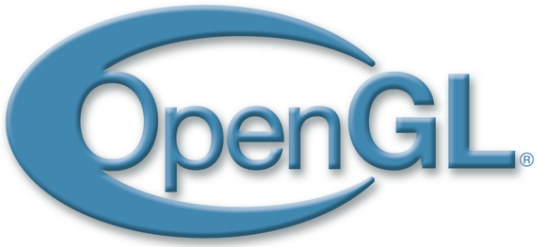
Introduction à OpenGL avec GLUT

Nikolas Stott[1] Hassan Bouchiba [2]

[1]LocalSolver

[2]Terra3d

21 mars 2019



Plan

Plan

Qui fait quoi ?

GLUT : haut niveau

Interface de programmation :

- Fenêtrage,
- Visualisation,
- Input-output utilisateur

OpenGL : bas niveau

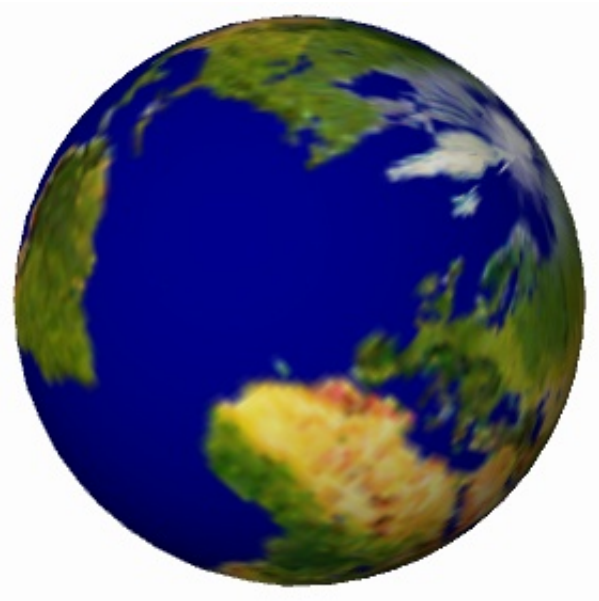
Interface avec le hardware graphique :

- Commandes de création de géométrie, de couleurs
- Manipulation des objets

Qu'est ce qu'OpenGL sait faire ?

Modélisation/Visualisation

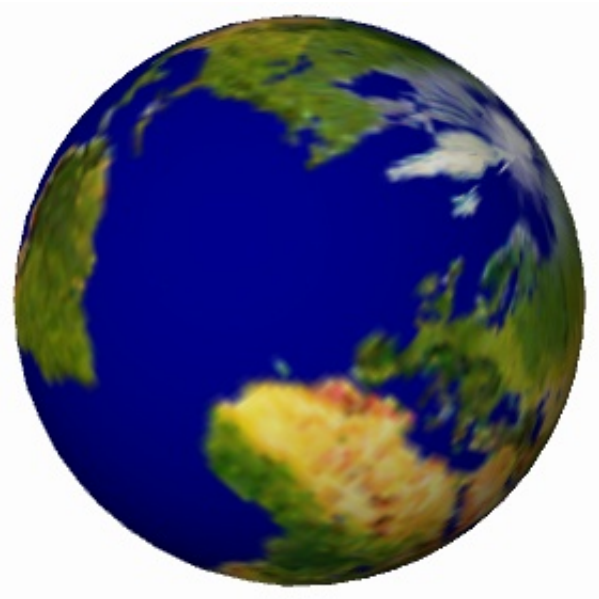
- Création de géométries complexes
- Habillage de la géométrie : couleur, texture, éclairage...

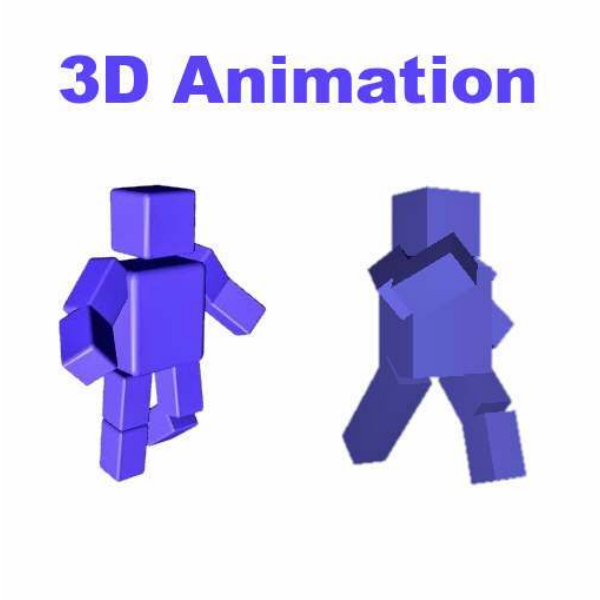


Qu'est ce qu'OpenGL sait faire ?

Modélisation/Visualisation

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





Animation

OpenGL permet d'animer :

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

Plan

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.

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...

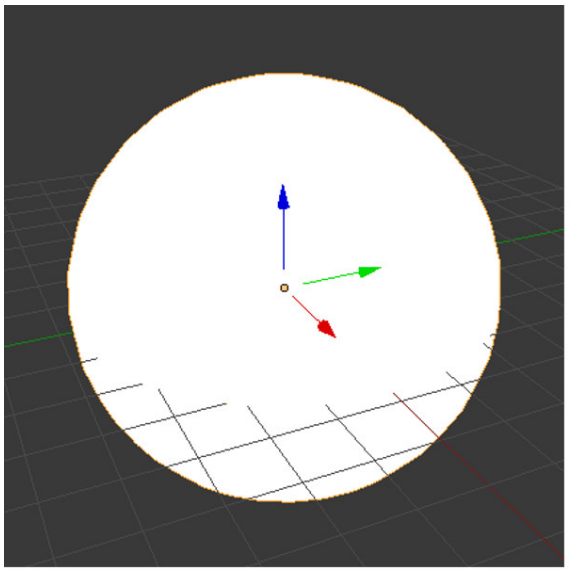
C'est tout ?

Ce n'est pas suffisant !

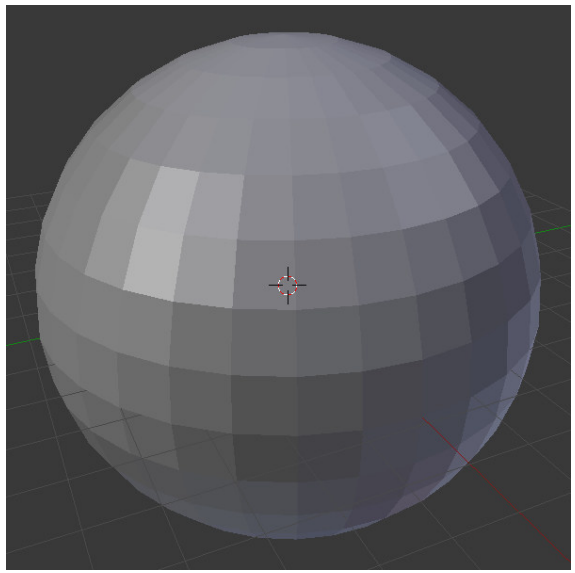
Il faut également déclarer des informations supplémentaires sur l'objet :

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

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



Sans aucune information :



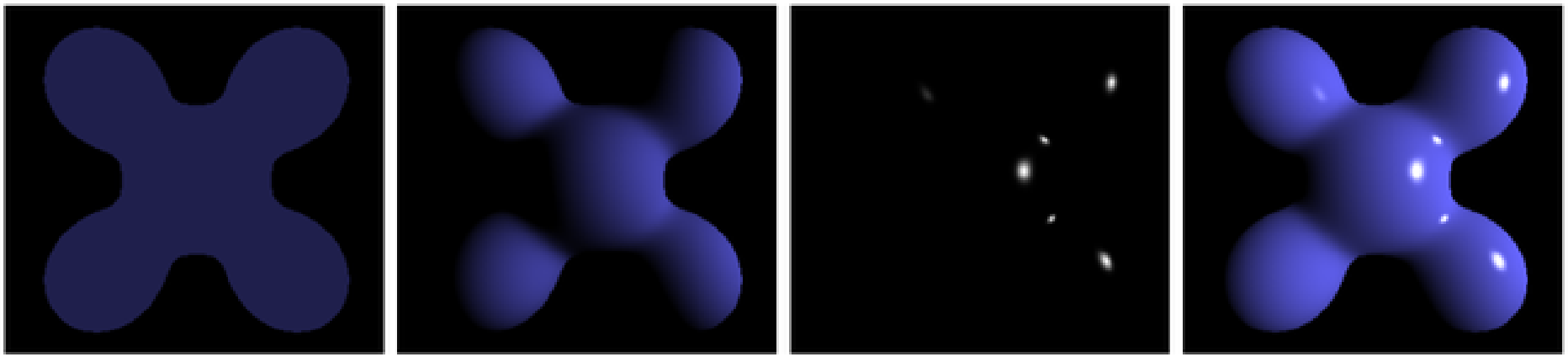
Avec orientation et matériaux :

Pas votre lampe habituelle

Les composantes lumineuses du modèle de Phong

Une lumière OpenGL est caractérisée par 5 paramètres :

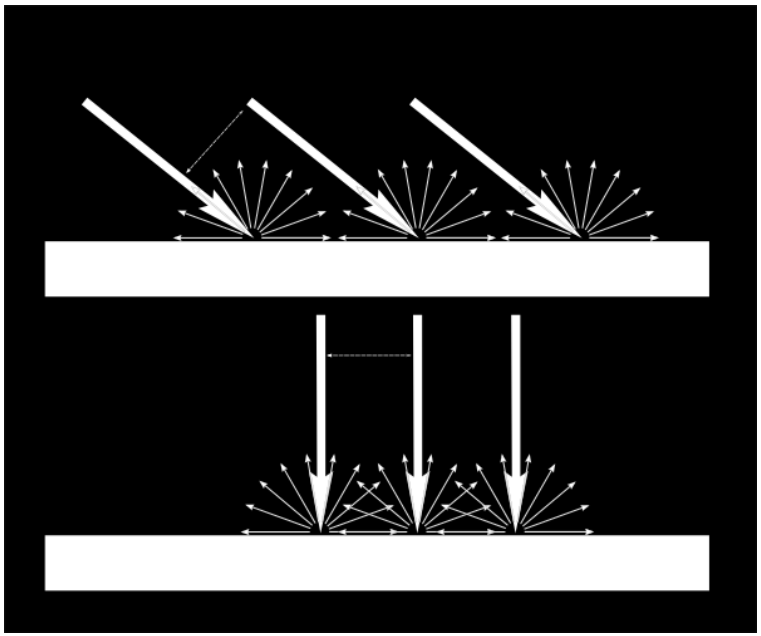
- Position : vecteur XYZ(W)
- Direction du spot : vecteur direction XYZ
- Intensité Diffuse : vecteur RGBA
- Intensité Spéculaire : vecteur RGBA
- Intensité Ambiante : vecteur RGBA

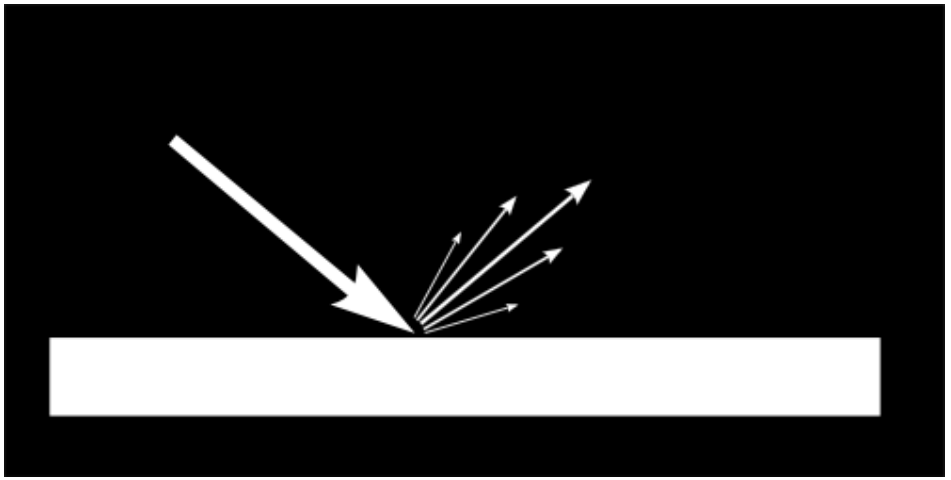


Lumières et Normales

Influence de la normale

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



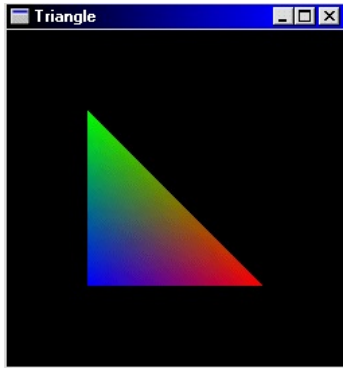


Il faut donner les normales à l'ordinateur : il ne sait pas les calculer...

Couleur et Textures

Colorier la géométrie

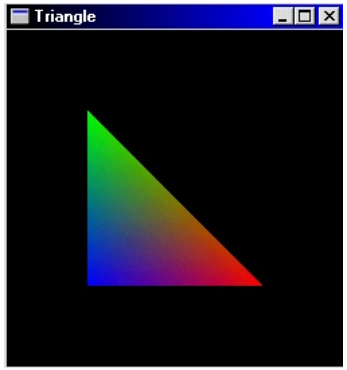
- Ce sont les sommets qui portent l'information de couleur ;
- Les faces sont coloriées par interpolation.



Couleur et Textures

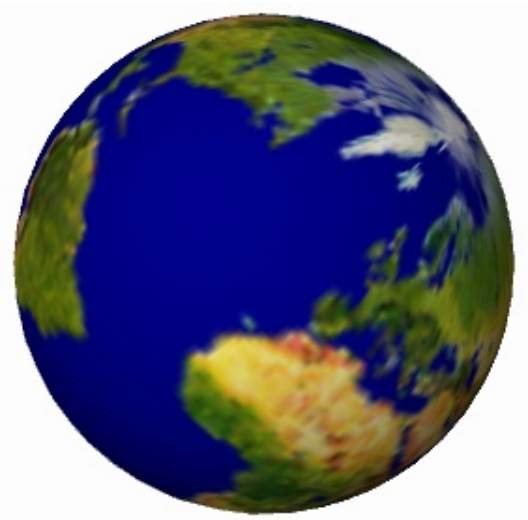
Colorier la géométrie

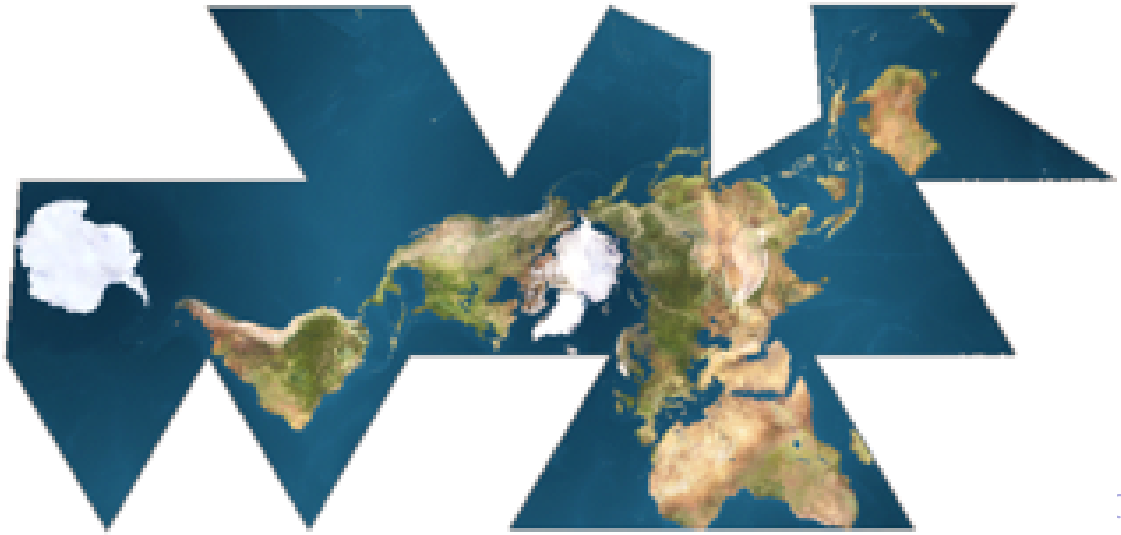
- Ce sont les sommets qui portent l'information de couleur ;
- Les faces sont coloriées par interpolation.



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, ...

Plan

Les briques élémentaires (1)

Les sommets

Les sommets sont déclarés par la commande `glVertex3f(x,y,z)`.

Exemples :

- `glVertex3f(2.4f,4.1f,-0.1f);`

Les briques élémentaires (2)

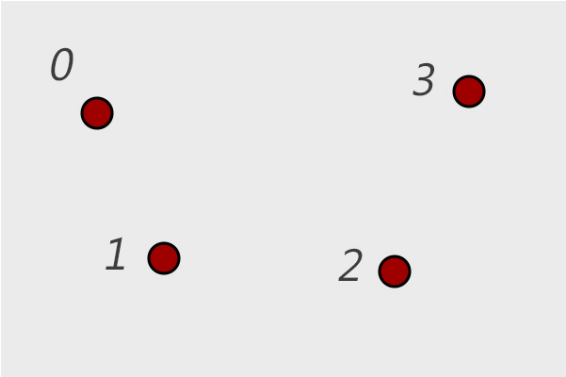
La déclaration de primitive

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

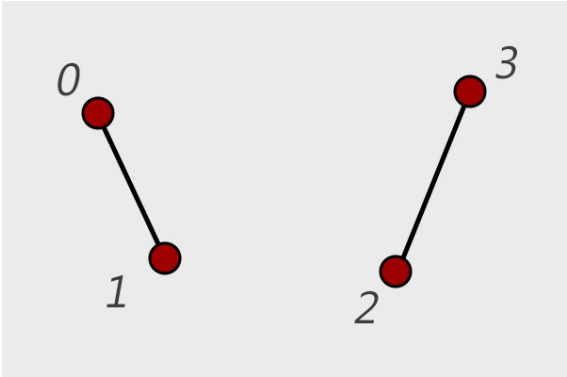
```
glBegin(TYPE_DE_LA_PRIMITIVE);
  //declarations optionnelles
  glVertex3f(...);
  ...
  glVertex3f(...);
  //declarations optionnelles
glEnd();
```

Nous verrons les autres déclarations plus loin.

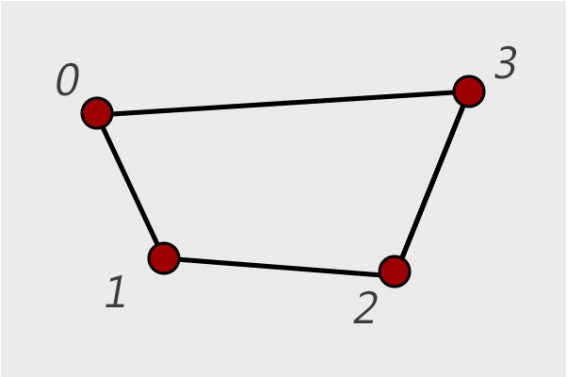
Primitives géométriques disponibles



GL_POINTS

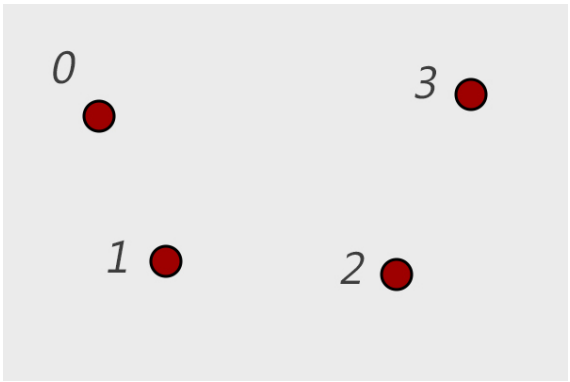


GL_LINES

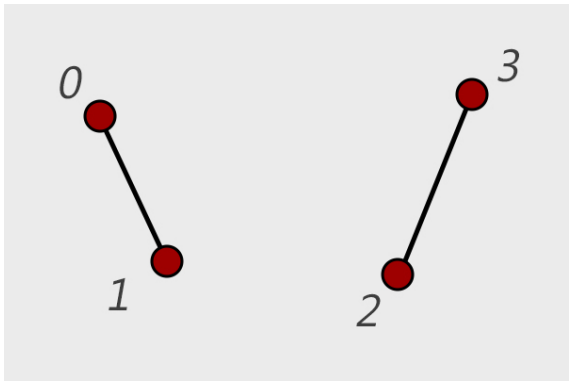


GL_LINE_LOOP

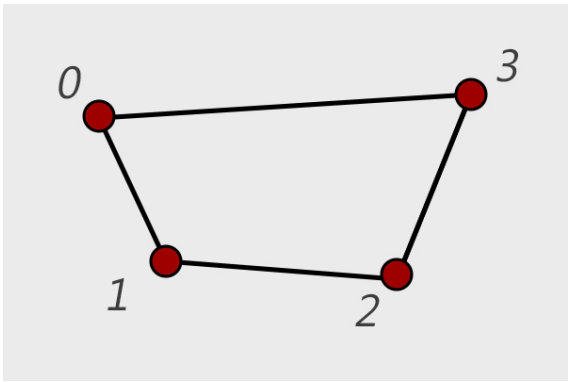
Primitives géométriques disponibles



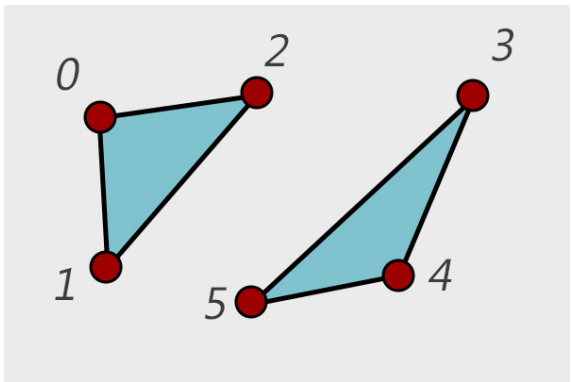
GL_POINTS



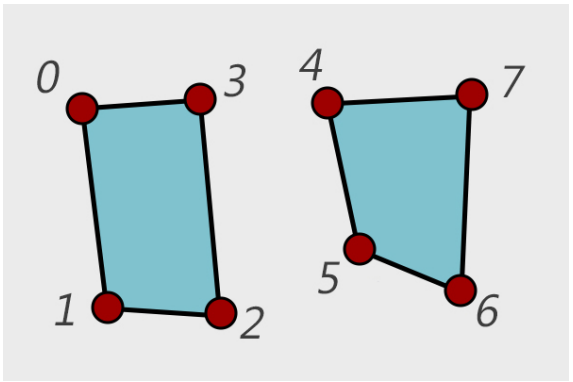
GL_LINES



GL_LINE_LOOP



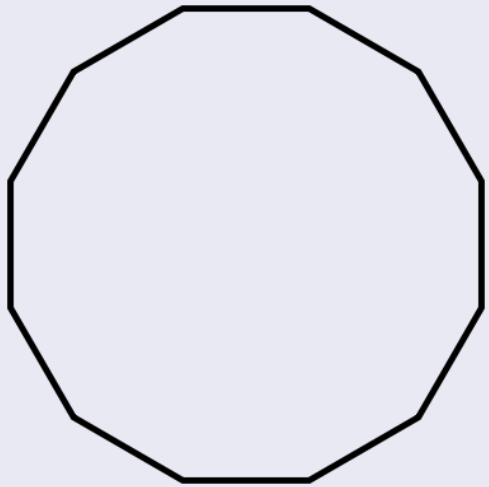
GL_TRIANGLES

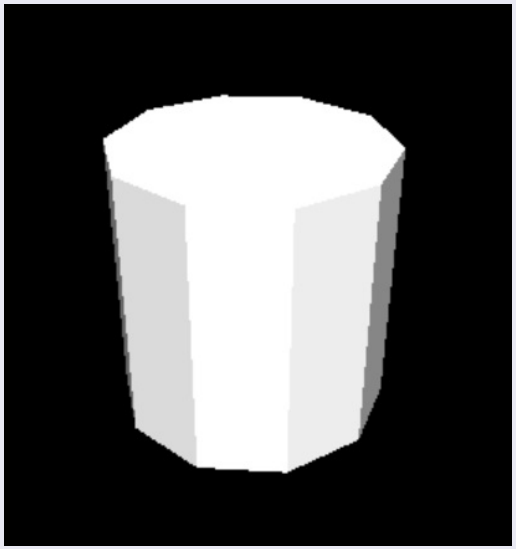


GL_QUADS

Exemples simples de modélisation : cylindre

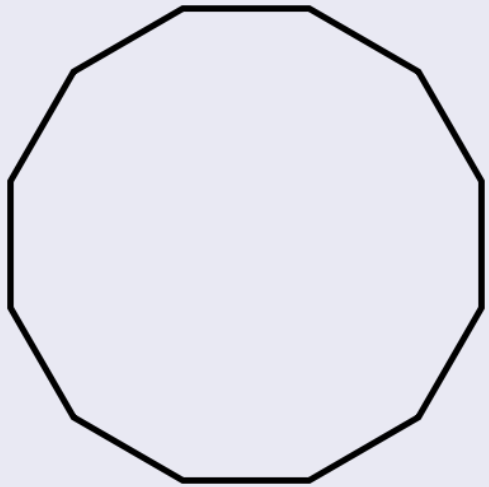
Cylindre

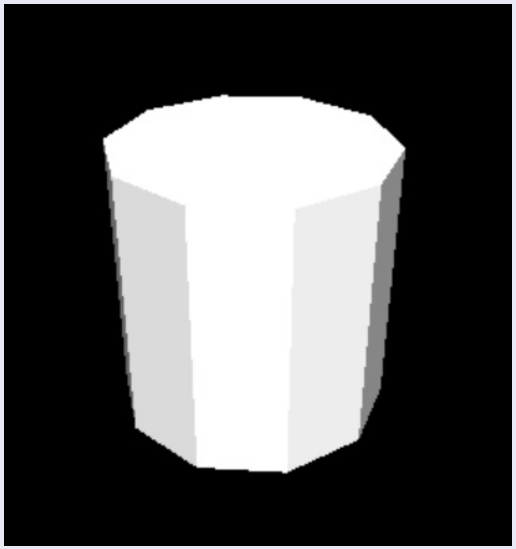




Exemples simples de modélisation : cylindre

Cylindre



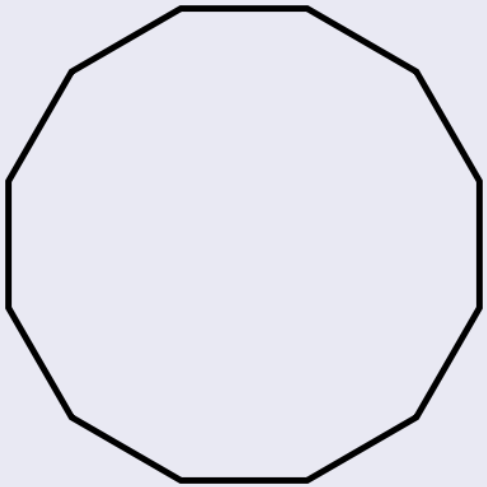


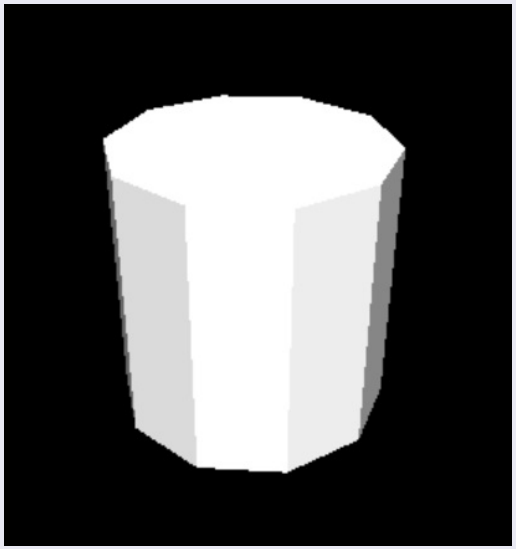
Réponse

```
for(int i=0; i<p; ++i){
  glBegin(GL_TRIANGLES);
  ... //base inferieure
  glEnd();
  glBegin(GL_TRIANGLES);
  ... //base superieure
  glEnd();
  glBegin(GL_QUADS);
  ... //faces laterales
  glEnd();
}
```

Exemples simples de modélisation : cylindre

Cylindre





Réponse

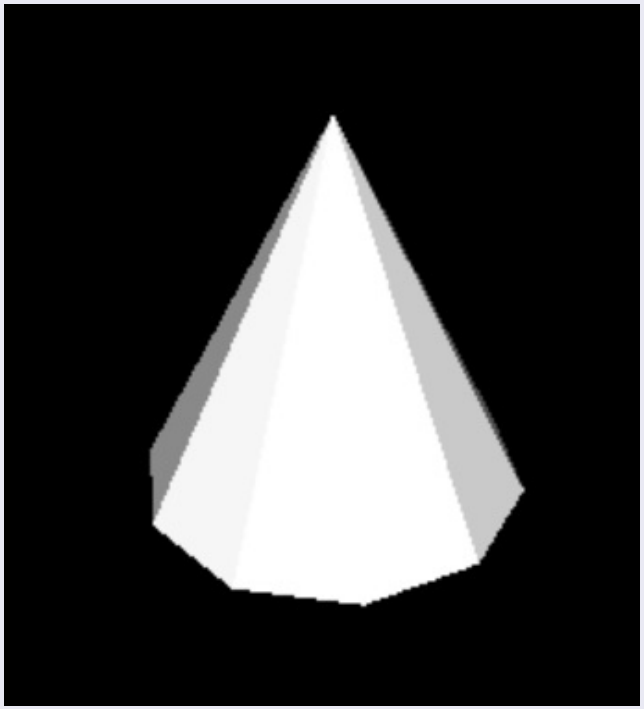
```
for(int i=0; i<p; ++i){
  glBegin(GL_TRIANGLES);
  ... //base inferieure
  glEnd();
  glBegin(GL_TRIANGLES);
  ... //base superieure
  glEnd();
  glBegin(GL_QUADS);
  ... //faces laterales
  glEnd();
}
```

Une autre solution

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

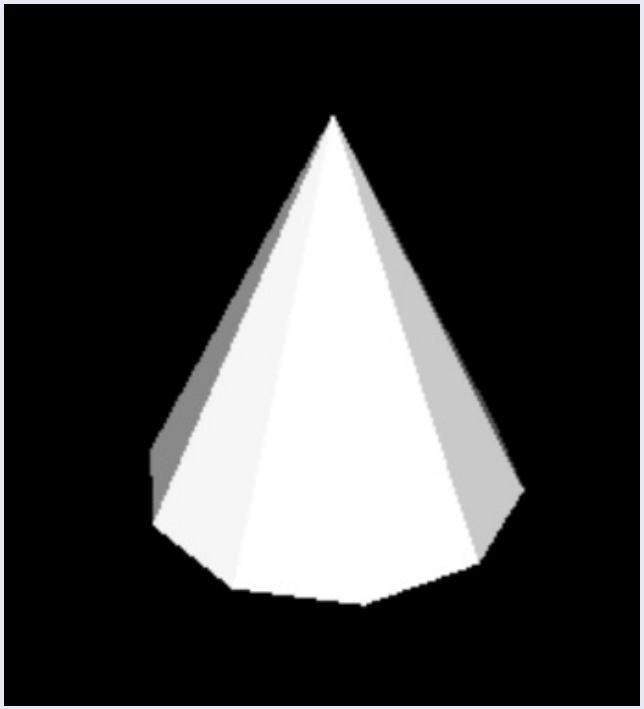
Exemple de modélisation : cône

Cône



Exemple de modélisation : cône

Cône

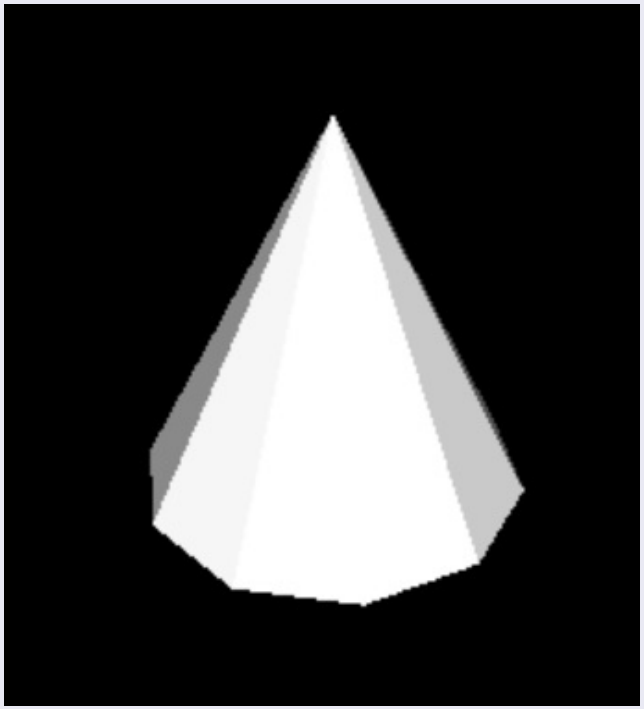


Réponse

```
for(int i=0; i<p; ++i){
    glBegin(GL_TRIANGLES);
    ... //face inferieure
    glEnd();
    glBegin(GL_TRIANGLES);
    ... //faces laterales
    glEnd();
}
```

Exemple de modélisation : cône

Cône



Réponse

```
for(int i=0; i<p; ++i){
    glBegin(GL_TRIANGLES);
    ... //face inferieure
    glEnd();
    glBegin(GL_TRIANGLES);
    ... //faces laterales
    glEnd();
}
```

Une autre solution

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

OpenGL est une machine à états

Deux nouvelles commandes

- glNormal3f
- glColor3f

Machine à états

Executer `glColor3f(1,1,1)` choisit la "couleur courante". Tout sera dessiné avec cette couleur jusqu'à ce que la couleur soit changée par une nouvelle exécution de `glColor()`.

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.

Une normale déclarée est utilisée jusqu'à être remplacée

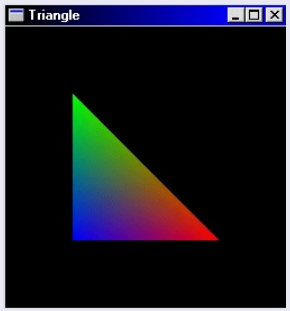
```
glBegin(GL_QUADS);
   glNormal3f(0,1,0);
      glVertex3f(1,0,1);
      glVertex3f(1,0,-1);
      glVertex3f(-1,0,-1);
      glVertex3f(-1,0,1);
glEnd();
```

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

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

glColor3f(r,g,b)

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);
  glNormal3f(0, 0, 1);
    glColor3f(1, 0, 0);
      glVertex3f(,,);
    glColor3f(1, 1, 0);
      glVertex3f(,,);
    glColor3f(1, 1, 1);
      glVertex3f(,,);
glEnd();
```

Récupération du TP

Dans un terminal :

- git clone https ://github.com/n-stott/mines.git

Architecture

- Boids :
  - le gros TP (plus tard)
- Canvas :
  - exercice de cours (maintenant)
- Supp :
  - supports de cours

Compilation

Dans un terminal, dans le dossier de l'exercice/du TP, exécuter :

- sh build.sh

Les transformations élémentaires

Principe de transformation

Chaque commande de transformation transforme le repère de dessin local.

Translation

`glTranslatef(x,y,z)`

Changement d'échelle

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

Rotation autour d'un axe

`glRotatef(t,x,y,z)` effectue une rotation de t (en degrés) 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 ?

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);
```

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 :

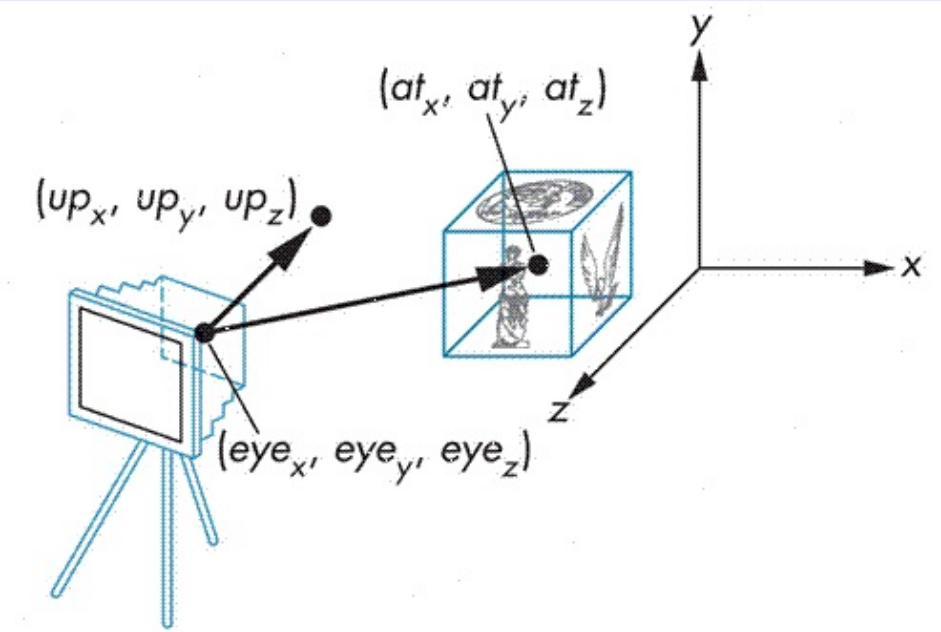
```
glRotatef(t,0,0,1);
glTranslatef(4,3,1);
...
glTranslatef(-4,-3,-1);
glRotatef(-t,0,0,1);
```

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

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);`

Plan

TP :

Simulation d'une flotte de robots

Plan

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)