Visualisation de données astronomiques en 3D

Pierre Lespingal

14 octobre 2015

1 Résumé

Ce rapport fait état de mon stage de deuxième année à l'ENSIIE Strasbourg qui s'est déroulé du 10 Juin au 18 Août 2015, au sein du Centre de Données astronomiques de Strasbourg. Le sujet porte sur la visualisation 3D sur navigateur web de données astronomiques. L'accent est porté sur les performances, la mise en valeur des informations et l'exploration de technologies émergentes telles que la réalité virtuelle.
Mon maitre de stage et tuteur est André Schaaff, Ingénieur de recherche au CNRS.

2 Introduction

2.1 L'entreprise

L'observatoire L'observatoire astronomiques de Strasbourg est un Observatoire des Sciences de l'Univers, une école interne de l'Université de Strasbourg ainsi qu'une unité de recherche mixte entre l'Université et le CNRS.
Construit en 1881, il possède la troisième plus grande lunette de France et intègre le planétarium de Strasbourg.
Il est actuellement dirigé pas Hervé Wozniak.

Départements L'observatoire astronomiques de Strasbourg comporte trois équipes de recherche distinctes :

- **L'équipe de recherche galaxies** S'intéresse à la formation des galaxies et plus particulièrement la voie lactée, afin d'en saisir la formation et l'évolution.
- **L'équipe de recherche hautes énergies** S'intéresse au sources émettrices de rayon X, aux objets compacts et aux noyaux des galaxies.
- **Le CDS** Le CDS s'occupe et développe des service d'accès aux données astronomiques. Équipe d'accueil du stage, ce département est présenté plus en détail ci-dessous.

Le CDS Le CDS a été créé initialement en 1972 sous le nom de Centre de Données Stellaires, pour finalement changer en 1983 et devenir le Centre de Données astronomiques de Strasbourg.
Il se consacre à récolter et rendre accessible des données astronomiques provenant du monde entier.
Ses missions sont :

- Collecter des informations sur les objets astronomiques et les rendre accessibles via des moyens informatiques.
- Mettre à jour ces informations.
- Diffuser les résultats à la communauté scientifique.
- Effectuer des recherches sur ces données et leur représentation.

Le CDS collabore notamment avec le Centre National d'Etude Spatiale, la European Space Agency et le European Southern Observatore.

Avec des problématiques tel que le Big Data, un enjeu important du monde de l'informatique actuel, et les avancés technologiques régulières, le CDS entreprend de nombreux projet R&D pour proposer de nouvelle façon de gérer et présenter ces données. L'essor des technologies liées à la représentation graphique et à la réalité virtuelle fait ainsi l'objet depuis quelques années de sujet de stage au sein du CDS.

2.2 Sujet

Le Centre de Données astronomiques de Strasbourg mène une forte activité de R&D afin de maintenir la qualité de ses services et d'anticiper les évolutions techniques.
Le volume croissant des données (notre domaine est Big Data par nature) implique des investigations au niveau de l'accès/recherche, la visualisation et l'interprétation.
Pour la visualisation et l'interprétation nous travaillons notamment dans le domaine de la réalité virtuelle en utilisant des outils comme l'Oculus Rift (et la Google Cardboard).
Après plusieurs prototypages en 2014 basés sur l'utilisation d'OpenGL et des kits de développements des outils concernés, nous souhaitons cette année aller plus loin en réalisant des applications plus ouvertes en exploitant WebGL pour visualiser en 3D tout en ayant la possibilité d'exploiter un masque de réalité virtuelle. Des librairies comme Three.js ou Babylon.js offrent la possibilité de générer des vues Oculus Rift. Nous souhaitons également que ces applications soient des prototypes avancés permettant leur utilisation au niveau professionnel ou à minima lors d'évènement publiques (journées de la science, planétarium, etc.).
Le travail du stage porte sur la visualisation de données de simulation et l'affichage de cubes de données en exploitant WegGL et l'Oculus Rift (DK2). La taille des cubes de données pourra atteindre dans les années à venir 2048 au cube et sans doute plus. Une donnée est composée d'une part de coordonnées X,Y,Z et de caractéristiques (masse, age, etc.). L'étudiant sera confronté à plusieurs problèmes : comment visualiser de façon fluide le volume de données ?, comment gérer de façon performante la mise à jour de l'affichage en fonction des caractéristiques que l'on veut mettre en avant ?, etc.
Il serait également intéressant d'étudier la prise en compte de la dimension tem-

porelle.
L'étudiant pourra s'appuyer sur l'existant constitué de travaux déjà réalisés dans le cadre d'études précédentes.

2.3 Cadre de travail

Le stage s'est déroulé dans les locaux du CDS. La bibliothèque du bâtiment Est était mise à disposition comme lieu de travail. Celle-ci a accueilli plusieurs stagiaires durant la période estivales, et chacun disposait d'une machine comme outil de travail.
Un autre élève de l'ENSIIE Strasbourg, Nicolas Buecher, avait pour les mêmes périodes un sujet de stage autour de cette même application de visualisation. Le stage avait donc un caractère collaboratif et une partie du travail a été effectué à deux.

L'observatoire dispose d'un parc ou se détendre et passer la pause de midi, ainsi que d'une cuisine mise à disposition de tous.
Pendant une première période du stage, il était possible d'assister à des conférences autour de l'astronomie le vendredi matin, précédées d'un petit déjeuner commun avec l'ensemble des occupants de l'observatoire. Le département informatique proposait en plus des réunions *infusion* pour partager autour du travail de chacun et des techniques liées à l'informatique.

2.4 Remerciement

Je tiens à remercier mon maitre de stage, Andre Schaaff, pour ses conseil et son attention à mon égard.
Je tiens aussi à remercier les stagiaires présents cet été, pour leur sympathie et une bonne ambiance de travail.
Je tiens de même à remercier l'ensemble du personnel de l'observatoire, pour son accueil, et plus particulièrement Nicolas Deparis et Nicolas Gillet, tous deux membre de l'équipe Galaxie, avec qui nous avons pu discuter de mon stage et du développement de l'application.
Je tiens enfin à remercier Hervé Wozniak, pour m'avoir accueilli au sein de L'observatoire astronomique de Strasbourg.

Première partie
Le sujet

3 Appréhension du sujet

3.1 Reformulation du sujet

Le CDS propose des fichiers de données de très grandes tailles décrivant les caractéristiques d'un ensemble de points dans l'espace, issue de simulations.
Ces fichiers sont appelés cube de données car ils décrivent une partie cubique de l'espace.
Ces ensembles de fichiers peuvent représenter des millions de points, avec pour chacun d'eux leurs positions dans l'espace en trois dimensions, ainsi que diverses informations telles que leurs masses, leurs ages ou des valeurs liées à l'énergie suivant le type d'objet. Ces points donc, peuvent correspondre à des données réelles, telle que des étoiles, ou bien être une représentation d'autres entités, comme une répartition de matière noir ou de gaz.
On dispose formellement d'un nuage de point, mais encodé dans des fichiers qui ne peuvent être interprétés par l'humain.
On souhaite pouvoir visualiser ces données via un moteur graphique pour rendre compte de la répartition du nuage de point dans l'espace, ainsi que des caractéristiques de chacun de ses points.
Le stage propose de réaliser ceci via les technologies web, i.e. dans un navigateur internet.
L'objectif est donc de concevoir une application web permettant la visualisation de cube de données ainsi que l'interaction avec le nuage de points obtenu.

3.2 Objectif

Dans une dynamique de recherche et développement, le sujet était libre autant au niveau des outils à utiliser que dans la forme qu'allait prendre l'outil de visualisation au fil des jours.
Puisque que le rendu 3D s'effectue dans un navigateur, et non pas nativement sur l'ordinateur, les performances étaient l'un des enjeux majeurs. Si la machine utilisé pour le développement offrait une bonne puissance de calcul et de rendu, l'application était voué à être utilisée sur différents ordinateurs de capacité moindre. De nombreuses recherches étaient donc à mettre en place, pour tirer profit au maximum des possibilités offertes par WebGL et Javascript. Une bonne compréhension des mécanismes liés à l'interaction entre le processeur de l'ordinateur et la carte graphique était requise pour proposer un rendu fluide tout en gérant en temps réel les calculs à effectuer et l'interaction utilisateur.
Le code de l'application se devait d'être maintenable, pour être facilement compréhensible et modifiable par d'autres personnes. Il fallait donc apporter un soin particulier à l'écriture et à l'architecture du code source.
Enfin, il fallait réfléchir aux fonctionnalités à offrir à l'utilisateur, via une interface convivial. Des échanges avec le personnel du CDS qui pouvait potentiellement être amené à utiliser l'application ont eu lieu pour mettre en place les bonnes spécifications.

3.3 Existant

Deux mois avant le début de ce stage un étudiant d'ITU, Arnaud Steinmetz, s'est lui aussi intéressé en tant que stagiaire au technologie du WEB et de la 3D.
Son objectif était de proposer un prototypage de l'application de visualisation, en étudiant les outils et bibliothèques disponibles et leurs fonctionnements.
ce premier défrichage a permit de rentrer très vite dans le sujet, avec une première mouture du logiciel opérationnel.
Il était déjà possible d'afficher un cube de donnée à l'écran, et d'observer une déplacement de particule en interpolant deux cube de données.
Plusieurs recherches avaient aussi été faites sur les performances de l'application, ce qui a permit de se diriger directement vers les bonnes technologies et pratiques, sans tomber sur de trop nombreux écueils et idées qui n'auraient pas été viables au final.

4 Travail réalisé

Voici l'application telle qu'elle l'était à la fin du stage. Cette vision globale des fonctionnalités retenues et implémentées est nécessaire à la compréhension des aspects techniques et à la phase de développement.

4.1 Les données

Fichier de données Les cubes de données sont issus de simulations.
Les types de données utilisées au fil du stage sont au nombres de trois :

- Données d'un million d'étoiles.
- Données de répartition de matière noir, pour deux millions de particules.
- Données de quelques centaines d'étoiles, associé à la matière noir.

De plus, chaque type de données peut comprendre plusieurs cube de données, appelés *snapshot*, qui correspondent aux différentes positions du nuage de point à des instant donnés.

Nous nous intéresseront plus particulièrement aux données de matière noir. Celles-ci sont encodées dans des fichiers au format binaire.
Ces fichiers sont au nombre de 127 pour un cube de données, pour une taille au total de 80Mo.
On dispose alors de deux cubes de données, l'un correspondant au temps 0, ou les particules sont correctement alignées, et l'autre à un temps ultérieur, ou les particules se sont agglomérées dans des régions de l'espace.

Pour comprendre le chargement et l'utilisation des fichiers de données, voici un exemple de structure des cubes de données associés à la matière noir :

Case	0	1	2	3	4	5	6	7	8	9
Information correspondante	Position			vecteur vitesse			identifiant	Masse	Epot	ekin
	x	y	z	x	y	z				

FIGURE 1 – Organisation des fichiers de matières noir

Les fichiers de données ne comportent que ces informations sur plusieurs lignes, et il ne reste plus qu'à les extraire dans l'application.

Script Pour charger et lire ces fichiers encodés, il faut disposer d'un script adéquat qui va parser les fichier et retourner une structure de donnée bien précise.
Le choix des script est à la responsabilité de l'utilisateur, et il est aisé d'ajouter ses propres scripts de chargement de fichier dans l'application pour pouvoir utiliser son format de fichier.

Chargement des fichiers L'application propose une interface simpliste pour charger les données.
Un explorateur de documents permet de sélectionner en une seul fois l'ensemble des fichiers correspondant à un cube de données.
Dès lors, le cube de données est créé et est près à être affiché.
On peut ajouter autant de données que nécessaire, et charger pour chaque type de données autant de snapshot que l'on souhaite.

4.2 Les vues

Une vue correspond à un canvas de rendus.
On peut voir une vue comme un espace de travail ou l'on pourra afficher toutes les données chargée préalablement et interagir avec.
Par défaut, l'ensemble de l'application correspond à une seul vue, mais d'autres modes de visualisation sont disponible.

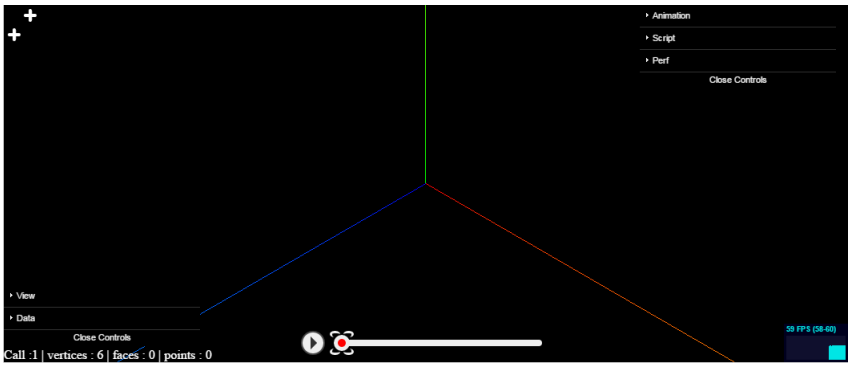


FIGURE 2 – Interface de l'application avec une seule vue

Multivues On peut en effet accéder à plusieurs vues en même temps, et ainsi scinder l'écran en deux parties.
Chaque partie ou vue et indépendante et permet d'afficher ou non les données chargés.
Pratiquement, on peut afficher un type de donnée sur l'une des vues et un autre type sur l'autre.

On peut aussi y afficher les même données mais avec des caractéristiques et une personnalisation différente.

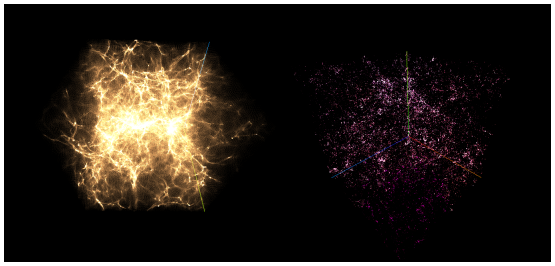


FIGURE 3 – Deux cube de données différents en mode multiview

Vue Cardboard On dispose aussi d'un affichage adapté à la visualisation avec une Google Cardboard.
Cette fonctionnalité a été implémentée par Nicolas Buecher et correspond à un dédoublement de l'image rendus avec un certain décalage pour créer l'effet d'immersion une fois l'application visualisée avec la carboard.

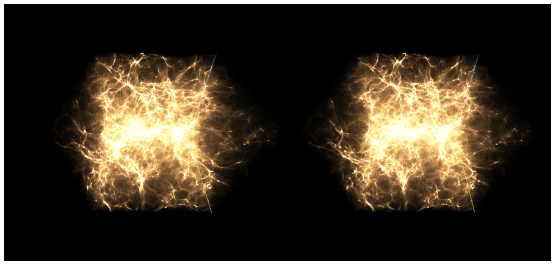


FIGURE 4 – Vue d'un cube de donnée en mode carboard

Vue Oculus Le vue Oculus rift se rapproche de la vue Carboard, en proposant de même une vue dédoublé dans un format HD, avec en plus une déformation. Cette vue est bien sur destinée à être observée via un casque de réalité virtuel Oculus.

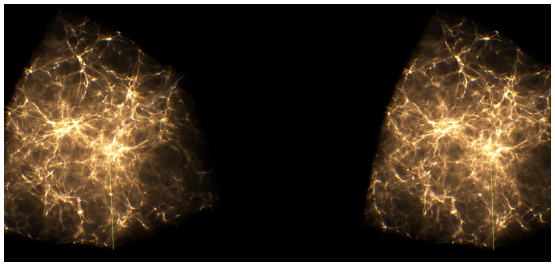


FIGURE 5 – Vue d'une cube de donnée en mode oculus

4.3 la navigation

Camera La camera est dirigeable à l'aide du clavier et de la souris.
Il est possible de se déplacer ou l'on souhaite dans l'application.
De base, chaque vue dispose de sa propre caméra, mais il est possible d'activer une caméra global qui est la même pour chaque vue, afin d'obtenir le même point de vue sur des jeux de données différents.
On dispose aussi de position prédéfini de la caméra, pour obtenir par exemple une vue de dessus ou une vue 3/4, ainsi qu'une rotation continue de la caméra autour du cube de données.

Sélection de particules On peut cliquer sur les particules avec la souris pour obtenir diverses informations, puis double cliquer pour se rapprocher avec une légère animation vers cette particule.

4.4 L'interface

Menu principal L'ensemble des vues précédentes peuvent être activés via un menu principal qui s'affiche par dessus l'espace de visualisation.

Menu global L'application dispose d'un menu principal qui est commun à l'ensemble des vues.
Celui ci dispose des options suivantes :

- Gestion du temps relatif de l'application utilisé pour l'interpolation temporelle des cubes de données.
- Gestion de la vitesse d'animation des particules.
- Choix du script de chargement de données à utiliser.

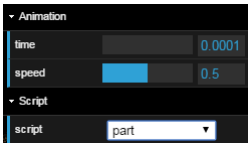


FIGURE 6 – Menu global

Gestion des données Un autre menu unique et global permet de gérer les données.
Il se présente sous la forme d'un tableau à double entrée, avec en colonne les types de données et en lignes les différents *snapshots*, i.e. les cube de données en des temps différents.
Il suffit d'élargir le tableau puis de charger via la case correspondantes les données voulues pour y avoir accès dans l'application.
L'interface permet aussi de savoir quelle donnée et quelles snapshot sont considérés comme actif, ce qui est utile par la suite pour appliquer des modifications sur les bons cube de données.

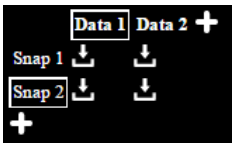


FIGURE 7 – Interface des données - ici la donnée 1 et le cube de données 2 sont actifs

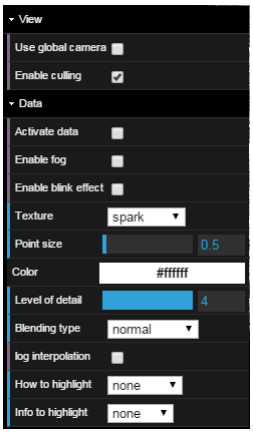


FIGURE 8 – Menu Local

Menu local Le menu local est associé à une seule vue, et est donc multiple en cas de multivues.
Celui ci propose de nombreuses fonctionnalités :

- Choix de l'utilisation de la camera global ou non
- Activation/Désactivation de l'occlusion de caméra
- Affichage ou non de la données active
- Activation/Désactivation de l'effet de brouillard
- Activation/Désactivation de l'effet de clignotement aléatoire des particules
- Choix de la textures des particules
- Choix de la taille des particules
- Choix de la couleur des particules
- Choix du niveau de définition du nuage de points
- Choix du type de blending, effet d'accumulation lumineuse

Les trois dernières options permettent au sein d'un nuage de points de mettre en valeur une information.
Sur la base d'une caractéristique des particules, tel que l'age ou la masse, on interpole les valeurs et l'on modifie un effet visuel en conséquence.
Par exemple, on peut choisir d'identifier l'age via la couleur.
Les particules les plus anciennes seront ainsi rouge pour basculer progressivement vers du bleu pour les plus jeunes.

Voici les options disponibles pour produire ces effets.

- Choix entre une interpolation linéaire ou logarithmique des informations.
- Choix de la manière de mettre en valeur l'information : couleur, taille, intensité lumineuse, vitesse de clignotement.

– Choix de l'information à mettre en valeur.

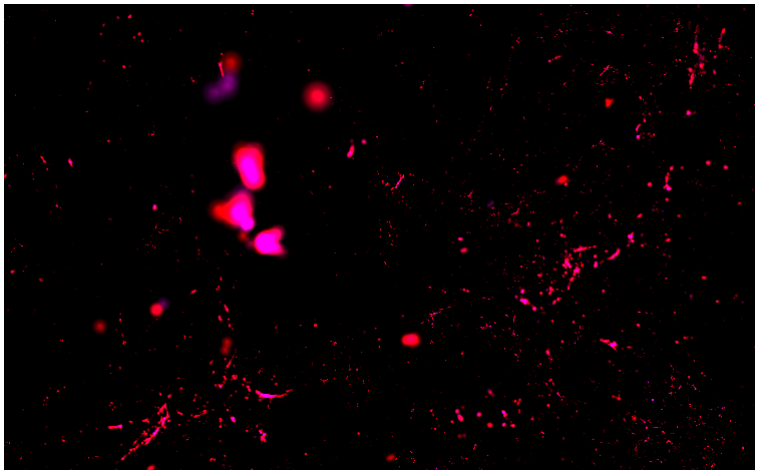


FIGURE 9 – Interpolation de couleur d'étoile par leur age

4.5 L'animation

Timeline Lorsque plus d'un *Snapshot* est disponible pour une même donnée, il est possible de naviguer dans le temps entre les deux en interpolant les positions de départ et d'arrivée.
Une *timeline* permet une navigation aisée, même lorsque que l'on dispose de plus de deux *Snapshots*.

FIGURE 10 – Timeline

4.6 Usage

L'application a pour vocation de servir d'interface de visualisation et de démo.
Ainsi, pour ne pas se restreindre à une page web spécifique à l'application, et en tirant profit des possibilités du web, l'application se présente sous forme d'un *plugin*, ou d'une simple balise web, pouvant être insérée dans n'importe quelle page web déjà existante.

4.7 Résultats

Sur une machine de puissance correct, comme celle utilisée lors de développement, l'affichage de deux millions de particules s'effectue à 60fps, ou images par secondes, vitesse de rafraichissement maximum souhaitée.
L'ajout de plusieurs autres cube de données, l'utilisation du multi-vues et autres

fonctionnalités gourmandes restent possible tout en oscillant entre 30 et 60fps, 30fps offrant une expérience fluide et agréable.
On peut observer une baisse dommageable des performances lorsque l'on tente d'afficher plus de 15 millions de points, pour un blocage complet de l'application au dessus de 30 millions.
Néanmoins pour ces valeurs l'application peut aussi s'arrêter lors du chargement des fichiers, la mémoire temporaire utilisée pour les opérations dépassant les capacités du navigateur.
Si l'on cherche sur le web les exemples de telles applications, les plus performantes affichent correctement 1 million de points, on peut donc affirmer que le travail sur les performances à permis de nettes améliorations.

Deuxième partie

Recherche

Dans un premier temps le stage s'est focalisé sur de la recherche autour du sujet et des outils utilisables, pour garantir un développement efficace.
Par la suite, il a bien sur été nécessaire d'entrecouper le développement de l'application par de nouvelles recherches dès lors que survenait plusieurs problèmes qui n'avaient pas été anticipés, mais le gros du travail avait été fait et est présenté dans cette partie.

5 THREE.js

Le domaine principal du stage étant la visualisation 3D, la technologie la plus importante utilisée est WebGL, API permettant l'affichage d'objet graphique 3D en temps réel dans la fenêtre du navigateur. Pourtant, WebGL reste une API assez complexe à utiliser en l'état. L'affichage d'une forme simple à l'écran demande un processus complexe et de nombreux appels de fonctions.
Pour remédier à cela, il existe actuellement deux bibliothèques de plus haut niveau qui encapsulent toute cette logique d'affichage : Babylone.js et THREE.js.
Babylone.js se présente comme un moteur de jeu, et propose divers contrôles sur la physique des objets en plus du rendus graphique.
Le choix s'est pourtant porté sur THREE.js, qui bien que proposant moins de fonctionnalités, semble permettre une interaction plus bas niveau, pour au final une liberté plus grande dans le choix des techniques de rendus.
Le développement de THREE.js a été initialisée par MrDoob, et la bibliothèque est depuis largement enrichi chaque jour par de nombreux collaborateurs.
Entièrement Open Source, elle dispose d'une documentation fournit et de nombreuses pages de discussions sur les principaux forums d'aides.

6 Rendu 3D

Le rendu d'image de synthèse à l'écran nécessite la mise en place de plusieurs processus qui se retrouvent dans tous les moteurs graphique tel que THREE.js.

Données A l'origine du rendu se trouve les données brutes des objets ou informations que l'on souhaite afficher à l'écran.
Ce sont généralement des ensembles de positions dans l'espace 3D, ainsi que des informations sur la couleur ou des coordonnées de textures.
Il existe nombre de moyen de stocker ces informations, par exemple dans des *buffers*, large tableau d'octets.

Scène La scène est l'ensemble des objets - forme géométrique - dont on souhaite effectuer le rendus.
Habituellement, et c'est le cas pour THREE.js, la scène propose de hiérarchiser les objets pour établir des relations de parentés entre les différents objets.
Un objet ne sera rendu à l'écran que si il appartient à une scène.

Camera Pour rendre compte du point de vue souhaité sur la scène, on symbolise une caméra par sa position et orientation dans l'espace.
Cela détermine exactement l'endroit d'où sera rendus la scène, et l'image final affichée à l'écran.

Pipeline de rendus Le processus de rendu peut alors se schématiser de la sorte :

- Les données sont stockées dans les objets
- Les objets sont intégrés dans une scène
- La position des objets, leurs caractéristiques ainsi que les informations sur la caméra sont envoyés à la carte graphique
- Le GPU, pour *Graphics Processing Unit*, par le biais de courts programmes appelés *Shader* va traiter chaque points récupérés pour déterminer leurs positions dans l'espace 2D de l'écran, et les pixels correspondant à colorier.
- L'image final est renvoyée et est affichée à l'écran.

Ce schéma fait abstraction de bon nombre de mécanismes mis en œuvre pour arriver à l'image final, mais est suffisant pour voir apparaitre certains aspects critiques pour les performances.
Il faut garder à l'esprit que ce processus se répète entre 30 et 60 fois par secondes selon la cadence d'affichage souhaitée, pour avoir une application fluide.
Cela revient à un laps de temps de l'ordre de la quinzaine de millisecondes accordé à la carte graphique pour calculer l'image finale.

Calcul vectoriel Avant d'aborder les problèmes récurrents lorsque l'on touche au rendu graphique, il est nécessaire de comprendre le fonctionnement en surface de la carte graphique, et pourquoi elle est complémentaire au processeur de l'ordinateur.
Dans une carte graphique les calculs sont hautement parallélisés, et un GPU excelle dans tous ce qui est calcul vectoriel, que l'on appel communément calcul de type *SIMD* pour *Single Instruction, Multiple Data*. Plutôt que d'effectuer les mêmes opérations sur chaque données une par une comme le ferait le CPU, l'opération est réalisée une fois sur un très grand vecteur de données.
Le GPU est donc idéal pour effectuer énormément de calcul identique sur des données différentes, en très peu de temps.
C'est pour quoi cette architecture est adaptée au rendu graphique, puisque les données récupérées subissent par exemple toutes la même opération matriciel correspondant à la transformation dans l'espace et à la projection dans l'espace de la caméra.

7 Goulot d'étranglement

Explication du problème Le CPU et le GPU ne partagent pas la même mémoire, et les informations côté CPU doivent être transférés au niveau du CPU. Cette copie d'information impacte grandement le temps de rendus de l'image.
Il est donc nécessaire de réduire au maximum les données à transférer sur le GPU. Éviter la redondance d'information, ne pas envoyer d'informations inutiles, tous cela améliore la rapidité d'exécution du GPU.

Partage des calculs Durant le stage, la question qui s'est posé régulièrement était le partage des calculs entre le CPU et le GPU.
Si l'on effectue des calculs à l'avance côté CPU, il y aura moins d'informations à envoyer au GPU. Par contre, ces calculs prennent plus de temps à se finir puisque que l'on n'exploite plus le processeur graphique.
Il est finalement apparus que dans le cas de notre application, il fallait privilégier la vitesse de calcul du GPU devant le problème du goulot d'étranglement.

persistance de l'information Pourtant, il a été nécessaire d'effectuer certains calculs côté CPU alors que le GPU était effectivement plus aptes à traiter ces données, pour la seul raison qu'on ne peut récupérer facilement le résultat des calculs effectués par le GPU.
Pour faire le lien avec un élément concret de l'application, on souhaite interpoler deux cubes de données pour obtenir la position des particules à un instant t quelconque.
On ne dispose dans la mémoire du CPU que de la position des particules à l'instant 0 et à l'instant final.
Le calcul d'interpolation à l'instant t quelconque est très bien géré par le GPU, mais le CPU lui n'a pas connaissances du résultats de l'interpolation.
Les positions affichées à l'écran pour l'utilisateur ne sont pas connus par l'application, ce qui pose des problèmes notamment si l'utilisateur souhaite sélectionner les particules qu'il voie dans l'outil de visualisation.
Il a donc été nécessaire dans certains cas d'effectuer ce calcul au niveau du CPU.
Il existe plusieurs optimisations du code qui permettent de diminuer l'impact de ces calculs sur les performances. Celles-ci seront abordés dans la partie suivante.

8 Autres améliorations

Type d'objet Les informations sur les particules - positions, age, masse, etc - sont stockées dans de large tableau appelés *geometry buffer*. Ces *buffers* sont ensuite envoyés en tant que données brutes à la carte graphique pour leur traitement.
THREE.js dispose de l'objet *PointCloud* qui permet de manipuler un nuage de points.
On aurait pu aussi considérer chaque point comme un objet unique, et demander l'affichage successif de tous les points de la scène.
Cette technique à l'avantage d'être facilement représentable et permet nombres d'optimisations déjà intégrées dans THREE.js, tel que l'occlusion d'objet ou le

picking, dont il sera question dans la section suivante.
Pourtant, il faut effectuer un appel au pipeline de rendus pour chaque point/objet, et l'impact sur les performances de l'application est conséquents.
Alors qu'il n'est possible d'afficher qu'une centaines de points via cette techniques, l'usage des *buffers* rend possible l'affichage en temps réel de plus de 2 millions de particules sans ralentissements.
Dès lors, l'implémentation des *geometry buffer* et des *point cloud* ne proposant pas beaucoup d'autres options que l'affichage d'un nuage de points basique, certaines fonctionnalités ont du être pensée et rajoutée à la main, de même que certaines optimisations ont du être abandonnées.

Entrelacement des données Si l'utilisation de THREE.js a soulagé grandement la construction de l'application, la librairie n'a pas permis certaine optimisations, tel que l'entrelacement des données.
On souhaite envoyer pour chaque particules à la fois ses coordonnées, sa couleur et son age. La technique retenu à l'issue du stage est de stocker les coordonnées dans un *buffer*, les couleurs dans un autre et les ages des particules dans un troisième.
La carte graphique n'a qu'à se référer à l'index des tableaux pour associer entre elles les bonnes informations.
Néanmoins, il aurait pu être plus habile de stocker dans un seul tableau les trois informations à la suite, c'est à dire une suite de triplés coordonnées, couleur et age pour chaque particule. Cette technique évite en effet à la carte graphique de faire des saut dans la mémoire pour chercher les bonnes données.
Pourtant, THREE.js ne permet pas en l'état ce type de manipulation via l'utilisation des *pointcloud*, et certains gains en performance ont donc été sacrifiés.

9 *Octree*

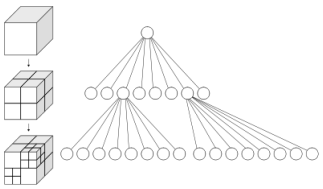


FIGURE 11 – *Octree*

octree Un *octree* est une structure d'arbre 8-aire. Il permet une partition de l'espace de manière récursive.
Dans la pratique, chaque nœud de l'arbre délimite une partie de l'espace prenant forme d'un cube. Ses fils sont alors parmi ses huit sous-cubes, ou *octant*.
Un nœud sans fils est appelé feuille.
Si la racine de l'*octree* englobe toute la scène, chaque élément se trouve donc dans l'une - voir plus - des feuilles.
Cette structure comporte de nombreux avantages en terme de performance et de rapidité d'exécution, moyennant sa construction préalable.

Plusieurs définitions sont nécessaire pour comprendre son rôle dans l'application.

9.1 Définitions

***Frustum* de camera** Pour déterminer la partie de la scène à afficher, on symbolise une camera disposant d'un cone de vue, ou *frustum*.
Ce dernier se compose de 6 plans, qui délimitent la partie de l'espace qui sera visible à l'écran, et la manière dont seront projetées les coordonnées 3D sur le plan 2D qu'est la fenêtre du navigateur.

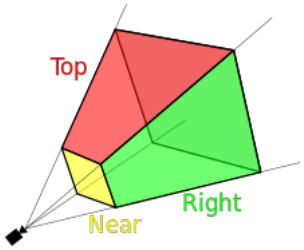


FIGURE 12 – View frustum

AABB Une boite d'encombrement, ou *Bounding box* en anglais, d'un objet ou ensemble d'objet est un volume fermé qui contient l'union du ou des objets. On appel AABB pour *axis-aligned bounding box* les cas particuliers ou la boite est un parallélépipède dont les côtés sont parallèles aux axes du repaire de la scène.

Frustum culling La technique de *View frustum culling*, ou élimination des objets hors du cône de vue, consiste à déterminer préalablement les objets ou parties d'objets qui ne sont pas dans le champ de vue de la caméra, pour faire l'économie de nombreux calculs puisqu'ils n'apparaitrons pas à l'écran de toute façon.

Bounding-box Pour éviter d'avoir à tester un par un les points des objets pour savoir si oui ou non ils se trouvent dans le cône de vision, on peut recourir à la *bounding-box* de l'objet et ne tester que ses sommets.
Après avoir déterminé pour chaque sommet si il se trouve à l'intérieur ou à l'extérieur du cône, on peut juger si l'objet est en dehors du cône, complètement compris dans le cône, ou partiellement visible.

9.2 utilisation

Initialisation Les objets constituant la scène se trouvent déjà à l'intérieur d'une cube.
La racine de l'*octree* est donc tout naturellement fixée.
Il suffit ensuite d'itérer sur les huit sous-cubes et de recommencer le processus.

Chaque nœud de l'*octree* donne alors les informations nécessaires pour accéder aux points compris dans le cube.
Il est évidant qu'il faille définir un cas d'arrêt à la construction de l'*octree*.
Le meilleur choix semble un compromis entre un nombre d'itération maximum - qui définit alors une profondeur maximum de l'arbre - et un nombre maximum de points compris dans une feuille.
La deuxième solution offre des résultats beaucoup plus souhaitables, car l'*octree* sera alors plus précis autour des parties denses du cube de données.
Couplé à une limite dans le nombre d'itération, cela évite des calculs trop long du à la complexité exponentiel de la construction de l'arbre.

Occlusion Une fois cette structure à portée de main, il est possible de contrôler le nombre de particules affichées à l'écran.
Plutôt que d'effectuer un algorithme de *frustum culling* sur chaque élément du nuage de points, on applique la formule à l'*octree*.
On obtient l'algorithme suivant :

Data: Un octree
Result: Une liste de particules à afficher
if *La bounding box de l'octree est en dehors du frustum* **then**
| Fin de l'algorithme
else if *La bounding box de l'octree est partiellement en dehors du frustum* **then**
| **for** *octan* $\in$ *octree* **do**
| | Appliquer l'algorithme à *octan*
| **end**
else
| Ajouter les points de l'octree à la liste des particules à afficher
end

Algorithm 1: Algorithme d'occlusion avec Octree

De cette manière seules les particules visibles à l'écran seront envoyées à la carte graphique.

9.3 *Ray casting*

Le *Ray Casting*, ou lancer de rayon, consiste dans l'application à déterminer l'intersection avec les particules d'un rayon imaginaire partant du pointeur de la souris.
L'utilisateur doit être capable de sélectionner à la souris avec une bonne précision une particule du nuage de point.
THREE.js propose une implémentation du *Ray Casting*, mais celui effectue un test pour chaque particules du nuage de point pour savoir si celle-ci est ciblée ou non.
Le calcul est assez long et impacte grandement le nombres d'images par seconde.
Il a donc été décidé de réécrire un algorithme de *Ray Casting*, prenant cette fois en compte la structure d'*octree*.
L'idée reste la même, il suffit de tester en premier lieu les cubes pour ensuite affiner la recherche et ne tester finalement qu'une parties des particules.

Supposons avoir transformé la position de la souris en un rayon traversant la scène dans l'espace 3D.
l'algorithme suivant déterminera pour un cube de données les feuille de l'*octree*, i.e. les cubes ne possédant pas de fils, traversé par le rayon, et ceux ci triés suivant la distance à l'origine du rayon.

Data: Un *octree*, le rayon
Result: Un tableau d'*octants*
if *l'octree possède des fils* **then**
  for *octan* ∈ *octree* **do**
    Appliquer l'algorithme à *octan*
  end
else
  Ajouter l'*octant* au tableau des *octants* traversés par le rayon
end

Algorithm 2: Algorithme de *Ray Casting* avec *Octree*

Il suffit ensuite d'effectuer le vrai test d'intersection avec chaque particule de chaque octant détecté.
On a pu observer un gain d'environ quinze image par seconde si l'on privilégie cette méthode plutôt que la technique implémentée dans THREE.js.

9.4 Niveau de détail

Pour des données tels que la répartition de matière noir, il est apparu dans certains cas qu'afficher toutes les particules n'est pas nécessaire.
Ne prendre qu'une partie des points, si ceux-ci sont choisit judicieusement, rends compte des mêmes informations pour un nombre réduit de points à afficher.
Puisque l'*octree* propose déjà un tris des particules suivant leur position dans l'espace, ne prendre qu'un point sur deux voir moins permet de réduire uniformément le nombres de particules.

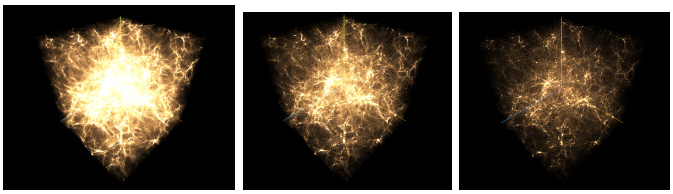


FIGURE 13 – De gauche à droite, tous les points, trois points sur quatre et un point sur deux

10 Oculus Rift

L'Oculus L'Oculus Rift est un casque de réalité virtuelle conçu par l'entreprise oculus VR, une filiale de Facebook.
Il existe actuellement deux kits de développements, le SDK1 et le SDK2.
Ces deux kit étaient disponibles lors du stage et l'on pouvait tester facilement

les démos natives.

Principe L'Oculus est constitué d'un écran numérique et de deux lentilles focalisées sur l'écran, de manière à former une image à l'infini.
L'écran propose deux images de la même scène, mais avec un angle de caméra légèrement différent, chaque partie de l'écran étant projetée sur l'un des deux yeux de l'utilisateur pour créer le sentiment d'immersion.

Correction La projection par les lentilles déforme d'une part l'image, et rajoute d'autre part des aberrations chromatiques qui irisent les contours des objets.
l'image d'origine doit donc être modifié en conséquence, en appliquant premièrement une déformation inverse, puis des effets pour inverser l'aberration chromatique.

Troisième partie
Développement

11 outils

11.1 matériels

FIGURE 14 – Ubuntu

L'intégralité du développement s'est effectué sur une machine mise à disposition par le CDS.
Le système d'exploitation est Ubuntu dans sa dernière version 14, et les spécifications du moniteur étaient à la hauteur des besoin pour tous ce qui touche au rendu graphique et au calcul en temps réel :

- 16Go de RAM
- Carte graphique dédiée

11.2 langages

FIGURE 15 – HTML, CSS & Javascript

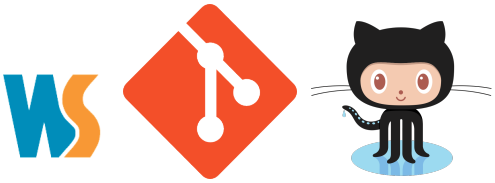
Puisque le sujet de stage avait une orientation fortement web, c'est sur le couple HTML5 et CSS3 que s'est basée l'application.
Le standard HTML5 permet désormais une grande liberté d'interaction avec le DOM, ainsi que la possibilité de faire appel aux fonctionnalités WebGL, le pendant du standard OpenGL sur navigateur Web.
Bien sur, pour afficher et contrôler une application tirant profit de l'accélération matériel et de l'affichage WebGL, c'est finalement le Javascript qui a été le langage le plus utilisé lors de ce stage. Ce dernier nous permet de gérer l'affichage et de modifier à la volée le HTML et la mise en page de notre page web, de traiter nos données, de gérer la logique de l'application et l'interaction avec l'utilisateur, ainsi que d'utiliser l'API WebGL.
Bien qu'ayant une syntaxe proche du C et des langages procéduraux en général, le Javascript ne dispose pas de compilateur, ce qui permet son exécution rapide sur tout navigateur, quelque soit l'environnement de l'utilisateur.

11.3 Librairie

En complément de THREE.js, d'autres librairies plus minimes ont été rajoutées au fil du temps pour combler les besoins :

- FirstPersonControls.js : Permet d'intégrer à THREE.js une caméra dirigeable à la première personne.
- OculusRiftEffet.js & StereoEffect.js : Permettent respectivement de diffuser des vues oculus et google cardboard.
- async.js : Permet d'effectuer des instructions en parallèle, en gardant une trace des résultats et de l'avancement des opérations.
- dat.gui.js : Propose une interface utilisateur facilement paramétrable pour modifier dynamiquement les variables de l'application.
- stats.min.js : Propose un élément html rendant compte de la vitesse d'affichage des éléments graphiques, tels que les fps.

11.4 Logiciel



L'écriture du code s'est effectué sous Webstorm, un IDE proposé par JetBRAINS adapté au langages web, permettant de gérer l'architecture d'un projet mêlant HTML, CSS et Javascript.
l'IDE propose en plus le déploiement d'un serveur local, pour faire tourner l'application dans un navigateur web.
La gestion du projet à nécessitée l'utilisation de Git, gestionnaire de version qui a permis le travail à deux sur le même projet.
En complément, l'utilisation de GitHub à permit de partager et d'échanger rapidement les sources entre les deux personnes travaillant sur le projet.

12 Gestion des données

12.1 Chargement

Le chargement des données utilise l'API *FileReader* , qui offre tout le nécessaire pour ouvrir un explorateur de document standard et récupérer le contenu des fichiers choisis par l'utilisateur.
L'API permet par divers callback de suivre la progression du chargement des fichiers, et de gérer les erreurs possibles.
Une fois le chargement terminé, on dispose d'un objet comprenant le résultat de la recherche, et il n'y a plus qu'à l'envoyer au script.

12.2 Script

Le script est une méthode dont l'entrée et la sortie sont fixés.
L'utilisateur peut écrire lui même le script qu'il souhaite et le rendre disponnible dans l'application facilement pour peu qu'il respecte ces deux contraintes :

- La méthode attend en entrée soit un *buffer* soit une chaine de caractère, qui correspond au contenu du ou des fichiers.
- la méthode doit renvoyer un objet comportant dans l'ordre :
  - Un attribut *index* ayant pour valeur le tableau des indexes
  - Un attribut *position* ayant pour valeur le tableau des positions
  - Un attribut *color* ayant pour valeur le tableau des couleurs
  - Pour chaque autres attributs optionnels, le nom de l'attribut, par exemple *age*, avec pour valeur le tableau correspondant.

12.3 Stockage des données

Type Ces tableaux de données doivent être au format Javascript *Float32Array* qui s'apparente au type *float* du C et est ainsi adapté aux valeurs manipulées. C'est aussi le format utilisé par THREE.js et qui ne nécessite pas de conversion complexe pour être transféré à la carte graphique.

Snapshot un première objet *Snapshot* va conserver toutes les informations relatives à un cube de données, donc au résultat d'un chargement de fichier.
De même, on garde en mémoire dans cet objet l'*octree* correspondant à ce cube de données.

Data Au dessus se trouve l'objet *Data*, qui lui va premièrement stocker les différents *Snapshot* d'un même type de donnée.
Il va ensuite garder en mémoire le *Snapshot* courant, et le suivant si il existe, pour pouvoir faire l'interpolation des positions dans le temps.
Il permet finalement d'accéder au temps relatif de l'interpolation et à l'*octree* correspondant au *Snapshot* courant.

Ces objets Data sont communs à toutes les vues, i.e. il n'existe qu'une seul instance pour chaque type de donnée utilisé dans l'application.

RenderableData Finalement, l'objet renderableData comporte toutes la logique d'affichage d'un type de donnée. En somme, c'est l'objet *pointcloud* de THREE.js qui est associé à un objet *Data*.
par contre, cette fois ci les instances sont multiples si les vues le sont aussi, car chaque vue gère de façon indépendante les caractéristiques et les méthodes de rendus des données disponibles.

13 Performance javascript

13.1 Timed array processing

Calculer les nouvelles positions des particules du côté du CPU peut prendre quelques secondes, pendant lesquelles l'utilisateur ne peut pas interagir avec

l'application.
Les commandes ne répondent plus et cela impact grandement la fluidité de l'application et l'expérience utilisateur.
Cela est du au fait que le Javascript effectue les instructions de manière synchrone et sans parallélisme.

Pour offrir à l'utilisateur une expérience plaisante sans figer l'application jusqu'à la fin du calcul, on peut alors faire en sorte de redonner temporairement le contrôle à l'affichage à intervalles réguliers, pour que s'effectue un rafraichissement de l'écran.

Il suffit de sauvegarder l'index de la particule que l'on vient de traiter, et de programmer un appel à la fonction dans les quelques millisecondes qui suivent. Cela suffit au navigateur pour prendre le temps de traiter les commandes de l'utilisateur et les animations à l'écran.
La fonction reprend ensuite la ou elle en était, et le processus se répète jusqu'à ce que toutes les données aient été traitées.

13.2 async.js

Async.js est une librairie Javascript qui permet d'effectuer entre autres une suite d'instruction de manière asynchrone.
Elle est utilisée pour le chargement des fichiers.
Puisqu'on dispose de plusieurs petits fichier, il est très intéressant d'effectuer la lecture des données pour chaque fichier en parallèle et de rassembler les résultats une fois chaque fichier traité.
Async.js propose des méthodes permettant de réaliser ceci très facilement, et assure un contrôle complet sur l'exécution de chaque lecture de fichier.

13.3 WebWorker

Definition Les *WebWorkers* permettent de paralléliser l'application, c'est à dire créer un nouvelle environnement Javascript qui effectuera des opérations en parallèles de l'application principale.
Les *webworkers* permettent ainsi de séparer différentes tâche que l'on aimerais mener à bien en même temps, sans avoir à les effectuer à la suite ce qui prendrais trop de temps.

Utilisation Le calcul de l'*octree* peut être très couteux en temps si les particules sont nombreuses ou si l'on souhaite avoir une très grande précision, donc un gain de performance par la suite.
La encore cela impactera le temps de réponse de l'application et le confort de l'utilisateur.
En effectuant la construction de l'*octree* au sein d'un *WebWorker*, l'application peut fonctionner pendant ce temps tout à fait normalement, et avertir l'utilisateur lorsque l'*octree* est enfin complet.

Implémentation Le *Thread* principale de l'application fait appel à un nouveau *WebWorker* en chargeant un fichier Javascript externe.
Dès lors, le programme Javascript contenu dans ce fichier mène une vie propre, dans un espace de variable et de mémoire dissocié du code principal.
Les deux *Thread* communique par envois de message.
Dans la pratique, on envoi les positions des particules au *WebWorker*. Celui ci va travailler dessus de son côté dès réception du message.
Lorsque les calculs sont finis, un message est renvoyé au *Thread* principal qui jusque la avait reprit son activité normal.
l'*octree* est alors récupéré est peut être utilisé dans l'application. le *WebWorker* est tué et le parallélisme prend fin.

13.4 Optimisation

Il existe plusieurs méthodes d'optimisation du Javascript en lui même, qui augmentent sensiblement les performances lors de calculs et d'opérations fastidieuses et répétitives.

Variable local Très vite, il à été nécessaire de supprimer toutes utilisations de variables global dans l'application.
Bien que celles ci offrent une utilisation plus simple puisqu'elles sont directement accessible dans l'ensemble de l'application, elles impactent les performances car elles font appel à divers espace de noms volumineux et leur emplacement en mémoire ne favorise pas un accès rapide.
Des variables local aux fonctions ont donc été privilégiés, quitte à copier en local dans un méthode une variable plus générale, car alors le compilateur optimise l'accès en mémoire à ces variables.

Chainage d'attributs Lorsque l'on s'intéresse au Javacript orienté objet, il est souvent nécessaire d'appeler en cascade plusieurs objets à travers différentes classe.
Si l'on souhaite accéder à cet élément plusieurs fois à la suite, voir des milliers de fois dans le cas de boucles sur tous les éléments du nuage de points, il est conseillé de stocker la référence à l'avance pour ne pas refaire un appel au chainage à chaque fois.

boucle Le choix des boucles à aussi un impact sur les performances.
Cela dépend du navigateur et de sa version, mais dans l'ensemble certaines écriture sont à privilégier, car elles font appel à moins de variables et à des opérations plus rapide à traiter.
Bien sur, il faut toujours faire un compromis entre l'efficacité du code et sa lisibilité, mais de simple écriture peuvent par exemple améliorer le parcours d'un tableau :

```
/* A éviter */
var n = 100;
while (array.length < n) array += x;

/* A privilégier */
var n = array.length;
```

```
while (n-- > 0) array += x;
```

Les *closures* Les *closures*, ou fermetures, sont des fonctions qui utilisent des variables libres définis en dehors.
les *closures* se souviennent de l'environnement dans lequel elles ont été crées.
En voici un exemple très simple :

```
function setupAlertTimeout() {
  var msg = 'Message to alert';
  window.setTimeout(function() { alert(msg); }, 100);
}
```

la fonction utiliser en argument de *setTimeout* se souvient de la valeur de msg.
Très utiles dans la pratique, elles peuvent engendrer de nombreuses fuites mémoires et ralentissent l'application. Elles sont donc à éviter le plus possible.

14 Interface

14.1 DatGui

DatGui est une petite librairie Javascript proposant une interface personnalisable et permettant de modifier facilement les variables de l'application.
Si l'on souhaite que l'utilisateur puisse modifier via l'interface de l'application un paramètre, on peut donc lier une variable à un élément de DatGui.
Plusieurs type de modificateurs sont proposés :

- Interrupteur pour modifier une valeur booléenne
- Liste déroulante pour choisir une option
- Curseur pour choisir une valeur dans une gamme continue

Les éléments de l'interface proposent aussi de rendre compte des modifications internes des variables.
Si une variable change de valeur sans que l'utilisateur en sois l'origine, l'interface changera pour s'adapter à cette nouvelle valeur.
Néanmoins, cette spécificité nécessite une vérification au niveau de la logique de DatGui à intervalles réguliers des valeurs des variables.
Cela peut ralentir l'application car DatGui sollicite alors régulièrement le système.

14.2 CSS

Certains éléments de l'interface tels que le menu de gestion des données ont étés réalisé en CSS pur.
Pourtant, l'on ne pouvait plus se permettre d'écrire le CSS dans un fichier à part, comme le veut l'usage, pour la seule et unique raison que l'application doit pouvoir être importée facilement dans un autre site web.
L'utilisateur devrait avoir seulement les sources Javascript à intégrer dans son site pour utiliser l'interface.
il a donc été nécessaire de créer le CSS à même le code Javascript, ce qui rend la gestion de l'ergonomie de l'interface plus ardus, prend plus de place en mémoire, mais permet une utilisation plus simple de l'application.

15 Visualisation

15.1 *Shader*

Les *shaders* sont de court programme à la syntaxe proche de C, qui permettent de modifier la manière dont la carte graphique va traiter les données qu'on lui envoi, et ajouter ainsi effet, lumière, textures à la simple visualisation dans l'espace des points.
pour simplifier, nous considérerons seulement deux types de *shader* :

- Le *shader* de sommet : Ce *shader* va récupérer chaque points envoyés par le CPU, et va calculer la position effective du sommet dans l'espace via les matrices de transformations et la matrice de projection de la camera.
- Le *shader* de fragment : Ce *shader* va ensuite appliquer à ces points leur couleur final, leur texture, et d'autre informations liées à l'apparence des particules.

La encore, ce processus est hautement simplifié mais suffit à la compréhension de l'implémentation des *shaders*.
THREE.js propose de base des *shader* déjà écris pour produire automatiquement divers effet.
néanmoins pour les besoins de l'application il a été nécessaire d'écrire soit même une partie de la logique des *shader*, pour réaliser toutes sortes d'effet souhaités.

15.2 Animation

Pour réaliser l'animation des particules au niveau du GPU, le *shader* de sommet est très utile.
A partir des positions de deux cubes de données, on peut calculer la position interpolée suivant le temps :

```
//Interpolation entre la position de départ et la direction de la particule
vec3 pos = departure+direction*t;
//Calcul de la position dans le repère de la caméra
mvPosition = modelViewMatrix*vec4(pos, 1.0);
```

15.3 Interpolation

Dans le même ordre d'idée, on peut alors interpoler n'importe quelle information pour déterminer l'une ou l'autre des caractéristique du point , cette fois ci dans le *shader* de fragment :

```
//Calcul du facteur d'information entre les valeur extrêmes
factor =  (minInformation - information)/(minInformation - maxInformation);
//Application à la couleur de la particule
color_out = color*factor;
```

16 Itération

Reprise de l'existant La première étape du développement consistait en la compréhension puis la re-factorisation du code écrit par Arnaud Steinmetz lors de son stage.

Le code était fragmenté en plusieurs petit projet de test, et tous on du être assemblé en un seul projet.

Seconde Itération une seconde itération a consisté en la suppression de la plupart des variables globales du code, ainsi qu'une séparation de la logique d'affichage et de celle liée aux données.
le code était alors plus lisible et compréhensible.

Orientation objet Finalement, devant l'ampleur du projet et la nécessité de travailler à deux dessus, c'est vers une logique objet que le développement s'est orienté.
Toutes la logique à été remanié sous formes de classes distinctes et plus aucunes variables globales n'a été utilisé.

Documentation une documentation a de plus été réalisée à l'aide de JSDoc, qui permet grâce à une syntaxe de commentaire précise non seulement de documenter les fichiers sources et les classes, mais aussi de générer automatiquement une documentation complète accessible sur navigateur web.

Conclusion

Ce stage m'a permis de consolider mes compétences dans le domaine de l'imagerie informatique.
La dynamique R&D a pu me proposer un bon compromis entre réflexion et développement pur.
Ce stage a confirmé mon intérêt pour l'imagerie et les nouvelles technologies.

Webographie

Exemples
- Potree, une utilisation des octree : `http://potree.org/wp/`
- Octree avec THREE.js : `http://threejs.org/examples/#webgl_octree_raycasting`

Rendu 3D
- Doc THREE.js : `http://threejs.org/docs/#Manual/Introduction/Creating_a_scene`

Javascript
- Timed array processing : `https://www.nczonline.net/blog/2009/08/11/timed-array-processing-in-javascript/`
- Speed comparaison : `http://jsperf.com/new-array-vs-splice-vs-slice/79`

– Outil de profilage : http://learningthreejs.com/blog/2013/04/05/debugging-with-chromes-canvas-inspection/
– Performance : http://www.webreference.com/programming/javascript/jkm3/index.html