BUECHER Nicolas
Élève ingénieur en informatique

RAPPORT DE STAGE

DEUXIEME ANNEE

Période du 10 juin au 18 août 2015

Visualisation de données astronomiques et interactions

Maître de stage :

M. André Schaaff
Ingénieur de recherche CNRS

CNRS
Observatoire astronomique de Strasbourg
11 Rue de l'Université
67000 STRASBOURG
FRANCE

ENSIIE Strasbourg
Bâtiment de l'ISU
1 Rue Jean-Dominique Cassini
67400 ILLKIRCH-GRAFFENSTADEN
FRANCE

REMERCIEMENTS

J'aimerais remercier l'ensemble du personnel de l'Observatoire astronomique de Strasbourg pour m'avoir permis de m'intégrer dans les meilleures conditions, et plus particulièrement, j'aimerais remercier les membres du CDS qui m'ont accompagné tout au long de mon stage.

Je tiens également à remercier tout particulièrement M. André SCHAAFF, mon maître de stage, pour avoir été constamment présent, à l'écoute et de très bon conseil. Il a su justement doser la balance entre liberté d'action et accompagnement, afin de me faire progresser.

Enfin, j'aimerais remercier M. Hervé WOZNIAK, directeur de l'Observatoire, pour m'avoir accueilli dans son établissement.

INTRODUCTION

Je suis actuellement élève ingénieur en informatique en deuxième année à l'Ecole Nationale Supérieure d'Informatique pour l'Industrie et l'Entreprise (ENSIIE) de Strasbourg.

Le diplôme d'ingénieur se prépare sur trois ans, à bac +2. La formation associe des connaissances générales sur l'entrepreneuriat, la communication, l'économie, la comptabilité et les mathématiques à des connaissances techniques en informatique. Elle comprend un séjour d'au moins deux mois à l'étranger ainsi que trois stages en entreprise de respectivement huit, dix et vingt-six semaines.

J'ai effectué mon stage du 10 juin au 18 août 2015 en intégrant les équipes du CDS de l'Observatoire astronomique de Strasbourg, UMR du CNRS et de l'Université de Strasbourg, situé au 11 rue de l'Université, 67000 Strasbourg.

L'Observatoire astronomique de Strasbourg, et plus particulièrement le CDS, a en charge l'acquisition de données d'observation, la maintenance des bases de données ainsi que la mise à disposition de services permettant la distribution et l'exploitation des données.

L'effectif moyen de l'Observatoire est de 80 personnes, dont une trentaine travaille au CDS. Parmi ceux-ci, on dénombre un tiers de documentalistes, un tiers d'astronomes et un tiers d'ingénieurs informaticiens.

Durant ces dix semaines de stage, j'ai eu en charge la conception et le développement d'une application de visualisation de données astronomiques avec un autre étudiant. Cela m'a permis d'appréhender de nouvelles technologies très prometteuses, de découvrir le monde de la recherche ainsi que le partage des tâches en équipe. Second stage en entreprise, celui-ci se différenciera notamment du précédent par l'aspect R&D du travail effectué qui m'a octroyé un nouveau degré de liberté.

SOMMAIRE

I. CONTEXTE DU STAGE

I.1. Historique de l'entreprise

I.1.a. Du CNRS…

Le 19 octobre 1939, le Président de la République Française Albert Lebrun acheva la création du Centre National de la Recherche Scientifique (CNRS) par la signature d'un décret-loi. A cette époque, le CNRS eut essentiellement pour but de réunir les moyens de recherche disponibles et de concentrer les recherches sur des solutions militaires tout en mettant les savants à l'abri du conflit armé.

A la fin de la Seconde Guerre Mondiale, le CNRS fut réorganisé et commença alors à concentrer ses efforts sur la recherche fondamentale.

Figure 1 - Logo CNRS

C'est à partir de 1966 que se met en place l'idée d'unités associées, qui ne sont autres que des laboratoires universitaires liés par contrat au CNRS. Ce dernier les soutiendra grâce à ses moyens humains et financiers.

Au fil des ans, la notion d'unité associée évoluera et les structures opérationnelles de recherche se diversifieront. Parmi elles, les unités mixtes de recherche (UMR) émergent vers les années 1990. Elles représentent l'association contractuelle d'un ou plusieurs laboratoires de recherche d'un établissement d'enseignement supérieur ou d'un organisme de recherche avec le CNRS.

Une UMR dispose d'un budget qui lui est propre, d'un personnel affecté à la fois par le CNRS et l'établissement d'enseignement supérieur (ainsi que des contractuels) et est administré par un directeur et un conseil de laboratoire qui définit sa stratégie de recherche de manière autonome.

I.1.b. … A l'Observatoire astronomique de Strasbourg

L'Observatoire astronomique de Strasbourg, fondé le 22 septembre 1881 par l'empereur Guillaume 1er d'Allemagne, fait donc partie des nombreuses UMR présentes en France (UMR 7550 précisément). Abritant sous sa grande coupole une lunette de 7m de long, son objectif initial était l'astronomie de position.

Figure 2 - Logo Observatoire

A partir de 1965, Pierre LACROUTE développe l'archivage informatique, ce qui contribuera à la création du Centre de Données Stellaires (CDS) en 1972 par l'Institut National d'Astronomie et de Géophysique (devenu l'INSU depuis). Le CDS deviendra le Centre de Données Astronomiques de Strasbourg et conservera néanmoins l'acronyme CDS. Ce dernier a pour mission de collecter, homogénéiser, distribuer et préserver l'information astronomique.

Figure 3 - Logo CDS

Finalement, en 1981, l'Observatoire se voit doté d'un Planétarium, destiné à la vulgarisation et la diffusion des connaissances au plus grand nombre. Le Planétarium appartient depuis 2008 à l'Université de Strasbourg mais continue d'entretenir des relations étroites avec le CDS.

I.2. Situation géographique de l'entreprise

Le siège social du CNRS se situe à Paris, cependant, les différentes UMR du CNRS sont réparties un peu partout en France.

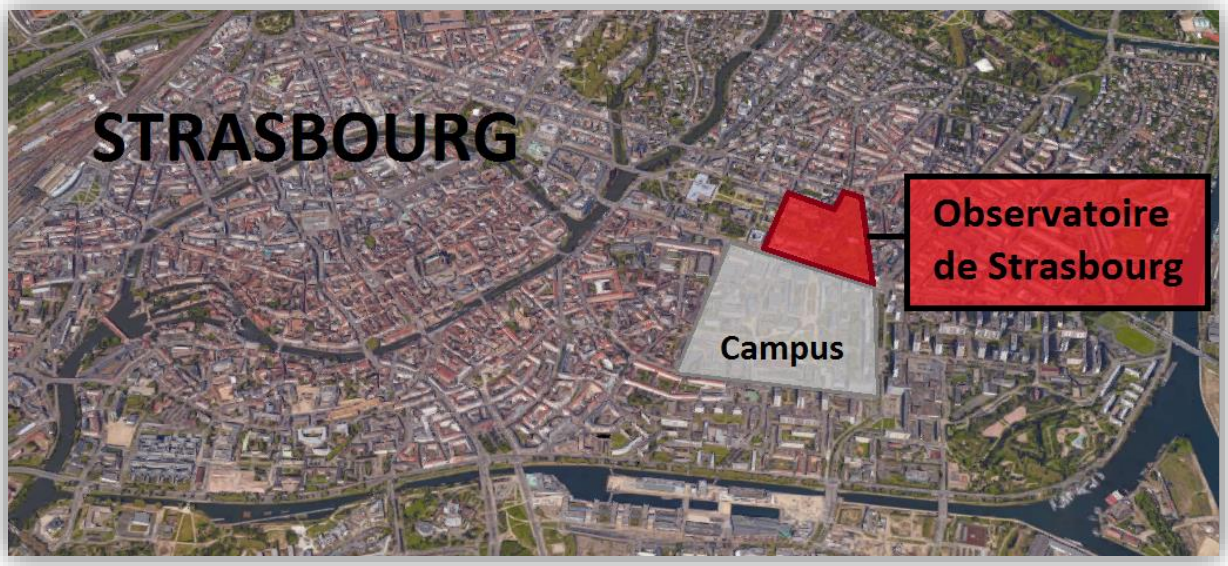


Figure 4 - Situation de l'Observatoire

L'Observatoire astronomique de Strasbourg se situe dans le quartier Conseil des XV - Rotterdam de Strasbourg, au 11 Rue de l'Université, 67000 Strasbourg. A moins de dix minutes du centre-ville, il jouxte le campus historique de l'Université de Strasbourg (campus de l'Esplanade).

Il est composé d'une Grande Coupole, d'un bâtiment des salles méridiennes avec deux coupoles et d'un bâtiment à usage administratif. Les trois bâtiments sont reliés entre eux par des couloirs couverts afin de permettre au personnel de circuler rapidement sans se soucier des intempéries. Il s'entoure d'un vaste parc abritant également le jardin botanique de l'Université de Strasbourg.

Le Planétarium, quant à lui, a été aménagé dans une des salles méridiennes du bâtiment Est.

La plupart des équipes du CDS se concentrent d'ailleurs dans ce bâtiment dans lequel sont actuellement aménagés plusieurs bureaux, salles de réunion ainsi qu'une bibliothèque.

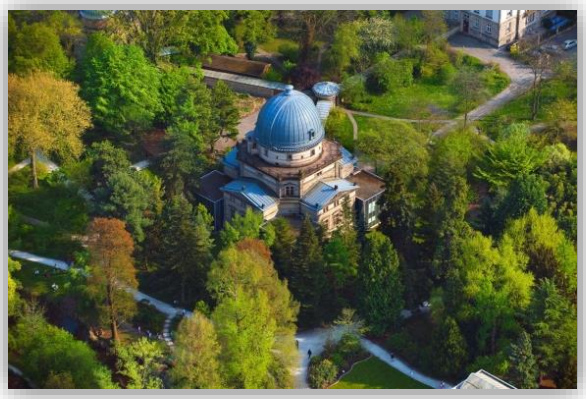


Figure 5 - Grande Coupole de l'Observatoire

I.3. Présentation de l'entreprise

I.3.a. L'Observatoire astronomique de Strasbourg

L'Observatoire astronomique de Strasbourg est un Observatoire des Sciences de l'Univers (OSU) de l'Institut National des Sciences de l'Univers (INSU) et l'UMR 7550 du CNRS et de l'Université de Strasbourg. Elle remplit à la fois des missions :

- D'enseignement : Master en astrophysique, doctorats, licences et Master Sciences, etc.
- D'observation : acquisition des données astronomiques.
- De recherche : exploitation des données astronomiques, partagée en trois équipes de recherches (Hautes Energies, Galaxies et CDS).
- De diffusion des connaissances : par le biais d'événements ou par celui du Planétarium.

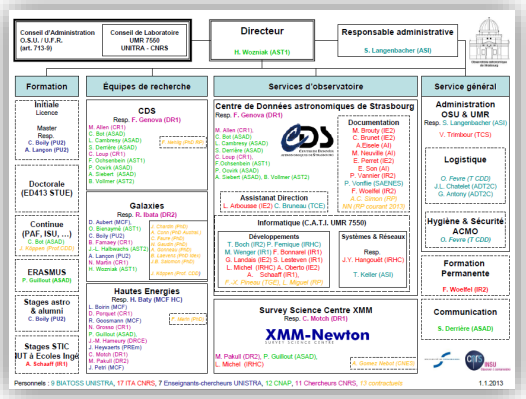


Figure 6 - Organigramme de l'Observatoire (Voir Annexe)

Parmi les services qu'il offre, l'Observatoire fait partie du consortium Survey Science Center de la mission XMM-Newton et proposent des services d'identification, de bibliographie et de catalogue via le CDS.

Enfin, l'Observatoire est administré par son directeur M. Hervé WOZNIAK, un conseil de Laboratoire (présent dans chaque UMR) et un conseil d'administration OSU/UFR.

I.3.b. Le CDS

Infrastructure de recherche du CNRS, le Centre de Données astronomiques de Strasbourg ou CDS est dédié à la collecte et à la distribution mondiale de données astronomiques.

Ses missions se traduisent par l'acquisition de données astronomiques et leur homogénéisation dans le système informatique, la maintenance et la mise à jour des bases de données et la mise à disposition de services permettant la distribution et l'exploitation des données astronomiques.

Le CDS héberge ainsi la base de référence mondiale pour l'identification d'objets astronomiques et propose les services suivants :

- Simbad : Base de données de référence pour l'identification et la bibliographie d'objets astronomiques (plus de 3 millions d'objets),
- VizieR : Collection de catalogues astronomiques et de tables publiées dans les journaux académiques,
- Aladin : Atlas interactif du ciel permettant la visualisation d'images ou données astronomiques provenant de nombreuses sources (plus de 5 téraoctets).

Figure 7 - Simbad

Figure 9 - Logo VizieR

Figure 8 - Logo Aladin

Composante de l'UMR 7550, le CDS fait partie intégrante de l'Observatoire et est donc administré par les services d'administration de l'Observatoire astronomique de Strasbourg.

II. DEROULEMENT DU STAGE

II.1. Sujet de stage

II.1.a. Contexte

Le CDS ayant pour mission la collecte et la distribution de données astronomiques, il est de son devoir de garantir leur accès en maintenant la qualité de son service et en anticipant les évolutions techniques.

Pour ce faire, le CDS mène une constante activité de R&D. Ces dernières années, les préoccupations principales du CDS concernent le volume croissant des données collectées et donc le domaine du Big Data. Dans ce cadre, la R&D implique des investigations au niveau l'accès et de la recherche des données, de leur visualisation et de leur interprétation.

En ce qui concerne la visualisation et l'interprétation des données, le CDS travaille notamment dans le domaine de la réalité virtuelle en cherchant à exploiter cette évolution technique en sa faveur. Pour ce faire, elle met à disposition de ses employés des outils tels que l'Oculus Rift de Facebook ou la Cardboard de Google.

En outre, le CDS entretenant des relations étroites avec le Planétarium, les résultats de cette R&D sont susceptibles de s'adresser non seulement aux chercheurs du CNRS mais également à un public moins averti tel que celui du Planétarium.

II.1.b. Objectifs

Afficher une scène 3D contenant toutes les étoiles d'une galaxie, comme celle d'Andromède par exemple, reviendrait à afficher quelques… mille milliards d'étoiles, soit mille milliards de points si on les réduit à l'élément le moins coûteux en ressource possible. Quelle que soit la configuration de votre système informatique, il est tout à fait impossible d'entrer autant de données en mémoire et encore moins de les afficher dans un écran de 2 073 600 pixels (exemple du Full HD), d'interagir avec ou de les mettre en mouvement.

Figure 10 - Objectif n°1

Ce qui nous amène à l'un des objectifs principaux de ce stage : **l'optimisation des performances**. Cette optimisation soulève plusieurs interrogations telles que :

- Combien de données pouvons-nous afficher en conservant des performances raisonnables ?
- Comment faire pour simuler la navigation dans des masses de données bien plus grandes que la capacité d'affichage ?

Evidemment, la navigation et l'interprétation des données sont elles aussi des opérations coûteuses en ressources et doivent être prises en compte.

Le public visé par le projet n'est pas encore totalement défini et peut autant être un scientifique averti qu'un visiteur du Planétarium. Quoi qu'il en soit, une chose est certaine, le public final ne sera pas ingénieur informaticien spécialisé en Big Data, réalité virtuelle ou bien en infographie. Le projet final se doit donc d'être à la portée de tous, **facile d'utilisation**, à l'aide d'une interface simple et intuitive.

Figure 11 - Objectif n°2

De même, le projet n'aurait pas beaucoup d'intérêt s'il fallait posséder la dernière carte graphique du marché pour le faire fonctionner. S'en ressort donc fortement un besoin de justement peser la balance entre performances et coût en ressources.

Figure 12 - Objectif n°3

La portabilité représentera d'ailleurs une part importante du projet. La 3D, l'exploitation de la carte graphique, la réalité virtuelle… sont autant de raisons qui vont multiplier le nombre de dépendances et de plugin nécessaires au fonctionnement du projet, au point de le rendre inaccessible à un utilisateur standard. C'est pourquoi le projet doit s'orienter dans le but de faciliter son exportation, son accès et son utilisation au plus grand nombre en priorisant l'emploi de technologies accessibles telles que WebGL.

Enfin, le projet est amené à évoluer constamment. C'est pourquoi le mener en se basant sur des critères de qualité logicielle est essentiel. En effet, le projet risquant de devenir de plus en plus dense et complexe, une architecture rigoureuse et une indépendance des modules est indispensable pour permettre **la maintenabilité et la réutilisabilité** du projet.

Figure 13 - Objectif n°4

II.1.c. Expression du besoin

Le besoin principal consiste à maintenir la qualité des services du CDS en innovant à l'aide de la réalité virtuelle. L'application finale pourrait bien s'ajouter à la liste des services mis à disposition par le CDS.

Ce besoin peut se décomposer en plusieurs points :

- Mener des recherches quant aux possibilités offertes par WebGL
- Mener des recherches quant aux solutions existantes optimiser le rendu graphique
- Mener des recherches quant aux applications possibles de la réalité virtuelle
- Mener des recherches quant à la compatibilité des nouvelles technologies

- Mener des recherches quant aux fonctionnalités qui puissent être utiles
- Développer les fonctionnalités
- Développer une interface
- Adapter le développement à une utilisation VR (Virtual Reality)
- Maintenir une architecture de code rigoureuse

Au cours de ce stage, nous avons été constamment en contact avec des chercheurs qui seraient amenés à utiliser notre application. Cela nous a permis de comprendre précisément ce dont ils avaient besoin et d'établir une liste de fonctionnalités à implémenter :

- Chargement de jeux de données personnels
- Visualisation de la scène sous différents angles à l'aide de plusieurs caméras
- Sélection de données
- Visualisation de l'évolution des données en fonction du temps
- Etc.

La liste n'est pas exhaustive puisque le nombre de fonctionnalités techniques (effet de flou, etc.) est grand. Il s'agit des fonctionnalités les plus essentielles.

Les objectifs de la précédente partie, le besoin et les fonctionnalités cités précédemment englobent tout le projet. Mon coéquipier et moi avons travaillé de concert sur ce projet et avons travaillé simultanément parfois sur des mêmes fonctionnalités, parfois sur des fonctionnalités tout à fait distinctes. Cela signifie que si tous ces objectifs me concernent, certains n'ont jamais été ma préoccupation majeure puisque mon coéquipier les prenait en charge et inversement.

II.1.d. Contraintes

La plupart des contraintes qui vont être détaillées ci-dessous ont un lien direct avec les objectifs de ce stage.

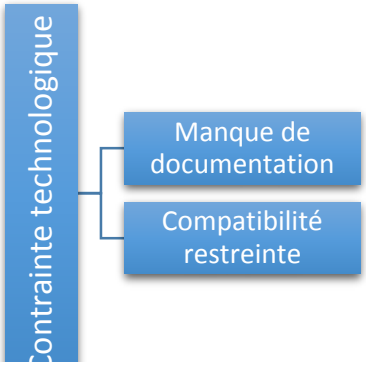


Figure 14 - Contrainte n°1

Dans un premier temps, il s'agit d'exploiter de nouvelles technologies, on se confronte donc à une **contrainte technologique**. D'une part, nous sommes limités par la quantité de connaissance que l'on peut acquérir sur ces technologies. Leur nouveauté implique **un manque de documentation et d'exemples**. D'autre part, nous sommes confrontés à des **problèmes de compatibilité**. En effet, lorsqu'une nouvelle technologie fait son apparition, les différentes entreprises œuvrent progressivement à la prendre en compte et donc à rendre leurs outils compatibles. En

l'occurrence la réalité virtuelle apporte son lot de restrictions : si la plupart des navigateurs web intègrent déjà la compatibilité WebGL pour afficher des scènes 3D, peu d'entre eux sont compatibles avec les différentes fonctionnalités de l'Oculus Rift ou de la Cardboard.

Par exemple, en ce qui concerne la Cardboard, aucun navigateur web ne propose d'API communiquant avec les différents composants d'un smartphone tels que le magnétomètre. Et un seul permet de récupérer les informations de la caméra externe d'un smartphone.

Ce qui nous ramène également à une **contrainte matérielle** en ce qui concerne la portabilité. Si l'application idéale ne fonctionne que sur un type de smartphone ayant une version précise d'un navigateur précis, il sera nécessaire de faire des sacrifices afin de la rendre plus accessible.

Dans un deuxième temps, et ceci concerne plus ce qui a été fait par mon coéquipier, l'objectif d'optimisation des performances nous confronte évidemment à une contrainte liée à la **gestion du Big Data** (qui est une contrainte matérielle si l'on considère qu'elle dépend des capacités offertes par une machine). Une fois arrivé à afficher un grand nombre de points, il faudra également être capable de les mettre en mouvement et d'interagir avec eux, et ce, tout en conservant des performances raisonnables.

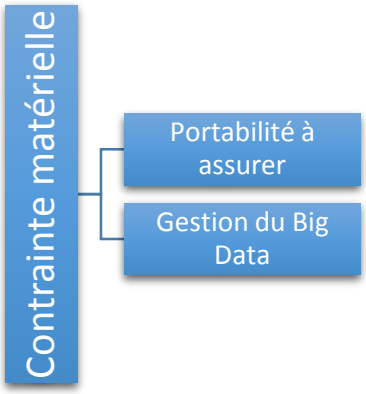


Figure 15 - Contrainte n°2

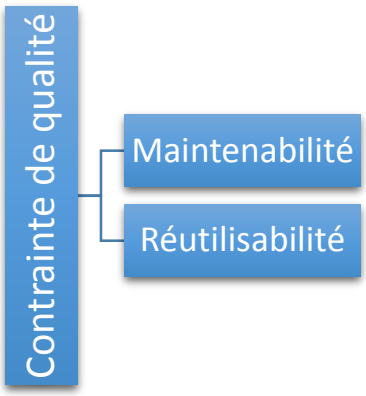


Figure 16 - Contrainte n°3

Enfin, pour rendre l'application facile d'utilisation, il faudra encapsuler tous les mécanismes de l'application pour ne laisser qu'une interface simple et intuitive à l'utilisateur. Si un projet web se prête fortement à une architecture lâche, il faut impérativement que nous développions notre application en suivant des **critères de qualité logicielle** stricts et ce, pour des questions de maintenabilité certes, mais également pour optimiser les performances au long terme et faciliter ladite encapsulation.

II.2. Organisation

II.2.a. Spécificités du stage

Cette partie est consacrée à l'organisation du travail tout au long du stage, mais j'aimerais en outre faire le contraste avec l'organisation du travail du stage précédent et expliquer les différences.

Tout d'abord, pour mener à terme cette R&D, je n'ai pas été seul contrairement à l'an passé. J'ai travaillé de concert avec Pierre LESPINGAL tout au long de ce stage. Il a donc fallu se partager le travail, parfois sur un même fichier, parfois sur des modules différents et parfois sous des angles différents.

Figure 17 - Spécificité n°1

Ce qui est intéressant ici, c'est que le partage du travail a été à notre charge, nous avions la liberté de décider de ce que nous faisions. Si nos sujets de stage étaient légèrement différents sur le papier (Pierre avait un sujet plus orienté optimisation des performances tandis que je devais pour ma part me concentrer sur les interactions possibles et sur l'utilisation d'une Google Cardboard), nous nous sommes rapidement rendu compte qu'ils étaient trop liés pour travailler chacun de notre côté.

Figure 18 - Spécificité n°2

Ensuite, nous avons repris un projet déjà entamé quelques mois auparavant par un précédent stagiaire. S'ajoute donc une phase d'étude et de compréhension de l'existant contrairement à l'an passé.

Enfin, étant une activité de R&D, notre objectif principal n'était pas clairement défini : il s'agissait d'aller aussi loin que possible en exploitant les possibilités offertes par les technologies. Il fallait donc se renseigner sur ces possibilités et souvent entreprendre des développements qui n'auraient pas pu être prévus en début de stage. Il s'agit là d'un aspect très intéressant de la R&D qui ne se retrouve pas forcément dans un projet d'entreprise au cahier des charges strict.

Figure 19 - Spécificité n°3

Si le rapport de stage de l'an dernier suivait la chronologie du développement, ce rapport de stage, pour des raisons évidentes apparaissant dans la section suivante, ne pourra mettre en valeur la démarche suivie et se décomposera plutôt en thèmes tels que l'interface, la Cardboard ou la prise en main des technologies et mettra donc en valeur les recherches effectuées ainsi que les résultats obtenus de la R&D.

II.2.b. Démarche

Dans un premier temps, j'ai pris du temps afin d'apprivoiser les nouvelles technologies que j'allais côtoyer tout au long de ce stage. J'ai mis de côté l'existant, les étoiles et la matière noire et me suis entraîné à utiliser WebGL via la bibliothèque Three.JS en créant des scènes

3D primaires me permettant de tester la plupart des fonctionnalités. Ensuite, j'ai fait de même en employant la Google Cardboard et en étudiant les spécificités qui l'accompagnent.

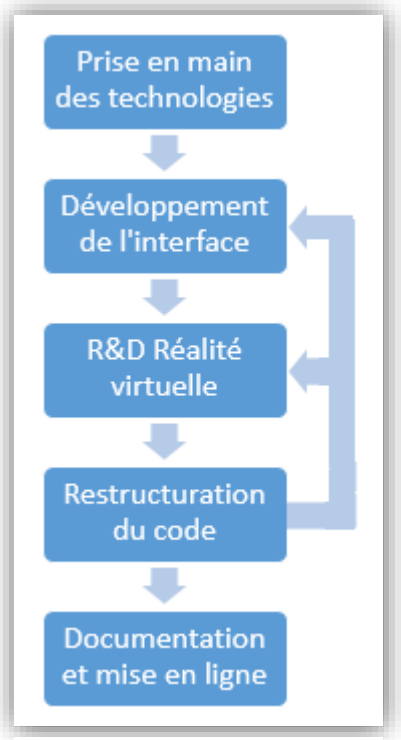


Figure 20 - Démarche

Dans un second temps, nous nous sommes mis d'accord sur l'importance d'apporter une interface menu à l'application afin de pouvoir jongler entre les modes Cardboard, Oculus Rift, vue simple et plusieurs vues dans la même application. S'est ensuivi une étude sur comment rendre la chose possible à partir de l'existant et le développement de la solution et d'une interface.

Dans un troisième temps, j'ai eu à cœur de prendre en charge la partie Cardboard de l'application, et me suis lancé dans le développement de celle-ci. L'avantage ici était la séparation des tâches : depuis le menu, l'utilisateur aurait le choix entre le mode Cardboard et la vue normale ou l'Oculus Rift. Pierre pourra ainsi travailler sur l'optimisation de la vue normale tandis que je me penche sur la partie Cardboard et ce, sans que nous nous marchions sur les pieds.

Si nos objectifs respectifs étaient encore vagues au début du stage, ils se sont grandement précisés au fur et à mesure de celui-ci. Pierre s'est focalisé sur l'optimisation des performances et l'ajout d'un grand nombre de fonctionnalités à l'application principale en exploitant la notion d'octree (structure de données complexe susceptible d'améliorer les temps de calculs) tandis que de mon côté, je me suis focalisé sur l'utilisation de la Google Cardboard dans l'application et sur certains éléments de l'interface de l'application principale.

La démarche de travail a commencé à être déstructurée lorsque le besoin de restructurer le projet s'est fait ressentir. D'une part, le projet étant amené à devenir de plus en plus dense et complexe, d'autre part la redondance et la personnalisation du code pour chaque fonction, ont rendu la mise en place d'une architecture rigoureuse indispensable. Nous avons donc fait le choix de reprendre le projet de base à 0 en lui appliquant une architecture orientée objet afin qu'il puisse respecter des critères de qualité logicielle telle que la maintenabilité et la réutilisabilité du code.

La majeure partie du projet n'étant pas notre création à ce moment-là et les solutions pour rendre indépendant des modules étant nombreuses, l'opération a été très chronophage. En outre, ayant eu à travailler sur des mêmes fichiers en début de stage, le partage des tâches concernant la restructuration du code était compliquée. Ne pouvant pas tout faire d'une

traite, plusieurs phases de restructuration ont parsemé le stage. Bien que le remaniement du projet représente un gros investissement en termes de temps, il était néanmoins indispensable pour des questions de performances et a permis un grand gain de temps à long terme.

La démarche est donc passée d'une démarche linéaire à une logique de besoin/solution. Si mon objectif principal était l'étude et le développement de fonctionnalités pour la Google Cardboard, il s'agissait là d'une tâche moins prioritaire que de développer l'interface dans la mesure où son absence pouvait gêner Pierre dans son développement. Ainsi à chaque nouvelle version de l'application principale, le besoin de mettre à jour l'interface étant plus important, je mettais en pause la R&D sur la Cardboard pour développer l'interface.

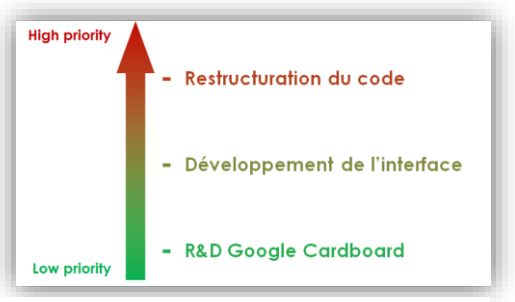


Figure 21 - Priorité des tâches

De même, la restructuration du code évoquée plus haut était prioritaire par rapport au développement de l'interface dans la mesure où elle gelait le développement de l'un et de l'autre.

Enfin, une documentation en ligne a été mise à jour régulièrement tout au long du stage et a été complétée en fin de celui-ci. De même, les fichiers ont étés « nettoyés » des lignes de code devenues obsolètes et l'ensemble du projet a été mis en ligne sur un serveur du CDS accompagné d'une notice d'utilisation (README).

II.3. Environnement de travail

II.3.a. Outils

Dans le cadre de ce stage, il a fallu allier modernité et efficacité dans les domaines du web et de l'infographie. Si l'HTML, le CSS et le JS sont de nos jours indissociables d'un projet web, les variantes concernant bibliothèques et frameworks JS sont nombreuses et ce, afin de convenir à tous les besoins.

La 3D et la réalité virtuelle sont en règle générale des domaines gourmands en ressources, c'est pourquoi se servir d'outils optimisés pour cette utilisation est important. Développer en JS pur est en théorie ce qu'il y a de plus performant et de plus léger, mais cela implique de réinventer la roue dans de nombreuses situations. Il s'agit donc de développer un bon

compromis et d'employer les bibliothèques et frameworks uniquement lorsque cela s'avère vraiment nécessaire.

Le choix des outils et des technologies fait partie intégrante de l'activité R&D dont nous avons fait partie. Cette partie est destinée à la description des outils à notre disposition et des technologies ayant été sélectionnées.

II.3.a.i. Matériel

L'Observatoire m'a fourni un ordinateur fixe dont la configuration était adaptée au développement dans le domaine 3D. La machine héberge un système Linux Ubuntu 14.04 LTS 64bits, tourne à l'aide d'un processeur Intel Core I5-4570 cadencé à 3,20GHz et possède 16Go de mémoire vive. Il renferme également une carte graphique dédiée AMD Radeon R7 200 Series ainsi qu'une carte vidé additionnelle.

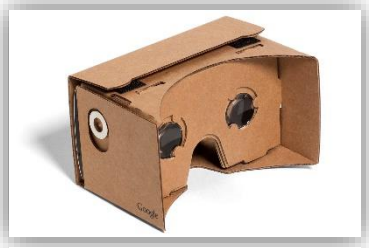


Figure 22 - Google Cardboard

Figure 23 - Oculus Rift

En outre d'un environnement de travail performant, l'Observatoire a également mis à notre disposition un équipement dédié à la réalité virtuelle. Celui-ci comprend un Oculus Rift SDK1, un Oculus Rift SDK2 ainsi qu'une Coogle Cardboard V1.

II.3.a.ii. Logiciels et services

- **Webstorm**

Figure 24 - Logo WebStorm

WebStorm est un IDE (Integrated Development Environment) spécialement dédié aux langages Web (HTML, CSS) et notamment au Javascript. Moderne et performant, il a été développé en 2010 par la société JetBrains et en est actuellement à sa version 10. Il s'agit d'un logiciel payant mais il est proposé gratuitement aux étudiants.

Nous avons choisi cet IDE parce que nous le maîtrisons, parce qu'il est totalement dédié JavaScript (langage dominant de l'application) et parce qu'il offre à nos yeux les meilleurs services et performances sur le marché.

❖ **GitHub**

GitHub est un service d'hébergement et de gestion de développement de logiciels. Celui-ci nous a permis de travailler en équipe sur notre projet, de sauvegarder nos fichiers et de conserver une trace du développement. Il a été lancé en 2008 par la société du même nom. La création d'un dépôt privé est généralement payante mais cinq dépôts privés sont offerts aux étudiants.

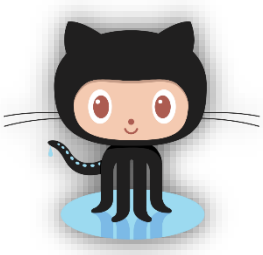


Figure 25 - Logo GitHub

Nous avons choisi ce système de gestion de version parce qu'il est moderne, réputé et que nous le maîtrisons.

❖ **Heroku**

Figure 26 - Logo Heroku

Heroku est un service de déploiement et d'hébergement d'applications web. Il nous a permis d'héberger temporairement notre application afin de la tester sur le terrain ainsi que sur mobile. Il a été créé en 2007 et appartient aujourd'hui à la société Salesforce.com. Il s'agit d'un service gratuit dans son utilisation basique.

Nous avons choisi ce service parce qu'il est réputé et gratuit.

❖ **JSDoc**

JSDoc est un moteur de documentation pour langage Javascript. Il nous a permis de générer une documentation en ligne de notre application à partir de commentaires spécifiques que nous avons placés dans l'application. Il s'agit du moteur de documentation le plus répandu, nous l'avons utilisé pour gagner en temps et en qualité.

Figure 27 - Logo JSDoc

II.3.a.iii. Langages, API et frameworks

❖ **OpenGL**

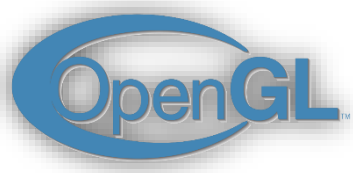


Figure 28 - Logo OpenGL

OpenGL est une technologie permettant de communiquer avec la carte graphique. Il s'agit d'une API (Application Programming Interface) permettant d'intervenir dans le processus de pipeline graphique et de personnaliser ce dernier selon ses besoins. OpenGL a été créé en 1992 par

la société Silicon Graphics, en est à la version 4 et est actuellement particulièrement bien supporté par la plupart des plateformes.

❖ WebGL

L'ensemble du projet se base sur la technologie WebGL. Il s'agit d'une API web qui exploite la technologie OpenGL afin de communiquer avec la carte graphique. WebGL permet ainsi le développement d'applications web 3D pouvant exploiter la carte graphique afin de personnaliser et d'optimiser le rendu graphique. Créé en 2009 par la fondation Mozilla et le groupe Khronos, et bien qu'il n'en est qu'à sa première version, WebGL est supporté par la plupart des navigateurs sans nécessiter l'installation de greffons.

Cette technologie a notamment été choisie pour son accessibilité puisqu'elle permet le développement d'une application 3D visualisable depuis n'importe quel navigateur, standard ou mobile.

Figure 29 - Logo WebGL

❖ Three.JS

Three.JS est un framework javascript dédié au développement d'applications web 3D. Il ne nécessite pas non plus de greffon pour fonctionner et ce, parce qu'il exploite la technologie WebGL. Three.JS contient fournit un grand nombre de fonctions optimisées pour le développement d'applications 3D et ne laisse pas en marge à la réalité virtuelle. Apparu en 2010, il a été développé par « mrDoob » qui héberge publiquement le code source sur GitHub.

Figure 30 - Logo Three.JS

Three.JS a été sélectionné parmi d'autres bibliothèques comme étant le framework le plus adapté à notre projet. Son principal concurrent, Babylon.JS s'est avéré être plus orienté jeux vidéo. Cette technologie a donc été choisie pour ses performances avant tout.

❖ HTML5, CSS3 & Javascript

Ces trois technologies sont indissociables du développement web. Le projet tend à suivre les bonnes pratiques de programmation, pour ce faire l'application est développéemajoritairement en Javascript, le HTML et le CSS étant uniquement employés dans un but de mise en page.

Figure 32 - Logo HTML5

Figure 33 - Logo Javascript

Figure 31 - Logo CSS3

Javascript est un langage de programmation web qui permet notamment de développer des pages web dynamiques. Créé en 1995, il en est actuellement à sa version 1.8.5.

HTML5 est la dernière révision majeure d'HTML (format de données). Créé en vers 1990, le World Wide Web Consortium (W3C) est actuellement en charge du développement des spécifications HTML.

CSS3 est un langage informatique permettant de décrire la présentation d'un document HTML. Créé en 1990, ses standards sont également publiés par le W3C.

II.3.b. L'existant

A notre arrivée, le projet avait déjà été entamé par un précédent stagiaire. Celui-ci en était arrivé au point de visualiser plus de deux millions de points dans une scène 3D.

Pour ce faire, nous avons à disposition des données de test. Chaque chercheur nous ayant fourni ses données de tests les a formatés dans un fichier à sa manière. Ainsi, le chargement des données est spécifique pour chaque fichier test.

Données	*Informations*					
	X	Y	Z	Densité	Âge	Etc.
1	0	0	0	1	2M	...
2	0	0	1	0	2M	...
3	0	1	1	1	2M	...
4	1	1	1	0	2M	...
5	1	0	1	1	2M	...
Etc	...	...	...	...	...	...

Figure 34 - Format des données

Globalement, un fichier de données se compose de lignes et de colonnes. Chaque ligne correspond à une donnée et chaque colonne correspond à une information sur la donnée. Une donnée astronomique peut représenter diverses entités astronomiques telles que des étoiles ou de la matière noire. Ces entités peuvent donc avoir des caractéristiques telles qu'une position dans l'espace, une densité, un âge ou une vitesse de déplacement et ce sont ces informations que l'on retrouve dans les colonnes.

Afin de charger les données, l'application demande à l'utilisateur de parcourir ses fichiers pour sélectionner le fichier de son choix. Une fois sélectionné, l'application va le parser afin de récupérer pour chaque donnée ses informations de position en X, Y et Z.

Ces données seront ensuite restreintes à des points afin d'économiser en ressources. Une fois que tout est chargé, les points s'affichent dans le repère orthonormé présent dans la scène 3D de l'application.

De plus, une navigation de type FPS (First Person Shooter) au clavier et à la souris est possible pour se déplacer dans l'espace 3D.

Le fichier test le plus volumineux contenait 128^3 données, soit 2 097 152 points à afficher. L'application de base les affichait sans problème de performance et la navigation était fluide.

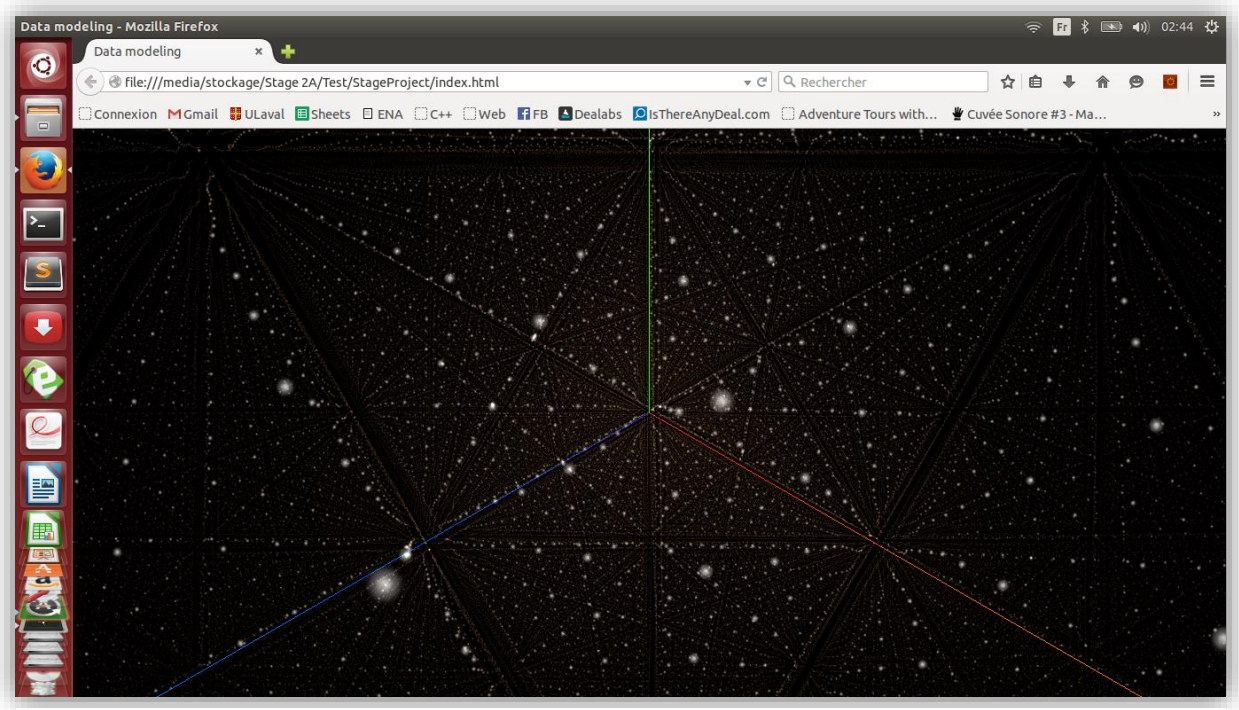


Figure 35 - Cube de 128^3 données affiché, ici les points sont lumineux et sphériques mais ils étaient carrés et blanc dans le projet de base

Cependant aucun optimisation n'était présente (puisque non nécessaire à ce stade), ni aucune interface ou possibilité d'interactions. Dans le futur, il s'agira de charger plus de deux millions de points et de récupérer non seulement leur position, mais également toutes leurs autres informations (ce qui multiplie rapidement le nombre de millions de données à mettre en mémoire). En outre, toute fonctionnalité agissant sur les données travaillera donc sur des millions de points à la fois. En somme, l'optimisation sera prédominante en ce qui concerne la visualisation des données et nous aurons comme objectifs de naviguer à tout moment avec au moins 30 images par seconde (30 fps).

III. TRAVAIL EFFECTUE

III.1. Développement de l'interface

En programmation orientée objet, il y a une logique de séparation en modules indépendants. Ainsi l'interface n'a pas conscience de l'application. On doit pouvoir l'ajouter ou l'ôter sans aucun problème de dépendance.

Comment l'interface peut-elle permettre d'agir sur l'application alors ?

Un fichier principal se charge de la communication entre les modules. Celui-ci fait donc l'intermédiaire entre le fichier de l'interface qui ne concerne que la partie visuelle de celle-ci (apparition/disparition/déplacement/animation des éléments) et les différents fichiers contenant les fonctions de l'application.

Dans cette partie, je présenterai donc l'interface en tant que telle, ses fonctions, mais je n'entrerai pas dans les détails de la logique avec laquelle elle communique, qui a en grande partie été développée par mon coéquipier. J'ai intervenu dans la logique en début de stage, mais une fois le projet ayant été remanié, mes interventions ont été soient rayées (puisque l'architecture objet nous a permis de supprimer les redondances), soient modifiées, il n'est donc pas pertinent de chercher à les présenter.

On voit ici apparaitre la facilité du partage des tâches en programmation orientée objet : interface et logique étant indépendants l'un de l'autre, nos travaux étaient complémentaires et ne posaient aucun problème de fusion.

III.1.a. Menu

III.1.a.i. Présentation

Le menu répond à un besoin simple, il s'agit de pouvoir naviguer entre les différents modes développés tout en restant dans la même application.

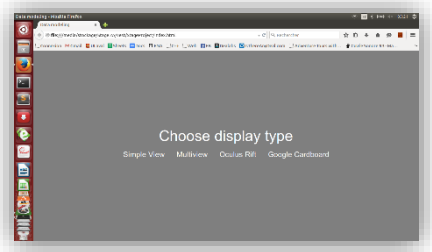


Figure 36 - Interface du menu (Voir Annexe)

Nous avons envisagé le développement de quatre modes d'affichage pour notre application :

- Vue simple
- Multi vues
- Oculus
- Cardboard

Sans menu, il aurait fallu soit développer quatre applications, soit recharger l'application avec des paramètres différents. L'interface de navigation entre les modes était donc indispensable pour assurer une expérience convaincante.

III.1.a.ii. Les différents modes

❖ **Vue simple (SimpleView)**

Le mode SimpleView correspond à l'application de base. Il permet de visualiser la scène à travers une unique caméra que l'on dirige l'aide contrôles de type FPS.

Ce mode est principalement pensé pour utilisation sur ordinateur et a accès à tous les fonctionnalités de l'application.

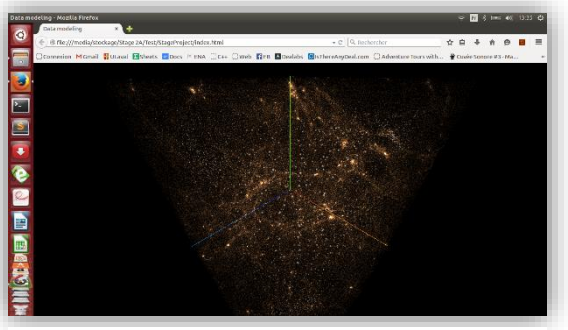


Figure 37 - Mode SimpleView (Voir Annexe)

❖ **Multi vues (MultiView)**

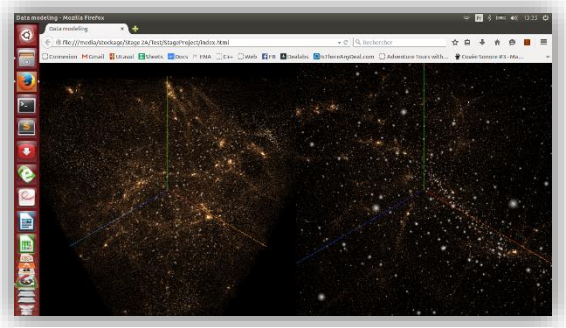


Figure 38 - Mode Multiview (Voir Annexe)

Le mode MultiView reprend le mode précédent et le double. Il permet de donc de visualiser la scène à l'aide de plusieurs caméras (par défaut, deux caméras). Chaque vue est indépendante, on peut donc se déplacer dans l'une indépendamment de la seconde. De même, les fonctionnalités s'appliquent aux vues de manière indépendante. Il est par exemple possible d'afficher les données dans la vue de gauche et de les masquer dans celle de droite.

Ce mode est principalement pensé pour une utilisation sur ordinateur, notamment parce que le nombre de vue réduit la taille de celles-ci et rendrait donc l'expérience désagréable sur mobile.

❖ **Oculus (Oculus Rift)**

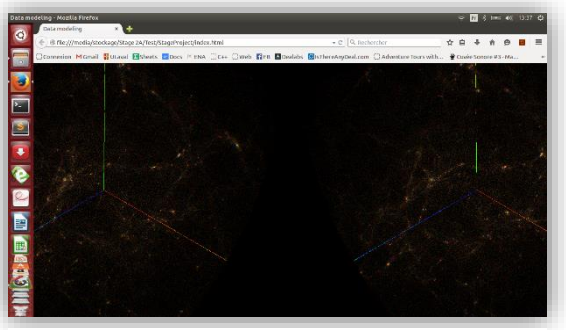


Figure 39 - Mode Oculus Rift (Voir Annexe)

Le mode Oculus est destiné à offrir une vue de réalité virtuelle pour une utilisation avec l'outil Oculus Rift de Facebook. Bien qu'en réalité virtuelle, l'application affiche une vue dédoublée (une pour chaque œil), l'application la gère comme étant une seule et unique vue. Sur ordinateur, toutes les fonctionnalités sont disponibles, mais il n'y a pas d'intérêt à cette utilisation. Le mode Oculus n'a pas été approché, quelques essais ont été effectués, mais il n'est pour l'instant pas fonctionnel avec un Oculus.

❖ **Cardboard (Google Cardboard)**

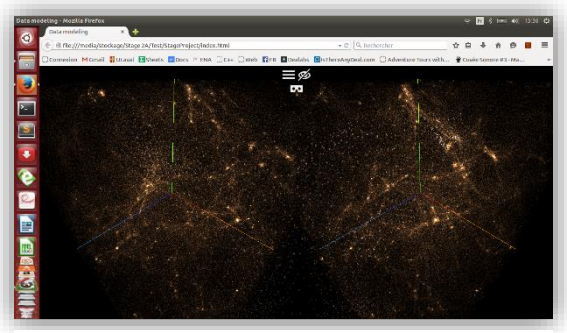


Figure 40 - Mode Google Cardboard (Voir Annexe)

Le mode Cardboard est destiné à offrir une vue de réalité virtuelle pour une utilisation avec l'outil Google Cardboard. De même que pour le mode Oculus, l'application gère les deux vues affichées comme étant une seule et unique vue. Les deux vues sont différentes de celles du mode Oculus puisque les deux outils ne gèrent pas la réalité virtuelle de la même façon.

Sur ordinateur, toutes les fonctionnalités sont disponibles, et il est possible de se déplacer à l'aide de contrôles de type FPS. Ceci peut permettre de faire une démonstration en cas d'absence du matériel nécessaire.

Sur mobile, les fonctionnalités ne sont pas disponibles puisque trop gourmandes et trop spécifiques pour une utilisation mobile. Les contrôles ne sont plus de type FPS mais utilisent l'orientation du mobile pour suivre les mouvements de la tête dans l'application.

III.1.a.iii. Spécifications

Le menu est le premier visuel auquel on a affaire lorsqu'on lance l'application.

Il se veut minimaliste : clair, simple et intuitif. Pour ce faire, il propose directement les quatre modes, et une légère animation se produit lorsqu'on passe le curseur sur chaque mode afin d'indiquer à l'utilisateur qu'il peut sélectionner ce mode.

Lorsqu'un mode est sélectionné et que la scène est visualisable, l'accès au menu est garanti quelle que soit l'action en cours. Cela est possible en ayant séparé l'interface de la logique de l'application. Ainsi, que l'application soit en train de charger des données ou en pleine animation, le menu peut être ouvert.

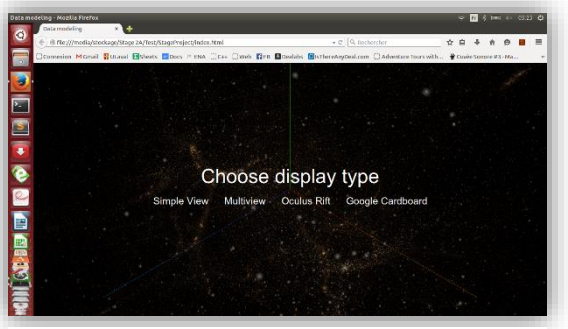


Figure 41 - Interface du menu avec le rendu en pause en arrière-plan (Voir Annexe)

Le menu est bloquant, cela signifie que la vue est mise en pause lorsque que le menu est affiché. Cela permet notamment de mettre en pause la boucle de rendu afin que l'ordinateur ne calcule pas inutilement le rendu graphique en arrière-plan lorsqu'on navigue dans le menu.

De même, afin de d'éviter au mieux les calculs redondants ou inutiles, les données chargées dans un mode sont conservées et réutilisées dans les autres modes. On peut donc à tout instant passer de la vue simple au mode Cardboard sans aucun temps d'attente, tout en conservant les données à l'écran (qui seront adaptées au mode sélectionné).

Enfin, puisque le menu met simplement en pause la vue et n'efface aucune donnée, la scène est toujours visualisable en transparence en arrière-plan du menu. Ce qui permet de savoir dans quel mode on se situe.

III.1.b. Interface Three.JS

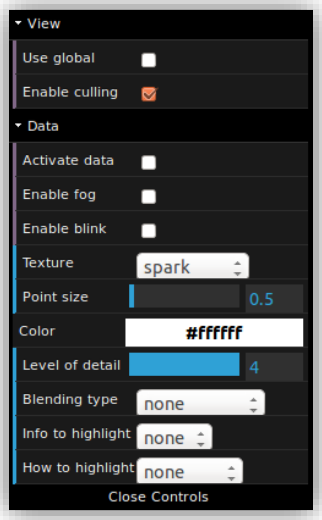


Figure 42 - Zoom sur l'interface Three.JS (Voir Annexe)

Outre le fait de créer des scènes 3D, la bibliothèque Three.JS propose également une interface sobre et efficace. Cette interface se base sur un principe simple : on considère une variable (le temps par exemple), on spécifie son aspect dans l'interface (checkbox, liste à choix, jauge, etc.), on lui attribue sa valeur initiale (0s pour le temps par exemple) et on précise les valeurs minimales et maximales qu'elle peut prendre s'il y a lieu.

Après cela, l'interface est générée automatiquement. La variable en question est alors disponible et peut être utilisée, cela signifie que si l'utilisateur modifie la valeur du temps dans l'interface, par exemple, on pourra à tout moment récupérer cette nouvelle valeur et l'utiliser dans l'application pour produire un résultat.

Cette interface très utile a cependant ses limites puisqu'elle ne propose que cette forme générique. Ainsi, quelques éléments d'interfaces ont nécessité d'être développés tels que le menu ou la timeline décrite dans la partie suivante.

On peut également lier une interface Three.JS à chaque vue afin de pouvoir rendre indépendantes les fonctionnalités dans chacune des vues. Ceci est possible grâce à l'architecture objet mise en place.

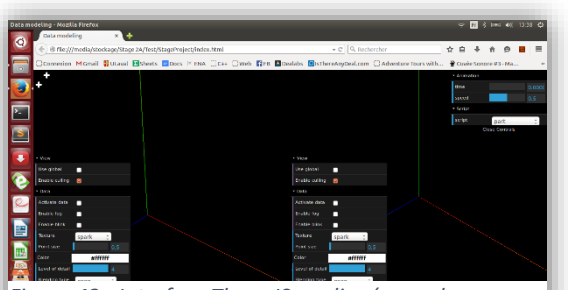


Figure 43 - Interface Three.JS appliquée sur deux vues indépendantes (Voir Annexe)

Il m'a paru important de préciser l'existence de cette interface puisqu'elle est omniprésente dans l'application et fait partie intégrante de celle-ci. Cependant je n'ai pas travaillé sur cette interface qui concerne particulièrement les fonctionnalités développées par mon coéquipier, je n'entrerai donc pas plus dans les détails.

III.1.c. Timeline

III.1.c.i. But

On nous a exprimé le besoin de pouvoir faire évoluer les données dans le temps afin de les étudier. Mon coéquipier a donc développé cette fonctionnalité et le besoin de développer une interface pour la contrôler s'est fait ressentir.

Three.JS ne propose pas de solution pour ce besoin, j'ai donc entrepris le développement d'une timeline, outil qui se prête complètement à cette fonctionnalité.

Figure 44 - Evolution du temps entre deux snapshots (Voir Annexe)

Afin de simuler l'évolution des données dans le temps, nous avons à disposition des snapshots. Un snapshot est un jeu de données comme n'importe quel autre et représente des données astronomiques relevées à une date précise ou correspondant à une date précise.

L'enchaînement de plusieurs snapshots décrivant les mêmes données mais à des dates différentes nous permet donc de simuler l'évolution des données dans le temps. Cependant, il est impensable de charger mille snapshots séparés chacun d'une année pour visualiser leur évolution, cela reviendrait à charger mille fois ce que nous avons déjà du mal à optimiser lorsqu'il est chargé une fois.

Pour répondre à ce problème, il s'agit de calculer la trajectoire prise par les données entre deux snapshots qui pourraient, par exemple, être séparés de mille années, ou dix millions. Pour ce faire, il faut implémenter des équations connues spécifiques au type de données que nous visualisons (des orbites par exemple) ou bien simplement interpoler la position des données entre les deux snapshots en prenant en compte les informations fournies dans les jeux de données (vitesse, direction, etc.). Pour l'instant, c'est cette dernière solution qui a été appliquée.

La timeline a été développée afin d'avoir à la fois un contrôle total et simple d'utilisation sur cette fonctionnalité.

III.1.c.ii. Spécifications

Une timeline est comme son nom l'indique une ligne. Cette ligne est parsemée de points d'arrêts (breakpoints) qui définissent des zones précises de la timeline, généralement pour indiquer un événement marquant dans le temps.

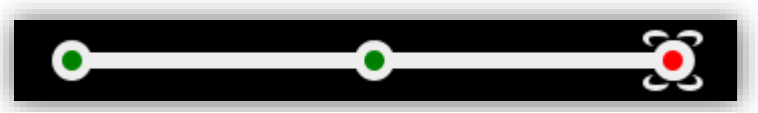


Figure 45 - Timeline

Dans notre cas, la timeline est personnalisée pour servir nos objectifs et est composée de trois éléments interactifs :

- Un curseur
- Les breakpoints
- Un bouton play/stop

Ces trois éléments sont donc les seuls avec lesquelles l'utilisateur peut interagir.

❖ Les breakpoints

Les breakpoints symbolisent les snapshots chargés. Aucun snapshot chargé signifie aucun breakpoint et la timeline est figée et inutilisable. De même pour un seul snapshot chargé puisqu'il faut nécessairement au moins deux snapshots pour calculer l'évolution du temps entre eux.

Les breakpoints se positionnent automatiquement sur la timeline et l'utilisateur ne peut les déplacer. Il est supposé que le temps séparant chaque snapshot est toujours le même, ainsi les breakpoints se positionnent sur la timeline de manière proportionnelle.

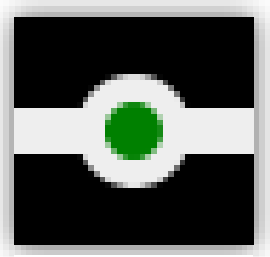


Figure 46 - Breakpoint

L'utilisateur peut ajouter à sa guise autant de breakpoints qu'il le souhaite, soit charger autant de snapshots qu'il le souhaite. Il est cependant limité à la fois par les capacités de sa machine et à la fois par la taille de son écran puisque les breakpoints se placent sur la timeline qui a une taille fixe, proportionnelle à la taille de l'écran.

La suppression de snapshots, quant à elle, est implémentée mais pas encore proposée dans l'interface.

Enfin, chaque snapshot est sélectionnable. Il est donc possible de téléporter directement le curseur dessus par un simple clic. Ce clic entraîne le chargement du snapshot et la modification de la vue en conséquence.

❖ Le curseur

Le curseur est un élément symbolisant la position dans le temps. Il est possible de le déplacer à l'aide du drag & drop sur toute la timeline. L'extrémité gauche de la timeline représente le temps zéro tandis que l'extrémité droite représente la position finale des données (le dernier snapshot chargé).

Le drag & drop implique que le curseur peut être positionné entre deux snapshots. Si tel est le cas, on peut alors visualiser la position des données à un instant T entre deux snapshots. Cela signifie que l'application calcule la position des données en fonction du temps et les affiche dans la vue comme s'il s'agissait d'un simple snapshot.

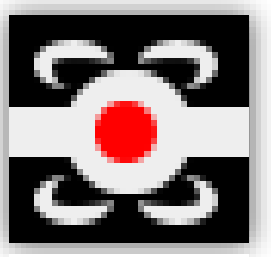


Figure 47 - Curseur

Enfin, le curseur a aussi des conséquences sur le reste de l'interface que je n'ai pas développée. Lorsque le curseur est en mouvement, la variable de temps de l'interface Three.JS est modifiée en parallèle et lorsqu'il passe un nouveau breakpoint, le snapshot « courant » affiché dans l'interface st modifié en conséquence.

❖ Le bouton play/stop

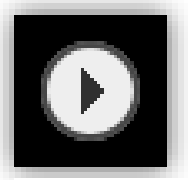


Figure 48 - Bouton Play

Ce bouton a été créé pour pouvoir visualiser l'évolution des données dans le temps à la manière d'une vidéo. Le survoler ou le sélectionner entraîne de légères animations explicites (indication pour signifier que l'élément est interactif, et visuel d'un symbole « stop » lorsque l'animation est en cours).

Quelle que soit la position du curseur sur la timeline (excepté la position finale), un clic démarrera l'animation. De même, un autre clic arrêtera l'animation. Le curseur se déplace automatiquement lors de l'animation (les conséquences visuelles du déplacement du curseur s'appliquent) et s'il arrive au bout de la timeline, l'animation s'arrête et le bouton reprend sa forme initiale.

Figure 49 - Bouton Stop

Toute action menée sur la timeline se répercute sur l'application et inversement. Ainsi, sélectionner un snapshot en dehors de la timeline déplacera le curseur sur le breakpoint correspondant, modifier la variable de temps dans l'interface Three.JS aura pour conséquence de déplacer simultanément le curseur de l'interface et lancer l'animation sans utiliser le bouton play/stop aura les mêmes effets visuels qu'un clic sur ledit bouton.

III.2. Google Cardboard

III.2.a. L'outil

La Google Cardboard est un support en carton comme l'indique son nom. Il s'agit du résultat d'une stratégie commerciale du géant Google. Alors que chaque firme s'attelle à développer son propre outil de réalité virtuelle (Oculus Rift, Gear VR, etc.) et s'apprête à les mettre sur le marché à prix fort, Google a mis en ligne publiquement les plans de construction d'un simple support en carton fonctionnant avec n'importe quel smartphone. A commander ou à construire soi-même, l'outil est d'une simplicité sans nom et se met donc à la portée de tous.

Le principe est simple, il s'agit de lancer une application VR (Virtual Reality) téléchargée sur l'App Store de son smartphone, de placer son smartphone à l'emplacement prévu à cet effet dans la Cardboard et de profiter de l'expérience sans avoir rien d'autre à faire.

Figure 50 - Google Cardboard

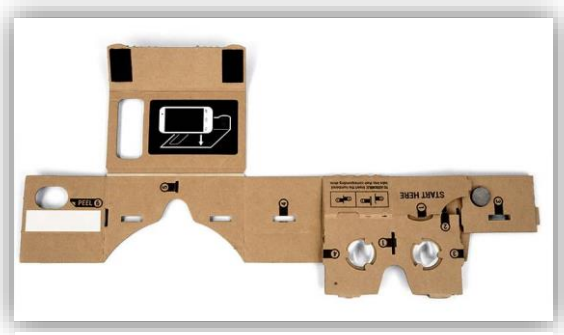


Figure 51 - Google Cardboard avant montage

Je vais tâcher d'expliquer comment cela est possible.

Une Google Cardboard V1 est composée de trois éléments principaux :

- Du carton
- Un magnet
- Des lentilles

Ces trois éléments rudimentaires (et le smartphone) sont tout ce qu'il faut pour pouvoir entrer le monde de la réalité virtuelle.

❖ Le carton

Il sert évidemment à former le support de la lunette VR qu'est la Google Cardboard et est prévu pour supporter le poids et maintenir en place le smartphone au bon emplacement.

Cette matière a été choisie non seulement pour son coût mais également pour son accès aisé. En effet, le but de l'opération est que tout le monde est accès à cette technologie et puisse l'étudier ou la développer. Pour ce faire, le carton permet de réduire le prix de la commande mais également de rendre sa construction très facile si l'on ne souhaite pas l'acheter.

Figure 52 - Carton pré-découpé

Enfin, il est formé de sorte à ne pas laisser la lumière entrer dans le support, ce qui pourrait gêner l'expérience.

❖ Le magnet

Le magnet est un aimant dont une seule face est aimantée. Il se situe au niveau de l'index de la main gauche lorsqu'on porte la lunette. Il est disposé de sorte à ce que l'on puisse le déplacer verticalement et qu'il revienne automatiquement à sa position d'origine grâce au magnétisme.

Figure 53 - Magnet

Cette action permet de simuler un interrupteur avec du matériel rudimentaire. En effet, tout smartphone récent dispose d'un magnétomètre et peut donc détecter l'impulsion magnétique provoquée par le retour brusque du magnet à sa position d'origine.

Cette impulsion peut alors être traitée par l'application mobile comme n'importe quelle autre action, telle que toucher l'écran. Le magnet a donc pour rôle de permettre l'interaction entre l'utilisateur et l'application VR alors que le smartphone, enfermé, est hors de portée de l'utilisateur.

❖ **Les lentilles**

Les lentilles se placent au niveau de nos yeux et forment en quelque sorte les « verres » de la lunette VR. Elles permettent de focaliser la vue de chacun de nos yeux sur une zone précise de l'écran. C'est l'alliance de l'application VR et des lentilles qui créent l'illusion de la réalité virtuelle et ce, à l'aide d'une astuce expliquée plus tard dans ce rapport.

Figure 54 - Lentilles

III.2.b. Pourquoi la Cardboard ?

Pourquoi ai-je choisi de focaliser mes recherches sur la Google Cardboard pour exploiter la réalité virtuelle alors que j'avais à ma disposition d'autres outils tels que l'Oculus Rift, plus précis et plus performant ?

Ce choix n'a pas été fait au hasard, j'ai choisi de mener une activité R&D sur la Google Cardboard pour la mobilité, l'accessibilité et la qualité de l'outil, outre le fait d'avoir été séduit par l'ingéniosité de l'outil.

❖ **Mobilité**

De nos jours tout le monde possède un smartphone. Et pour expérimenter la réalité virtuelle avec la Cardboard, il suffit de télécharger une application VR, disponible gratuitement ou non sur l'App Store. On est donc très loin de toutes les complications techniques que présente l'Oculus Rift dans son utilisation. En effet, ce dernier, encore dans sa version de développement, nécessite de nombreux réglages et de nombreuses compatibilités qui le rendent très difficile à utiliser.

De ce fait, pour utiliser notre application à l'aide de l'Oculus Rift, il faut se brancher à une machine compatible et déjà paramétrée tandis qu'il suffit juste de sortir son smartphone de sa poche quel que soit l'endroit où l'on se trouve avec la Google Cardboard.

❖ **Accessibilité**

Il ne s'agit pas seulement de posséder un smartphone pour profiter de l'expérience, il faut aussi pouvoir se procurer l'outil. Le côté abordable (une dizaine d'euros) de la Google Cardboard vient donc en complément de sa mobilité afin de rendre l'outil accessible à un très large public.

❖ **Qualité**

Du carton, de simples lentilles, un prix au rabais… On peut s'attendre à ce que l'expérience soir elle aussi au rabais. Evidemment, la Google Cardboard ne permet pas une expérience optimale et ne peut réellement concurrencer l'Oculus Rift. Ce dernier est entièrement développé dans un but de réalité virtuelle tandis qu'un smartphone ne l'est pas, il prend en compte un certain nombre de paramètres que ne propose pas la Google Cardboard, l'écran étant entièrement optimisé pour la réalité virtuelle.

Cependant, contre toute attente, l'expérience proposée par la Google Cardboard est très satisfaisante. L'immersion est totale pour peu que la lumière ne vienne pas interférer et que les dimensions du support soient strictement respectées. Ainsi, je n'ai pas l'impression de sacrifier les performances en utilisant ce « carton », détail qui aurait pu rendre négligeable la mobilité et l'accessibilité de l'outil.

L'application n'ayant pas un public définitivement ciblé, la mobilité et l'accessibilité de l'application est primordiale. Fournir un outil trop technique à un personnel non formé n'est pas efficace. De même, si l'outil présente trop de contraintes, telles que le fait de l'utiliser sur une machine précise, il sera évidemment peu utilisé ou abandonné.

En outre, le Planétarium serait intéressé par l'application en tant que complément aux visites habituelles. Il s'agirait là de pouvoir faire profiter à plusieurs visiteurs à la fois de l'expérience et donc d'acheter l'outil en grand nombre, de le rendre mobile et facile d'utilisation. Encore une fois, le prix et la mobilité de la Cardboard correspond totalement à ces attentes contrairement aux autres lunettes VR qui s'annoncent sur le marché.

III.2.c. L'effet Stéréo

Une fois l'outil déterminé, il s'agissait de comprendre son fonctionnement et celui de la réalité virtuelle en général. Pour commencer, comment l'association des lentilles et d'une image crée-t-elle l'illusion ?

L'effet Stéréo est l'effet appliqué à une vue 3D pour la transformer en une vue compatible avec la Google Cardboard. Il consiste globalement à prendre une vue 3D quelconque, à la dédoubler et à décaler légèrement l'image dans la seconde vue.

Figure 55 - Application VR et effet Stereo

Pour bien comprendre comment cela fonctionne, je propose une expérience empirique. Comment se fait-il que dans la réalité on puisse avoir cette impression de profondeur ? Il s'agit de la superposition des images reçues par chaque œil. L'effet Stéréo a pour but de simuler ces deux images. Si vous vous cachez un œil, puis l'autre, vous constaterez que vous voyez la même scène, mais légèrement décalée, tout comme les deux vue de l'effet Stéréo.

Mais dédoubler les vues ne suffit pas à créer l'illusion, il faut « remplacer » ce que nos yeux voient par ces deux vues. Pour ce faire, les lentilles permettent de focaliser chacun de nos yeux sur une partie spécifique de l'écran. Cette partie correspond à la vue que l'on souhaite envoyer à l'œil.

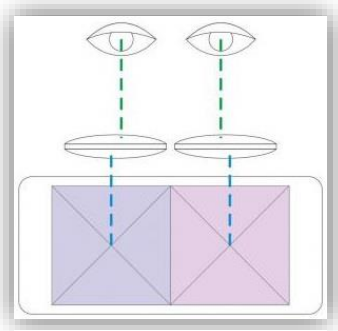


Figure 56 - Fonctionnement d'une Google Cardboard

Afin de créer l'illusion, l'œil ne doit rien voir d'autre que ce qu'on lui envoie. Cela signifie que s'il voit le bord du smartphone, l'intérieur du carton, le contour des lentilles ou que si de la lumière vient s'immiscer, l'immersion est compromise.

En ce qui concerne le développement de l'effet Stéréo, la bibliothèque Three.JS s'en charge. Il suffit de lui donner en entrée une scène 3D dans laquelle on a placé une caméra pour la visualiser et la bibliothèque créera la seconde vue. Des tests ont permis de paramétrer le décalage entre les vues afin d'améliorer l'expérience.

III.2.d. Les contrôles

En ce qui concerne l'application principale, on a opté pour des contrôles à la première personne qui s'utilisent à l'aide du clavier et de la souris. Cependant, en réalité virtuelle, nous n'avons pas accès à ce matériel.

Pour répondre à ce manque, diverses solutions existent. En ce qui concerne le déplacement, j'ai développé cette partie plus loin dans ce rapport. En ce qui concerne l'orientation de la caméra, la réalité virtuelle se base sur le suivi des mouvements de la tête.

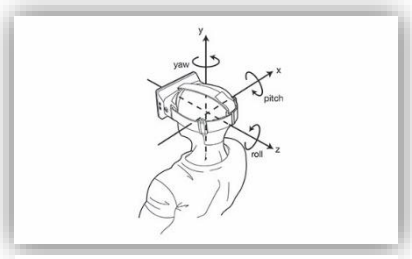


Figure 57 - Head Tracking

Pour ce faire, il s'agit de récupérer en continu les données de position et d'accélération du smartphone, notamment fournis par le gyroscope et l'accéléromètre (d'autres composants peuvent intervenir comme la boussole). Ces données permettent de calculer les mouvements de la caméra afin de suivre les mouvements de la tête de l'utilisateur.

Ce suivi est absolument indispensable et doit être garanti en tout temps. Aucune fonctionnalité ou programme ne doit bloquer le suivi (imaginez une popup dan l'application web). Si la caméra venait à subir des ralentissements ou si la direction qu'elle prend était légèrement erronée, l'immersion serait alors totalement compromise et cela pourrait même provoquer la nausée. Google insiste beaucoup sur ce point, rien dans l'application ne doit empêcher l'utilisateur d'effectuer un geste aussi naturel que de tourner la tête et regarder ailleurs que ce que l'on souhaite lui montrer. Cette liberté fait partie intégrante de la réalité virtuelle.

Enfin, tout comme pour l'effet Stéréo, je n'ai heureusement pas eu à traiter les données de position et d'accélération du mobile, Three.JS implémentant une base des contrôles de la Cardboard. Cependant, les contrôles se limitent à regarder autour de soi et ne proposent donc aucune solution pour se déplacer dans la scène.

III.2.e. Etude des interactions possibles

Si, en réalité virtuelle, nous sommes limités à nos seuls mouvements de tête, quelles solutions s'offrent à nous pour interagir avec l'environnement ?

Cette question a suscité de nombreuses investigations et les solutions se sont avérées être aussi nombreuses que diversifiées. En règle générale, il s'agit de faire preuve d'ingéniosité et de faire avec ce que l'on a sous la main : le principe même de la Google Cardboard.

III.2.e.i. Les composants mobiles

Un smartphone est bourré de pleins de petits composants dont on n'a même pas conscience :

- Le magnétomètre,
- L'accéléromètre,
- Le capteur de pression
- Celui d'humidité
- Celui de lumière
- Celui du bruit ambiant
- Celui de proximité ou encore de température.

On découvrira ainsi que certaines personnes ont tourné à leur avantage l'existence de ces capteurs pour créer autant de possibilités d'interactions.

Les composants mobiles les plus exploités restent néanmoins le gyroscope, l'accéléromètre, le microphone et la caméra.

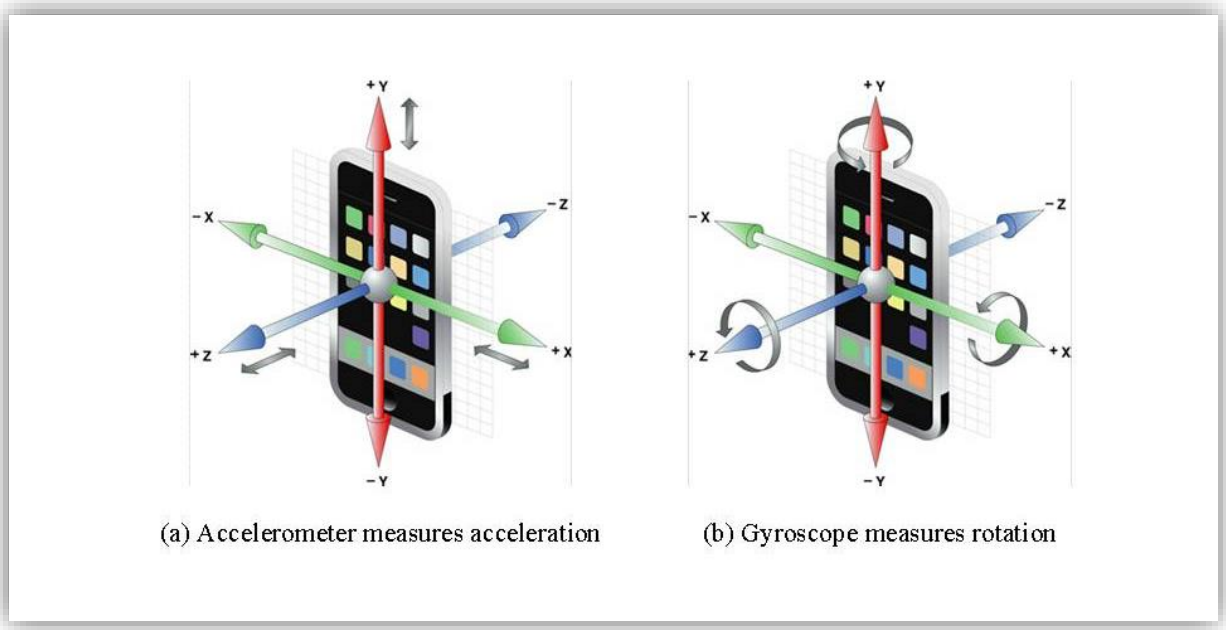


Figure 58 - Ce que mesurent l'accéléromètre et le gyroscope

Le gyroscope permet de détecter la position précise du smartphone. Cela peut permettre, par exemple, de déclencher une action quand le smartphone est dans une certaine inclinaison. Par exemple, regarder ses pieds implique que le smartphone soit à l'horizontale.

L'accéléromètre, quant à lui, permet de détecter un mouvement brusque du mobile. Ainsi, un utilisateur pourrait secouer rapidement sa Google Cardboard pour déclencher une action.

Le microphone peut être utilisé pour détecter une hausse du niveau sonore (un cri par exemple) ou bien des mots et des ordres précis.

Enfin, la caméra, et plus précisément la caméra externe puisque la frontale est enfermé dans l'obscurité de la Cardboard, peut avoir beaucoup d'utilisations. Toute forme de reconnaissance, que ce soit de forme, de couleur, de mouvement, etc. peut être exploitée.

III.2.e.ii. Le magnet

Le magnet présent sur la cardboard est l'unique outil explicitement disponible pour interagir en réalité virtuelle. L'impulsion magnétique qu'il crée peut être détectée par le magnétomètre du mobile afin d'être traitée comme une action. Il s'agit là d'un autre composant mobile souvent exploité.

Figure 59 - Utilisation du magnet

Sur la Google Cardboard V2 (deuxième version), le magnet est remplacé par un bouton capacitatif qui utilise l'induction pour simuler un toucher à l'écran.

III.2.e.iii. En jeu

La dernière manière d'interagir avec son environnement demeure la plus naturelle : directement dans le monde virtuel à l'aide de son unique regard. Les possibilités d'interactions se limitent à notre imagination. On peut imaginer une infinité de combinaisons, de la plus simple à la plus complexe, permettant d'effectuer des actions diverses avec l'environnement virtuel.

Figure 60 - Interaction "in game", fixation d'un objet

Communément, fixer un élément pendant un certain temps pourrait signifier que l'on souhaite interagir avec celui-ci (ouvrir une porte, prendre un objet, etc.). De même, j'ai pu expérimenter des applications où fixer ses pieds pendant deux secondes permettait d'ouvrir un menu dans lequel on pouvait fixer les options disponibles, et d'autres où fixer ses pieds permettait de démarrer ou d'arrêter le déplacement.

III.2.f. Problèmes de compatibilité

L'étude des différentes possibilités d'interaction avait principalement pour but de déterminer quelle était la manière façon de se déplacer dans l'application. Mais elle avait également pour but de déterminer dans quelle mesure nous pourrons adapter les fonctionnalités de l'application principale à la Google Cardboard.

Les possibilités d'interaction les plus prometteuses ont cependant été tuées dans l'œuf pour une raison malheureuse : la compatibilité.

La Google Cardboard est prévue pour être utilisée avec des applications mobiles développées en Android et proposées en téléchargement sur la boutique. Une application mobile de ce genre a accès à tous les composants mobiles et peut les exploiter. Par exemple, les applications mobiles de météo peuvent exploiter le capteur de température du mobile.

Un navigateur est une application mobile et a donc accès à ces composants.

Cependant notre application web 3D n'est pas une application mobile. Pour communiquer avec les composants mobiles, il faut nécessairement passer par l'intermédiaire du navigateur qui, lui, a accès aux composants. Et cela n'est possible que si un standard ou une API a été développé afin de permettre à une application web (développée en Javascript par exemple) de communiquer avec tel ou tel composant mobile.

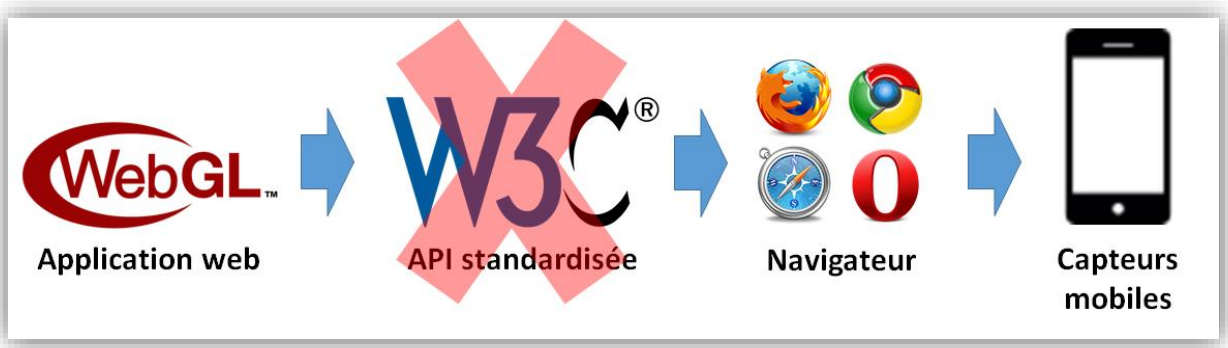


Figure 61 - Processus intermédiaires permettant à une application web de communiquer avec les capteurs mobiles

Malheureusement, très peu de composants sont accessibles pour l'instant puisque ces API sont des spécifications en cours de développement. Il y a beaucoup d'engouement autour du développement d'API web permettant d'exploiter les composants mobiles car ces dernières années, un fort besoin se fait ressentir. Le World Wide Web Consortium (W3C) a en charge de développer les standards de la technologie HTML qui permettrait la communication avec les différents composants mobiles.

On retrouve ici la contrainte technologique liée aux nouvelles technologies. Si de nos jours la plupart des navigateurs web supportent WebGL et permettent le développement d'applications 3D, la communication avec les composants mobiles n'a jamais semblé très utile. Cependant, les utilisations que peut avoir un smartphone se multiplient de jour en jour, les capteurs sont de plus en plus exploités par les applications et c'est pourquoi, ce standard est dans la ligne de mire de W3C.

Enfin, cela diminue grandement le nombre de possibilités d'interaction. Cependant il nous paraissait indispensable de pouvoir se déplacer et nous avons donc cherché à utiliser au moins une unique interaction. Pour les raisons citées précédemment, le magnet n'est plus exploitable puisqu'il dépend du magnétomètre. Heureusement, certaines API existent pour les composants les plus utilisés tels que le GPS. Il me restait donc le choix entre activer le déplacement à l'aide du regard dans le monde virtuel ou explorer les API disponibles.

La première solution, pour avoir expérimenté des applications VR utilisant ce principe pour se déplacer, ne m'a absolument pas convaincu. Le déplacement ne se fait pas à l'aide du regard, ce n'est pas naturel. J'ai donc orienté mes recherches sur les API disponibles pour les composants les plus connus tels que la caméra ou le microphone.

Finalement, l'utilisation du microphone a été écartée pour plusieurs raisons et celle de la caméra a été retenue. D'abord parce que l'expérience dépendrait alors du niveau sonore de la pièce et que le Planétarium accueille notamment des enfants. Ensuite parce qu'une

reconnaissance vocale précise est trop complexe pour ce que l'on souhaite en faire. Son utilisation reste néanmoins à prendre en compte pour de futures recherches puisqu'elle pourrait s'appliquer à des cas spécifiques.

III.2.g. Utilisation de la caméra

III.2.g.i. Récupération du flux

A ma grande surprise, il n'existait aucune API complète permettant la communication avec la caméra ou le microphone.

Par défaut, tout navigateur a accès par défaut au premier flux caméra du mobile, et ce à l'aide de diverses API. Cependant, il s'agit de la webcam qui ne nous est d'aucune utilité. Après maintes recherches, seules deux API se sont révélées avoir accès à la caméra externe.

L'API Stream est une API très bien supportée par la plupart des navigateurs. Elle fournit une méthode appelée getUserMedia qui permet de récupérer tous les flux caméra du mobile. Cependant, je n'ai aucun contrôle sur le flux camera, tout est encapsulé. Après avoir longuement cherché à détourner l'API et à contourner son utilisation pour récupérer le flux et l'étudier, j'ai dû abandonner.

L'API MediaStreamTrack permet quant à elle de récupérer le flux vidéo de manière incertaine. C'est-à-dire qu'elle récupère tous les flux et qu'on ne peut pas réellement savoir si celui qu'on manipule sera la webcam ou la caméra externe. 100% des tests effectués ont montré que le second flux représentait le flux de la caméra externe, cependant, mes tests ne sont pas exhaustifs. Le principal inconvénient que présente cette API est sa compatibilité. Elle est en théorie supportée par la plupart des navigateurs, cependant, cette compatibilité n'est que partielle. La méthode permettant de récupérer les flux n'est supportée que par le navigateur Google Chrome.

Finalement, bien que peu agréable, l'API MediaStreamTrack s'est avérée le meilleur compromis possible à l'heure actuelle, l'idée d'abandonner la caméra n'étant pas envisageable. Son utilisation implique donc de sacrifier la compatibilité avec la plupart des navigateurs, ce qui rentre en conflit avec notre objectif de portabilité et d'accessibilité. Cette solution est donc classée comme temporaire dans l'attente du développement des spécifications prévues par W3C.

III.2.g.ii. Possibilités d'utilisation

Une fois le flux de la caméra externe récupéré, comment l'utiliser ?

Les solutions sont multiples, mais il faut songer au coût en ressources que cela représente. En effet, analyser en continu le flux caméra c'est procéder quatre fois ou plus par seconde à un scan de celui-ci, d'appliquer un algorithme plus ou moins complexe et coûteux à chacun des pixels (qui sont au nombre de 2 073 600 pour un écran Full HD), le tout afin d'obtenir un résultat instantané nous indiquant si l'on a détecté ou non un signe de l'utilisateur.

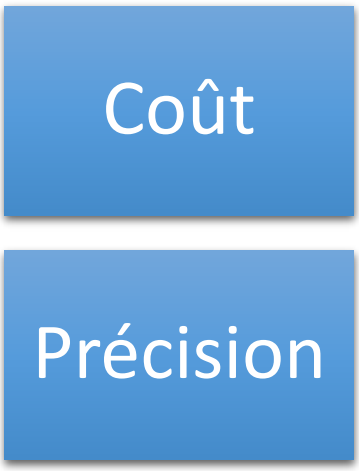


Figure 62 - Les deux facteurs principaux pour déterminer le type de détection

De ce fait, il s'agit de déterminer le meilleur compromis entre précision et coût en ressources.

Les différentes détections envisagées étaient :

- La détection de mouvement
- La détection d'obscurité
- La détection de couleur

La détection de mouvement implique des algorithmes trop coûteux en ressources tandis que la détection d'obscurité a été un échec lorsqu'elle a été testée. C'est pourquoi je me suis tourné vers la détection de couleur qui s'est avérée être un succès.

III.2.g.iii. Détection de couleur

La détection de couleur se base sur un principe simple. Une fois le flux caméra récupéré, il s'agit d'analyser une zone représentative de l'image, de relever ses valeurs de rouge, de vert ou de bleu et si la valeur moyenne dépasse un certain seuil, alors on valide la présence de la couleur.

Afin de rendre l'opération la moins coûteuse en ressources possibles, la couleur à détecter doit être l'une des trois couleurs primaires en informatique. S'il s'agissait d'un mélange d'entre elles, cela multiplierait le nombre de comparaisons à effectuer. Evidemment, dans la réalité, la couleur n'est pas parfaite : si l'on passe une couleur rouge face à la caméra, les valeurs de rouge seront variables. C'est pourquoi il faut déterminer un seuil indiquant que si le rouge est la couleur prédominante du scan, alors on suppose que l'utilisateur a volontairement fait passer cette couleur devant la caméra pour déclencher une action.

La zone à analyser nécessiterait quant à elle une R&D en soi. Il s'agit de déterminer une zone de l'image qui soit à la fois suffisamment étendue pour ne manquer aucun détail et à la fois suffisamment restreinte pour minimiser le nombre de calculs. La zone en elle-même doit être représentée par des formes géométriques simples pour éviter de devoir calculer la position de la zone en plus (une zone circulaire ne conviendrait pas).

La détection de couleur étant une solution temporaire, j'ai donc décidé d'une zone par défaut au lieu de mener une étude sur la zone la plus adaptée. Le scan s'effectue donc sur un carré centré dans l'image.

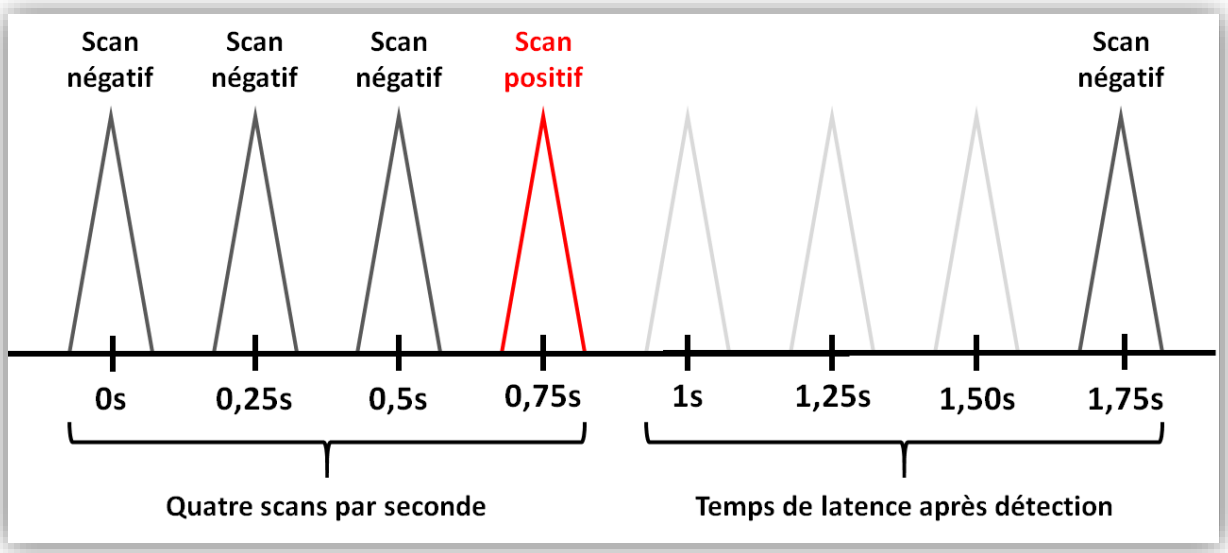


Figure 63 - Schéma de fonctionnement des scans

Enfin, il a fallu rendre cette détection fluide : il n'était pas envisageable de devoir repasser plusieurs fois la couleur pour activer l'action ou bien que l'action s'active plusieurs fois à cause d'un seul passage. Les tests ont permis de déterminer qu'un scan toutes les 250ms était suffisant pour détecter sans temps de latence une couleur (et donc activer instantanément le déplacement) et qu'après chaque détection validée, arrêter les scans pendant 1s permettait de ne pas détecter à nouveau la même couleur (et donc d'annuler le déplacement tout juste démarré).

III.2.g.iv. Déplacement

A présent que l'utilisateur peut interagir avec la Google Cardboard, il s'agit de relier cette interaction à une action : le déplacement. La solution la plus naturelle aurait était d'utiliser le magnet pour démarrer et arrêter le déplacement.

Pour simuler le déplacement, il s'agit de faire avancer la caméra dans la direction qu'elle regarde. Pour ce faire, il faut simplement récupérer la normale au plan de la caméra, puis, de déplacer la caméra selon ce vecteur de direction obtenu.

Bien que Three.JS soit bien documentée, certaines méthodes ont été oubliées. En fouillant le code source de la bibliothèque, j'ai pu dénicher une méthode permettant d'obtenir automatiquement ce vecteur de direction. Il n'était pas compliqué de la développer soi-même mais il vaut mieux ne pas réécrire ce qui existe déjà et utiliser une méthode optimisée. C'est donc encore via la bibliothèque Three.JS que le déplacement va se faire puisqu'une fois le vecteur obtenu, il suffit de modifier le vecteur de position de la caméra.

Puisqu'en réalité virtuelle, rien ne doit empêcher le suivi des mouvements de la tête, il est tout à fait possible de regarder autour de soi en cours de déplacement. De ce fait, lorsque le déplacement est activé, il est possible de s'orienter dans n'importe quelle direction en orientant son regard.

III.2.h. Chargement des données

Dernier point concernant la Google Cardboard, il n'est pas envisageable de demander à l'utilisateur de rechercher ses fichiers de données sur sa machine. Les fichiers de données sont trop volumineux pour être stockés sur un mobile.

Les données doivent donc être hébergées sur un serveur et récupérées à l'aide de requêtes HTTP. On notera qu'il est important de prévenir l'utilisateur avant qu'il ne télécharge les données : celui-ci sera averti qu'il est conseillé d'être en connexion Wi-Fi puisque les données sont volumineuses.

Une fois les fichiers hébergés, deux solutions se sont offertes à moi :

- Soit on charge automatiquement les données au lancement du mode Cardboard si un appareil mobile est détecté
- Soit on propose à l'utilisateur une interface lui permettant de choisir les données qu'il souhaite visualiser parmi une sélection

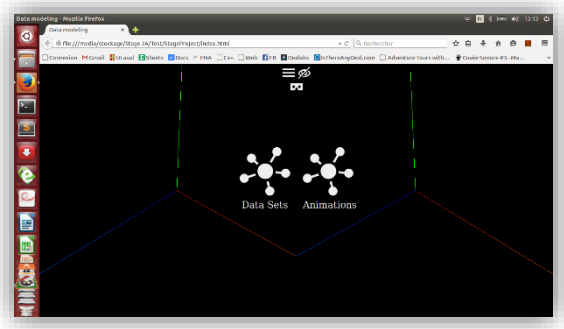


Figure 64 - Sélection de jeux de données préenregistrés (Voir Annexe)

Les deux solutions ont été successivement implémentées. Une fois les données récupérées par la méthode GET (requête au serveur), il s'agit de parser le fichier et de distribuer les bonnes données aux bons buffers.

Pour ce faire, j'ai réutilisé ce qui existait déjà dans le chargement local de fichier, la seule différence étant la manière de récupérer les fichiers (local d'une part, serveur d'autre part).

En ce qui concerne la seconde solution, il a fallu améliorer le menu en proposant une nouvelle interface en mode Cardboard. Celle-ci permet de masquer l'interface générale, qui peut rapidement être gênante sur l'écran d'un mobile, d'ouvrir le menu (puisque sur mobile on ne peut utiliser le clavier pour l'ouvrir) et d'ouvrir un autre menu permettant de sélectionner le jeu de données à charger.

Figure 65 - Interface Cardboard minimaliste

Arrivant au bout du stage, cette dernière fonctionnalité n’a cependant pas été totalement développée.

IV. BILAN

IV.1. Résultat du projet

Globalement, le projet a bien avancé. Il renferme beaucoup de nouvelles fonctionnalités, emploie des structures de données spécifiques pour optimiser l'affichage des données et s'organise proprement autour d'une architecture orientée objet. Il dispose également d'une interface très fournie et d'un début de solution concernant la réalité virtuelle.

L'apprivoisement de nouvelles technologies et le remaniement du projet nous aura pris un temps précieux que l'on aurait pu investir pour aller plus loin dans le projet. Mais il s'agissait là de tâches nécessaires évidemment.

En ce qui concerne le mode Cardboard, s'il est fonctionnel, il n'est cependant pas près pour une utilisation sur le terrain. D'une part parce que les solutions implémentées sont temporaires pour cause de nombreuses contraintes de compatibilité et d'autre part parce qu'il n'est pas complet au niveau des fonctionnalités. Il reste encore beaucoup à faire pour que la réalité virtuelle puisse avoir une réelle utilité dans ce projet.

IV.2. Difficultés rencontrées

Parmi les difficultés rencontrées au cours de ce stage, je retiendrai notamment celle de la compatibilité liée au manque de documentation. En effet, non content d'avoir une compatibilité très restreinte, le statut de nouvelle technologie de la réalité virtuelle fait qu'il est difficile de trouver des sources d'information permettant de savoir clairement s'il existe une solution ou non à tel ou tel problème rencontré. En particulier la récupération du flux de la caméra s'est révélée bien plus fastidieuse que je ne le pensais.

Ensuite, on a temporairement rencontré la difficulté de travailler en équipe sur le même projet. Au début du stage, il nous est arrivé à plusieurs reprises d'avoir à travailler sur les mêmes fichiers, ce qui pouvait entraîner des conflits lorsqu'il fallait regrouper les travaux. Cette difficulté s'est fait rare une fois la structure objet mise en place.

Enfin, reprendre un projet déjà commencé et comprendre son fonctionnement a pris du temps, mais la restructuration du code et le remaniement total du projet alors que nous n'étions pas les seuls auteurs a pris également beaucoup de temps.

IV.3. Expérience professionnelle acquise

Ce stage ayant été diamétralement opposé à celui de l'an dernier, j'ai beaucoup appris lors de celui-ci puisque je découvrais, à nouveau, tout.

Dans un premier temps, c'est la première fois que j'ai pu travailler en équipe dans une réelle situation de travail. Cela m'a appris d'une part à partager les tâches et d'autre part à utiliser les outils prévus à cet effet tels que le gestionnaire de version GitHub.

Dans un deuxième temps, j'ai pu apprécier une nouvelle facette du monde du travail, celui de la recherche. Si lors de mon précédent stage j'ai suivi un cahier des charges strictement établi, ici j'ai appris à rechercher et comparer des solutions, à mener des investigations dans des domaines encore peu documentés, à faire mes propre choix de développement. Par conséquent, outre le monde de la recherche, c'est particulièrement l'activité R&D qui a été nouvelle pour moi.

Dans un troisième temps, j'ai appris à exploiter WebGL et le pipeline graphique, ce qui vient étoffer mes compétences en infographie à l'aide d'une technologie moderne.

Enfin, j'ai découvert le domaine de la réalité virtuelle, j'ai appris à utiliser ses outils et j'ai pu développer des solutions. J'ai donc acquis de nouvelles compétences et un nouveau domaine s'ouvre à moi.

IV.4. Conclusion

Outre le fait de m'avoir grandement apporté en compétences techniques, ce stage m'aura fait découvrir une nouvelle passion. Le domaine de la réalité virtuelle est un domaine que je compte exploiter dans un futur très proche puisque je vois à présent les possibilités qu'elle offre.

Ce stage m'a donc permis d'orienter mes projets professionnels qui étaient jusque-là plutôt vagues. Le domaine de la 3D, plus généralement, me passionne.

S'adapter à un nouveau milieu et m'intégrer aux équipes en place ne m'a pas semblé spécialement difficile. Je n'ai d'ailleurs jamais ressenti mon statut de stagiaire tant j'ai été bien accueilli à l'Observatoire.

Etant quelqu'un de très individuel, travailler de concert m'a beaucoup apporté également et m'a appris à faire des concessions.

Finalement, je garde un excellent souvenir de ce stage, tant pour le projet passionnant sur lequel j'ai eu la chance de travailler que pour le contact que j'ai pu avoir avec le personnel. Il m'aura permis d'avoir une vision précise de ce que je souhaite faire plus tard alors que j'étais dans le flou depuis longtemps.

GLOSSAIRE

CNRS : Centre National de Recherche Scientifique, plus grand organisme public français de recherche scientifique

CDS : Centre de Données astronomiques de Strasbourg, infrastructure de recherche du CNRS

UMR : Unité Mixte de Recherche, entité administrative associant un établissement d'enseignement supérieur au CNRS

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

CSS : Cascading Style Sheets, langage informatique qui décrit la présentation des documents HTML

WC3 : World Wide Web Consortium, organisme de standardisation à but non lucratif chargé de promouvoir la compatibilité des technologies du web

VR : Virtual Reality, sigle anglophone désignant la réalité virtuelle

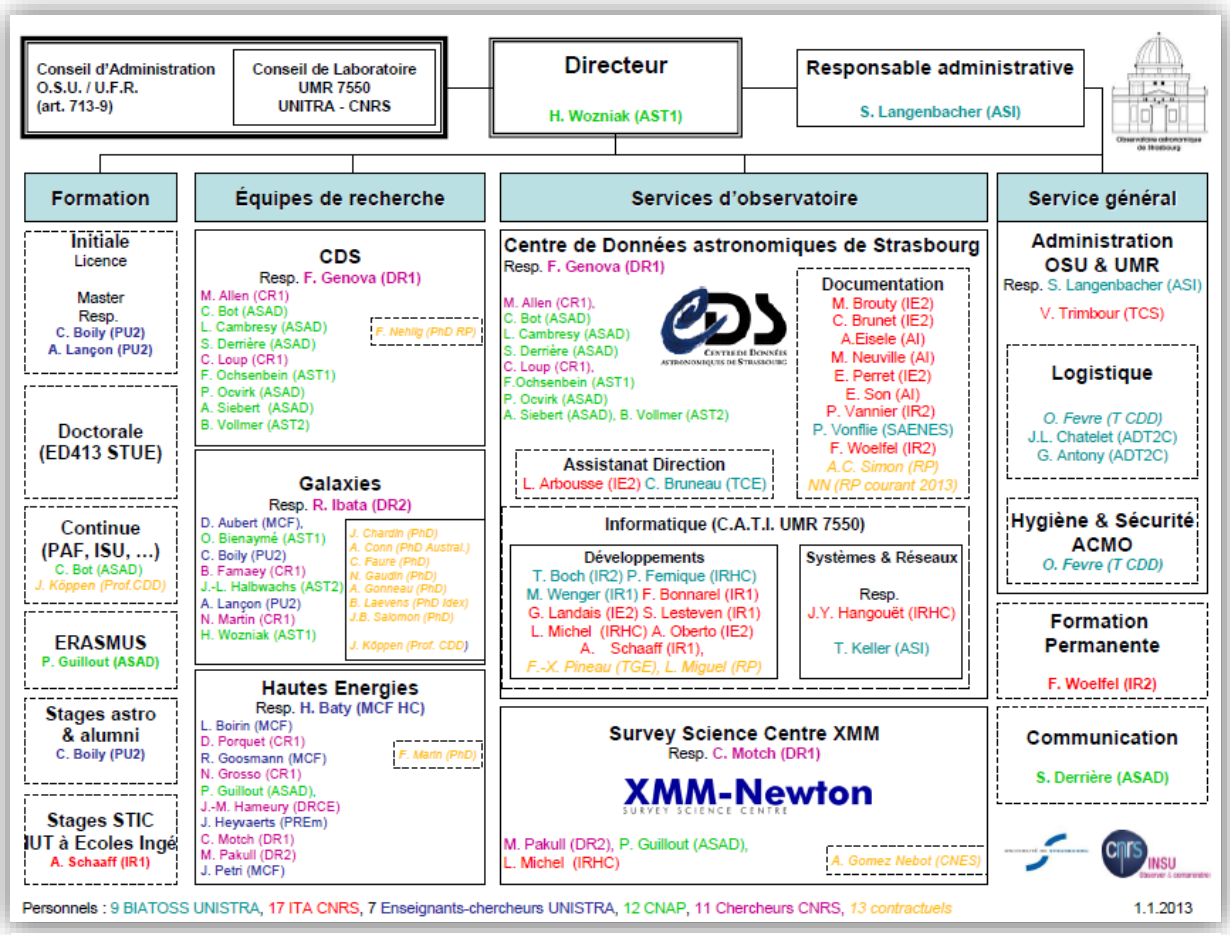
FPS : First Person Shooter, ou tir à la première personne, désigne la vue à la première personne majoritairement utilisée dans les jeux de tir

Big Data : Désigne le domaine concernant les grandes masses de données et leur exploitation

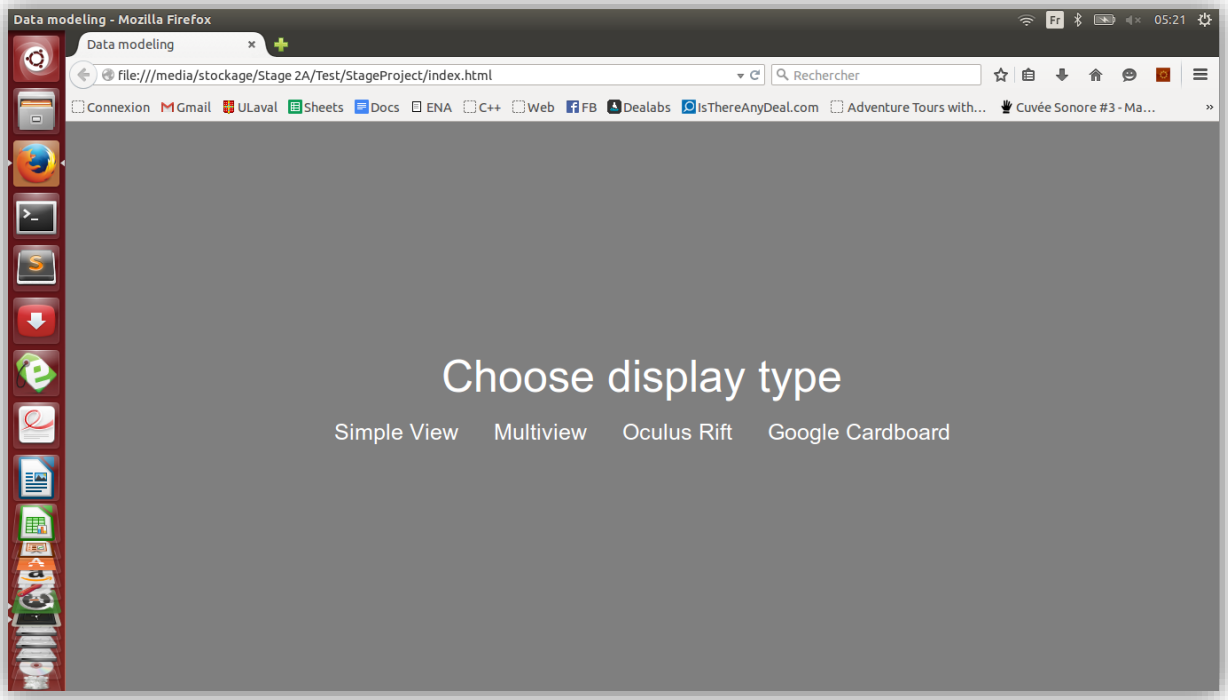
Scène 3D : Ensemble de plusieurs objets créés en trois dimensions

Caméra : Point dans l'espace à partir duquel la scène est visée, il détermine sous quel angle on visualise la scène

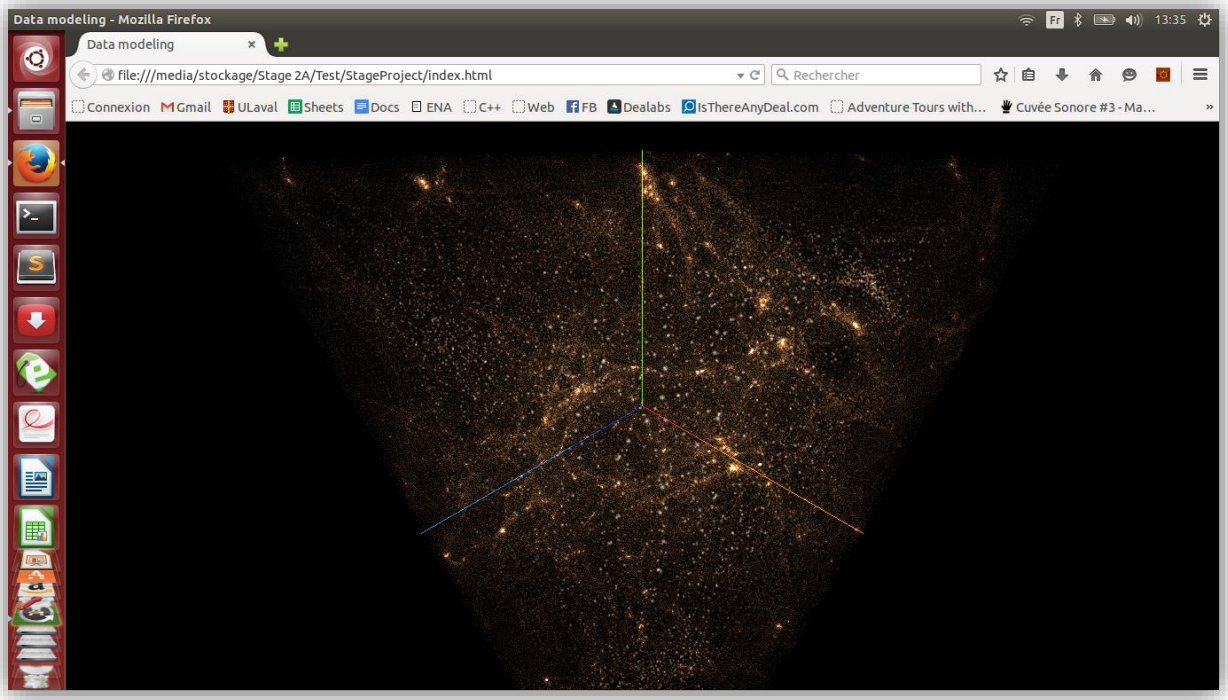
ANNEXE



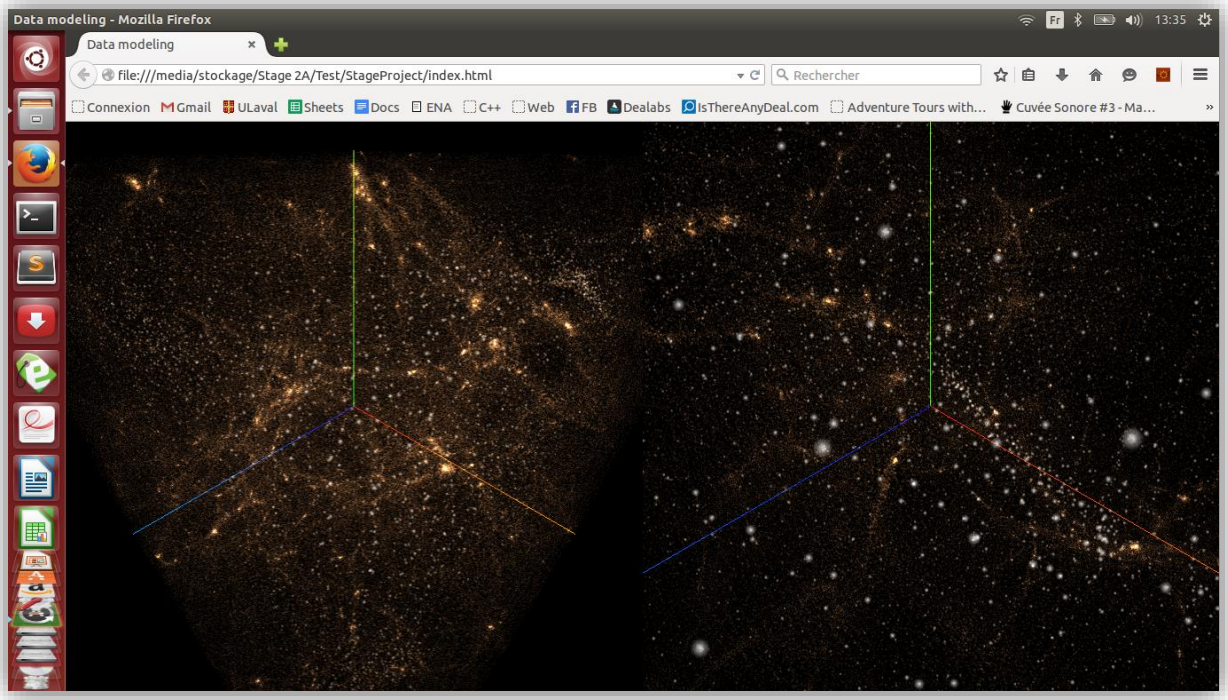
Annexe 1 - Organigramme de l'Observatoire



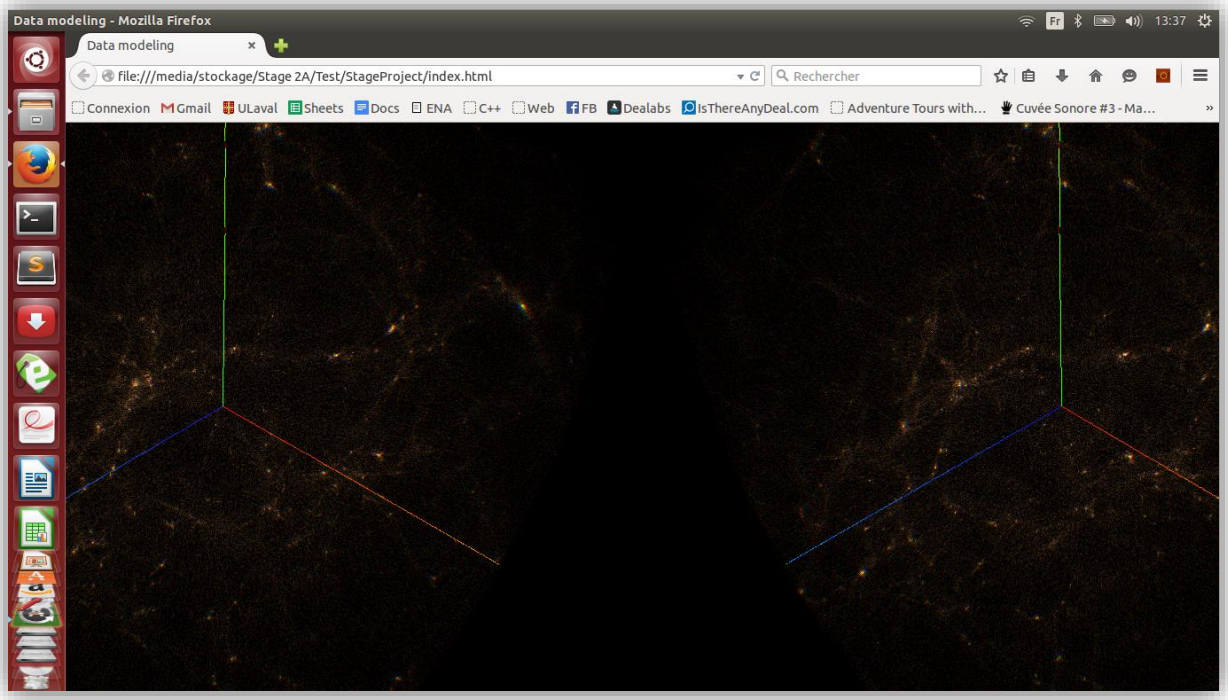
Annexe 2 - Interface du menu



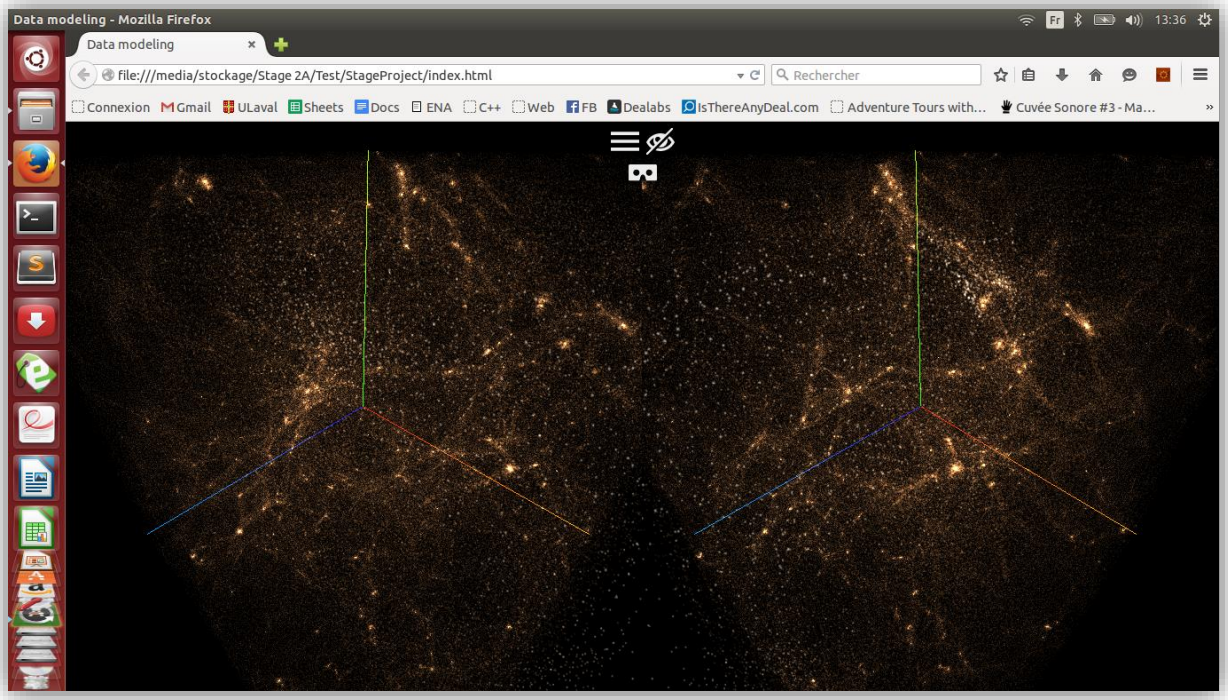
Annexe 3 - Mode SimpleView



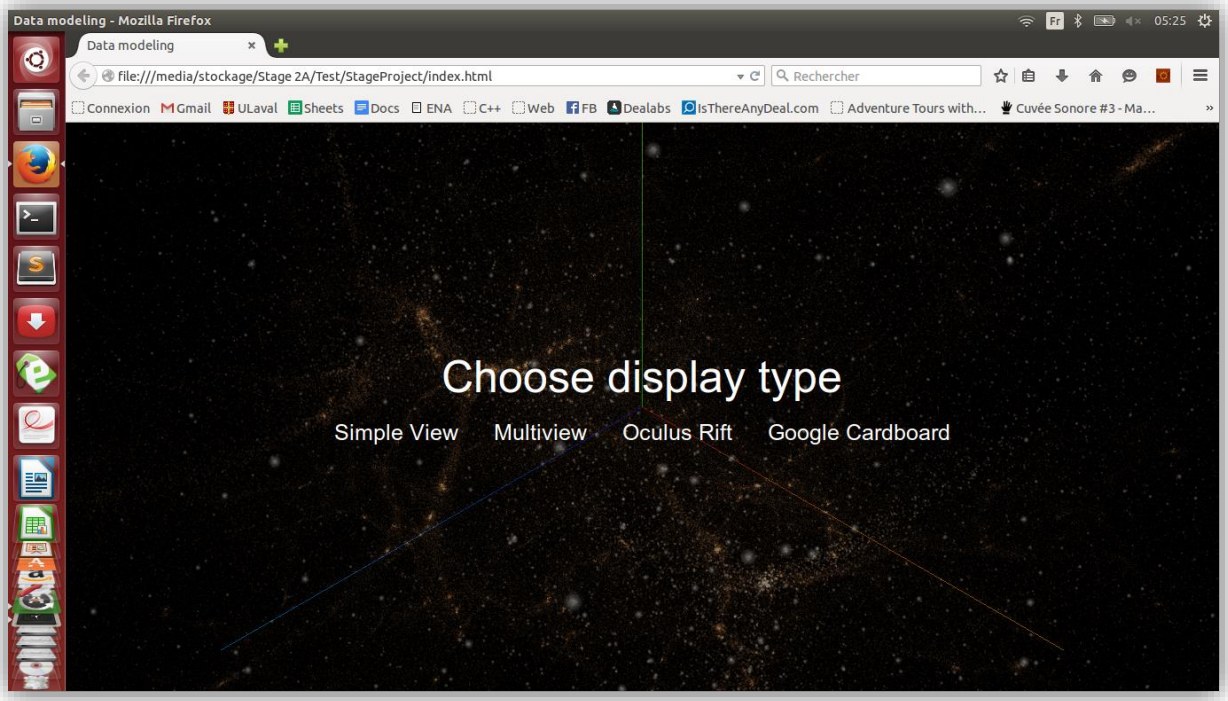
Annexe 4 - Mode MultiView



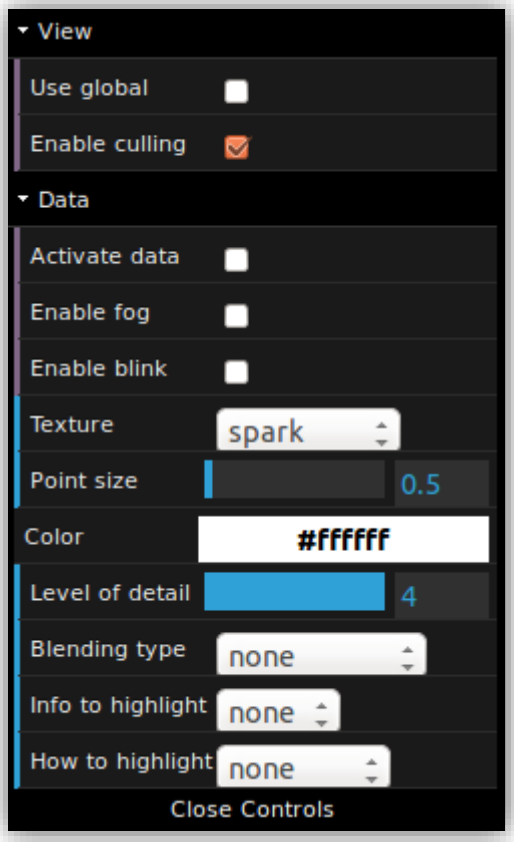
Annexe 5 - Mode MultiView



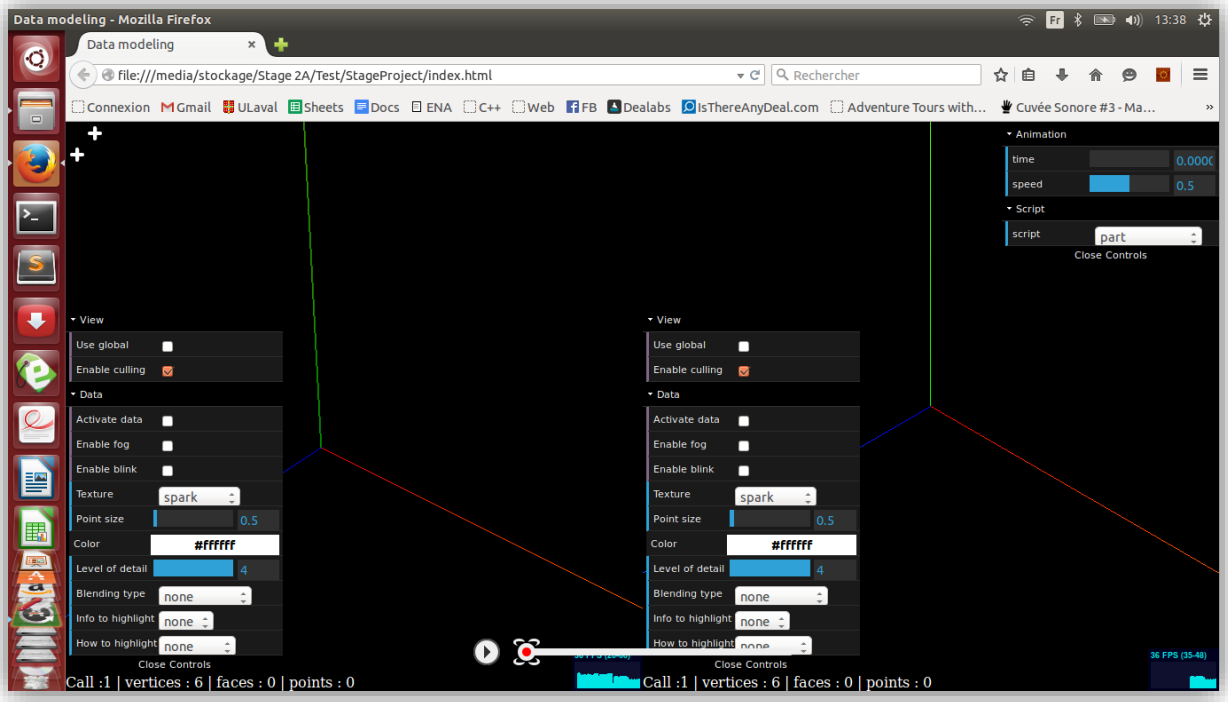
Annexe 6 - Mode Google Cardboard



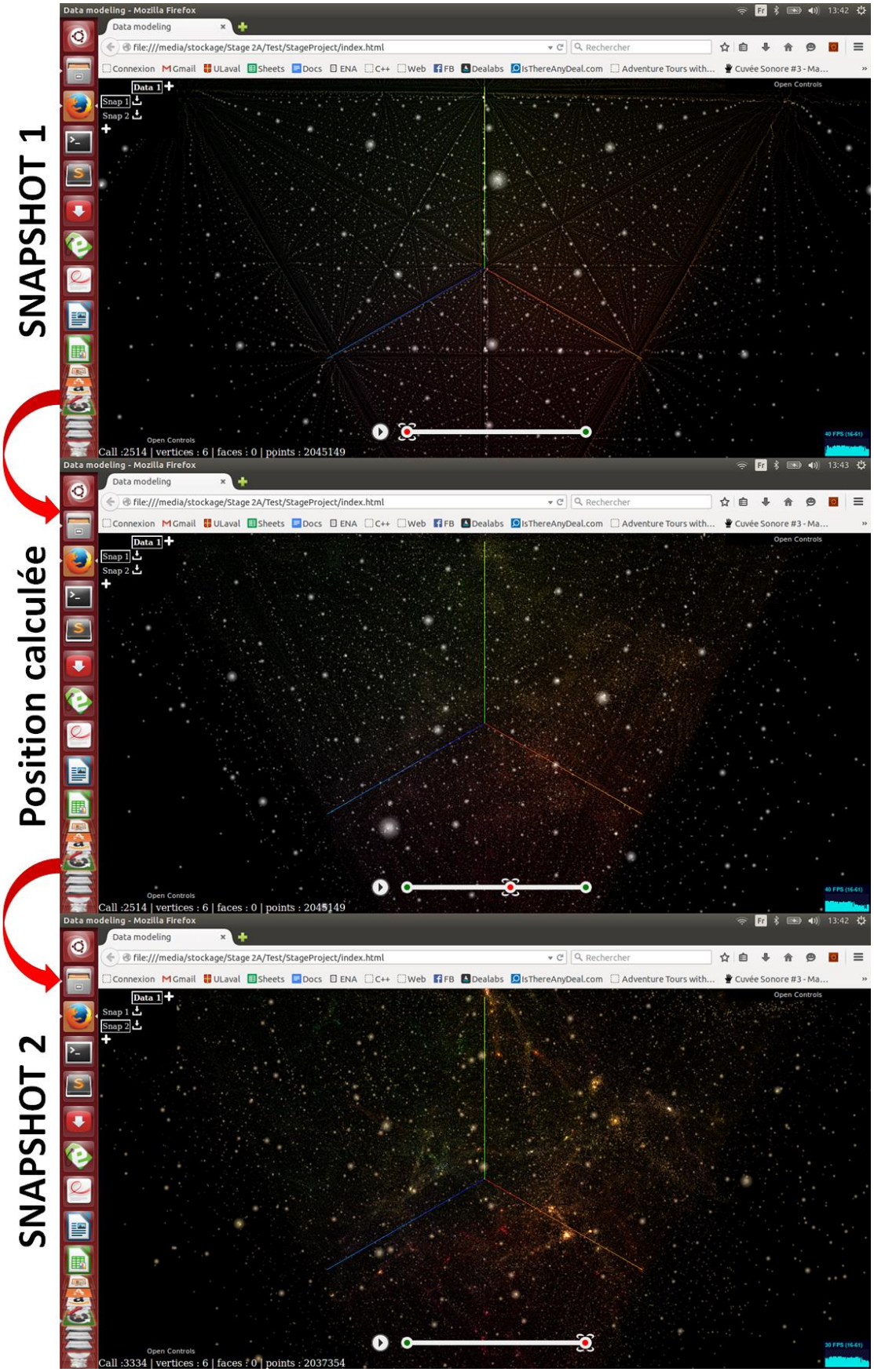
Annexe 7 - Interface du menu avec le rendu en arrière-plan



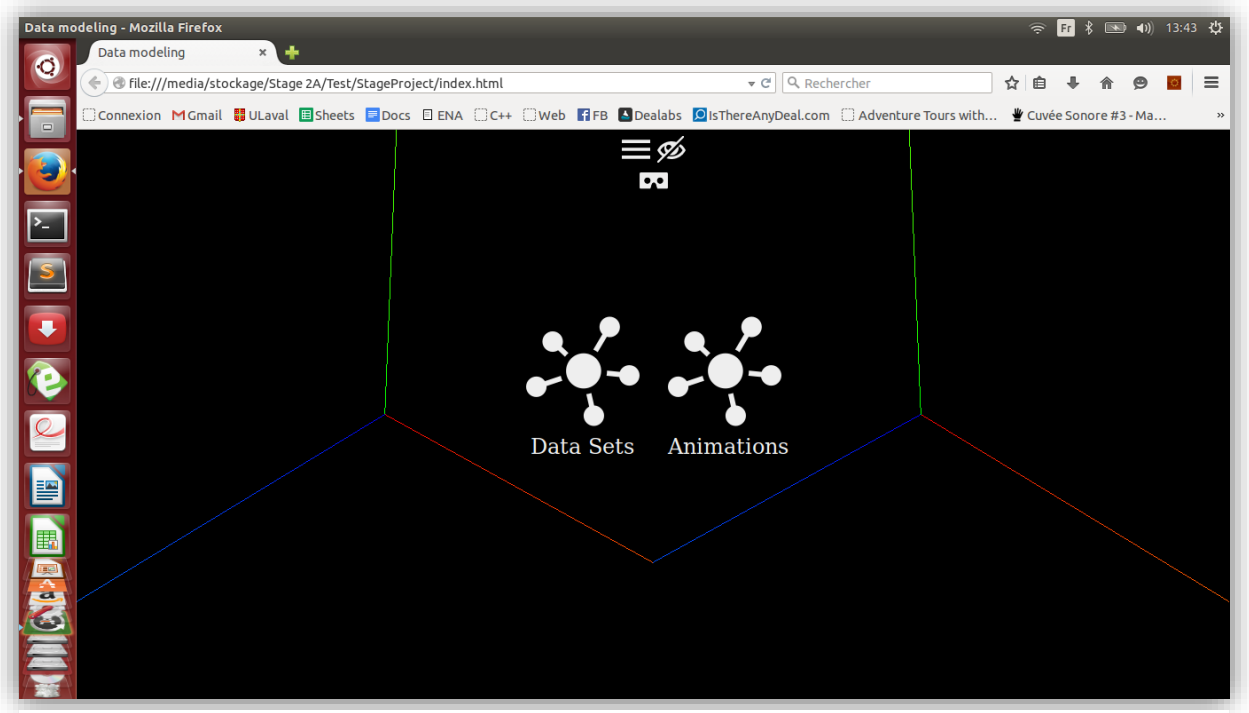
Annexe 8 – Zoom sur l'interface Three.JS



Annexe 9 - Interface Three.JS appliquée sur deux vues indépendantes



Annexe 10 - Evolution du temps entre deux snapshots



Annexe 11 - Sélection de jeux de données préenregistrés