Compte-rendu de la réunion avec le commanditaire

4 mars 2015, 13h30 à 15h

Salle M209

# Besoin du commanditaire

Le commanditaire reformule notre sujet en « étude des performances [=benchmark] de différentes méthodes pour assurer la fluidité de l’affichage d’un très gros nuage de points ». Nous avons un fichier de départ, qu’il faudra sans doute « stocker d’une autre façon », et même de plusieurs façons différentes. Ce nuage nous sera sans doute fourni sous un ensemble de points décrits par leurs trois coordonnées, avec peu de métadonnées. Il faudra ensuite afficher notre nuage, de la même manière afin que les méthodes de stockage soient comparables. Voici un aperçu visuel de la démarche de notre projet :

fichier

stock

affichage

À partir du besoin formulé, nous devons extraire des **fonctionnalités**. Par exemple, « lire le fichier », « découper le fichier », « stocker les feuilles (et l’arbre) ». Ces fonctionnalités devront ensuite être détaillées en un ou deux niveaux de détail supplémentaires, par exemple « lire un fichier » pourra être détaillé en « choisir un conteneur de données », où le conteneur de données est dynamique et stocké dans la mémoire vive ; « déterminer la taille des feuilles acceptable » ; etc.

Avant l’affichage, il va falloir effectuer des requêtes. Ces requêtes doivent-elles être identiques quelle que soit la méthode de stockage employée, ou pourront-elles être adaptées ?

Comme notre projet est une étude de performances, il va nous falloir des indicateurs. Par exemple, le nombre d’images par secondes (fps) qui mesure la vitesse de rafraîchissement de l’écran. Attention, cet indicateur dépend de la carte graphique utilisée. On pourra aussi mesurer le temps mis pour transformer le fichier, la durée d’une requête, le temps mis pour rechercher les voisins.

Nous avons fait un point sur le diagramme de classes : stocker des objets et leurs relations.

# Bribes de commentaires techniques

## Web et WebGL

L’IGN a travaillé sur une interface Web mettant à profit l’architecture client-serveur. L’idée ayant conduit à ce choix est de ne pas avoir à recopier le nuage sur chaque machine où l’on doit l’afficher.

Une telle architecture convient peu à notre projet, étant donné que la qualité du réseau peut fortement altérer les performances que nous cherchons justement à évaluer. Nous devons être conscients que WebGL utilise la carte graphique du client

## C, OpenGL, et librairies

Pour OpenGL, notre référent technique nous conseille vivement d’utiliser une caméra qui existe déjà. (freefly ?)

En C/C++, la STL (Standard toolkit library) fournit plusieurs structures de données et leurs méthodes utiles. En ce qui concerne l’affichage, libQGLviewer est une interface Qt et OpenGL.

La librairie OpenMP s’interface avec plusieurs langages. Elle repose sur la notion de pragmas (directives de précompilation)

## Multithreads

La plupart des processeurs ont plusieurs cœurs et peuvent donc exécuter plusieurs files de tâches en même temps. À la différence d’autres langages, JavaScript n’est pas capable d’utiliser cette technique (cependant, on peut s’en rapprocher avec NodeJS avec des setTimeout). Attention, si on décide d’utiliser cette technique dans notre projet, les trois méthodes doivent y avoir recours ce qui complexifie la tâche.

## Stockage

Nous avons déjà vu la possibilité d’un stockage dans une base de données ou un système de fichier. On a brièvement parlé d’une base de données stockant des fichiers binaires (« blobs »)

## GitHub

À la racine des répertoires se trouve en général un fichier qui décrit le projet. Tous les commits sont attribués à une adresse mail associée à un compte, donc il n’y aura pas d’ambiguïté lors de l’évaluation.

L’école dispose d’un serveur GitHub qui n’est accessible que localement.

Il existe également des services tels qu’une plateforme de gestion de contenu ou un bug-tracker, mais ils ne nous seront guère utiles pour un si petit projet. Attention, si on le mentionne, il faudra s’être documenté : « Pouvez-vous me citer le nom d’une plateforme de gestion de contenu ? »

## Cloud Compare

Cloud compare utilise une version compressée des octrees, qui sont générés au chargement du fichier.

## Rapport

Nous aurons un glossaire : il faudra définir des mots tels « vexel », « texel », « pixel », « point 3D » ainsi que les formats de fichier (ply pour « PoLYgone », alors que nous on ne stocke que des points).

## Soutenance

Si nous faisons une démonstration, il nous faut la scénariser. Par exemple, choisir un nuage et l’ouvrir avec un logiciel indépendant, dire qu’on va essayer de faire pareil avec d’autres méthodes, et là seulement l’ouvrir avec notre logiciel.

## Comité de pilotage

Lors de la réunion du comité de pilotage, nous devront présenter le recueil du besoin, les fonctionnalités que nous avons isolées, le **détail** de ces fonctionnalités, et les tests unitaires associés. Les fonctionnalités doivent être détaillées au point de pouvoir déterminer précisément le temps nécessaire à leur réalisation. Pour les tests unitaires, nous savons déjà que Python permet de les réaliser facilement : comment faire en C++, ou dans n’importe quel autre langage que nous choisirons ? Un exemple de test unitaire pour la fonctionnalité fictive « lecture d’un fichier ply » est : « le point n°42 a pour coordonnées …, …, … ». [Pourquoi 42 ?]

En pratique, on pourra stocker ces fonctionnalités dans un tableau : fonctionnalité, priorité, réalisation (%), difficulté, tests unitaires associés.

Lors du comité de pilotage, ne pas oublier de mentionner notre GitHub.

## Divers

Licence : l’école est propriétaire des travaux que nous effectuons, mais une licence libre est possible.

En pratique, nous nous apercevrons sans doute qu’une fusion des trois méthodes peut être intéressante.

En conclusion, nous avons abordé des idées relatives à la solution technique sans pour autant la fixer, ce qu’il va nous falloir faire lors d’une prochaine séance. Les langages suivants ont été mentionnés : WebGL (et JavaScript), Python, C/C++.