Projet développé par :

Chef de Projet

Alban KRAUS

Avec

Maryame RHEZALI

David NIAMIEN

Commanditaire

Emmanuel BARDIÈRE

**Benchmarks de navigation dans un nuage de points 3D**

**BenchOKR : Cahier des Charges Fonctionnelles**

**Sommaire**

[Introduction 2](#_Toc415047847)

[I. Analyse fonctionnelle 2](#_Toc415047848)

[1. Diagramme de cas d’utilisation 2](#_Toc415047849)

[1.1. Fonctionnalité : *Lire les données* 3](#_Toc415047850)

[1.2. Fonctionnalité : Découper les données 3](#_Toc415047851)

[1.3. Fonctionnalité : *Afficher les données* 4](#_Toc415047852)

[2. Tableau des fonctionnalités 5](#_Toc415047853)

## **Introduction**

Tout projet informatique nécessite une analyse avancée afin de définir les différentes fonctionnalités auxquelles notre système doit répondre. Pour cela, nous avons recours aux diagrammes UML (Unified Modeling Language).

En effet, ces diagrammes permettent de définir une application selon plusieurs points de vue :

* Fonctionnel : diagramme de cas d’utilisation
* Statique : diagrammes de classes, d’objets et de structure composite
* Dynamique : diagramme de séquence, d’états, d’activité, d’interaction, de communication et de temps.
* Implémentation : diagramme de composants, de déploiement et de paquetage.

Parmi ces 13 diagrammes que comporte UML, nous utiliserons pour l’aspect fonctionnel le diagramme de cas d’utilisation. Concernant, l’aspect dynamique le diagramme d’activité et enfin le diagramme de classe, pour l’aspect statique.

# **Analyse fonctionnelle**

# **Diagramme de cas d’utilisation**

Le diagramme de cas d’utilisation "*use cases*" permet la description du comportement du système du point de vue de l’utilisateur sous forme d’actions et de réactions. Un acteur représente le rôle joué par un objet (personne ou chose) qui interagit avec le système (DEBRAUNWER & VAN DER HEYDE, 2008). Un objet peut jouer plusieurs rôles vis-à-vis du système.

Nous avons donc réalisé un diagramme de cas d’utilisation comportant trois principales fonctionnalités :

* ***Lire les données***
* ***Découper les données***
* ***Afficher les données***

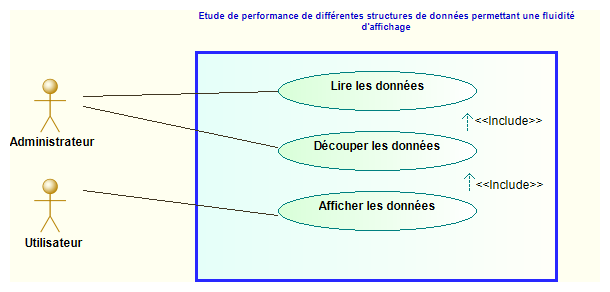


Figure *1*: diagramme de cas d'utilisation

## **Fonctionnalité : *Lire les données***

Cette fonctionnalité permet de "*lire les données*". Elle nécessite différentes sous-fonctionnalités dont :

1. *Stocker le nuage sur un support physique* : hors de notre projet

2. *Ouvrir les données* : opération réalisée par le système d'exploitation, hors du projet

3. *Connaître la structure du format de fichier*

4. *Stocker en mémoire* : cela suppose de définir une structure de données adaptée. En pratique, il sera impossible de stocker le nuage entier en mémoire. En dehors des métadonnées, un point est constitué, sur une machine 64 bits, de :

* indice (int)=32 bits=4 octets
* trois coordonnées : 3\*
* une coordonnée (double) = 128 bits = 16 octets soit 48 octets

Un point aura donc une taille en mémoire de 52 octets. Un nuage d'un milliard de points a donc une taille de 52 milliards d'octets soit une cinquantaine de Go, pour une mémoire vive de 4 à 8 Go), il faudra donc soit utiliser l'espace d'échange, soit ne lire qu'une partie du nuage. La structure de données devra dans tous les cas être adaptée à un grand nombre de points.

|  |  |
| --- | --- |
| **Cas d’utilisation** | **Lire les données** |
| Résumé | Ce cas d’utilisation permet de lire les données. |
| Acteur | Administrateur |
| Système | BenchOKR |
| Niveau | 1 |
| Préconditions |  |
| **Opérations** | |
| 1.1 | Stocker en mémoire les données lues |

Tableau 1: fonctionnalité "lire les données"

## **Fonctionnalité : *Découper les données***

Cette fonctionnalité permet de "*découper les données*". Elle sera implémentée trois fois, par nos trois méthodes. Quelle que soit la manière dont on les codera (trois exécutables ou un seul ?), l'opérateur chargé de l'étude doit être capable de sélectionner une méthode (**fonctionnalité 2.1.**). Le point commun de nos trois méthodes de découpage est qu'elles sont arborescentes : il faut donc définir ce que représente l'arbre, et plus précisément ce que sont les nœuds (fonctionnalité 2.3.) et les feuilles (**fonctionnalité 2.2**). D'un point de vue programmeur, la définition de ces deux éléments est liée à la spécification des structures de données associées. La définition de la structure de données des feuilles requiert de définir le mode de stockage (base de données ou système de fichiers) (**fonctionnalité 2.2.2**), et de définir le nombre de points par fichier (**fonctionnalité 2.2.1**). En ce qui concerne les nœuds, la définition de la taille des feuilles permet de déduire la hauteur de l'arbre (**fonctionnalité 2.3.1**), et il faut définir la structure de données pour les nœuds différente pour chaque méthode.

|  |  |
| --- | --- |
| **Cas d’utilisation** | **Découper les données** |
| Résumé | Ce cas d’utilisation permet de construire la structure des données. |
| Acteur | Administrateur |
| Système | BenchOKR |
| Niveau | 1 |
| Préconditions | Lire les données |
| **Opérations** | |
| 2.1 | Choisir une structure |
| 2.2 | Stocker les feuilles   * 2.2.1. identifier la taille * 2.2.2. choisir le mode de stockage |
| 2.3 | Stocker la structure de l’arbre   * 2.3.1. Déterminer la hauteur de l’arbre * 2.3.2. stocker les nœuds |

Tableau 2: fonctionnalité "découper les données"

## **Fonctionnalité : *Afficher les données***

La fonctionnalité n°3 est "*Afficher les données*". Cette fonctionnalité fait le lien entre le nuage de points et l'écran de l'utilisateur. L'utilisateur définit préalablement un champ de vision, il nous faut d'abord récupérer tous les points qui doivent y être affichés (**fonctionnalité 3.1**). Selon la demande de l'utilisateur, on peut distinguer deux méthodes pour récupérer les bons points : soit l'utilisateur vient de lancer le logiciel, et nous devons donc récupérer les points qui se situent dans son volume de vision (**fonctionnalité 3.1.1**) ; soit l'utilisateur vient de déplacer légèrement son champ de vision, et une requête sur un voisinage pourrait suffire (**fonctionnalité 3.1.2**). Une fois tous les points pertinents récupérés, nous devons être capables de les afficher à l'écran (**fonctionnalité 3.2**).

|  |  |
| --- | --- |
| **Cas d’utilisation** | **Afficher les données** |
| Résumé | Ce cas permet de passer des données structurées à la représentation visuelle. |
| Acteur | Utilisateur |
| Système | BenchOKR |
| Niveau | 1 |
| Préconditions | Découper les données |
| **Opérations** | |
| 3.1 | Obtenir les points à afficher   * Requête sur un point pour obtenir un voisinage * Requête sur un volume pour obtenir les points inclus dans ce volume |
| 3.2 | Afficher les points |

Tableau 3: fonctionnalité "afficher les données"

# **Tableau des fonctionnalités**

La lecture du nuage revient à lire un fichier : c'est une opération de base très documentée, donc facile. Un test intuitif est de vérifier la correspondance entre les données en mémoire (à afficher) et ce qui est contenu dans le fichier (ouverture avec un autre programme).

Tout le reste de notre démarche se fonde sur une structure arborescente. Il est primordial que cette structure soit valide afin que les algorithmes restent prévisibles. Stocker les feuilles signifie que tous les points doivent être représentés dans notre nouveau système. La taille des fichiers influe directement et indirectement sur la vitesse des requêtes ; c'est un paramètre de la méthode. Ce paramètre influe également sur la hauteur de l'arbre, donc encore une fois sur la vitesse des requêtes. Le mode de stockage est une solution technique, et doit permettre de stocker durablement le nuage de points et permettre un grand nombre d'accès en lecture. Dans l'idéal, on souhaiterait que la taille du nuage découpé n'augmente pas trop. La structure de l'arbre a une forte influence sur la durée des requêtes ; dans la plupart des méthodes de découpage, la hiérarchie est définie et la hauteur est paramétrée. Ces fonctionnalités ne sont pas si faciles, car dans l'hypothèse où elles sont déjà codées, nous devons avoir un regard critique sur la manière dont elles le sont ; et nous devront peut-être les coder nous-même. Néanmoins, nous n'avons pas de contrainte forte sur le temps mis pour découper le nuage.

La troisième partie concerne la requête des points et leur affichage. C'est cette partie qui est la cible de l'étude de performances, et elle doit donc être particulièrement soignée. Dans l'hypothèse où on n'aurait pas le temps de coder deux méthodes de requête (par voisinage et par volume englobant), on peut noter qu'il est toujours possible de récupérer facilement des points dans un volume, alors qu'une méthode par voisinage nécessite de définir rigoureusement ce qu'est le voisinage. L'affichage des points proprement dit nécessite de bonnes connaissances en matériel graphique, à moins que cette fonctionnalité ne soit déléguée à une bibliothèque existante.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fonctionnalité** | **Priorité** | **Réalisation** | **Difficulté** | **Tests** |
| 1. Lire les données | 1 | 10% | Facile |  |
| 1.1. Stocker en mémoire les données lues | 1 | 0% | Facile | Les 5 premiers points sont identiques à ceux du fichier |
| 2. Découper les données | 1 | 0% | Moyen | La structure de l’arbre est valide |
| 2.1. Choisir une structure | 1 | 0% | Facile | On peut changer la méthode utilisée |
| 2.2. Stocker les feuilles | 1 | 50% | Facile | Tous les points du nuage initial sont dans au moins une feuille. (\*) ou un échantillon représentatif … |
| 2.2.1 Identifier la taille | 1 | 0% | Moyen | Minimise l’indicateur « durée des requêtes » |
| 2.2.2 Choisir le mode de stockage | 5 | 0% | Moyen | Minimise l’indicateur « rapport taille du nuage initial / stocké » |
| 2.3 Stocker la structure de l’arbre | 1 | 0% | Moyen | Cf. infra |
| 2.3.1 Déterminer la hauteur de l’arbre | 1 | 0% | Moyen | Minimise l’indicateur « durée des requêtes » |
| 2.3.2 Stocker les nœuds | 1 | 0% | Moyen | La structure de l’arbre est vérifiée |
| 3. Afficher les données | 1 | 0% | Difficile | Cf. infra |
| 3.1 Obtenir les points à afficher | 1 | 0% | Difficile | Tous les points contenus sont récupérés |
| 3.1.1 Requête sur un point pour obtenir un voisinage | 2 | 0% | Difficile | Tous les points correspondants sont récupérés |
| 3.1.2 Requête sur un volume | 1 | 0% | Moyen | Tous les points contenus sont récupérés |
| 3.2 Afficher les points | 2 | 0% | Difficile | Les points sont affichés au bon endroit.  Les points sélectionnés sont affichés. |

Tableau 4: tableau des fonctionnalités