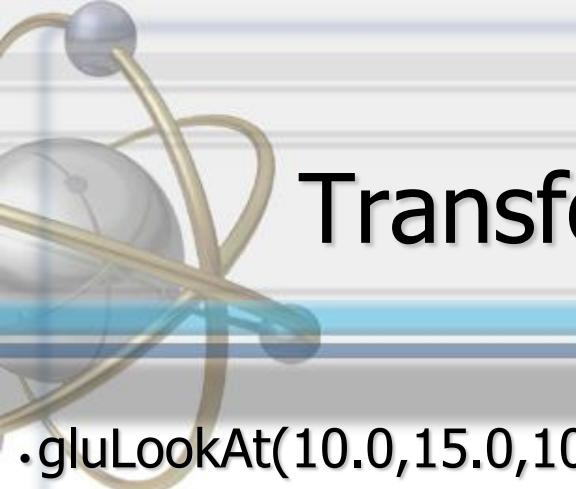
[eX, eY, eZ] : Position de la caméra

[cX, cY, cZ] : Point visé (centre de l'objet observé)

[uX, uY, uZ] : Vecteur qui nous informe de la direction du haut de la caméra.  
(par défaut : 0.0,1.0,0.0)



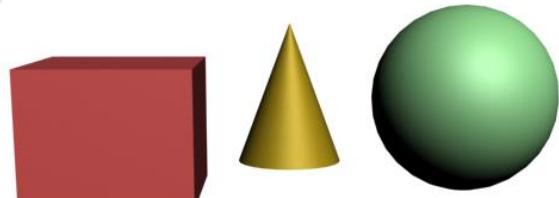
- Dans la pratique, gluLookAt place et oriente la scène devant la caméra de telle manière que la caméra semble placée et orientée selon les paramètres d'entête.
- gluLookAt est une fonction de la bibliothèque GLU
- Deux styles de navigation:
  - Pilote
  - Polaire (tourner autour d'un objet)



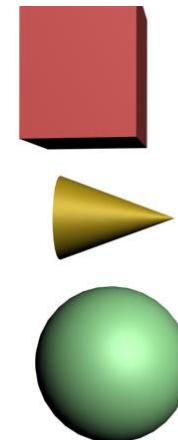
# Transformation de visualisation

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

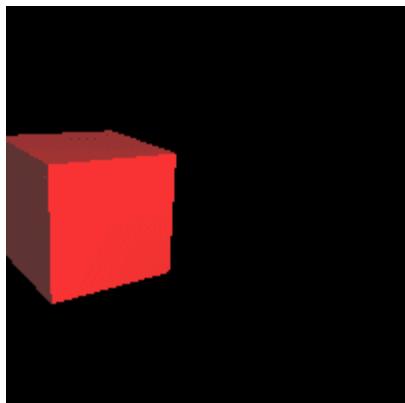
- place la caméra en position (10.0,15.0,10.0),
- l'oriente pour qu'elle vise le point (3.0,5.0,-2.0)
- et visualisera la direction (0.0,1.0,0.0) de telle manière qu'elle apparaisse verticale dans la fenêtre de visualisation.



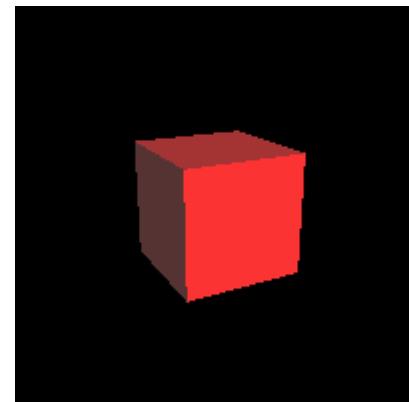
`.gluLookAt(10.0,15.0,10.0,3.0,5.0,-2.0,1.0,0.0,0.0)`



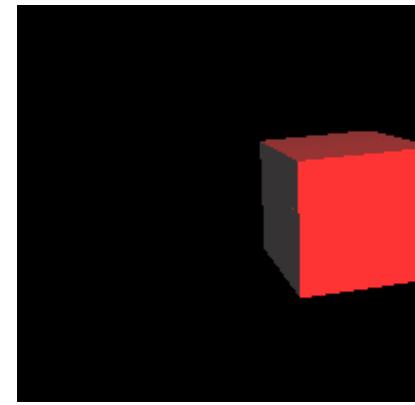
# Transformation de visualisation



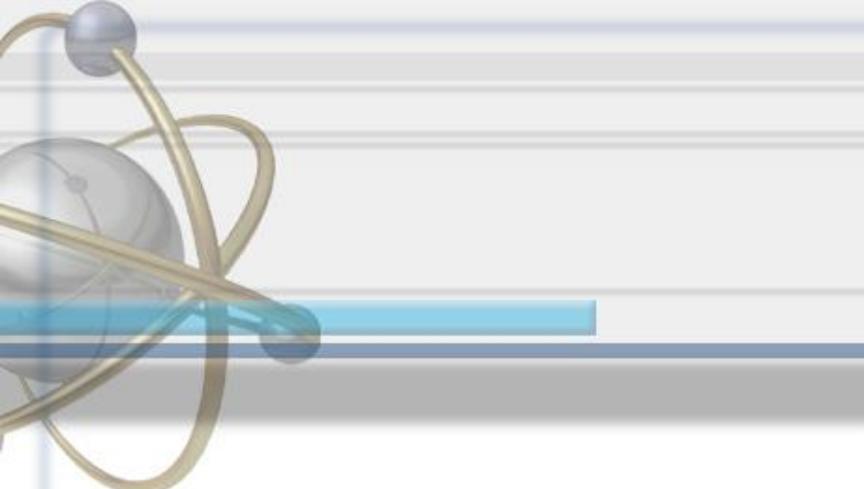
`gluLookAt(0, 0, 4, 1, 0, 0, 0, 1, 0)`



`gluLookAt(0, 0, 4, 0, 0, 0, 0, 1, 0)`



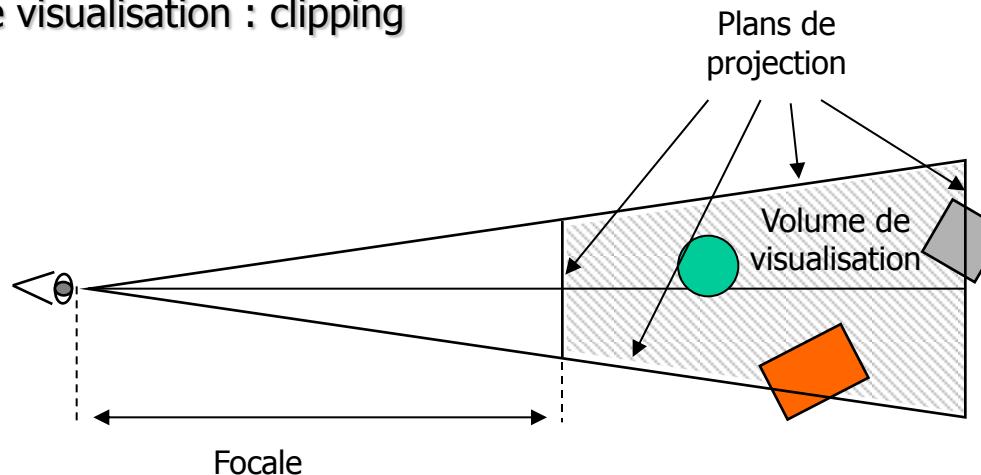
`gluLookAt(0, 0, 4, -1, 0, 0, 0, 1, 0)`



# Transformation de projection

# Transformation de projection

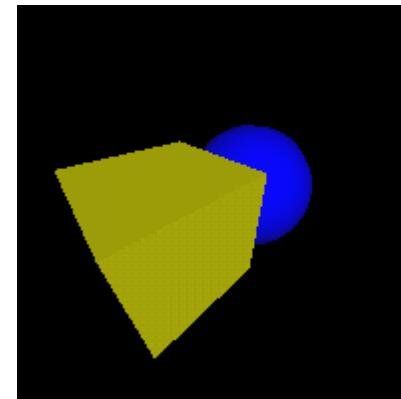
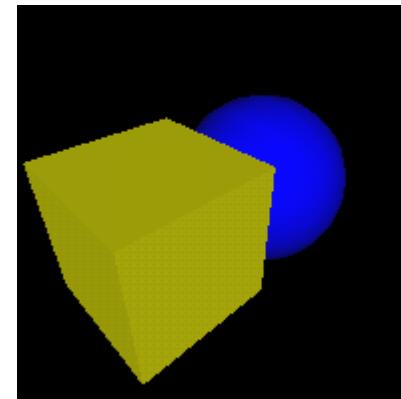
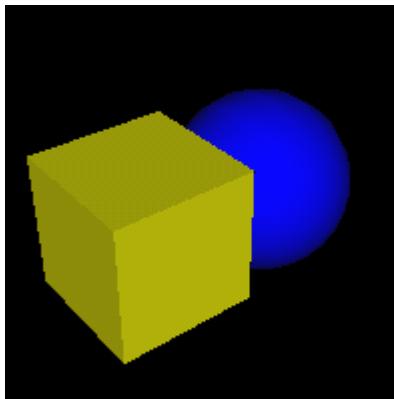
- Les Matrices de PROJECTION du MODELVIEW peuvent être modifiées indépendamment pour permettre une indépendance des scènes modélisées vis à vis des caractéristiques de la « caméra » de visualisation.
- Manipulation de la matrice de projection
  - `glMatrixMode( GL_PROJECTION );`
- L'étape de projection consiste à définir :
  - Les paramètres optiques de projection : focale
  - Le volume de visualisation : clipping





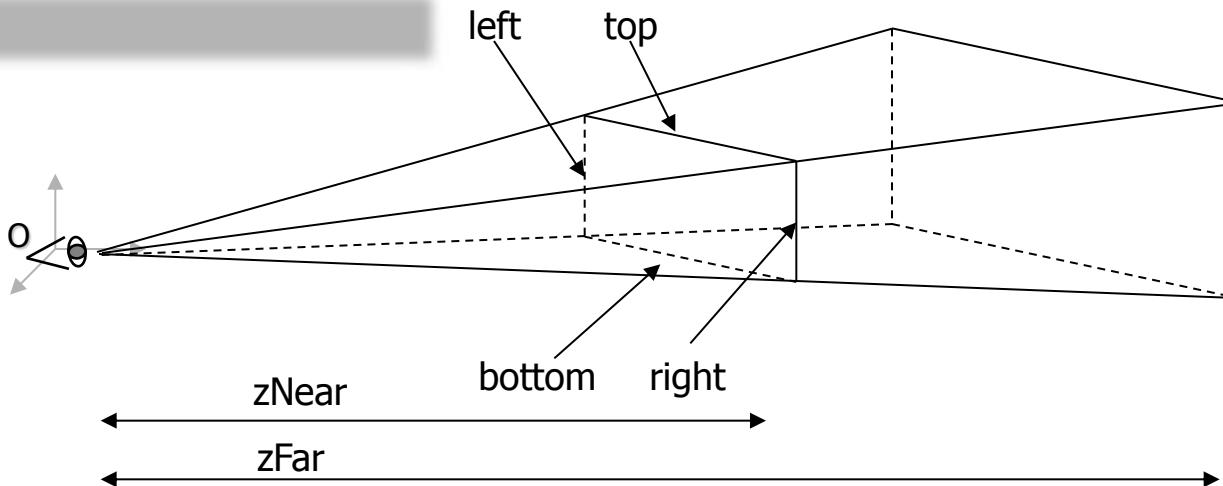
# Transformation de projection

- Projection perspective



# Transformation de projection

```
void glFrustum(  
    GLdouble left,  
    GLdouble right,  
    GLdouble bottom,  
    GLdouble top,  
    GLdouble zNear,  
    GLdouble zFar);
```

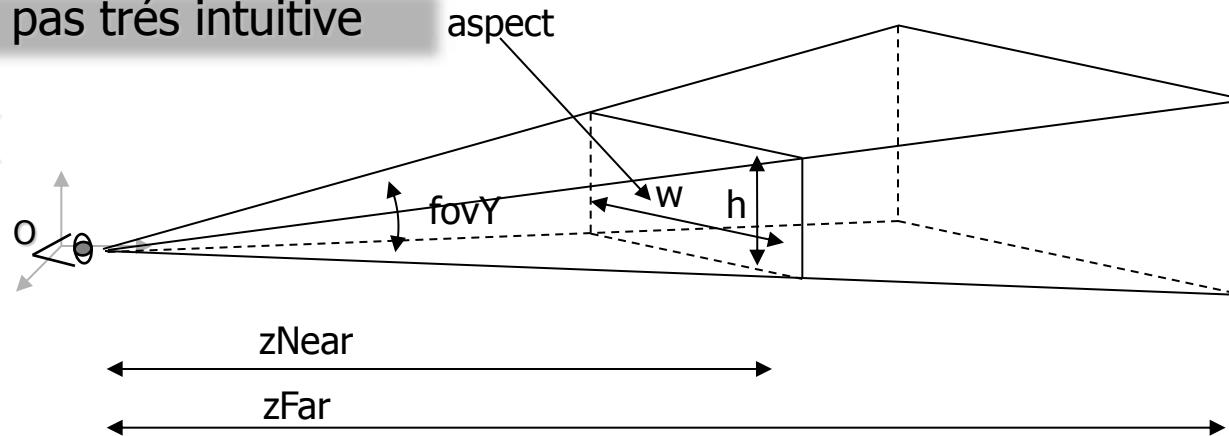


- Compose la transformation courante par la transformation de projection en perspective de volume de visualisation défini par la pyramide tronquée
  - Sommet : l'origine O
  - Orientée selon l'axe  $-z$
  - Plan supérieur défini par la diagonale ( $left, bottom, -zNear$ ) & ( $right, top, -zNear$ ).
- $zNear$  et  $zFar$  sont les distances entre l'origine et les plans de clipping proche ( $-zNear$  en  $z$ ) et éloignés ( $-zFar$  en  $z$ ).
  - $zNear$  et  $zFar$  doivent avoir une valeur positive et respecter  $zNear < zFar$  car il s'agit de distances à l'origine selon l'axe  $-z$ .

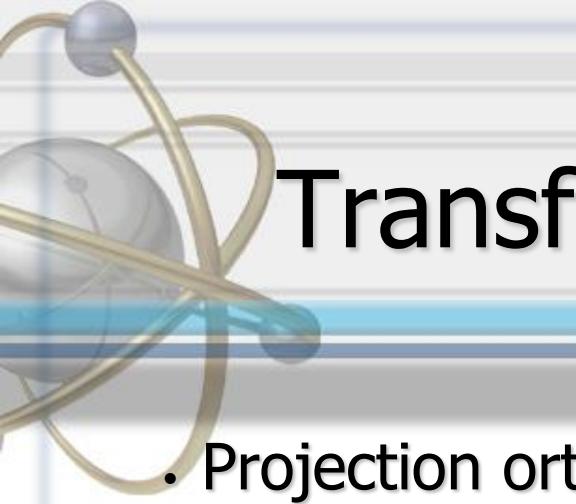
# Transformation de projection

- glFrustum(); N'est pas très intuitive

```
void gluPerspective(  
GLdouble fovY,  
GLdouble aspect,  
GLdouble zNear,  
GLdouble zFar);
```

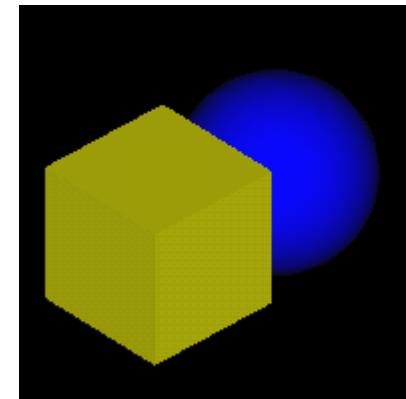
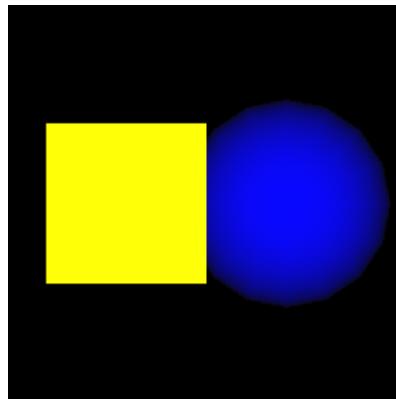


- Compose la transformation courante par la transformation de projection en perspective de volume de visualisation défini par la pyramide tronquée :
  - Sommet : l'origine O
  - Orientée selon l'axe -z
  - Possédant l'angle  $\text{fovY}$  comme angle d'ouverture verticale,
  - L' « aspect » ratio (rapport largeur(w)/hauteur(h))
  - Les plans de clipping proche et éloignés -  $\text{zNear}$  et -  $\text{zFar}$ .
- $\text{zNear}$  et  $\text{zFar}$  doivent avoir une valeur positive et respecter  $\text{zNear} < \text{zFar}$  car il s'agit de distances à l'origine selon l'axe -z.



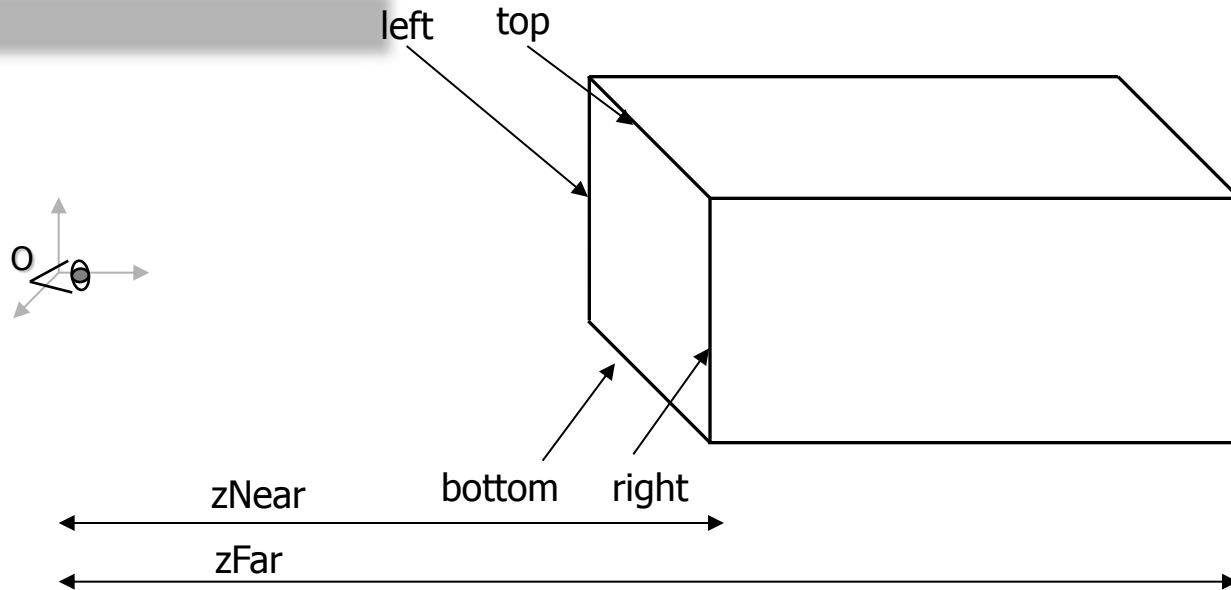
# Transformation de projection

- Projection orthographique

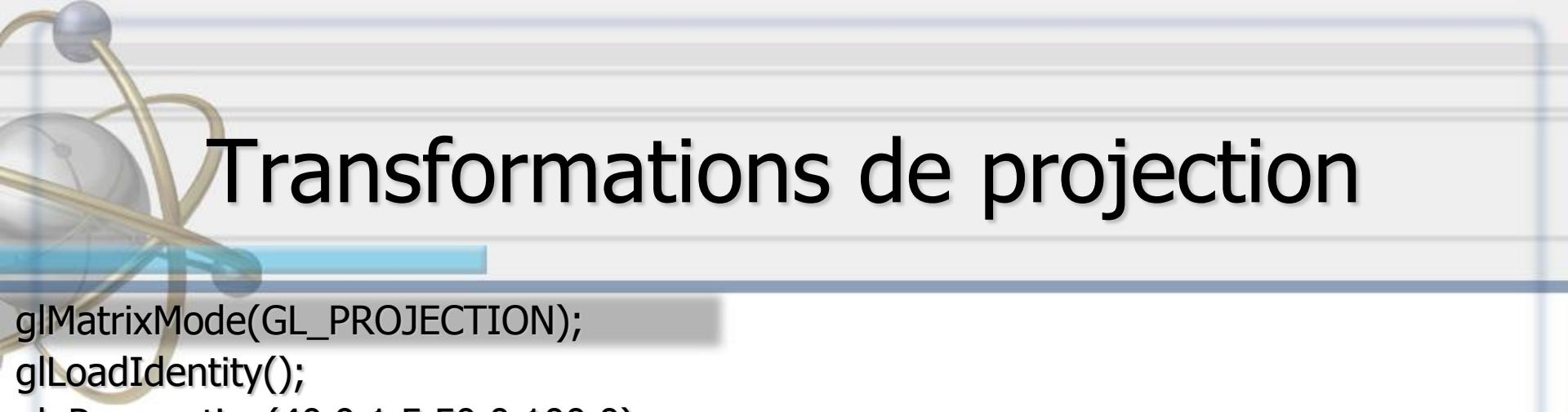


# Transformation de projection

```
void glOrtho(  
    GLdouble left,  
    GLdouble right,  
    GLdouble bottom,  
    GLdouble top,  
    GLdouble zNear,  
    GLdouble zFar);
```



- Compose la transformation courante par la transformation de projection orthographique selon l'axe  $-z$  et définie par le volume de visualisation parallélépipèdique ( $left$ ,  $bottom$ ,  $right$ ,  $top$ ,  $-zNear$ ,  $-zFar$ ).
- $zNear$  et  $zFar$  peuvent être positifs ou négatifs mais doivent respecter  $zNear < zFar$ .



# Transformations de projection

```
glMatrixMode(GL_PROJECTION);
```

```
glLoadIdentity();
```

```
gluPerspective(40.0,1.5,50.0,100.0);
```

```
glMatrixMode(GL_MODELVIEW);
```

```
glLoadIdentity();
```

```
//Dessin scène//
```

## . Configure une caméra de visualisation en perspective

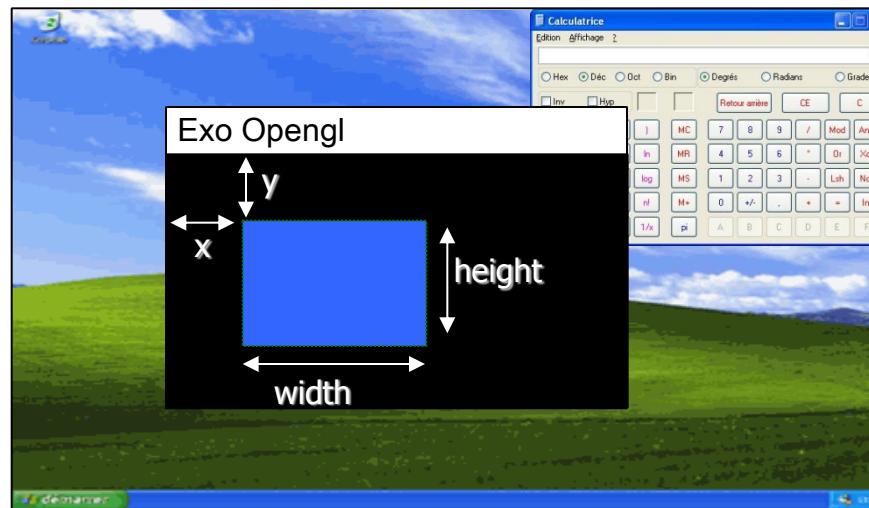
- avec un angle d'ouverture verticale de 40.0°,
- un angle d'ouverture horizontale de  $40.0 \times 1.5 = 60.0^\circ$ ,
- un plan de clipping qui élimine tous les objets ou morceaux d'objet situés en  $z > -50.0$
- un plan de clipping qui élimine tous les objets ou morceaux d'objets situés en  $z < -100.0$ .
- Ces valeurs sont considérées dans le repère global.



# Transformation d'affichage

# Transformation d'affichage

- Définit le rectangle de pixels dans la fenêtre d'affichage dans lequel l'image calculée sera affichée.



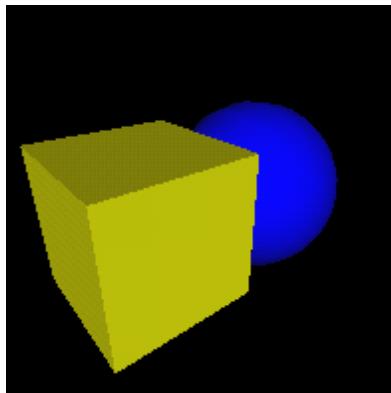
- . **void glViewport(GLint x, GLint y, GLsizei width, GLsizei height);**
  - Fonction indépendante de la matrice de PROJECTION et du MODELVIEW



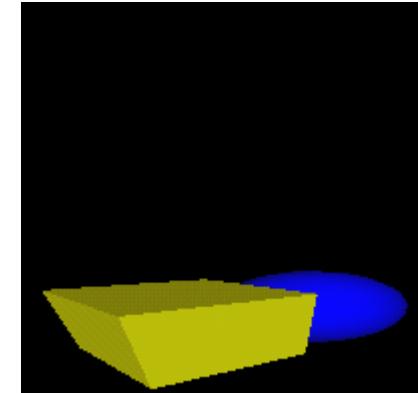
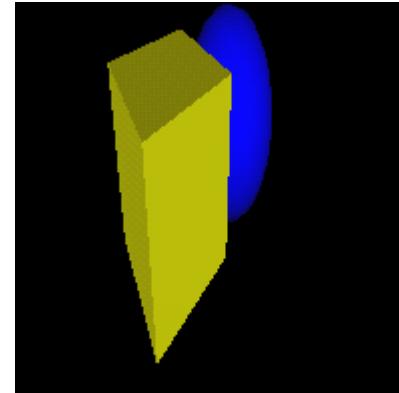
# Transformation d'affichage

- Par défaut le viewport a la taille de la fenêtre
- Le ratio du viewport doit être le même que celui du volume de vision, sinon déformations

Utilisation de toute la surface de la fenêtre d'affichage



Utilisation d'une partie seulement de la fenêtre d'affichage

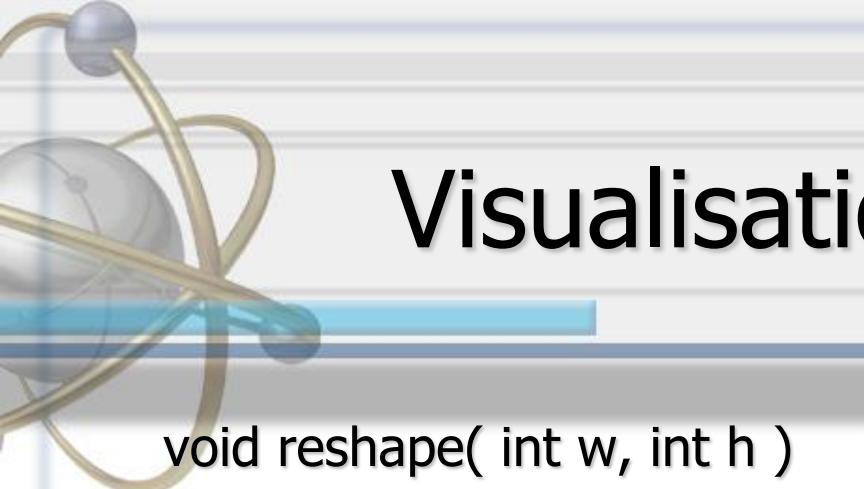




# Transformation d'affichage

- La transformation d'affichage peut s'exprimer sous forme matricielle

```
float v[16] = { width*0.5f,      0,      0,      0,  
                0,      height*0.5f,      0,      0,  
                0,      0,      0.5f,      0,  
                x+0.5f*width,  y+0.5f*height,  0.5f,      1};
```



# Visualisation en OpenGL

```
void reshape( int w, int h )
{
    glViewport( 0, 0, (GLsizei) w, (GLsizei) h );

    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    gluPerspective( 65.0, (GLdouble) w / h, 1.0, 100.0 );

    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();
    gluLookAt(      0.0, 0.0, 5.0,
                  0.0, 0.0, 0.0,
                  0.0, 1.0, 0.0 );
}
```



# Visualisation en OpenGL

```
void reshape( intw, inth )
{
    glViewport( 0, 0, (GLsizei) w, (GLsizei) h );

    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    gluPerspective( 65.0, (GLdouble) w/h, 1.0, 100.0 );

    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();
    glTranslatef( 0.0, 0.0, -5.0 );
}
```



# Visualisation en OpenGL

perspective et vue « polaire »

```
void reshape( int w, int h ) {  
    glViewport( 0, 0, (GLsizei) w, (GLsizei) h );  
  
    glMatrixMode( GL_PROJECTION );  
    glLoadIdentity();  
    gluPerspective( 60.0, (GLfloat) w / h, 1.0, 100.0 );  
  
    glMatrixMode( GL_MODELVIEW );  
    glLoadIdentity();  
    glRotated(-twist, 0.0, 0.0, 1.0);  
    glRotated(-incidence, 1.0, 0.0, 0.0);  
    glRotated(azimuth, 0.0, 0.0, 1.0);  
    glTranslated(0.0, 0.0, -distance);  
}
```



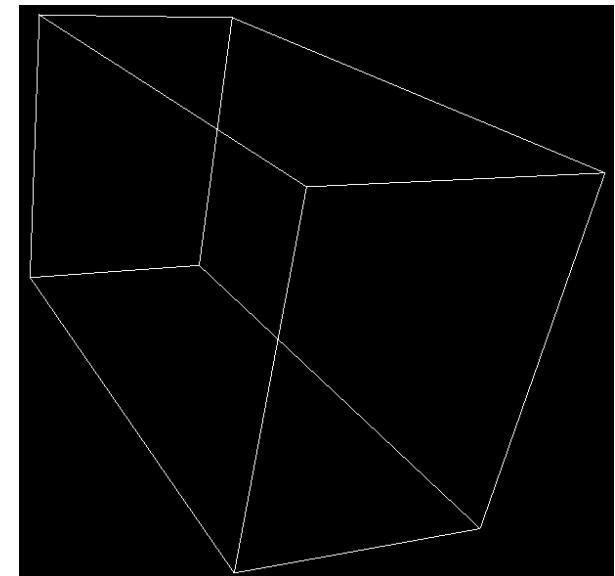
# Visualisation en OpenGL

perspective et vue « pilote »

```
void reshape( int w, int h ) {  
    glViewport( 0, 0, (GLsizei) w, (GLsizei) h );  
  
    glMatrixMode( GL_PROJECTION );  
    glLoadIdentity();  
    gluPerspective( 60.0, (GLfloat) w / h, 1.0, 100.0 );  
  
    glMatrixMode( GL_MODELVIEW );  
    glLoadIdentity();  
    glTranslated(-x, -y, -z);  
    glRotated(roll, 0.0, 0.0, 1.0);  
    glRotated(pitch, 0.0, 1.0, 0.0);  
    glRotated(heading, 1.0, 0.0, 0.0);  
}
```

# Visualisation en OpenGL

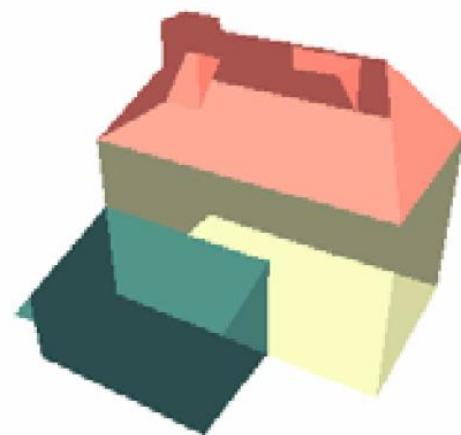
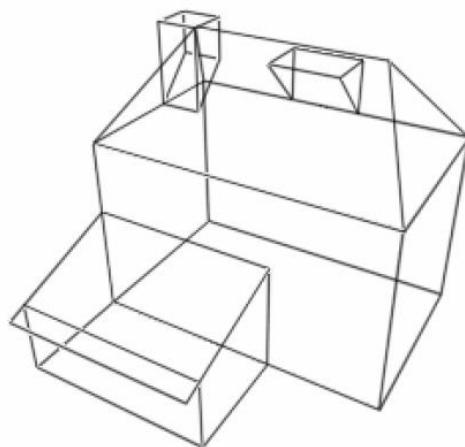
```
/*1 : Transformation de modélisation seulement (Model) */  
/*2 : Transformation visualisation & modélisation */  
/*3 : Transformation de projection */  
/*4 : Transformation d'affichage */  
  
void reshape(int w, int h) {  
    glViewport(0,0,w,h);  
    glMatrixMode(GL_PROJECTION);  
    glLoadIdentity();  
    glFrustum(-1.0,1.0, -1.0,1.0, 1.5,20.);  
    glMatrixMode(GL_MODELVIEW);  
    glLoadIdentity();  
    gluLookAt( 0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0 ); /*2*/  
}  
  
void display(void) {  
    glClear(GL_COLOR_BUFFER_BIT);  
    glColor3f(1.0,1.0,1.0);  
    glTranslatef(0.0,0.0,-5.0);  
    glRotatef(rotx,1.0,0.0,0.0); /*1*/  
    glRotatef(roty,0.0,1.0,0.0); /*1*/  
    glRotatef(rotz,0.0,0.0,1.0); /*1*/  
    glScalef(1.0,2.0,3.0); /*1*/  
    auxWireCube(1.0);  
    glFlush();  
}
```





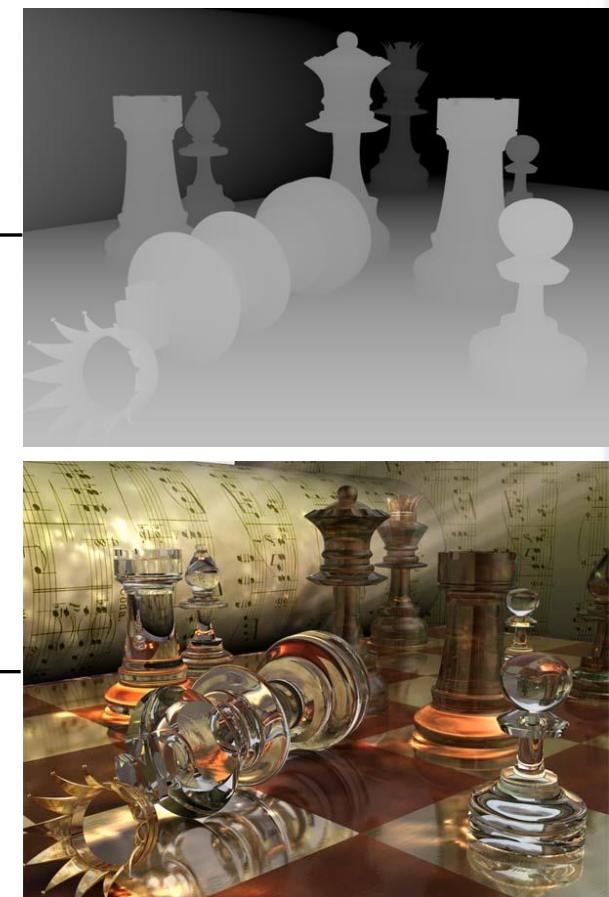
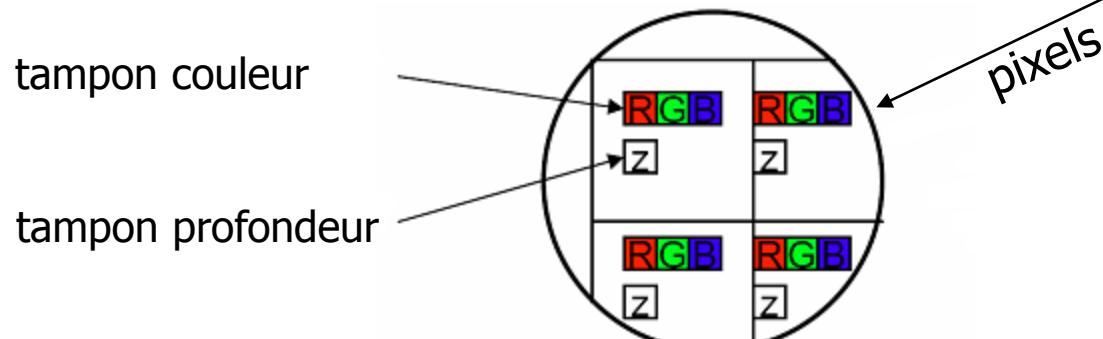
# Élimination des parties cachées

# Élimination des parties cachées



# Élimination des parties cachées

- L'algorithme d'élimination des parties cachées sous OpenGL et le Z-Buffer (Tampon de profondeur)
  - Deux zones mémoire sont concernées :
    - tampon couleur (Color Buffer)
      - 3 composantes : R, G, B
    - tampon profondeur (Z Buffer)
      - Valeurs : [0.0,1.0] par défaut initialisé à 1.0



# Élimination des parties cachées

Initialiser le tableau PROF à  $-\infty$

# Initialiser de tableau COUL

à la couleur de fond

Pour chaque objet o à représenter

Pour chaque pixel  $p=(x,y)$  de  $O$

Calculer la profondeur z de  $p=(x,y)$  ;

Si  $z > \text{PROF}(x,y)$  alors

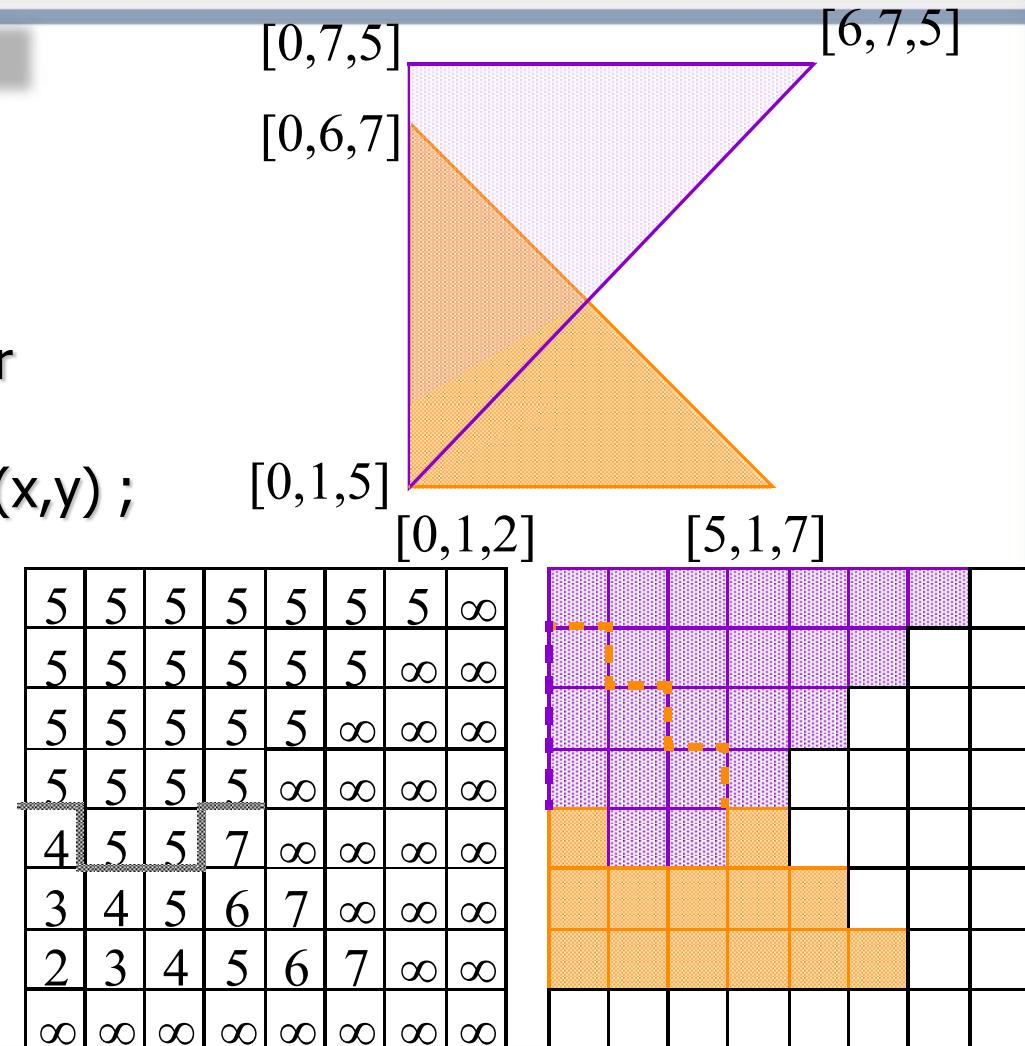
**PROF(x,y) = z ;**

**COUL(x,y) = couleur(o,x,y) ;**

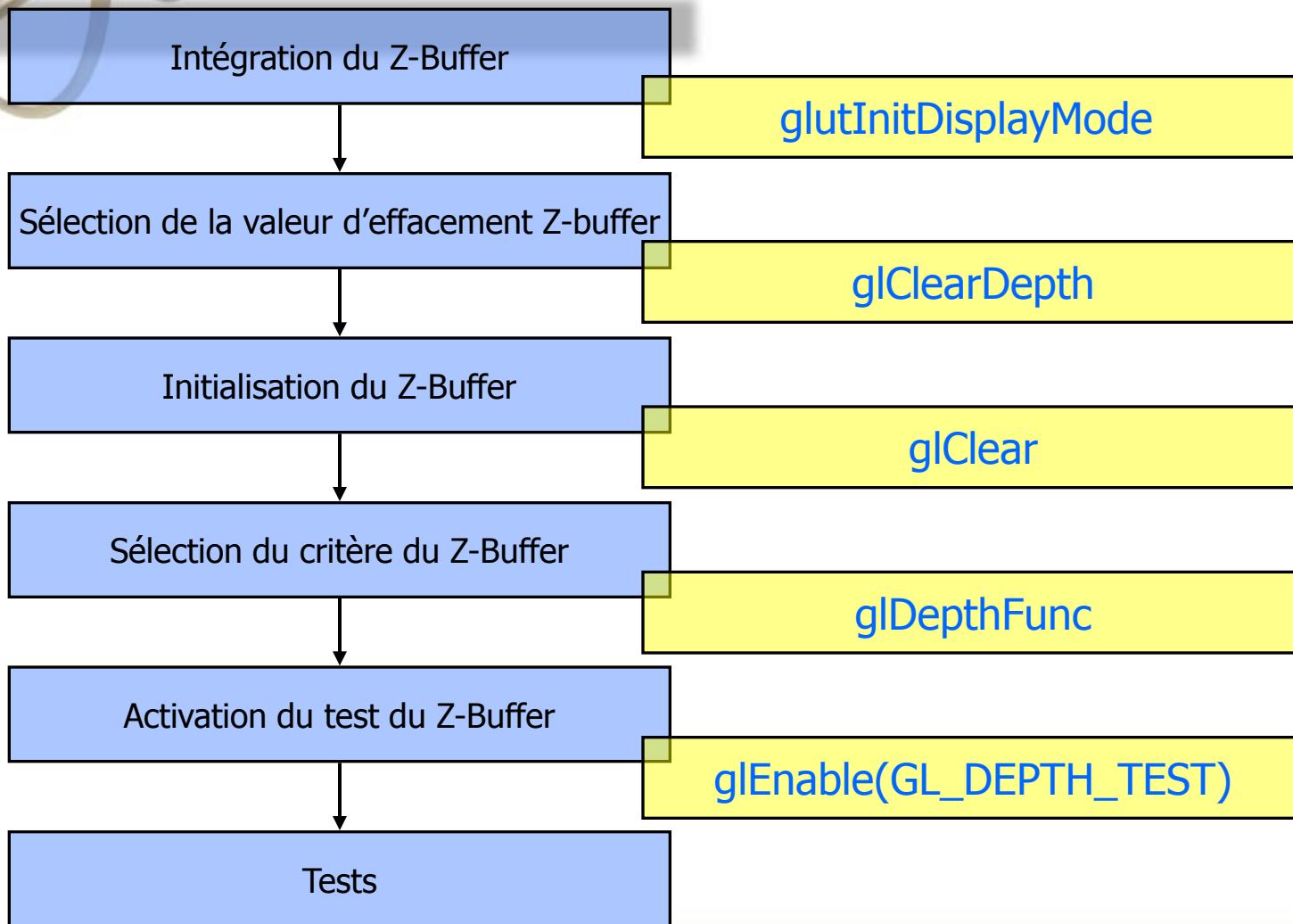
# Finsi

# Finpour

# Finpour



# Étapes d'activation et d'utilisation du Z-Buffer



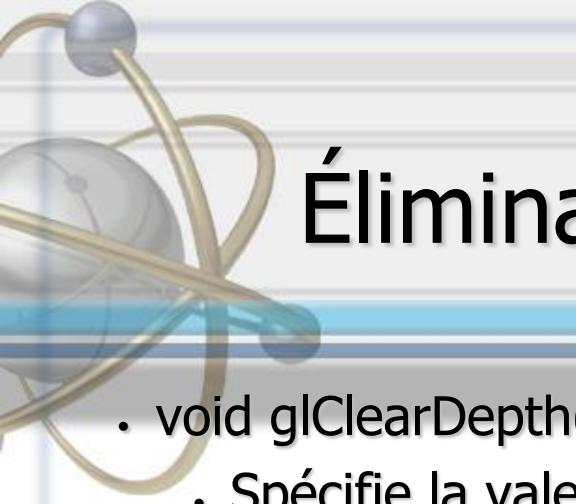


# Élimination des parties cachées

- Intégration du Z-Buffer avec GLUT :

- void glutInitDisplayMode ( unsigned int mode );
  - mode :
    - GLUT\_RGBA / GLUT\_INDEX
    - GLUT\_STENCIL
    - ...
    - GLUT\_DEPTH

```
glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH );
```



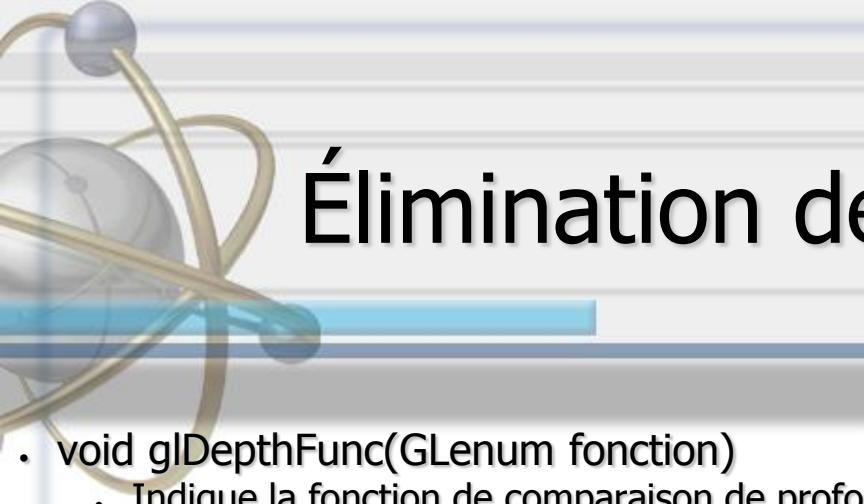
# Élimination des parties cachées

- void glClearDepth( GLclampd depth ) ;
  - Spécifie la valeur de profondeur utilisée quand le buffer de profondeur est effacé: La valeur initiale est 1.

```
glClearDepth(1.0);
```

- void glClear(unsigned intmask);
  - Effacer le Buffer par la valeur indiquée/défaut
  - Intmask :
    - GL\_COLOR\_BUFFER\_BIT
    - GL\_STENCIL\_BUFFER\_BIT
    - ...
    - GL\_DEPTH\_BUFFER\_BIT

```
glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
```



# Élimination des parties cachées

- void glDepthFunc(GLenum fonction)
  - Indique la fonction de comparaison de profondeurs à utiliser
  - Fonction : indique les conditions dans lesquelles le pixel sera dessiné
    - GL\_NEVER
      - Le pixel n'est jamais dessiné.
    - GL\_LESS
      - Le pixel est dessiné si son z est plus petit que le z du tampon.
    - GL\_EQUAL
      - Le pixel est dessiné si son z est égal au z du tampon.
    - GL\_LEQUAL
      - Le pixel est dessiné si son z est inférieur ou égal au z du tampon.
    - GL\_GREATER
      - Le pixel est dessiné si son z est supérieur au z du tampon.
    - GL\_NOTEQUAL
      - Le pixel est dessiné si son z est différent du z du tampon.
    - GL\_GEQUAL
      - Le pixel est dessiné si son z est supérieur ou égal au z du tampon.
    - GL\_ALWAYS
      - Le pixel est toujours dessiné
- La valeur par défaut est GL\_LESS.
- Initialement, le test de profondeur est désactivé.



# Élimination des parties cachées

- Activation / désactivation du test de profondeur
  - glEnable(GL\_DEPTH\_TEST)
  - glDisable(GL\_DEPTH\_TEST)

# Élimination des parties cachées

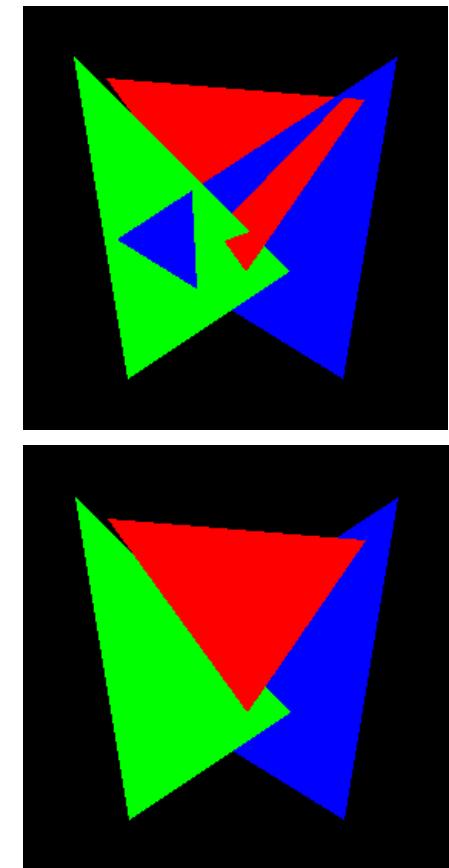
static int mode = 0 ;  
// Variable manipulée  
//par clavier

```
...
glutInitDisplayMode(GLUT_RGBA|GLUT_DOUBLE|GLUT_DEPTH);
...

void display(void) {
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    ...
    if ( mode )
        glEnable(GL_DEPTH_TEST);

    glBegin(GL_TRIANGLES) ;
    glColor4fv(couleurBleu()) ;
    glVertex3f(1.0F,-1.5F,-0.5F) ;
    glVertex3f(1.5F,1.5F,0.5F) ;
    glVertex3f(-1.1F,-0.2F,0.2F) ;
    glColor4fv(couleurVert()) ;
    glVertex3f(-1.0F,-1.5F,-0.5F) ;
    glVertex3f(-1.5F,1.5F,0.5F) ;
    glVertex3f(0.5F,-0.5F,0.2F) ;
    glColor4fv(couleurRouge()) ;
    glVertex3f(-1.2F,1.3F,-0.5F) ;
    glVertex3f(1.2F,1.1F,0.5F) ;
    glVertex3f(0.1F,-0.5F,0.3F) ;
    glEnd() ;

    glDisable(GL_DEPTH_TEST);
}
```



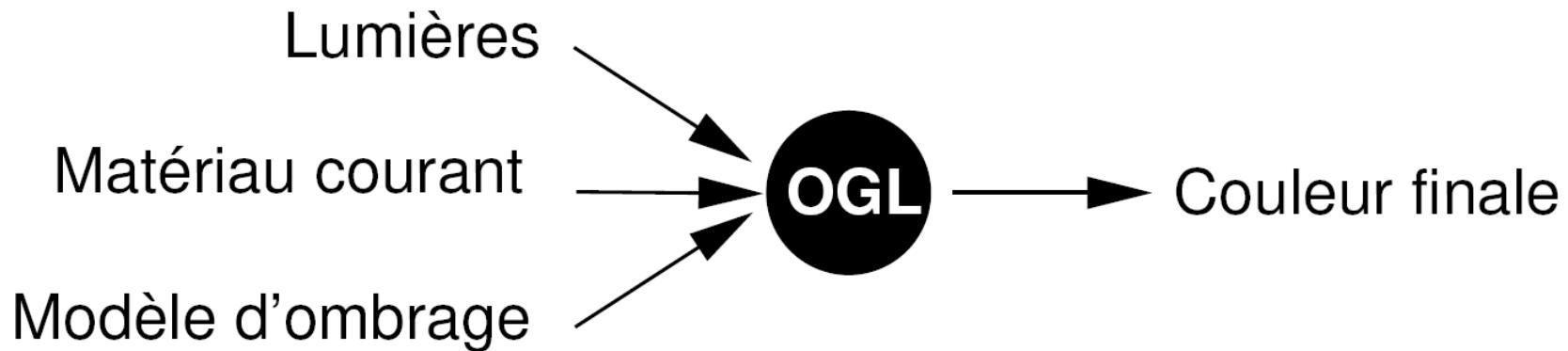
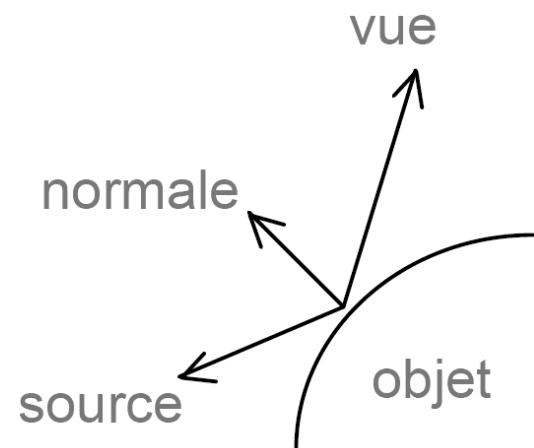


# Eclaircissement et ombrage

# Eclaircissement et ombrage

- 3 Composantes :

- Sources lumineuses : conditions d'éclairage
- Matériaux des objets : réactions à la lumière
- Modèle d'ombrage : Plat, Gouraud, Phong.





# Sources lumineuses

- Définition des propriétés d'une source de lumière

```
void glLight{if}[v](GLenum light, GLenum proprie, TYPE[*] valeur[s]);
```

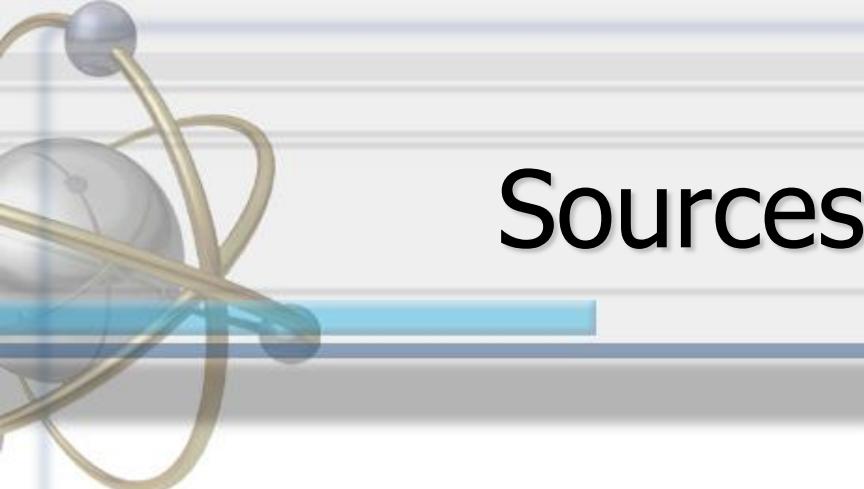
- **light** : Identifiant de la source de lumière
  - GL\_LIGHT*i* avec *i* < GL\_MAX\_LIGHTS.
  - OpenGL dispose de 8 lumières (GL\_LIGHT0, ..., GL\_LIGHT7)
- **proprie** : propriété manipulée
  - composantes : ambiante, diffuse, spéculaire
  - position
  - direction
  - etc.
- **valeur ou valeurs** : valeur(s) attribuée à « proprie » (scalaire/vecteur)



# Sources lumineuses

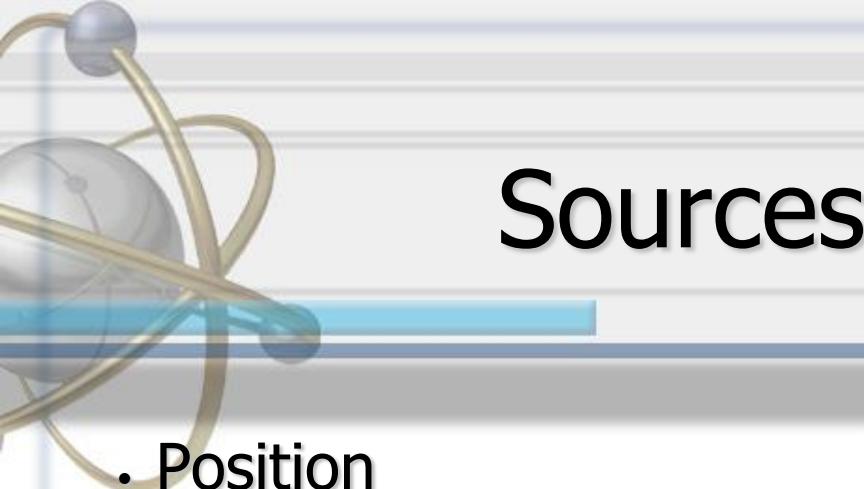
- 3 composantes : ambiante, diffuse, spéculaire
- « propriete » :
  - `GL_AMBIENT` : composante ambiante
  - `GL_DIFFUSE` : composante diffuse
  - `GL_SPECULAR` : composante spéculaire
- « valeurs » : composantes (R,G,B,A)
- Utiliser les versions vectorielles de `glLight` : `glLight{if}v`

Identification du paramètre	Valeur par défaut
<code>GL_AMBIENT</code>	(0.0, 0.0, 0.0, 1.0)
<code>GL_DIFFUSE</code>	(1.0, 1.0, 1.0, 1.0)
<code>GL_SPECULAR</code>	(1.0, 1.0, 1.0, 1.0)



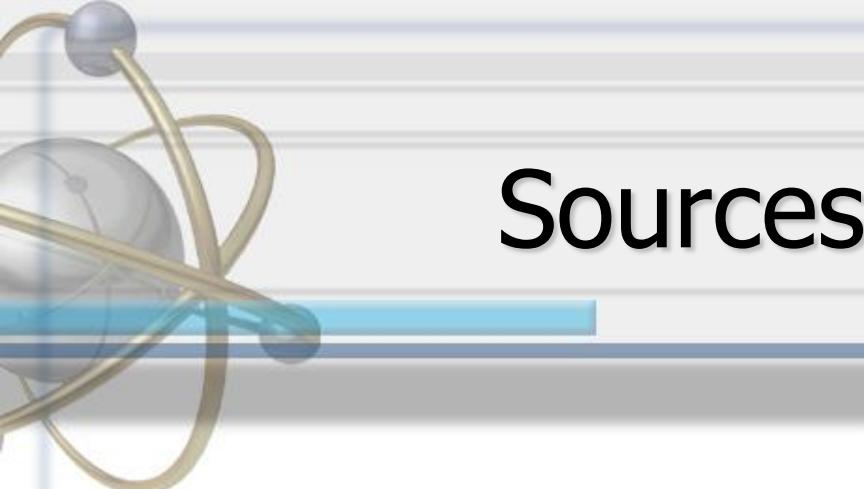
# Sources lumineuses

```
void init(){  
...  
GLfloat lum_ambiante [] = { 0.0, 0.0, 0.0, 1.0};      /* {Black, 1.0}; */  
GLfloat lum_diffuse [] = { 1.0, 0.0, 0.0, 1.0};      /* {Red, 1.0}; */  
GLfloat lum_speculaire [] = { 0.0, 0.0, 1.0, 1.0};    /* {Blue, 1.0}; */  
...  
glLightfv ( GL_LIGHT0, GL_AMBIENT, lum_ambiante);  
glLightfv ( GL_LIGHT0, GL_DIFFUSE, lum_diffuse);  
glLightfv ( GL_LIGHT0, GL_SPECULAR, lum_speculaire);  
...  
}
```



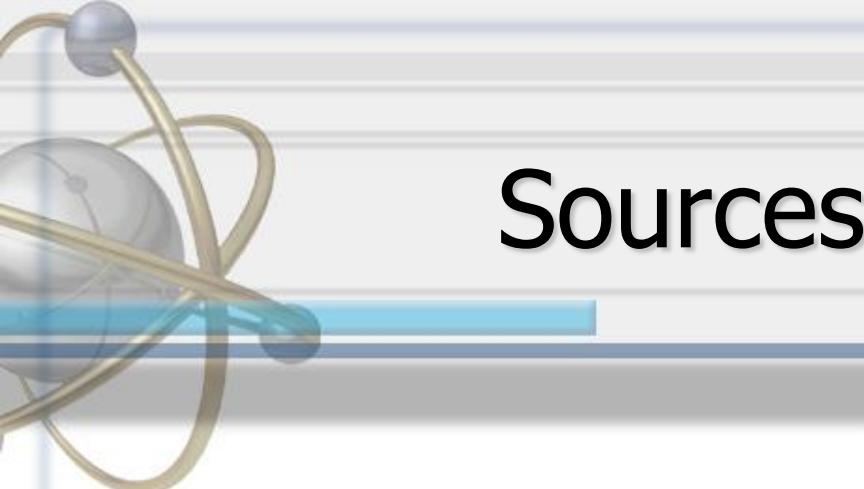
# Sources lumineuses

- Position
  - contrôle la position et le type de lumière
- « propriete » : **GL\_POSITION**
- « valeurs » : position ( $x,y,z,w$ )
- Source omni-directionnelle ( $w \neq 0$ ) : émission isotropique
- Source directionnelle ( $w=0$ ) : direction  $(-x,-y,-z)$
- Défaut :  $(0,0,1,0)$



# Sources lumineuses

```
void init(){  
...  
GLfloat lum_ambiante [] = { 0.0, 0.0, 0.0, 1.0};      /* {Black, 1.0}; */  
GLfloat lum_diffuse [] = { 1.0, 0.0, 0.0, 1.0};        /* {Red, 1.0}; */  
GLfloat lum_speculaire [] = { 0.0, 0.0, 1.0, 1.0};     /* {Blue, 1.0}; */  
GLfloat lightposition[4]={1.0f,1.0f,1.0f,1.0f};  
  
...  
glLightfv ( GL_LIGHT0, GL_AMBIENT, lum_ambiante);  
glLightfv ( GL_LIGHT0, GL_DIFFUSE, lum_diffuse);  
glLightfv ( GL_LIGHT0, GL_SPECULAR, lum_speculaire);  
glLightfv( GL_LIGHT0, GL_POSITION, lightposition);  
...  
}
```



# Sources lumineuses

- Comme pour les objets, les sources de lumière sont sujettes aux matrices de transformations.
- La position et la direction sont affectées par la matrice MODELVIEW courante et stockées en coordonnées caméra.



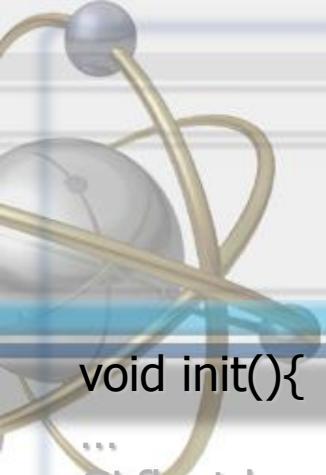
# Sources lumineuses

Atténuation de la source lumineuse

- Variation de l'intensité en fonction de la distance
  - plus un objet est éloigné d'une source de lumière, moins il est éclairé par cette dernière :
    - $Ke$  : GL\_CONSTANT\_ATTENUATION
      - Défaut : 1.0.
    - $k_l$  : GL\_LINEAR\_ATTENUATION
      - Défaut : 0.0.
    - $k_q$  : GL\_QUADRATIC\_ATTENUATION
      - Défaut : 0.0.
    - $d$  distance par rapport à la lumière

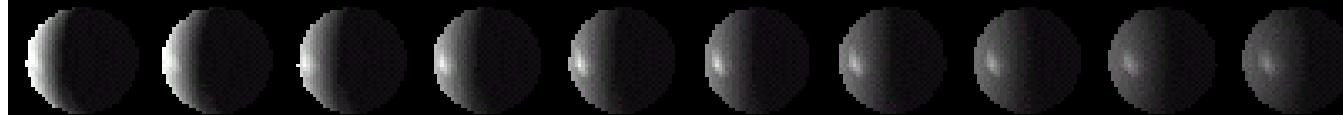
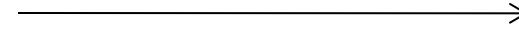
$$f = \frac{1}{k_e + k_l d + k_q d^2}$$





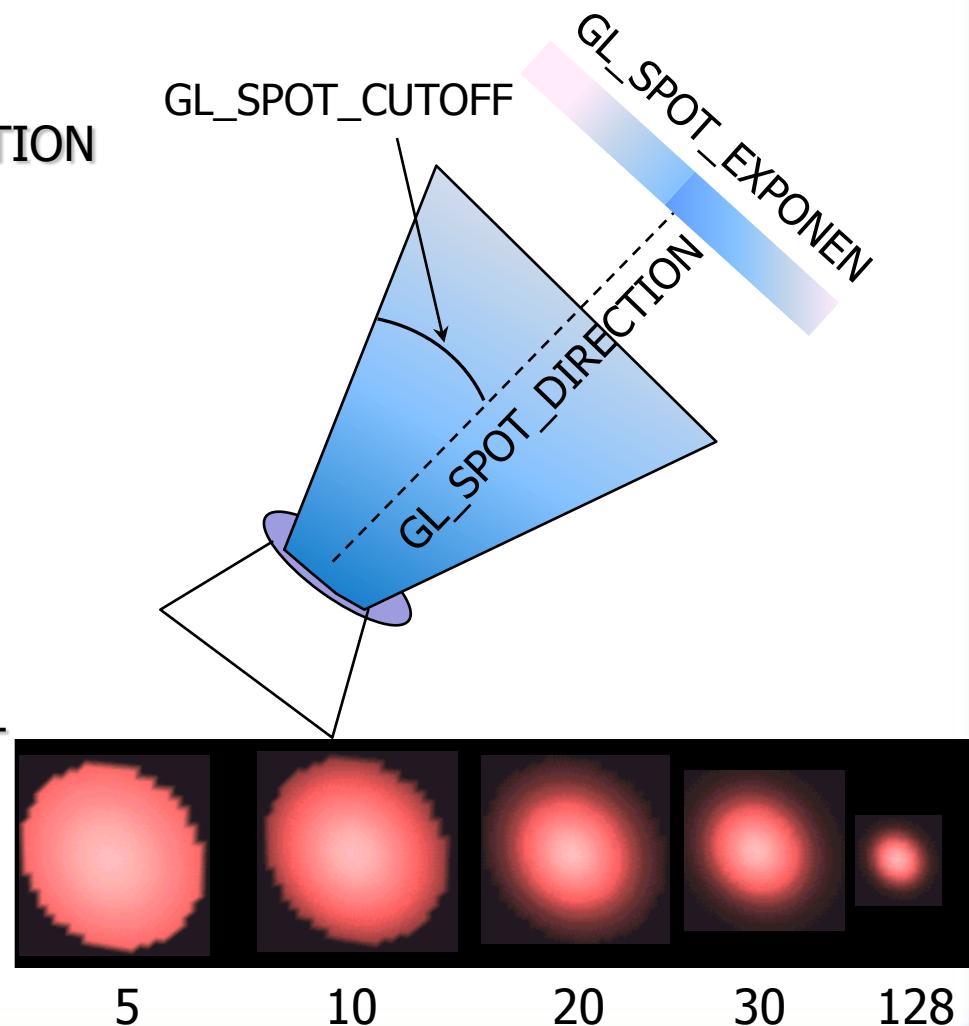
# Sources lumineuses

```
void init(){  
    ...  
    GLfloat lum_ambiante [] = { 0.0, 0.0, 0.0, 1.0}; /* {Black, 1.0}; */  
    GLfloat lum_diffuse [] = { 1.0, 0.0, 0.0, 1.0}; /* {Red, 1.0}; */  
    GLfloat lum_speculaire [] = { 0.0, 0.0, 1.0, 1.0}; /* {Blue, 1.0}; */  
    GLfloat lightposition[4]={1.0f,1.0f,1.0f,1.0f};  
    GLfloat LinearFacort = 1.0f ;  
  
    ...  
    glLightfv ( GL_LIGHT0, GL_AMBIENT, lum_ambiante);  
    glLightfv ( GL_LIGHT0, GL_DIFFUSE, lum_diffuse);  
    glLightfv ( GL_LIGHT0, GL_SPECULAR, lum_speculaire);  
    glLightfv( GL_LIGHT0, GL_POSITION, lightposition);  
    glLightf(GL_LIGHT0,GL_LINEAR_ATTENUATION,LinearFacort);  
    ...  
}
```



# Sources lumineuses

- Définition d'un cône lumineux : Spot
  - Direction:
    - « propriété » : GL\_SPOT\_DIRECTION
    - « valeurs » :  $(x,y,z)$ 
      - Défaut :  $(0,0,1)$
  - Angle d'ouverture :
  - « propriété » : GL\_SPOT\_CUTOFF
  - « valeurs » :
    - Valeurs :  $[0,90]$  et  $180$
    - Défaut :  $180$  (degrés)
  - Fonction d'atténuation
  - « propriété » : GL\_SPOT\_EXPONENT
  - « valeurs » :
    - Valeurs :  $[0,128]$
    - Défaut :  $0$

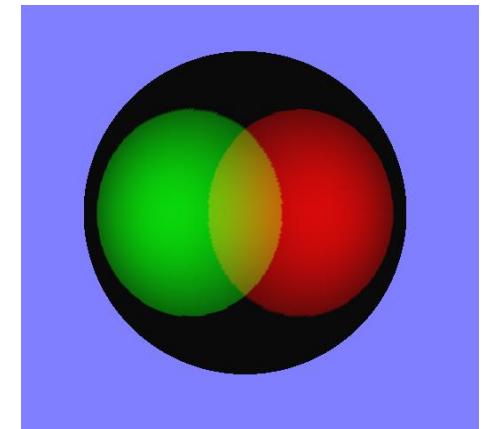


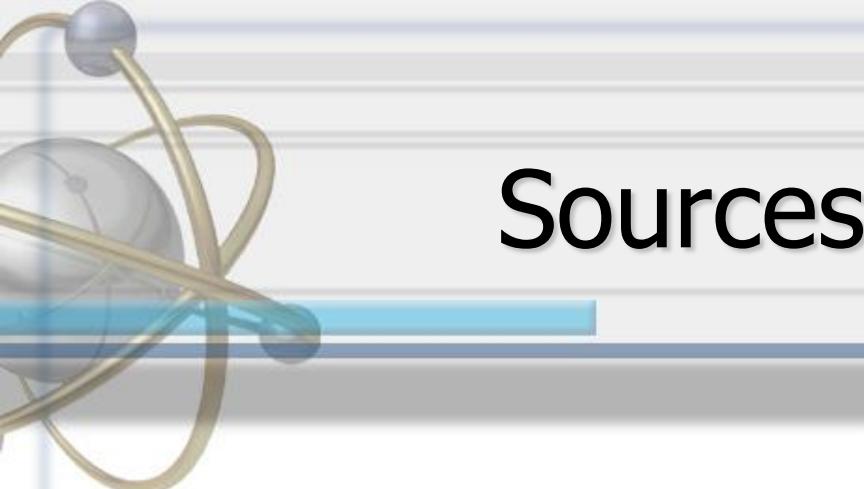
# Sources lumineuses

```
void myinit(void) {  
    static float o0 = 10.0F;  
    static float o1 = 10.0F;  
  
    GLfloat rouge[] = { 1.0F,0.0F,0.0F,1.0F };  
    GLfloat vert[] = { 0.0F,1.0F,0.0F,1.0F };  
  
    GLfloat l_pos0[] = { 1.0F,0.0F,2.0F,1.0F };  
    GLfloat l_dir0[] = { -1.0F,0.0F,-2.0F,1.0F };  
  
    GLfloat l_pos1[] = { -1.0F,0.0F,2.0F,1.0F };  
    GLfloat l_dir1[] = { 1.0F,0.0F,-2.0F,1.0F };  
}
```

```
...  
    glLightfv(GL_LIGHT0,GL_DIFFUSE,rouge);  
    glLightfv(GL_LIGHT0,GL_POSITION,l_pos0);  
    glLightfv(GL_LIGHT0,GL_SPOT_DIRECTION,l_dir0);  
    glLightf(GL_LIGHT0,GL_SPOT_CUTOFF,o0);
```

```
    glLightfv(GL_LIGHT1,GL_DIFFUSE,vert);  
    glLightfv(GL_LIGHT1,GL_POSITION,l_pos1);  
    glLightfv(GL_LIGHT1,GL_SPOT_DIRECTION,l_dir1);  
    glLightf(GL_LIGHT1,GL_SPOT_CUTOFF,o1);  
}
```





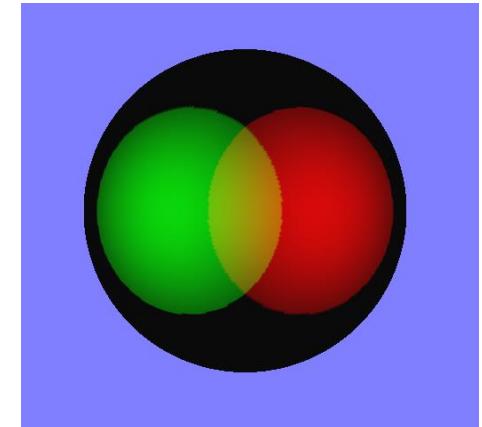
# Sources lumineuses

- Activation/désactivation de l'éclairage
  - glEnable/Disable(GL\_LIGHTING);
- Activation/désactivation d'une source de lumière i
  - glEnable/Disable(GL\_LIGHTi);
    - $i < GL\_MAX\_LIGHTS$
    - Huit sources de lumières au maximum

# Sources lumineuses

```
void myinit(void) {  
    static float o0 = 10.0F;  
    static float o1 = 10.0F;  
  
    GLfloat rouge[] = { 1.0F,0.0F,0.0F,1.0F };  
    GLfloat vert[] = { 0.0F,1.0F,0.0F,1.0F };  
  
    GLfloat l_pos0[] = { 1.0F,0.0F,2.0F,1.0F };  
    GLfloat l_dir0[] = { -1.0F,0.0F,-2.0F,1.0F };  
  
    GLfloat l_pos1[] = { -1.0F,0.0F,2.0F,1.0F };  
    GLfloat l_dir1[] = { 1.0F,0.0F,-2.0F,1.0F };
```

```
    glLightfv(GL_LIGHT0,GL_DIFFUSE,rouge);  
    glLightfv(GL_LIGHT0,GL_POSITION,l_pos0);  
    glLightfv(GL_LIGHT0,GL_SPOT_DIRECTION,l_dir0);  
    glLightf(GL_LIGHT0,GL_SPOT_CUTOFF,o0);  
  
    glLightfv(GL_LIGHT1,GL_DIFFUSE,vert);  
    glLightfv(GL_LIGHT1,GL_POSITION,l_pos1);  
    glLightfv(GL_LIGHT1,GL_SPOT_DIRECTION,l_dir1);  
    glLightf(GL_LIGHT1,GL_SPOT_CUTOFF,o1);  
  
    glEnable(GL_LIGHTING);  
    glEnable(GL_LIGHT0);  
    glEnable(GL_LIGHT1);  
}
```



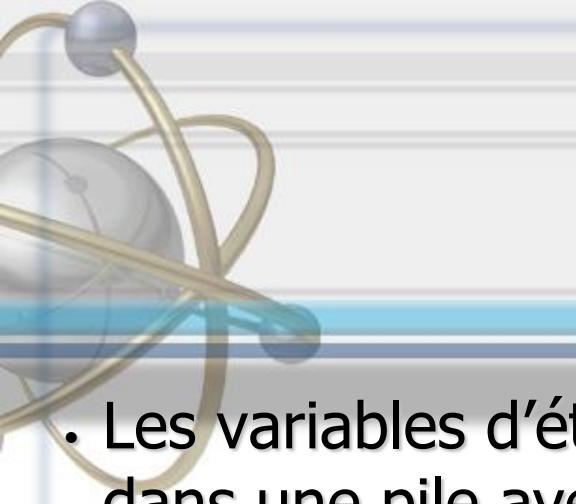


# Sources lumineuses

- Paramètres généraux
  - Non spécifiques à chaque source

```
void glLightModel{if }[v] (GLenum proprie, TYPE valeurs);
```

- **Lumière ambiante globale** : Spécifier l'intensité de la lumière ambiante globale
- « proprie » : GL\_LIGHT\_MODEL\_AMBIENT
- « valeurs » : composantes RGBA
  - Défaut : (0.2,0.2,0.2,1.0)
- **Gestion des faces arrières** : Choisir entre l'éclairage d'un seul côté ou des deux côtés
- « proprie » : GL\_LIGHT\_MODEL\_TWO\_SIDE
- « valeurs » : GL\_TRUE, GL\_FALSE
  - Défaut : GL\_FALSE
- **Calcul des reflets spéculaires** : Spécifie comment les angles pour la réflexion spéculaire seront calculés.
- « proprie » : GL\_LIGHT\_MODEL\_LOCAL\_VIEWER
- « valeurs » : GL\_TRUE, GL\_FALSE
  - Défaut : GL\_FALSE

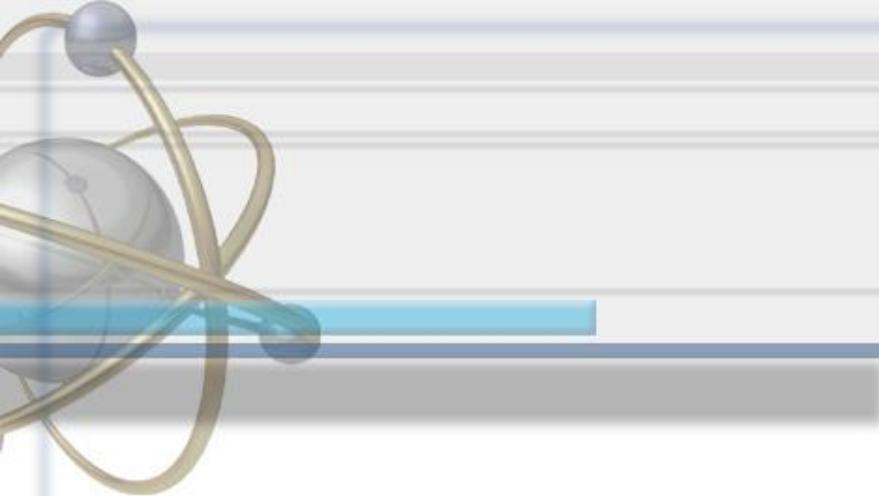


# Pile d'Attributs

- Les variables d'états peuvent être sauvegardées et restaurées dans une pile avec les commandes

**glPushAttrib(mask) et glPopAttrib(mask)**

- Dans le cas des sources d'éclairage :
  - mask : GL\_LIGHTING\_BIT
- GL\_LIGHTING\_BIT fait référence aux variables d'états concernant l'éclairage : couleur des matériaux, intensité émise, ambienne, diffuse et spéculaire de la lumière, une liste de sources actives, la direction des spots lumineux.

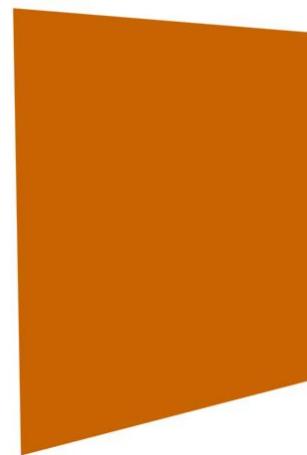


# Les matériaux



# Matériaux

- Sous OpenGL pour définir le comportement d'un objet vis-à-vis d'une couleur RVB, il suffit de dire quel pourcentage de chaque composante est réfléchi
  - un matériau avec les pourcentages
$$(R=100\%, V=50\%, B=0\%) \Leftrightarrow (R=1, V=0.5, B=0)$$
  - éclairé avec une lumière blanche
$$(R=1, V=1, B=1)$$
  - réfléchi un rayon de couleur
$$(R=1, V=0.5, B=0) : \text{orange}$$
  - il absorbe le reste (théoriquement)
$$(R=0, V=0.5, B=1)$$





# Matériaux

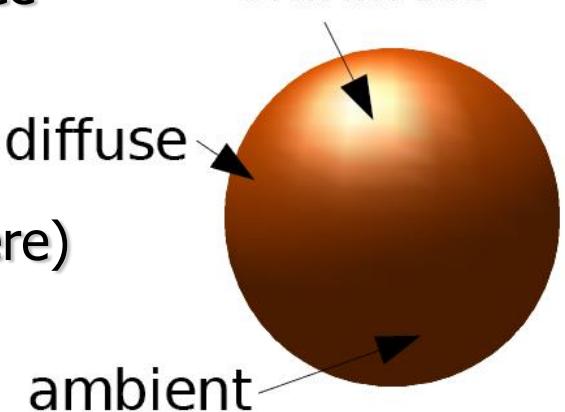
- un matériau avec les pourcentages  
 $(R=100\%, V=50\%, B=0\%) \Leftrightarrow (R=1, V=0.5, B=0)$
- éclairé avec une lumière bleu  
 $(R=0, V=0, B=1)$
- réfléchi un rayon de couleur  
 $(R=0, V=0, B=0)$  : orange
- il absorbe tout (théoriquement)  
 $(R=1, V=1, B=1)$  : absence de lumière.
- Un matériau
  - pourcentage réfléchi de chaque composante de couleur

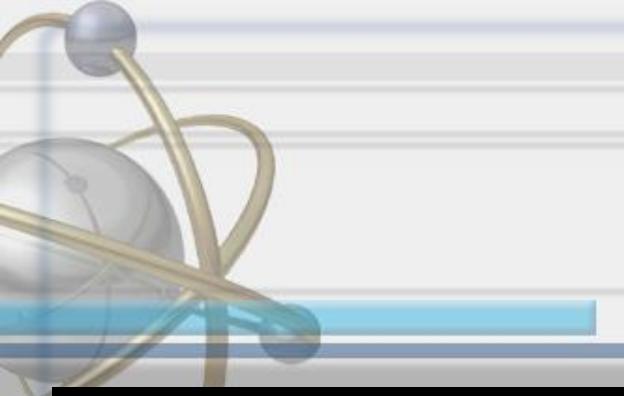




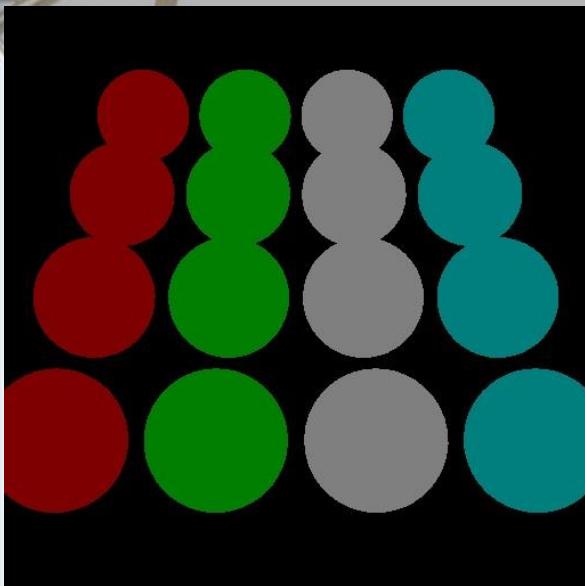
# Matériaux

- La spécification d'un matériau pour un objet se fait par l'intermédiaire de 5 paramètres :
  - La couleur diffuse : couleur "de base"
  - La couleur ambiante : couleur des zones non éclairées directement
  - La couleur émise : couleur émise par le matériau
  - La couleur spéculaire : couleur des reflets spéculaires
  - Le coefficient de brillance : coefficient de brillance specular/shininess
- Combinaison lumières & matériaux
  - ambient = (ambient Materiau) \* (ambient Lumière)
  - diffuse = (diffuse Materiau) \* (diffuse Lumière)
  - ...

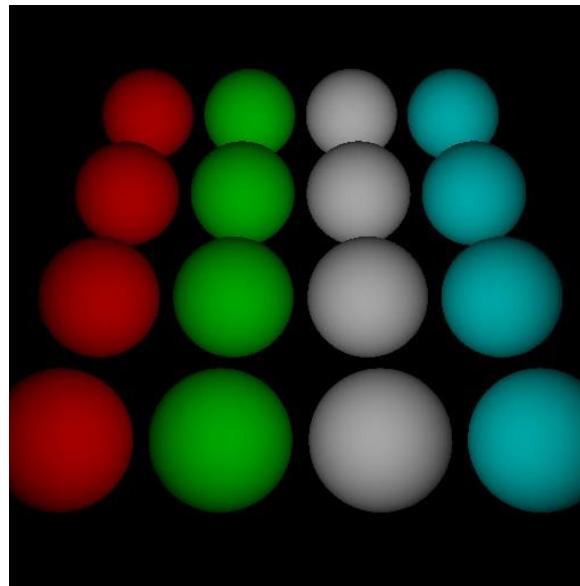




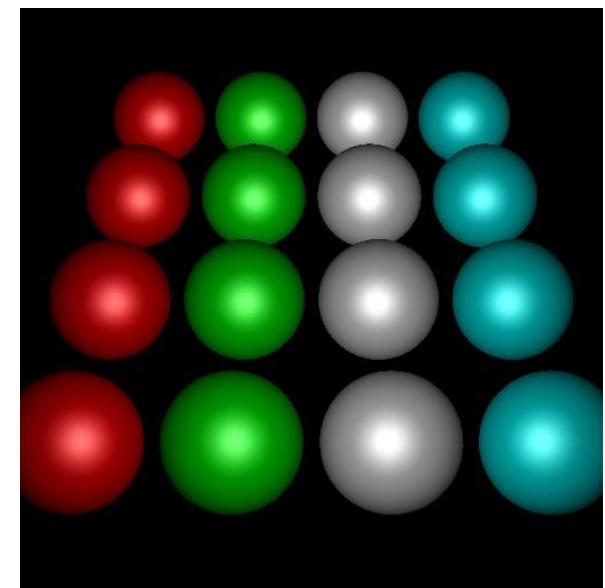
# Matériaux



Composante ambiante



Composante diffuse



Composante spéculaire

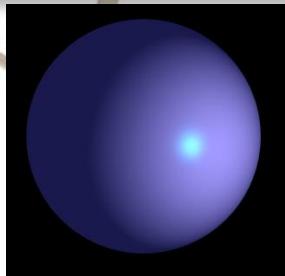


# Matériaux

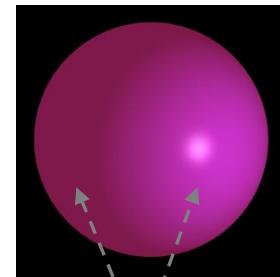
```
void glMaterial{if}[v](GLenum face, GLenum proprieté, TYPE[*] valeur[s])
```

- « face » : faces où la matière doit être appliquée : face avant, face arrière ou les deux côtés de la face.
  - GL\_FRONT
  - GL\_BACK
  - GL\_FRONT\_AND\_BACK
- « propriété » : composantes du matériau
  - GL\_AMBIENT
    - Défaut : (0.2, 0.2, 0.2, 1.0)
  - GL\_DIFFUSE
    - Défaut : (1.0, 1.0, 1.0, 1.0)
  - GL\_SPECULAR
    - Défaut : (0.0, 0.0, 0.0, 1.0)
  - GL\_EMISSION
    - Défaut : (0.0, 0.0, 0.0, 1.0)
  - GL\_SHININESS:
    - Défaut : 0.0 => [0, 128]
- « valeur[s] » : valeur (scalaire ou vecteur (R, G, B & A)) de la propriété « propriété » manipulée

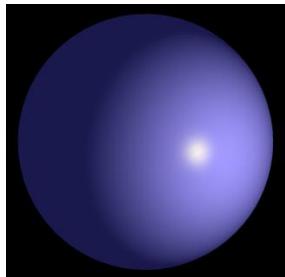
# Matériaux



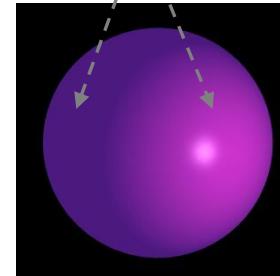
GL\_DIFFUSE : 0.0,0.0,0.9,1.0  
GL\_AMBIENT : 0.1,0.1,0.3,1.0  
GL\_SPECULAR : 0.0 ,0.5,0.5,1.0  
GL\_SHININESS : 100



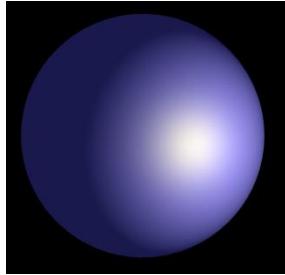
GL\_DIFFUSE : 0.3,0.1,0.5,1.0  
GL\_AMBIENT : 0.5,0.1,0.3,1.0



GL\_SPECULAR : 0.4 ,0.4,0.4,1.0  
GL\_SHININESS : 100



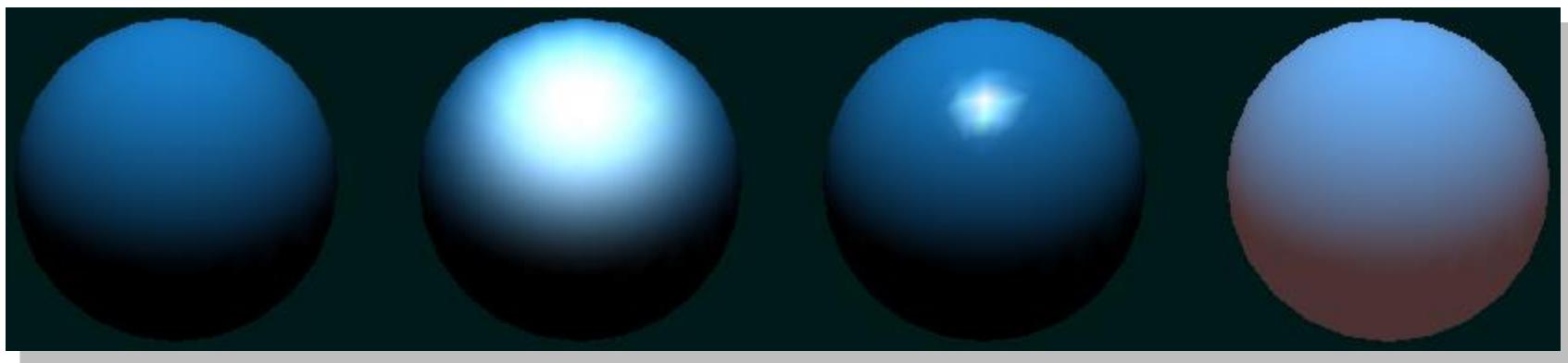
GL\_DIFFUSE : 0.5,0.1,0.3,1.0  
GL\_AMBIENT : 0.3,0.1,0.5,1.0

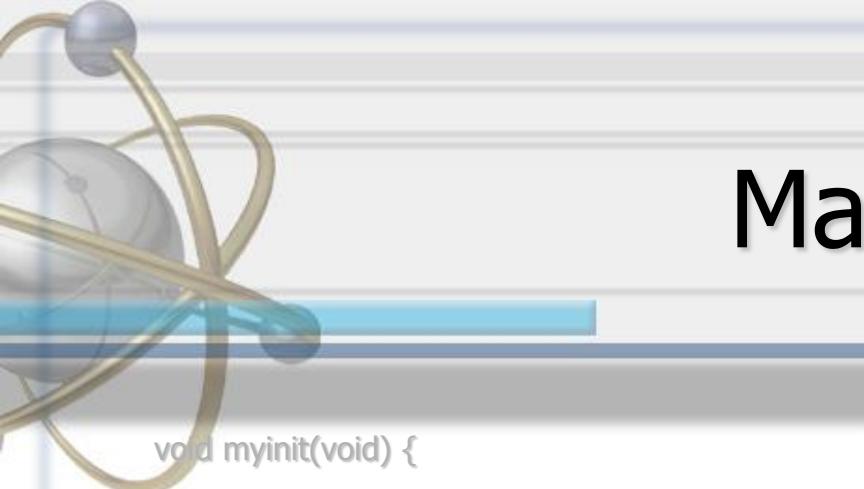


GL\_SPECULAR : 0.4 ,0.4,0.4,1.0  
GL\_SHININESS : 10



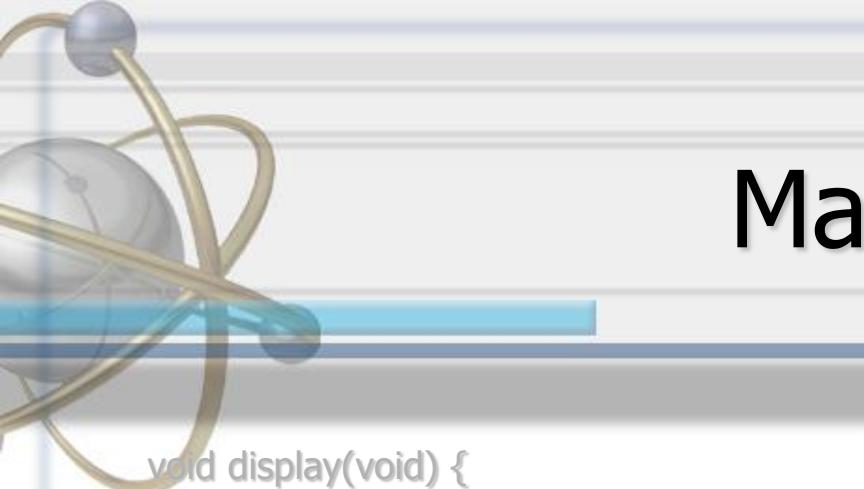
# Matériaux





# Matériaux

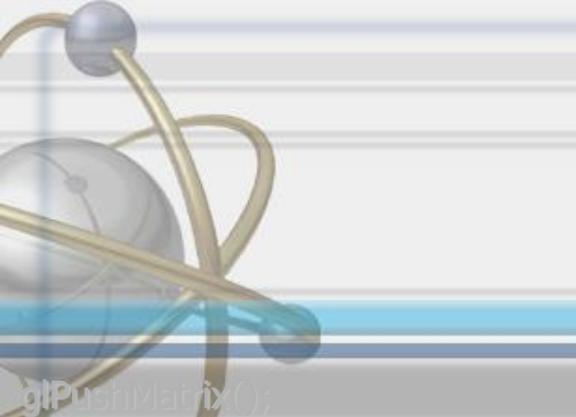
```
void myinit(void) {  
    GLfloat noir[] = { 0.0F,0.0F,0.0F,1.0F };  
    GLfloat blanc[] = { 1.0F,1.0F,1.0F,1.0F };  
  
    GLfloat grismoyen[] = { 0.5F,0.5F,0.5F,1.0F };  
  
    GLfloat position[] = {0.0F,3.0F,2.0F,0.0F};  
    GLfloat local_view[] = {0.0F};  
  
    ...  
    glEnable(GL_DEPTH_TEST);  
    glDepthFunc(GL_LESS);  
  
    glLightfv(GL_LIGHT0,GL_AMBIENT,noir);  
    glLightfv(GL_LIGHT0,GL_DIFFUSE,blanc);  
    glLightfv(GL_LIGHT0,GL_POSITION,position);  
  
    glLightModelfv(GL_LIGHT_MODEL_AMBIENT, grismoyen);  
    glLightModelfv(GL_LIGHT_MODEL_LOCAL_VIEWER,local_view);  
  
    glEnable(GL_LIGHTING);  
    glEnable(GL_LIGHT0);  
}
```



# Matériaux

```
void display(void) {  
  
    GLfloat no_mat[] = { 0.0F,0.0F,0.0F,1.0F };  
    GLfloat mat_ambient[] = { 0.7F,0.7F,0.7F,1.0F };  
    GLfloat mat_ambient_color[] = { 0.8F,0.8F,0.2F,1.0F };  
    GLfloat mat_diffuse[] = { 0.1F,0.5F,0.8F,1.0F };  
    GLfloat mat_specular[] = { 1.0F,1.0F,1.0F,1.0F };  
    GLfloat no_shininess[] = { 0.0F };  
    GLfloat low_shininess[] = { 5.0F };  
    GLfloat high_shininess[] = { 100.0F };  
    GLfloat mat_emission[] = {0.3F,0.2F,0.2F,0.0F};
```

n<sub>1</sub> n<sub>2</sub> n<sub>3</sub>



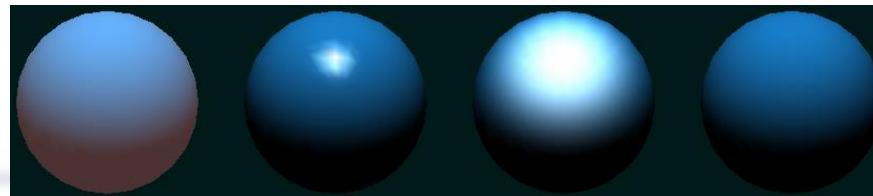
# Matériaux

```
glPushMatrix();
glTranslatef(-3.75F,3.0F,0.0F);
glMaterialfv(GL_FRONT,GL_AMBIENT,no_mat);
glMaterialfv(GL_FRONT,GL_DIFFUSE,mat_diffuse);
glMaterialfv(GL_FRONT,GL_SPECULAR,no_mat);
glMaterialfv(GL_FRONT,GL_SHININESS,no_shininess);
glMaterialfv(GL_FRONT,GL_EMISSION,no_mat);
Sphere();
glPopMatrix();
```

```
glPushMatrix();
glTranslatef(-1.25F,3.0F,0.0F);
glMaterialfv(GL_FRONT,GL_AMBIENT,no_mat);
glMaterialfv(GL_FRONT,GL_DIFFUSE,mat_diffuse);
glMaterialfv(GL_FRONT,GL_SPECULAR,mat_specular);
glMaterialfv(GL_FRONT,GL_SHININESS,low_shininess);
glMaterialfv(GL_FRONT,GL_EMISSION,no_mat);
Sphere();
glPopMatrix();
```

```
glPushMatrix();
glTranslatef(1.25F,3.0F,0.0F);
glMaterialfv(GL_FRONT,GL_AMBIENT,no_mat);
glMaterialfv(GL_FRONT,GL_DIFFUSE,mat_diffuse);
glMaterialfv(GL_FRONT,GL_SPECULAR,mat_specular);
glMaterialfv(GL_FRONT,GL_SHININESS,high_shininess);
glMaterialfv(GL_FRONT,GL_EMISSION,no_mat);
Sphere();
glPopMatrix();
```

```
glPushMatrix();
glTranslatef(3.75F,3.0F,0.0F);
glMaterialfv(GL_FRONT,GL_AMBIENT,no_mat);
glMaterialfv(GL_FRONT,GL_DIFFUSE,mat_diffuse);
glMaterialfv(GL_FRONT,GL_SPECULAR,no_mat);
glMaterialfv(GL_FRONT,GL_SHININESS,no_shininess);
glMaterialfv(GL_FRONT,GL_EMISSION,mat_emission);
Sphere();
glPopMatrix();
```





# La gestion d'événements



# La gestion d'événements

- GLUT propose des fonctions pour gérer les événements
- Évènements :
  - Fenêtre de dessin
  - Fenêtre de redimensionnement
  - Interaction avec la scène
    - Clavier
    - Souris
  - animation
- La gestion se fait au moyen de fonction callbacks
  - routines de rappelles



# La gestion d'événements

Fenêtre redessinée

```
void glutDisplayFunc(void (*displayFunc)(void));
```

- Définit la fonction exécutée automatiquement par GLUT quand un événement intervient entraînant le rafraîchissement de l'image affichée (création initiale de la fenêtre d'affichage, déplacement, agrandissement, réduction, réactivation en avant-plan, ordre par programme, ...).
- La fonction `display` passée en paramètre ne doit assurer l'affichage que d'une seule image. Son prototype doit obligatoirement correspondre à

```
void (*display)(void).
```



# La gestion d'événements

- Gestion des redimensionnements fenêtre

```
void glutReshapeFunc(void (*fonct)(int larg,int haut));
```

- Définit la fonction exécutée automatiquement par GLUT avant le rafraîchissement (lui aussi automatique) d'une fenêtre dont la taille est modifiée.
- Cette fonction "reshape" est habituellement employée pour configurer deux caractéristiques liées à la visualisation :
  - la zone de la fenêtre d'affichage dans laquelle l'image est affichée,
  - le mode de représentation (projection parallèle ou en perspective) et ses paramètres,
  - et éventuellement pour amorcer le dessin de la scène en fixant un point de vue. Elle doit obligatoirement avoir en entête deux paramètres de type entier qui représentent les tailles en x et en y de la fenêtre après changement de dimension.
- Ces paramètres permettront éventuellement à la fonction reshape d'adapter son action aux nouvelles dimensions de la fenêtre.



# La gestion d'événements

```
void glutIdleFunc(void *func(void))
```

- Définit une fonction à exécuter par GLUT s'il n'y a pas d'événement en attente
  - Réalisation d'une tâche annexe : tâche de fond.
- Fonction fréquemment utilisée pour programmer des animations.
  - Généralement dédiée à la modification de variables globales employées dans la fonction display
  - ne contient pas d'appel de fonction OpenGL.
- Il faut demander l'affichage d'une nouvelle image via :
  - glutPostRedisplay ou glutPostWindowRedisplay



# La gestion d'événements

```
double T = 0;  
  
int main(int argc, char **argv){  
    ...  
    glutIdleFunc(idle);  
    ...  
}  
  
void idle(void){  
    T += 0.05;  
    glutPostRedisplay();  
}  
  
void display(void) {  
    ...  
    Translatef(0.0f,T,0.0f);  
    DrawObject();  
    ...  
}
```



# La gestion d'événements

- Gestion des événements clavier (1)

```
void glutKeyboardFunc(void (*fonct) (unsigned char key,int x,int y))
```

- Définit la fonction exécutée automatiquement par GLUT lorsqu'une touche ASCII du clavier est frappée.
- Cette fonction recevra les paramètres
  - Key : le code ASCII de la touche de clavier
  - x, y : les positions instantanées de la souris relativement à la fenêtre au moment de la frappe clavier.



# La gestion d'événements

- Gestion des événements clavier (2)

```
void glutSpecialFunc(void (*fonct)(int key,int x,int y))
```

- Définit la fonction exécutée automatiquement par GLUT lorsqu'une touche de fonction ou une touche de curseur du clavier est frappée.
- Cette fonction recevra les paramètres
  - Key : code de la touche de clavier (GLUT\_KEY\_F1 à GLUT\_KEY\_F12, GLUT\_KEY\_LEFT, GLUT\_KEY\_RIGHT, GLUT\_KEY\_UP, GLUT\_KEY\_DOWN, GLUT\_KEY\_PAGE\_DOWN, GLUT\_KEY\_PAGE\_UP, GLUT\_KEY\_HOME, GLUT\_KEY\_END, GLUT\_KEY\_INSERT)
  - x, y : positions instantanées en x et en y de la souris relativement à la fenêtre.



# La gestion d'événements

```
double rot_x;

int main(int argc, char **argv){
...
glutKeyboardFunc(keyboard);
...
}

void Keyboard(unsigned char key, int x, int y){
    Switch (key)
    {
        Case 'A':
            rot_x +=0.1;
            glutPostRedisplay();

            break();
        Case 'Z':
            rot_x -=0.1;
            glutPostRedisplay();
            break();
    }
}

void display(void) {
...
    glRotatef(rot_x,1.0,0.0,0.0);
    DrawObject();
...
}
```



# La gestion d'événements

- Gestion des événements souris (1)

```
void glutMouseFunc(void (*fonct)(int bouton,int etat,int x,int y));
```

- fonction exécutée automatiquement par GLUT quand les boutons de la souris sont utilisés.
- paramètres retournés par la fonction :
  - bouton : GLUT\_LEFT\_BUTTON, GLUT\_MIDDLE\_BUTTON ou GLUT\_RIGHT\_BUTTON
  - etat : GLUT\_UP ou GLUT\_DOWN
  - x,y : position de la souris dans la fenêtre au moment de l'événement



# La gestion d'événements

- Gestion des événements souris (2)

```
void glutMotionFunc(void (*fonct)(int x,int y));
```

- Fonction exécutée automatiquement par GLUT quand la souris se déplace devant la fenêtre avec un bouton appuyé.
- paramètres retournés par la fonction :
  - x, y : positions de la souris dans la fenêtre au moment de l'appel de la fonction.

```
void glutPassiveMotionFunc(void (*fonct)(int x,int y));
```

- Fonction exécutée automatiquement par GLUT quand la souris se déplace devant la fenêtre sans bouton appuyé.
- paramètres retournés par la fonction :
  - x, y : positions de la souris dans la fenêtre au moment de l'appel de la fonction.



# La gestion d'événements

```
double x_inc, y_inc;

int main(int argc, char **argv){
    ...
    glutPassiveMotionFunc(MousePassive);
    ...
}

void MousePassive(int x,int y){
    x_inc = x;
    y_inc = y;
    glutPostRedisplay();
}

void display(void) {
    ...
    Translatef(x_inc, y_inc,0.0f);
    DrawObject();
    ...
}
```