Étude de différentes technologies dans le cadre de la réalisation d'un outil de visualisation de pavés

Alain MARGUERITE

30 juin 2011

1 Choix technologiques

1.1 Langage principale et ses librairies

La création d'un tel outil a naturellement amené à choisir un langage de programmation orienté objet. La technologie en elle même fut choisie en fonction de mes connaissances initiales et de sa facilité d'implantation. La volonté de faire évoluer l'outil en une Applet a aussi orienté la décision. C'est donc le langage JAVA qui fut retenu. La bibliothèque d'interface graphique SWING, s'est par conséquences imposée d'elle même pour la création des IHM. Elle est en effet au sein de JAVA la bibliothèque standard pour cet usage.

1.1.1 Documentation

Sources

- `http://www.oracle.com/technetwork/java/javase/downloads/index-jdk5-jsp-142662.html`

Tutoriels

- `http://jmdoudoux.developpez.com/cours/developpons/java/`
- `http://www.siteduzero.com/tutoriel-3-10601-apprenez-a-programmer-en-java.html`
- `http://download.oracle.com/javase/tutorial/uiswing/components/index.html`

1.2 Format d'entrée

Le cahier des charges demandait une indépendance entre la sources des données (en sortie de realpaver par exemple). Il était donc nécessaire d'utiliser une technologie permettant de rendre lisible ces données sur n'importe quelle plateforme. Le choix de XML, est principalement du à cet aspect d'universalité et de

portabilité que présente le langage. L'utilisation de JSON a été envisagée, mais son fonctionnement de type JavaScript et le peu de classes permettant d'interagir avec JAVA (contrairement à XML) ont entraîné des réticences. Par exemple la sérialisation XML en Java est rendu aisée par des classes et objets java préexistants. De même JSON a la particularité d'accéder aux données selon la structure d'un objet lorsque l'on utilise avec un autre langage. La connaissance totale de la structure du document est nécessaire dans l'utilisation de JSON, ce qui peut poser problème dans une future évolution du langage d'entrée.

1.2.1 Documentations

Sources

- `http://www.w3.org/XML/#wgs`
- `http://www.json.org/`

Tutoriels

- `http://gilles-chagnon.developpez.com/cours/xml/dtd-et-schemas/?page=dtd`
- `http://gilles-chagnon.developpez.com/cours/xml/dtd-et-schemas/?page=schemas#LII-A`
- `http://ydisanto.developpez.com/tutoriels/j2se/serialisation/partie2/`

1.3 Visualisation du pavage JAVA3D

Java 3D est une technologie développée par une organisation indépendante : Java Community Process. Elle ne fait donc pas encore partie de JDK 6 (JRE et outils de développement de Java version 6), mais on prévoit que ce sera le cas dans une version future. L'API Java 3D emploie des technologies déjà existantes telles que DirectX (Windows) et OpenGL (Linux). Il existe d'autres librairies telle JOGL considérée comme la meilleur interface entre Java et OpenGL ou encore LWJGL. Cette dernière est plutôt destinée aux développeurs de jeux 3D basés sur la technologie OpenGL.

1.3.1 Installation et compilation

API utilisée : Java 3D 1.5.1 for Linux
`http://www.oracle.com/technetwork/java/javase/tech/index-jsp-138252.html`
Pour la compilation avec *javac* il peut être nécessaire d'utiliser l'option $-extdirs$ permettant de spécifier l'emplacement des librairies JAVA3D. De même pour l'exécution avec *java*, l'option -classpath permet de spécifier le dossier contenant les .jar de l'API JAVA3D nécessaires à l'exécution. Leur emplacement est indiqué lors de l'installation de JAVA3D.

1.3.2 Documentations

Sources

– http://java3d.java.net/

1.3.3 Tutoriels

– http://www.siteduzero.com/tutoriel-3-10456-une-histoire-de-conteneur.html#ss_part_2
– http://rvirtual.free.fr/programmation/java3d/intro.html
– http://www.web3d-fr.com/tutoriels/Java3D/1/tutorialjava3d.php

1.4 Visualisation du pavage JAVA2D

Technologie plus ancienne, elle est elle intégrée au JDK. JAVA2D permet de tracer diverses de figures géométriques en 2 dimensions Si les tests de JAVA2D s'avéraient très bon, il pourrait être envisagé de dédiée la représentation graphique en deux dimensions à JAVA2D. Sinon JAVA3D serait choisi pour toute la représentation graphique (2D et 3D)

1.4.1 Documentations

Sources

– http://java.sun.com/j2se/1.5.0/docs/guide/2d/index.html

1.4.2 Tutoriels

– http://www.siteduzero.com/tutoriel-3-10456-une-histoire-de-conteneur.html#ss_part_2
– http://duj.developpez.com/tutoriels/java/dessin/intro/

2 Tests pour la visualisation

Les résultats donnés aux sections 2.2, 1.3.1 et 1.3.2 sont tous obtenus selon un même type de test. L'objectif étant de choisir un langage permettant d'illustrer un pavage donné par realpaver. On rappelle qu'un pavage est un ensemble de pavés (ou boîtes) représentés par des intervalles flottants. Il est donc nécessaire d'avoir la connaissance des performances d'affichage d'un grand nombre d'entités graphiques. Ainsi l'algorithme conçu permet au lancement de l'application de fournir en paramètres le nombre d'entités (solutions) à afficher et la taille de la fenêtre en pixels. Le programme va par la suite générer une couleur, une position(2D ou 3D), et une taille aléatoire. La position et la taille sont cependant conditionnés par la taille de la fenêtre pour que l'entité reste dans le cadre de visualisation. La fenêtre est scrollable et peut être redimensionnée. L'algorithme à du être adapté pour supprimer les différents bugs graphiques lorsque l'on redimensionnait ou «scrollait» la fenêtre de test. Pour chaque technologies, les tests ont été effectué par des instances de classes prédéfinies. Ces classes étant optimisées, leurs utilisations ont été préféré à l'utilisation de primitives JAVA.

2.1 Données et outils utilisés dans l'évaluation des performances

CPU : Utilisation du processus en secondes. Elle est relevée à l'aide de le commande $\$top$. Au lancement de l'application test, c'est la valeur la plus élevée qui est retenue (cf man page top partie 2c)

Mémoire totale / Mémoire utilisée : Ces données s'obtiennent de la manière suivante. Une instance de l'objet Runtime que l'on initialise à l'aide de la méthode de classe Runtime.getRuntime () permet de stocker de nombreuses informations au lancement de l'application. Ainsi il suffit d'afficher les valeurs de $s\_runtime.totalMemory()$ et $s\_runtime.freeMemory()$ pour obtenir respectivement les valeurs de la mémoire totale dédiée à l'application et la valeur non utilisée de cette dernière en octets. Une simple différence permet d'obtenir la mémoire réellement utilisée par l'application.

2.2 Test de JAVA2D

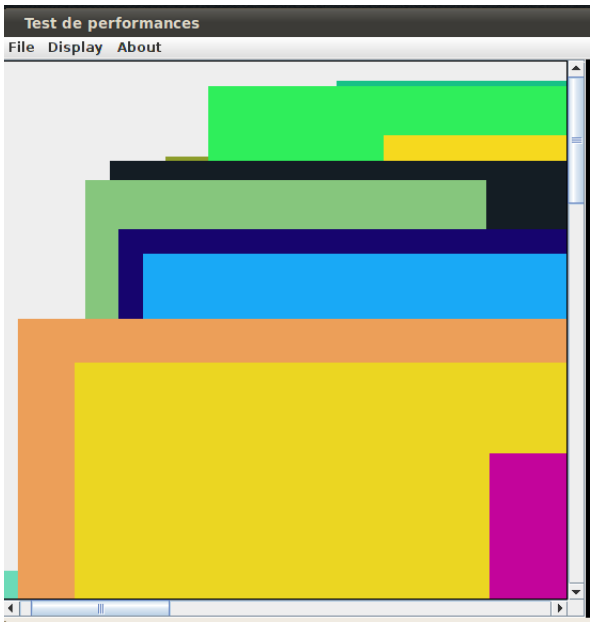

FIGURE 1 – Test dans une JFrame 500x500

Caractéristiques / Nombre de boîtes	CPU	Mémoire totale	Mémoire utilisée
10	0.39	52	4
100	0.4	52	4
1000	0.4	52	5
10000	0.44	52	6
100000	0.49	52	12

On constate que l'on peut atteindre un nombre de boîtes élevé, sans réquisitionner un taux de mémoire important. Cependant lors du test pour 1 000 000 de boîtes le système s'est retrouvé complétement bloqué, et un redémarrage matériel fut nécessaire.

2.3 JAVA3D

2.3.1 Test avec l'objet CubeColor

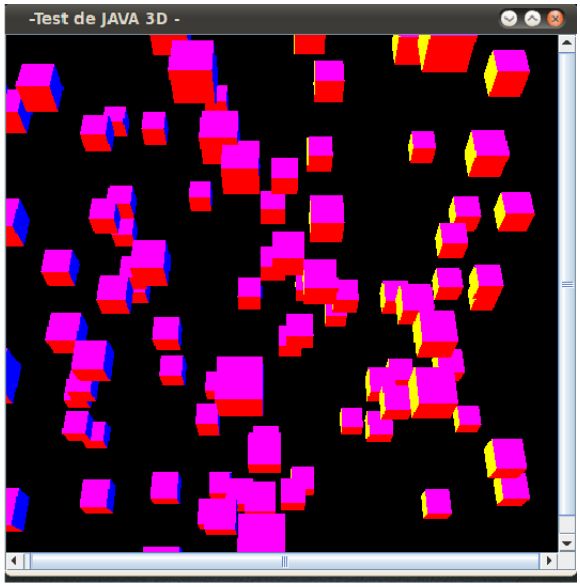

FIGURE 2 – Test dans une JFrame 500x500

Le motif est dessiné est une instance de la classe ColorCube. C'est le dessin d'un cube de couleur. Sa position t son orientation lui sont attribués au a la suite de sa création. C'est l'aspect de classe «primitive» qui a conditionné le choix de cette classe pour effectuer les premières tests de JAVA3D.

Caractéristiques / Nombre de boîtes	CPU	Mémoire totale	Mémoire utilisée
10	38	52	2
100	39	52	3
1000	53	52	10
10 000	66	91	51
100 000	100	636	494
125 000	100	776	563
150 000	100	776	678

On observe un rapport nombre de boîtes / occupation mémoire bien plus importante que pour le test JAVA2D. De plus le CPU est à partir d'un certain nombre de boîtes totalement monopolisé. A nouveau le système risque de se bloquer totalement pour un nombre de boîtes supérieur, entrainant un redémarrage manuel de la machine.

2.3.2 Test avec l'objet Box

Le motif dessiné est une instance de la classe Box. Elle permet de construire un parallélépipèdes aux dimensions et X, Y et Z différentes. Cette classe est celle qui se rapproche le plus du type d'entité (pavés ou boîtes) que l'on désire dessiner. Son implémentation est plus lourde que celle de ColorCube.

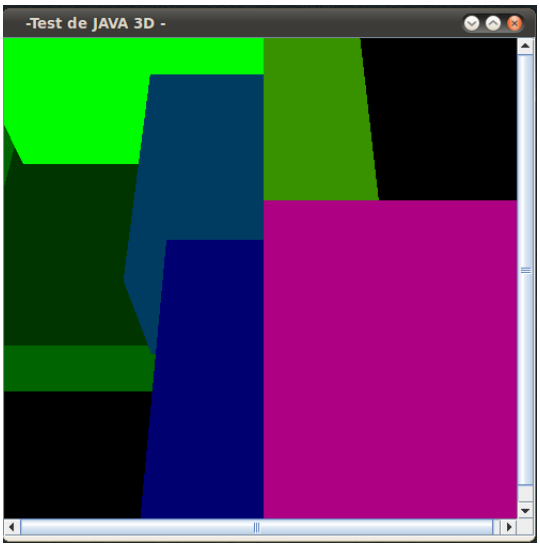

FIGURE 3 – Test dans une JFrame 500x500

Caractéristiques / Nombre de boîtes	CPU	Mémoire totale	Mémoire utilisée
10	40	52	3
100	46	52	7
1000	90	66	35
10000	100	360	255
25 000	100	743	587

Comme l'on pouvait si attendre les performances sont plus faible par rapport à l'utilisation de ColorCube. Presque 5 fois inférieur (Pour 10 000 boîtes 51Mo pour le cube contre 255Mo pour la box). On assiste à une erreur de compilation (Exception in thread «main» java.lang.OutOfMemoryError : Java heap space) pour un nombre de boîtes supérieur à 25 600.

2.4 Observations et conclusions

L'erreur soulevée au cours ces deux tests (Exception in thread «main» java.lang. OutOfMemoryError : Java heap space) peut être contournée en utilisant les options de compilation suivantes : $-Xms[initialheapsize]-Xmx[maximumheapsize]$. Cela permet d'augmenter le taux de mémoire dédiée à l'application.

Les résultats bien que plutôt satisfaisant devraient être probablement plusieurs dizaines de fois supérieurs sur une carte graphique moderne avec et le pilote adéquat. En effet les tests furent effectués sur une machine standard de laboratoire via un pilote générique Linux.

L'idée de poursuivre exclusivement avec la technologies JAVA3D même pour les l'aspect 2D est à l'heure actuelle privilégiée. Un test supplémentaire : représentation 2D avec la technologie JAVA3D pourrait rendre définitif ce choix.

3 Patrons de conception

L'application étant dotée d'une IHM, le patron de conception MVC (Modèle-Vue-Contrôleur) semble être approprié. L'idée d'y intégrer également un patron de conception Observer reste encore à approfondir.

3.1 Première tentative d'application

3.2 AbstractModel

```
public abstract class AbstractModel  implements Observable{
  protected DisplayCaracteristics displaycaracteristics;
  private ViewPointDefinition  viewpointdefinition;
  // couleur rvb (string) taille des pts (Number) police (String)
  /*Les items suivant sont sous forme de listes*/
  private String listIdandTypeCaracteristics = new ArrayList<String>();
  private ReferencesPoint listreferencespoint = new ArrayList<ReferencesPoint>();
  private Filter listfilter = new ArrayList<Filter>();
  private ConditionalDisplay listonditionalisplay = new ArrayList<ConditionalDisplay>();
  private Observer listObserver = new ArrayList<Observer>();

  //Afficher le pavage en memoire
  public abstract void reset();
  //Effacer le pavage
  public abstract void reset();
  //Ajouter un point de reference
  public abstract void addRP();
  //Effacer un point de reference
  public abstract void addRP(int id);
```

```
  //Ajouter un affichage conditionnel
  public abstract void addCD();
  //Effacer un  affichage conditionnel
  public abstract void addCD(int id);

  //Ajouter un filtre
  public abstract void addF();
  //Effacer un filtre
  public abstract void addF(int id);

  //Deplacement de caméra (clavier souris)
  public abstract void Move(Coord coord);

  //getter setter..

}
```