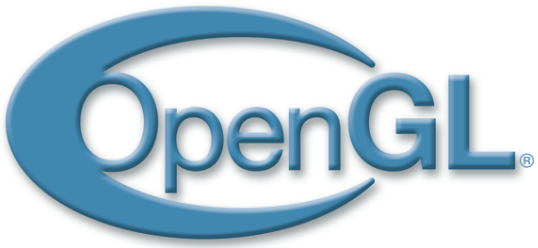
Introduction à OpenGL avec GLUT

Nikolas Stott

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

16 mars 2017



Plan

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

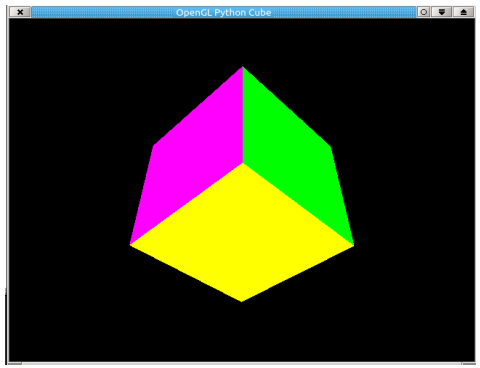
OpenGL ne gère pas le fenêtrage ou l'interface graphique.

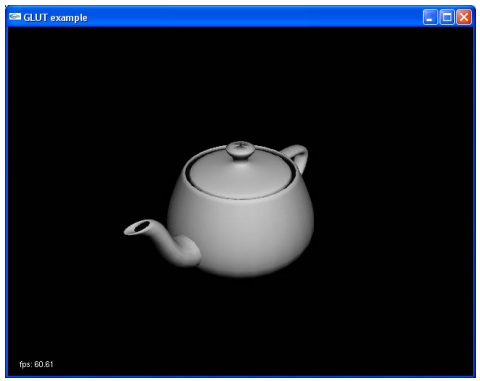
Qu'est ce que GLUT ?

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.

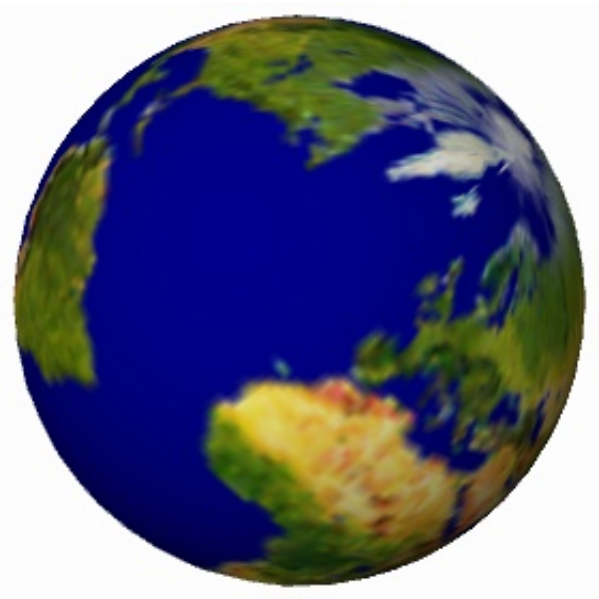


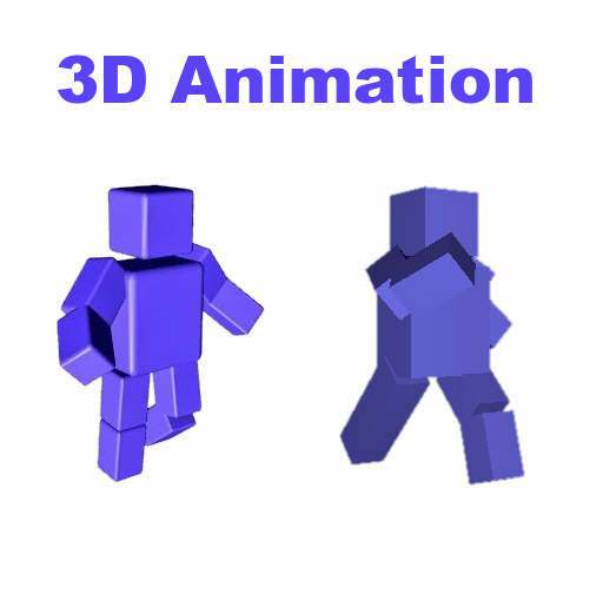


Qu'est ce qu'OpenGL sait faire ?

Modélisation/Visualisation

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





Animation

OpenGL permet d'animer :

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

La brique de base : le sommet

Dans le cadre de ce cours, ...

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

$\Rightarrow$ Pas de courbe, pas de boule, pas de géométrie lisse.

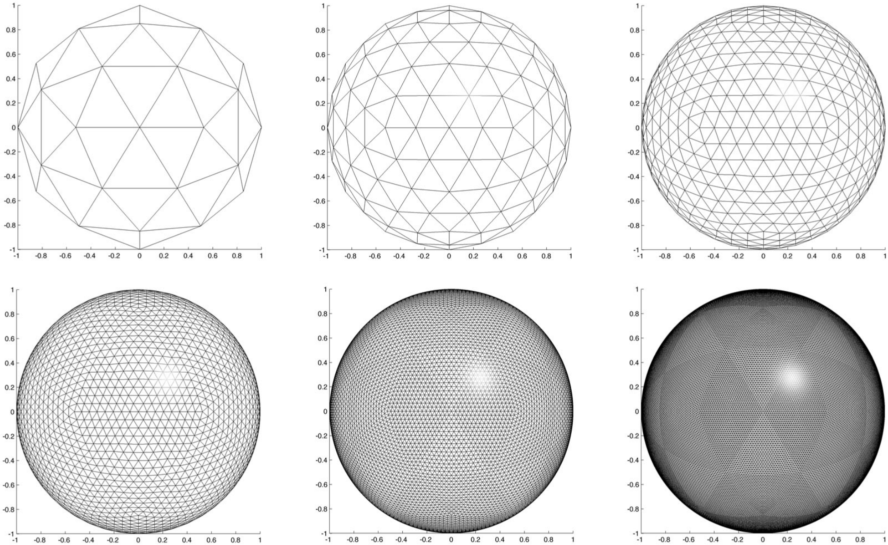
Assemblage de géométrie

On peut créer des objets simples à partir de sommets :

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

$\Rightarrow$ On va assembler ces objets simples en objets plus complexes...

Illustration



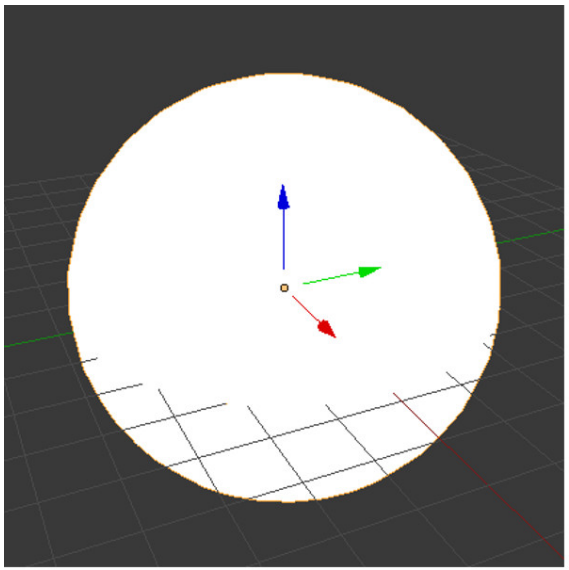
C'est tout ?

Ce n'est pas suffisant !

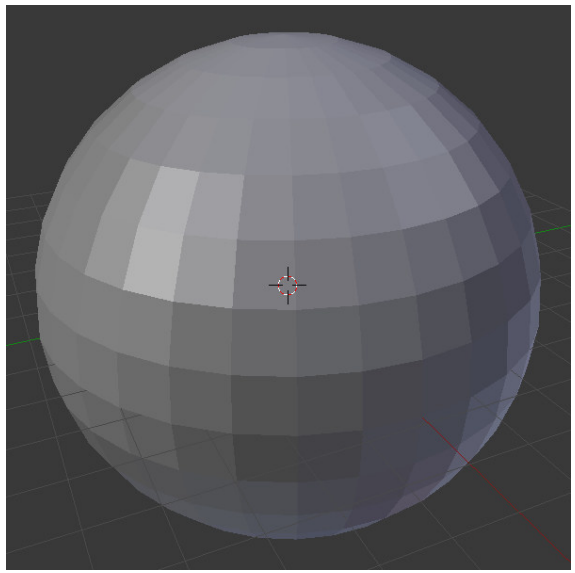
Il faut également déclarer des informations supplémentaires sur l'objet :

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

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



Sans aucune information :



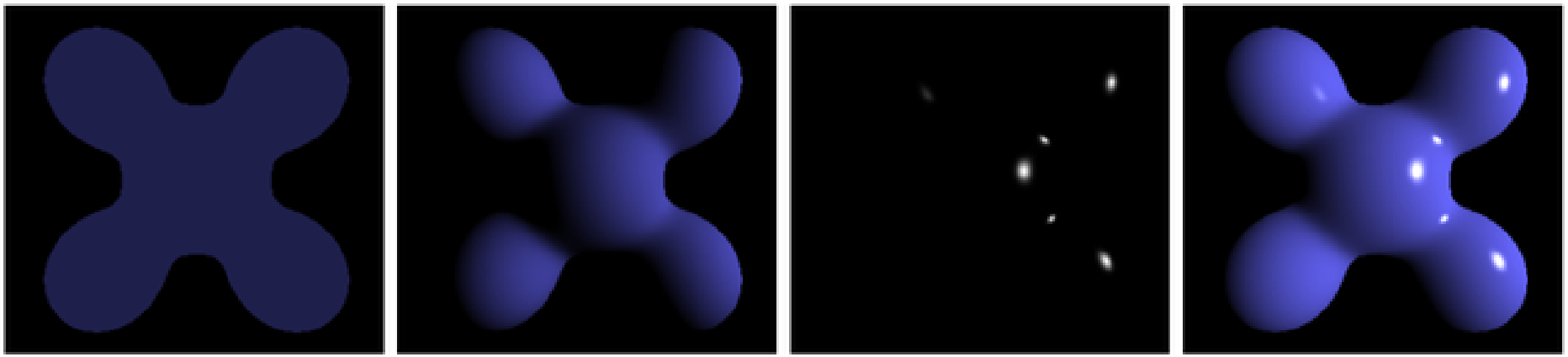
Avec orientation et matériaux :

Pas votre lampe habituelle

Les composantes lumineuses du modèle de Phong

Une lumière OpenGL est caractérisée par 5 paramètres :

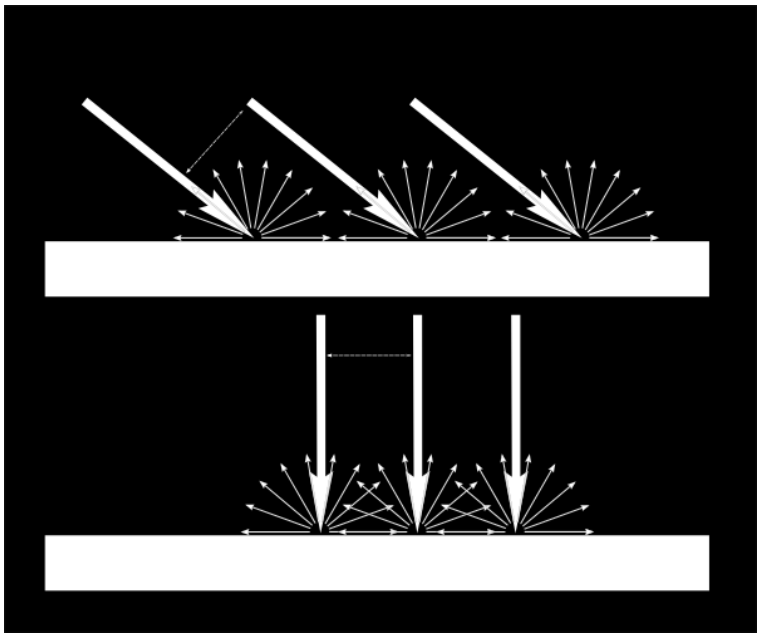
- Position : vecteur XYZ(W)
- Direction du spot : vecteur direction XYZ
- Intensité Diffuse : vecteur RGBA
- Intensité Spéculaire : vecteur RGBA
- Intensité Ambiante : vecteur RGBA

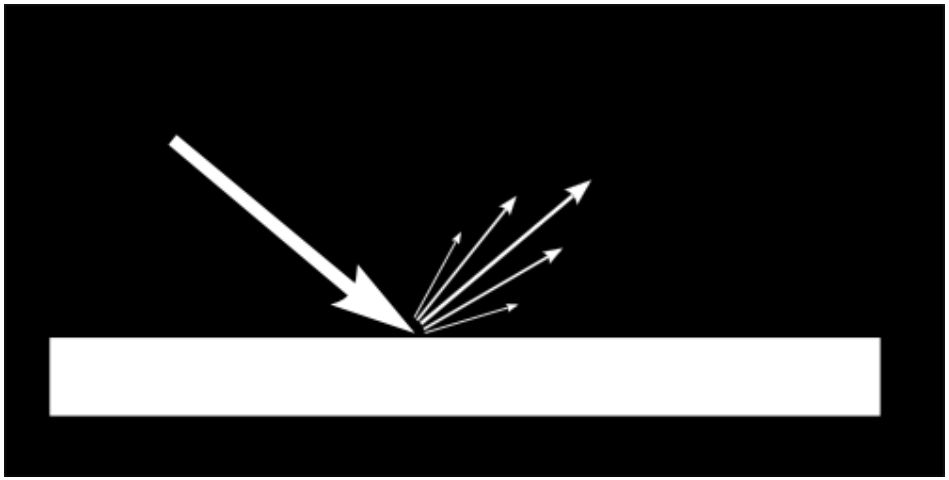


Lumières et Normales

Influence de la normale

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



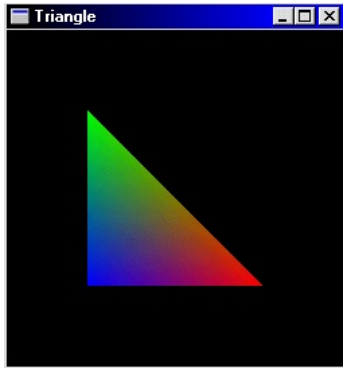


Il faut donner les normales à l'ordinateur : il ne sait pas les calculer...

Couleur et Textures

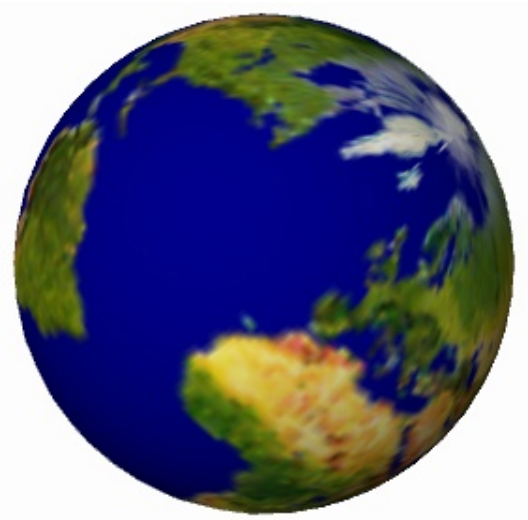
Colorier la géométrie

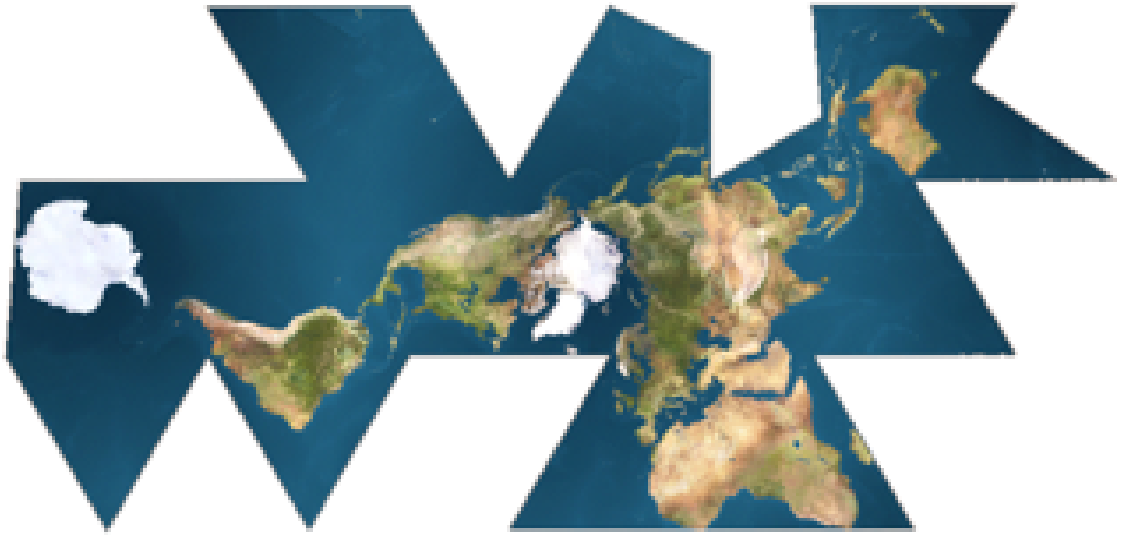
- Ce sont les sommets qui portent l'information de couleur ;
- Les faces sont coloriées par interpolation.



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)

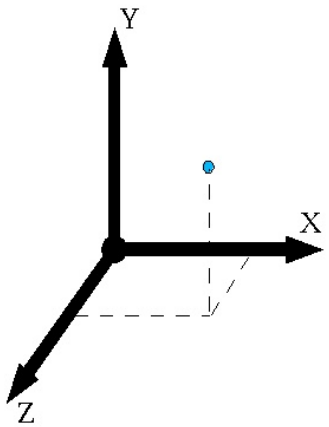
Les sommets

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

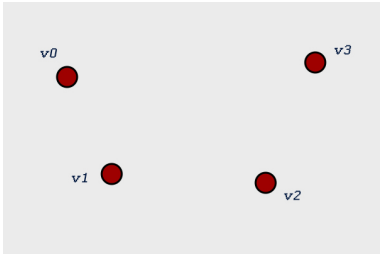
L'assemblage de sommets en faces se fait entre les instructions

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

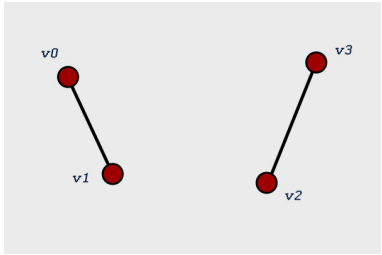
Nous verrons les autres déclarations plus loin.

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

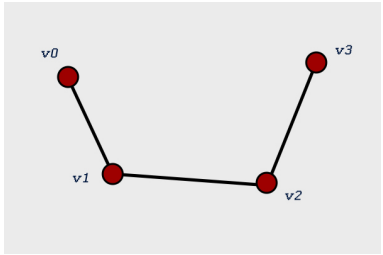
Primitives géométriques disponibles



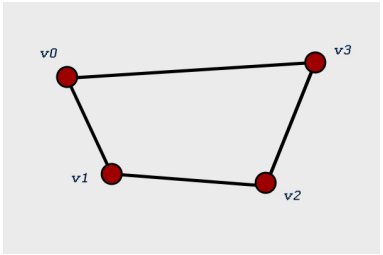
GL_POINTS



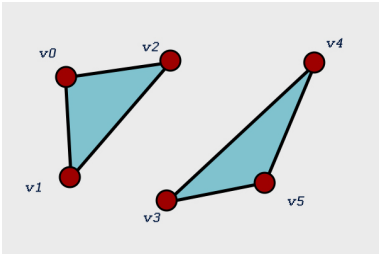
GL_LINES



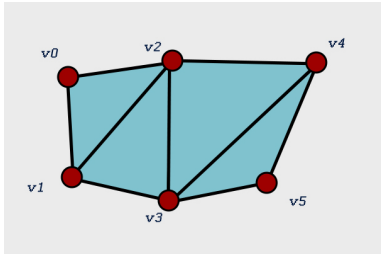
GL_LINE_STRIP



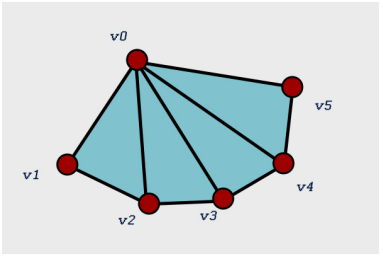
GL_LINE_LOOP



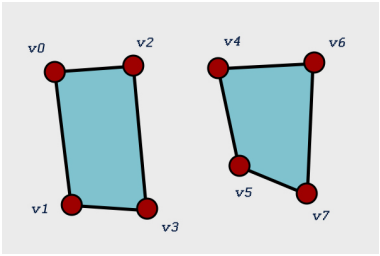
GL_TRIANGLES



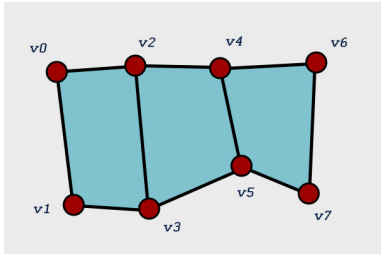
GL_TRIANGLE_STRIP



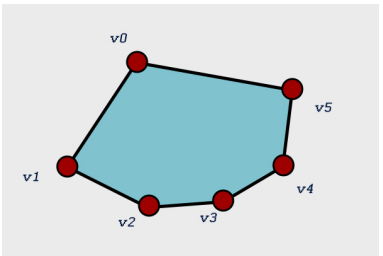
GL_TRIANGLE_FAN



GL_QUADS



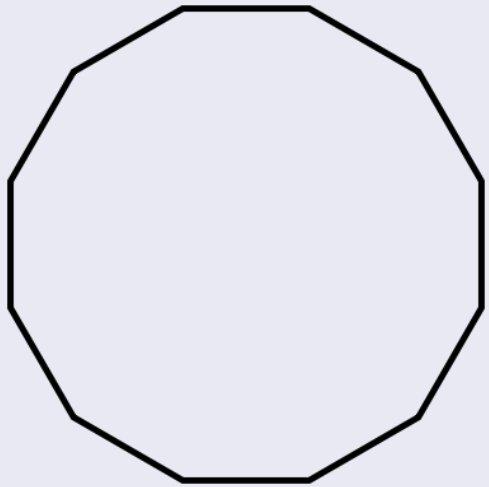
GL_QUAD_STRIP

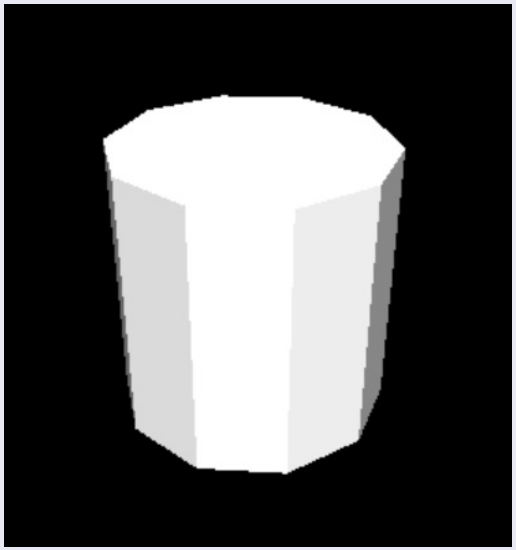


GL_POLYGON

Exemples simples de modélisation : cylindre

Cylindre





Réponse

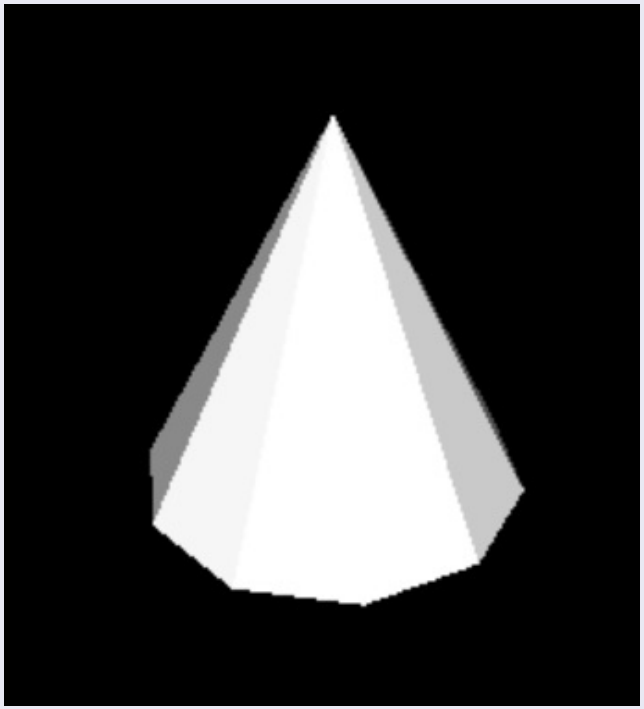
```
glBegin(GL_TRIANGLE_FAN);
... //base inferieure
glEnd();
glBegin(GL_TRIANGLE_FAN);
... //base superieure
glEnd();
glBegin(GL_QUAD_STRIP);
... //faces laterales
glEnd();
```

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)

glVertex3?(x,y,z)

glNormal3?(x,y,z)

Déclarer la normale au polygone que l'on trace.

Une normale déclarée est utilisée jusqu'à être remplacée

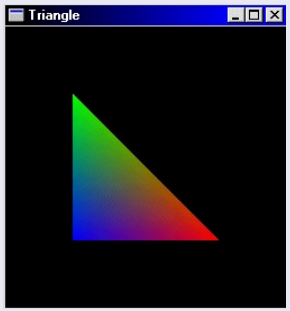
```
glBegin(GL_QUADS);
  glNormal3f(0,1,0);
    glVertex3f(1,0,1);
    glVertex3f(1,0,-1);
    glVertex3f(-1,0,-1);
    glVertex3f(-1,0,1);
glEnd();
```

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

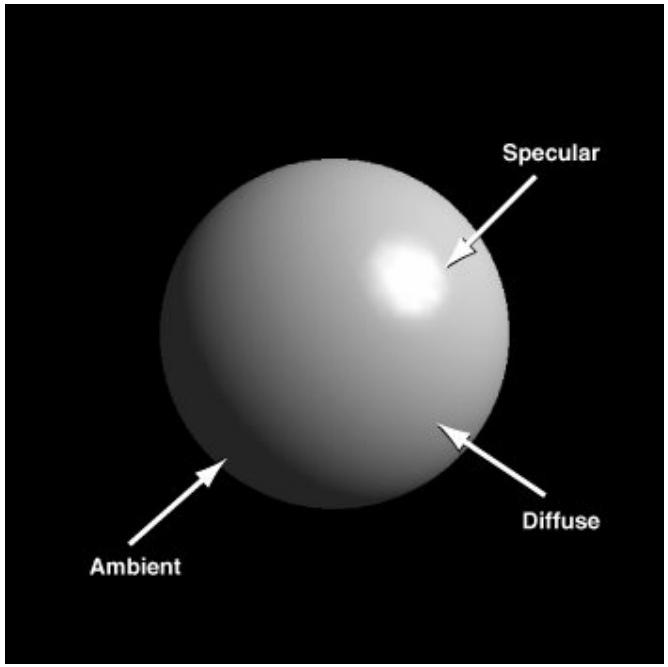
face détermine quelle face est calculée :

- 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 $[x\ y\ z]^T$

Modélisation, transformation et animation

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);
```

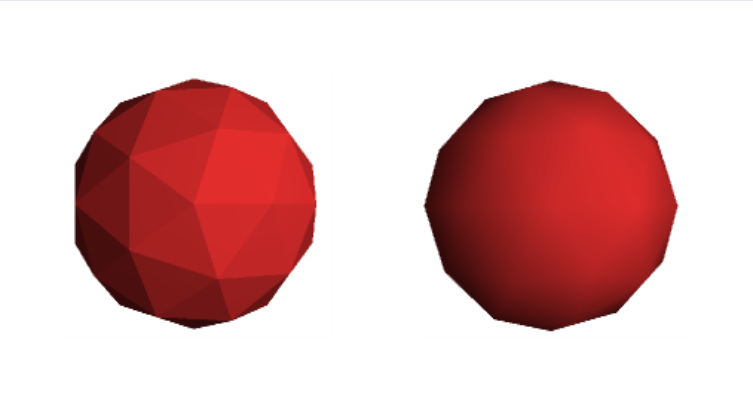
```
glPushMatrix();
   glRotatef(t,0,0,1);
   glTranslatef(4,3,1);
   ...
glPopMatrix();
```

L'éclairage dans OpenGL

Type d'ombrage :

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

- `GL_FLAT`
- `GL_SMOOTH`



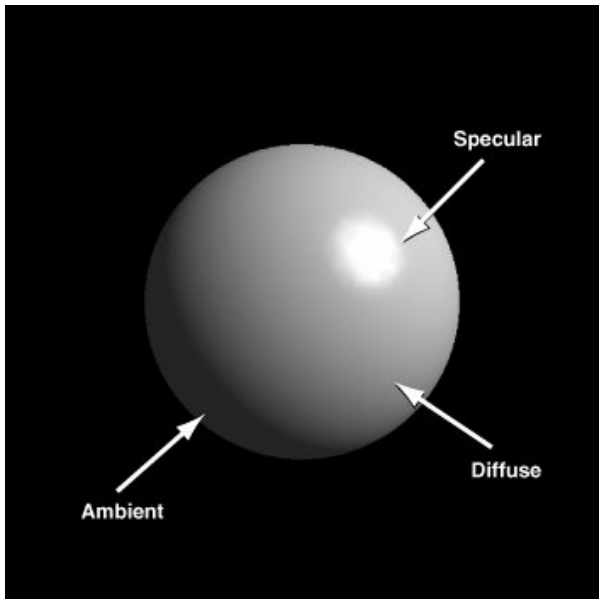
Le modèle d'éclairage dans OpenGL

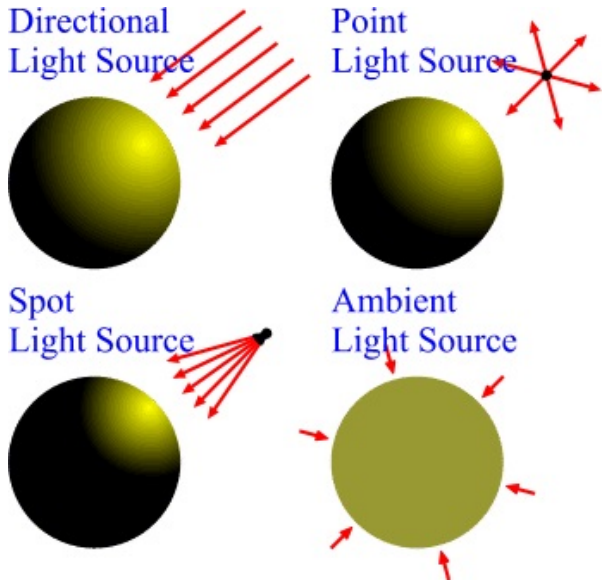
Composantes lumineuses

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

Types de sources lumineuses

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





Utiliser les lumières

Les lumières dans OpenGL

- OpenGL sait gérer jusqu'à 8 sources de lumières simultanément,
- Elles sont nommées `GL_LIGHT`i, 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)

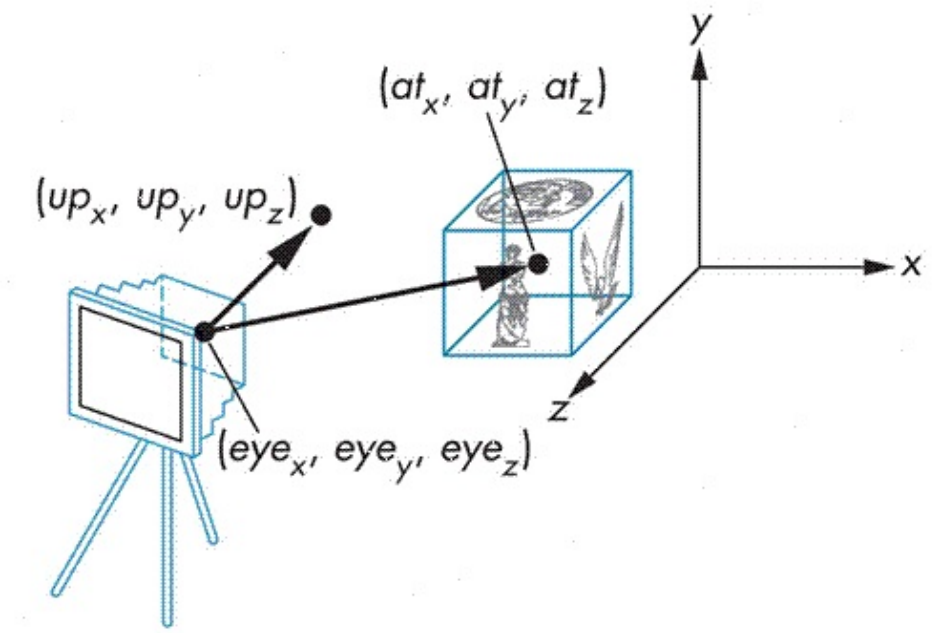
Offrir plus de contrôle sur la compréhension des paramètres lumineux :

- 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);`

La fonction Main

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.

TP :

Simulation d'une flotte de robots

Commandes principales

Eigen : :Vector3f v

- Vecteur de 3 floats x,y,z : Eigen : :Vector3f(x,y,z)
- Acces 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 : $\mathrm{u_{norm}}$ = u.normalized()
- Autre version (inplace) : u.normalize()

Produit vectoriel $u \wedge v$

- Calculé par u.cross(v)