

3D4J - Manuel d'Utilisation

Version 0.1

Matthieu EXBRAYAT, Lionel MARTIN
LIFO, Université d'Orléans

17 septembre 2009

Table des matières

1	Avertissement	2
2	Introduction	2
3	Installation	2
4	Mise en oeuvre	2
4.1	Objets principaux	2
4.1.1	Data3D	2
4.1.2	DescriptionData	3
4.1.3	Création de la fenêtre 3D	3
4.2	Exemple	3
5	Détail sur les classes de description des objets	3
5.1	Data3D	3
5.2	DescriptionData	4
5.2.1	float[][]data	4
5.2.2	Vector[]symb	4
5.2.3	DatasetParams params	5
6	Fenêtre de visualisation	8
7	Manipulation de la fenêtre 3D	8

1 Avertissement

3D4J est un développement universitaire. Considérez le comme une librairie en cours d'évolution. N'hésitez pas à nous adresser vos remarques et suggestions pour améliorer ou ajouter des fonctionnalités.

2 Introduction

3D4J est une librairie java permettant de manipuler facilement des objets dans un espace 3D. Les objets sont des formes simples (sphères, cubes, etc.), que l'on place en indiquant leurs coordonnées spatiales. A chaque objet peuvent être adjointes des informations complémentaires (fichier image, numéro de classe, etc.) qui seront utilisables durant la visualisation. Cette librairie est notamment utilisée pour visualiser des proximités entre objets et aider à la création de groupes cohérents (clustering).

Techniquement parlant, un tableau d'objets est fournie au lancement de l'application. Ces objets peuvent être accompagnés d'informations complémentaires. La structuration des informations complémentaires est indiquée plus loin. A partir de cette liste d'objets, on crée une scène 3D. Chaque objet est placé aux coordonnées indiquées et représenté par une sphère.

On peut par la suite ajouter des objets complémentaires, mais ceux-ci ne disposent pas de toutes les fonctionnalités (ajout d'informations visuelles) associés aux objets initiaux. On peut par exemple ajouter un segment reliant deux objets, une sphère translucide englobant un groupe d'objets, etc.

3 Installation

3D4J se présente sous la forme d'une librairie java *3d4j.jar*. Son utilisation nécessite l'installation préalable de java 6 (<http://java.sun.com/javase/downloads>) et de java3D(<https://java3d.dev.java.net>).

4 Mise en oeuvre

4.1 Objets principaux

Pour créer la scène 3D, on crée d'abord la liste des objets initiaux, à l'aide des deux classes *Data3D* et *DescriptionData*.

4.1.1 Data3D

D'une part on crée un objet de type *Data3D*. Il s'agit, sommairement, d'un objet contenant un tableau de réels. Chaque ligne contient trois réels correspondant aux coordonnées 3D de l'objet. Ces coordonnées peuvent prendre n'importe quelle valeur. Pour créer cet objet, le plus simple consiste à créer un tableau de réels (float) de taille $n \times 3$ (n =nombre d'objets), puis de faire un *new Data3D* en passant ce tableau en paramètre.

4.1.2 DescriptionData

On crée ensuite un objet de type *DescriptionData*, qui contient les informations complémentaires que l'on souhaite visualiser. Dans la version actuelle, ces informations peuvent se matérialiser par la coloration des objets, par la forme qui leur est associée (par défaut : sphère), par un texte apparaissant dans la fenêtre 3D à côté de l'objet, ou par une image apparaissant dans la scène 3D à côté de l'objet ou apparaissant dans une fenêtre indépendante. La structure de cet objet est définie en section 5.2. Pour l'instant, on peut se contenter de créer un objet à l'aide du constructeur vide (ie : *new DescriptionData()*).

4.1.3 Création de la fenêtre 3D

On invoque enfin la méthode *DisplayFactory.createDisplay(data, desc)*, qui retourne un objet de type *Display*, et crée la scène 3D. Les deux premiers paramètres sont les objets de type *Data3D* et *DescriptionData* créés précédemment.

4.2 Exemple

```
package explorer;

import explorer.data.Data3D;
import explorer.data.DescriptionData;
import explorer.display3D.Display;
import explorer.display3D.DisplayFactory;

public class SampleDisplay {

    public static void main(String [] args) {
        // Creation d'un tableau de 5 objets
        float [][] points=new float [5][3];
        for (int i=0;i<5;i++) {
            points[i][0]=points[i][1]=points[i][2]=(float)i/4-0.5f;
        }
        Data3D d3d=new Data3D(points);
        // Pas de donnees complementaires
        DescriptionData dd=new DescriptionData();
        // Creation de la fenetre
        Display disp=DisplayFactory.createDisplay(d3d,dd);
    }
}
```

5 Détail sur les classes de description des objets

5.1 Data3D

Data3D est une classe contenant essentiellement les coordonnées des objets à visualiser. La signature du constructeur est :

```
public Data3D(float tt[][])
```

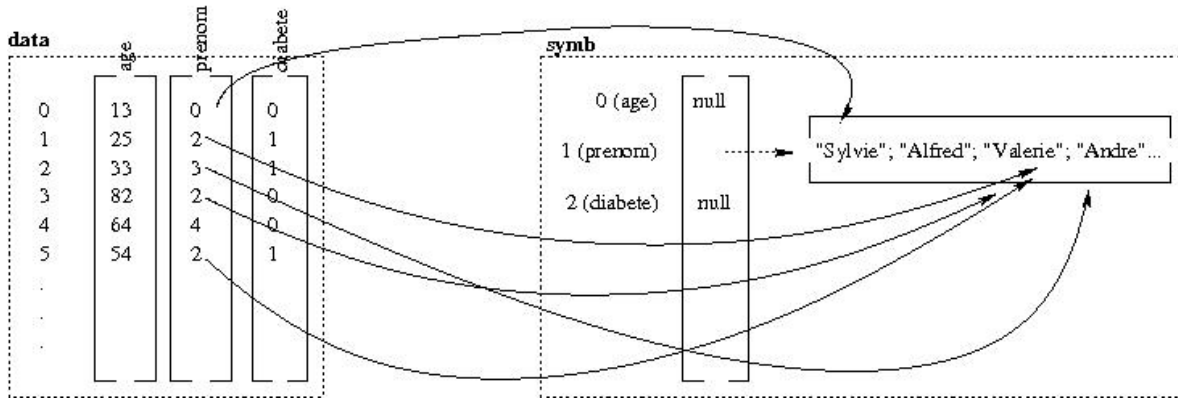


FIG. 1 – Exemple de structure de description complémentaire

L'attribut contenant les coordonnées s'appelle *tab3D*. Pour des raisons de simplicité sa visibilité est *public*. il s'agit d'un tableau de réels (float) à deux dimensions, de taille $n \times 3$, n étant le nombre d'objets. Il existe également une méthode *public float getValue(int i, int j)* retournant la $j^{\text{ème}}$ coordonnée de l'objet i . Le nombre d'objets contenus dans un objet *Data3D* peut être obtenu par invocation de la méthode *public int getNbObjets()*. Les coordonnées des objets auront de préférence une valeur entre -1 et +1 sur chaque axe.

5.2 DescriptionData

DescriptionData contient les attributs complémentaires correspondant à chaque objet. Par exemple, si les objets sont déjà classifiés (regroupés en catégories), une colonne de *DescriptionData* peut contenir le numéro de classe associé à chaque objet. On peut ensuite associer une couleur à chaque numéro de classe, et donc visualiser par le biais de la couleur la dispersion des objets de chaque classe dans l'espace 3D.

Le principal constructeur de cette classe est :

```
public DescriptionData(float[][] data, Vector[] symb, DatasetParams params)
```

5.2.1 float[][] data

Le tableau *data* est un tableau de taille $n \times m$, n étant le nombre d'objets, et m le nombre d'attributs complémentaires. Ces informations complémentaires peuvent être de différentes natures : numérique ou textuelle (exemples : numéro de classe, description, nom de fichier, etc.). Si l'attribut est de type numérique, alors il est directement accessible via le tableau *data*. Si en revanche il est de type textuel, alors on a recours à une structure secondaire : le vecteur *symb*.

5.2.2 Vector[] symb

Symb est un tableau de vecteurs de taille m . L'entrée i de ce tableau contient un vecteur, qui contient lui même la liste des valeurs associées à la colonne i du tableau *data*.

Par exemple, considérons un jeu de données (voir figure 1) correspondant à des personnes, pour lesquelles on connaît trois informations complémentaires : l'âge, le prénom et le fait que la personne soit diabétique ou non.

Data contient alors trois colonnes, et *symb* contient trois lignes. La première ligne correspond à la première colonne de *data* etc. La première colonne de *data* correspond à l'âge. La lecture est directe. La première colonne de *symb* (*symb*[0]) est donc *null*. Il en va de même pour la troisième colonne et la troisième ligne (*symb*[2]), car on peut encoder le diabète par 1 et son absence par 0. La deuxième colonne de *data* correspond au prénom. La seconde ligne de *symb* (*symb*[1]) contient donc un vecteur de chaînes de caractères. Dans le cas le plus simple, ce vecteur contient *n* chaînes, la première correspondant à la première personne, la seconde à la seconde personne, etc. Mais on peut imaginer que plusieurs personnes aient le même prénom. Dans ce cas, le vecteur contient une liste de chaînes, sans doublons. On introduit ainsi une indexation. Par exemple, si les personnes n° 1, 3 et 5 portent le prénom "Valérie", alors, une seule des chaînes contenues dans *symb*[1] vaut "Valérie", par exemple l'élément n° 2. Du côté de *data*, on aura : *data*[1][1] = *data*[3][1] = *data*[5][1] = 2.

5.2.3 DatasetParams params

Reste l'objet de type *DatasetParams*. Cet objet permet d'indiquer le type des colonnes de *data* (et donc des lignes de *symb*).

Il contient essentiellement 5 attributs accessibles par getters / setters :

- *private int formCol*
- *private int colorCol*
- *private int labelCol*
- *private int assocDataCol*
- *private int classCol*

Par défaut, ces paramètres ont la valeur -1, ce qui indique qu'aucune colonne ne leur est associée.

formCol indique le numéro de la colonne associée à une forme. Si *formCol* vaut -1, tous les objets sont représentés par des sphères. S'il vaut *x* (*x* ≥ 0), alors l'objet de rang *y* a une forme correspondant à la valeur indiquée par *data*[*y*][*x*]. Cela implique que la colonne *x* ne contient que des entiers compris entre 0 et 3 (on dispose de quatre formes de base : sphère, cube, cône et cylindre).

colorCol indique le numéro de la colonne associée à une couleur. Si *colorCol* vaut -1, tous les objets sont de la couleur par défaut (bleu). S'il vaut *x* (*x* ≥ 0), alors l'objet de rang *y* a une couleur correspondant à la valeur indiquée par *data*[*y*][*x*]. Cela implique que la colonne *x* ne contient que des entiers.

Les couleurs ne sont gérables que pour une colonne à valeurs indexées, c'est à dire correspondant à une ligne définie dans symb.

La palette de couleurs est automatiquement adaptée au nombre de couleurs nécessaires (i.e. le nombre d'éléments dans la ligne *x* de *symb*).

Pour que les couleurs soient correctement gérées, il est impératif, dans cette version de la librairie, que les attributs colorCol et classCol aient la même valeur.

labelCol indique le numéro de la colonne associée à un label. Le label est une information textuelle, que l'on peut faire apparaître dans la fenêtre 3D à côté de l'objet concerné. Si *labelCol* vaut -1, l'information affichée est le numéro de l'objet (numéro de ligne dans

l'objet `Data3D` et dans `data`). S'il vaut x ($x \geq 0$), alors le label associé à l'objet de rang y est pris dans `symb[x]`, au rang `data[y][x]`. Cela implique que `symb[y]` n'est pas null.

assocDataCol indique le numéro de la colonne associée à un nom de fichier. L'idée sous-jacente est de pouvoir appeler à la volée une application qui nous retourne une information contenue dans le fichier dont le nom est indiqué. Dans la version actuelle cette colonne sert exclusivement à afficher des images. On peut faire apparaître l'image associée à un objet, soit dans la fenêtre 3D soit dans une fenêtre complémentaire. Si `assocDataCol` vaut -1, rien ne se passe. S'il vaut x ($x \geq 0$), alors le nom du fichier image associé à l'objet de rang y est pris dans `symb[x]`, au rang `data[y][x]`. Cela implique que `symb[y]` n'est pas null.

classCol indique le numéro de la colonne associée à la classe à laquelle appartient un objet. Par défaut, cette colonne est intimement liée à la colonne `colorCol`, car on associe une couleur à chaque classe. Si `classCol` est défini (`classCol` ≥ 0), alors les objets sont colorés et une légende apparaît, indiquant la couleur associée à chaque classe.

La fenêtre de légende offre diverses fonctionnalités complémentaires. Elle permet de dissimuler les objets d'une classe donnée, de modifier la couleur associée, etc. Il est également possible de faire apparaître une ellipse englobant les objets d'une classe donnée.

Les classes ne sont gérables que pour une colonne à valeurs indexées, c'est à dire correspondant à une ligne définie dans `symb`. La palette de couleurs est automatiquement adaptée au nombre de valeurs différentes se trouvant dans la ligne x de `symb`. Pour que les classes et la coloration conséquente des objets soient correctement gérées, il est impératif, dans cette version de la librairie, que les attributs `colorCol` et `classCol` aient la même valeur.

La fenêtre de la figure 2 reprend l'exemple du diabète, pour lequel les paramètres ont été affectés comme suit : `labelCol=1`(prénom) ; `classCol=colorCol=formCol=2`(diabete). Notons que pour pouvoir gérer proprement classes et couleurs, `symb[2]` est défini et contient deux valeurs (diabétique/non diabétique). Le code correspondant est présenté ci-dessous.

```
package explorer ;

import java.util.Vector ;

import explorer.data.Data3D ;
import explorer.data.DatasetParams ;
import explorer.data.DescriptionData ;
import explorer.display3D.Display ;
import explorer.display3D.DisplayFactory ;

public class SampleDisplay2 {

    public static void main(String[] args) {
        float [][] points=new float [5][3] ;
        for (int i=0;i<5;i++) {
            points[i][0]=points[i][1]=points[i][2]=(float)i/4-0.6f ;
        }
        Data3D d3d=new Data3D(points) ;
```

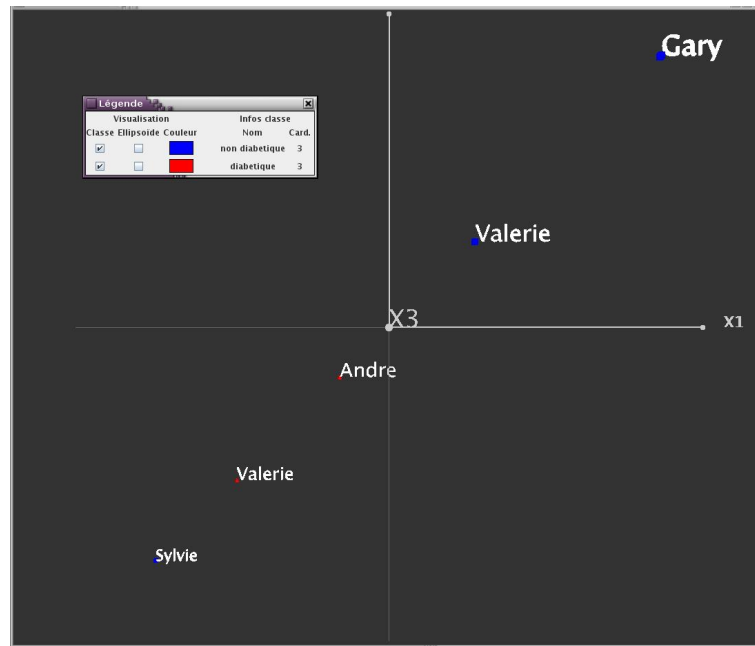


FIG. 2 – Visualisation de 6 objets

```
float [][] data=new float [6][3];
data [0][0]=13; data [0][1]=0; data [0][2]=0;
data [1][0]=25; data [1][1]=2; data [1][2]=1;
data [2][0]=33; data [2][1]=3; data [2][2]=1;
data [3][0]=82; data [3][1]=2; data [3][2]=0;
data [4][0]=64; data [4][1]=4; data [4][2]=0;
data [5][0]=54; data [5][1]=2; data [5][2]=1;
```

```
Vector symb[]=new Vector [3];
symb[0]=null;
symb[1]=new Vector<String >();
symb [1].add(" Sylvie ");
symb [1].add(" Alfred ");
symb [1].add(" Valerie ");
symb [1].add(" Andre ");
symb [1].add(" Gary ");
symb[2]=new Vector<String >();
symb [2].add(" non diabétique ");
symb [2].add(" diabétique ");
```

```
DatasetParams params=new DatasetParams ();
params.setColorCol (2);
params.setClassCol (2);
params.setFormCol (2);
params.setLabelCol (1);
```

```

        DescriptionData dd=new DescriptionData(data,symb,params);
        Display disp=DisplayFactory.createDisplay(d3d,dd);
        disp.setLabelSize(0.5f);
    }
}

```

6 Fenêtre de visualisation

La fenêtre de visualisation est de type `Display`. On instancie cette fenêtre par le biais d'une usine, en invoquant :

```
DisplayFactory.createDisplay(Data3D d3d, DescriptionData dd)
```

les deux paramètres ayant été définis en section 5.

Parmi les méthodes disponibles dans `Display`, notons :

- *public void setVisible(boolean isVisible)* qui permet de gérer la visibilité de la fenêtre
- *public void dispose()* qui permet de libérer la mémoire allouée à cette fenêtre.

Les méthodes suivantes permettent d'ajouter des objets "après coup" (ces objets ne feront pas partie de la structure gérée par les instances de *Data3D* et *DescriptionData*) :

- *public int addCone(float[] center, float[] direction, float scale)* qui permet d'ajouter un cône de hauteur *scale*, dont le centre se trouve aux coordonnées *center*, et qui est orienté suivant le vecteur défini par *direction*.
- *public int addBox(float x, float y, float z, float mx, float my, float mz)* qui permet d'ajouter un pavé, dont le centre a les coordonnées *x,y* et *z*, et dont les côtés ont la taille $2*mx$, $2*my$ et $2*mz$.
- *public int addCube(float[] centre, float scale)* qui ajoute un cube de centre *centre* et de côté $2*scale$.
- *public int addCylinder(float[] center, float scale)* qui ajoute un cube de centre *centre* et de diamètre/hauteur $2*scale$.
- *public int addSphere(float[] center, float scale)* qui ajoute une sphère de centre *centre* et de rayon *scale*.
- *public int addText(float[] start, String text)* qui ajoute un texte *text* commençant aux coordonnées *start*.
- *public int addLine(float[] from, float[] to)* qui ajoute un segment dont les extrémités sont *from* et *to*.

Toutes ces méthodes retournent un entier qui indique l'index associé à l'objet créé. Par défaut, ces objets sont noirs. On peut modifier leur couleur grâce à :

```
public void setObjectColor(int index, java.awt.Color color)
```

index indiquant l'index de l'objet retourné lors de sa création.

Notons également la méthode *public int[] getSelectedObjects()*, qui permet de connaître la liste des objets sélectionnés dans la fenêtre de visualisation (voir section suivante).

7 Manipulation de la fenêtre 3D

La manipulation se fait intégralement à la souris.

- Clic droit sur un objet : ouvre un menu contextuel, permettant de faire apparaître les images, les labels...

- Clic droit (maintenu enfoncé) hors objet : permet de déplacer latéralement la scène (l'ensemble des objets) dans la fenêtre 3D.
- Clic droit hors objet : permet de définir une zone de sélection rectangulaire. Le premier clic place le coin supérieur gauche. Le déplacement de la souris fait apparaître un rectangle translucide, dont le coin inférieur droit est fixé par un second clic. Tous les objets situés sous le rectangle sont sélectionnés.
- Clic centre (maintenu enfoncé) : permet de zoomer dans la fenêtre 3D.
- Clic gauche (maintenu enfoncé) : permet de faire tourner la scène 3D.
- Shift+clic droit : permet de sélectionner un objet (l'objet apparaît en surbrillance).

Lorsqu'il est sélectionné, un objet apparaît en surbrillance. La liste des objets sélectionnés peut être obtenue grâce à la méthode *getSelectedObjects()* (voir section précédente).