Cours 1

Franck HÉTROY

3A IRV, 05 octobre 2005

Planning

- 05/10/2005 (1h30) : cours
  ⇝ Introduction à OpenGL
- 16/11/2005 (1h30) : cours
  ⇝ Fonctionnalités OpenGL avancées
- 23/11/2005 (1h30) : T.P.
  ⇝ Introduction à libQGLViewer et petits exercices
- 07/12/2005 (3h) : T.P.
- 04/01/2005 (3h) : T.P.

Aujourd'hui

1. Introduction
2. Primitives 3D
3. Du 3D à l'écran
4. Couleurs et lumières

Livre de chevet

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

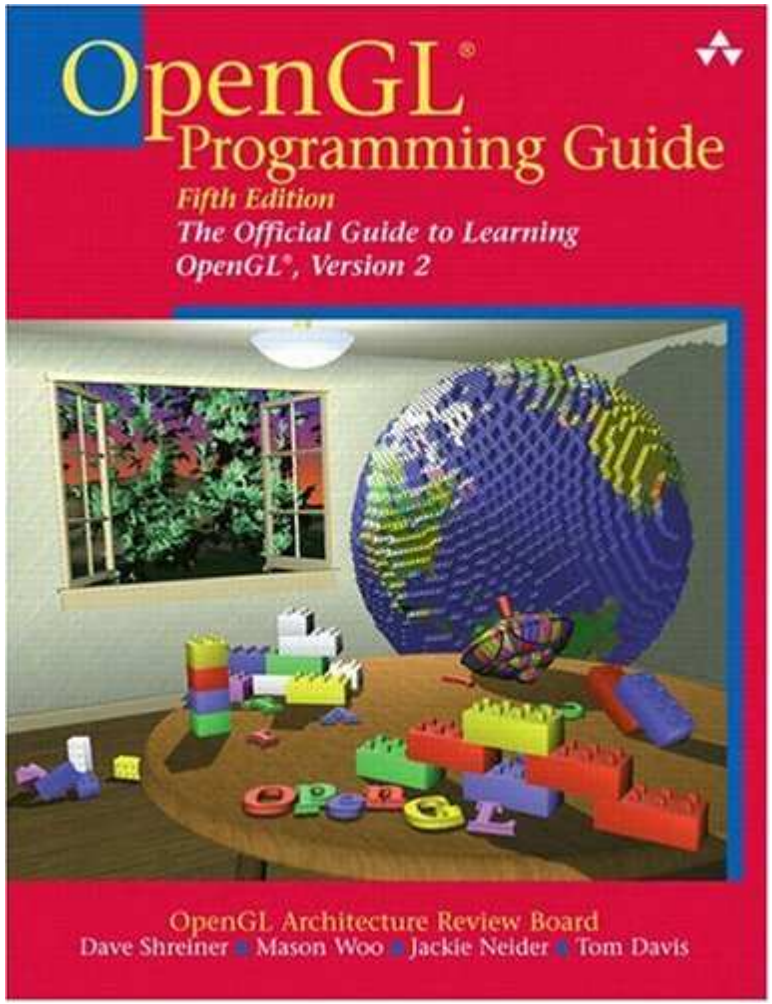
OpenGL® Programming Guide Fifth edition

Addison-Wesley

alias le red book

`http://opengl-redbook.com/`

⇝ LA Bible du programmeur OpenGL !



OpenGL - Cours 1

1 Introduction

2 Primitives 3D

3 De la 3D à l'écran

4 Couleurs et lumières

C'est quoi ?

- Une API (interface logicielle) pour l'architecture graphique
- Indépendant de l'archi : existe sous Unix/Linux, NT, OS2, ...
- Développée en 1989 (GL) par Silicon Graphics, portée sur d'autres architectures en 1993 (OpenGL)
- Comprend environ 250 commandes, servant à décrire les objets et les opérations qu'on peut effectuer dessus afin d'obtenir des applis 3D interactives

OpenGL ne fait pas tout

- Pas de commande pour créer et gérer un *viewer*
- Pas de commande de haut-niveau pour gérer les objets : seulement trois types de primitives géométriques (points, lignes et polygones)
- Besoin de bibliothèques complémentaires :
  - GLU (surcouche OpenGL, contient routines de base)
  - GLX (extension pour système XWindow) ou WGL (interface M$ Windows) ou AGL (interface pour Apple)
  - GLUT (boîte à outils : gestion de la fenêtre, création de modèles 3D un peu plus compliqués, . . .)

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
}
```

Voir aussi libQGLViewer

Le pipeline graphique

1. Construction des formes à partir des primitives géométriques.
2. Arrangement des objets dans l'espace 3D et sélection du point de vue.
3. Calcul des couleurs des objets (peut dépendre des conditions de lumière ou de texture).
4. Rasterization : conversion en image 2D (pixels)

D'autres opérations sont possibles pendant ces étapes (élimination parties cachées, . . .) ou entre la rasterization et l'affichage à l'écran (opérations sur les pixels).

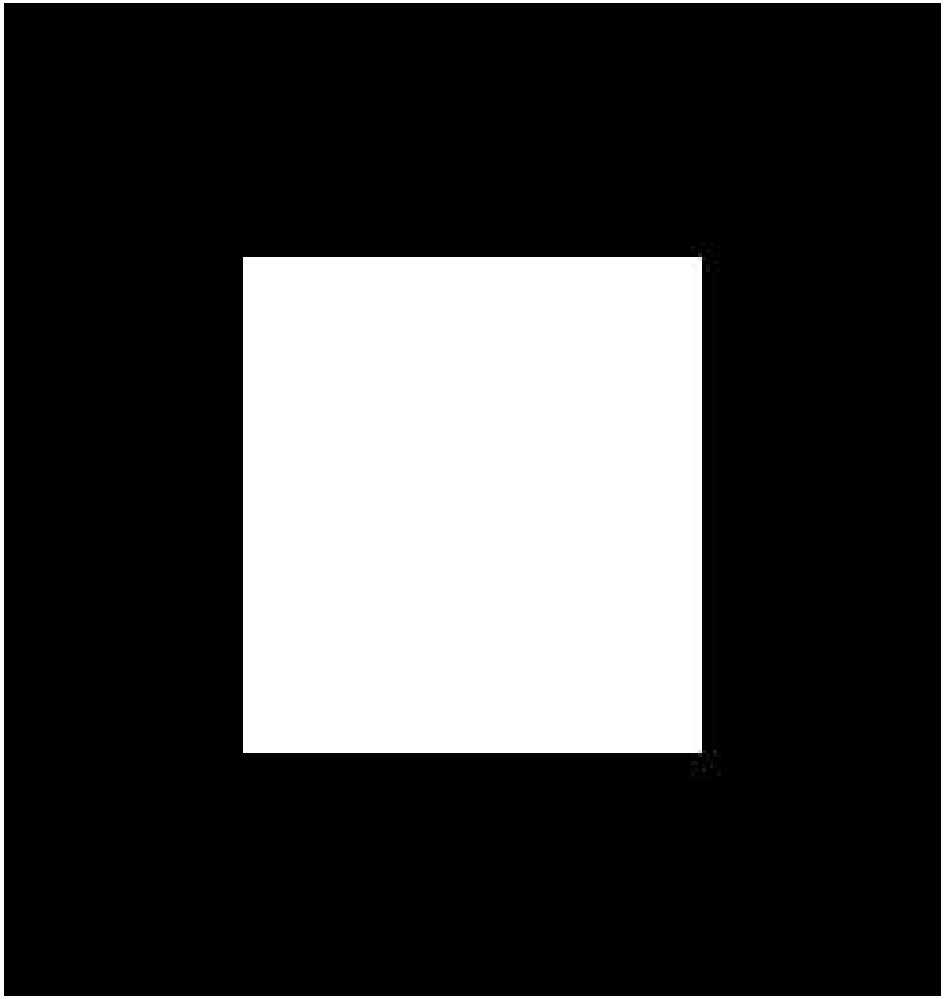
Un peu de vocabulaire

- Rendu (rendering) : processus de création d'images (2D) à partir de modèles (3D).
- Modèles : construits à partir de primitives géométriques (points, lignes, polygones), définies par leurs sommets (vertices).
- Bitplane : zone de la mémoire contenant un bit d'information pour chaque pixel de l'image (ex. : info sur les valeurs R,G ou B).
- 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
}
```

Le résultat



Elements 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

Possible : `glColor3fv(color_array);` : le paramètre est un vecteur (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 restent 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);`

OpenGL - Cours 1

1 Introduction

2 Primitives 3D

3 De la 3D à l'écran

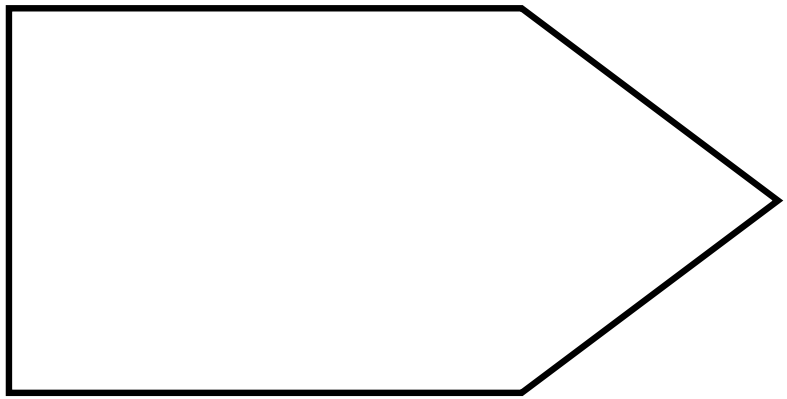
4 Couleurs et lumières

Primitives de base

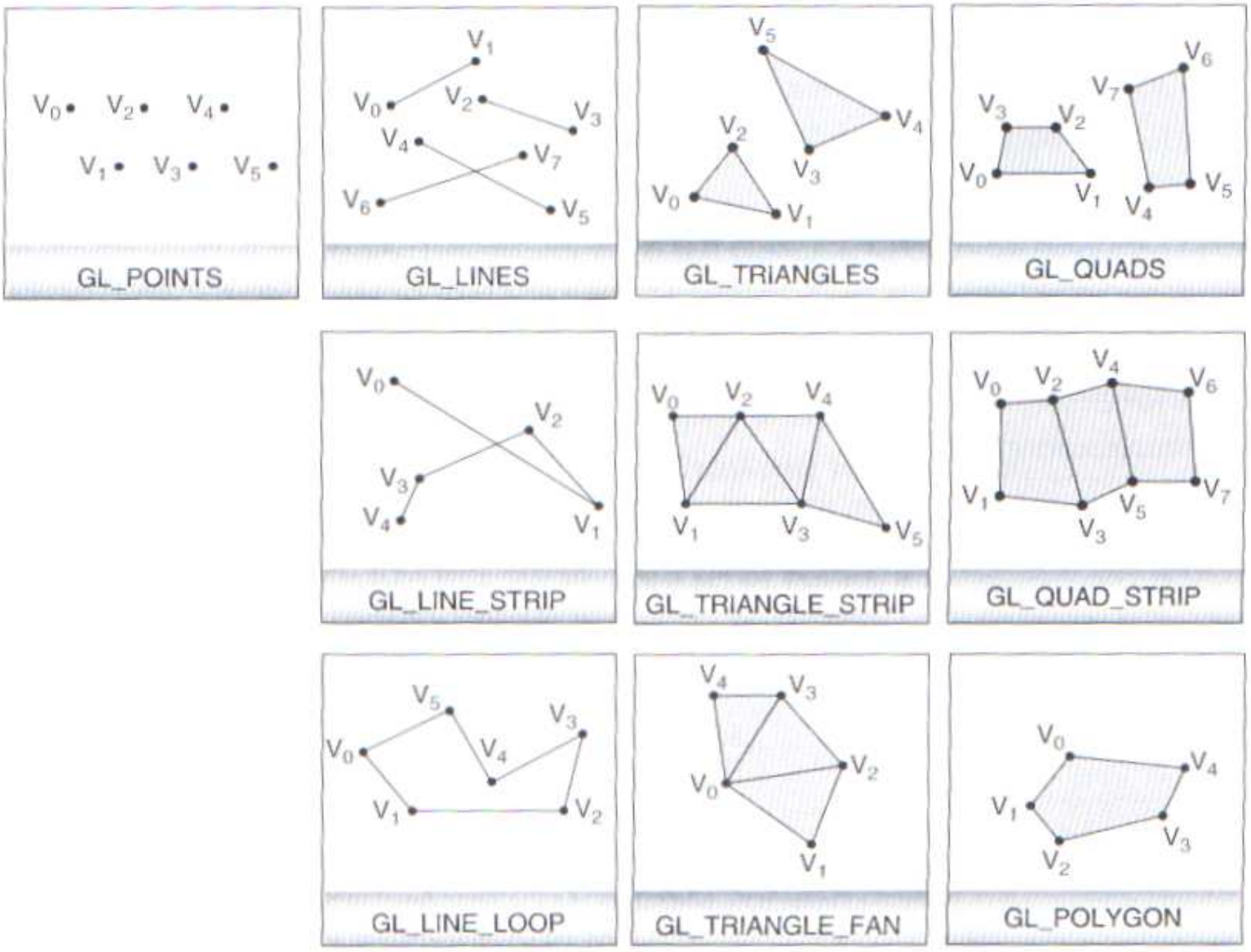
1. Sommets (vertices) : vecteur de flottants
   $\rightsquigarrow$ coordonnées homogènes ($w = 1.0$ par défaut)
2. Lignes : segments
3. Polygones : polygones convexes simples
4. Rectangles : dans le plan $z = 0$, côtés parallèles aux axes

Comment tracer un 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



Trucs supplémentaires

- Taille des sommets (en pixels) : `glPointSize(2.0);`
- Epaisseur des lignes (en pixels) : `glLineWidth(3.0);`
- Connaître les 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);`
- On peut préciser la couleur, la normale aux sommets, ...

Inclure les normales

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

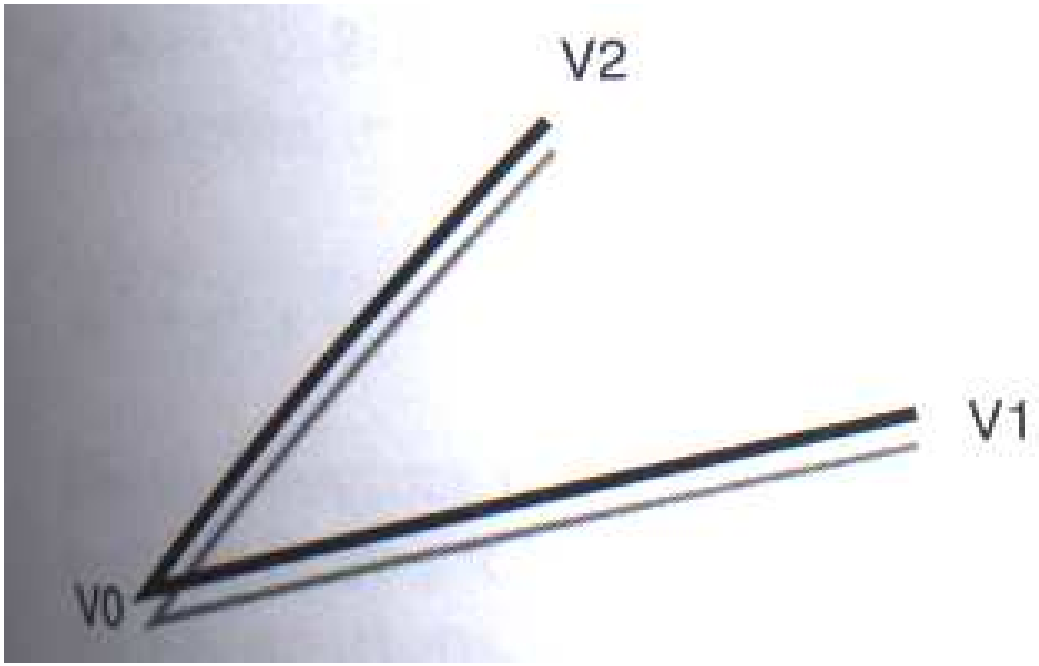
Attention à l'ordre : normale avant coordonnées

Comment ne tracer qu'une partie d'un contour

$\rightsquigarrow$ 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();
```

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);
```

  ⇝ *stride* = offset (en octets) entre 2 données consécutives

- Accès à un élément : `glArrayElement(0);`
- Accès à plusieurs éléments :

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

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

Exemple (1)

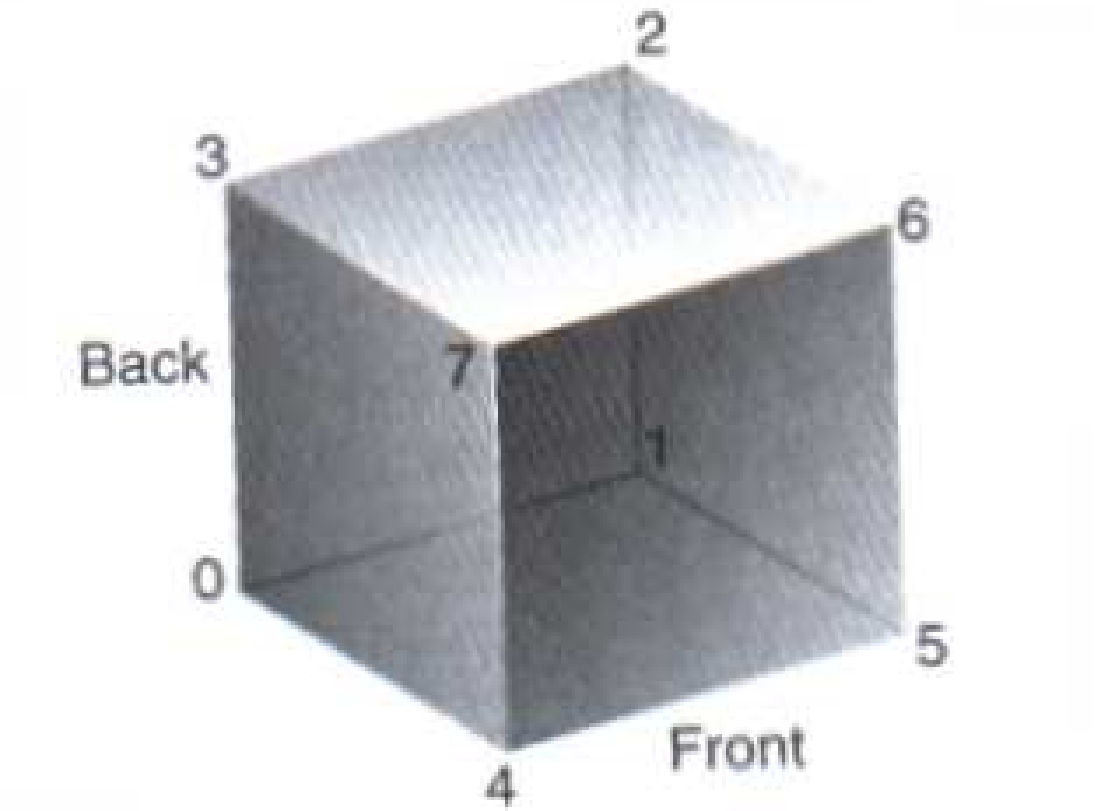
```
static GLint vertices[] = {
  25, 25, 100, 325, 175, 25,
  175, 325, 250, 25, 325, 325
};
static GLfloat colors[] = {
  1.0, 0.2, 0.2, 0.2, 0.2, 1.0,
  0.8, 1.0, 0.2, 0.75, 0.75, 0.75,
  0.35, 0.35, 0.35, 0.5, 0.5, 0.5
};
glEnableClientState(GL_COLOR_ARRAY);
glEnableClientState(GL_VERTEX_ARRAY);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

Exemple (2)

```
// 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);
```

OpenGL - Cours 1

1 Introduction

2 Primitives 3D

3 De la 3D à l'écran

4 Couleurs et lumières

Voir en 3D

- Modeling transformations
  (passage repère de l'objet $\longrightarrow$ repère du monde)
- Viewing transformations
  (passage repère du monde $\longrightarrow$ repère caméra)
- Projection sur l'écran

$\Rightarrow$ cf. cours d'Antoine Bouthors

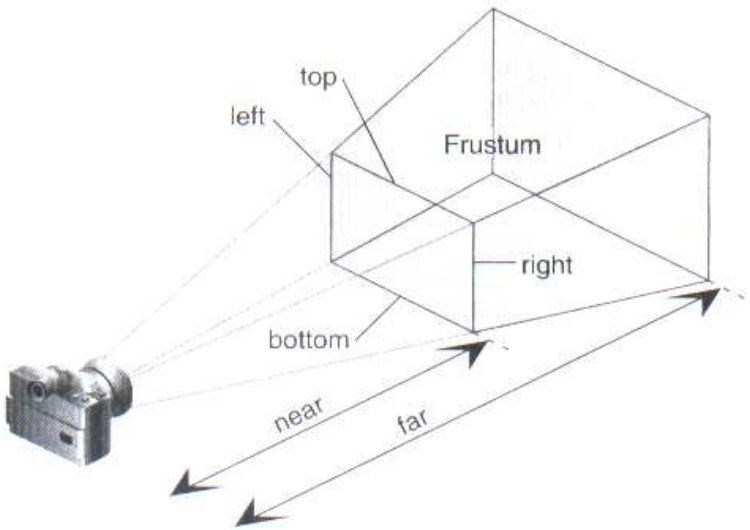
$\rightsquigarrow$ matrices 4x4

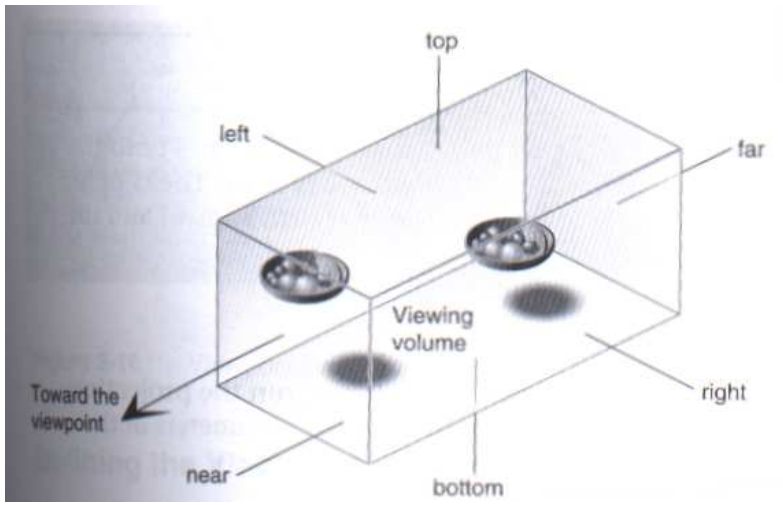
Manipuler les matrices 4x4 sous OpenGL

- `glMatrixMode(GL_MODELVIEW)` // ou `GL_PROJECTION` ou `GL_TEXTURE`
  ⇝ 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()`

OpenGL - Cours 1

1 Introduction

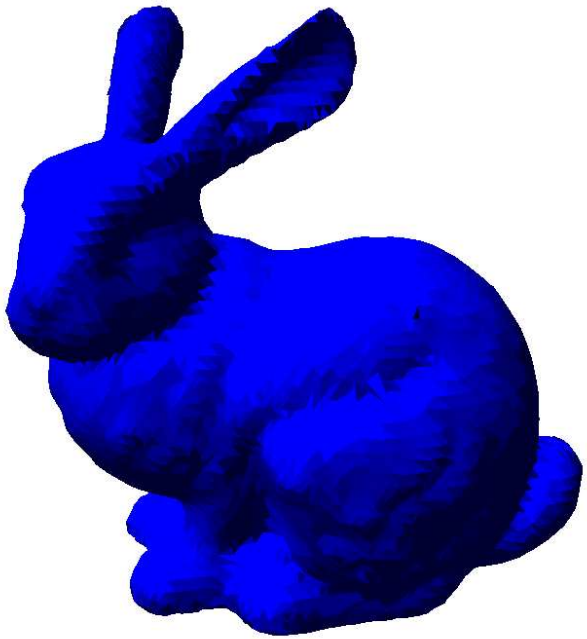
2 Primitives 3D

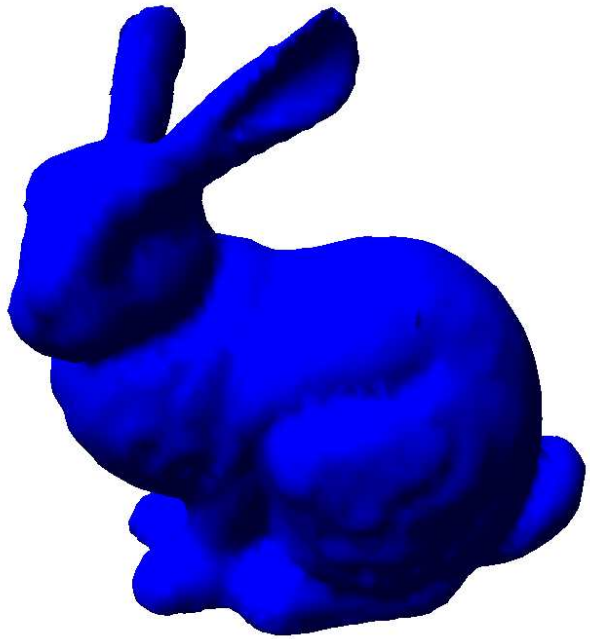
3 De la 3D à l'écran

4 Couleurs et lumières

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)`



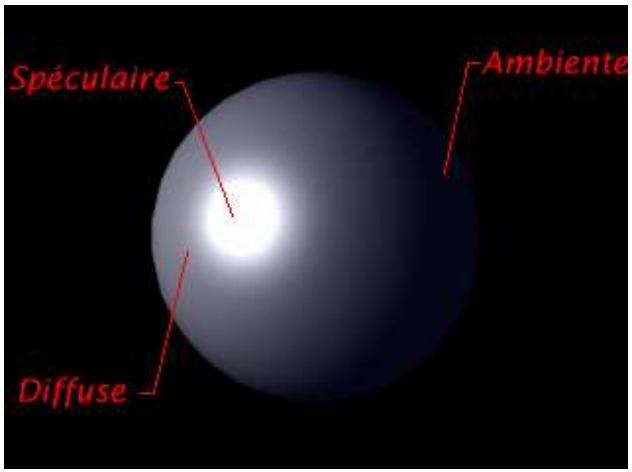


Gestion des couleurs en mode RGBA

- Rappel commande : `glColor3f(1.0,0.0,0.5)`
- Ajouter `glEnable(GL_COLOR_MATERIAL)` pour activer le coloriage et `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)

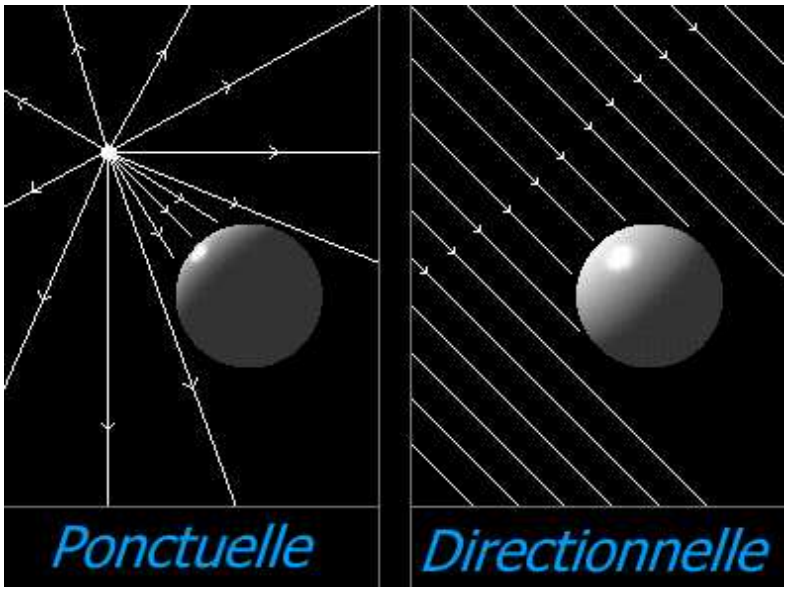
Lumière : création des sources

- 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



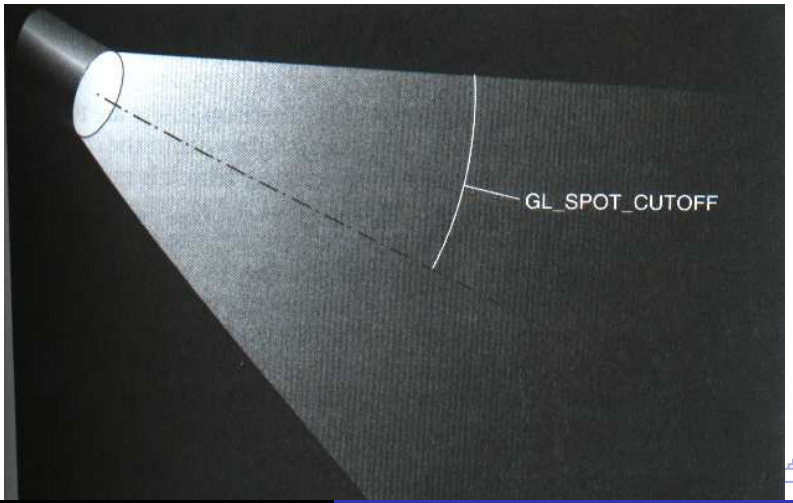
Cas particuliers (1)

- Cas particulier 1 :
  argument 2 = `GL_POSITION`, argument 3 = (x, y, z, w)
  $\Rightarrow$
  - 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)

- Cas particulier 2 :
  argument 2 = `GL_*_ATTENUATION`
  $\Rightarrow$ change les constantes du facteur d'atténuation
  $\frac{1}{k_c + k_l d + k_q d^2}$ (par défaut $(1.0, 0.0, 0.0)$)
- 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

Fin

C’est tout pour aujourd’hui !

Prochain cours

- Blending, antialiasing et autres effets
- *Display lists*
- Plaquage de textures
- Les buffers OpenGL
- Sélection d'objets