IUT Nancy Charlemagne
Université de Lorraine
2 ter Boulevard Charlemagne
54052 Nancy Cedex
Dépt.Informatique

Observatoire astronomique de Strasbourg

Réalité Virtuelle appliquée aux services et aux données astronomiques du CDS (utilisation du kit Oculus VR)

Rapport de stage DUT Informatique

Observatoire astronomique de Strasbourg
Centre de données astronomiques de Strasbourg (CDS)

Joffrey DA ROCHA
2013-2014

IUT Nancy Charlemagne
Université de Lorraine
2 ter Boulevard Charlemagne
54052 Nancy Cedex
Dépt. Informatique

Réalité Virtuelle appliquée aux services et aux données astronomiques du CDS (utilisation du kit Oculus VR)

Rapport de stage DUT Informatique

Observatoire astronomique de Strasbourg
Centre de données astronomiques de Strasbourg (CDS)

Joffrey DA ROCHA

André SCHAAFF
Bernard MANGEOL

Remerciements

Je tiens à remercier particulièrement M.Schaaff pour son encadrement, sa disponibilité malgré ses responsabilités et l'intérêt qu'il a porté à mon travail.

Je remercie toute l'équipe du centre de données astronomiques de Strasbourg pour m'avoir accueilli chaleureusement dans leurs locaux et m'avoir assuré toutes les conditions afin que le déroulement de mon stage se passe de la meilleure manière possible, en particulier la présentation durant la première semaine de tous les services proposés par l'observatoire ainsi que leur fonctionnement, me permettant ainsi d'être immédiatement immergé dans mon environnement de travail et de commencer sur de bonnes bases.

Merci également au cadre professoral de l'IUT Nancy-Charlemagne pour la formation de qualité qu'il assure. En espérant que le travail réalisé soit en adéquation avec les espérances et les attentes de mon encadrant.

Table des matières

Introduction

Etant actuellement en deuxième année de DUT Informatique, un stage de 10 semaines était obligatoire afin de valider mon diplôme. Voulant continuer mes études une fois ce dernier obtenu, j'ai orienté ma recherche de stage vers la R&D et c'est grâce aux annonces de stage publiées sur Arche que j'ai pu trouver ce que je cherchais. En effet l'Observatoire de Strasbourg propose chaque année de nombreux stages pour les DUT et LP de l'IUT Nancy-Charlemagne, ce qui m'a permis de choisir le sujet qui m'intéressait le plus.

La réalité virtuelle étant en vogue dernièrement et n'ayant pas eu l'occasion de m'y confronter, j'ai pensé que pouvoir travailler sur ce sujet pendant mon stage serait une bonne expérience et me permettrait d'acquérir de nouvelles connaissances ainsi que de mieux connaitre cette nouvelle technologie.

Au vu du nombre de services proposés par le CDS et les domaines dans lesquels il se conforte, l'application de la réalité virtuelle a donc été envisagée. En effet, la visualisation du sky-atlas au travers de l'Oculus par exemple, pourrait proposer une nouvelle approche dans l'utilisation des services. L'application de la réalité virtuelle pourrait être également une nouvelle manière pour aborder un public tel que les personnes visitant le planétarium de manière à rendre leur expérience encore plus intéressante et ludique, permettant également de viser un public plus jeune.

Présentation de l'entreprise

Présentation générale

L'Observatoire Astronomique de Strasbourg est un Observatoire des Sciences de l'Univers (OSU), un UFR de l'Université de Strasbourg et une Unité Mixte de Recherche (UMR 7550) entre l'Université et le CNRS. Il est actuellement dirigé par Hervé WOZNIAK. L'Observatoire avait pour vocation initiale l'astronomie de position et l'observation de comètes, de météorites et d'étoiles variables. Plus tard, on s'y intéressa à la photométrie et à l'observation d'étoiles doubles. Ce n'est qu'à partir de 1965 que les astronomes comprirent que l'observation au sol avait atteint ses limites, et proposèrent le concept du Satellite Hipparcos à l'Agence Spatiale Européenne. Historiquement, il y a eu trois Observatoires Astronomiques. Le premier fut construit en 1673 sur les tours d'enceinte de la ville, avec l'aide de l'astronome Julius REICHELT. Le second a quant à lui été bâti en 1828 sur les bâtiments de l'Académie de Strasbourg. L'Observatoire Astronomique obtint son emplacement actuel en 1881, année de son inauguration. Celui-ci fait partie, avec le jardin botanique situé juste à côté, des éléments qui symbolisent la décision politique de faire de Strasbourg une vitrine, prise par l'empereur Guillaume 1er d'Allemagne à la suite de la guerre de 1870.

Il est composé de trois bâtiments dont le plus célèbre est celui de la Grande Coupole qui accueille la 3e plus grande lunette de France par sa taille. L'Observatoire héberge également le Planétarium de Strasbourg dont il a eu la responsabilité de 1986 à 2008. Celui-ci est désormais intégré au Jardin des Sciences de l'Université de Strasbourg. L'Observatoire dispose également d'un riche patrimoine d'instruments et d'ouvrages anciens. Il est structuré en trois équipes de recherche et deux Services d'Observation de l'Institut National des Sciences de l'Univers (INSU), le Survey Science Centre d'XMM-Newton (SSCXMM) et le Centre de Données astronomiques de Strasbourg (CDS). Son statut d'OSU (voir décret n°85-657 du 27 juin 1985) place l'Observatoire astronomique au cœur du dispositif national mis en œuvre par l'INSU. Par conséquent, l'Observatoire astronomique de Strasbourg a pour mission de contribuer aux progrès de la connaissance par :

- l'acquisition de données d'observation
- le développement et l'exploitation de moyens appropriés
- l'élaboration des outils théoriques nécessaires

L'observatoire est également chargé:

- de fournir des services liés à son activité de recherche
- d'assurer la formation des étudiants et du personnel de recherche
- d'assurer la diffusion des connaissances
- de prendre part à des activités de coopération internationale

Présentation des équipes de recherche

Galaxies

L'équipe « Galaxies » étudie la formation et l'évolution des galaxies au travers de leurs populations stellaires, ainsi que de la dynamique des étoiles et de la matière noire. Elle est impliquée dans la préparation de la mission satellitaire astrométrique Gaia de l'Agence Spatiale Européenne, dont le lancement est prévu pour octobre prochain, ainsi que dans le grand relevé cinématique RAVE, qui fournira des données d'une précision inédite sur la Voie Lactée. Les recherches sont concentrées sur les caractéristiques et les propriétés physiques de notre galaxie et de ses plus proches voisines. Afin de pouvoir étudier la dynamique gravitationnelle qui régit les mouvements du gaz et des étoiles, on analyse les évolutions de la population stellaire de ces galaxies. Les résultats obtenus permettent de reconstituer les événements clés de la vie d'une galaxie. L'équipe mène également d'autres recherches sur les amas stellaires, qui sont des composants fondamentaux d'une galaxie. Grâce aux connaissances accumulées concernant les mouvements gravitationnels des étoiles, il est possible de simuler numériquement l'évolution d'un amas stellaire, afin de mieux comprendre la formation des galaxies.

Hautes Énergies

L'équipe « Hautes Énergies » s'intéresse aux sources galactiques et extragalactiques émettrices en rayons X, objets compacts (étoiles à neutrons, naines blanches, etc.) et noyaux actifs de galaxies. Elle est impliquée dans le SSC-XMM, un consortium international de laboratoires sélectionnés par l'ESA, labellisé par l'INSU comme Service d'Observation, qui est en charge de fournir des catalogues complets de sources X observées par le satellite XMM-Newton à la communauté internationale.

CDS

Le Centre de Données astronomiques de Strasbourg (CDS) est à la fois une équipe de recherche et un Service d'Observation. Les services de bases de données (Simbad, VizieR) et de visualisation (Aladin), développés par le CDS, sont utilisés par l'ensemble de la communauté astronomique mondiale. Celui-ci est l'un des acteurs majeurs du développement de l'Observatoire Virtuel International en astronomie. Fin 2008, le CDS a été labellisé « Très Grande Infrastructure de Recherche » (TGIR), ce qui le place au même niveau que des infrastructures internationales comme l'European Southern Observatory ou RENATER à l'échelon national. J'ai effectué mon stage dans ce service, c'est pourquoi il va maintenant vous être présenté plus en détail.

Le Centre de Données astronomiques de Strasbourg

Créé en 1972 sous le nom de « Centre de Données Stellaires », il devint le Centre de Données astronomiques de Strasbourg en 1983. Il héberge maintenant la base de données de référence mondiale pour l'identification d'objets astronomiques. Par conséquent, ses missions consistent :

- à rassembler des informations utiles concernant les objets astronomiques sous forme informatisée
- à mettre à jour ces données en les critiquant et en les comparant
- à les distribuer à la communauté internationale
- à mener des recherches utilisant ces données.

Le CDS emploie actuellement 8 chercheurs/astronomes-adjoints, 1 chercheur postdoctoral, 11 informaticiens, 10 documentalistes et 3 administratifs, soit 33 personnes au total. Les services principaux du CDS se nomment Simbad, VizieR et Aladin.

Figure 1 - Simbad VizieR Aladin

Simbad

Simbad est la base de données de référence pour la nomenclature et la bibliographie des objets astronomiques situés hors du système solaire, qu'il a accumulé depuis plus de 40 ans. Ce service permet aux astronomes de trouver facilement des informations sur plus de 7 millions d'objets, grâce à plus de 280 000 références bibliographiques. De plus, la base de données Simbad contient un résolveur de noms permettant de retrouver l'objet désigné par l'un des 18 millions d'identifiants qu'elle contient. Des liens peuvent bien évidemment être faits avec d'autres services fonctionnant sur Internet. Simbad recevait en 2011 une moyenne de 284 000 requêtes par jour.

VizieR

VizieR est une base de données qui regroupe à l'heure actuelle plus de 11 000 catalogues d'objets astronomiques. Ces catalogues sont constitués de données relevées durant des missions d'observation de l'espace puis ajoutées à VizieR par les documentalistes du CDS. Ce procédé permet d'homogénéiser les données astronomiques afin de pouvoir les comparer et de les exporter sous différents formats. Les interrogations sur la base de catalogues peuvent porter sur de multiples critères comme la longueur d'onde avec laquelle les objets ont étés observés ou encore le nom de la mission correspondante. Entre 2011 et 2012, les requêtes de VizieR ont augmenté d'une moyenne de 370 000 requêtes par jour à une moyenne de 377 000 requêtes par jour. Le service connaît des pics d'utilisation pouvant atteindre 2 millions de requêtes en un jour.

Aladin

Aladin est un atlas interactif du ciel qui permet de visualiser et de comparer des images astronomiques. Il a été entièrement développé en Java pour l'ensemble de la communauté internationale par Pierre FERNIQUE, Thomas BOCH et François BONNAREL. Ces images proviennent d'observatoires sol et spatiaux et peuvent être utilisées pour former des cartes du ciel en trois dimensions, les boules HEALPix (cf. Figure 2). Celles-ci sont des ensembles de losanges de surfaces égales. Ces images disposent de plusieurs niveaux de précision, étant chacun constitués de 4 fois plus d'images que le précédent. Elles peuvent être fournies par l'utilisateur, par la base de données d'Aladin ou par d'autres bases de données telles que la NASA Extragalactical Database. Aladin est à la fois une application téléchargeable et une applet Java, toutes deux disponibles sur le site internet du CDS. Aladin recevait en 2012 une moyenne de 18586 requêtes par jour, ce qui représente une hausse de 11% par rapport à l'année précédente, dont la moyenne était de 16 800 requêtes par jour. Cependant, on peut remarquer que l'usage d'Aladin se resserre sur l'application standalone au détriment de l'applet. Cela témoigne d'une diminution de l'audience « grand public » au profit d'une utilisation scientifique.

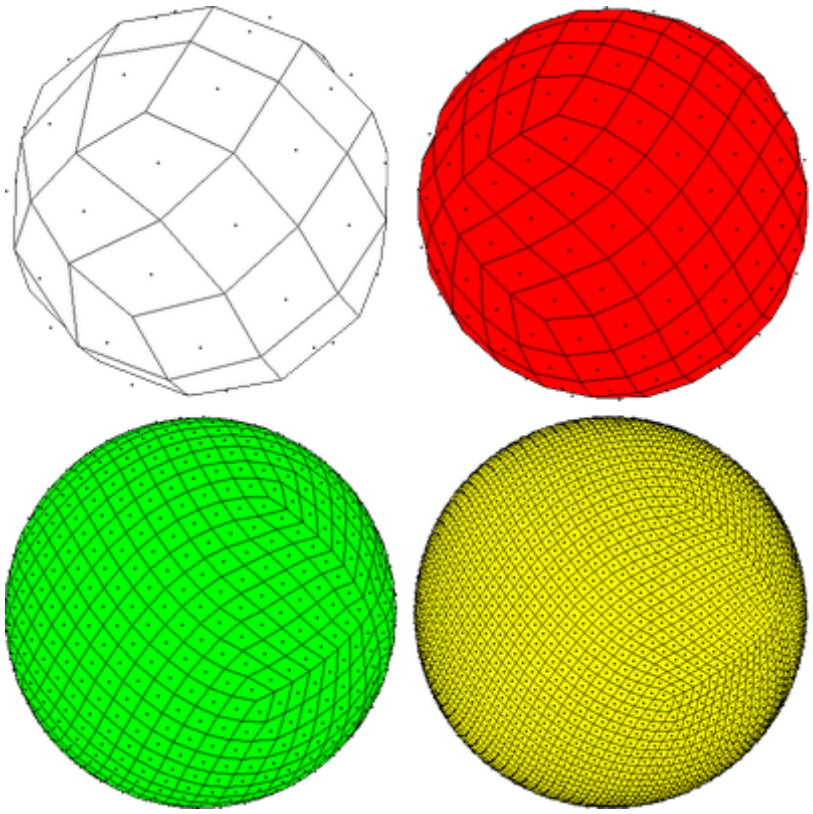

Figure 2 - Niveaux des boules HEALpix

L'International Virtual Observatory Alliance (IVOA)

Il s'agit d' une organisation internationale regroupant 20 projets d'Observatoire Virtuel (VO) provenant de 17 pays différents (voir Figure 3) : Afrique du Sud, Allemagne, Argentine, Arménie, Australie, Brésil, Canada, Chili, Chine, Espagne, États-Unis, France, Hongrie, Inde, Italie, Japon, Pologne, Royaume-Uni, Russie et Ukraine. Ce projet international a un but similaire au World Wide Web Consortium : mettre en place des standards et décider quels seront les travaux les plus importants à faire ainsi que les stratégies à adopter. Ceci permet de faciliter les échanges internationaux et la collaboration nécessaire à l'élaboration de standards d'interopérabilité et au déploiement d'outils, de systèmes et de structures organisés nécessaires pour permettre une utilisation internationale des archives astronomiques (images, catalogues, etc..). Telle est la mission confiée à l'IVOA lors de sa création en juin 2002. Le CDS participe activement à ce projet notamment par la réalisation des services cités précédemment. En tant que Très Grande Infrastructure de Recherche (TGIR), le CDS a des besoins conséquents en terme de puissance de calcul et de stockage de données. Par ailleurs, on peut remarquer que l'utilisation de ses services est en augmentation, ce qui confirme la tendance des années précédentes. Cela permet d'expliquer que des redimensionnements de l'architecture serveur soient envisagés.

Figure 3 - IVOA et ses membres

Présentation du stage

Sujet et Objectifs

Le stage que j'ai effectué durant ces 10 semaines m'a mis dans la situation d'un précurseur vis-à-vis de l'utilisation de l'Oculus Rift au CDS. Il s'agissait donc d'un stage plutôt orienté Recherche & Développement. En effet, cette nouvelle technologie n'est pas encore très répandue et le CDS souhaitait connaître son potentiel et savoir ce qui était faisable aussi bien dans le domaine professionnel que dans le domaine éducatif. Ce stage constitue donc en la première partie d'une étude concernant la réalité virtuelle à l'observatoire qui durera encore quelques mois probablement. Il s'agissait d'apprendre à maîtriser cette nouvelle technologie qu'est l'Oculus Rift et au fur et à mesure de mon avancée, réaliser des prototypes illustrant mes recherches et déterminer la faisabilité et la crédibilité de certains sujets. Le déroulement de mon stage s'est effectué en plusieurs étapes à savoir:

- La découverte du kit de développement Oculus Rift et étude de sa technologie
- La phase de recherches, essais et prototypage au fur et à mesure
- Réflexion sur l'utilisation à l'observatoire et adaptabilité à la technologie déjà utilisée

Le plan de ce rapport suivra donc ces différentes étapes afin de permettre de cerner le meilleure manière possible les détails de ces différentes étapes.

Planning du stage

ID	Nom de la tâche	Avril				Mai				Juin	
		7/4	14/4	21/4	28/4	5/5	12/5	19/5	26/5	2/6	9/4
1	**Présentation de l'Observatoire**										
2	**Découverte du kit**										
3	Etude de la technologie										
4	Essais langages										
5	Outils/Logiciels										
6	Choix techniques										
7	**Essais et prototypages**										
8	Objets 3D										
9	Scripts génération objets par code										
10	Visualisation fonctions 2D/3D										
11	Création trajectoire elliptique modifiable										
12	**Réflexion application**										
13	Mise en place utilisation Java dans C#										
14	Recherche utilisation HEALpix dans Unity										
15	Modélisation galerie d'images										
16	**Rédaction Rapport de Stage**										
17	**Préparation Soutenance**										

Découverte du kit de développement et étude technologique

Présentation rapide de l'Oculus Rift

Origine

L'Oculus Rift est un casque de réalité virtuelle développé par la société Oculus. Son financement s'est effectué en 2012 et depuis lorsque l'on entend parler de réalité virtuelle, le Rift n'est jamais bien loin. Actuellement, le Rift n'est disponible que dans sa version développeur et l'ouverture au grand public n'est pas prévue avant 2015. Un second kit de développement est en préparation et sera distribué sur commande en aout 2014.

Matériel

Le Rift se présente comme un masque, relié par un câble à un boitier d'acquisition (Figure 4). Depuis ce boitier d'acquisition partent trois autres câbles. Un câble USB relié à l'ordinateur qui permet d'utiliser les fonctionnalités de détection d'orientation et de mouvement de la tête, un câble vidéo soit HDMI soit DVI au choix, afin de pouvoir diffuser l'image dans le casque et un câble d'alimentation pour que le Rift soit branché sur secteur.

Le masque est en réalité un écran (1280*800), devant lequel sont disposées deux lentilles, une pour chaque œil. Sont également fournies 3 paires de lentilles dont deux appliquent une correction visuelle afin que les personnes portant des lunettes puissent tout de même se servir du kit.

Figure 4 - Kit de développement Oculus Rift

Fonctionnement

Le masque agit comme un deuxième écran, il est donc possible de dupliquer ou d'étendre l'affichage de l'ordinateur. La particularité de l'écran du masque est qu'il est divisé en deux, dans le but d'obtenir une image pour chaque œil. On a donc un écran par œil de 640*400 px. Attention cette division n'est pas faite automatiquement par le Rift est c'est bel et bien au développeur de faire en sorte que son application s'affiche deux fois. Les images diffusées par l'écran du Rift passent chacune par la lentille déformante face à elle avant d'arriver jusqu'à vos yeux.

But

Le système Rift a pour but d'immerger l'utilisateur dans ce qu'il voit grâce à un angle de vue de 110° (limite du champ visuel humain) et à des lentilles qui appliquent une déformation permettant cette illusion. Le potentiel du Rift est réellement immense car, au-delà du jeu vidéo, beaucoup de chercheurs, astronomes, médecins ou même l'armée s'intéressent à cette technologie.

Etude des différents outils utilisables

Développer un logiciel ou une application en utilisant l'Oculus Rift présente bien des différences avec les développements normaux. En effet, la réalité virtuelle n'a d'intérêt que si l'application présente des éléments en 3D. L'utilisation de la 2D est tout à fait possible mais le rendu serait moins impressionnant et n'exploiterait pas les capacités du Rift.

C'est dans cette optique que l'étude du Rift a été faite et c'est pour cela que plusieurs langages et outils ont été envisagés.

C++/C combinés à OpenGL

Le kit de développement de l'Oculus est utilisable uniquement si les librairies ont été préalablement téléchargée sur le site du fournisseur. Ces bibliothèques sont, comme souhaité par Oculus, open-source. Il est donc possible de les partager et de les modifier autant que souhaité. Ces dernières sont de base disponibles pour Windows/Linux/Mac. Etant donné qu'il était plus facile de trouver de la documentation, des informations et des démonstrations sur Windows, notre choix s'est porté sur cet OS en ce qui concerne le développement. Les bibliothèques

fournies par Oculus sont en C++, il s'agit de .dll. Il est donc logique d'envisager comme langages de base le C++/C afin de développer des applications et autres prototypes pour le Rift. En effet, la documentation relative à ces bibliothèques est assez complète et compréhensible à condition de maîtriser tout de même ces langages. Concernant la partie graphique, l'utilisation de la librairie graphique OpenGL permettrait de fournir un panel d'outils conséquent et suffisant pour commencer le prototypage. Ne maitrisant pas correctement le C/C++ (uniquement les bases), j'ai préféré chercher vers des technologies que je connaissais, c'est pourquoi je me suis penché vers le Java.

Java/jMonkeyEngine

Java est un langage puissant mais sous estimé concernant la 3D. Il existe un grand nombre de librairies 3D, comme par exemple JavaOpenGL(JOGL) et également celle fournie par Oracle, Java3D. jMonkeyEngine est quand à lui un moteur de jeu, basé sur une interface NetBeans permettant de générer de manière simplifiée des objets en 3D et quelques interactions de base. Il ne possède malheureusement pas de gestion de caméra assez complexe pour convenir au développement pour le Rift. Basées sur OpenGL, ces libraires seraient suffisantes pour le prototypage de petites applications. Il y a cependant un problème, non des moindres, à savoir l'intégration du SDK (Software Development Kit) pour le rendre fonctionnel avec du Java. Il s'agit d'une solution envisageable, mais couteuse en terme de temps étant donné qu'avant d'avoir le moindre résultat, il est nécessaire de réécrire le SDK du Rift en Java. Pour réaliser ceci il faut créer une bibliothèque permettant de lire le contenu des .dll afin de s'en servir dans du code Java, une Java Native Interface (JNI). La JNI est une méthode très utilisée lorsque qu'il s'agit d'utiliser du code .NET dans une application Java par exemple. Lorsque l'on utilise la JNI, il faut également penser à modifier les types et ne pas utiliser les mêmes que dans Java, car ils portent des noms différents (jboolean plutôt que boolean par exemple) . Cela se passe en trois étapes:

- Tout d'abord il faut obtenir le fichier source,à savoir ici OVR.c ainsi que le header correspondant OVR.h (OVR pour Oculus Virtual Reality)

- Ceci fait, on va pouvoir créer notre classe intermédiaire, ici JRiftLibrary.cpp. Il est nécessaire d'utiliser la ligne #include OVR.h afin de pouvoir utiliser le code de OVR.c.

Il s'agit ici en réalité de créer les méthodes que l'on souhaite utiliser en Java. On écrit donc ce programme en C++, en utilisant OVR.c et lorsque l'on souhaite qu'une

méthode/attribut ou autre soit utilisable dans un programme Java, on précise JNIEXPORT. Les méthodes proviennent de OVR.c et nous les redéfinissons afin qu'elles soient utilisables en Java.

- Une fois notre classe intermédiaire créée, il faut en faire un .dll, que nous chargerons par la suite dans une classe Java grâce à la ligne *System.loadLibrary(JRiftLibrary).* Il s'agit en fait d'une sorte d'héritage, et il est donc nécessaire de rajouter toutes les méthodes qui sont dans la librairie, sans les redéfinir. Pour cela il suffit d'utiliser le mot clé *native,* par exemple:

```
private native boolean _initSubSystem;
```

Cette méthode est fonctionnelle mais relativement longue et rigoureuse. L'idéal serait de ne pas avoir à réécrire les librairies déjà créées et de pouvoir les utiliser directement.

Réaliser une application qui puisse être affichée par le Rift n'a rien de plus compliqué que les applications ordinaires. Le vrai problème est de réussir à rendre la réalité virtuelle visible et faire que cette illusion ne soit pas agressive pour les yeux tout en étant immersive. Afin d'obtenir la réalité virtuelle il est nécessaire d'afficher notre image deux fois, tout en appliquant ce que l'on appelle la Barrel-Distorsion. Conscients de cette difficulté non des moindres, lors de cette phase de découverte des outils, une petite application Java a été réalisée ayant pour but de visualiser un schéma de cette distorsion. La réalisation de cette petite application avait également pour but de familiariser le développeur avec l'environnement JOGL, pour la 3D Java. Voici donc le résultat de l'exécution qui permet de comprendre le fonctionnement de la Barrel-Distorsion:

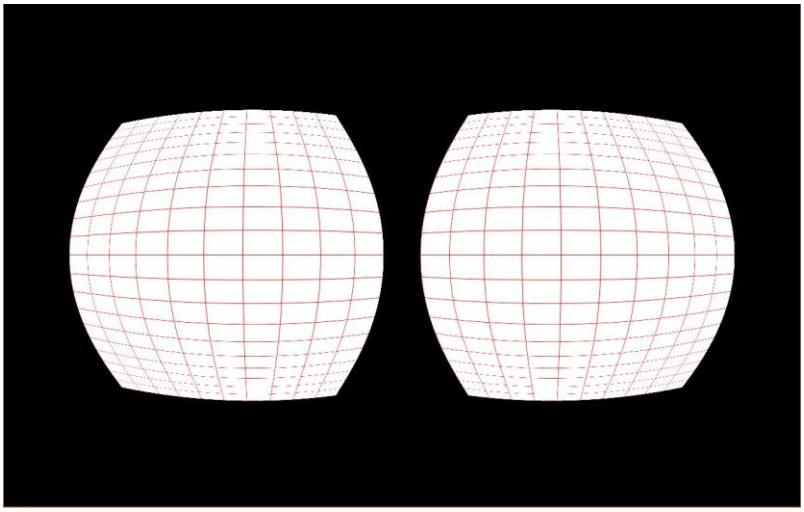

Figure 5 - Barrel Distortion

Afin de pouvoir visualiser les applications de cette manière, Oculus a créé des objets, des "prefabs" destinés à être utilisés dans le moteur de jeu Unity. Il s'agit d'une caméra possédant le filtre Barrel-Distorsion / Dual Screen permettant une adaptation à la réalité virtuelle et c'est pour cela qu'après avoir envisagé Java, nous nous sommes penchés vers ce logiciel qu'est Unity.

Unity/C#/JS

Unity est un moteur de jeu 2D/3D extrêmement puissant basé sur une technologie OpenGL retravaillée. Il est composé de deux grosses parties:

- l'API UnityEngine qui propose des solutions de gestions 3D complètes et polyvalentes

- le logiciel Unity2D/3D (cf Figure 7) qui permet à l'utilisateur de créer ses scènes, d'y placer des objets et de générer son application selon les paramètres souhaités (Système d'exploitation, plateforme, définition, touches etc...).

L'interface de Unity propose à l'utilisateur de créer des objets ou de les importer (par exemple des modèles 3D-Blender), de les disposer sur une scène, régler la lumière etc... Une fois ceci fait, il est possible de créer des scripts en C# ou Javascript en utilisant l'API UnityEngine qui seront attachés aux objets de la scène permettant ainsi à l'utilisateur de gérer ses animations, les transitions, l'intelligence artificielle etc...

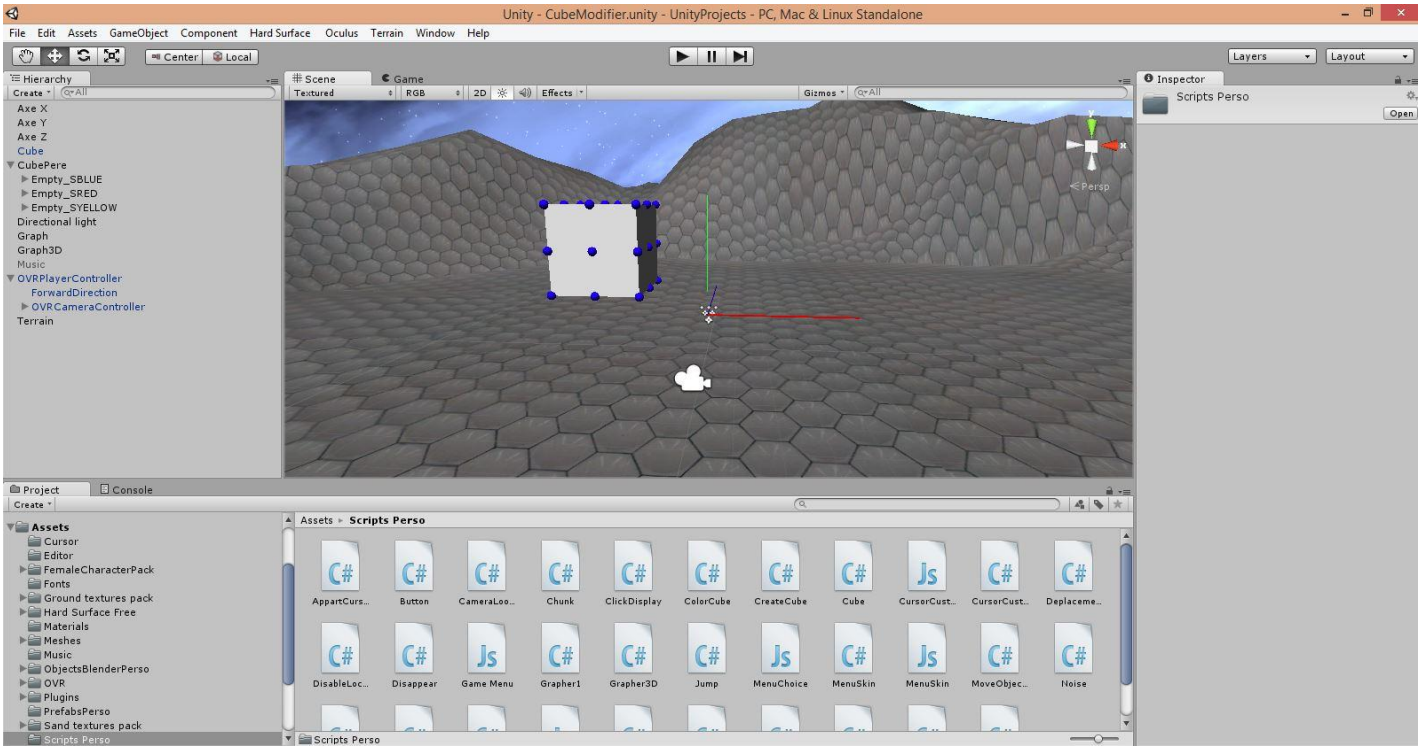

Figure 6 - Interface Unity3D

Pour résumer avec Unity, il est donc possible de générer des objets 3D grâce au logiciel plutôt qu'au travers d'un code, de générer la vue propre à l'Oculus Rift quasi-automatiquement et d'utiliser un langage tel que le C# afin d'écrire des classes et des scripts qui nous permettront de gérer notre application. Il n'était donc pas nécessaire d'apprendre à maîtriser la 3D parfaitement avant de pouvoir l'utiliser et le C# étant très semblable au Java, son apprentissage serait simplifié.

Après ces tests, nous avons décidé de commencer avec Unity/C#/JavaScript pour les recherches et le prototypage concernant le Rift.

Recherches, essais et prototypages

Une fois la phase de découverte effectuée, les outils choisis et le tout mis en place, il était alors possible de commencer la phase de recherche et d'essais.

Objets 3D et transformations basiques

Nous avons convenu que le début de la recherche devait s'effectuer en s'orientant tout d'abord vers la maîtrise de Unity et la mise en place d'une simple scène 3D accompagnée de quelques scripts pour exécuter des opérations et des interactions de base dans cette scène afin de voir si cette gestion était simple et envisageable pour la conception d'une future application.

Lors de cette première phase il fallait vraiment chercher à faire les opérations de base sur Unity c'est-à-dire la création d'un polygone (ici en l'occurence des sphères), fixer une texture sur un objet etc...

Les polygones à afficher ont été choisis comme étant des sphères dans l'idée d'y fixer des textures de planètes par exemple pour rester dans le thème de l'observatoire. Une fois ces polygones créés, on remarque qu'un objet est composé en fait de plusieurs éléments obligatoires ou non:

- le Transform qui est l'élément contenant des attributs de l'objet à savoir : sa position, son orientation etc... C'est un élément obligatoire de l'objet.
- le Mesh qui est l'ensemble des côtés, arêtes et formes de l'objet. C'est en fait sa structure aérienne, son maillage. Il s'agit également d'un élément obligatoire de l'objet qui est vide si l'objet n'a pas de structure, ce qu'on appelle un emptyObject dans Unity. Le Mesh est composé de deux éléments le Filter et le Renderer, le premier contenant le type de maillage et le second la façon dont il se comporte (ombres, rendu texture, filtres etc...)
- le Collider, un élément qui permet de gérer les collisions avec d'autres objets. Cet élément est dispensable mais fortement recommandé pour ne rien laisser au hasard.
- le Texture Renderer qui comme son nom l'indique contient les informations de la texture de l'objet.

Il existe d'autres éléments que l'on peut ajouter à un objet comme des générateurs de particules etc...

Ceci étant fait, l'application n'affichait rien du tout une fois lancée, alors que les objets et les textures étaient bien en place. En effet pour visualiser la scène, il faut y placer une caméra ainsi qu'une source de lumière. La position de la caméra doit être adaptée au reste de la scène afin d'avoir un champ de vision suffisant et de pouvoir apprécier sa globalité. Ceci étant assez contraignant, un script permettant à la caméra de se déplacer a été créé. Ce script a donc été écrit en C# en utilisant l'API UnityEngine. La méthode ci dessous est la méthode Update qui est appelée à chaque nouvelle frame (entre 25 et 60 par seconde):

```
void Update()
{
    bool moveForward = false;
    bool moveLeft    = false;
    bool moveRight   = false;
    bool moveBack    = false;

    MoveScale = 1.0f;

    if (Input.GetKey(KeyCode.W)) moveForward = true;
    if (Input.GetKey(KeyCode.A)) moveLeft    = true;
    if (Input.GetKey(KeyCode.S)) moveBack    = true;
    if (Input.GetKey(KeyCode.D)) moveRight   = true;
    if (Input.GetKey(KeyCode.UpArrow))    moveForward = true;
    if (Input.GetKey(KeyCode.LeftArrow))  moveLeft    = true;
    if (Input.GetKey(KeyCode.DownArrow))  moveBack    = true;
    if (Input.GetKey(KeyCode.RightArrow)) moveRight   = true;

    if (!Controller.isGrounded)
        MoveScale = 0.0f; //On vérifie qu'on est au sol sinon on empeche de bouger

    MoveScale *= DeltaTime; //Illusion de l'accélération

    if (Input.GetKey(KeyCode.LeftShift) || Input.GetKey(KeyCode.RightShift))
        MoveScale *= 2.0f; //Possibilité de courir

    if (moveForward)
            TransformDirection(Vector3.forward * MoveScale);
    if (moveBack)
            TransformDirection(Vector3.back * MoveScale);
    if (moveLeft)
            TransformDirection(Vector3.left * MoveScale);
    if (moveRight)
            TransformDirection(Vector3.right * MoveScale);
```

Toutes les transformations dans Unity (comme dans la plupart des outils 3D) se font grâce à des vecteurs. Ici on va utiliser des vecteurs par défaut existant dans la classe Vector3 du package

UnityEngine afin de faire nos transformations. Il faut également noter que l'on va utiliser des float, qui permettent un traitement plus rapide en informatique. Simplement, ce script va tester à chaque frame si on appuie sur un bouton de déplacement et si c'est le cas, alors on va se déplacer dans une direction prédéfinie.

Il faut savoir que chaque script dans Unity doit être attaché à un objet. Une fois le script créé, on le dépose dans le répertoire Assets du projet Unity et il sera compilé automatiquement. S'il y a une erreur, Unity avertit l'utilisateur et empêche de lancer le programme. Une fois ce script déposé, si la compilation a eu correctement lieu, il suffit de le glisser/déposer sur l'objet de notre scène. Par conséquent les dénominations dans le script correspondent aux éléments de l'objet. Par exemple dans le script précédent, on a TransformDirection. Si l'objet est attaché à la caméra alors c'est la transformDirection de la caméra qui sera affectée et pas une autre. Il est évidemment possible d'agir sur un autre objet auquel le script n'est pas attaché en le récupérant par des méthodes de UnityEngine.

La figure 8 montre la création de deux sphères et leur visualisation:

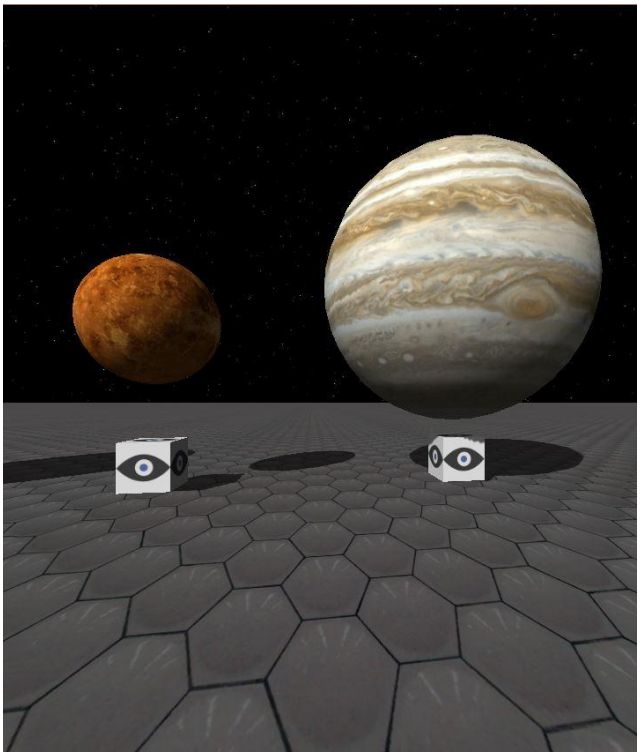

Figure 7 - Venus & Jupiter

Une fois l'affichage de simples objets fait et le déplacement dans la scène possible il fallait penser à animer ces objets, rendre l'application un peu plus complexe, c'est pourquoi un script permettant à un objet de tourner sur lui même ou encore un script permettant de déplacer un objet avec la souris ont été créés.

Quelques petits scripts et objets nous permettaient d'avoir une application basique, mais il fallait à présent permettre à cette application d'être correctement visualisée par le Rift et c'est ici que le préfab, OVRCamera, fourni par Oculus est intervenu. Cet objet est donc une caméra disponible sur le site Oculus dont les réglages optiques, c'est à dire champ de vision/zoom/angle, sont déjà mis en place au travers d'un script et elle permet alors au Rift de devenir fonctionnel. Non seulement ce script règle la vision de façon à ce que la réalité virtuelle soit fonctionnelle mais en plus il suffit de lui lier la bibliothèque OVR pour que les systèmes de gyroscopes du Rift soient opérationnels. Une fois cette caméra mise en place sur la scène comme n'importe quel objet, le rendu correspond alors à ce qu'il faut pour l'Oculus Rift (cf. Figure 10).

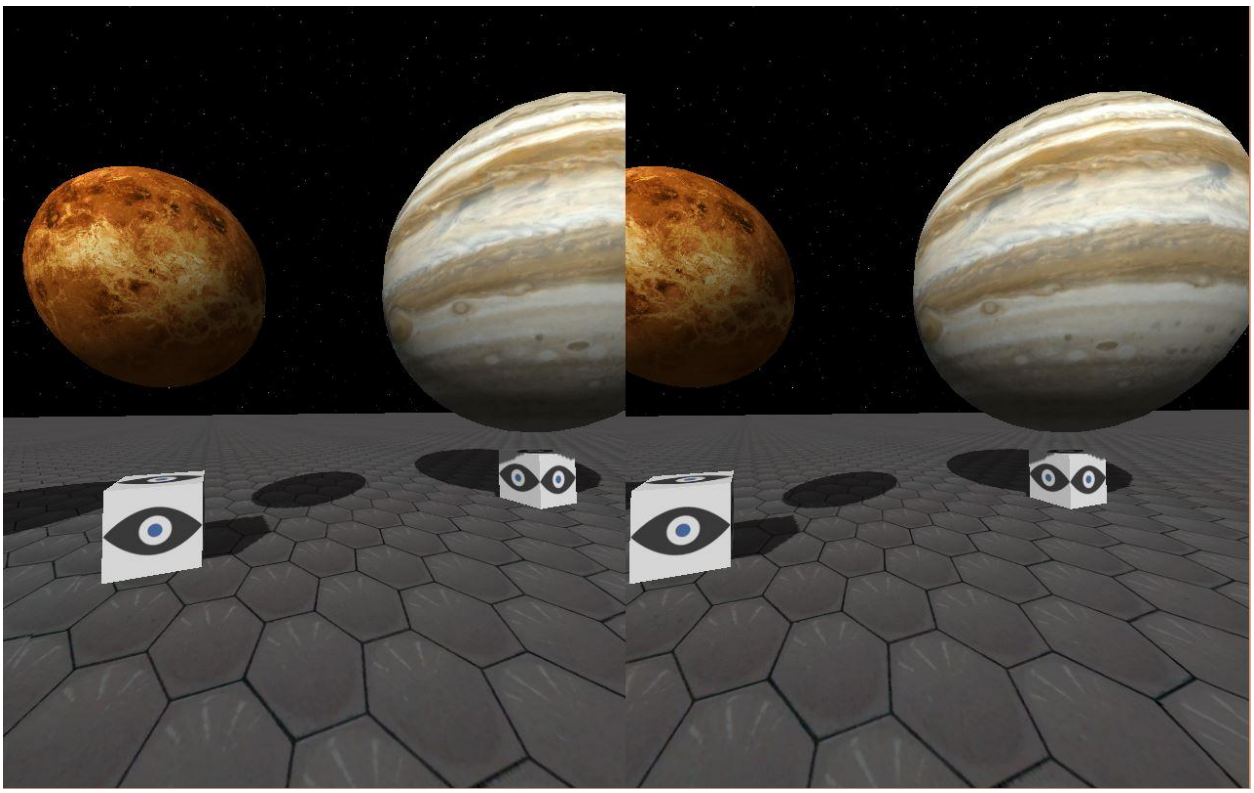

Figure 8 - Venus & Jupiter affichage réalité virtuelle

On observe le décalage caractéristique de l'affichage 3D qui permet aux images de se superposer correctement. Le script fourni par Oculus étant perfectible, il a été légèrement retravaillé et

adapté afin d'avoir un rendu optimal. Une fois l'application produite, le test avec le Rift fut concluant: l'impression d'immersion était là et les gyroscopes permettant d'orienter le regard étaient totalement fonctionnels.

Génération d'objets par script

Une fois que les tests basiques sur des objets créés depuis l'interface Unity3D furent effectués, nous avons décidé d'étudier la question de la génération d'objets 3D non par le biais du logiciel mais directement par des scripts C# en utilisant simplement la bibliothèque UnityEngine. Pour commencer, il fallait trouver comment générer à partir de rien, un simple plan.

Les bibliothèques 3D fonctionnent à peu près toutes de la même manière. En effet chaque objet est créé à partir de deux choses: une liste de vecteurs qui pour la plupart servent à définir les arêtes de l'objet et une liste de triangle permettant de relier ces arêtes entre elles afin de former le maillage de l'objet 3D.

Pour la création d'un objet purement au travers d'un script il a fallu apprendre à maîtriser la 3D et les connaissances acquises en mathématiques au cours du DUT furent très utiles. Au-delà du fait de créer un plan, l'idée était de créer plusieurs de ces plans afin de les assembler et d'en faire un cube. L'approche effectuée par cette démarche a permis de comprendre la manière dont fonctionne les objets 3D au sein de Unity et ainsi nous ouvrir à la perspective de générer n'importe quel polygone au travers d'un script C#. En effet, à partir du moment où il est possible de positionner et d'orienter un plan, il est possible de créer tous les cotés possibles d'un objet et donc n'importe quel polygone. Cette possibilité établie et ajoutée à la liste des choses faisables, il fallait alors se pencher sur la manipulation de ces objets (grossir/déplacer/effacer).

Manipulation des objets

La première chose à faire était de voir s'il était possible de sauvegarder un objet généré afin de pouvoir recréer ce même objet mais à un autre endroit avec une autre taille etc... Pour cela, une saisie était nécessaire lors de l'utilisation du programme sous la forme de case et de texte. UnityEngine propose dans son package des fonctionnalités de GUI (Interface utilisateur) très efficaces et simples d'utilisation mais à condition de les visualiser sur un seul écran, ce qui n'était pas le cas ici. En effet pour visualiser du texte par l'Oculus, le fait de faire apparaitre la GUI n'est pas suffisant car elle n'apparait qu'une seule fois étant donné qu'elle n'est pas sujette à

l'observation au travers de la caméra mais qu'elle vient se rajouter au dessus de l'image. Il fallait donc placer au bon endroit sur l'écran le texte et par conséquent le dupliquer deux fois et l'ajuster aux deux écrans du Rift, en l'adaptant à la largeur de l'écran et en appliquant un ratio afin de permettre une bonne visualisation de l'écrit à travers le casque (cf Figure 11).

```
GUI.Label(new Rect(Screen.width / 5 * 1, 230, 80, 20),"Scale");
GUI.Label(new Rect(Screen.width / 5 * 3, 230, 80, 20),"Scale");
textFieldStringScale=GUI.TextField(new Rect(Screen.width / 5 * 1, 250, 80, 20),textFieldStringScale,10);
textFieldStringScale=GUI.TextField(new Rect(Screen.width / 5 * 3, 250, 80, 20),textFieldStringScale,10);
```

Figure 9 - Placement GUI

Ce morceau de code montre le placement horizontal des différents éléments de la GUI et le calcul nécessaire pour que cela reste relatif à la définition de l'écran du Rift. Ceci étant fait, il fut alors possible de saisir des informations au travers de l'interface de Rift, dans des cases affichées à l'écran. Il était donc possible de saisir par exemple trois coordonnées pour placer un objet à ces coordonnées (cf. Figure 12)

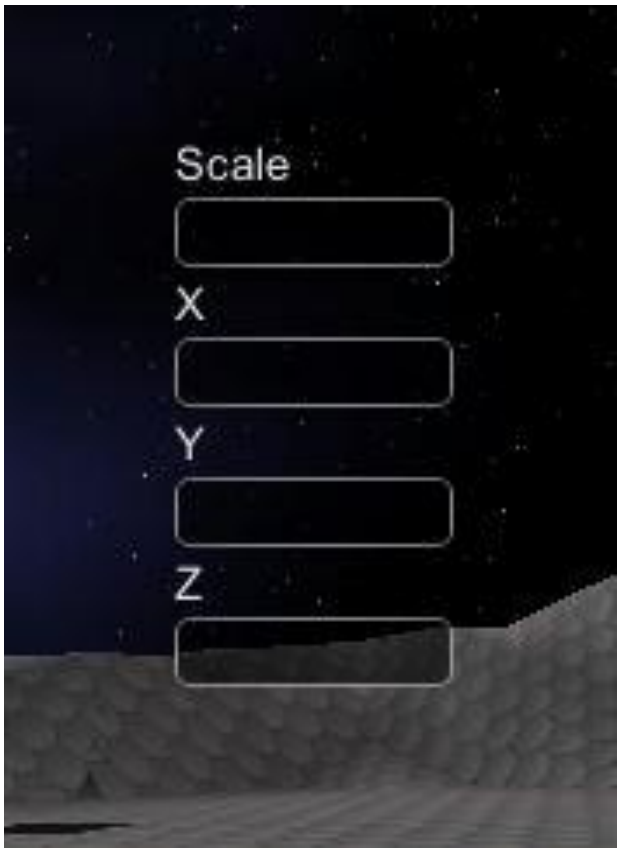

Figure 10 - Saisie coordonnées

Une fois ces coordonnées saisies, le plus complexe fut de les faire comprendre au programme, ceci étant possible grâce à UnityEngine et C# qui contiennent des classes mathématiques et String permettant d'effectuer la récupération du texte saisi pour ensuite le convertir en float, le rendant utilisable dans le programme.

Il était alors possible de placer l'objet souhaité, généré au préalable par un script, à l'endroit désiré. Une grosse avancée en soi, nous permettant d'aller plus loin: vers la visualisation de fonctions mathématiques.

Visualisation de fonctions mathématiques en 2D puis en 3D

Il faut avoir à l'esprit que la plupart des corps célestes dans l'espace suivent une équation de trajectoire qui peut évoluer au fur et à mesure que le temps passe. Il était donc intéressant de voir, dans un premier temps, s'il était possible de faire bouger un objet en suivant une fonction mathématique. Une fonction mathématique se visualisant au travers d'une série de point, l'outil choisi fut donc les ParticleSystem de UnityEngine. Cet objet permet de générer et gérer les systèmes de particules dans les jeux, comme la neige ou la pluie par exemple. Ces particules sont en fait contenues dans un tableau ouvert. On peut donc en ajouter autant que l'on veut, et ainsi les modifier individuellement. En créant un système de particules par le script, il est possible de régler absolument tout ce que l'on souhaite et en particulier les transformations appliquées à chaque particule. Les particules contenues dans le système de particules possèdent leur propre trajectoire et il est donc possible de leur appliquer chacune une fonction mathématique.

Premièrement nous avons créé un système de particules ayant une taille variable (choisie arbitrairement entre 10 et 200 pour cause de performance). On a ensuite enregistré dans le même script C# des méthodes correspondant à des fonctions mathématiques que l'on pourra ensuite appliquer à chaque particule. Chaque particule ayant une position précise, c'est à dire une abscisse connue, il faut alors appliquer la fonction désirée au système de particules donc à toutes les particules. Grâce à cette méthode, il est alors possible de voir apparaitre la courbe de la fonction choisie (cf. Figure 13 - Figure 14).

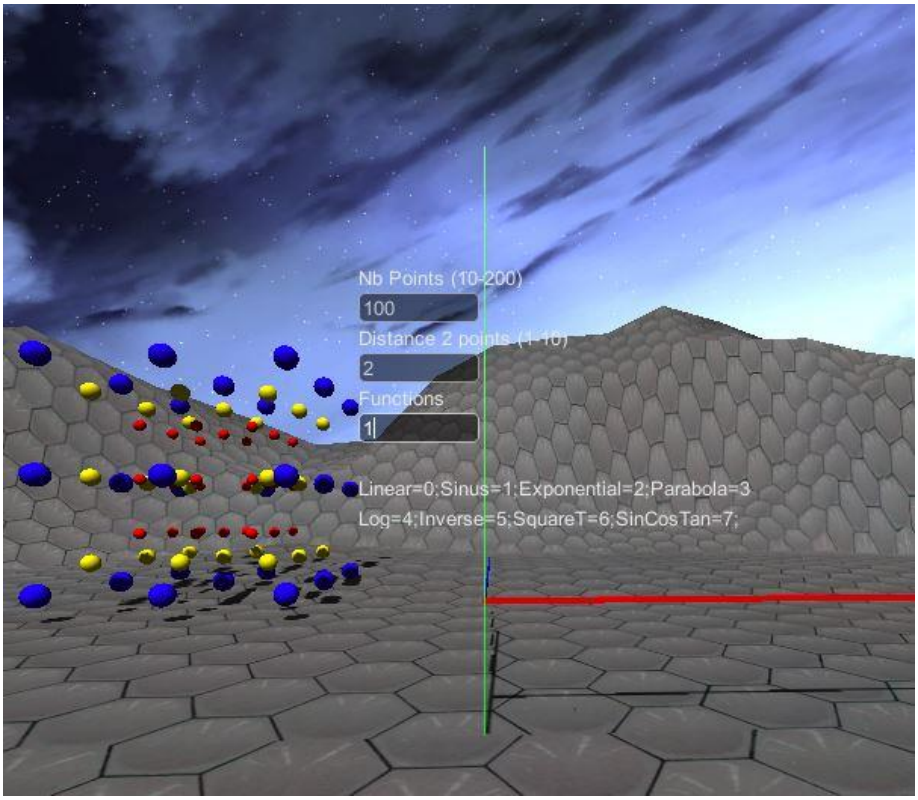

Figure 11 - Saisie Fonctions

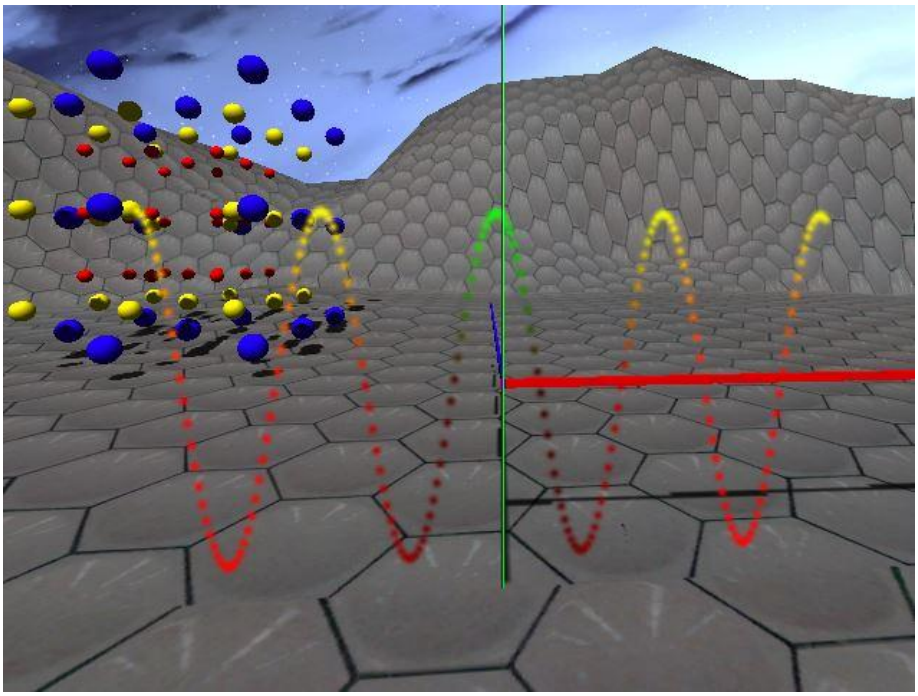

Figure 12 - Sinus Résultat écran précédent

A noter également que les couleurs sont des objets qui peuvent s'additionner dans UnityEngine et qu'il est possible de récupérer les couleurs des axes du repère dans Unity et ainsi coloriser la particule selon sa distance par rapport aux axes de référence, donnant une couleur variable. Il est également possible d'animer une fonction qui a lieu de l'être. Il faut donc que cette fonction soit périodique (aucun intérêt avec une fonction constante par exemple), et dans le calcul de la fonction il faut ajouter "+time.DeltaTime". La variable DeltaTime est une variable très utile et utilisée car elle est créée dès le lancement de l'application et augmente au fur et à mesure que le temps passe. Appliquée à Sinus ou Cosinus, on voit donc un mouvement dans la courbe.

Les fonctions étant uniquement visibles sur un plan, il eut été intéressant de visualiser certaines fonctions en 3D. Le principe est le même mais il faut également utiliser l'axe z en tant que repère et faire appliquer la fonction à l'abscisse et l'ordonnée (cf. Figure 15)

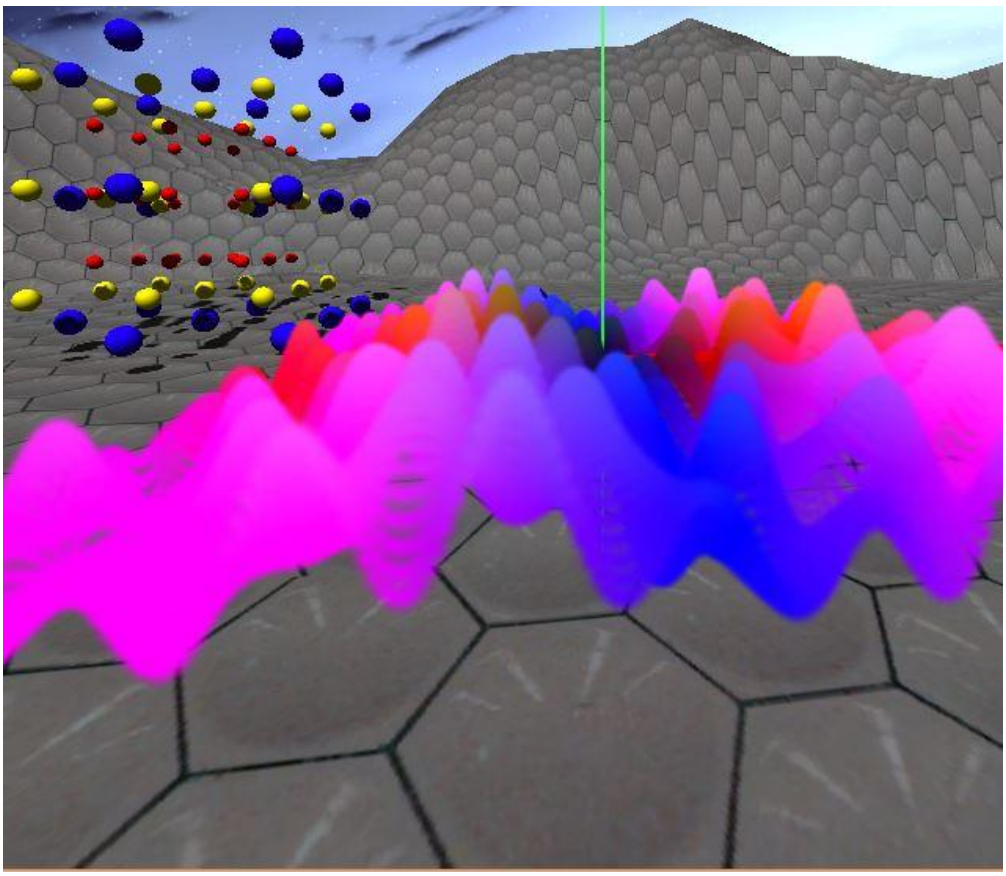

Figure 13 - Sin(Cos(Tan X,Z)))

Les tests concernant les fonctions étant concluant, nous avons décidé de nous orienter vers le passage d'une équation de trajectoire à un objet, en l'occurrence une sphère représentant une planète.

Trajectoire elliptique d'un objet modifiable

Le CDS voulant évidemment rester dans sa thématique, nous avons convenu de nous orienter vers la visualisation d'un objet possédant une trajectoire précise. Pour aller plus loin il fallait également voir s'il était possible de faire varier les valeurs d'un script, donc ses attributs en temps réel au cours de l'exécution du programme. Nous avions précédemment vu que passer une fonction à un objet était possible, donc passer l'équation d'une trajectoire elliptique est donc logiquement possible. Les objets possédant une trajectoire particulière avec des attributs et des constantes modifiables, le premier pas vers une telle représentation fut de créer un objet possédant une trajectoire elliptique, ayant des axes de longueur variable ainsi qu'un centre aux coordonnées modifiables.

Ainsi, nous avons donc la possibilité de créer un objet à un endroit précis et il est possible de lui passer des coordonnées particulières. Cet objet devient blanc lors de sa sélection afin de faciliter la visualisation permettant de savoir sur quel objet la transformation va être appliquée. On distingue également le centre de sa trajectoire elliptique afin de voir son changement de situation (cf. Figure 16).

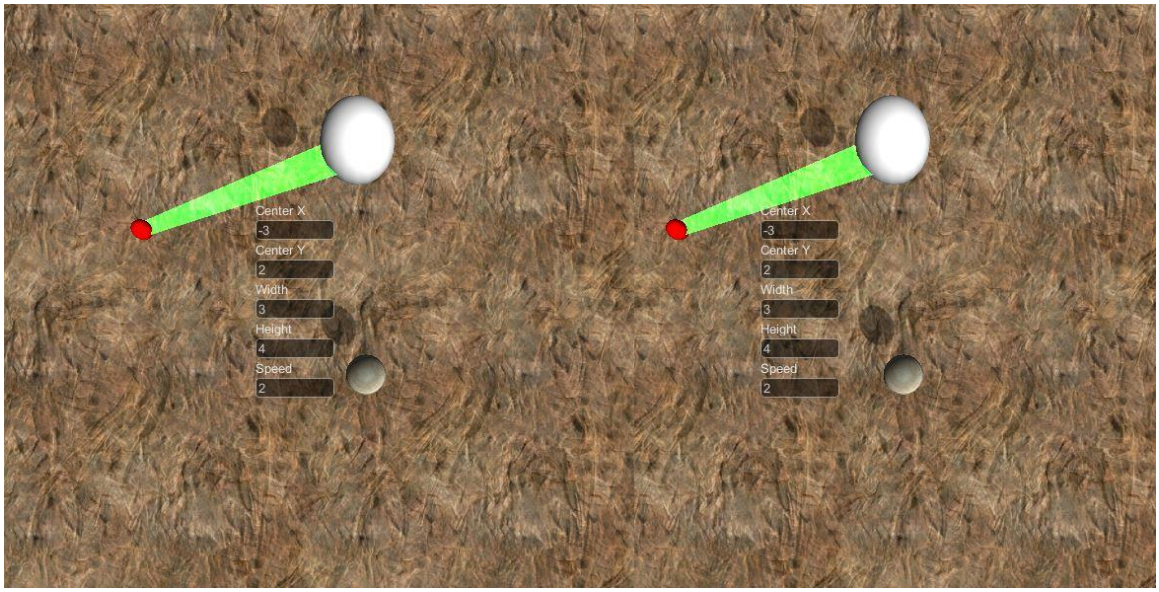

Figure 14 - Visualisation trajectoire elliptique

Les tests essentiels étant convainquant et nous permettant ainsi de constater la puissance de Unity, nous avons alors décidé d'étudier la manière dont il serait possible d'adapter cet outil aux technologies utilisées au CDS, voir comment s'en servir en corrélation avec une application déjà existante, pouvoir réutiliser des objets déjà existants etc... Ce qui nous mène à la deuxième thématique abordée durant la durée de ce stage: les utilisations et l'adaptation aux services et aux données du CDS.

Adaptation aux services et aux données du CDS

Le CDS propose plusieurs applications et services qu'il serait intéressant de pouvoir utiliser, ou du moins une partie, avec le kit Oculus Rift, c'est pourquoi les recherches se sont ensuite orientées dans ce sens. Etant donné que le CDS fournit le service Aladin codé en Java, l'orientation s'est donc faite vers l'intégration de code Java dans le code C#. En effet, il eut été intéressant d'intégrer une bibliothèque telle que HEALPix afin de pouvoir réutiliser la technologie en C# sans avoir à réécrire le tout.

Utiliser Java dans C# - IKVM

L'utilisation de bibliothèques existantes est une chose possible pour Java mais en ce qui concerne le passage de code .NET vers le Java et non l'inverse. Pour cela on utilise ce que l'on appelle la Java Native Interface (JNI). Il s'agit d'une interface permettant de lier le code Java au code .NET (principalement en C/C++) sans avoir besoin de le ressaisir totalement. Malheureusement, il faut tout de même retaper quelques lignes de codes (qui peuvent devenir nombreuses si on a affaire à une grosse bibliothèque), comme les déclarations de fonctions car JNI utilise une syntaxe propre différente du Java et du langage de destination. Cette solution est, quoique peu utilisée, applicable en ce qui concerne le passage du Java vers le C#. Dans un soucis de rapidité et de simplification nous avons orienté nos recherches vers d'autres technologies et une revenait souvent, utilisée par le privé et les professionnels: IKVM.

IKVM se définit comme une implémentation de Java vers le code .NET. Cette machine intègre une machine virtuelle Java implémentée pour le .NET (sous la forme de .dll) ainsi qu'un lot de .dll permettant l'utilisation du code Java dans le .NET. Cet outil permet également de compiler des .java et .class et ensuite d'en faire un .jar. Le principe de base de IKVM est de transformer une librairie java (.jar) en une bibliothèque .NET (.dll). Cette bibliothèque n'est pas interprétable naturellement par le C# et c'est là qu'intervient la deuxième fonction d'IKVM. IKVM fournit tous les .dll permettant d'interpréter du Java en C#. Il existe plusieurs façons de s'en servir. On peut en effet créer une application .NET (grâce à Visual Studio par exemple) et intégrer ces bibliothèques. Dans notre cas, comme l'utilisation de Unity a été choisie, il fallait que cela fonctionne facilement avec du C# et c'est là que Unity a encore démontré sa puissance.

A chaque fois que le dossier "Assets" du projet Unity est modifié, le moteur de jeu recompile chaque script et va chercher dans ce même dossier tous les fichiers sources nécessaires pour les intégrer. Pour résumer, premièrement on créé le jar des classes que l'on souhaite utiliser dans Unity. Ensuite on les transforme grâce à IKVM puis on importe les .dll d'IKVM et nos .dll perso dans le projet Unity. S'il faut alors utiliser une classe Java importée, il suffira simplement d'appeler l'objet comme s'il s'agissait d'une bibliothèque .NET.

Des tests ont été faits avec IKVM pour observer la crédibilité de la méthode et ils furent pour le moins concluant. D'abord un simple test avec des classes Java basiques (Carre, Triangle etc...) et des appels à leurs méthodes respectives puis un test plus avancé avec la librairie HEALPix fournie par le CDS (divisée en packages, dépendante d'autre librairies). Il est d'ailleurs notable que la plupart du temps IKVM va permettre la compilation et la création du .dll même s'il ne parvient pas à trouver les fichiers sources. En soi il s'agit d'un avantage et d'un inconvénient car on peut alors utiliser des éléments qui ne dépendent pas des fichiers mais on risque d'avoir de graves erreurs dans le cas inverse. C'est là que Unity intervient, prévenant au préalable s'il manque une source (classe ou bibliothèque) et empêchant le lancement de l'application le cas échéant.

Il est donc possible d'utiliser des ressources déjà disponibles et immédiatement adaptables à l'environnement Unity.

HEALPix/Aladin et Unity

L'application Aladin du CDS utilise la technologie énoncée précédemment dans le rapport, HEALPix. La question a été posée de savoir s'il était possible d'utiliser cette technologie dans Unity. Malheureusement, il s'avère que les objets dans Unity possèdent un maillage très strict, peu modifiable, à moins de le créer soi-même. En effet, chaque objet, qu'il soit créé par le script ou par l'interface Unity possède un maillage unique. Etant donné que HEALpix utilise une sphère vue de l'intérieure pour le découpage, nous avions pensé à utiliser une sphère de Unity et son maillage. Mais le maillage ne permet pas de disposer certaines textures à certains endroits comme le suggère la bibliothèque HEALPix.

Nous avons réfléchi quant à une alternative concernant l'utilisation de HEALPix. Tout d'abord, la création de morceaux de sphères extérieurs à Unity. Il serait envisageable de créer un objet 3D

extérieur à Unity en forme de partie de sphère (par exemple en losange comme dans HEALPix). Il faudrait pouvoir réaliser le maillage souhaité, en le rendant adaptable au besoin.

Autre solution envisageable, faire fusionner les morceaux de texture avant de les fixer sur la sphère. Il s'agit là d'une solution envisageable pour un petit atlas, avec une petite bibliothèque. Malheureusement ce n'est pas le cas d'Aladin et cette solution serait beaucoup trop couteuse en temps et en performance, car cela signifie reformer la texture à chaque zoom.

Galerie d'image

Dans le but de conclure le stage et de produire une petite application nous avons fait le choix de concevoir une sorte de galerie d'images, sous la forme d'une visite interactive (cf. Figure 17). L'utilisateur enfile le masque de réalité virtuelle et se retrouve plongé dans un musée dans lequel il pourra visualiser des images, choisies par le CDS. Les images peuvent être de toutes sortes et l'objectif de cette application est de permettre à un public de découvrir des objets célestes à travers une application pédagogique. Les tableaux représentant les différents objets seront accompagnés d'un texte contenant les informations relatives à la photo observée. L'avantage d'une telle application est qu'elle est entièrement modifiable et facile d'accès. On peut envisager de modéliser plusieurs salles dans le musée et permettre ainsi de présenter encore plus de photographies. Cela nous donne une idée de l'étendue des possibilités d'adaptation de la réalité virtuelle à des applications à but, par exemple, pédagogique.

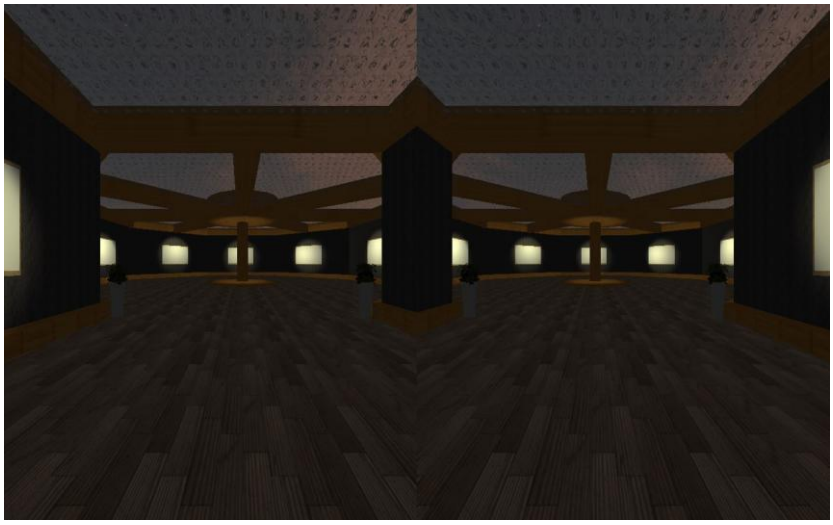

Figure 15 - Aperçu galerie Photo

Conclusion

Tout au long du déroulement de ce stage, il a été possible d'appréhender les différents aspects concernant le développement d'application intégrant la réalité virtuelle. Nous retiendrons en effet que la réalité virtuelle est plus orienté vers un développement 3D, mais que les possibilités sont énormes. En ce qui concerne le développement d'une application totalement nouvelle, nous avons vu que Oculus fournissait des fonctionnalités pour des outils tels que Unity permettant d'utiliser des langages interprétés mais en offrant la possibilité d'intégrer des librairies déjà existantes. Cela permet en effet de simplifier le développement utilisant la 3D car il est possible de gérer la scène "physiquement" et pas uniquement par du code. Des applications telles que la visualisation d'objets, le tout orienté pour un public relativement jeune dans un but pédagogique sont donc tout à fait envisageables pour l'observatoire ou le planétarium.

Nous avons vu toutefois vu qu'il existe quelques limites concernant le développement avec Unity. En effet le maillage des objets 3D est restreint et ne permet pas directement d'utiliser des technologies telles que HEALpix. Il est envisageable de passer par un logiciel de 3D tel que Blender pour créer des objets propres à l'application mais il s'agit d'une solution couteuse en terme de temps et qui requiert certaines qualifications en modélisation 3D.

Les outils étant étudiés et des prototypes réalisés, il serait intéressant de choisir une application à développer orientée réalité virtuelle et c'est dans ce cadre qu'ont lieu de nouveaux stages dans lesquels va être étudier l'intégration de la réalité virtuelle à une application déjà existante.

Concernant les apports personnels et professionnels de ce stage, ce fut une expérience intéressante et enrichissante. Voulant continuer mes études, je ne pouvais imaginer un meilleur stage. Les personnes que j'ai pu rencontrer, le sujet passionnant que l'on m'a confié et le cadre de travail agréable dans lequel j'ai évolué sont autant d'éléments que je garderai en mémoire. J'ai eu accès à une technologie méconnue du grand public et grâce à son étude j'ai appris des langages tels que le C# que je ne maitrisais pas avant. J'ai également pu comprendre comment se passait une recherche en partant de rien, lorsque l'on se retrouve confronté à une technologie totalement inconnue et j'ai trouvé cela extrêmement intéressant. A ajouter à cela la gratification de la part de M.Schaaf qui a présenté notre travail à l'IVOA lors de sa conférence à Madrid.

Bibliographie

Oculus Forum : https://developer.oculusvr.com

Oculus Share : https://share.oculusvr.com

Unity Forum : http://forum.unity3d.com

Unity Docs : http://docs.unity3d.com

GitHub : https://github.com

Developpez.com : http://www.developpez.com

O**penclassrooms :** fr.openclassrooms.com

CgPersia : forum.cgpersia.com

CodeProject : http://codeproject.com

Artisant numérique : artisant.karma-lab.com

IKVM : www.ikvm.net

CatLikeCoding : catlikecoding.com

StackOverFlow : www.stackoverflow.com

Wikipédia : fr.wikipedia.org (Voxels/Rift/HEALpix...)

Youtube : www.youtube.com (Jacksepticeye, Voxels...)

Kite & Lightning : www.kiteandlightning.la

FICHE RAPPORT DESTINEE A LA BIBLIOTHEQUE

RAPPORT CONFIDENTIEL ET NE DEVANT PAS FIGURER A LA BIBLIOTHEQUE :

☐ Oui ☑ non

NOM ET PRENOM DE L'ETUDIANT : DA ROCHA Joffrey

DUT : INFORMATIQUE

☑ S4 ☐ S4 bis ☐ Année Spéciale/AETP

LICENCE PROFESSIONNELLE

☐ ASRALL ☐ CISII

TITRE DU RAPPORT :

Réalité Virtuelle appliquée aux services et aux données astronomiques du CDS

(utilisation du kit Oculus VR)

Nom de l'Entreprise : Centre de données astronomiques de Strasbourg

Adresse :

Observatoire Astronomique de Strasbourg

11 rue de l'université

67000 Strasbourg

Type d'activité (domaines couverts par l'entreprise) : Enseignement et recherche

Nom du parrain (enseignant IUT) : Bernard MANGEOL

Mots-clés (sujets traités) : Réalité Virtuelle, Oculus Rift, Unity, Données Spatiales

---

Résumé

Prise en main du kit de réalité virtuelle Oculus Rift, évaluation du potentiel d'adaptabilité de la réalité virtuelle aux services du CDS et prototypages d'applications diverses.