Observatoire astronomique
de Strasbourg

Rapport de stage

Visualisation progressive de données astronomiques en 3D dans un navigateur

Auteur :
Nicolas Adam

Responsables :
M. André Schaaff
M. Pierre Tellier

5 août 2016

Rapport de stage

Visualisation progressive de données astronomiques en 3D dans un navigateur

Auteur :
Nicolas Adam

Responsables :
M. André Schaaff
M. Pierre Tellier

ENSIIE Strasbourg : 1 rue Jean-Dominique Cassini,
67400 Illkirch-Graffenstaden

Observatoire Astronomique de Strasbourg : 11 Rue de l'Université,
67000 Strasbourg

5 août 2016

Remerciements

Je tiens avant tout à remercier André Schaaff, qui m'a encadré et conseillé durant ce stage et m'a donné les moyens de le réaliser dans de bonnes conditions.

Je souhaiterais également remercier Pierre Tellier, qui a suivi l'avancée de mon stage et m'a rendu visite à l'observatoire.

Mes remerciements s'adressent également à l'ensemble du personnel de l'observatoire pour son accueil chaleureux, ainsi qu'aux stagiaires qui m'ont tenu compagnie à la bibliothèque.

J'adresse des remerciements particuliers à Jérôme Desroziers, pour son aide précieuse et ses connaissances de l'application.

Enfin je tiens à remercier Hervé Wozniak, directeur de l'Observatoire Astronomique de Strasbourg, ainsi que Mark Allen, directeur du Centre de Données astronomiques de Strasbourg, pour m'avoir accueilli dans leur établissement.

Sommaire

Introduction

Pour valider l'obtention de mon diplôme d'ingénieur en informatique à *l'École Nationale Supérieur d'Informatique pour l'Industrie et l'Entreprise*, j'ai l'opportunité d'effectuer chaque année un stage en entreprise. Étant actuellement en deuxième année, mon stage doit durer 10 semaines.

J'ai choisi d'effectuer mon stage à *l'Observatoire Astronomique de Strasbourg*, du lundi 31 mai au vendredi 5 août. J'ai été encadré par M. André Schaaff, ingénieur de recherche en informatique à l'Observatoire depuis 2001. Le sujet de ce stage fût la *Visualisation progressive de données astronomiques en 3D dans un navigateur.*

J'ai très vite été intéréssé par ce sujet de stage. Il nécessite un grand travail de recherche et de réflexion pour être mené à bien. Les technologies utilisées sont intéréssantes et sont directement liés au master d'*Informatique et Sciences de l'Image* que je suis à l'Université de Strasbourg (*tree.js - WebGL*). Une autre facette de ce stage est la manipulation de gros volumes de données, domaine que je suis curieux d'aborder.

Enfin, l'opportunité d'intéragir avec des ingénieurs expérimentés est une bonne occasion de découvrir le quotidien d'un métier que je souhaite exercer.

1 L'environnement professionnel

1.1 L'Observatoire Astronomique de Strasbourg

L'Observatoire astronomique de Strasbourg est un Observatoire des Sciences de l'Univers, une école interne et UFR de l'Université de Strasbourg, ainsi qu'une Unité Mixte de Recherche entre l'Université et le CNRS.

FIGURE 1 – Photographie de la grande coupole

Il a été fondé en 1881 avec pour vocations initiales l'astronomie de position et l'observation de comètes, de météorites et d'étoiles variables. Il possède la troisième plus grande lunette astronomique de France. L'observation du ciel n'est plus pratiquée depuis longtemps à l'Observatoire en raison de la polution lumineuse de Strasbourg.

Il doit remplir plusieurs missions :

— **Recherche** : *Voir les équipes de recherche ci-dessous.*

— **Enseignement** : L'observatoire délivre la spécialité astrophysique du master de sciences mention physique de l'université de Strasbourg. Il participe également aux enseignements des licences et masters ainsi qu'à la préparation à l'agrégation et au CAPES.

— **Service d'Observation** : Mise à disposition d'outils par le CDS

- *Simbad* (identification, bibliographie de 3 millions d'objets hors système solaire), *VizieR* (service de catalogues), *Aladin* (atlas du ciel donnant accès à plus de 5 téraoctet d'images).

— **Diffusion des connaissances** : C'est le planétarium qui a pour vocation la diffusion des connaissances auprès du grand public. L'Observatoire y participe lui aussi lors de divers évènements, comme *La nuit des étoiles* dernièrement.

L'Observatoire est structuré en trois équipes de recherche :

— **L'équipe Hautes énergies** : étudie les sources galactiques et extragalactiques émettrices en rayons X, les objets compacts (étoiles à neutrons, naines blanches, etc.) et les noyaux actifs de galaxies.

— **L'équipe Galaxies** : étudie la formation, l'évolution et la dynamique des galaxies, ainsi que les composantes stellaires et gazeuses qui les composent.

— **Le CDS** : *Centre de Données Stellaires* puis *Centre de Données Astronomiques de Strasbourg* à partir de 1983. Collecte des informations sur les objets astronomiques et les diffuse à la communauté scientifique. Développe de nombreux outils d'accès aux données astronomiques. J'ai fait partie de cette équipe.

1.2 Le cadre de travail

L'Observatoire Astronomique de Strasbourg se situe au coeur de Strasbourg, au 11 Rue de l'Université, à côté du jardin botanique. L'Observatoire dispose lui aussi d'un jardin, dans lequel il est possible de se restaurer.

L'Observatoire Astronomique de Strasbourg est composé de 3 bâtiments reliés entre eux par un tunnel en forme de Y :

— **La grande coupole** : C'est ici que sont situés l'administration et les salles de cours. La lunette astronomique se trouve dans ce bâtiment.

— **Le bâtiment SUD** : Des chercheurs travaillent dans ce bâtiment. C'est ici que se trouve la *menuiserie*, composée d'une cuisine et d'une salle de pause.

— **Le bâtiment EST** : Ce bâtiment comporte lui aussi des bureaux et la bibliothèque.

J'ai travaillé dans la bibliothèque de l'Observatoire avec plusieurs autres

stagiaires. 4 postes informatiques y sont disposés en *open-space*[1], ce qui facilite les discussions et les débats.

Tous les vendredis sont organisés des présentations suivies d'une pause café accompagnée de viennoiseries. J'ai également eu la chance de participer à une représentation très intéréssante au Planétarium ainsi qu'au *Repas du Solstice*, un repas convivial organisé à l'Observatoire où chacun amène un peu à manger.

Le personnel très sympathique, la bonne ambiance qui règne dans les locaux et le cadre de travail verdoyant m'ont permis de passer un excellent stage.

1.3 Ma mission

Durant mes 10 semaines de stage, j'ai eu la chance de travailler sur une application de visualisation dans le cadre de mon sujet de stage *Visualisation progressive de données astronomiques en 3D dans un navigateur.*

Le développement de l'application n'a pas débuté lorsque je suis arrivé, plusieurs stagiaires ayant déjà travaillé dessus au cours de l'année passée. L'application était déjà fonctionnelle lorsque j'ai commencé mon stage. Elle permettait alors de charger et visualiser des données astronomiques dans le navigateur à partir de fichiers présents sur l'ordinateur de l'utilisateur.

Une fois les données chargées, elles étaient affichées sous la forme d'un nuage de points dans le navigateur. Il était ensuite possible de se déplacer dans le nuage de points et de manipuler les données à l'aide de certains outils : un outil de zoom, un outil de coloration, etc.

Les performances de l'application étaient correctes : il était possible d'afficher 2 millions de points de façon fluide avec une carte graphique basique[2]. Ce n'est cependant pas un nombre suffisant pour les énormes jeux de données des astronomes. L'objectif de mon stage est la recherche et la mise en place d'une méthode pour contourner cette limitation.

1. Espace de travail dans lequel les bureaux ne sont pas séparés par des cloisons.
2. cf. Figure 2

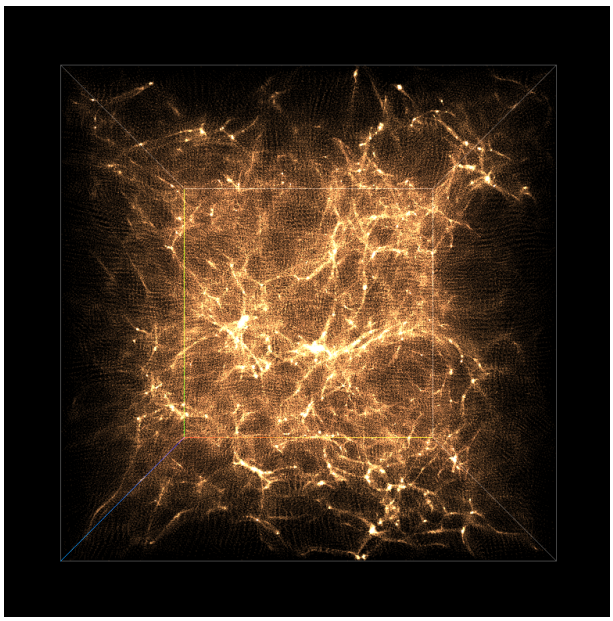

FIGURE 2 – Capture d'écran de l'affichage de 2 millions de points

2 La mise en oeuvre de la mission

2.1 Le déroulement du stage

2.1.1 L'objectif final

Comme expliqué précédemment, l'application était assez vite limitée par le nombre de points à afficher. Il est intéressant de pouvoir afficher des volumes de données très élevés, de l'ordre du téraoctet [3] (en sachant que le volume des 2 millions de points affichés est de 80 Mo).

Cette limitation est due au mode de chargement des données. Comme elles sont chargées depuis l'ordinateur de l'utilisateur, il est nécessaire de tout insérer dans la RAM de la machine. De plus, il est essentiel de traiter les données avant de pouvoir les afficher, ce qui prend un temps proportionnel à leurs volumes. Il est donc impossible de charger des fichiers trop importants.

La recherche et la mise en place d'une solution à ce problème est l'objectif de mon stage. Il devrait être possible de charger des données d'un volume d'un téraoctet, ce qui signifie plusieurs milliards de points.

Dans ce but, je vais développer une partie serveur pour l'application. Une base de données sera utilisée pour stocker les nombreuses données astronomiques. Le serveur enverra dynamiquement les données nécessaires à l'application lors de son utilisation.

Une fois son développement terminé, cette application sera utilisée par des astronomes pour visualiser leurs données astronomiques. Elle pourra aussi être utilisée dans d'autres domaines, en imagerie médicale par exemple.

2.1.2 Les outils utilisés

Ce projet est plutôt important et a nécessité de nombreuses technologies. Voici les principales :

— **Javascript** : la logique de l'application a été écrite en javascript *simple*, c'est à dire sans utiliser de framework.

— **Three.js** : bibliothèque 3D javascript. Permet l'affichage et le raf-

3. Unité multiple de l'octet. Correspond à 10^{12} octets.

fraichissement du nuage de points et des différents élements de l'application.

— **Node.js** : permet l'exécution de code javascript côté serveur. Les entrées et sorties sont traitées de manière non bloquante.
— **MongoDB** : Base de données NoSQL [4] orientée document. MongoDB a été utilisé pour stocker les points des simulations dans une base de données.
— **CSS** : Les menus de l'outil ont été écrit par un autre stagiaire (*Thibault Bouchard*) en CSS [5].

En complément de ces technologies, j'ai utilisé certains programmes pour faciliter mon travail :

— **Webstorm**, un IDE [6] pour javascript.
— **Firefox**, un navigateur web que j'utilisais pour exécuter l'application. J'ai installé l'extension *firebug*, permettant de débugger l'application de manière efficace.
— **Robomongo**, un outil permettant la visualisation et la modification d'une base de données MongoDB dans une interface graphique.
— **GitLab**, un logiciel client Git, permettant la gestion de versions.

2.1.3 Les recherches préliminaires

Lorsque je suis arrivé le premier jour, André, mon tuteur, m'a rapidement présenté les locaux ainsi que les personnes que j'allais cotoyer durant mon stage.

J'ai ensuite commencé la lecture des rapports de stage de Pierre Lespingal et Arnaud Steinmetz, deux stagiaires ayant travaillé sur l'application l'an dernier. J'ai également effectué beaucoup de tests pour tenter de comprendre le fonctionnement des différentes classes du programme les unes par rapport aux autres.

4. Not only SQL : Catégorie de système de gestion de base de données qui n'est pas uniquement fondée uniquement sur l'architecture classique des bases de données relationnelles.
5. Cascading Style Sheets - feuilles de style en cascade. Langage qui décrit la présentation des documents HTML.
6. Integrated Development Environment - Environnement de développement intégré. Ensemble d'outils pour augmenter la productivité des programmeurs.

J'ai ensuite recherché les différents moyens qui permettraient de stocker de gros volumes de données et d'effectuer des requêtes dessus. Trois solutions s'offraient à moi :

— **Les nanocubes** : structure de données permettant la visualisation rapide de très gros volumes de données spatio-temporelles et d'attributs, tout en restant modeste en taille[7]. J'ai décidé d'écarter cette idée. Le nanocube décrit dans le papier de recherche ne nous intéresse pas puisque les attributs et la *temporalité* ne nous sont d'aucune utilité. Seul la dimension spatiale est intéressante.

— **Une base de données noSQL** : c'est la solution que j'ai retenue, avec MongoDB (SGBD orienté document). L'idée est de compartimenter les données astronomiques dans une structure d'octree[8], puis d'insérer cet octree dans la base de données. Chaque document représente un nœud ou une feuille de l'octree. Cela à l'avantage de permettre l'exécution de requêtes très optimisées, et donc très rapides.

— **L'utilisation de système de fichiers** : solution ressemblant à la précédente. De la même façon que cette dernière, un octree représentant les données astronomiques est créé. Ces données sont insérées dans le *File System*[9] du serveur : chaque nœud est un répertoire comportant ses fils et chaque feuille est un fichier comportant des points. Le manque de souplesse et de rapidité de cette méthode m'ont décidé à la mettre de côté. Elle fut cependant utilisée pour résoudre un problème génant[10].

2.2 Le côté serveur

Comme expliqué précédemment, j'ai choisi de développer le serveur à l'aide de node.js. Les données astronomiques ont été stockées dans une base de données mongoDB, base de données noSQL orientée documents. Les données sont manipulées sous la forme d'objets au format BSON (du JSON

7. Plus d'informations : `http://nanocubes.net/`.
8. cf. partie 2.2.1.
9. Système de fichiers. Méthode de stockage des informations et d'organisation de fichiers, localisés par des chemins d'accès.
10. cf. partie 3.1.1.

binaire) sans schéma prédéterminé.

2.2.1 Création de l'octree

Pour pouvoir stocker les données dans la base, il a fallu les partitionner en fonction de leurs coordonnées spatiales (x, y et z). De cette façon, il est possible de travailler sur une sous-partie des données sans avoir à effectuer de requêtes sur l'ensemble de celles-ci.

Une structure d'octree a été utilisée pour partionner les données. C'est une structure de données de type arbre dans laquelle chaque nœud peut compter jusqu'à huit enfants. Elle permet de subdiviser un espace tridimensionnel en le subdivisant récursivement.

Voici la structure de données que j'ai utilisée pour la création de l'octree (javascript) :

```
1       var Octree = function(depth, box, id, limit) {
2           this.depth = depth;
3           this.box = box;
4           this.id = id;
5           this.limit = limit;
6           this.children = [];
7           this.isLeaf = true;
8           this.count = 0;
9           this.points = [];
10      };
```

— **depth** : La profondeur du nœud ou de la feuille. La racine est de profondeur 0.
— **box** : la boite englobante des données du nœud. Celle de la racine est le cube de côtés de longueur 1.
— **id** : chaîne de caractères représentant l'identifiant du noeud[11].
— **limit** : le nombre maximum de points dans une feuille.
— **children** : tableau vide si l'objet est une feuille, contenant 8 fils si c'est un nœud.

11. cf. partie 2.2.3

— **isLeaf** : booléen. *True* si c'est une feuille, *false* sinon.
— **count** : le nombre de points contenu dans la feuille. 0 Si l'objet est un noeud.
— **points** : tableau de points. Vide si l'objet est un noeud.

Et l'algorithme utilisé pour la création d'une partie d'octree[12], en pseudo-code :

Algorithm 1 Création d'un octree

```
1: function CREATEOCTREE(listPoints, box)
2:     tabBoxes ← GETSUBBOX(box)            ▷ Retourne les 8 sous-boxes
3:     for i ← 0, nbPoints do
4:         point ← listPoints[i]
5:         if CONTAINS(tabBoxes[0], point) then
6:             ADD(tabBoxes[0], point)
7:         else if CONTAINS(tabBoxes[1], point) then
8:             ADD(tabBoxes[1], point)
9:                     ⋮
10:        else CONTAINS(tabBoxes[7], point)
11:            ADD(tabBoxes[7], point)
12:        end if
13:        i ← i + 1
14:    end for
15:    return tabBoxes
16: end function
```

Une feuille de l'octree est divisée en 8 nouveaux fils si son total de points est supérieur à une valeur fixée arbitrairement, variable globale dans le code de l'application.

Une fois l'octree entièrement créé, il est inséré dans la base de données.

2.2.2 Les données dégradées

Comme les données seront beaucoup trop importantes pour être toutes affichées lorsque la caméra se situe *hors du cube de visualisation*, il est nécessaire de générer des données dégradées représentatives des données réelles.

12. Version simplifiée de la fonction qui ne renvoie qu'un tableau de listes de points. Ce tableau représente les fils du noeud courant.

Une première idée fut le calcul du barycentre de chaque feuille de l'octree. Cette méthode posait un souci lors de l'affichage des données dégradées. Comme les points sont répartis de façon plutôt homogène dans les feuilles, le barycentre était souvent centré dans la feuille. Comme les points étaient très alignés, des lignes étaient très visibles lors de l'affichage [13]. J'ai donc abandonné cette idée.

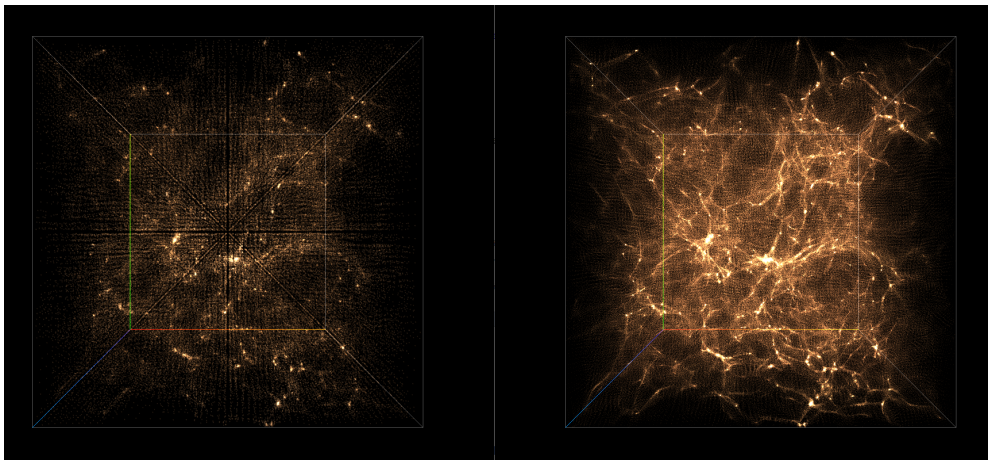

FIGURE 3 – 2 millions de points : données dégradées à gauche, données non dégradées à droite

J'ai choisi de garder cette deuxième méthode : à la place de calculer les barycentres de chaque feuille, il suffisait de choisir aléatoirement un point dans chaque feuille. Cela permet de résoudre le problème des lignes, tout en représentant les données de façon correcte.

Un poids est associé à chaque point des données dégradées. Il représente le nombre d'étoiles associé à ce point. Lors de l'affichage des données dégradées, il est possible de modifier leur aspect (couleur, taille) en fonction du poids de chacun des points, pour être encore plus représentatif des données réelles.

2.2.3 Système d'indexation de l'octree

Chaque document (nœud ou feuille) inséré dans la base de données doit avoir un identifiant unique, qui permet de le reconnaître. J'ai choisi d'utili-

13. cf. Figure 3

ser *le code de Morton*[14], un système permettant d'identifier des octants en fonction de leurs positions par rapport à leurs parents et de la profondeur qu'ils occupent dans l'arbre.

Le code fonctionne de la façon suivante : chaque fils d'un octant est numéroté de 0 à 7 au format binaire, soit de 000 à 111. Les trois chiffres binaires correspondent dans l'ordre aux axes z, y et x. Un chiffre égal à 0 signifie que le cube est du côté de l'axe. Un chiffre égal à 1 signifie qu'il est à l'opposé de l'axe[15].

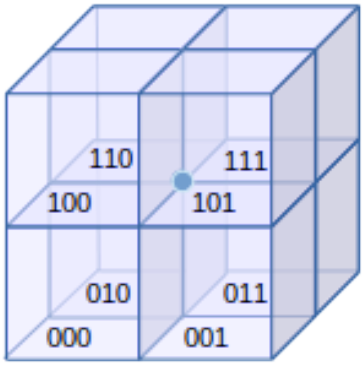

FIGURE 4 – Numérotation des octants d'un octree

Par exemple, l'octant numéroté 000 sera placé au niveau de l'origine ($z = 0$, $y = 0$, $z = 0$) tandis que l'octant numéroté 111 sera placé dans le coin opposé ($z = 1$, $y = 1$, $x = 1$).

La profondeur est gérée de la manière suivante :

— La racine, qui a une profondeur de 0, a un identifiant vide - "".

— Pour chaque nouveau nœud ou feuille, il suffit de concaténer sa position (représentée par un nombre de 3 chiffres binaires) avec l'identifiant de son père.

Par exemple, les 8 fils du nœud d'identifiant 000111 auront les identifiants suivants : 000111000, 000111001, 000111010, 000111011, 000111100, 000111101, 000111110, 000111111.

L'identifiant généré par la méthode de Morton est donc une sorte de *che-*

14. Plus d'information : `http://pierre-benet.developpez.com/tutoriels/algorithme-3d/octree-morton/`.

15. cf. figure 4.

min d'accès, unique pour chaque octant et qui prend en compte la position de celui-ci ainsi que sa pronfondeur. Il est ainsi possible avec l'identifiant d'un octant de retrouver :

— **Sa profondeur** - Le nombre de chiffres composant son identifiant divisé par 3.

— **Ses parents** - Suppression des 3 derniers chiffres de l'identifiant à chaque fois que l'on souhaite accéder au parent direct.

2.2.4 Requêtes sur la base de données

J'ai développé de nombreuse requêtes différentes durant mon stage, pour parer à de nombreuses éventualités. Seules quelques-unes de ces requêtes sont finalement utilisées :

La première requête est *getDegradedData*. Comme son nom l'indique, elle retourne les points des données dégradées. Elle est utilisé une seule fois dans l'application, au moment où l'on se connecte au serveur.

La seconde requête est celle qui est le plus utilisée, et qui forme le service majeur du serveur : *getPointsFromPattern*. Cette requête renvoie la liste de tous les points contenus dans un nœud et dans ses fils.

En raison de la façon dont sont stockés les noeuds dans mongoDB (au même *niveau*, sans référence les uns aux autres), cette requête est très rapide. Il n'est pas nécessaire de parcourir les documents de manière récursive comme ce serait le cas si le traitement était effectué sur un octree. Grâce à l'identifiant des nœuds, une expression régulière suffit.

Par exemple, pour retrouver tous les points contenus dans l'octant d'identifiant 011001, il suffit de rechercher les points des octants ayant un identifiant commançant par 011001. Ce sont les fils de l'octant 011001. C'est ce qu'effectue la requête suivante :

```
db.collection.find({id : /^011001/}, {points : true})
```

Un index sur les identifiants des documents est mis à jour à chaque fois qu'un document est inséré. De cette façon, les requêtes sur les identifiants sont très efficaces.

2.3 Le côté client

Le côté client est formé par l'application, qui est délivré par le serveur.

2.3.1 Les requêtes ajax

Des requêtes Ajax [16] sont effectuées par l'application pour le chargement des points. Il n'est donc pas nécessaire de recharger la page chaque fois qu'on souhaite charger de nouvelles données.

Une petite api a été devéloppée côté serveur, permettant l'accès aux différentes ressources. La bibliothèque qwest.js [17] a été utilisée pour effectuer les requête Ajax très facilement.

Voici les requêtes les plus utilisées :

```
get : /api/octrees/pattern/ + id
```

Renvoie les points de l'octant dont l'identifiant est *id*, ainsi que les points de ses fils.

```
get : /api/octrees/degradedData
```

Renvoie toutes les données dégradées.

2.3.2 La récupération des données

Lorsque l'utilisateur *navigue* à l'intérieur du cube de visualisation, les données sont chargées automtiquement. Un test est effectué à chaque fois que la caméra se déplace. Si elle change d'octant, c'est à dire si elle quitte l'octant dans lequel elle se trouve pour entrer dans un nouveau, une requête Ajax est effectuée pour charger des données.

16. Asynchronous JAvascript and Xml. Méthode de chargement de données en arrière plan, sans avoir à charger de page web.
17. Voir `https://github.com/pyrsmk/qwest`

Note : La profondeur utilisée pour les changements d'octants est définie en variable globale dans le code de l'application. Elle doit être ajustée au cas par cas. L'application sera grandement ralentie si trop de données sont téléchargées.

Dans un premier temps, le chargement des données n'était effectué que pour l'octant courant. Cela permet de visualiser correctement les données mais n'est pas satisfaisant lorsque la caméra se rapproche d'un bord de l'octant : aucun point n'est affiché au-delà.

Il fallait donc choisir une méthode pour charger les données hors de l'octant courant. J'ai choisi de charger les données des octants voisins de l'octant courant, en plus de ce dernier. Nous avions donc un total de 29 octant à charger (3 *tranches* de 9 octants).

Pour déterminer la liste des octants à charger : on ajoute ou l'on retire tout d'abord la longueur du côté d'un octant au centre de l'octant central, dans chacune des directions. Cela permet de déterminer une liste de 26 points (un pour chaque octant voisin), chacun contenu dans un des octants voisins.

On cherche ensuite le code de l'octant contenant chacun de ces 26 points. J'ai utilisé l'algorithme suivant :

Algorithm 2 Recherche du code de l'octant contenant un point

```
1:  function getCodePoint(point)
2:      curBox ← getRootBox        ▷ Retourne la boîte englobante de la racine.
3:      curDepth ← 0
4:      code ← ""
5:      while curDepth < octantDepth do
6:          res ← subBox(curBox, point)   ▷ Retourne un objet contenant la boîte englobante du niveau inférieur contenant le point et le code de l'octant.
7:          code ← concat(code, res.code)
8:          curBox ← res.box
9:          curDepth ← curDepth + 1
10:     end while
11:     return code
12: end function
```

Une fois que l'on a les codes de tous les octants, nous pouvons effectuer

les requêtes vers le serveur. Il faut cependant vérifier pour chaque octant que nous n'avons pas déjà téléchargé ses données. Pour cela, nous gardons en mémoire un tableau contenant les codes des octrees déjà téléchargés. Il suffit donc de vérifier si ce tableau contient déjà le code de l'octant, et, dans le cas échéant, faire une requête vers le serveur.

Une fois les points récupérés depuis le serveur, une fonction est chargée de les formater en un tableau contenant différents objets clé-valeur :

- **index** : L'index des points. Utilisé lors du chargement depuis des fichiers locaux. Depuis le serveur, l'index de chaque point correspond à sa case dans le tableau.
- **position** : tableau contenant toutes les positions des points de la façon suivante : p1.x, p1.y, p1.z, p2.x, p2.y, p3.z, etc.
- **info** : contient des paramètres associés aux données astronomiques (âge, énergie potentielle, etc.).
- **metaType** : chaîne de caractères identifiant les données.
- **bornes** : boîte englobante de tous les points chargés. Contient les valeurs maximales et minimales des coordonnées x, y et z.

Ce tableau est envoyé à l'application, qui se charge de créer un octree avec les points et de l'afficher.

3 Les difficultés rencontrées

3.1 Les données finales

Toutes les opérations décrites dans la partie 2 fonctionnaient avec les données de test que j'ai utilisées : les données *part*, contenant 2 millions de points. J'ai eu de gros soucis lorsque j'ai voulu passer aux données finales, d'un volume grandement supérieur : presque 1 To.

3.1.1 La création de l'octree

Pour les données de test, l'insertion dans la base de données se faisait de la manière suivante :

1. Lecture des fichiers.
2. Création de l'octree et répartition simultanée des points.
3. Insertion de l'octree dans la base de données.

Cette méthode n'est pas utilisable avec de gros volumes de données, puisqu'il est nécessaire de charger tous les fichiers dans la RAM [18] de la machine. Elle n'était donc pas envisageable dans notre cas.

Il a fallu trouver une autre solution pour la création de l'octree. De plus, il n'est pas possible de créer un octree pour chacun des fichiers de données et de les insérer un par un dans la base de données. Les fichiers ne correspondent pas à des sous-octrees. Il est donc obligatoire de créer entièrement l'octree avant de pouvoir commencer l'insertion dans la base de données.

Après réflexion, deux solutions s'offraient à moi :

— **L'utilisation du File System** : À la place de créer entièrement l'octree dans la RAM de la machine, celui-ci est créé dans le File System de la machine. Les répertoires correspondent aux nœuds de l'octree tandis que les fichiers correspondent aux feuilles, et contiennent les points. J'ai choisi cette méthode.

18. Random Access Memory : Mémoire informatique dans laquelle un ordinateur place les données lors de leur traitement.

— **La création de sous-octrees** : La méthode est la suivante. Pour chaque sous-octree que l'on souhaite créer :

1. Lecture des fichiers.
2. Création du sous-octree avec les points qu'il contient.
3. Ajout du sous-octree à la base de données.

Une fois tous les sous-octrees créés et ajoutés à la base de données, il faut encore créer et insérer les parents de ces octrees, jusqu'à atteindre la racine. J'ai choisi d'ignorer cette méthode pour deux raisons :

— Elle nécessite un parcours de tous les fichiers pour chaque sous-octree que l'on souhaite créer.
— Cette méthode utilise encore beaucoup la RAM de la machine. Pour être certain qu'elle fonctionne, il faudrait créer beaucoup de sous-octrees, assez profondément dans l'arbre (au moins au 3^{e}niveau, ce qui correspond à $8^3 = 512$ sous-octrees).

La première version de l'utilisation du File System

Structure du File System : Comme expliqué précédemment, les nœuds de l'octree correspondent aux répertoires et les feuilles sont des fichiers qui contiennent les points. Il y a un point par ligne et les coordonnées x, y et z sont séparés par une virgule, de la même façon que dans les fichiers CSV[19]*. Un header indique le nombre de points contenu dans chaque feuille.*

J'ai tout d'abord développé une version *naïve* de la création de l'octree dans le File System. Au fur et à mesure que les données étaient lues, je rajoutais les points un par un aux différents fichiers, en parcourant à chaque fois l'octree depuis la racine. Lorsqu'un fichier contenait trop de points, il était supprimé et un nouveau répertoire était créé, ainsi que 8 fichiers à l'intérieur de ce dernier. Les points du fichier supprimé étaient alors répartis dans les 8 nouveaux fichiers créés, qui correspondaient aux nouvelles feuilles.

Bien que fonctionnelle, cette méthode ne convenait pas une utilisation réelle car bien trop lente. Les performances des opérations d'entrées et sor-

19. Comma-separated values : format informatique représentant des données tabulaires sous forme de valeurs séparées par des virgules.

ties vers des fichiers sont trop peu performantes pour écrire les points un par un.

D'après une grossière estimation, l'insertion des données avec cette méthode dans la base de données prendrait environ 4 mois.

La version optimisée de l'utilisation du File System
J'ai developpé une nouvelle version de la création de l'octree dans le File System, beaucoup plus optimisé que la précédente. J'ai gardé la même structure de fichiers.

L'algorithme est plutôt simple mais difficile à expliquer correctement : je crée un sous-octree (dans la RAM, ancienne méthode) pour chaque fichier de données, que j'insère dans le File System. Il y a deux cas de figure :

- S'il n'y a pas de conflit avec des fichiers déjà présents dans le File System, tout va bien. L'insertion se passe comme prévu.
- S'il y a des conflits, le sous-octree que j'ai créé et le sous-octree avec lequel il est en conflit sont *fusionnés* pour créer un nouveau sous-octree qui est inséré dans le File System.

Cette méthode fonctionne beaucoup mieux que la précédente. Les points sont insérés en masse dans le File System, ce qui diminue grandement le nombre d'entrées et sorties vers des fichiers.

Les données dégradées et insertion dans la base de données
Je n'ai malheureusement pas eu le temps de travailler sur la dégradation de l'octree du File System et sur l'insertion de celui-ci dans la base de données. Jérôme Desroziers, stagiaire ayant travaillé sur l'application et étant en CDD les mois de juillet et août à l'observatoire s'en est occupé.

3.1.2 L'affichage des points

L'affichage des points m'a aussi posé quelques soucis. Il a fallu déterminer avec précision quel niveau d'octant considérer lorsque l'on se déplace dans le cube de visualisation, les données finales étant très denses. Finalement, les octants d'une profondeur de 7 sont chargés.

3.2 Les performances des requêtes

Pour effectuer les manipulations de la base de données avec node.js, j'ai utilisé Mongoose, un module basé sur des schémas d'objets permettant la validation et la manipulation de documents.

Au cours de mon stage, je me suis rendu compte que les performances des requêtes qui renvoyaient beaucoup de données étaient vraiment mauvaises par rapport aux résultats attendus. Je ne savais pas à ce moment là que Mongoose *wrap* les documents renvoyés par la base de données avec des *getter*, *setter* et autres fonctions utilitaires. Cela modifie très peu les performances quand il n'y pas beaucoup de documents, mais elles chutent drastiquement dès qu'il y a plusieurs milliers de documents. Ce comportement augmente aussi la quantité de RAM utilisée par l'application.

J'ai découvert la méthode *lean* au gré de mes recherches pour optimiser les requêtes. Ajoutée à une requête, elle indique à Mongoose de ne pas *wrapper* les données renvoyées par la base. Cela ôte la souplesse des objets manipulés par Mongoose mais améliore énormément les performances.

Pour une requête sur les 2 millions de points de la base de données :

— `octreeSchema.find().select('points').exec()` : Une erreur d'allocation de mémoire se produit après 30 secondes.

— `octreeSchema.find().select('points').lean().exec()` : les points sont retournés en 5 secondes.

Dans l'optique d'améliorer encore les performances, j'ai transformé certaines requêtes en aggrégation. Ce changement consiste à aggréger tous les documents renvoyés par une requête en un seul document. Cela permet d'améliorer les performances sur certaines requêtes, mais les réduit sur d'autres. Il a fallu que je fasse des tests pour être sûr de ne pas réduire les performances en transformant les requêtes. Pour le renvoit de l'id des octants des données de 2 millions de points, le gain de performances est de 80 millisecondes (260 millisecondes contre 380 millisecondes avant le changement).

Une dernière optimisation consiste à compresser les données côté serveur avant de les envoyer vers le client. Ces données sont ensuite décompressées par le client, avant d'être utilisées.

3.3 Les autres problèmes

J'ai aussi eu quelques soucis de compréhension de l'application. Le développement de celle-ci était particulièrement avancé lorsque j'ai commencé mon stage. Les classes étaient imposantes, tant en nombre de lignes, qu'en nombre d'objets ou d'imbrications avec d'autres classes. J'ai mis un long moment avant de saisir le fonctionnement de l'application.

J'ai heureusement pu poser des questions à Jérôme, qui était encore en stage lorsque j'ai commencé le mien et était en CDD à l'observatoire par la suite. Après avoir passé son stage plongé dans le code de l'application, il a su répondre à la plupart de mes questions assez facilement.

Finalement, le dernier problème que j'ai rencontré est la limitation de la RAM de node.js. Par défaut, le processus ne peut accéder qu'à environ 1,5 go de RAM. Cela ne suffit pas toujours pour les traitements que j'effectue côté serveur, même lorsque tous les fichiers ne sont pas chargés en mémoire [20]. Il a été assez facile de résoudre ce souci : il est possible d'augmenter la mémoire utilisable par node.js avec la commande `--max-old-space-size=newSize`.

20. (cf. partie 3.1.1).

Conclusion

Ce stage a été extrêmement intéressant et instructif. Il m'a permis d'améliorer mes connaissances dans des technologies actuelles, telles Node.js et MongoDB. Une grande partie de ce stage reposait sur ma capacité de réflexion :

— Avant de commencer le développement, dans le choix des technologies.

— Pendant le développement, lors de la résolution des nombreux problèmes que j'ai rencontrés.

Certaines de ces difficultés étaient complexes à traiter, mais toutes ont été résolues.

Je ne suis pas certain d'avoir toujours effectué les meilleurs choix possibles durant mon stage. Je pense cependant avoir fait au mieux avec les connaissances que je possédais à ce moment-là. Il est probablement possible d'optimiser certaines parties du code et d'en produire un plus propre et factorisé.

Mon tuteur m'a laissé une grande autonomie durant le stage. J'ai pu organiser mon travail librement et prendre mes propres décisions. Des réunions avec des intervenants du projet ont cependant été organisées ainsi que des discussions ponctuelles pour suivre l'avancée du stage.

Je suis ravi de mon stage et j'espère que mon travail sera utilisé. Dans tous les cas, l'expérience acquise tout au long du projet sera très bénéfique à la poursuite de mes études.

Références

[1] Site de l'Observatoire Astronomique de Strasbourg : `https://astro.unistra.fr/`

[2] StackOverflow, site de questions et réponses pour développeurs : `http://stackoverflow.com/`

[3] Documentation de Node.js : `https://nodejs.org/en/docs/`

[4] Site officiel de MongoDB : `https://www.mongodb.org/`

[5] Site officiel de three.js : `http://threejs.org/`

[6] Documentation sur les nanocubes : `http://nanocubes.net/`

[7] Documentation sur le code de Morton : `http://pierre-benet.developpez.com/tutoriels/algorithme-3d/octree-morton/`

[8] Wikipédia, encyclopédie universelle collaborative : `https://fr.wikipedia.org/`

Lexique

AJAX - Asynchronous JAvascript and XML : Méthode de chargement de données en arrière plan, sans avoir à charger de page web.

API - Application Programming Interface : ensemble normalisé de classes, méthodes et fonctions qui sert de façade par laquelle un logiciel offre des services.

Callback - fonction de rappel : fonction passée en argument à une autre fonction, qui peut alors l'utiliser.

Compilation : transformation d'un code source écrit dans un langage de programmation dans un autre langage informatique.

CSS : feuilles de style en cascade. Langage qui décrit la présentation des documents HTML.

CSV - Comma Separated Values : format informatique représentant des données tabulaires sous forme de valeurs séparées par des virgules.

DOM - Le Document Object Model : interface permettant à des programmes informatiques et à des scripts d'accéder ou de mettre à jour le contenu, la structure ou le style de documents HTML et XML.

Expression régulière : chaîne de caractères décrivant un ensemble de chaînes de caractères possibles selon une syntaxe précise.

File System : méthode de stockage des informations et d'organisation de fichiers, localisés par des chemins d'accès.

Git : logiciel libre de gestion de versions décentralisé.

GitLab :

HTML - HyperText Markup Language : format de données conçu pour représenter les pages web.

IDE - Environnement de Développement : ensemble d'outils pour augmenter la productivité des programmeurs. Contient toujours un éditeur de texte et peut contenir des outils de création d'interface graphique, de réalisation automatique de tests, de contrôle de versions, etc.

MongoDB : système de gestion de base de données orienté documents, répartissable sur un nombre quelconque d'ordinateurs et ne nécessitant pas de schéma prédéfini des données.

Mongoose :

Node.js : plateforme logicielle libre et événementielle en JavaScript orientée vers les applications réseau qui doivent pouvoir monter en charge.

NoSQL - Not only SQL : catégorie de systèmes de gestion de base de données qui n'est plus fondée sur l'architecture classique des bases relationnelles.

Open-space : espace de travail où les bureaux ne sont pas séparés par des cloisons.

Promise : objet représentant la valeur ou l'exception retournée par une fonction asynchrone.

RAM - Random Access Memory : mémoire informatique dans laqeulle un ordinateur place les données lors de leur traitement.

RoboMongo : outil d'administration graphique pour *MongoDB*.

Téraoctet : unité multiple de l'octet. Correspond à 10^{12} octets.