Synthèse d'images et animation
OpenGL - Cours 1

Estelle Duveau - estelle.duveau@inria.fr

3A IRV, 24 septembre 2008

Planning

- **Cours :** introduction à OpenGL - 24/09/2008 (1h30)
- **Cours :** OpenGL avancé - 01/10/2008 (1h30)
- **TP1 :** découverte et affichage - 08/10/2008 (1h30)
- **TP2 :** découverte et affichage - 15/10/2008 (1h30)
- **TP3 :** utilisation des textures - 05/11/2008 (1h30)
- **TP4 :** animation de personnages + **TP5 :** modèles physiques de base - 19/11/2008 (3h)
- **TP6 :** modèles physiques avancés - 26/11/2008 (1h30)
- **Projet** - semaine du 1er décembre (2 jours)

Plan

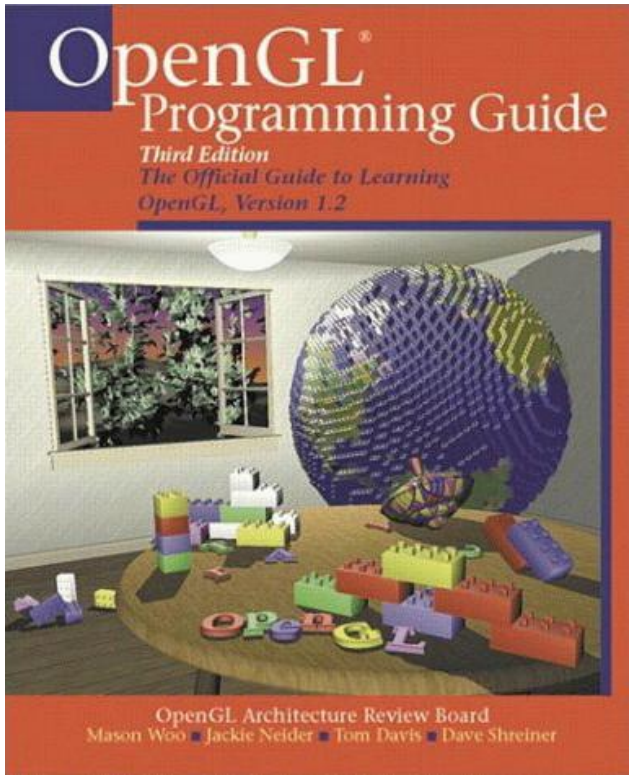
Références

D. Shreiner, M. Woo, J. Neider, T. Davis

OpenGL Programming Guide

alias le red book

http://opengl-redbook.com



Plan

C'est quoi?

- Une API (Application Programming Interface) pour l'architecture graphique
- Indépendant de l'architecture
- Développée en 1989 (GL) par Silicon Graphics, portée sur d'autres architectures en 1993 (OpenGL)
- Comprend environ 250 commandes

OpenGL ne fait pas tout

- Pas de commande pour créer ou gérer un *viewer*
- Pas de commande haut-niveau pour gérer les objets : seulement 3 types de primitives géométriques (points, lignes, polygones)

- $\Rightarrow$ **besoin de bibliothèques complémentaires :**
  - **GLU :** *openGL Utility library* routines de base, création de modèles 3D un peu plus compliqués
  - **GLUT :** *openGL Utility Toolkit* gestion de la fenêtre

Exemple d'utilisation de GLUT

```
int main(int argc, char** argv){
   glutInit(&argc,argv);
   glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
   glutInitWindowSize(250,250);
   glutInitWindowPosition(100,100);
   glutCreateWindow("hello"); // pas encore affichée
   init(); // à définir avec OpenGL
   glutDisplayFunc(display); // exécute display
                             // (à définir, OpenGL)
   glutMainLoop; // C'est parti ! Fenêtre affichée
   return 0; // Langage C
}
```

Pipeline graphique

Pipeline de base :

1. Construction des formes à partir de primitives géométriques (avant Modeling transformations)
2. Arrangement des objets dans l'espace 3D et sélection du point de vue (Modeling transformations + Viewing transformation)
3. Calcul des couleurs des objets (lumière, texture...) (Illumination)
4. Rasterization : conversion en image 2D (Scan conversion)

Autres opérations possibles : élimination des parties cachées, opérations sur les pixels...

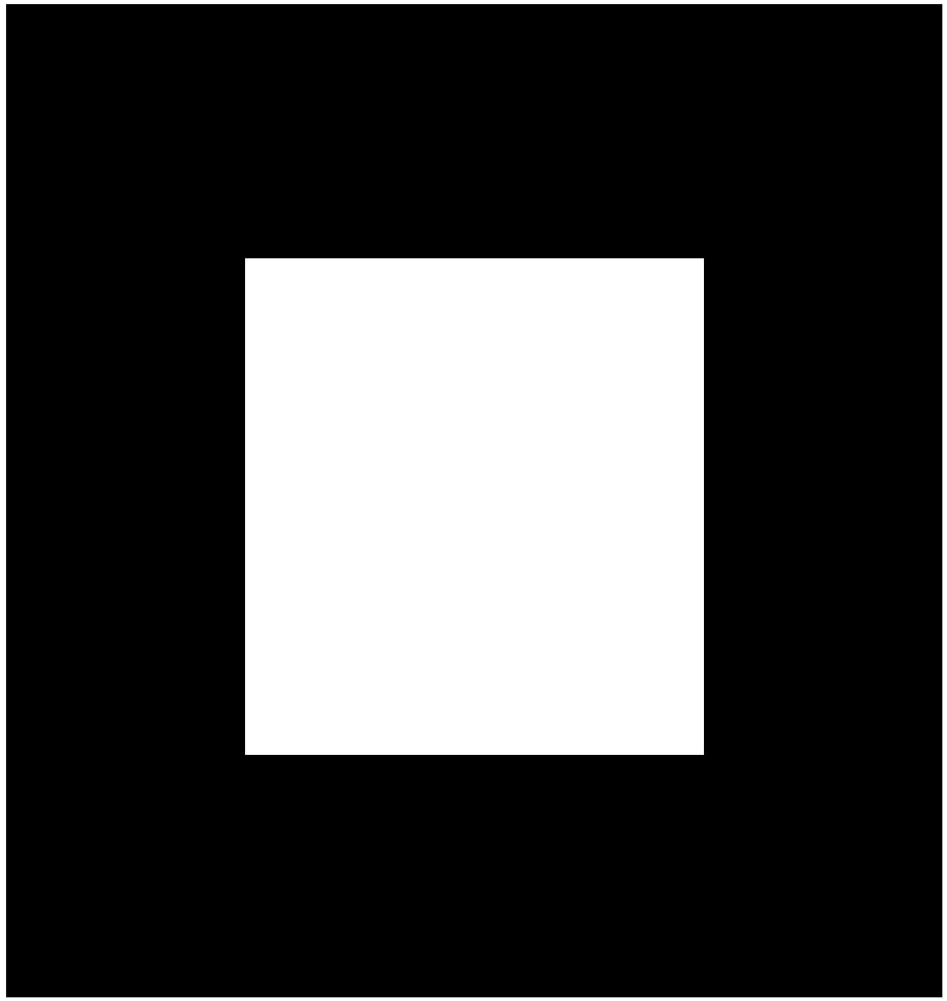
Rappel du vocabulaire

- **Rendu :** processus de création d'images 2D à partir de modèles 3D
- **Modèles :** construits à partir de primitives géométriques définies par leurs sommets (*vertex/vertices*)
- **Bitplane :** zone de la mémoire contenant un bit d'information pour chaque pixel de l'image
- **Framebuffer :** zone de la mémoire réunissant l'ensemble des bitplanes

Exemple de base

```
# include <WhateverYouNeed.h>
main () {
   InitializeAWindowPlease(); // Pas OpenGL
   glClearColor(0.0,0.0,0.0,0.0); // Couleur d'init.
   glClear(GL_COLOR_BUFFER_BIT); // Initialiser la couleur
   glColor3f(1.0,1.0,1.0); // Couleur de dessin
   glOrtho(0.0,1.0,0.0,1.0,-1.0,1.0); // Syst. de coord.
   glBegin(GL_POLYGON); // Dessinons un polygone
      glVertex3f(0.25,0.25,0.0);
      glVertex3f(0.75,0.25,0.0); // Coordonnées des
      glVertex3f(0.75,0.75,0.0); // sommets du polygone
      glVertex3f(0.25,0.75,0.0);
   glEnd();
   glFlush(); // Exécute les commandes tout de suite
   UpdateTheWindowAndCheckForEvents(); // Pas OpenGL
}
```

Résultat



Eléments de syntaxe OpenGL

```
glColor3f(1.0,1.0,1.0);
```

- **gl** : commande OpenGL
  (les constantes OpenGL commencent par **GL_** : `GL_COLOR_BUFFER_BIT`)
- **3** : cette commande a 3 arguments
- **f** : les arguments sont des flottants

```
glColor3fv(color_array);
```

⇒ Le paramètre est un **v**ecteur (ou tableau) de 3 flottants
(`GLfloat color_array[] = { 1.0,0.0,0.0 } ;`)

Suffixes et types OpenGL

b	entier (8 bits)	signed char	GLbyte
s	entier (16 bits)	short	GLshort
i	entier (32 bits)	int ou long	GLint
f	flottant (32 bits)	float	GLfloat
d	flottant (64 bits)	double	GLdouble
ub	entier non signé (8 bits)	unsigned char	GLubyte
us	entier non signé (16 bits)	unsigned long	GLushort
ul	entier non signé (32 bits)	unsigned int ou long	GLuint

Machine à états

Etats/modes actifs jusqu'à ce qu'on les change
Exemples :

- couleur courante
- point de vue
- mode de dessin des polygones
- position et caractéristiques des sources de lumières
- ...

Activation/Désactivation :

- `glEnable(GL_LIGHTING);`
- `glDisable(GL_COLOR_MATERIAL);`

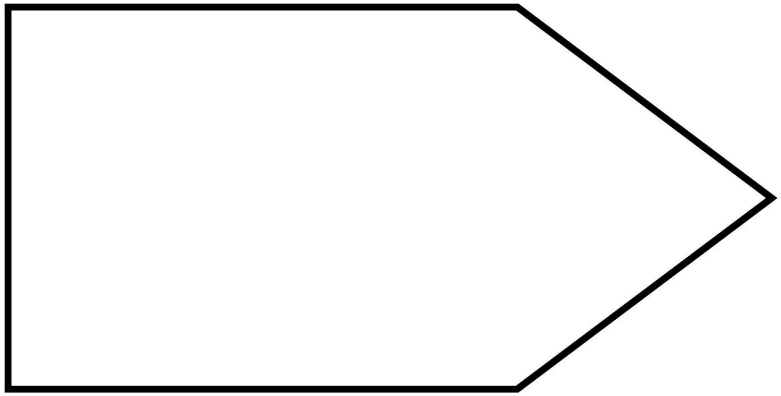
Plan

Primitives de base

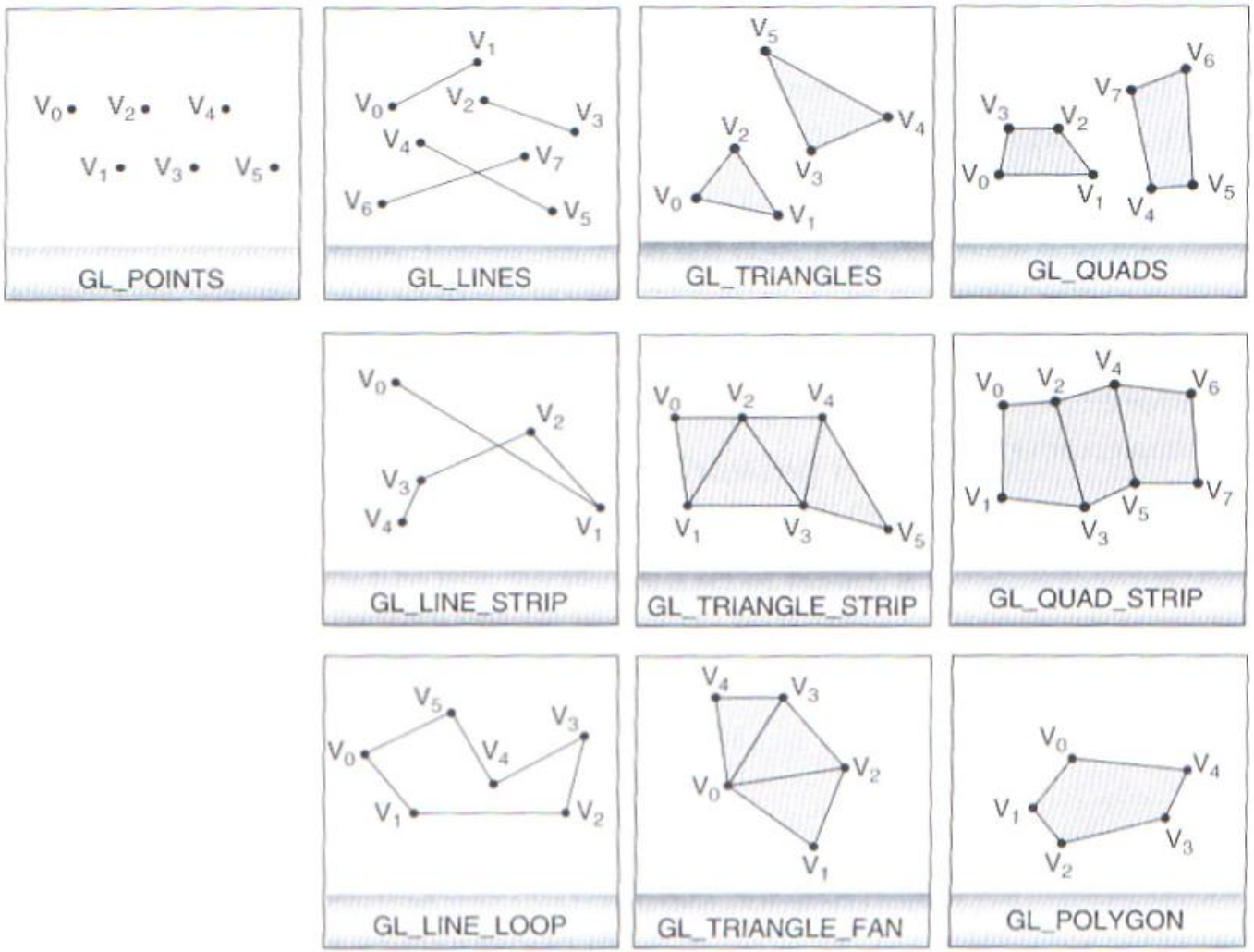
1. Sommets (vertices) : vecteur de flottants
2. Lignes : segments
3. Polygones : polygones convexes simples

Exemple : pentagone plan

```
glBegin(GL_POLYGON);
   glVertex2f(0.0, 0.0);
   glVertex2f(0.0, 3.0);
   glVertex2f(4.0, 3.0);
   glVertex2f(6.0, 1.5);
   glVertex2f(4.0, 0.0);
glEnd();
```



Ensemble des primitives géométriques



Paramètres supplémentaires

- Taille des sommets (en pixels) : `glPointSize(2.0);`
- Epaisseur des lignes (en pixels) : `glLineWidth(3.0);`
- Connaître valeurs courantes : `glGetFloatv(GL_LINE_WIDTH);`
- Dessin des lignes : on peut préciser de nombreux motifs de pointillés
- Le rendu des faces avant et arrière d'un polygone peut être différent :`glPolygonMode(GL_FRONT,GL_FILL);` `glPolygonMode(GL_BACK,GL_LINE);`
- Culling : `glCullFace(GL_BACK);` : l'arrière des faces n'est pas visible
- On peut préciser la couleur, la normale aux sommets, ...

Les normales

```
glBegin(GL_POLYGON);
   glNormal3fv(n0);
   glVertex3fv(v0);
   glNormal3fv(n1);
   glVertex3fv(v1);
   glNormal3fv(n2);
   glVertex3fv(v2);
glEnd();
```

Attention à l'ordre : normale avant coordonnées

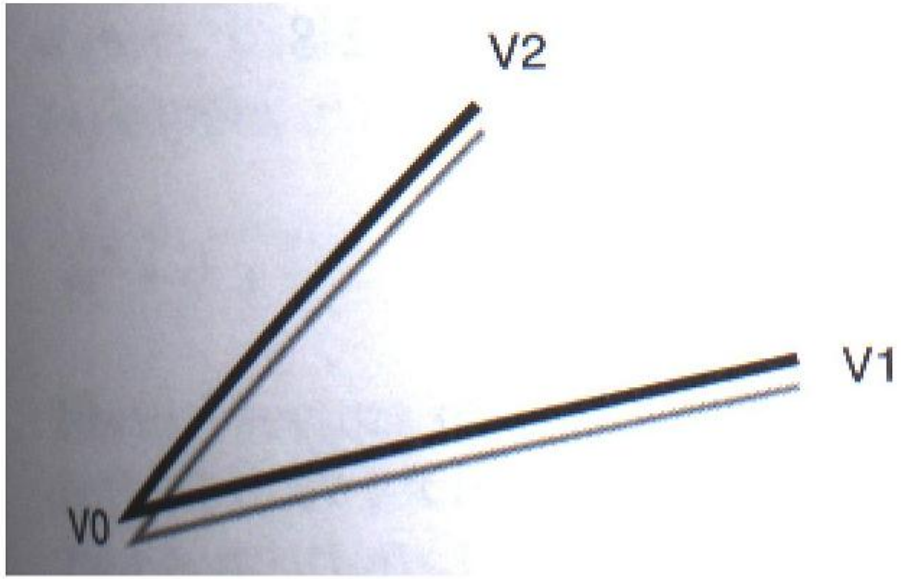
Comment ne tracer qu'une partie d'un contour

$\longrightarrow$ très utile pour tracer un polygone non convexe

```
glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);
glBegin(GL_POLYGON);
   glEdgeFlag(GL_TRUE);
   gVertex3fv(v0);
   glEdgeFlag(GL_FALSE);
   gVertex3fv(v1);
   glEdgeFlag(GL_TRUE);
   gVertex3fv(v2);
glEnd();
```

$\Rightarrow$ le sommet suivant est/n'est pas le début d'un bord

Résultat



Tableaux de sommets

- Un tableau par type de données : coordonnées, normale, couleur, texture, ...
- Création d'un tableau :

```
glEnableClientState(GL_NORMAL_ARRAY);
```

- Utilisation d'un tableau :

```
glColorPointer(3,GL_FLOAT,5*sizeof(GLfloat),color);
```

$\longrightarrow$ *stride* = offset (en octets) entre 2 données consécutives

- Accès à un élément `glArrayElement(0);` se fait dans tous les tableaux activés
- Accès à plusieurs éléments :

```
glDrawElements(GL_POLYGON,5,GL_UNSIGNED_INT,vertices);
```

- Aussi `glMultiDrawElements(...)`, `glDrawRangeElements(...)` , etc.

Exemple 1/2

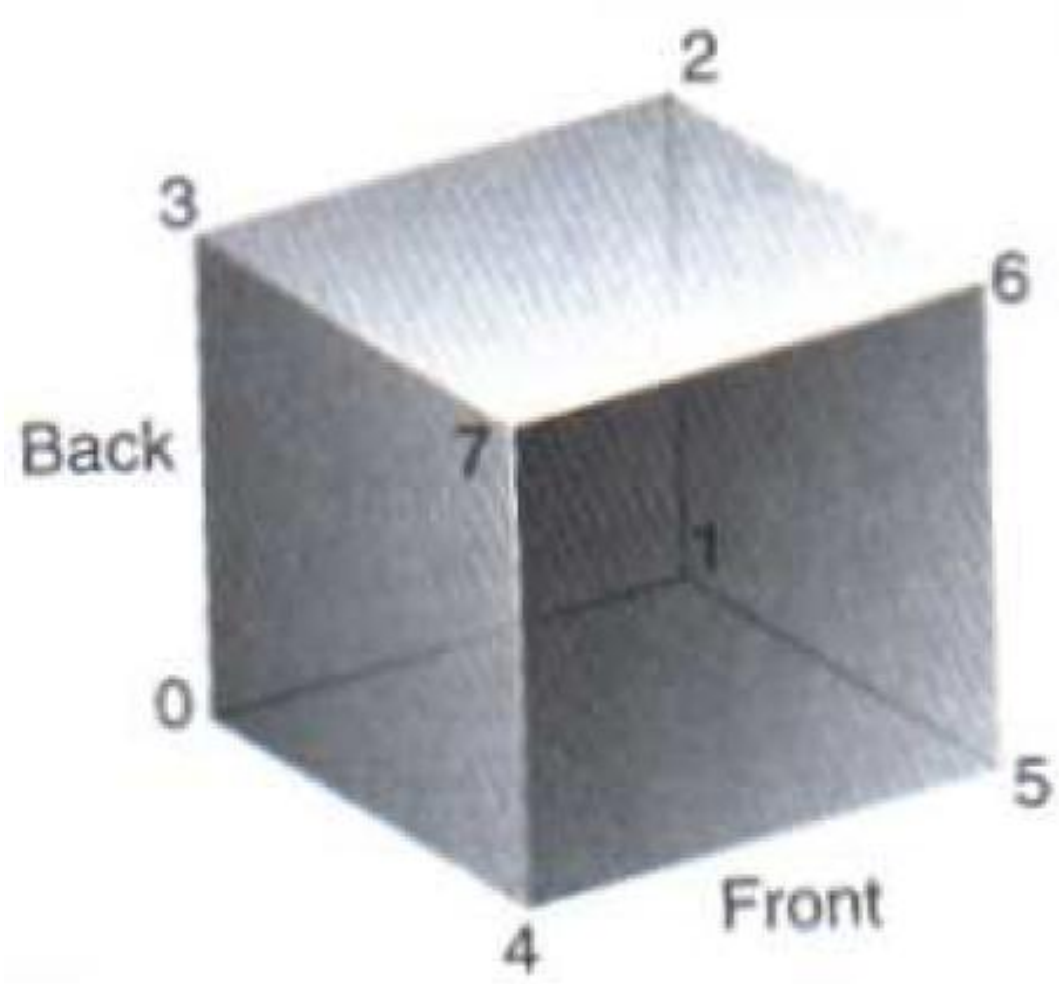
```
static GLint vertices[] = {
   25, 25, 100, 325, 175, 25,
   175, 325, 250, 25, 325, 325
};
static GLfloat colors[] = {
   1.0, 0.2, 0.2, 0.2, 0.2, 1.0,
   0.8, 1.0, 0.2, 0.75, 0.75, 0.75,
   0.35, 0.35, 0.35, 0.5, 0.5, 0.5
};
glEnableClientState(GL_COLOR_ARRAY);
glEnableClientState(GL_VERTEX_ARRAY);
glColorPointer(3, GL_FLOAT, 0, colors);
glVertexPointer(2, GL_INT, 0, vertices);
```

Exemple 2/2

```
// Création d'un triangle à partir de trois sommets quelconques
glBegin(GL_TRIANGLES);
glArrayElement(2);
glArrayElement(3);
glArrayElement(5);
glEnd();

// Création d'un triangle à partir des trois premiers sommets
glDrawElements(
   GL_TRIANGLES,3,GL_UNSIGNED_SHORT,vertices);
```

Un exemple concret : dessiner un cube



Un exemple concret : dessiner un cube

```
static GLubyte frontI[] = { 4, 5, 6, 7 };
static GLubyte rightI[] = { 1, 2, 6, 5 };
static GLubyte bottomI[] = { 0, 1, 5, 4 };
static GLubyte backI[] = { 0, 3, 2, 1 };
static GLubyte leftI[] = { 0, 4, 7, 3 };
static GLubyte topI[] = { 2, 3, 7, 6 };

glDrawElements(GL_QUADS,4,GL_UNSIGNED_BYTE,frontI);
glDrawElements(GL_QUADS,4,GL_UNSIGNED_BYTE,rightI);
glDrawElements(GL_QUADS,4,GL_UNSIGNED_BYTE,bottomI);
glDrawElements(GL_QUADS,4,GL_UNSIGNED_BYTE,backI);
glDrawElements(GL_QUADS,4,GL_UNSIGNED_BYTE,leftI);
glDrawElements(GL_QUADS,4,GL_UNSIGNED_BYTE,topI);
```

Plan

Voir en 3D

- Modeling transformations :

  repère objet $\longrightarrow$ repère monde

- Viewing transformations :

  repère monde $\longrightarrow$ repère caméra

- Projection sur l'écran

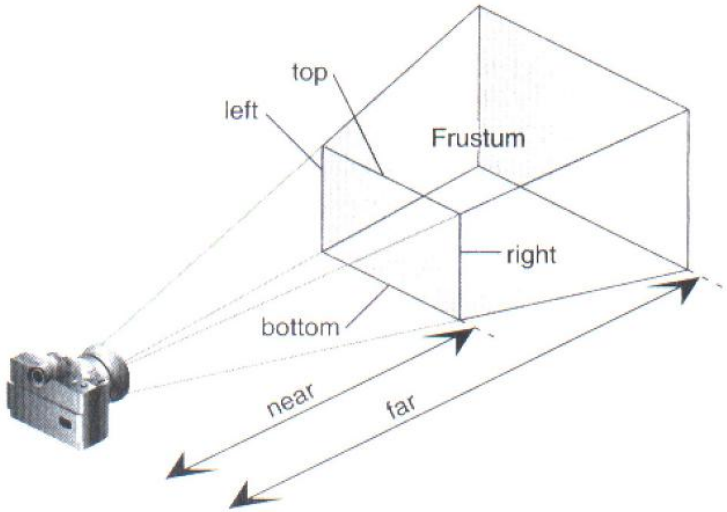
$\longrightarrow$ matrices 4x4

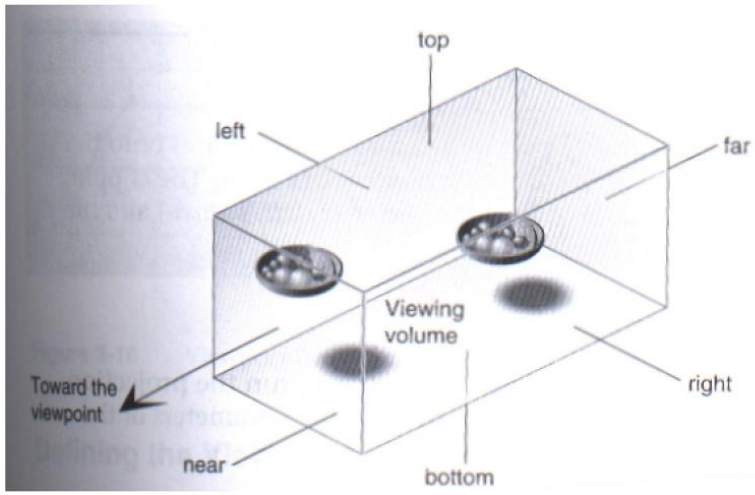
Manipuler les matrices 4x4 sous OpenGL

- `glMatrixMode(GL_MODELVIEW)` ou `GL_PROJECTION` ou `GL_TEXTURE`
  $\longrightarrow$ Matrice courante : en général la matrice MODELVIEW, sauf lorsqu'on doit modifier les paramètres intrinsèques de la caméra
- `glLoadIdentity()`
- `glLoadMatrixf(M)`, `glLoadTransposeMatrixd(N)`
- `glMultMatrixd(P)`, `glMultTransposeMatrixf(Q)`
- `glTranslatef(1.0,2.0,-1.5)`
- `glRotated(90.0,0.0,1.0,0.0)`
- `glScalef(2.0,-0.5,1.0)`

Manipuler la matrice PROJECTION

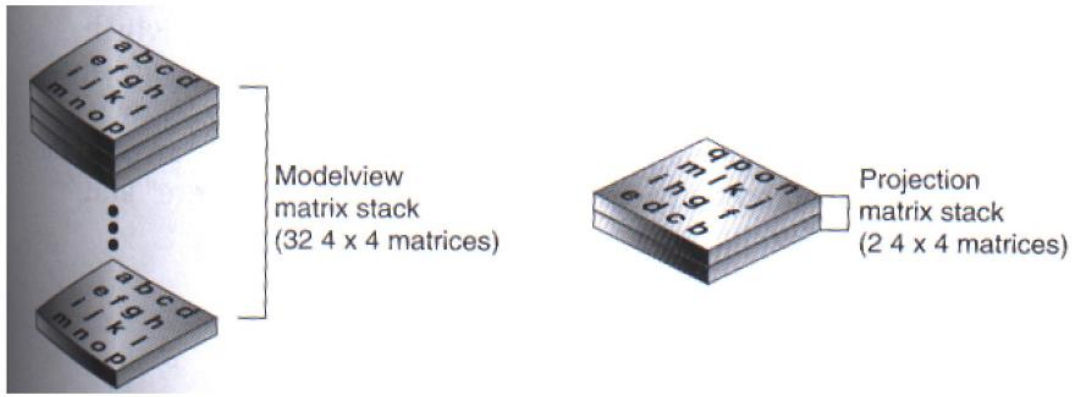
- Changer le mode : `GL_PROJECTION`
- `glFrustrum(`*left*`,`*right*`,`*bottom*`,`*top near*`,`*far*`)`
- `glOrtho(`*left*`,`*right*`,`*bottom*`,`*top near*`,`*far*`)`





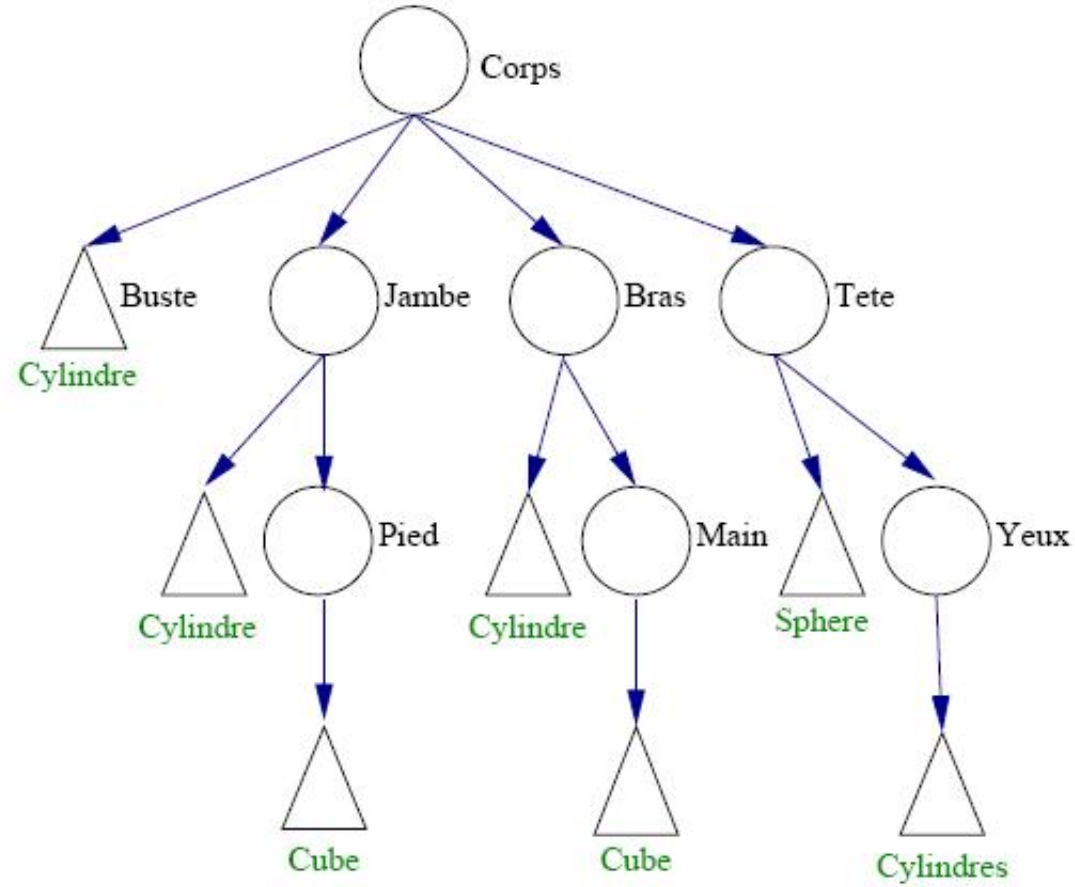
Piles de matrices

Toutes les opérations se font sur la matrice courante, or nécessité de manipuler plusieurs matrices
$\Rightarrow$ deux piles de matrices (une pour MODELVIEW, une pour PROJECTION)



- La matrice courante est la matrice du haut de la pile
- `glPushMatrix()`, `glPopMatrix()`

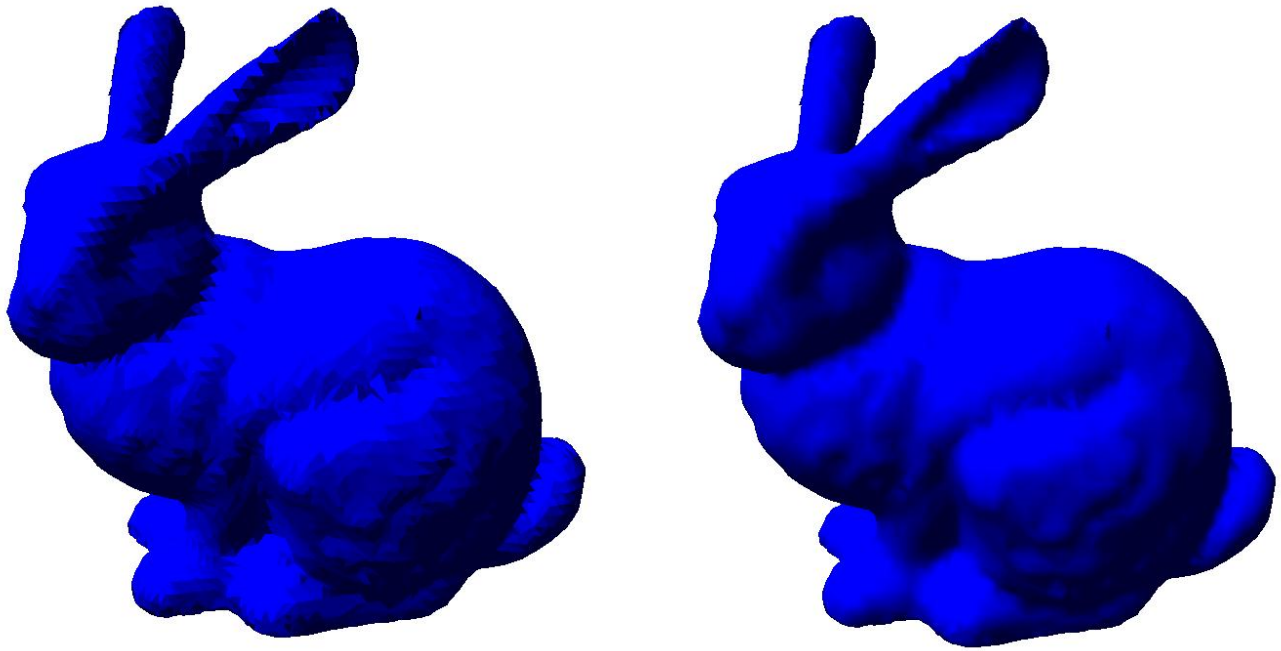
Piles de matrices - Exemple



Plan

Gestion des couleurs

- 2 modes de gestion des couleurs : RGBA ou color index (tableau de couleurs prédéfinies)
- RGBA : `glcolor4ub(1.0,0.0,0.5)`
- Color index : `glIndexi(50)`
- Flat shading : `glShadeModel(GL_FLAT)` ;
  Gouraud shading : `glShadeModel(GL_SMOOTH)`

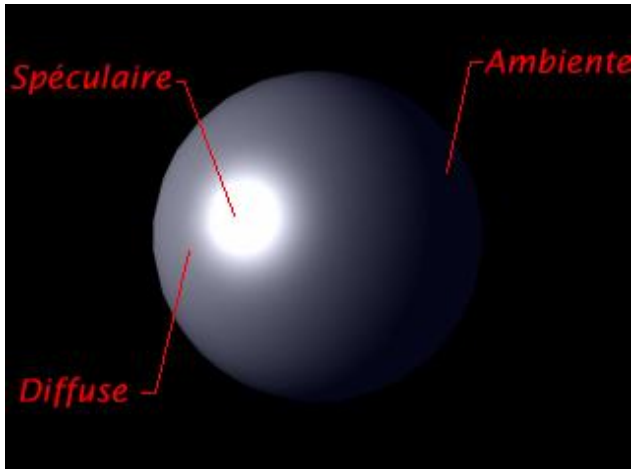


Gestion des couleurs en mode RGBA

- Rappel commande : `glColor3f(1.0,0.0,0.5)`
- `glEnable(GL_COLOR_MATERIAL)` pour activer le coloriage
  `glEnable(GL_DEPTH_TEST)` pour ne pas afficher les parties cachées
- Ne pas oublier :
  `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)`
  pour réinitialiser le color buffer et le Z-buffer à chaque fois qu'on redessine (gestion des buffers : cf. prochain cours)

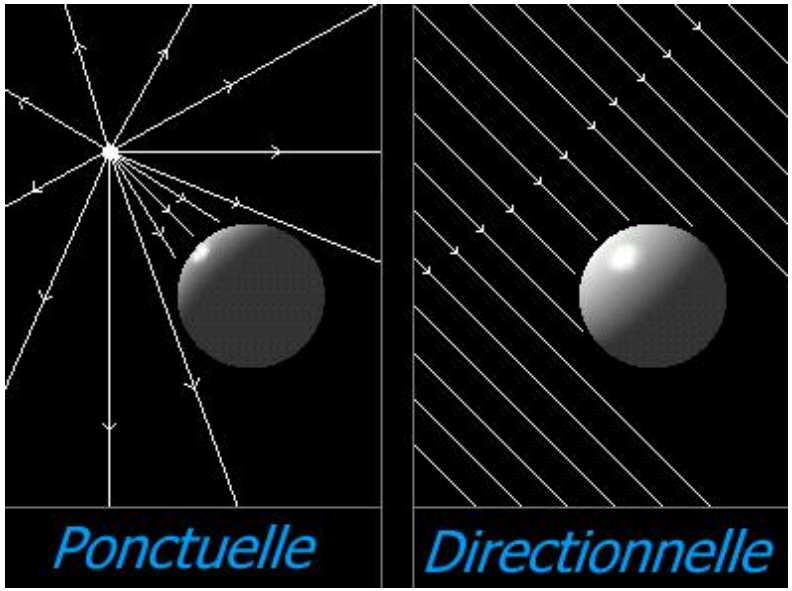
Lumière : création des sources

- **Initialisation** :
  - Activer l'éclairage : `glEnable(GL_LIGHTING)`
  - Allumer une lumière : `glEnable(GL_LIGHT0)`
    → attention, trop de lumières atténue les performances
- `glLightfv(GL_LIGHT0, GL_AMBIENT, l_ambient)`
  - Argument 1 : nom de la source (`GL_LIGHT0` . . . `GL_LIGHT7`)
  - Argument 2 : type de lumière (`GL_AMBIENT`, `GL_DIFFUSE`, `GL_SPECULAR`, . . .)
  - Argument 3 : intensité RGBA de la lumière



Cas particuliers 1/2

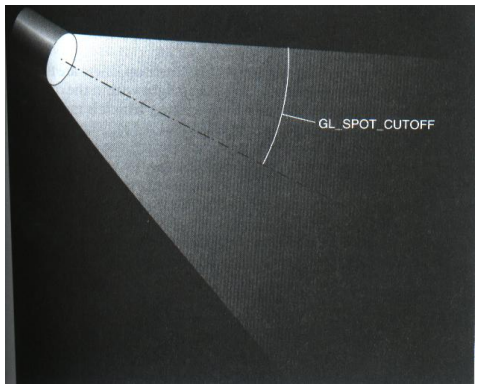
- **Cas particulier 1** :
  Argument 2 = `GL_POSITION`, Argument 3 = (x, y, z, w)
  - si $w = 0$: lumière directionnelle, (x, y, z) = direction
  - sinon : lumière ponctuelle, (x, y, z) = position dans le repère de l'objet



Cas particuliers 2/2

- **Cas particulier 2** :
  Argument 2 = `GL_*_ATTENUATION`
  $\Rightarrow$ change les constantes du facteur d'atténuation $\frac{1}{k_c + k_l d + k_q d^2}$

- **Cas particulier 3** : spots
  - Argument 2 = `GL_SPOT_CUTOFF`,
    Argument 3 = demi-angle du cône
  - Argument 2 = `GL_SPOT_DIRECTION`,
    Argument 3 = direction dans le repère de l'objet



Matériaux des objets

- `glMaterialfv(GL_FRONT,GL_AMBIENT,mat_ambient)`
  Permet de modifier les paramètres du matériau courant : couleur ambiante/diffuse/spéculaire, brillance, ...
- `glColorMaterial(GL_FRONT_AND_BACK,GL_DIFFUSE)`
  - Besoin de `glEnable(GL_COLOR_MATERIAL)`
  - **Attention !** Pas entre `glBegin()` et `glEnd()`
  - Utile pour changer un seul paramètre de matériau avec `glColor(...)`
  - Ne pas oublier `glDisable(GL_COLOR_MATERIAL)` ensuite pour ne pas avoir de surprises et de problèmes de performance

Plan

Nous avons vu :

- fonctionnement général
- modélisation : primitives géométriques
- modélisation : transformations repères
- rendu : couleurs
- rendu : lumières
- rendu : matériaux

Ce qu'il faut retenir:

- Machine à états
- `GL_MODELVIEW`, `GL_PROJECTION`
- le redbook

Prochain cours :

- blending, antialiasing, autres effets
- display lists
- textures
- buffers OpenGL
- interactivité