Projet de Java
SolidVision
Ingénieurs2000 – IR1

forax@univ-mlv.fr, pfinkel@cantor.fr

But

Le but de ce projet est d'afficher dans un fenêtre une scène graphique 3D décrite dans un fichier XML suivant un format bien précis.

Ce projet est découpé en deux parties.
Ce document décrit uniquement la première partie.

Première partie du projet

Le but de cette première partie est de préparer la création du programme
que l'on vous demandera d'effectuer dans la seconde partie.
Pour la première partie, vous devez fournir un document contenant les informations suivantes :

- une introduction présentant un peu votre vision du projet
- une explication de comment installer et exécuter un exemple en utilisant la library JOGL – JSR231 sous linux
- Une description de l'architecture de votre projet
  C'est-à-dire une description en français de l'ensemble des classes[1] (interfaces, etc.) que vous allez devoir implanter pour votre projet.
  Le document devra contenir une description de chaque classe en indiquant :
    - son rôle/sa responsabilité
    - ses relations d'héritage ou d'implémentation avec les autres types
    - les champs de la classes
    - les méthodes de celle-ci

  faites en sorte que l'ordre dans lequel vous décrivez les classes soit logiques (donc pas alphabétique)
  - une description de comment va se faire l'affichage de la scène en indiquant les créations d'objets et les appels de méthodes
- Des exemples de codes de test commentés effectuant :
  - le parsing d'un fichier XML en utilisant un parseur SAX
  - et en OpenGL :
    - l'affichage d'un cube avec une couleur ambiante verte
    - l'affichage du même cube mais tourner de 45 degrés
    - l'affichage d'un cube en utilisant une lumière et des couleurs ambiante, diffuse et séculaire
    - l'affichage d'un plan (optionel)
    - l'affichage d'un cube avec une texture 256x256 (optionel)

  Les tests ne doivent pas être des copier/coller d'exemple sur internet mais de vrai exemple ou chaque ligne (non évidente) est commentée et ou seul ce qui est nécessaire est présent.

1Pour vous donnez une idée, j'en ai trouvé douze.

- Une conclusion

  Je vous conseil fortement de faire les tests avant de vous lancez dans la création de l'architecture.

Le format XML de SolidVision

Ce format est le même à l'ensemble de vos projet, vous devez donc porter un soin particulier pour en respecter la syntaxe et la sémantique (son sens).
Ce fichier décrit une scène contenant

- des objets (`cube`, `plane`)
  - ayant un identifiant unique (`id`)
  - placé dans la scène (`location`)
  - ayant des couleurs ambiantes, diffuses, spéculaires etc.
  - possédant une texture (optionnel)

  ces objets possèdent de plus une ou plusieurs balises qui leur sont propres, par exemple cube possède une balise `dimension` qui indique la taille du cube et plane une balise `axis` qui indique sur quel axe se situe le plan
- des lumières (`light`)
  - placé dans la scène (`location`)
  - ayant une direction, un spot-exponent et un spot-cutoff
  - ayant des intensités ambiantes, diffuses, spéculaires.
- une camera
  - placé dans la scène (`location`)
  - ayant une largeur et une hauteur correspondant à la fenêtre de rendu de l'application
- des animateurs (optionnels) permettant de faire bouger un objet (idref indique l'identifiant de l'objet) qui doit subir une transformation (`rotate`, `translate`, `scale`), et ce 30 fois par seconde.

Les parties optionnelles ne doivent pas forcément être traitée par votre programme mais celui-ci ne doit pas s'arrêter s'il rencontre une balise optionnelle mais simplement l'ignorer.

Un exemple complet :

```
<?xml version="1.0"?>
<scene>
 <cube id="cube1">
  <location x="2" y="3" z="4"/>
  <dimension dx="1" dy="1" dz="1"/>
  <color>
    <material kind="ambiant" red="0.2" green="0.2" blue="0.2"/>
    <material kind="diffuse" red="0.5" green="1.0" blue=""0/>
  </color>
  <texture file="liquid-metal.jpg"/>
 </cube>

 <plane id="plane1">
  <location x="0" y="0" z="2"/>
  <axis coord="z"/>
  <color>
    <material kind="ambiant" red="0.2" green="0.2" blue="0.7"/>
  </color>
  <texture file="liquid-metal.jpg"/>
 </plane>
```

```
 <light>
  <location x="5" y="5" z="5"/>
  <direction x="0" y="0" z="0"/>
  <spot-exponent intensity="64"/>
  <spot-cutoff spread-angle="90"/>
  <color>
    <material kind="ambiant" red="1.0" green="0.2" blue="0.2"/>
    <material kind="diffuse" red="1.0" green="0.2" blue=""0.2/>
  </color>
 </light>

 <camera>
   <dimension width="800" height="600"/>
   <location x="0" y="0" z="-12" />
 </camera>

 <animator idref="cube1">
  <rotation angle="1" x="1" y="0" z="0"/>
 </animator>
</scene>
```

OpenGL

OpenGL est une spécification sous forme de bibliothèque développée en C puis porté dans différents langages (C++, Java, C#). Cette bibliothèque permet principalement d'envoyer des ordres d'affichages à la carte 3D. Si celle-ci ne peut effectuer le travail, une partie logiciel émulera (en moins efficace) le travail à effectuer.

Nous utiliserons cette bibliothèque pour afficher la scène en 3D.
Les appels en OpenGL sont de deux sortes, les premiers déclaratifs permettent de changer le contexte courant (comme `glEnable()` par exemple) mais ne produise rien.
Le code qui effectue un affichage se situe en un `glBegin()` et un `glEnd()`.
Il n'est pas possible de faire des appels déclaratif entre un `glBegin()` et un `glEnd()`, il faut le faire avant.

OpenGL et Java

Il existe plusieurs bibliothèques permettant de faire de la 3D en Java, Java3D, LGWJ et JOGL. Le port officiel de la bibliothèque OpenGL est basé sur JOGL et à pour nom JOGL – JSR 231. Pour ceux qui chez eux ont autre chose que linux, JOGL marche aussi sur Mac, la configuration sera quasi identique pensez juste que pour la soutenance de la deuxième partie nous testerons votre logiciel sur un linux (dans une des salles de TP).

En OpenGL il existe des méthodes canonique, comme par exemple glVertex() qui permet d'indiquer un point, pour lesquelles il existe plusieurs versions, dans notre exemple, une version qui prend des entiers glVertex3i(), une version qui prend des flottants glVertex3f() et une qui prend des doubles glVertex3d(). La version qui prend des flottants est souvent la méthode qui offre le meilleur compromis (au moins pour les cartes 32 bits).

JOGL à la Fac

On ne pas pas utiliser direcement les binaires JOGL à la fac car la libc est un peu veille et le serveur X est n'est toujours pas un xorg. Pour cela, je vous est compilé une version

spécifique[2] que vous trouverez ici :
http://www-igm.univ-mlv.fr/~forax/ens/java/ir06-07/jogl-linux-fac.zip

Les Facettes

OpenGL ne connaît pas la notion d'objet, pour la carte 3D tous se fait à base de facettes ; triangles ou quads (facette à 4 points). Les facettes sont définies en indiquant chacun de leurs points avec la méthode `glVertex3f()`. Ces points doivent être entre un `glBegin()` et un `glEnd()`. Le `glBegin()` indiquant le type de figure.

Par exemple pour un triangle, on aura :
```
glBegin(GL_TRIANGLE)
  glVertex3f(x1,y1,z1);
  glVertex3f(x2,y2,z2);
  glVertex3f(x2,y2,z2);
glEnd();
```

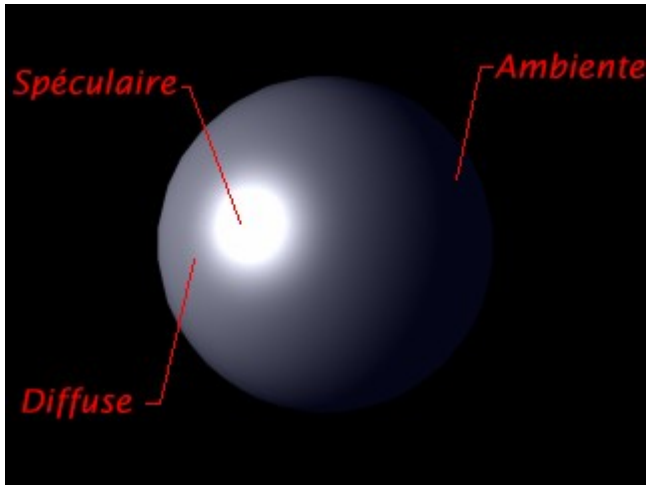
avec (x1,y1,z1), (x2,y2,z2) et (x3,y3,z3) les 3 points du triangle.
Il est possible de définir plusieurs facettes à l'intérieur d'un même `glBegin()`, `glEnd()`.

Ne pas voir les objets cachés

Comme les objets sont en 3D, certain objet sont devant les autres,
OpenGL possède un mécanisme qui permet de ne garder que les pixels des facettes les plus proche, pour cela il faut activer le buffer de profondeur
`glEnable(GL_DEPTH_TEST);`
et le ré-initialiser avec
`glClear(GL_DEPTH_BUFFER_BIT);`

Il faut faire de même avec le buffer stockant les couleurs
`glClear(GL_COLOR_BUFFER_BIT);`
Noter que `glClearColor()` permet d'indiquer une couleur de fond

Les couleurs



OpenGL permet de définir plusieurs couleurs différentes :

- la couleur ambiante qui est une couleur indépendante de toute source de lumière.
- La couleur d'émission est indépendante d'une lumière est correspond à la couleur émise par l'objet lui-même.
- La couleur diffuse qui est une couleur que renvoie un objet lorsqu'il est éclairer par une lumière. L'intensité de cette lumière dépend de l'angle entre le vecteur normal de la face et le vecteur entre le vertex et la position de la lumière, de l'intensité de la lumière. Il faudra donc définir ce vecteur (cf plus loin)
- La couleur spéculaire est la couleur de la lumière qui se reflète directement dans l'objet. Elle aussi dépend du vecteur normal

2 Je ne pas compilé les extensions OpenGL par flemme.

`glMaterial()` permet d'indiquer quel type de couleur on veut changer ainsi que la nouvelle couleur. Cette méthode doit être appelée avant un appel à `glVertex()`. Attention il existe aussi `glColor3f()` mais elle permet de remplace la couleur ambiante et diffuse par la même couleur.

Normale

L'intensité lumineuse de la lumière diffuse et spéculaire dépend de la position de la lumière par rapport au vecteur normal de la facette.
Malheureusement, OpenGL ne calcul pas ce vecteur tous seul, il faut donc lui indiquer avec la méthode `glNormal()`.

En savoir plus:
http://www-evasion.imag.fr/Membres/Antoine.Bouthors/teaching/opengl/opengl6.html

Lumières

OpenGL permet de définir des lumière numéroté (0 à GL_MAX_LIGHT-1) ayant chacune une intensité ambiante, diffuse et spéculaire.
Premièrement il faut dire que l'on veux le calcul des lumières avec `glEnable(GL_LIGHTING);`
Puis il faut activer chaque lumière (de GL_LIGHT0 à GL_LIGHT0+GL_MAX_LIGHT-1) avec `glEnable(GL_LIGHT0)`. Il est possible de définir le mode d'éclairage avec `glLightModel()`. L'ensemble des propriétés d'une lumière est modifiable avec les méthodes `glLightf()` et `glLightfv()`.

En savoir plus,
http://www-evasion.imag.fr/Membres/Antoine.Bouthors/teaching/opengl/opengl5.html

Les Textures (optionnel)

L'idée consiste à pouvoir appliquer une texture c'est à dire une image sur les objets créés.
La première chose à faire est d'activer le mode texture `glEnable(GL_TEXTURE_2D)`.

Il faut ensuite créer une texture avec `glGenTextures()`, cette méthode permet d'obtenir l'identifiant représentant la texture.

Associer une texture au contexte courant se fait avec la méthode `glBindTexture()`, la texture sera alors utiliser par toutes les facettes définies par `glBegin()`/`glEnd()`.

Pour modifier une texture, il faut utiliser `glTexImage2D()` qui modifie les données de la texture associée au contexte courant. Par défaut, il faut que la largeur et la longueur soient des puissances de 2.

Il faut ensuite indiquer comment se comporte la texture en cas d'agrandissement ou de réduction de l'image de la texture avec `glTexParameter()`.
Si vous voulez un résultat qui ne soit pas trop pixelisé, il faut utiliser un sampling au moins linéaire.

Enfin il faut associer les coordonnées de la texture aux points des facettes en appelant `glTexCoord2f()` avant `glVertex()`. Les coordonnées d'une texture sont des valeurs en 0.0 et 1.0 indiquant quelle partie de la texture est associé à quelle partie de la facette.

En savoir plus:
http://www-evasion.imag.fr/Membres/Antoine.Bouthors/teaching/opengl/opengl7.html

Cette partie est optionnelle dans le projet sinon vous pouvez aussi vous limiter à un seul type de texture (disons ARGB) et au image ayant une taille en puissance de 2.

Les Matrices

Il existe plusieurs sortes de matrice en OpenGL, nous n'en utiliseront que deux :

- les matrices de transformation (`GL_MODELVIEW`) qui permettent d'effectuer des translations, rotation etc.
- la matrice de projection (`GL_PROJECTION`) qui indique comment transformer une scène 3D en une image 2D

Il faudra donc indiquer le mode des matrices avant d'effectuer des opération sur celle-ci avec `glMatrixMode();`

Effectuer des transformations en OpenGL se fait en appliquant une matrice sur le contexte courant. La matrice est un matrice 4x4 en coordonnée homogène (http://fr.wikipedia.org/wiki/Coordonn%C3%A9es_homog%C3%A8nes)

La transformation décrite par la matrice sera appliquée à l'ensemble des points (vertex) qui seront définie postérieurement.

- Initialisation avec la matrice identité
  `glIdentity()`

- Appliquer une translation,rotation, homothétie sur la matrice
  `gltranslate()`, `glRotate()`, `glScale()`.

- Sauvegarder l'état d'une matrice
  Il est possible de sauvegarder l'état d'une matrice dans un tableau de 16 flottants pour pouvoir la réutiliser plus tard avec `glGetFloat()` en passant en paramètre `GL_MODEL_MATRIX`.
  Il est possible de recharger cette matrice en utilisant `glLoadMatrix()`.

- Empiler/Depiler une matrice
  Il est possible d'empiler (`glPushMatrix()`) une nouvelle matrice qui sera celle modifiée par les opérations sur les matrices puis de dépiler (`glPopMatrix()`) celle-ci pour ré-obtenir l'ancienne matrice. Les valeurs de la nouvelle matrice empilée sont des copies des valeurs de l'ancienne.

En savoir plus:
http://www-evasion.imag.fr/Membres/Antoine.Bouthors/teaching/opengl/opengl3.html

Projection

La projection est l'étape qui consiste à passer de la 3D à la 2D,

comme pour les transformations, OpenGL utilise une matrice pour effectuer la projection. En règle générale, on ne rentre pas les coordonnées de cette matrice directement, on préfère passer par des méthodes de plus haut niveau de la bibliothèque GLU qui elle remplisse cette matrice. Par exemple, `gluPerspective() ou gluOrtho2D()`.

Ouvrir un fenêtre en OpenGL avec JOGL

```
// création de la zone de dessin
GLCanvas canvas=new GLCanvas();
canvas.addGLEventListener(new GLEventListener() {
  public void displayChanged(GLAutoDrawable drawable, boolean modeChanged,
      boolean deviceChanged) {
    // do nothing
  }

  public void init(GLAutoDrawable drawable) {
    // on effectue ici les initialisations et les déclarations
  }

  public void reshape(GLAutoDrawable drawable, int x, int y, int width,
      int height) {

    // appelé lorsque la fenêtre s'agrandit,
    // on initialise la projection ici
  }

  public void display(GLAutoDrawable drawable) {
    // on effectue l'affichage proprement dit ici
  }
});

// création de la fenêtre
JFrame frame=new JFrame();
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.add(canvas);
//frame.setResizable(false);
frame.setSize(800, 600);
frame.setVisible(true);
```

Conditions de rendu

Le projet est à faire par binome (les mêmes pour la partie 1 et pour la partie 2).
Le document concernant la première partie est à envoyer par mail aux adresses (notez le pluriel) indiquées au début de ce document au plus tard le vendredi 27 avril 23h59 au format PDF.

Références :

Eclipse :

www.eclipse.org
http://www.forax.org/ens/java-avance/cours/pdf/Eclipse%20pour%20les%20null.pdf

Netbeans :

www.netbeans.org

JOGL - JSR 231 :

https://jogl.dev.java.net/

OpenGL Référence

http://opengl.org/sdk/docs/man/

Exemples de prog en OpenGL

http://www-evasion.imag.fr/Membres/Antoine.Bouthors/teaching/opengl/
http://nehe.gamedev.net/

Parseur XML :

http://java.sun.com/javase/6/docs/technotes/guides/xml/index.html