Argann BONNEAU
Nicolas GAILLARD
Yinuo LI
Tom MARRUCCI
Etienne MARTIN

POLYTECH NANTES

# Mini-projet de programmation en langage C++ : Satellite++

Hash Code 2016

## Table des matières :

# 1. Introduction

## a. Présentation rapide du projet

Hash code est un concours de programmation par équipe organisé par Google ouvert aux étudiants et aux ingénieurs professionnels d'Europe, du Moyen-Orient et d'Afrique. Il consiste à plancher sur un problème inspiré de situations rencontrées par les ingénieurs Google. La compétition est organisée en deux temps : une première phase de qualification se déroulant dans toute l'Europe puis une phase finale qui a lieu directement dans les locaux de Google France à Paris.

## b. Problème à résoudre

Cette année, nous avons été confrontés à la phase finale du Google Hashcode 2016 visant à maximiser la prise de photos de coordonnées géographiques. En effet, les satellites équipés d'appareils photos de haute qualité sont de très bonnes sources d'imagerie satellite puisqu'ils peuvent fournir un flux constant de données. Une nouvelle division de Google, nommée *Terra Bella*, déploie et gère tout ce qui touche à l'imagerie satellite. Le nombre grandissant de constellation et le besoin constant d'images récentes en ont fait un important enjeu.

# 2. Stratégie

## a. Diagramme de classe

Struct data est le squelette qui va contenir les données des différents fichiers. Nous avons créé des structures afin de regrouper les données de même nature. Ces dernières sont instanciées dans des tableaux à l'intérieur d'une classe nommée SimulationData. Les données sont extraites avec la classe DataReceiver. Puis le SimulationData est donné en argument à NaiveSolution qui s'occupe de résoudre le problème ainsi que d'écrire les résultats.

## b. Solution naïve

Au début de ce projet, nous avons commencé par réfléchir sur la meilleure façon de représenter mathématiquement et algorithmiquement le problème lié à ce projet.

Nous avons eu de nombreuses idées, notamment sur la possibilité ou non de pouvoir trouver une solution sans avoir à "lancer" une simulation tour par tout, via différents calculs de positions, de résolution d'équation. Finalement toutes ces idées soient ne prenaient pas suffisamment en compte les contraintes

du projet, soit étaient beaucoup trop ambitieuses, ou parfois pas suffisamment optimales pour être intéressantes.

Comme beaucoup de groupes, notre premier objectif "pratique" a donc été de nous pencher sur la solution dite "naïve". Cette solution à un fonctionnement assez simple :

- Pour chaque tour de simulation (de 0 au nombre total de tour précisé dans le fichier d'entrée)...
- Nous parcourons chaque satellite, que nous déplaçons convenablement, puis pour chaque satellite…
- Nous parcourons la liste des collections existantes, puis pour chaque collection…
- Nous parcourons la liste d'images à prendre. Si le satellite courant à la possibilité de prendre une photo, alors il l'a prend. Si des conflits surviennent, c'est à dire si deux photos peuvent être prises au même moment par un unique satellite alors il capture la première qu'il trouve, et abandonne la deuxième.

Cette solution, bien que peu optimale développée telle qu'elle, a le mérite d'être simple à imaginer, et relativement rapide à programmer.

Nous avons ensuite pensé à différentes façon de l'optimiser, que nous allons voir dans la partie suivante.

### c. Optimisations implantées

Notre idée pour l'implémentation était la suivante.
Il est extrêmement dur de dépasser les 10% de réussite sur une collection.
Partant de ce fait nous trions les collections avec l'indice suivant

score que rapporte la collection / nombre de points à prendre ²

Avec la tailles des données du jeu nous n'avons aucunes garanties de pouvoir prendre un point donné.
Plus le nombre de points augmente plus il est dur de finir une collection
D'où l'importance sur le dénominateur
Une fois la collection triée nous supprimons les pire d'entre elles ( cad celle avec l'indice le plus faible ) selon un seuil que nous expérimentons.
Les seuils les plus performants varient selon les collections.
Nous supprimons en plus les photos qui dépassent un nombre d'images trop grand ou un score trop petit.

De plus comme la collection est classée le programme prendra en priorité les images des collections avec un fort indice.

## 3. Arbitre

### a. Classement

#### i. Calcul du score :

Le calcul du score est l'étape la plus importante lors de l'arbitrage. Pour ce faire, on récupère en premier lieu l'ensemble des images capturées à partir du fichier de sortie puis l'ensemble des collections (issu du jeu de données initial). Une fois cela fait, on parcourt la totalité de chaque collection et on vérifie que celle-ci soit complète (via un parcours des photographies capturées récupérées précédemment). Si elle ne l'est pas, on ne compte pas les points de la collection. Il suffit ensuite de sommer les points de chaque collection photographiée.

#### ii. Calcul du temps d'exécution :

Le temps d'exécution est une donnée supplémentaire permettant de mesurer les performances d'une solution. Afin de le mesurer, on relève une première fois le temps avant son exécution puis une seconde fois à la fin de celle-ci. Il suffit de faire la différence entre ces deux temps relevés pour déterminer le temps d'exécution.

#### iii. Le meilleur compromis :

Comme dit précédemment, le score ainsi que le temps d'exécution déterminent les performances d'une solution. Un simple ratio est calculé (score divisé par le temps d'exécution) afin de départager des solutions ayant un score semblable.

### b. Validation

#### i. Vérification 1 :

La première validation consiste à vérifier la construction syntaxique du fichier de sortie. Pour cela nous avons utilisé des *regex* (ou expressions régulières) qui permettent de vérifier simplement chaque ligne du fichier.

Une rapide vérification sémantique a également été réalisée, où l'on s'assure que :

- Les coordonnées des images ne dépassent pas les coordonnées maximales
- Le tour courant ne dépasse pas le nombre maximal de tours
- Le nombre d'images prises ne dépasse pas le nombre maximal d'images

#### ii. Vérification 2 :

La deuxième vérification consiste, quant à elle, à s'assurer que chaque image peut bien être prise par un satellite à un tour donné. Pour cela, on

parcourt chaque ligne du fichier de sortie, on récupère le satellite (grâce à son identifiant) et on calcule sa position au tour donné. Il suffit ensuite de contrôler si l'image était bien dans la portée du satellite ou non.

iii. Vérification 3 :

La dernière vérification du fichier de sortie consiste à contrôler qu'entre deux images consécutives prises par un même satellite, la caméra a bien le temps de tourner pour capturer l'image suivante.

Pour ce faire, un parcours de chaque ligne du fichier de sortie est réalisé. On sauvegarde alors la première image prise par un satellite puis on cherche la suivante. On peut alors calculer la variation de longitude et de latitude que la caméra a à réaliser afin de prendre la seconde photo. Si cette variation est inférieure au produit de la vitesse de rotation par la différence des deux tours on rejette alors la solution.

## 4. Difficultés rencontrées

Plusieurs problèmes (plus ou moins importants) ont été rencontrés lors du projet.

Le premier a été lié à un problème de communication interne. En effet, nous n'avions jamais travaillé ensemble et il a fallu un certains temps d'adaptation.

Des solutions plus complexes (et surtout plus optimales) avaient été imaginées. Nous avons notamment pensé à réaliser un arbre de solutions potentielles puis de le parcourir afin de proposer la meilleure solution. Suite à des tests non concluants, nous avons abandonné cette idée. D'autres optimisations mineures étaient prévus mais n'ont pas été implémentées de part leur complexité et par manque de temps.

En effet, le temps imparti pour la réalisation de ce projet coïncidait avec d'autres rendus.

## 5. Résultats

Voici nos meilleurs résultats sur les collections :

| | | | | | |
|---|---|---|---|---|---|
| overlap.in | 10overlap.out | 54041 (4.65%) | 10poverlap | 15/12/2016 19:22:25 | |
| constellation.in | weekendtestwhithoutseuil.out | 89922 (3%) | Point20Deleted10P120scoreminconst | 16/12/2016 15:31:08 | |
| weekend.in | weekendtestwhithoutseuil.out | 60624 (0.9%) | 20pweekendcorrection | 16/12/2016 14:54:35 | |
| forever_alone.in | 6sqaureperalone2.out | 47376 (33.99%) | 6psquarealone | 15/12/2016 19:01:20 | |