Rapport de stage

Réalité virtuelle dans le cadre de services et de données astronomiques

Romain Houpin

Année 2013-2014

Stage de 2e année réalisé au Centre de Données astronomiques de l'observatoire de Strasbourg

Maître de stage : *André Schaaff*
Encadrant universitaire : *Hervé Panetto*

Déclaration sur l'honneur de non-plagiat

Je soussigné(e),

Nom, prénom : _____

Elève-ingénieur(e) régulièrement inscrit(e) en 2e année à TELECOM Nancy

N° de carte d'étudiant(e) : _____

Année universitaire : 20__ - 20__

Auteur(e) du document, mémoire, rapport ou code informatique intitulé :

_____

_____

Par la présente, je déclare m'être informé(e) sur les différentes formes de plagiat existantes et sur les techniques et normes de citation et référence.

Je déclare en outre que le travail rendu est un travail original, issu de ma réflexion personnelle, et qu'il a été rédigé entièrement par mes soins. J'affirme n'avoir ni contrefait, ni falsifié, ni copié tout ou partie de l'œuvre d'autrui, en particulier texte ou code informatique, dans le but de me l'accaparer.

Je certifie donc que toutes formulations, idées, recherches, raisonnements, analyses, programmes, schémas ou autre créations, figurant dans le document et empruntés à un tiers, sont clairement signalés comme tels, selon les usages en vigueur.

Je suis conscient(e) que le fait de ne pas citer une source ou de ne pas la citer clairement et complètement est constitutif de plagiat, que le plagiat est considéré comme une faute grave au sein de l'Université, et qu'en cas de manquement aux règles en la matière, j'encourrais des poursuites non seulement devant la commission de discipline de l'établissement mais également devant les tribunaux de la République Française.

Fait à _____, le ___/ ___/ 20___

Signature :

Résumé *(250 mots max)*

Le stage consistait en la réalisation d'un prototype de visualisation 3D. Ce prototype devait afficher des nuages de points et des cubes de données à l'aide de la bibliothèque OpengGL. De plus, afin d'améliorer le rendu, l'Oculus Rift devait être intégré au projet afin de mettre à profit la réalité virtuelle aux résultats de simulations affichés. Ce rapport reprend le développement de ce prototype en insistant notamment sur les logiciels et matériels utilisés et sur les différentes méthodes mises en place.

Mots-clés : CDS, *OpenGL, Oculus Rift*

Abstract *(250 words max)*

Keywords: CDS, *OpenGL, Oculus Rift*

Sommaire

Remerciements

Je tiens particulièrement à remercié pour ce stage :

- André Schaaff, mon maître de stage,
- L'équipe Skybot 3D : Jérôme Berthier et Jonathan Normand,
- Brad Davis, auteur du livre «Oculus Rift in Action», pour l'aide qu'il m'a apporté sur les forums d'Oculus Rift,
- Nicolas Deparis et Philippe Gaultier, stagiaires à l'Observatoire sur des sujets de rendu graphique 3D,
- Sébastien Derriere, astronome adjoint au CDS,
- Pierre Ocvirk, astronome adjoint au CDS, qui nous a fourni les données de simulation et nous a apporté son expertise dans ce domaine
- Hervé Panetto, mon encadrant universitaire,
- Et toutes les autres personnes travaillant à l'observatoire pour leur accueil et leur sympathie

Introduction

Dans le cadre de cette 2ème année de TELECOM Nancy, je devais effectuer un stage de technicien informatique afin de mettre en pratique les connaissances acquises durant la formation et en apprendre de nouvelles. Passionné depuis toujours par l'astronomie, je souhaitait mettre à profit mes services à ce domaine. Pour cette raison, depuis 2012, je rejoignais l'observatoire astronomique de Strasbourg et plus particulièrement le Centre de Données astronomiques de Strasbourg (CDS). Le CDS correspondait parfaitement à ce qui pouvait être réalisé en stage de deuxième année car il s'agit d'une entité de recherche et d'enseignement et c'est pour cela que j'ai décidé de recontacter André Schaaff afin de savoir s'il était possible de les rejoindre à nouveau pour une troisième année. La réponse ayant été positive, c'est donc avec joie que j'ai rejoint pour ce stage le CDS.

Dans ce rapport, je présenterai tout d'abord l'observatoire astronomique de Strasbourg, en présentant notamment les équipes de recherche et le CDS. Ce dernier étant le service qui m'a accueilli, nous détaillerons plus précisément son fonctionnement et ses services. Ensuite, dans une deuxième partie, je reviendrai sur le déroulement du stage en lui même en parlant du contexte de celui-ci, du cahier des charges, de l'environnement de travail, du déroulement en lui-même, des difficultés rencontrées, des solutions trouvées et des perspectives du projet.

Commençons donc dès à présent par la présentation de l'entreprise.

Chapitre 1 :
Présentation de l'entreprise

1.1) L'Observatoire Astronomique de Strasbourg

1.1.1) Présentation générale

L'Observatoire astronomique de Strasbourg est un Observatoire des Sciences de l'Univers (OSU). Il est une Unité Mixte de Recherche (UMR 7550) entre l'Université de Strasbourg (l'observatoire est d'ailleurs situé au cœur de l'université) et le CNRS (Centre National de la Recherche Scientifique).

L'observatoire a été construit en 1881 et possède en son sein la 3ème plus grande lunette astronomique de France de par son diamètre. L'observatoire accueil également le planétarium qui depuis 2008 n'est plus sous sa responsabilité. On peut trouver également à l'observatoire des ouvrages anciens et un patrimoine d'anciens instruments

Pour réaliser ses missions, l'observatoire possède deux équipes de recherche que nous présenterons plus tard et deux Services d'Observation de l'Institut National des Sciences de l'Univers (INSU), le Survey Science Centre d'XMM-Newton (SSC-XMM) et le Centre de Données astronomiques de Strasbourg (CDS) dans lequel j'ai travaillé.

L’observatoire est au cœur du dispositif national mis en œuvre par l’INSU et doit assurer plusieurs missions comme l’acquisition de données d’observation, l’élaboration d’outils théoriques ainsi que le développement et l’exploitation de moyens appropriés. En parallèle de ses missions et pour remplir ses conditions d’observatoire, il doit fournir des services liés à son activité, assurer la formation des étudiants et des personnels de recherche, assurer la diffusion des connaissances et il est chargé d’activités de coopération internationale.

1.1.2) Les équipes de recherche

L’observatoire possède principalement deux équipes de recherche en plus du CDS (qui lui cumule le service d’observation avec la recherche) Nous allons donc voir dans cette partie les deux équipes de recherche à part entière qui sont « l’équipe Hautes Energies » et « l’équipe Galaxies ».

L'équipe Galaxie :

L’équipe « Galaxies » étudie de manière précise comment se passe la naissance des galaxies et comment elles évoluent dans le temps avec leurs populations stellaires. Pour mener à bien cela, elle utilise les populations stellaires connues, la dynamique des étoiles et la matière noire (catégorie de matière hypothétique jusqu'à présent non détectée). L'équipe est également impliquée dans le relevé RAVE : un grand relevé cinématique.

La population stellaire des galaxies est analysée, après avoir été observée, de manière à pouvoir étudier la dynamique gravitationnelle qui régit les mouvements du gaz et des étoiles. Ces études permettent ensuite de lier l'histoire avec l'information ce qui permet de connaître les moments clés de la vie des galaxies.

L'équipe possède également un autre domaine de recherche qui se porte sur les amas

stellaires (concentration locale d'étoiles d'origine commune et liées entre elles par la gravitation). Ce domaine de recherche consiste entre autre à construire numériquement des amas en fusionnant toutes les connaissances sur la physique des étoiles et sur l'évolution gravitationnelle des amas. Cela est très utile pour comprendre la formation des galaxies.

L'équipe Hautes Énergies :

L'équipe « Hautes Énergies » s'intéresse entre autre aux sources galactiques et extragalactiques émettrices en rayons X mais également aux objets compacts (étoiles à neutrons, naines blanches, etc.) et aux noyaux actifs de galaxies. L'équipe participe très activement depuis 1996 au SCC (Survey Science Center) qui est un consortium international de laboratoires sélectionnés par l'ESA et labellisés par l'INSU comme Service d'Observation. Le SCC-XMM est en charge de fournir des catalogues de manière la plus complète possible de sources X.

L'équipe s'occupe des problèmes d'astrophysiques de population comme la caractérisation de la densité ou encore les échelles de hauteurs des sources X ainsi de l'évolution des sources X qui peuvent provenir par exemple de couronnes stellaires. Enfin, elle s'occupe également de la physique des disques d'accrétion et des jets.

1.2) Le CDS

1.2.1) Présentation :

Le Centre de Données astronomiques de Strasbourg (CDS) a été créé en 1972. La mission principale du CDS est de gérer un ensemble de données astronomiques qu'il faut recueillir, unifier et homogénéiser, conserver et enfin distribuer de manière à ce que la communauté scientifique internationale puisse en bénéficier.

L'équipe du CDS est composée d'une trentaine de personnes environ réparties de manière équivalente entre trois métiers : astronomes, documentalistes et bien évidemment informaticiens.

Le CDS possède ses propres infrastructures, il possède de nombreux postes de travail (sous Ubuntu dans la plupart des cas) ainsi que d'une salle de serveurs, …

Il est également à noter que le CDS est une référence mondiale dans le domaine astronomique et que depuis fin 2009, son utilisation est en augmentation continue. Grâce à sa

notoriété mais surtout sa très grande base de données pour le référencement d'objets astronomique, le CDS fait partie des TGIR (les Très Grandes Infrastructures de Recherches).

1.2.2) Les services du CDS :

Le CDS possède de nombreux services, mais, de manière générale, trois grands services représentent le CDS : Simbad, VizieR et Aladin. Ils ont chacun leur propre utilité mais sont complémentaires entre eux.

Simbad :

Simbad est une base de données regroupant des informations sur un nombre remarquable d'objets astronomiques (un objet astronomique peut être une galaxie, une étoile, un trou noir, un nuage de gaz, etc.). Cette base de données fut créée il y a près de 40 ans et contient aujourd'hui sept millions d'objets. Pour chaque objet, Simbad est capable de fournir de nombreuses caractéristiques physiques comme la position de l'objet, sa magnitude, sa parallaxe, etc. Ces informations proviennent toutes de catalogues recensant et décrivant les objets astronomiques.

De part son efficacité et son nombre important d'objets référencés de manière très complète, Simbad est devenue la base de données de référence pour la nomenclature et la bibliographie des objets astronomiques. Aujourd'hui, Simbad traite des centaines de milliers de requêtes par jour venant de très nombreux pays et a su devenir une référence mondiale sans équivalent.

VizieR :

Nous avons vu plus tôt que les objets référencés dans Simbad provenaient de catalogues astronomiques. Ces catalogues sont très nombreux (plus de 10000) et sont en général constitués de nombreuses tables (ces tables regroupent des informations sur de nombreux objets). Toutes ces tables et catalogues ont été stockés par le CDS (plus précisément par les documentalistes) dans ce qui s'appelle aujourd'hui VizieR. VizieR permet donc d'accéder aux objets astronomiques via des catalogues de

manière homogène alors que les données des catalogues sont à la base hétérogène. VizieR est aussi un outil très puissant pour la recherche par critères.

Aladin :

Aladin est le dernier né parmi les 3 services (1997) et est extrêmement puissant. Il permet de visualiser un atlas du ciel de manière très interactive. De par sa nature, ce service est encore en développement par plusieurs personnes du CDS

Aladin est en réalité une entité mixte. En effet, on trouve la base de données Aladin qui est une base contenant une quantité très importante d'images, et, on trouve Aladin, un logiciel (disponible également en applet) écrit en Java. Ce dernier s'appuie sur la base de données Aladin ainsi que d'autres pour former un atlas très interactif.

1.3) L'observatoire Virtuel et l'IVOA

1.3.1) Présentation

Un observatoire virtuel est un ensemble de données distribuées et accessibles de partout avec des logiciels puissant ayant un fort potentiel en calculs contrairement à un vrai observatoire qui dispose, lui, de ses instruments (télescopes, radio-télescopes, etc.) qui permettent de réaliser des observations ou des relevés du ciel a un instant donné et dans une direction donnée (suivant la localisation de l'instrument). L'accès à de multiples sources de données nécessite la définition de standards d'accès à ces données. C'est pour cette raison qu'est né l'IVOA.

L'IVOA (International Virtual Observatory Alliance) est une organisation internationale ayant pour objectif de définir des standards pour faciliter les échanges internationaux et les collaborations afin de fournir des services utilisables par toute la communauté mondiale sans adaptation nécessaire. Cette organisation crée donc un contexte d'Interopérabilité homogénéisant l'accès aux données et l'utilisation des outils et des services an général afin de rendre leurs données compatibles.

L'IVOA fut créée en juin 2002 et compte désormais de nombreux membres comme euro-VO, Aus-VO, China-VO, VO France, etc. La volonté de standardiser les données est une opération complexe mais nécessaire pour pouvoir avancer de nos jours en Astronomie. De

nombreux projets sont en cours et l'ESA ainsi que l'ESO participent de manière active à cette internationalisation et standardisation des échanges. L'IVOA compte désormais plus de 20 pays ou organisations membres.

Une fois que les standards sont validés par l'IVOA, il tient à la responsabilité des différents acteurs de les respecter que ce soit au niveau de ses outils ou de ses données.

1.3.1) Le CDS au sein de l'IVOA

Le CDS participe à l'Action Concertée Observatoire Virtuel France, autrement dit de l'observatoire virtuel Français et est depuis plus de 40 ans, reconnue comme impliquée dans les collaborations internationales. Les astronomes du monde entier connaissent le CDS et il joue donc un rôle majeur dans le cadre de l'IVOA. Ainsi, de manière très active, le CDS fait partie de la plupart des groupes de travail de l'IVOA (un groupe de travail est constitué de plusieurs partenaires qui discutent des standards à mettre en place).

Chapitre 2 :
Le stage

2.1) Contexte du stage

2.1.1) Ouverture et modernisation

Le CDS a toujours cherché à fournir des services très efficaces et pouvant fonctionner sur tous les supports. De plus, le CDS souhaite également que ses services soient disponibles où que l'on se trouve et a pour objectif d'être toujours ouvert aux nouvelles technologies afin de rendre ses informations disponibles toujours au plus grand nombre. Dans un objectif de recherche et d'éducation, le CDS cherche également à rendre les informations toujours plus visuelles et interactives. Or, ces dernières années, l'informatique a grandement évolué et nous avons vu naître beaucoup de nouveautés. Parmi elles, on retrouve les casques à réalité virtuelle avec notamment leur ambassadeur l'Oculus Rift. Le CDS s'est alors posé la question de ce qu'ils pourraient apportés pour eux.

2.1.2) Origine et objectif du stage

Les casques a réalité augmentées permettant de se projeter dans un environnement 3D suivant les mouvements de notre tête sont perçus par le grand public comme de nouveaux outils pour les jeux-vidéo. Le CDS a cependant eu une vision différente, pourquoi ne pas utiliser ces casques a réalité virtuelle a des fins de recherche et de visualisation de données astronomiques ?

Dans cette optique, un kit de développement de l'Oculus VR (Oculus Rift) a été commandé par M.André Schaaff afin de mieux comprendre le fonctionnement de ce dernier et d'y développer des prototypes. Deux principaux projets ont alors vu le jour :

- Intégrer l'Oculus Rift à une visualisation 3D du système solaire existante (Skybot 3D)
- Développer un programme de simulation 3D d'objets célestes en y intégrant l'Oculus Rift.

L'objectif principal de mon stage allait se porter sur le deuxième projet.

2.2) Cahier des charges

Le projet de développer un programme de simulation 3D d'objets célestes en y intégrant l'Oculus Rift renfermait en réalité deux utilisations différentes :

- Pouvoir visualiser des étoiles et d'y représenter leurs caractéristiques
- Pouvoir visualiser des cubes de données représentant l'ionisation au cours du temps des galaxies.

Cependant, les deux utilisations se rapprochant du même cahier des charges, il fut possible de regrouper ces deux utilisations dans un seul programme. Le cahier des charges de ce dernier était donc le suivant :

- Le programme doit être capable de lire des fichiers en entrées (texte ou binaire) contenant les données d'étoiles (avec position, âge, masse, taille, etc) ou des cubes de données d'ionisation.
- Le programme doit calculer et générer des objets 3D correspondant aux données reçues et permettant leurs visualisations de manière aisée.
- Le programme doit afficher tous les objets dans une scène et adapter automatiquement les échelles en fonction des données de positions notamment.
- Le programme doit pouvoir laisser le choix entre une vue « normale » ou une vue en mode Oculus.

2.3) Environnement de travail, software et hardware

Durant toute la durée du stage, j'ai rejoins les bureaux situés à la bibliothèque de l'observatoire. Cette dernière offrant en effet quatre postes de travail utilisés principalement par des stagiaires ou des CDD.

Devant développer un programme générant des objets 3D et devant intégrer l'Oculus Rift, j'ai été équipé d'un ordinateur spécial possédant entre autre : un processeur Intel Core I5-4570 3,2Ghz possédant 4 cœurs, une carte graphique RADEON HD 8570, 16Gb de RAM et 1Gb de VRAM. Cette machine tournait à la base sous Windows puis j'ai procédé à l'installation de Ubuntu en Dual Boot durant le stage.

2.3.1) OpenGL

Pour réaliser la simulation en 3D, la question du moteur 3D à utiliser s'était vite posée. OpenGL étant parfaitement compatible avec l'Oculus Rift et étant très performant, la décision de l'utiliser comme moteur 3D fut prise rapidement.

OpenGL est un ensemble de fonction qui permet le calcul d'images 2D ou 3D et a été lancé par Silicon Graphics en 1992. OpengGL est compatible sur tous les ordinateurs de nos jours en effectuant simplement des mises à jours de nos cartes graphiques. OpenGL est aujourd'hui en version 4.5 et utilise un fonctionnement en « tout shader » .

2.3.2) L'Oculus Rift

2.3.2.1) Introduction

Comme dit précédemment, l'Oculus Rift est un masque de réalité virtuelle (figure 1). Celui-ci est développé par Oculus VR, entreprise basée en Californie et rachetée par Facebook en mars 2014. L'origine de l'Oculus Rift remonte à un projet financé par une plateforme de financement collaboratif (Kickstarter). Ce projet avait alors levé prêt de 91 millions de dollars. Le masque permet une immersion totale dans une scène en trois dimensions en suivant notamment les mouvements de la tête. Les personnes le portant se sentent alors présentes physiquement dans la scène. Oculus VR a donc créé une nouvelle expérience utilisateur qu'il souhaite rendre disponible au grand public en 2015 à un prix avoisinant les 300€. Utiliser principalement pour le divertissement, le masque est désormais utilisé dans d'autres domaines comme l'éducation ou les simulations scientifiques. Le masque possède aujourd'hui deux versions de développement, la dernière étant distribuée depuis aout 2014.

Figure 1 : L'Oculus Rift

2.3.2.2) Fonctionnement

L'Oculus Rift est composé de (Figure 2):

- Un écran 60 Hz d'une résolution de 640*800 par œil (DK1) (augmenté a 75Hz et 1080p avec le DK2)
- Deux lentilles (une pour chaque œil)
- Un gyroscope 3 axes pour mesurer les accélérations angulaires
- Un magnétomètre 3 axes pour mesurer les champs magnétiques
- Un accéléromètre 3 axes pour mesurer l'accélération (gravitationnelle inclus)
- Un port USB
- Un port HDMI
- Une quarantaine de LED infrarouges et une caméra pour positionner le casque dans l'environnement (uniquement dans le DK 2)

Afin de faire fonctionner le masque et d'interagir avec lui, un SDK écrit en C++ est disponible et permet :

- d'accéder aux différents capteurs,
- d'accéder aux propriétés du masque comme la distance inter-pupillaire, la hauteur des yeux, etc
- d'appliquer les différents filtres de rendu graphique afin d'ajouter une déformation à l'image pour qu'elle corresponde au mieux à ce que verraient nos yeux et ainsi obtenir un rendu réaliste.

Figure 2 : A l'intérieur de l'Oculus Rift

Afin de mieux comprendre le fonctionnement du masque, nous allons passer par un peu de théorie. L'Oculus Rift demande à ce que la scène soit rendue graphiquement en « split-screen stereo », autrement dit, l'écran doit être divisé en deux verticalement afin d'avoir une partie pour l'œil gauche et une autre partie pour l'œil droit. Lors du procédé de rendu, la distance inter-pupillaire (distance entre les deux yeux (en moyenne : 65mm) est utilisée. En effet, la scène est rendue deux fois (une fois par œil) en translatant la caméra d'une vue a l'autre de cette distance inter-pupillaire. On obtient ainsi un effet stéréoscopique qui permet de voir en 3D et d'augmenter grandement l'effet d'immersion. De plus, la taille des écrans n'est pas capable de reproduire l'ampleur de notre champ de vision, c'est à ce moment là qu'interviennent les lentilles qui agrandissent l'images fournissant un champ de vision très large. L'immersion est encore une fois augmentée mais les lentilles déforment considérablement l'image en créant une distorsion en coussinets (figure 3.1). Les filtres à appliquer au rendu dont nous parlions précédemment servent donc en réalité à contrer cette déformation qui n'est donc plus visible pour l'œil humain. Pour ce faire, les filtres produisent une distorsion égale et opposée à savoir une distorsion en barillets (figure 3.2).

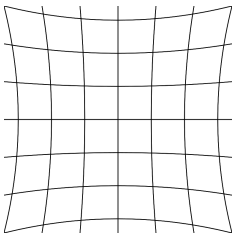

Figure 3.1 : Distorsion en coussinets

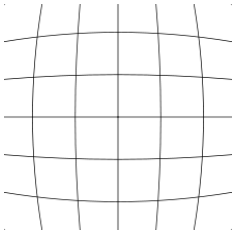

Figure 3.2 : Distorsion en barillets

Enfin, le programme doit corriger les aberrations chromatiques qui consistent en un effet d'arc en ciel aux contours des objets (figure 4). Ce phénomène bien connu en optique est produit par les lentilles.

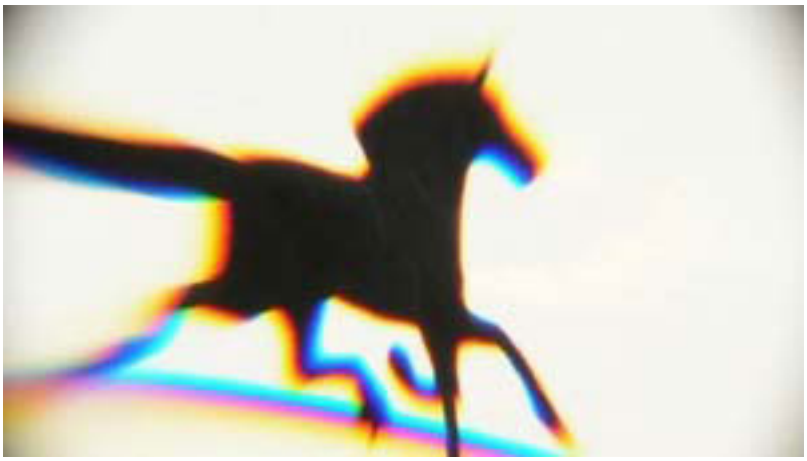

Figure 4 : Aberration chromatique

En pratique, lors du développement de l'application, l'algorithme principal de rendu graphique pour l'Oculus sera le suivant (figure 5, 6, 7 :

```
Initialisation des ressources graphique ;
Initialisation du SDK Oculus ;
Remplissage de la scène avec nos objets graphique ;
While l'application est en route do
        On récupère les entrées et on traite les capteurs de l'Oculus ;
        On met à jour les objets de la scène ;
        For chaque œil do
                On translate la caméra de la distance inter-pupillaire ;
                On applique les effets de distorsions Oculus ;
                On rend les objets de la scène ;
        End
End
On libère le SDK Oculus ;
On libère les ressources graphiques ;
```

Figure 5 : algorithme de rendu avec l'Oculus

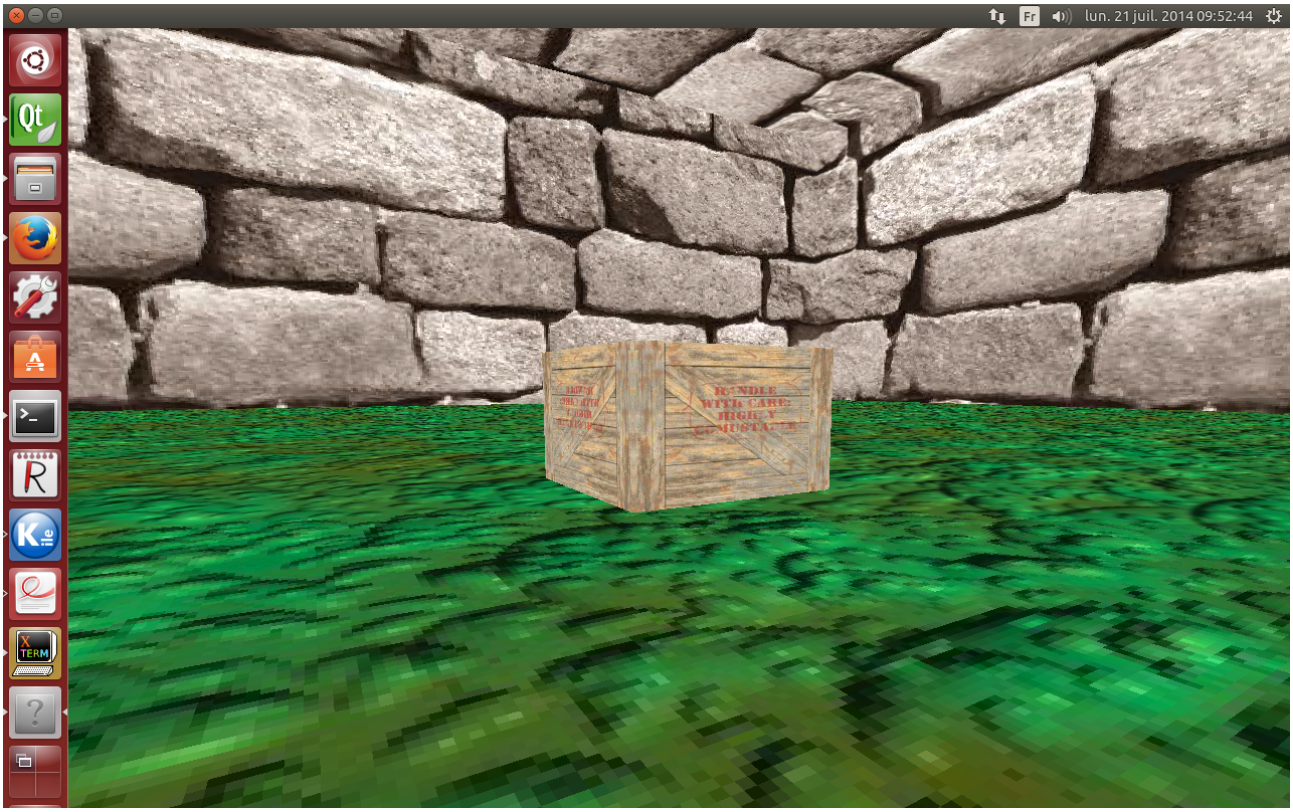

Figure 6 : un rendu sans Oculus

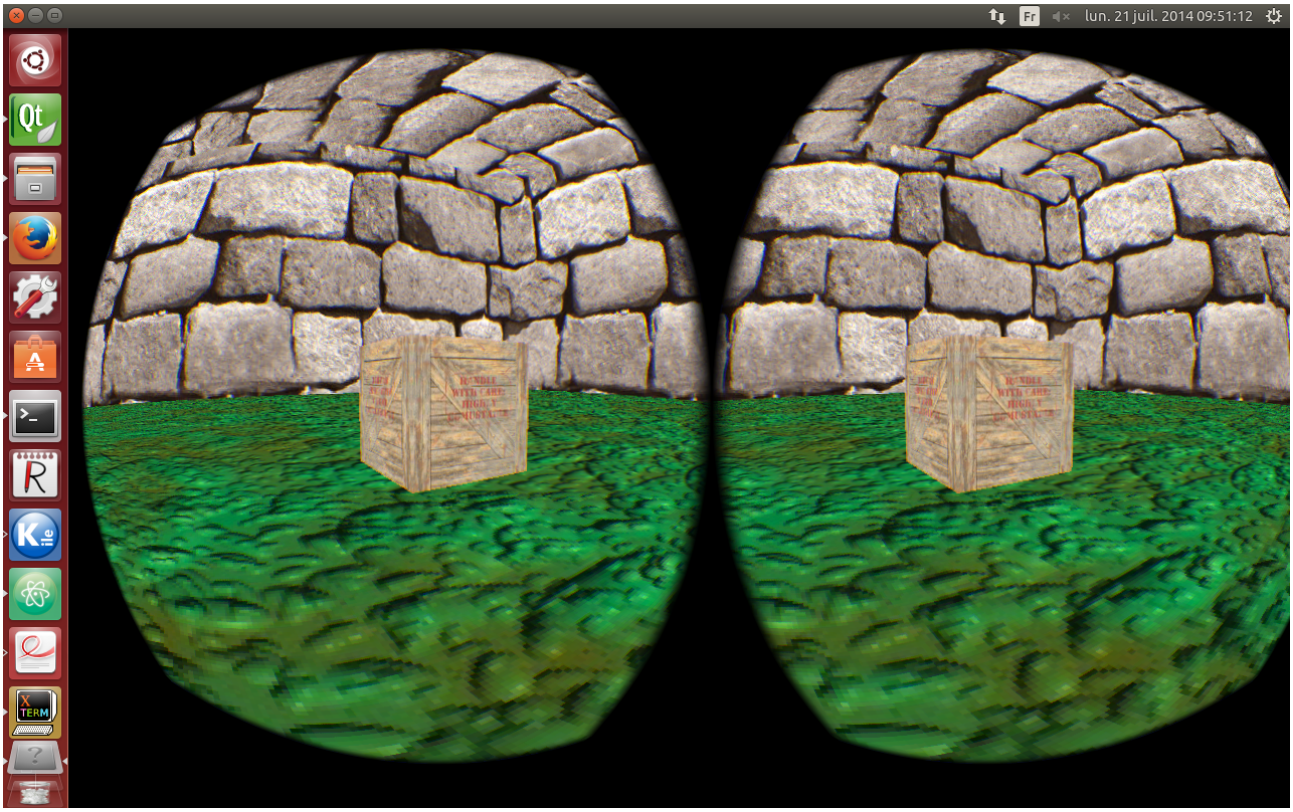

Figure 7 : un rendu avec Oculus

2.4) Contraintes

2.4.1) Langages

Avant de commencer le développement s'était posées la question du langage informatique à utiliser. Or, le SDK de l'Oculus est écrit en C++ et nécessite donc impérativement un compilateur C++. Je n'avais donc plus que le choix entre deux langages : le C++ bien évidemment et le C (pouvant être compilé avec le compilateur C++ et pouvant fonctionner ensemble dans un même programme). Afin d'harmoniser le code et de simplifier l'utilisation d'OpenGL, j'ai décidé d'opter pour le langage C++.

2.4.2) OpenGL

Il existe aujourd'hui quatre versions principales d'OpenGL mais une véritable fracture a eu lieu entre la version 2 et la version 3. En effet, depuis la version 3, OpenGL utilise le « tout shader » et la méthode de programmation a donc totalement changé. Le « tout shader » permet la configuration complète de toute notre scène (éclairages, rendu des vertices, etc). Ainsi, bien configurer ses shaders permet un grand gain de vitesse par rapport aux anciennes versions d'OpenGL. Enfin, OpenGL 4 n'étant encore pas présent sur tous les ordinateurs, le choix de version d'OpenGL s'est donc porté sur la troisième qui était ainsi pour moi le meilleur compromis entre vitesse et niveau de déploiement.

2.4.3) Portabilité

Le CDS, toujours dans une optique de disponibilité au plus grand nombre, souhaite toujours que ses outils et programmes soient disponibles quelque soit la plateforme que l'on utilise (Windows, Linux, Mac). Il a donc fallu que mon programme respecte cela et utilise du code pouvant être compilé sur toutes les plateformes. Le choix s'étant porté sur le langage C++ ne posait pas de problème puisque des compilateurs de C++ existent sur les trois plateformes. En revanche, il fallait faire abstraction des API spécifiques à une plateforme donnée, en fournissant es API générique. Un exemple est l'utilisation de la SDL, une bibliothèque de fenêtrage, ou d'OpenGL, une API de rendu graphique.

De plus, un soin particulier a été porté dans le développement afin d'éviter l'introduction de code spécifique à une plateforme donnée. Une solution a été l'utilisation maximale de la libraire standard du langage.

Enfin j'ai utilisé un outil de compilation multiplateforme, qmake, qui permet de compiler le code sur plusieurs plateformes différentes automatiquement.

2.5) Déroulement du stage et développement

2.5.1) Déroulement du stage

De manière générale, le stage s'est déroulé en trois temps : apprentissage, développement, optimisation.

En effet, la première semaine de stage a due être consacrée à la découverte de l'Oculus Rift (à travers l'essai de démos, de lecture de documents, etc.), à la découverte du SDK de ce dernier et à l'installation des outils de développement.

La deuxième semaine a consisté à se former à l'Oculus SDK et OpenGL, et à tester la faisabilité de l'intégration de l'Oculus Rift à un moteur de rendu 3D existant, Irrlicht. Malheureusement cela s'est révélé plus complexe que prévu (moins facilement intégrable avec l'Oculus) et la décision a été prise d'utiliser OpenGL sans framework, vu le temps imparti. Cette période a aussi servi à se former au C++

La troisième semaine a été centrée sur le développement de programmes minimalistes mettant en place OpenGL et l'intégration de l'Oculus Rift.

Les quatrième, cinquième et sixième semaines furent consacrées au développement du prototype en lui même

La septième et huitième semaines furent quand à elle utile à l'optimisation du prototype.

2.5.2) Développement

2.5.2.1) Bonnes pratiques

J'ai tout au long de mon stage versionné (avec Philippe, l'autres stagiaire) mon code avec git et Github. De plus j'ai régulièrement utilisé des outils d'analyse statique et dynamique pour jauger de la qualité de mon code et l'améliorer. En sus, j'ai activé les options de compilation aidant dans cette tâche : «-Wall», «-Wextra», «-Werror», . . . J'ai également gardé une constance dans le nommage des variables, l'indentation du code, etc. Enfin, j'ai utilisé deux compilateurs différents, clang et gcc, en comparant leurs performances, notamment avec les options d'optimisations «-O1», «-O2», «-O3», «-Ofast». Pour finir, j'ai utilisé le débuggeur gdb et l'analyseur de mémoire Valgrind pour vérifier que mon application était exempte de fuites mémoires (mis a part celles provenant du SDK Oculus pour lesquelles je ne peux rien faire).

2.5.2.2) Design pattern

Les design patterns sont des motifs de programmation, des «façons de faire» que l'on retrouve régulièrement pour résoudre des problèmes bien connus. Ce ne sont pas des «tours de magie» ou des «gadgets à la mode», mais des solutions pratiques à mettre en place quand le besoin s'en fait sentir, ou pour améliorer le code. Pendant ce stage, j'ai utilisé plusieurs design patterns. Voici lesquels et pourquoi :

Singleton pattern

Design pattern bien connu et parfois décrié, il permet de limiter le nombre d'instances d'un objet, typiquement à 1. Je l'ai utilisé pour la classe Oculus, afin d'éviter de faire l'initialisation (et la libération) du SDK Oculus plusieurs fois.

Game Loop pattern

Ce pattern décrit la boucle de rendu typique d'une application de rendu graphique. Je l'ai utilisé pour le rendu normal et pour le rendu Oculus.

Flyweight Pattern

Ce pattern sert à améliorer les performances d'une application où sont présents de nombreux objets identiques, mis à part la valeur de certaines propriétés, et qui utilisent des ressources similaires. Au lieu que chaque objet possède une instance de la ressource, générant une perte de performances inutile, ce pattern propose de mettre en commun tout ce qui peut l'être. Chaque objet aura alors un pointeur vers la ressource partagée. Ce partage est transparent pour l'application qui continue à utiliser les objets comme bon lui semble, en

utilisant les informations particulières de chaque objet.

Dans ma situation, la simulation concernait des milliers voire des millions d'objets célestes. Dans les cas que j'ai rencontrés, il s'agissait toujours d'un seul type d'objet céleste, par exemple une étoile. Si l'on choisit de représenter une étoile par un modèle 3D, une sphère par exemple, et une texture plaquée sur cette sphère, et que l'on charge la ressource «texture» à partir d'un fichier, il est inconcevable de lire ce fichier des millions de fois par seconde (à chaque rendu), ou même des millions de fois au démarrage du programme (initialisation des textures des objets célestes). On choisit alors de charger cette ressource une fois pour toute au démarrage. On lit une et une seule fois le fichier. Chaque étoile possède un pointeur vers cette texture partagée. Les seules différences entre les étoiles sont les informations contextuelles (position, densité, âge, …).

On peut alors faire le rendu sans problème de performances : au final, on fait une seule lecture de fichier, et l'on a une seule texture en mémoire, au lieu de milliers/millions auparavant. L'initialisation prend ainsi moins de temps et la libération mémoire également à la fin du programme.

En pratique, l'application de ce design pattern a permis de transformer un démarrage poussif (dizaines de secondes) avec 100 objets célestes, en un démarrage en environ 3s pour 1000 objets.

RAII

«Resource Acquisition Is Initialization», l'acquisition de ressources c'est leur initialisation. Ce pattern permet une gestion efficace et simple de la mémoire et est inhérent au C++.

Il repose sur l'assurance fournie par le langage que le destructeur d'un objet sera appelé lorsque l'on sort du bloc où il a été défini. Ainsi, lorsque l'on acquiert une ressource, par exemple ouvrir un fichier, et que cette ressource a besoin d'être libérée plus tard, par exemple fermer ce fichier, on peut automatiser ces deux actions en les plaçant à un endroit adéquat du code.

En pratique, on fait l'acquisition dans le constructeur de l'objet utilisant/représentant la ressource (d'où le nom), et la libération dans le destructeur. Pour l'utilisateur de la classe ces opérations sont transparentes et il est assuré qu'elles seront exécutées auto- matiquement. Pour ma part, j'ai utilisé ce pattern pour l'initialisation et la libération des ressources graphiques et du SDK Oculus.

Data Locality

Ce design pattern est un moyen d'optimiser les performances d'un programme en tirant parti du cache du CPU.

La fragmentation est l'utilisation inefficace de l'espace mémoire. C'est un phénomène qui se produit lorsque sont effectuées de multiples allocations et désallocations de mémoire, engendrant alors une baisse de la capacité mémoire et/ou des performances.

Dans mon cas, j'ai donc transformé mon tableau de pointeurs sur objets célestes en un tableau d'objets célestes. Cela est rendu possible par le fait que je ne dispose que d'un seul type d'objet céleste dans ma simulation. De plus l'usage de pointeurs dans mon programme est majoritaire dans la partie génération et rendu d'objets célestes de part leur nombre. Cependant il est à noter que dans le cas d'un nombre très grand d'objets (de l'ordre du million), il est possible de dépasser la taille de la pile, créant alors un «stack overflow».

2.6) Difficultés rencontrées et optimisations

2.6.1) Les librairies

Une des principales difficulté rencontrée fut de faire cohabiter ensemble les différentes librairies (quatre au total) et cela sur les trois plateformes différentes. Tout au long du développement, j'ai du changer de version de certaines librairies car certaines provoquaient entre elles des conflits, étaient mal chargées, … Au final, c'est la réalisation même du Makefile de l'application qui fut loin d'être simple.

2.6.2) Oculus Rift et OpenGL

Je ne connaissais aucun de ces deux éléments, l'Oculus Rift malgré son SDK ne possède pas beaucoup de documentations en ligne. De plus, ce dernier étant en développement intense, de nouvelles versions de SDK sortent régulièrement et il fallait a chaque fois réadapter son code pour qu'il fonctionne sur les nouvelles versions. Ainsi, en l'espace de deux mois, j'ai du apprendre l'utilisation du SDK de l'Oculus quatre fois. De plus, un bug lors du rendu Oculus faisait complètement « freezer » l'ordinateur obligeant a le redémarrer au bouton. Ce bug fut résolu par la suite après une longue période.

Concernant OpenGL, je ne connaissais pas grand chose sur lui et de manière générale, je n'avais encore jamais réalisé de 3D (en état aussi bas niveau). Ce fut donc pour moi une grande découverte et malgré les nombreuses lignes de codes à écrire pour de simples choses (vu que je n'ai pas utilisé de surcouche à OpenGL), son apprentissage fut tout de même relativement aisé car, à l'inverse de l'Oculus Rift, OpenGL est fortement documenté sur le web.

2.6.3) Taille des données

J'ai travaillé sur des données avoisinant le million d'objets. Ces données proviennent de simulations à haute résolution de la réionisation de galaxies. Cela a posé principalement un problème de performances. En effet, c'est en travaillant avec de tels nombres que l'on se rend compte de la disparité CPU (processeur) / GPU (carte graphique). En effet, malgré un processeur avec 16 GB de RAM, le fait de parcourir tous les objets pour les afficher (une fois par frame, O(n)), prenait plus de 16 millisecondes, nombre critique dans le domaine du rendu graphique, puisqu'il correspond au temps de rendu maximal d'une frame si l'on veut un rendu à 60 FPS (frame per seconds), ce qui fournit une expérience correcte pour l'utilisateur : 1000ms/60 = 16.666ms.

En sus, comme j'ai travaillé avec des cubes de données (corps célestes dont les coordonnées spatiales se trouvent toutes contenues dans un cube, typiquement de taille 64*64*64 ou 100*100*100), ce qui peut donner une boucle de rendu de complexité $O(n^3)$, empirant alors le temps de rendu.

De plus, il faut garder à l'esprit que l'objectif final est d'avoir un rendu Oculus valide. Or le SDK Oculus fait un double rendu (un pour chaque oeil), en appliquant des transformations matricielles pour chacun des rendus. Il est donc primordial d'avoir des FPS corrects dans le rendu graphique normal. Cependant, je me suis aperçu que la carte graphique ne rencontrait pas de problème de temps de rendu, gardant la plupart du temps un temps de rendu d'une frame inférieur à la milliseconde.

Dès lors, plusieurs solutions se sont présentées :

Travailler avec un seul objet graphique

Cela consiste à avoir un seul objet dans le programme qui contient les coordonnées de tous les objets célestes. On «boucle» donc sur un seul objet et on envoie toutes les positions des objets célestes en une seule fois comme s'il n'y avait qu'un objet et la carte graphique fait tout le travail. Cela fonctionne mais est peu flexible (comment faire pour sélectionner un seul objet céleste pour afficher des informations à son sujet?) et on atteint les limites de la carte graphique pour un très grand nombre d'objets. Cependant le CPU a un minimum de travail. Dans le cadre du prototype, c'est tout de même cette solution qui fut gardé car nous sommes passés de 20 000 objets affichable à près d'un million.

Octree

Un Octree (cf figure 8 et 9) est un arbre où chaque nœud (appelé «octant») compte jusqu'à 8 fils. Il correspond à une partition d'un espace cubique, à la manière d'un quadtree en 2D, et permet de diviser notre scène en régions, contenant elles-mêmes des sous-régions et ainsi de suite. On peut alors décider d'afficher seulement les régions voisines de notre position sans afficher les régions que l'on ne peut pas voir ou qui sont trop lointaines. Cette solution est applicable uniquement lorsque nous sommes à l'intérieur du cube. En effet, si l'on visualise le cube depuis l'extérieur, on ne voit plus le cube dans son ensemble mais une partie ce qui n'est pas ce que l'on souhaite.

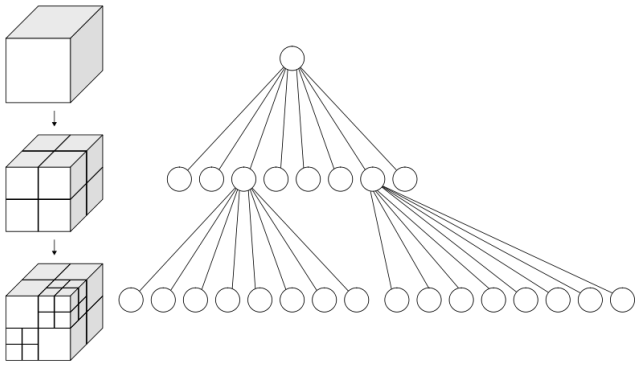

Figure 8 – Schématisation d'un octree

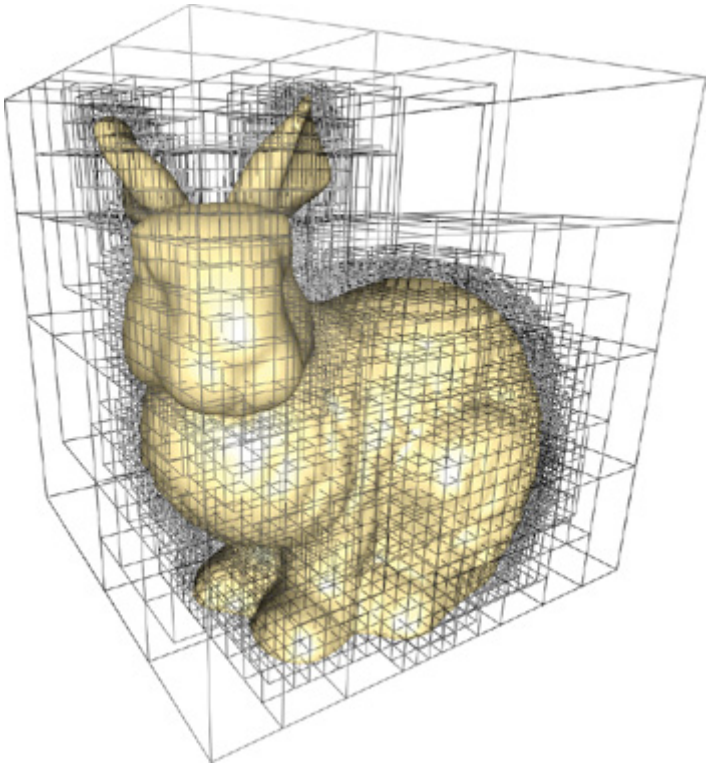

Figure 9 – Octree en action

On optimise alors le rendu à la fois sur le CPU (moins d'objets parcourus dans la boucle de rendu à chaque frame) et sur le GPU (moins de données envoyées et rendues graphiquement à chaque frame).

Initialement, mes objets étaient tous stockés dans un tableau que je parcourais à chaque frame pour les afficher. Pour 1000 objets, j'avais une moyenne de 5 FPS. J'ai alors mis en place un Octree. J'ai eu alors des FPS variant entre 30 et 60 FPS, avec une moyenne de 55 FPS (le rendu est limité à 60 FPS maximum pour ne pas surcharger le CPU/GPU), quelque soit le nombre d'objets présents dans la scène, ce qui est acceptable. En effet, l'Octree permet d'afficher un nombre moyen constant d'objets quelque soit notre position. Avec un cube de taille 128*128*128 et 32768 objets, le temps de génération de l'Octree est d'environ 120s. J'affiche l'octant (et donc tous les objets célestes se trouvant dans cet octant) où la caméra se trouve et les 6 octants immédiatement voisins, avec une taille d'octant de 8. Pour résumer, on sacrifie le temps de démarrage du programme au profit des performances à l'exécution (toujours en cas de visualisation interne comme dit précédemment).

2.7) Résultat et perspectives

2.7.1) Résultat :

Au final, le cahier des charges fut réalisé malgré de nombreuses difficultés notamment au niveau de l'optimisation et de l'apprentissage du SDK de l'Oculus. Le programme possède bien ses deux fonctionnalités à savoir l'affichage d'un nuage d'étoile en modifiant les couleurs des étoiles pour suivre ses données (figure 10 et 11) et l'affichage de cube de données utilisant de la même manière une échelle de couleurs pour représenter les informations de la déionisation de la galaxie (figure 12 et 13). Octree a également été implémentée pour y faire des tests (figure 14 et 15).

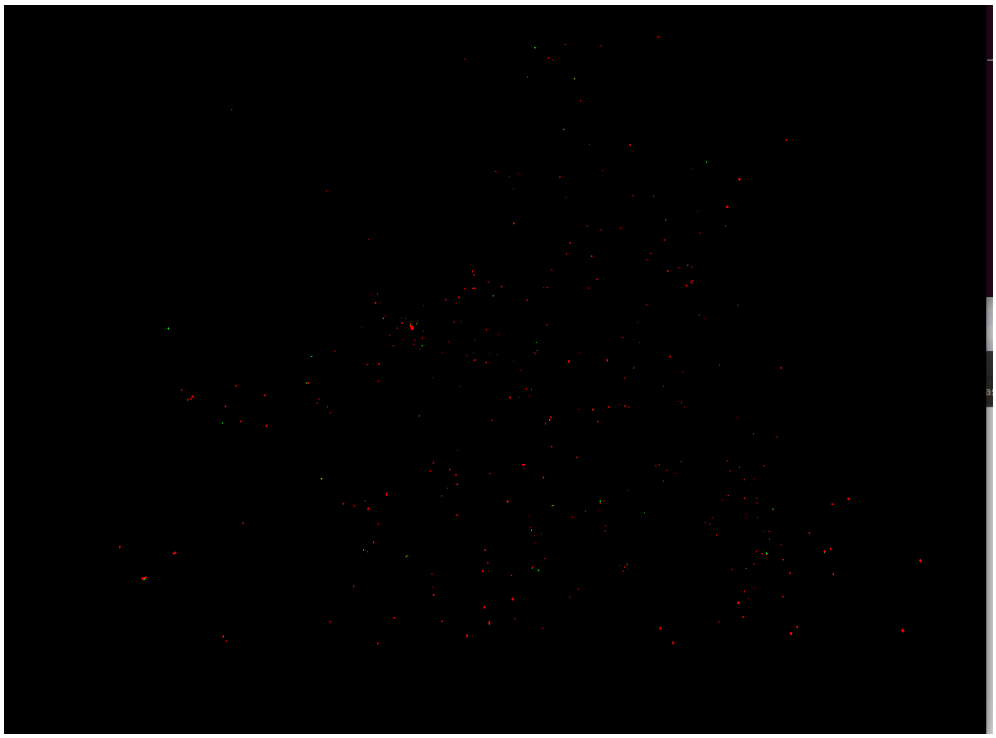

Figure 10 : Prototype avec vu d'un nuage d'étoile 1

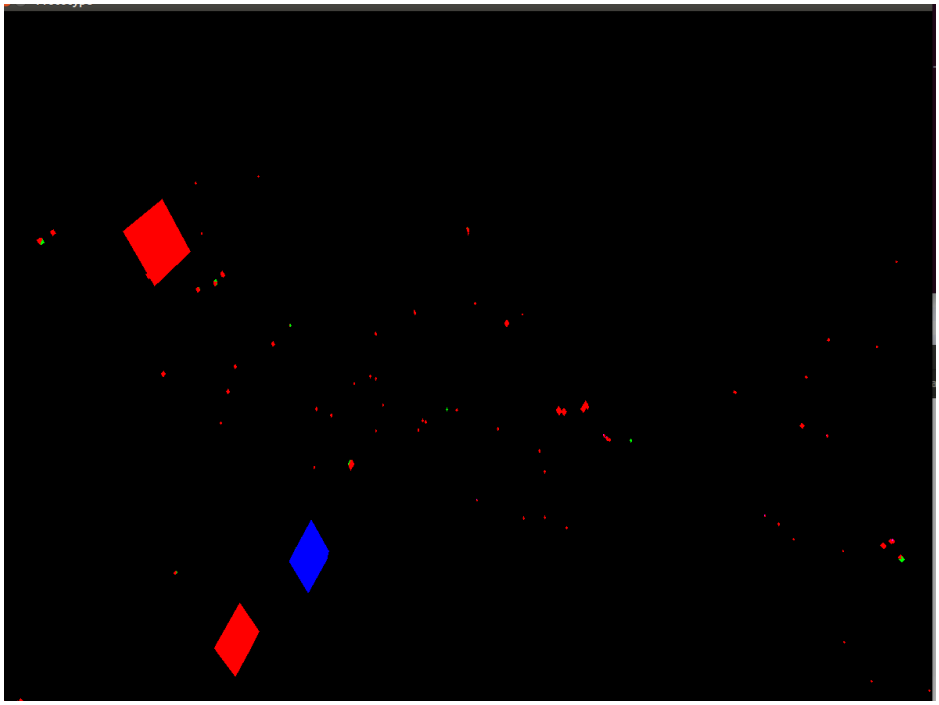

Figure 11 : Prototype avec vu d'un nuage d'étoile 2

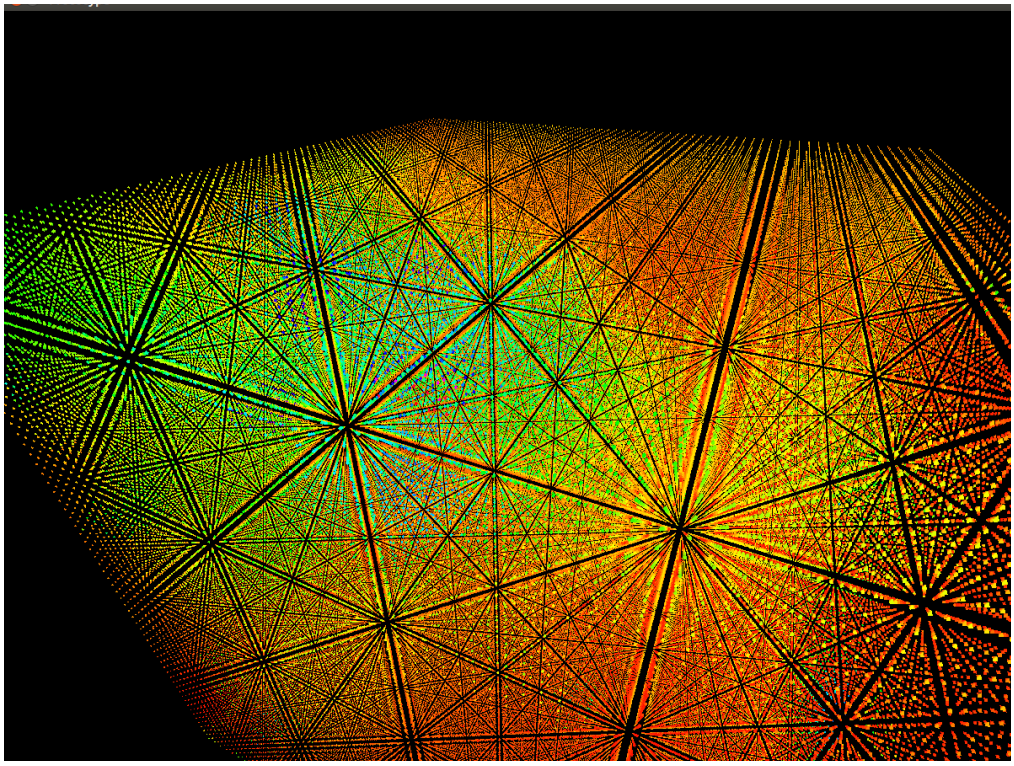

Figure 12 : Prototype avec vu d'un cube de données 1

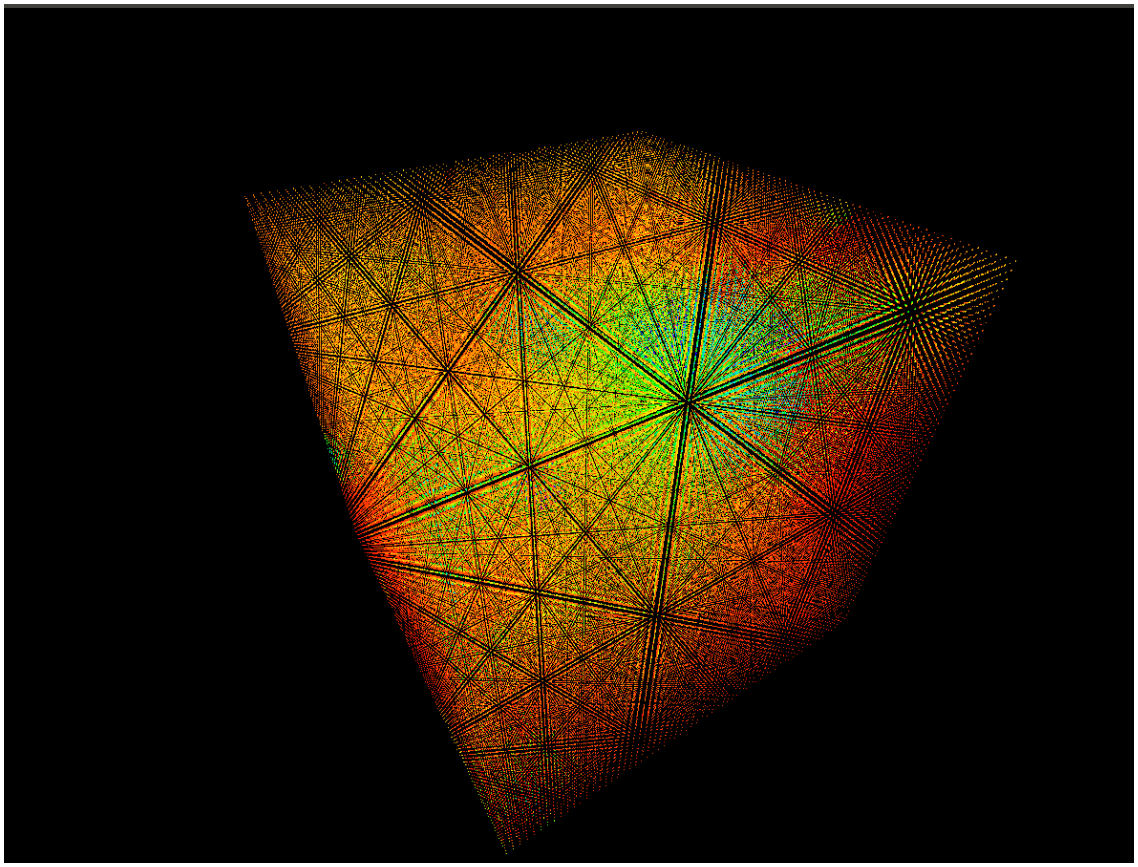

Figure 13 : Prototype avec vu d'un cube de données 2

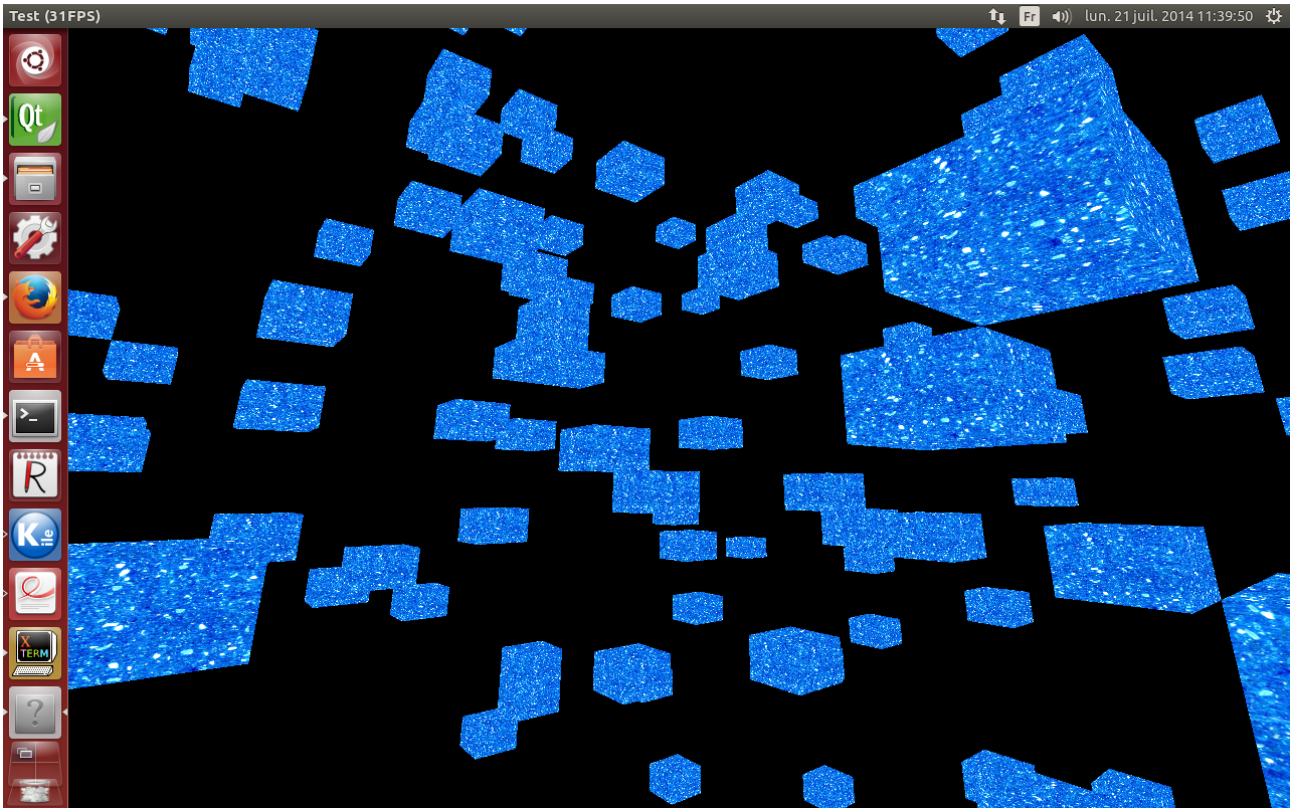

Figure 14 : Prototype en mode Octree

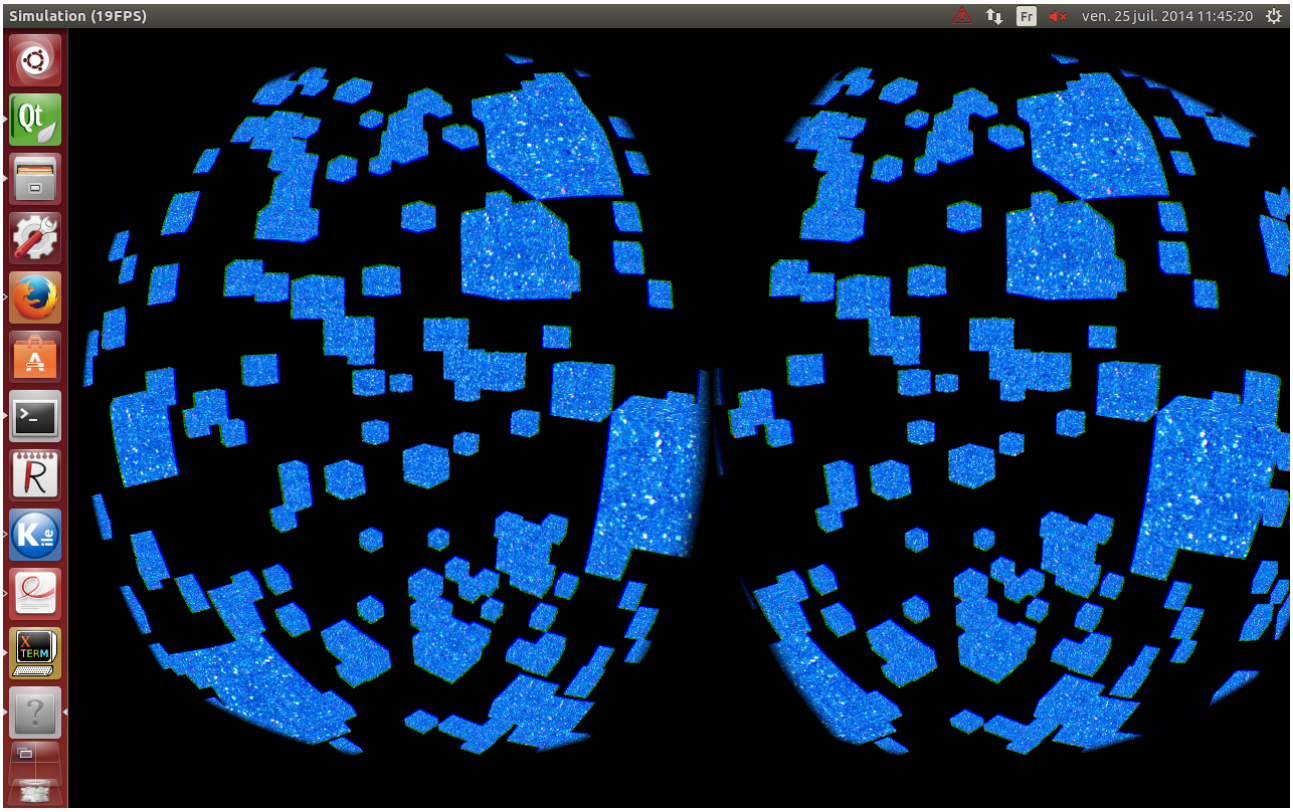

Figure 15 : Prototype en mode Octree avec la vue Oculus

2.7.2) Perspectives

Toujours dans l'optique de ce prototype, trois perspectives ont dors et déjà vu le jour :

- Optimisation : malgré un gros travail porté sur l'optimisation, l'objectif futur sera d'optimiser encore plus afin de réussir à afficher des cubes de données toujours plus conséquent.
- Options d'affichages : une échelle de couleurs permet l'affichage des données mais il serait préférable pour la suite de pouvoir afficher différentes informations de différentes manières selon le choix de l'utilisateur. Rajouter la possibilité de sélectionner une étoile ou une donnée rentrant également dans cette perspective.
- Meilleur affichage : dans le futur, il faudrait également améliorer l'affichage du cube de données afin de relier tous les points entre eux et rendre l'ensemble visible avec un choix d'opacité pour voir plus ou moins loin dans le cube.

Conclusion

Après ces 8 semaines de stage passées au Centre de Données astronomiques de Strasbourg, j'ai réussi à mettre au point un prototype utilisant des technologies que je ne connaissais pas auparavant. Ainsi, mes connaissances se sont trouvées fortement enrichies par ce dernier et l'intérêt du stage était à la hauteur des défis rencontrés. J'ai également pu approfondir mes connaissances sur des sujets génériques dans le domaine correspondant a ma formation d'ingénieries logiciel. Je remercie fortement l'observatoire et M.André Schaaff mon encadrant pour m'avoir laissé carte libre dans la réalisation de ce prototype. Je ressors de ce stage fortement enrichi en connaissances, satisfait du travail accompli, intéressé par de nouveaux domaines. Enfin, ce stage m'a permis de confirmer mon choix d'ingénierie logicielle en me permettant de mettre en application les différentes notions abordées dans le cadre de cette spécialisation.

Glossaire