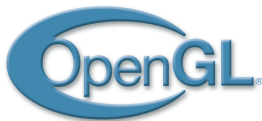


# Introduction à OpenGL avec GLUT

Nikolas Stott

INRIA Saclay - CMAP, École Polytechnique, Université Paris-Saclay

9 février 2017



- 1 OpenGL et GLUT : présentation
  - OpenGL : quoi et comment ?
  - GLUT : quoi et comment ?
- 2 Les fonctions principales GLUT
- 3 La librairie mathématique Eigen
- 4 Éléments de modélisation avec OpenGL
- 5 Les commandes OpenGL
  - Les commandes basiques
  - Commandes avancées
  - Transformations géométriques
  - L'éclairage
  - Gestion de la caméra

# Qu'est ce qu'OpenGL ?

OpenGL (Open Graphics Library) est une bibliothèque graphique 2D/3D pour des applications 3D (interactives) :

- Interface logicielle bas niveau avec le hardware graphique
- 150 commandes différentes pour spécifier objets et opérations

OpenGL est indépendant du hardware et utilisé dans différents langages à travers différentes bibliothèques :

- GLU/GLUT en C/C++
- Java OpenGL (JOGL) en Java
- WEBGL en Javascript

OpenGL ne gère pas le fenêtrage ou l'interface graphique.

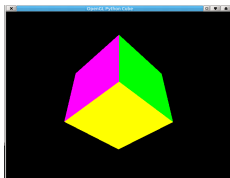
# Qu'est ce que GLUT ?

GLUT (OpenGL Utility Toolkit) est une interface de programmation pour OpenGL qui gère le fenêtrage.

GLUT est simple, petit et utile pour apprendre à découvrir OpenGL.

GLUT contient les fonctionnalités suivantes :

- gestion de l'affichage de fenêtres de rendu OpenGL
- gestion du temps, d'événements et d'interaction utilisateur
- création rapide d'objets primitifs (cube, tétraèdre, sphère, cône, etc) pleins ou maillage seul.



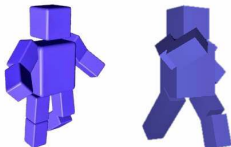
# Qu'est ce qu'OpenGL sait faire ?

## Modélisation/Visualisation

- Création de géométries complexes
- Habillage de la géométrie : couleur, texture, éclairage...
- Visualisation des objets



## 3D Animation



## Animation

OpenGL permet d'animer :

- la caméra dans la scène
- les objets dans la scène
- le maillage des objets

## Contenu

La fonction main doit :

- initialiser GLUT : `glutInit(&argc,argv);`
- paramétrer l'affichage avec `glutInitDisplayMode( ... );` :  
GLUT\_RGBA, GLUT\_DEPTH, GLUT\_SINGLE ou GLUT\_DOUBLE
- créer la fenêtre : `glutCreateWindow("C'est bientôt fini ;)")`;
- initialiser les variables/objets du programme
- déclarer les fonctions
  - de dessin : `glutDisplayFunc( ... );`
  - de redimensionnement : `glutReshapeFunc( ... );`
  - d'interaction souris : `glutMouseFunc( ... );`
  - d'interaction clavier : `glutKeyboardFunc( ... );`
  - d'évolution autonome : `glutIdleFunc( ... );`
  - de temporisation : `glutTimerFunc( ... );`
- lancer la boucle infinie : `glutMainLoop();`

# Autres fonctions (1)

## Fonction de dessin

Donnée en paramètre de `glutDisplayFunc(...)`

Elle ne prend rien en paramètre.

Son rôle est de tracer l'image courante :

- effacer l'image précédente : `glClear(GL_COLOR_BUFFER_BIT);`
- dessiner ce que l'utilisateur souhaite
- demander de l'afficher : `glFlush();` ou `glSwapBuffers();`

## Fonction de redimensionnement

Donnée en paramètre de `glutReshapeFunc( ... );`

Elle prend en paramètre les dimensions du viewport.

Elle doit assurer la cohérence de la fenêtre de tracé :

- déclarer le viewport : `glViewport(x1,y1,x2,y2);`
- charger les paramètres caméra initiaux

# Autres fonctions (2)

## Fonction d'interaction souris

Donnée en paramètre de `glutMouseFunc( ... );`

Elle prend en paramètre le bouton activé, l'état du bouton et la position écran lors de l'action.

- Le bouton prend les valeurs `GLUT_LEFT/MIDDLE/RIGHT_BUTTON`
- L'état prend les valeurs `GLUT_UP` et `GLUT_DOWN`

## Fonction d'interaction clavier

Donnée en paramètre de `glutKeyboardFunc( ... );`

Elle prend en paramètre la touche activée et la position écran de la souris lors de l'action.



## Autres fonctions (3)

### Fonction d'évolution autonome

Donnée en paramètre de `glutIdleFunc( ... );`

Appelée lorsqu'aucune action n'est déclenchée, elle ne prend aucun paramètre. C'est la fonction qui calcule le nouvel état du système, etc.

### Fonction de temporisation

Donnée en paramètre de `glutTimerFunc( ... );`

Fonction avancée qui permet d'introduire des paramètres temporels dans le programme.

`glutTimerFunc(DeltaT,timer,0)` appelle la fonction de temporisation `timer` au moins toutes les `DeltaT` ms.

### Fonction d'actualisation

`glutPostRedisplay();` est la fonction qui demande à GLUT de calculer et d'afficher une nouvelle image sur l'écran.

# Commandes principales

## Eigen : `:Vector3f v`

- Vecteur de 3 floats  $x,y,z$  : Eigen : `:Vector3f(x,y,z)`
- Accés aux valeurs : `v[i]`
- Compatible avec les opérations standards sur des vecteur :  $3*u - 2*v$  est une syntaxe valide

## Produit scalaire $\langle u, v \rangle$

- Calculé par `u.dot(v)`
- Norme d'un vecteur : `sqrt(u.dot(u))` ou `u.norm()`
- Vecteur normalisé :  $u_{\text{norm}} = u.\text{normalized}()$
- Autre version (inplace) : `u.normalize()`

## Produit vectoriel $u \wedge v$

- Calculé par `u.cross(v)`

TP :

Simulation d'une flotte de drones

# La brique de base : le sommet

Dans le cadre de ce cours, ...

... l'ordinateur ne connaît que le point (vecteur ou *vertex*) !

⇒ Pas de courbe, pas de boule, pas de géométrie lisse.

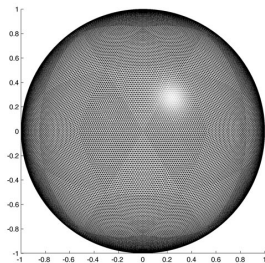
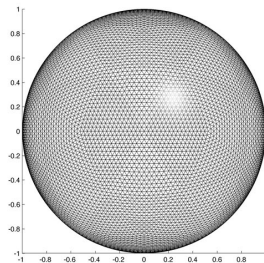
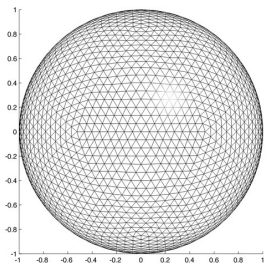
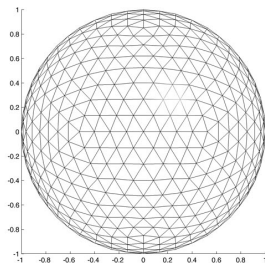
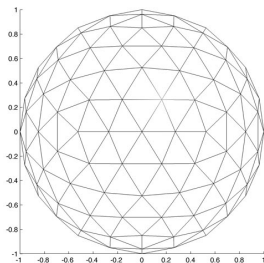
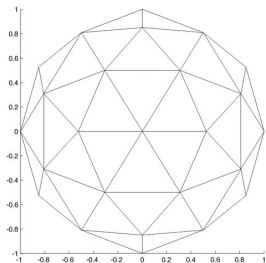
## Assemblage de géométrie

On peut créer des objets simples à partir de sommets :

- lignes (ou *edges*),
- polygones (ou *faces*).

⇒ On va assembler ces objets simples en objets plus complexes...

# Illustration



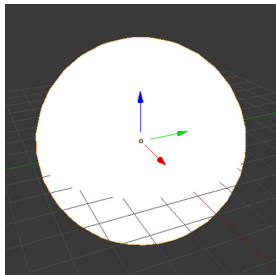
# C'est tout ?

## Ce n'est pas suffisant !

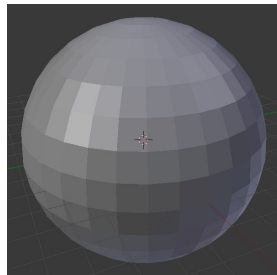
Il faut également déclarer des informations supplémentaires sur l'objet :

- Orientation des faces,
- Matériaux de l'objet,
- Couleur des faces,
- Texture des faces...

pour qu'il interagisse avec son environnement : lumières...



Sans aucune information :



Avec orientation et matériaux :

# Pas votre lampe habituelle

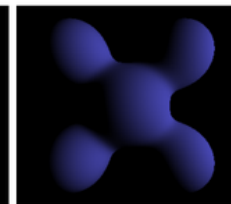
## Les composantes lumineuses du modèle de Phong

Une lumière OpenGL est caractérisée par 5 paramètres :

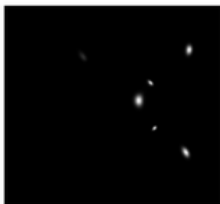
- Position : vecteur XYZ(W)
- Direction du spot : vecteur direction XYZ
- Intensité Diffuse : vecteur RGBA
- Intensité Spéculaire : vecteur RGBA
- Intensité Ambiante : vecteur RGBA



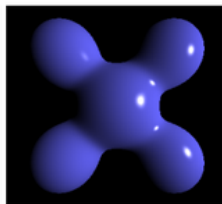
Ambient



Diffuse



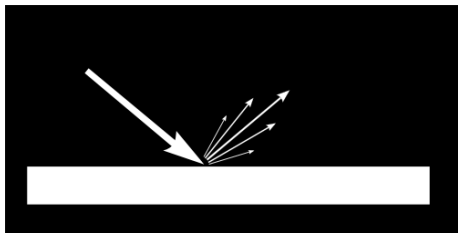
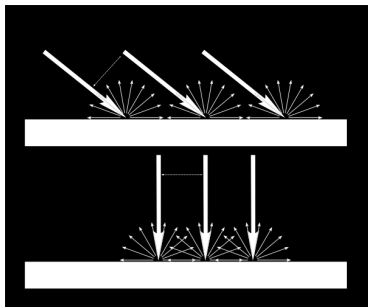
Specular



= Phong Reflection

## Influence de la normale

Sur une face, l'éclairage est calculé avec :  $\left\{ \begin{array}{l} \text{le vecteur lumineux incident,} \\ \text{la normale à la face} \end{array} \right.$



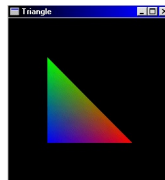
Il faut donner les normales à l'ordinateur : il ne sait pas les calculer...



# Couleur et Textures

## Colorier la géométrie

- Ce sont les sommets qui portent l'information de couleur ;
- Les faces sont coloriées par interpolation.



## Texturer la géométrie

- On attribue un point de la texture à chaque sommet ;
- Les faces sont texturées par interpolation.



# Ce que l'on ne va pas apprendre à faire

## Les ombres

Les ombres portées ne sont pas calculées automatiquement : c'est à l'utilisateur de les calculer.

## Texturer un objet

C'est encore à l'utilisateur de définir comment la texture s'applique sur l'objet (UV mapping).

## Traitement des shaders

Pas de traitement en détail des normales, effets post-traitement, etc...  
Pas de canaux de textures améliorés : normal map, bump map, displacement map, ...

# Les briques élémentaires (1)

## Les sommets

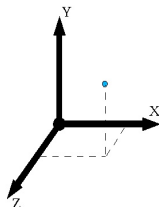
Les sommets sont déclarés par la commande `glVertex??(x,y,z,w)`.

Exemples :

- `glVertex2i(1,2);`
- `glVertex3f(2.4f,4.1f,-0.1f);`
- `glVertex4d(4.0,3.0,2.0,1.0);`

## Par défaut

- `glVertex2?` place les points dans le plan  $z = 0$  (et  $w = 1$ )
- `glVertex3?` fixe  $w$  à 1.



# Les briques élémentaires (2)

## La déclaration de primitive

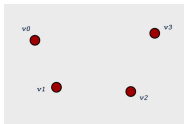
L'assemblage de sommets en faces se fait entre les instructions

```
glBegin (TYPE_DE_LA_PRIMITIVE);  
    //declarations optionnelles  
    glVertex ?? (...);  
    ...  
    glVertex ?? (...);  
    //declarations optionnelles  
glEnd ();
```

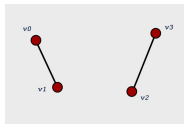
Nous verrons les autres déclarations plus loin.

Attention : puisque nous travaillons sur de la géométrie, OpenGL doit être configuré en mode GL\_MODELVIEW : `glMatrixMode(GL_MODELVIEW);` !

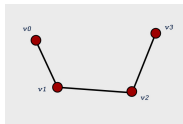
# Primitives géométriques disponibles



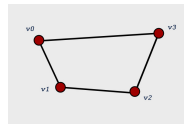
GL\_POINTS



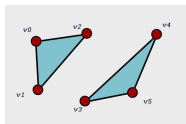
GL\_LINES



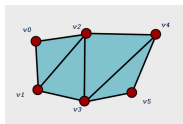
GL\_LINE\_STRIP



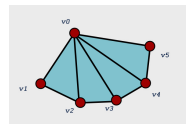
GL\_LINE\_LOOP



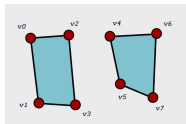
GL\_TRIANGLES



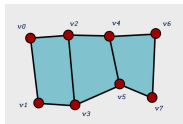
GL\_TRIANGLE\_STRIP



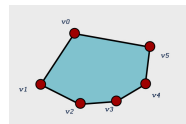
GL\_TRIANGLE\_FAN



GL\_QUADS



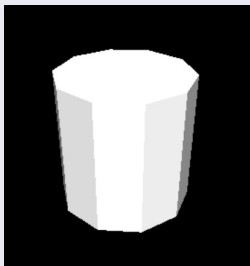
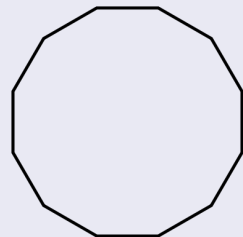
GL\_QUAD\_STRIP



GL\_POLYGON

# Exemples simples de modélisation : cylindre

## Cylindre



## Réponse

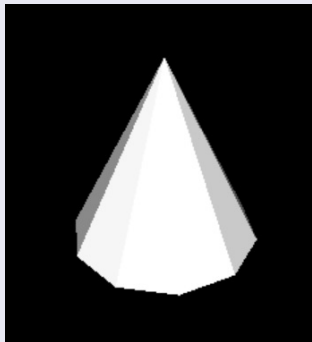
```
glBegin(GL_TRIANGLE_FAN);  
... //base inferieure  
glEnd();  
glBegin(GL_TRIANGLE_FAN);  
... //base superieure  
glEnd();  
glBegin(GL_QUAD_STRIP);  
... //faces laterales  
glEnd();
```

## Une autre solution

```
gluCylinder(...);
```

# Exemple de modélisation : cône

## Cône



## Réponse

```
glBegin(GL_TRIANGLE_FAN);  
... //face inferieure  
glEnd();  
glBegin(GL_TRIANGLE_FAN);  
... //faces laterales  
glEnd();
```

## Une autre solution

```
glutSolidCone (...);
```

# Commandes à disposition entre glBegin() et glEnd() (1)

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

```
glNormal3f(x,y,z)
```

Déclarer la normale au polygone que l'on trace.

**Doit impérativement être déclaré avant de finir de tracer le polygone !**

**Utile** : initialiser glEnable(GL\_NORMALIZE); : normalisation auto.

```
glBegin(GL_QUADS);  
    glNormal3f(0,1,0);  
    glVertex3f(1,0,1);  
    glVertex3f(1,0,-1);  
    glVertex3f(-1,0,-1);  
    glVertex3f(-1,0,1);  
glEnd();
```

```
glBegin(GL_TRIANGLE_FAN);  
    glVertex3f(1,0,1);  
    glVertex3f(1,0,-1);  
    glNormal3f(0,1,0);  
    glVertex3f(-1,0,-1);  
    glNormal3f(0,1,0);  
    glVertex3f(-1,0,1);  
glEnd();
```



# Commandes à disposition entre glBegin() et glEnd() (2)

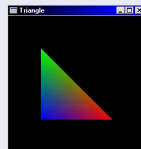
## glColor??(r,g,b,a)

Déclarer la couleur à utiliser dans la suite du programme.

r,g,b,a doivent prendre des valeurs entre 0.0 et 1.0.

Par défaut : glColor4d(1,1,1,1);.

Pour activer la composante alpha : glEnable(GL\_BLEND);.



```
glBegin(GL_TRIANGLE);  
    glNormal3d(0, 0, 1);  
    glColor3d(1, 0, 0);  
    glVertex3dv(a);  
    glColor3d(1, 1, 0);  
    glVertex3dv(b);  
    glColor3d(1, 1, 1);  
    glVertex3dv(c);  
glEnd();
```

# Commandes à disposition entre glBegin() et glEnd() (3)

```
glMaterial?(face, gl_param,  
values)
```

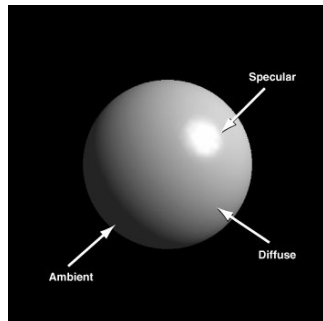
face détermine quelle face est calculée :

- **GL\_FRONT\_AND\_BACK**

gl\_param est le paramètre du matériau :

- GL\_AMBIENT : 4 variables  $[-1.0, 1.0]$
- GL\_DIFFUSE : 4 variables  $[-1.0, 1.0]$
- GL\_SPECULAR : 4 variables  $[-1.0, 1.0]$
- GL\_EMISSION : 4 variables  $[-1.0, 1.0]$
- GL\_SHININESS : 1 variable  $[0, 128]$

values est un pointeur vers le tableau de paramètres



# Les transformations élémentaires

## Principe de transformation

Chaque commande de transformation agit sur les objets qui suivent !

## Translation

`glTranslate?(x,y,z)`

## Changement d'échelle

`glScale?(x,y,z)` multiplie l'échelle par  $x,y,z$  dans les directions  $x,y,z$  respectivement.

## Rotation autour d'un axe

`glRotate?(t,x,y,z)` effectue une rotation de  $t$  autour de l'axe  $[x \ y \ z]^T$

## Exercice

Comment faire pour animer un objet dans une scène ? Par exemple, faire tourner un carré autour d'une de ses diagonales ? Et un cube ?

### Solution 1 :

```
glBegin (GL_QUADS);  
...  
glVertex3f (cosf (t) , sinf (t) , 0);  
...  
glEnd ();
```

### Solution 2 :

```
glRotatef (t , 0 , 0 , 1);  
glBegin (GL_QUADS);  
...  
glVertex3f (1 , 0 , 0);  
...  
glEnd ();  
glRotatef (-t , 0 , 0 , 1);
```

En général, la 2ème solution est préférable, surtout si les objets sont complexes : calcul sur le GPU et non sur le CPU.

# Manipulation de transformations (1)

## Pas très pratique...

Faire toutes les transformations en double, avant et après avoir tracé mon objet ?

```
glRotatef(t, 0, 0, 1);  
...  
glRotatef(-t, 0, 0, 1);
```

## Pile de matrices : sauvegarder un état de transformation

On choisit la pile de matrices sur laquelle on agit avec `glMatrixMode( GL_MODELVIEW | GL_PROJECTION )`.

- `glLoadIdentity()` : annuler toutes les transformations
- `glPushMatrix()` : sauvegarder l'état de transformation courant
- `glPopMatrix()` : retour à l'état précédemment enregistré

# Manipulation de transformations (2)

## Beaucoup plus pratique

On reprend l'exemple précédent :

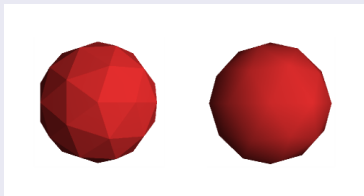
```
glRotatef(t,0,0,1);  
glTranslatef(4,3,1);  
...  
glTranslatef(-4,-3,-1);  
glRotatef(-t,0,0,1);
```

```
glPushMatrix();  
    glRotatef(t,0,0,1);  
    glTranslatef(4,3,1);  
    ...  
glPopMatrix();
```

## Type d'ombrage :

La commande `glShadeModel(??)` permet de choisir entre 2 types d'ombrages :

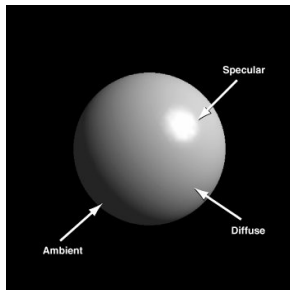
- `GL_FLAT`
- `GL_SMOOTH`



# Le modèle d'éclairage dans OpenGL

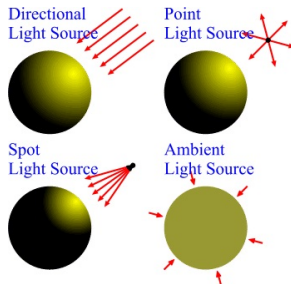
## Composantes lumineuses

- Composante Émissive
- Composante Ambiante
- Composante Diffuse
- Composante Spéculaire



## Types de sources lumineuses

- Source Ambiante
- Source Ponctuelle
- Source Directionnelle
- Source Spot





## Les lumières dans OpenGL

- OpenGL sait gérer jusqu'à 8 sources de lumières simultanément,
- Elles sont nommées `GL_LIGHTi`, avec  $0 \leq i < 8$ .

## Activation

- Configuration d'OpenGL pour utiliser les lumières :

```
glEnable(GL_LIGHTING);
```

- Activation d'une lumière :

```
glEnable(GL_LIGHT0);
```

- On déclare les couleurs de faces comme des matériaux :

```
glEnable(GL_COLOR_MATERIAL);
```

# Les commandes de lumière

## `glLight(gl_light, gl_param, values)`

Les paramètres ajustables et leurs valeurs :

- `GL_AMBIENT` : 4 variables (intensité RGBA)
- `GL_DIFFUSE` : 4 variables (intensité RGBA)
- `GL_SPECULAR` : 4 variables (intensité RGBA)
- `GL_POSITION` : 4 variables (position XYZW)
- `GL_SPOT_DIRECTION` : 3 variables (vecteur direction)

## `glLightModel(gl_param, gl_value)` (Description)

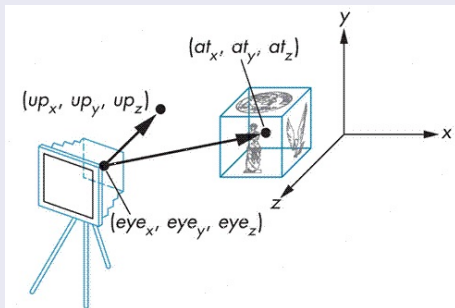
Offrir plus de contrôle sur la compréhension des paramètres lumineux :

- `GL_LIGHT_MODEL_AMBIENT` : contrôle l'intensité de la lumière ambiante
- `GL_LIGHT_MODEL_COLOR_CONTROL` : contrôle des éventuels conflits entre textures et lumière

# Utilisation simple de la caméra

## Une fonction GLU utile

- `gluLookAt(eyeX, eyeY, eyeZ, atX, atY, atZ, upX, upY, upZ)`



On manipule des matrices géométriques :  
`glMatrixMode(GL_MODELVIEW);`