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 seconde partie.

Second partie du projet

La seconde partie du projet correspond à l'implantation du visualisateur de scène 3D.
Même si vous pensez avoir fourni un architecture valide (voir vachement bien™),
pour cette second partie, il vous est demandé de vous conformez à une architecture,
donc vous devez pour cela implanter les interfaces décrites ci-dessous.

Ces interfaces sont regroupés dans le jar solidvision-lib.jar. Votre programme devra utiliser cette librairie et implanter les interfaces de celle-ci.

Architecture

L'objet Scene permet de stocker l'ensemble des objets (SolidObject) et lumières (Light) de la scène. De plus, cette objet possède un objet camera (Camera) qui permet de savoir à partir de quelle point la scène sera rendu et quelle est la résolution d'affichage.

Dans le but de manipuler indifféremment l'ensemble des objets de la scène,
ceux-ci possède un supertype commun SceneObject.
De plus comme l'ensemble des objets de la scène a une position,
l'interface SceneObject possède les méthodes get/setLocation().

Un programme OpenGL se compose en règle générale de deux parties, la première, l'initialisation et la seconde l'affichage.
Tous les objets SceneObject doivent donc au moins s'initialiser,
c'est le rôle de la méthode setup.

En OpenGL, le rendu d'une scène est effectuer par une projection,
dans notre programme, c'est l'objet Camera qui s'occupe de cette responsabilité,
la camera est donc un SceneObject possèdant deux méthodes getWidth() et getHeight() indiquant la taille de l'écran.
La méthode setup de l'objet Camera s'occupera de la projection.
En regardant le fichier de description de la scène ou la spécification OpenGL, on observe qu'il est possible de spécifier des couleurs et pour un objet de la scène (un SolidObject) et pour une lumière (Light). Donc ces deux types ont un supertype commun (MaterialObject) permettant de spécifier les couleurs.

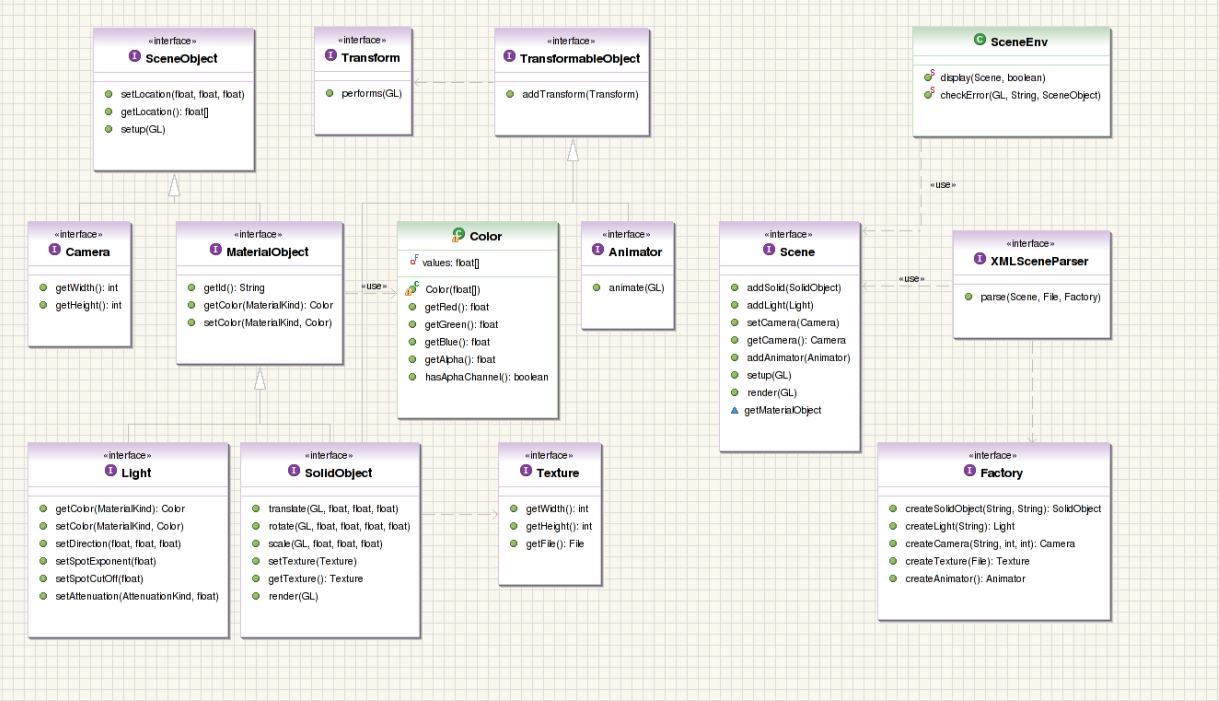
Nous verrons par la suite pourquoi il y a aussi une méthode getId().

La classe Color encapsule la notion de couleurs qui peut être codé sur 3 ou 4 flottants suivant qu'il y ait ou nom une composante alpha.
L'enumération MaterialKind représente les types de couleurs OpenGL.
Notez que tout les types de MaterialKind ne sont pas possible dans le cas des lumières.

L'interface SolidObject représente le type de base des cubes, plans et autre[1].
L'affichage d'un SolidObject s'effectue par sa méthode render, cette méthode est appelé à chaque affichage contrairement à la méthode setup qui est appelé une fois.

Un SolidObject peut avoir une texture (get/setTexture()).

Les SolidObject possèdent les méthodes rotate, translate et scale permettant respectivement de faire tourner, bouger ou grandir/rétrécir lors de l'exécution un objet SolidObject. Ces méthodes seront appelés soit par un animateur (voir plus loin) soit lors de l'initialisation.
Ces méthodes ont besoin d'un context GL (l'objet GL) car celles-ci effectue des multiplications de matrice, elle ne peuvent donc pas être appelée lors de la création de la scène mais elle peuvent l'être lors de l'initialisation.



Comme un animateur (Animator) doit pouvoir appliquer des transformations sur un objet et qu'un objet (SolidObject) peut-être transformé lors de son initialisation. Ces deux types possède un supertype commun TransformableObject qui permet d'ajouter des transformations (par l'intermédiaire d'un objet Transform).

Il est possible de charger une scène à partir d'un fichier au format XML Solidvision, pour cela on utilise un XMLSceneParser et sa méthode parse.
La méthode parse crée les objets décrits dans le fichier XML par l'intermédiaire de la classe Factory puis insère ceux-ci dans la scène. De plus le parseur appele les différentes méthodes add* et set* des objets en fonction des valeurs du fichier XML.

1 Pour les autres, regarder les méthodes de l'objet GLU.

Enfin, l'affichage de la scène s'effectue grâce à la méthode statique de la classe SceneEnv[2] nommée display.
L'affichage se fait donc en deux temps, dans un premier temps, les méthodes setup() de tous les objets de la scène sont appelées pour initialiser le contexte OpenGL (L'objet GL). Puis l'affichage proprement dit se fait grâce à la méthode render des SolidObjects.

En cas d'animation, la méthode render de chaque objet est appelée une fois pour chaque « frame » de l'animation.

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)
  - **peuvent subir des transformations (`transform`) pendant l'initalisation (optionnel)[3]**

  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="0" y="0" z="-20"/>
  <dimension dx="1" dy="1" dz="1"/>
  <color>
    <!-- material kind="ambiant" red="0.5" green="0.2" blue="0.2"/ -->
```

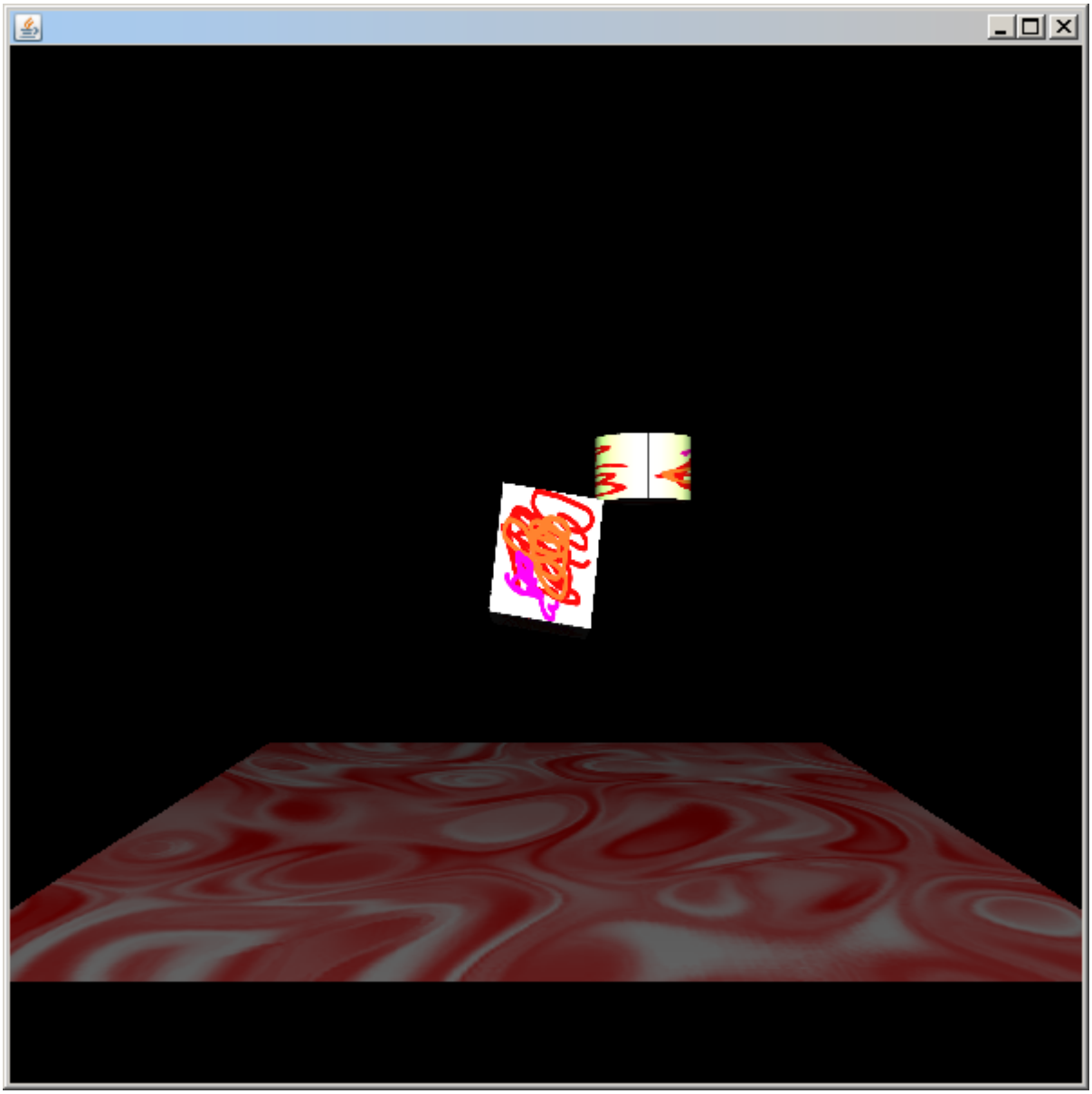
2 Cette classe possède de plus une méthode permettant de générer une exception correspondant à une erreur OpenGL.
3 Cette feature n'existait pas dans la partie 1 mais à été proposé par plusieurs groupes (plus ou moins bien d'ailleurs).

```
    <material kind="diffuse" red="0.5" green="1.0" blue="0"/>
    <material kind="specular" red="1" green="1" blue="1"/>
    <material kind="shininess" red="1" green="1" blue="1"/>
  </color>
  <texture file="drawing.png"/>
 </cube>

 <cylinder id="sphere1">
  <location x="2" y="2" z="-20"/>
  <color>
    <!-- material kind="ambiant" red="0.5" green="0.2" blue="0.2"/ -->
    <material kind="diffuse" red="0.5" green="1.0" blue="0"/>
    <material kind="specular" red="1" green="1" blue="1"/>
    <material kind="shininess" red="1" green="1" blue="1"/>
  </color>
  <transform>
    <rotate angle="90" x="1" y="0" z="0"/>
  </transform>
  <texture file="drawing.png"/>
 </cylinder>

 <plane id="plane1">
  <location x="0" y="-2" z="-10"/>
  <color>
    <material kind="diffuse" red="0.2" green="0.2" blue="0.2"/>
    <material kind="specular" red="1.0" green="1" blue="1"/>
    <material kind="shininess" red="1.0" green="1" blue="1"/>
  </color>
  <transform>
    <scale fx="4" fy="1" fz="4"/>
  </transform>
  <texture file="liquid-metal.jpg"/>
 </plane>

 <light>
  <location x="0" y="2" z="0"/>
  <!-- direction x="0" y="0" z="0"/ -->
  <!-- spot-exponent intensity="64"/>
  <spot-cutoff spread-angle="90"/ -->
  <color>
    <!-- material kind="ambiant" red="0.2" green="0.2" blue="0.2"/ -->
    <material kind="diffuse" red="0.8" green="0.8" blue="0.8"/>
    <!-- material kind="specular" red="0.8" green="0.8" blue="0.8"/ -->
  </color>
 </light>

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

 <animator idref="cube1">
  <rotate angle="1" x="1" y="0" z="1"/>
 </animator>
 <!-- animator idref="plane1">
  <scale fx="1.001" fy="1" fz="1.001"/>
 </animator -->
</scene>
```

Exemple de main (utilisant le parseur XML pour construire la scène):

```
    SceneImpl scene=new SceneImpl();
    XMLSceneParser parser = new XMLSceneParserImpl();
    parser.parse(scene, new File("example.sldv"), new FactoryImpl());
```

```
SceneEnv.display(scene, true);
```



Exemple de main (la scène est construite à la main, le résultat est la copie d'écran ci-dessus):

```
SceneImpl scene=new SceneImpl();
Camera camera=new CameraImpl(800,600);
scene.setCamera(camera);

LightImpl light=new LightImpl(0);
light.setLocation(2, 2, 2);
light.setColor(MaterialKind.AMBIANT, new Color(0.2f, 0.2f, 0.2f));
light.setColor(MaterialKind.DIFFUSE, new Color(0.8f, 0.8f, 0.8f));
light.setColor(MaterialKind.SPECULAR, new Color(1f, 1f, 1f));
scene.addLight(light);

final Cube cube1=new Cube("cube1");
cube1.setLocation(-2,0, -7);
//cube.setColor(MaterialKind.AMBIANT, new Color(0.2f, 0.2f, 0.2f));
//cube.setColor(MaterialKind.DIFFUSE, new Color(0.2f, 0.5f, 0.5f));
```

```
  cube1.setTexture(TextureImpl.createTexture(new File("liquid-metal.jpg")));
  scene.addSolid(cube1);

  final Cube cube2=new Cube("cube2");
  cube2.setLocation(2,0, -7);
  cube2.setColor(MaterialKind.SHININESS, new Color(1f, 1f, 1f));
  cube2.setTexture(TextureImpl.createTexture(new File("drawing.png")));
  scene.addSolid(cube2);

  scene.addAnimator(new Animator() {
    public void addTransform(Transform transform) {
      // do nothing
    }
    public void animate(GL gl) {
      cube1.translate(gl, 0.1f, 0, -0.1f);
      cube1.rotate(gl, 1, 0, 1, 0);
      cube2.rotate(gl, 1, 1, 1, 0);
    }
  });

  SceneEnv.display(scene, true);
```

Parsing XML

Comme l'ensemble d'entre vous n'a pas réussi à utiliser un parseur SAX, voilà comment faire :

```
  SAXParserFactory factory = SAXParserFactory.newInstance();
  factory.setNamespaceAware(true); // important
  try {
    SAXParser parser=factory.newSAXParser();
    parser.parse(file, new DefaultHandler() {
      @Override
      public void startElement(String uri,String localName,String name,
          Attributes attrs) throws SAXException {
         // appelé lors de la balise ouvrante
         // on utilise localName et pas name
      }
      @Override
      public void endElement(String uri,String localName,String name)
          throws SAXException {
        // appelé lors de la balise fermante
      }
      @Override
      public void setDocumentLocator(Locator locator) {
        super.setDocumentLocator(locator);
        lineNumber = locator.getLineNumber();
        columnNumber = locator.getColumnNumber();
      }
      private int lineNumber=-1;
      private int columnNumber=-1;
    });
  } catch(ParserConfigurationException e) {
    throw new IOException(e);
  } catch(SAXException e) {
    Throwable cause = e.getCause();
    if (cause instanceof IOException)
      throw (IOException)cause;
    throw new IOException(e);
  }
}
```

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 lundi 21 mai 8h30 sous forme d'une archive au format ZIP (pas de rar, ni de tar, ni de tgz, ni de lhz ni etc.)

Le second rendu devra impérativement être effectué par les mêmes binôme que le précédent
rendu.
Voici le nom des répertoires et fichiers qui doivent être contenus dans l'archive zip.

1. un fichier **readme.txt** indiquant comment compiler et exécuter le programme, où se trouve la doc, *etc.*
2. un fichier Ant **build.xml** permettant de compiler les sources du programme, et de créer le jar *exécutable* **solidvision.jar**.
3. un répertoire **src** contenant l'ensemble des sources (.java) sous forme de plusieurs paquetages. Les interfaces et classes contenues dans fr.umlv.solidvision ne doivent pas être modifiées.
4. un répertoire **classes** contenant l'ensemble des classes (.class) correspondant aux sources sous forme de plusieurs paquetages.
5. un répertoire **lib** contenant l'ensemble des JARs correspondant aux bibliothèques externes utilisées.
6. un répertoire **bin** contenant deux fichiers, **solidvision.sh**, script shell démarrant le logiciel sous Linux (en fait Unix) et **solidvision.bat** démarrant le logiciel sous Windows.
7. un répertoire **docs** contenant deux documents au format PDF :
    - La documentation utilisateur **user.pdf** contenant en plus des informations classiques (comment compiler, exécuter, etc.) une description de l'application, et comment l'utiliser ainsi qu'une liste des bugs connus.
    - La documentation développeur **dev.pdf** contenant
        - une description détaillé et au niveau de chaque classe que vous avez implantée.
        - une documentation indiquant comment implanter les parties optionnelles (même si vous ne les implanter pas) dans l'architecture
        - une documentation indiquant comment ajouter de nouveaux objets types: tore, théière, rhinocéros etc.

    En plus des deux documents, le répertoire **docs** devra contenir un sous-répertoire **api** contenant la documentation complète du logiciel au format javadoc.

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