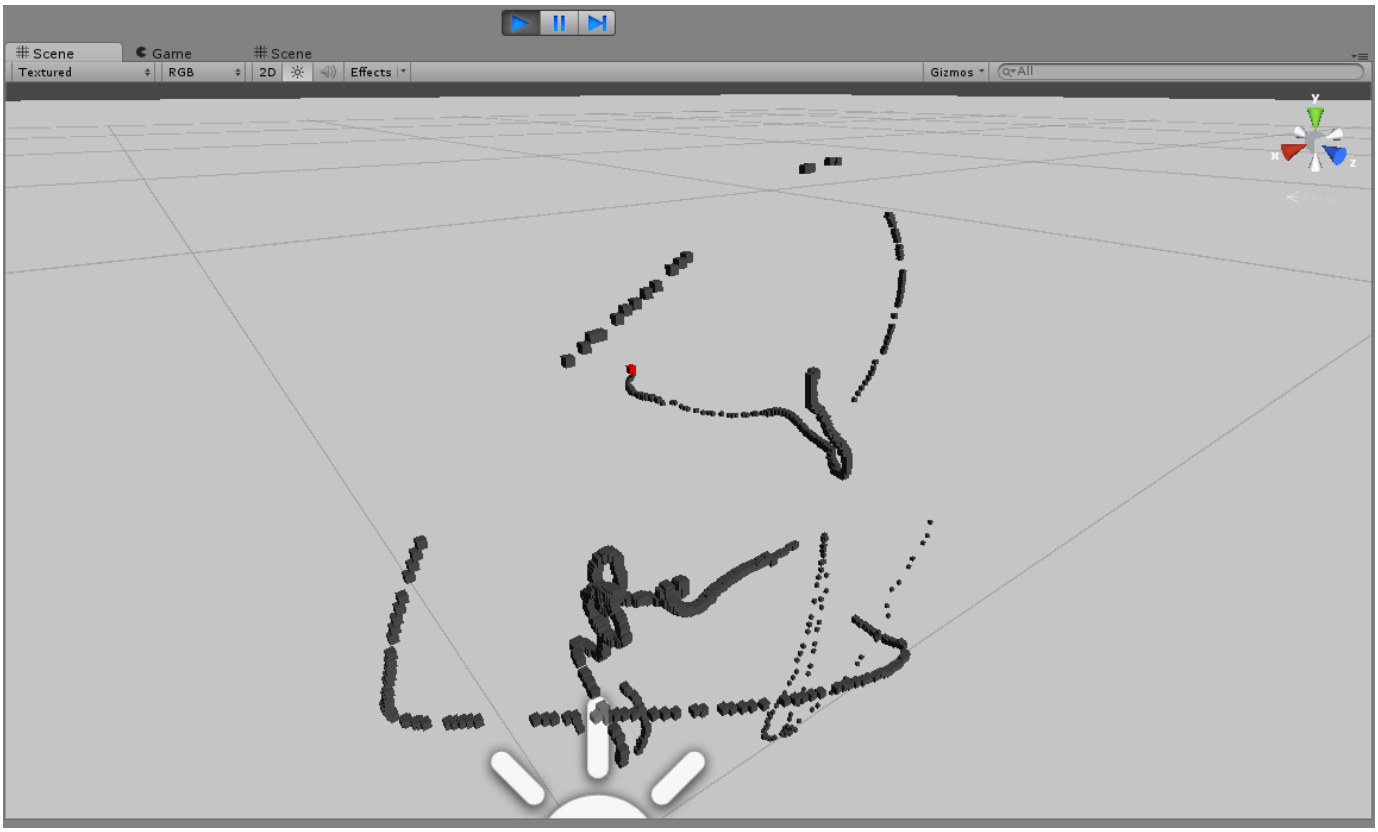
Rapport de projet IMA4

Dessin 3D en environnement immersif



Alexandre Jouy

Romain Libaert

Table des matières

Remerciements

Nous tenons à remercier Laurent Grisoni, notre tuteur, ainsi que Samuel Degrande qui nous a aidé tout au long du projet sur les aspects techniques liés au matériel que nous avons utilisé.

Nous remercions également Xavier Wielemans pour son aide sur le logiciel Unity ainsi que Pauline De Chalendar pour sa disponibilité tout au long du projet.

Introduction

Dans le cadre de notre 4ème année à Polytech Lille en IMA, nous avons réalisé un projet intitulé « Dessin 3D en environnement immersif ». Ce projet s'inscrit dans la collaboration entre l'IRCICA et l'institut d'art du Fresnoy.

Notre but est de pouvoir arriver à dessiner dans un environnement virtuel tout en étant libre de ses mouvements. Pour cela nous allons utiliser le système de caméras infrarouges ART-Track. Ce système permet de récupérer en temps réel les mouvements d'un utilisateur équipé des marqueurs infrarouges associés.

Nous allons ensuite récupérer les coordonnées renvoyées par le système afin de pouvoir les utiliser pour recréer les mouvements du dessinateur. Ces données seront ensuite récupérées dans le logiciel Unity à l'aide d'un serveur UDP, puis elles seront utilisées afin de faire en sorte que tous les mouvements captés soient reproduits dans l'environnement virtuel que nous avons créé.

Cet environnement est ensuite projeté dans l'Oculus afin de fournir au dessinateur un environnement immersif en temps réel.

Cette application sera alors utilisée par pauline De Chalendar, une artiste du Fresnoy, pour son projet de fin d'année, mais également dans le cadre d'une exposition temporaire à l'Imaginarium de Tourcoing au mois de septembre 2015.

1. Le système ART-Track

1.1. Présentation générale

Le système ART-Track est composé d'un ensemble de caméras infrarouges (8 dans notre cas) et de marqueurs infrarouges passifs ou actifs. Ces caméras permettent d'émettre des pulsations infrarouges et de capter leur réflexion sur les marqueurs passifs. Elles captent aussi les pulsations infrarouges émises par les marqueurs actifs.

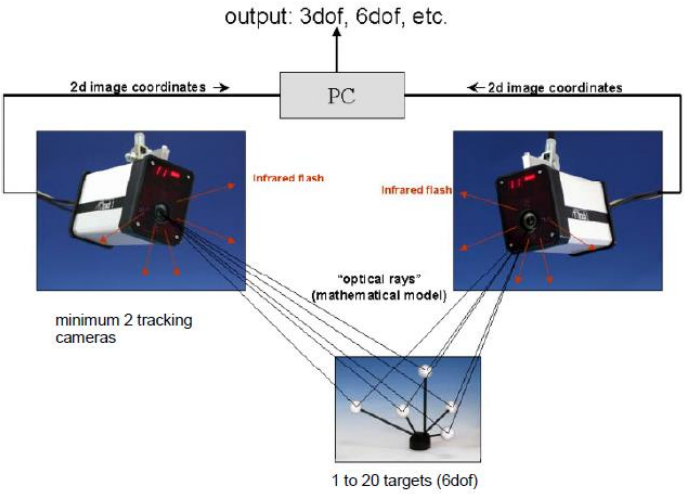


Schéma de fonctionnement du système ART-Track

Toutes les informations ainsi récupérées sont alors transmises (via un câble Ethernet) à un ordinateur s'occupant du calcul des positions des marqueurs.

Le fait que nous ayons accès à 8 caméras pour l'application finale nous permet d'avoir une liberté de mouvements quasi totale sur 6 degrés de liberté (3 axes de translation et 3 axes de rotation). Cependant, à cause d'une indisponibilité d'une partie du système au début du projet, nous avons fait tous les premiers tests avec seulement 4 caméras.

1.2. Les caméras

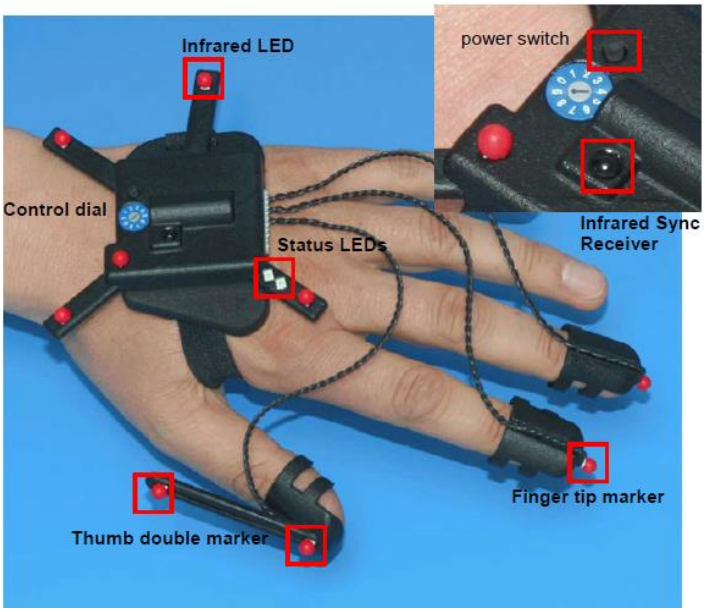
Comme énoncé précédemment, les caméras ART-Track permettent d'émettre et de recevoir dans l'infrarouge. Elles émettent des flashs qui sont reflétés par les marqueurs passifs. La lumière réfléchie est ensuite récupérée par chacune des caméras qui envoient les données récupérées à l'ordinateur.

Dans le cas des marqueurs actifs, le principe de fonctionnement est un peu plus complexe. A chaque cycle de récupération des positions, une trame infrarouge correspondant à un marqueur précis (la main droite par exemple) est émise. Quand le marqueur reçoit cette trame, il en émet ensuite une qui lui est spécifique. Il peut ainsi être situé par la caméra.

Toutes les données sont envoyées à un ordinateur central qui calcule alors les positions de chacun des marqueurs et les exprime selon les 6 degrés de liberté.

1.3. Les marqueurs infrarouges

Les marqueurs utilisés sont des marqueurs infrarouges. Nous utilisons un marqueur passif pour l'Oculus Rift et des marqueurs actifs pour chacune des mains.



Exemple d'un marqueur de main actif

Au début du projet nous avons commencé par n'utiliser qu'un seul marqueur, celui de la main droite. Nous pensions en effet n'avoir besoin que de cette main pour dessiner et réaliser une première version fonctionnelle de notre projet.

Cependant, après avoir discuté avec Pauline, nous avons dû modifier la manière dont nous envisagions le fonctionnement. Tout d'abord, elle est gauchère et donc dessine uniquement avec sa main gauche. De plus, comme elle a besoin de se déplacer autour des dessins qu'elle crée, nous nous somme rendus compte que l'utilisations de la deuxième main pour le déplacement pouvait être un choix judicieux.

Un marqueur passif a également été fixé sur l'Oculus afin de pouvoir le repérer dans l'espace si nécessaire.

2. L'Oculus Rift et le logiciel Unity

2.1. Présentation de l'Oculus Rift

L'Oculus Rift est un casque de réalité virtuel. Il permet à l'utilisateur qui l'équipe de pouvoir évoluer dans un environnement généré en temps réel par un ordinateur.

En effet, l'Oculus est composé de deux mini écrans, un devant chaque œil, qui projettent chacun une image légèrement décalée afin de générer un effet stéréoscopique* pour l'utilisateur.

L'Oculus Rift DK2

Dans notre projet, nous utilisons l'Oculus Rift DK2 (pour Development Kit 2). Il a l'avantage d'avoir une meilleure résolution d'affichage ainsi qu'une latence moins grande lors de la reproduction des mouvements de tête. En effet, le léger décalage entre le mouvement réel et virtuel du premier Oculus donnait la nausée à certaines personnes.

Afin de suivre précisément les mouvements de tête de l'utilisateur, la version 2 de l'Oculus Rift utilise donc plusieurs capteurs : un gyroscope, un accéléromètre et un magnétomètre*. Le fait d'utiliser 3 capteurs différents permet ainsi d'augmenter la précision et la fiabilité lors de la reproduction du mouvement.

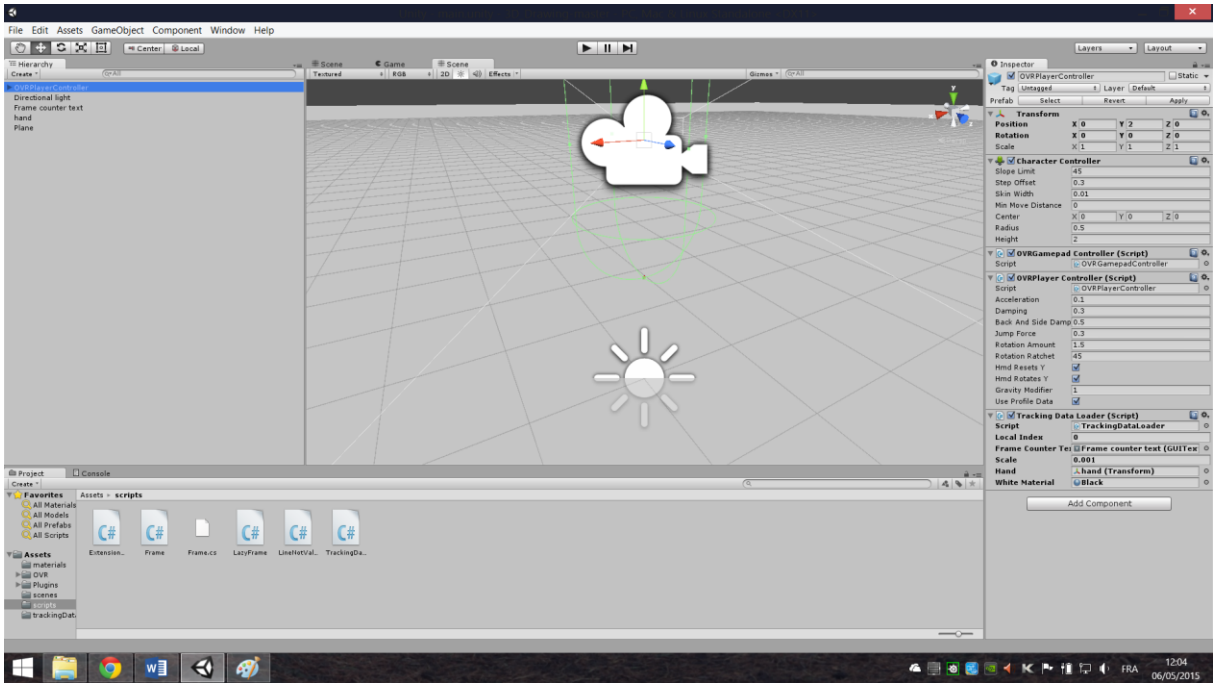
Dans notre projet, nous intégrons donc l'Oculus Rift à un environnement que nous avons décidé de créer à l'aide du moteur graphique Unity.

2.2. Unity

Unity est un moteur graphique utilisé principalement pour créer des jeux vidéo. Il permet en effet au concepteur de manipuler des objets en 3D directement dans un environnement graphique, ce qui le rend relativement simple à utiliser.

Il permet cependant de créer des applications plus complexes à l'aide de scripts rédigés en C#, JavaScript ou Boo.

Nous avons choisi le C# car c'est un langage qui se rapproche de ceux que nous avons l'habitude d'utiliser.



Exemple de l'interface d'Unity

Cela nous permet également d'utiliser toutes les classes et méthodes définies dans ce langage, en plus de celles propres à Unity. Nous avons donc pu créer un serveur UDP simple qui nous permet de recevoir les données envoyées par l'ordinateur relié aux caméras ART-Track et de les utiliser directement dans l'environnement graphique que nous avons créé.

3. Déroulement du projet

Lors du premier après-midi que nous avons passé à l'IRCICA, Laurent Grisoni nous a montré l'endroit où nous allions travailler pendant la première moitié du projet ainsi que le matériel qui était à notre disposition.

Nous avons également rencontré Pauline De Chalendar et les personnes travaillant avec elle. Lors de cette réunion nous avons établi plus précisément les enjeux du projet et ce qui était attendu de nous. Cela nous a donc permis de dresser une première version du cahier des charges et de pouvoir concrètement commencer à travailler.

3.1. Réception et utilisation des données de position

La première étape a été de déterminer la manière dont il fallait récupérer les données de position générées par l'ordinateur relié au système ART-Track. Nous nous sommes alors penchés sur la documentation technique afin de comprendre plus en détail le fonctionnement technique de ce système.

Nous avons donc constaté que les positions calculées par l'ordinateur pouvaient être envoyées sur un réseau. Soit sur le localhost, soit sur un réseau local choisi par l'utillisateur. Nous avons décidé de le laisser de la manière dont il était configuré.

Nous avons alors récupéré un programme écrit en C que nous avons compilé sur une machine tournant sous Debian. Ce programme est en fait un serveur UDP basique qui récupère les données formatées envoyées sur le réseau précisé, sur le port 5000. Il affichait ensuite les informations sur un terminal.

Nous nous sommes alors penchés sur la question suivante :

Comment intégrer ce programme dans un environnement graphique ?

Le moteur graphique qui nous a été conseillé lors de la réunion est Unity. Comme nous l'avons dit précédemment, nous avons choisi d'écrire nos programmes en C#. Cela se rapproche en effet de langages informatiques que nous avons déjà utilisés. De plus au premier abord, cela nous a paru le moyen le plus simple d'intégrer le code C déjà existant.

Nous avons alors fait des recherches sur la manière d'intégrer du C dans du C#, mais les seules manières que nous avons trouvé étaient soit très compliquées, soit trop longues à mettre en place.

Nous avons alors finalement choisi de recréer un serveur UDP en C# pour qu'il puisse « communiquer » avec Unity.

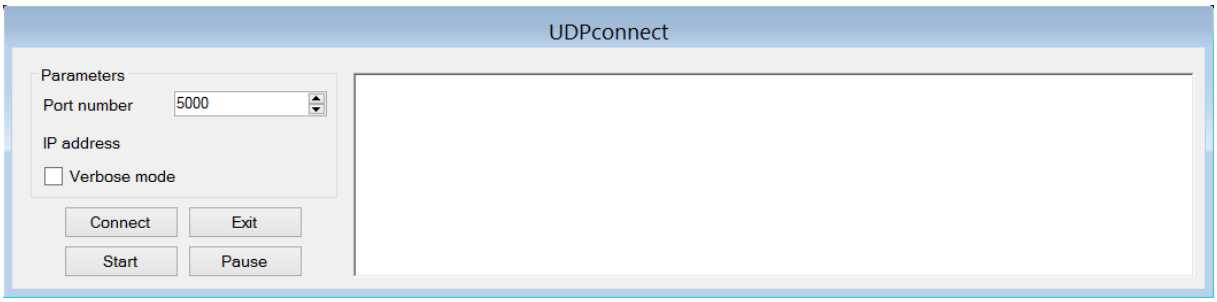
3.2. Création d'un serveur UDP en C#

Pour la création d'un serveur UDP en C#, nous avons décidé d'utiliser l'environnement de développement Visual Studio. En effet cet éditeur est relativement simple à prendre en main et permet de créer des interfaces graphiques très simplement. De plus, la licence de Visual Studio est gratuite avec Polytech.

Le logo de Visual Studio

Nous avons alors dans un premier temps créé une interface graphique afin de pouvoir contrôler la mise en marche et l'arrêt de la réception des données. Nous avons ensuite créé le serveur UDP et les fonctions nécessaires au traitement et à l'affichage formaté des données reçues.

Dans ce programme nous avons notamment créé la classe Frame. Pour chaque paquet UDP reçu nous constituons un objet Frame. Celui-ci contient les informations nécessaires, à savoir les coordonnées des mains et des doigts, de la tête, le type de ligne, l'orientation de la tête, … Elle contient aussi un constructeur qui permet de ranger ces informations en lui donnant comme paramètre le paquet UDP reçu.



L'interface graphique de notre programme

Nous avons alors réalisé les premiers tests afin de savoir si notre programme était bien fonctionnel. Nous avons réussi à le faire marcher après quelques petites modifications, mais il est ressorti plusieurs problèmes : un temps de latence important entre les données affichées sur l'ordinateur connecté aux caméras et celles affichées par notre programme ainsi que notre interface qui ne répondait plus une foi la réception lancée.

Le premier problème était simplement dû au fait que nous affichions beaucoup d'informations dans la console de debug, ralentissant ainsi la réactivité du programme. Pour corriger cela, nous pouvions donc soit stopper les affichages dans la console, soit exécuter le programme en mode release*.

Afin que notre programme puisse répondre alors qu'il est en pleine réception de données nous avons dû séparer les processus sur plusieurs threads séparés. En effet le serveur UDP n'est ni plus ni moins qu'une boucle infinie lisant un fichier. Par conséquent tant que nous sommes dans cette boucle, l'interface ne pouvait plus répondre. Nous avons donc séparé ces deux parties dans deux threads distincts.

3.3. Intégration à Unity

Nous nous sommes penchés sur la communication de notre programme avec Unity. Nous avons cherché un moyen simple et efficace de transmettre les données. On a tout d'abord pensé à les transmettre par un fichier, mais cela a été jugé inefficace.

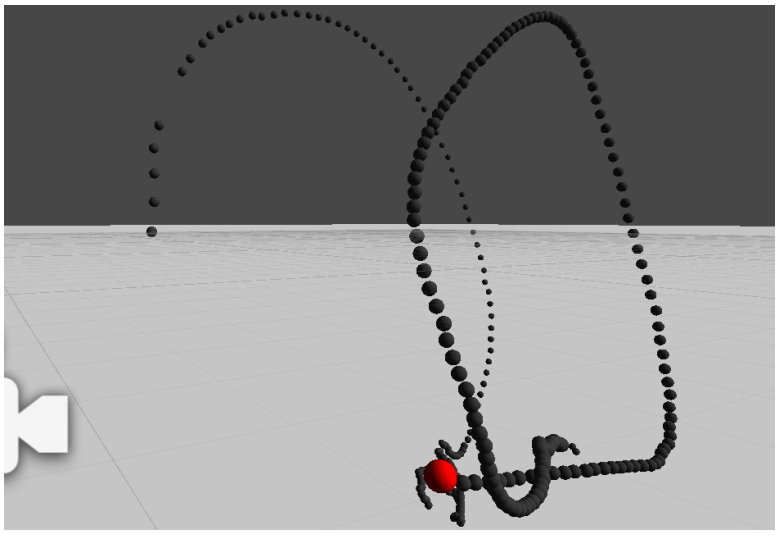
On s'est alors penché un peu plus sur l'intégration du C# sous Unity et nous nous sommes aperçus qu'il n'utilisait pas seulement la syntaxe de ce langage pour ses scripts, mais qu'il incluait bien toutes les fonctionnalités et bibliothèques prévues par Microsoft.

Nous avons alors décidé d'intégrer notre serveur UDP dans les scripts de notre projet Unity. Cela est en effet la méthode la plus performante car elle nous permet d'utiliser directement les valeurs de position récupérées à chaque frame*. Cela garantie donc un programme plus fiable et plus réactif, avec une étape en moins entre les mouvements de l'utilisateur et le rendu.

Nous avons alors fait les premiers tests pour être sûrs que la réception des données était bien fonctionnelle. Pour cela nous avons utilisé la méthode DrawLine de la classe debug. Cette méthode permet de tracer simplement des lignes entre deux points définis dans l'espace. Comme cette méthode fait partie de la classe debug, les erreurs sont bien gérées et cela évite donc également des crashs ou des arrêts inopinés de l'éditeur. Cette technique d'affichage n'était que temporaire dans le sens où elle permet effectivement de dessiner des lignes mais ces lignes ne sont visibles que dans le mode éditeur d'Unity. Les lignes ne sont pas visibles dans un programme en mode release*.

3.4. Génération des lignes

Nous avons dans un premier temps modélisé nos lignes comme une succession de points. Le principe est donc simple : à chaque rafraichissement de l'image, nous positionnons un objet ponctuel dans l'espace à la coordonnée mesurée. Nous obtenons alors à ce moment-là ce genre de rendu :



Cette technique présente les avantages d'être très simple à mettre en place et d'être relativement efficace. Cependant, la génération des objets est infinie, par conséquent nous avons observé une latence progressive du dispositif. Le nombre d'objets est en croissance permanente, et le rendu des images devient par conséquent de plus en plus complexe. Nous avons observé différentes pistes pour finir par limiter le nombre d'objets affichés. A chaque nouvelle image, un objet est donc créé et remplace l'objet le plus ancien. A noter qu'il est important de détruire explicitement l'objet inutile : en effet il n'est plus référencé dans le programme mais il est toujours présent dans la scène.

Pour améliorer les performances d'affichage, nous avons essayé de remplacer les sphères par des cubes. En effet ceux-ci sont moins complexes à réaliser. Nous avons aussi éliminé les ombres qui consomment une bonne partie des ressources de la machine.

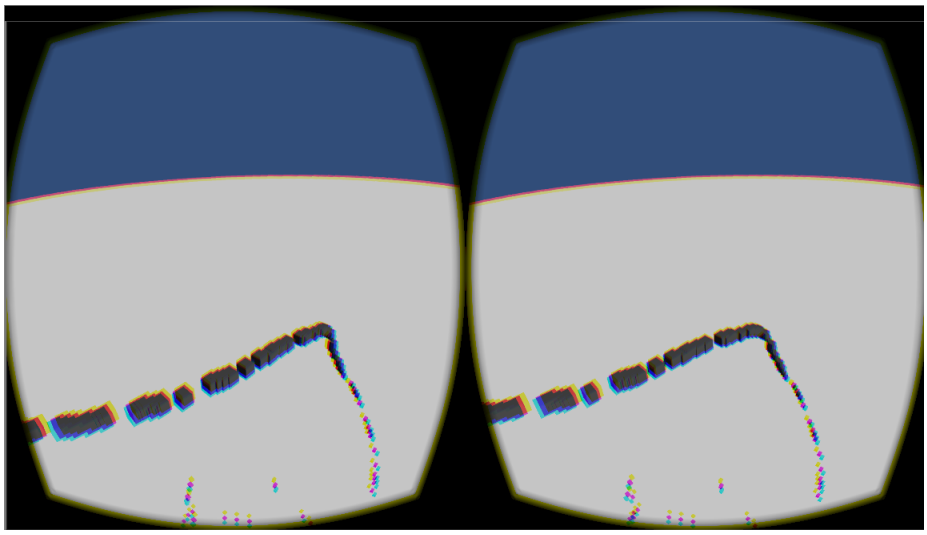
Le rendu des lignes étant peu satisfaisant d'un point de vue esthétique, nous avons cherché d'autres méthodes de rendu. Nous nous sommes penchés notamment sur le plugin Vectrosity, suggéré par Xavier Wielemans. Nous avons repris le même principe que l'affichage avec des objets en les remplaçant par des fragments de ligne. Malheureusement, nous n'avons pas eu suffisamment de temps pour pouvoir développer une version fonctionnelle avec cette méthode.

3.5. Intégration de l'Oculus Rift

L'intégration en elle-même de l'Oculus Rift sous Unity fut une chose plutôt facile. En effet, il existe sur le site developer.oculus.com un package déjà prêt à être implanté. Nous avons donc téléchargé et installé le runtime Oculus, puis nous avons téléchargé le package Unity.

Pour utiliser ce package il suffit de sélectionner « Import package… » puis de choisir celui que nous venons de télécharger. Cela permet donc d'avoir directement accès dans l'éditeur aux éléments liés à l'Oculus.

Au lieu de choisir une caméra standard comme vue dans notre environnement graphique nous avons donc pris l'élément OVRPlayerController permettant d'obtenir le rendu de vision interne de l'Oculus.



Rendu visuel de l'élément OVRPlayerController sur un écran d'ordinateur

Nous avons ensuite placé cet élément sur le centre du repère défini dans Unity qui correspond également au centre du repère ART-Track.

3.6. Problèmes récurrents

Nous avons rencontré plusieurs problèmes récurrents lors de ce projet.

Le programme que nous avons écrit sous Unity ne s'exécute parfois pas correctement. Pour des raisons inconnues il est possible que parfois l'affichage dans l'Oculus Rift ne se fasse pas. Le programme est lancé mais les images n'apparaissent pas.

La vision de l'Oculus Rift avait une probabilité assez importante pour se positionner à des endroits imprévus. En effet nous avons souvent eu une vision très haute des objets dessinés, pour finalement se repositionner normalement après un certain temps.

Aussi, avec la première version d'affichage de lignes, utilisant des objets ponctuels (sphères / cubes), nous avons été étonnés du peu d'objets que la machine soit capable de gérer. En effet lorsqu'on compare notre scène à la qualité et le niveau de détail des jeux vidéo actuels, il est étonnant qu'un ordinateur capable de faire fonctionner ce type de programme ne soit pas capable d'afficher fluidement plus de 3000 objets simples tels que des sphères.

Conclusion

La réalisation de ce projet nous a permis de travailler sur un sujet motivant et enrichissant. Le fait de travailler avec des personnes venant d'un milieu artistique fut également profitable car cela nous a amené un point de vue et une approche différents de ceux auxquels on est habitués. Cette expérience fut très enrichissante pour nous car elle nous a permis de découvrir un nouveau langage informatique et de travailler avec du matériel performant.

Nous avons bien réussi à mener ce projet jusqu'au bout malgré deux semaines où nous n'avons pas pu accéder au matériel. Cela nous a par contre malheureusement empêché de pouvoir apporter certaines améliorations que nous aurions souhaité effectuer.

Ce projet servira à Pauline pour son projet de fin d'année, et sera également utilisé pour une exposition temporaire à l'Imaginarium de Tourcoing en septembre. Cependant, d'autres utilisations pourraient être envisagées, comme par exemple s'en servir comme base pour développer une application pour l'architecture ou des jeux vidéo 3D interactifs.

Glossaire

Effet stéréoscopique : L'effet stéréoscopique est le fait de projeter deux images décalées à un utilisateur de manière à reproduire le décalage des yeux. Cela a pour but que la personne puisse voir en 3 dimensions dans l'environnement projeté.

Frame : Frame est à l'origine la traduction anglaise d'une image parmi une série constituant une vidéo. Dans notre cas nous avons étendu cette définition à l'objet C# frame, qui contient les informations de chaque série de mesures et les méthodes nécessaires à leur utilisation.

Magnétomètre : un magnétomètre est un capteur servant à mesurer le champ magnétique ambiant.

Mode release : En opposition au mode debug, le mode release est un mode où le programme n'a plus la possibilité d'être débugué. En contrepartie le programme n'utilise plus de ressources pour pouvoir être débugué et il est optimisé pour l'exécution. Il est donc plus rapide.