VoxeLidar API :

Sommaire

Les modules :

VoxeLidar est composé de plusieurs modules qui sont :

Le module de visualisation, qui se trouve dans le package engine3d.
Le module de voxelisation, qui se trouve dans le package voxelisation.
Le module qui génère des graphes, qui se trouve dans le package chart.

L'utilisation des modules :

Chaque module possède une classe « interface » pour permettre l'utilisation simplifiée des classes du package.
On a donc dans l'ordre:

la classe JOGLWindow qui sert à afficher un espace voxel dans une fenêtre,
la classe VoxelisationTool qui permet de générer des fichiers voxels,
la classe Chart qui génère des graphiques

Les listeners :

Chacune de ces classes possède une classe listener et une classe adapter.
Le listener permet de connaître l'avancement du traitement effectué.
L'adapter permet de ne pas avoir à implémenter toutes les méthodes du listener.

Utilisation d'un listener :

```
Chart chart = new Chart() ;
chart.addChartListener(new ChartListener() {

  @Override
  public void chartCreationProgress(String progress, int ratio) {

  }

  @Override
  public void chartCreationFinished() {
  }

});

chart.doSomething();
```

Engine3d : moteur de rendu 3d d'espace voxel

Simple visualisation

Utilisation :

```
VoxelSpace voxelSpace = new VoxelSpace(File voxelspace) ;

JOGLWindow window = new JOGLWindow(int width, int height, String title,
VoxelSpace voxelSpace, Settings settings);

window.show();
```

Paramètres :

int *width* : largeur de la fenêtre
int *height* : hauteur de la fenêtre
String *title* : Titre de la fenêtre
File *voxelSpace* : fichier voxels

Changer l'attribut de voxel à visualiser

```
window.getJoglContext().getScene().getVoxelSpace().changeCurrentAttribut(
String nom de l'attribut);
```

Voxelisation : permet la voxelisation de fichiers ALS et TLS

Paramètres de voxelisation

Il faut instancier l'objet VoxelParameters.

Utilisation :

```
VoxelParameters parameters = new VoxelParameters(
new Point3d(bottomCornerX, bottomCornerY, bottomCornerZ),
new Point3d(topCornerX, topCornerY, topCornerZ),
new Point3i(splitX, splitY, splitZ)) ;
```

Paramètres:

Point3d *bottomCorner* : coin inférieur gauche
Point3d *topCorner* : coin supérieur droit
Point3i *split* : nombre de découpes, est égal à :

$$\frac{maxPoint - minPoint}{resolution}$$

Par exemple avec:

bottomCornerX = -10 , bottomCornerY= -10 ,bottomCornerZ = -2
topCornerX = 10 , topCornerY = 10 , topCornerZ = 8
et une resolution de 2 on obtient:

$$splitX = \frac{topCornerX - bottomCornerX}{resolution} = \frac{10 - (-10)}{2} = 10$$

$$splitY = \frac{topCornerY - bottomCornerY}{resolution} = \frac{10 - (-10)}{2} = 10$$

$$splitZ = \frac{topCornerZ - bottomCornerZ}{resolution} = \frac{8 - (-2)}{2} = 5$$

Génération d'un fichier voxel à partir d'un fichier las :

Utilisation :

```
VoxelisationTool voxelisationTool = new VoxelisationTool() ;
voxelisationTool.generateVoxelsFromLas(File ouput, File input, File trajectory, File dtm, VoxelParameters parameters, Mat4D vop) ;
```

Paramètres:

File *output* : fichier de sortie
File *input* : fichier las
File *trajectory* : fichier trajectoire
File *dtm* (optionnel) : fichier MNT
VoxelParameters *parameters* : paramètres de voxelisation
Mat4D *vop* (optionnel): Voxels Orientation And Position, matrice de transformation global -> local

Génération d'un fichier voxel à partir d'un projet rsp:

Utilisation :

```
voxelisationTool.generateVoxelsFromRsp(File output, File input,
VoxelParameters parameters,  Mat4D vop, boolean mon) ;
```

Paramètres:

File *output* : répertoire de sortie
File *input* : fichier rsp
VoxelParameters *parameters* : paramètres de voxelisation
Mat4D *vop* (optionnel): Voxels Orientation And Position, matrice de transformation global -> local
boolean mon : Si vrai, voxelise les fichiers « .mon » (basse résolution)
Si faux, voxelise les fichiers non « mon » (haute résolution)

Information :

Les fichiers de sortie porteront le même nom que les fichiers rxp d'origine, plus l'extension « .vox »

Génération d'un fichier voxel à partir d'un fichier rxp :

```
voxelisationTool.generateVoxelsFromRxp(File output, File input,
VoxelParameters parameters,  Mat4D vop) ;
```

Paramètres:

File *output* : fichier de sortie
File *input* : fichier rxp
VoxelParameters *parameters* : paramètres de voxelisation
Mat4D *vop* (optionnel): Voxels Orientation And Position, matrice de transformation global -> local

Fusion des fichiers voxels :

```
voxelisationTool.mergeVoxelsFile(ArrayList<File> files, File output) ;
```

Paramètres:

ArrayList<File> *files* : liste des fichiers à fusionner
File *output* : fichier de sortie

Attributs

Attributs: Les attributs sont les valeurs des voxels.

Les attributs par défaut sont :

i	Indice du voxel en x
j	Indice du voxel en y
k	Indice du voxel en z
nbSampling	Nombre de rayon échantillonnés qui ont traversés le voxel
interceptions	Nombre de rayon échantillonnés qui été interceptés dans le voxel
lgTraversant	Somme des longueurs des rayons qui ont traversés le voxel Lorsque le rayon est intercepté dans le voxel la longueur ajoutée est égale à la distance entre le point d'entrée dans le voxel et le point d'interception Lorsque le rayon traverse le voxel sans interception la longueur ajoutée est égale à la distance entre le point d'entrée et le point de sortie
lgInterception	Somme des longueurs des rayons qui ont interceptés le voxel. La longueur ajoutée est égale à la distance entre le point d'entrée et le point de sortie
PAD	LAD (Leaf Area Density) Est égal à -1 lorsque le voxel ne contient pas de vegetation

	Est fixé à 10 lorsque le LAD est supérieur à 10 Autrement la formule est : $\frac{\log(\text{transmitance})}{-0.5*l}$ où transmittance = $\frac{\text{BFEntering}-\text{BFIntercepted}}{\text{BFEntering}}$
BFIntercepted	Beam fraction Intercepted , somme des fractions de faisceau interceptées
BFEntering	Beam Fraction Entering, somme des fractions de faisceau entrants
BSIntercepted	Beam Surface Intercepted, somme des fractions de surface interceptée Est égal à : $BFIntercepted * surface$ où *surface* = $\left(\tan\left(\frac{laserbeamdivergence}{2}\right) * distance\right)^2 * pi$ Où *distance* est la distance de l'interception à la source du rayon (le lidar) et *laserbeamdivergence* est une constante fixée à 0.0005 et correspond à la divergence du laser
BSEntering	Beam Surface Entering, somme des fractions de surface entrantes Est égal à : $BFEntering * surface$
dist	Distance du centre du voxel au sol représenté par le modèle numérique de terrain (MNT). Si aucun MNT n'a été chargé, la distance est simplement la distance par rapport à 0

Créer un attribut à partir des attributs existants :

```
Attribut attribute = new Attribut(String name, String expression,
String[] variablesNames);
```

Il est possible de créer un attribut dont l'expression utilise plusieurs variables connues.
Ce système fonctionne grâce à la librairie exp4j qui résout les expressions mathématiques.
Voir le site officiel : http://www.objecthunter.net/exp4j/

Paramètres:

String *name* : nom du nouvel attribut
String *expression* : formule

```
Exemple : "3 * (sin(pi) - 2 )/ PAD"
```

String[] *variablesNames* : nom des inconnues
Si on prend l'exemple du dessus, l'inconnue est « PAD »

Création d'un attribut sur un espace voxel:

```
VoxelSpace voxelSpace = new VoxelSpace(File voxelspace);
Attribut attribute = new Attribut("monAttribut", "3 * (sin(pi) - 2 )/
PAD", new String[]{"PAD"});
voxelSpace.addAttribut(attribute);
```

Les filtres

Création d'un filtre:

```
Filter filter = new Filter (String variable, double value, int
condition);
```

Paramètres:

String *variable* : nom de l'attribut à filtrer
double *value* : valeur à tester
int *condition* : type de la condition

0 : différent de
1 : égal
2 : moins que
3 : moins que ou égal
4 : plus grand que
5 : plus grand que ou égal

Exemple de filtrage :

```
VoxelFilter filter = new VoxelFilter();
voxelFilter.addFilter(new Filter(("monAttribut", 2.0, 0));
```

Objets pouvant être filtrés

```
Chart chart = new Chart();
chart.setFilter(voxelFilter);
```

Chart : génère des graphiques à partir de fichiers voxels

Utilisation :

Génération d'une fenêtre contenant le profil de végétation :

```
Chart chart = new Chart() ;
VoxelSpace voxelSpace = new VoxelSpace(File voxelspace);
ChartJFrame jFrame = chart.generateVegetationProfile(voxelspace,
filter);
jFrame.setVisible(true);
```

Génération d'un graphe représentant une fonction de deux variables :

```
Chart chart = new Chart() ;
VoxelSpace voxelSpace = new VoxelSpace(File voxelspace);
ChartJFrame jFrame = chart.generateXYChart(VoxelSpace voxelspace,
VoxelFilter filter, String title, String horizontalAxis, String
verticalAxis);
jFrame.setVisible(true);
```

L'extraction des données rxp

Le format rxp étant un format propriétaire de Riegl, nous avons besoin d'utiliser une librairie permettant d'extraire des données de ces fichiers.

Cette librairie se nomme RiVLIB et est téléchargeable avec un compte membre sur le site de Riegl.

http://www.riegl.com/members-area/software-downloads/libraries/

Pour utiliser cette librairie avec Java nous avons besoin de passer par JNI.

Mise en place :

Génération et utilisation d'une dll avec JNI

Vous avez besoin de ces fichiers d'entête pour votre projet :
Jni.h et jni_md.h
Ils se trouvent dans votre JDK dans le répertoire include.
Vous devez générer un fichier header avec l'outil javah.

Tutoriel : http://www.jmdoudoux.fr/java/dej/chap-jni.htm

Attention :

La commande javah permet de générer le fichier header, celui-ci contient des définitions vers le nom pleinement qualifié de la classe qui va utiliser la librairie dynamique.
Ce chemin ne doit pas être changé dans le projet java.

Dans le projet VoxeLidar, la classe d'extraction des données rxp a le chemin suivant :
fr.ird.voxelidar.voxelisation.extraction.RxpExtraction

Si vous souhaitez modifier changer le nom pleinement qualifié de la classe, c'est-à-dire déplacer le package ou renommer la classe, vous devez recompiler les librairies dynamiques (linux et windows) avec le nouveau fichier header.

Maven

Maven est un outil fourni par Apache qui permet la gestion et l'automatisation de production des projets logiciels Java (source : wikipedia).

Le projet VoxeLidar utilise cet outil pour plusieurs fonctions :

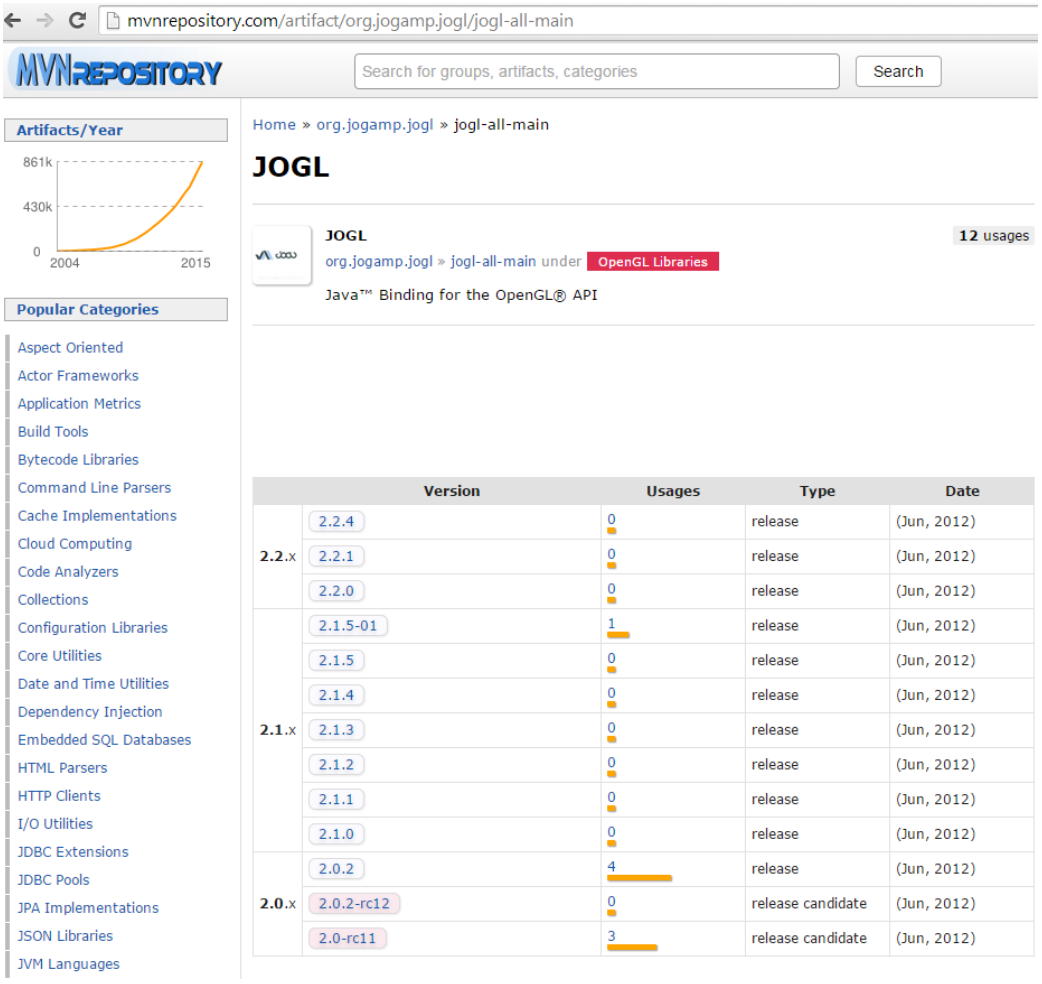
->Import de dépendances externes
->Génération d'un exécutable .jar
->Inclusion de fichiers ressources

Import de dépendances externes

Il faut tout d'abord rechercher la librairie dans la base de données de Maven.

http://mvnrepository.com/

En faisant une recherche sur JOGL on tombe sur une page listant les versions de la librairie



On choisit la dernière version, ici la 2.2.4 et on copie le contenu de la balise dependency :

```
<dependency>
      <groupId>org.jogamp.jogl</groupId>
      <artifactId>jogl-all-main</artifactId>
      <version>2.2.4</version>
</dependency>
```

On colle le contenu dans le fichier pom.xml à la racine du projet VoxeLidar, dans la balise <dependencies>
On enregistre et vous pouvez utiliser la librairie.

Génération d'un exécutable (.jar)

Le projet est déjà configuré pour générer un jar, si vous voulez en savoir plus voici un tutoriel : http://www.mkyong.com/maven/how-to-create-a-jar-file-with-maven/

Inclusion de fichiers ressources

Ce plugin décrit comment les ressources doivent être exportées :

```
<plugin>
      <artifactId>maven-resources-plugin</artifactId>
      <version>2.7</version>
      <executions>
            <execution>
                  <id>copy-resources</id>
                  <phase>validate</phase>
                  <goals>
                        <goal>copy-resources</goal>
                  </goals>
                  <configuration>
                        <outputDirectory>${basedir}/target/</outputDirectory>
                        <resources>
                              <resource>
                              <directory>${basedir}/src/main/resources</directory>
                              <filtering>true</filtering>
                              </resource>
                        </resources>
                   </configuration>
             </execution>
      </executions>
</plugin>
```

Ici le plugin est configure pour exporter les resources du dossier ayant pour chemin relatif *src/main/resources*
Donc si vous voulez ajouter une ressource c'est dans ce dossier qu'il faut la placer.

Pour utiliser une ressource texte:

```
InputStreamReader ressource = new
InputStreamReader(this.getClass().getClassLoader().getResourceAsStream(
"maRessource.ext"));

BufferedReader reader;
reader = new BufferedReader(ressource);
```

Pour utiliser une image :

```
ImageIcon image = new
ImageIcon(getClass().getResource("/icons/image.gif")));
```