

RAPPORT D'INGÉNIEUR
STAGE DE FIN D'ÉTUDES
FILIÈRE F4 : CALCUL ET MODÉLISATION SCIENTIFIQUES

Rendu d'un nuage de points en monde collaboratif et immersif

Présenté par : Sandy LE PAPE

Tuteur de projet :

Matt ADCOCK

Professeur référent :

Vincent BARRA

Date de la soutenance :

27 août 2019

Durée du stage :

5 mois et 2 semaines

Année scolaire 2018-2019

Remerciements

J'aimerais tout d'abord remercier mon tuteur d'entreprise, Matt ADCOCK, qui m'a accordé sa confiance tout au long du projet, bien que ma spécialisation était relativement éloignée du domaine d'application du laboratoire. Je lui suis également infiniment reconnaissant d'avoir pris le temps de trouver un financement pour le stage ; De plus, il s'est intéressé à mes envies professionnelles afin de construire un projet, à la fois adapté pour moi et utile aux objectifs du laboratoire. J'aimerais également remercier Stuart ANDERSON, Simon MALNAR et Mingze XI qui ont su me conseiller sur les divers aspects techniques de mon travail et de ces technologies, nouvelles pour moi jusqu'alors. D'autre part, je tenais à remercier chaudement Brian CHEN, dynamique et passionné, qui m'a proposé son aide de nombreuses fois. Dans le cadre de ce stage, je remercierai également le concepteur de l'outil de visualisation de données *ImAxes*, Maxime CORDEIL, outil qui a été une base solide pour le projet. Je remercie également Vincent BARRA en tant que professeur pédagogique référent du projet, pour sa grande réactivité et sa disponibilité, et m'ayant permis d'initier en chaîne le contact avec Olivier SALVADO, Dadong WANG puis mon tuteur. Mes remerciements vont à Pascale GRANET et Megan MCDONALD pour la gestion de l'ensemble des papiers de stage. J'aimerais remercier l'ISIMA, qui m'a permis d'acquérir une partie des bases mathématiques et informatiques indispensables à la pleine compréhension du projet, ainsi que le CSIRO, lieu de travail épanouissant ayant un profond respect pour la nature et l'individu. Enfin, je souhaite une bonne continuation à tous les membres de l'équipe que j'ai intégrée pour ce stage.

Table des figures

1	Bâtiment <i>Synergy</i> du CSIRO à Canberra	1
2	Inventions les plus marquantes du CSIRO	2
3	Logo du laboratoire	3
4	Exemple d'une visualisation de données immersive [1]	5
5	Système unifié créé par le CSIRO pour le programme NEAR [2]	6
6	Diagramme de Gantt réel	6
1.1	Correspondance entre mondes réel et virtuel	7
1.2	Exemple d'un kit complet de RV VIVE PRO	8
1.3	Environnement de jeu	8
1.4	Jeu AR HoloChess tiré de la Guerre des Etoiles	9
1.5	Casque HoloLens et Menu d'accueil	10
1.6	Interactions basiques HoloLens	11
1.7	Fenêtres du moteur de jeu Unity	12
1.8	Fenêtre pour <i>push / pull</i> des modifications	13
1.9	SteamVR	13
1.10	Panorama des fonctionnalités de MRTK	15
1.11	Ensemble de graphiques visualisés dès l'ouverture de l'application [5]	16
1.12	Construction de nuages de points [5]	16
1.13	Représentation par coordonnées parallèles	17
1.14	Représentations émergentes [5]	17
1.15	Les deux types de curseurs existants	18
1.16	Les différentes démos de l'outil [6]	19

1.17	Scène obtenue après indication de la Bing key, longitude et latitude de Canberra	19
1.18	Densité de population en Australie (étude fournie par AREMI) [8]	21
1.19	Besoin de chauffage et de climatisation en Australie (étude fournie par NEAR) [9]	22
2.1	Comparaison de l'architecture réseau entre UNet et PUN	24
2.2	Salons et sessions en Photon PUN	25
2.3	Droit de possession et mises à jour sur le Cloud en Photon PUN	26
2.4	Diagramme Bloc de l'enchaînement des scènes	27
2.5	Informations de connexions possibles	28
2.6	Menu initial et erreurs indiquées	28
2.7	Différents points de vue pour le joueur	29
2.8	Orientation du nom du joueur selon le regard du joueur	30
2.9	Changement de couleur du cube selon les <i>colliders</i>	31
2.10	Différence des points d'ancrage selon la fonction utilisée	32
2.11	Cube visualisé sur l'écran vue Scène du computer 2	33
2.12	Zone de jeu demandée par SteamVR	34
2.13	Décalage persistant entre les cubes représentant les <i>stations</i>	35
2.14	Evolution de la position des <i>stations</i>	36
2.15	Composants d'un adaptateur sans fil Wi-Fi VIVE	37
3.1	Pipeline graphique de Direct3D	39
3.2	Principe de fonctionnement du rendu en Unity3D [12]	40
3.3	Principe de fonctionnement du <i>shader vertex & fragment</i> en Unity [12]	41
3.4	Calcul des deux premiers points du polygone	42
3.5	NUAGE de points sphérique et ses caractéristiques	43
3.6	NUAGE de points de données et ses caractéristiques	45
3.7	Différents shaders disponibles sous MRTK	45
3.8	Illustration du principe de clipping	46
3.9	Distance du point au plan par projection	47

3.10	Illustration du <i>backface culling</i>	48
3.11	Résultats pour le plan coupant	48
3.12	Problème remarqué avec des plans coupants infinis	49
3.13	Evolution des frontières d'un plan selon la modification apportée	50
3.14	Plan coupant selon la normale au plan colinéaire à l'axe x . . .	51
3.15	Résultats obtenus pour la boîte coupante	52
3.16	Résultats du plan coupant avec boîte accolée	52
3.17	Illustration des <i>scaptics</i> et des <i>highlight planes</i> [14]	53
3.18	Illustration de la fonctionnalité de mise en surbrillance	54
3.19	Premier prototype des Highlight planes	54
3.20	Modèle de réflexion lambertienne et vecteurs utiles	55
3.21	Résultats pour le modèle de réflexion lambertienne	56
3.22	Modèle de réflexion Blinn-Phong et vecteurs utiles	56
3.23	Pipeline graphique de l'application	57
1	Différences entre RV, RA et RM	xv
2	Continnum de Milgram	xv
3	Améliorations de l'analyse disponible dans ImAxes [1]	xvii
4	Représentations innovantes, non présentes dans ImAxes [1] . . .	xviii
5	Couches HLAPI réseau UNet	xix

Résumé

Afin de maîtriser pleinement l’analyse de grands jeux de données, de nombreux secteurs souhaiteraient pouvoir interagir en temps réel avec les visualisations correspondantes. Depuis quelques années, le domaine des *analytics immersifs*, à mi-chemin entre technologies virtuelles et analyse de données, offre une interactivité intuitive et plus souple avec la donnée que les logiciels d’ordinateur. Cependant, ces dernières exploitent rarement la possibilité de vivre une expérience à plusieurs.

Le stage m’a permis d’implémenter une solution réseau et d’apporter des améliorations concernant la perception de la donnée en environnement immersif, en travaillant notamment sur le rendu graphique. La fin du stage sera pleinement consacrée à l’intégration de mon travail à un outil de visualisation déjà existant.

Mots-clés : *analytics immersifs*, technologies virtuelles, solution réseau, rendu

Abstract

In order to fully master the analysis of large datasets, many sectors would like to be able to interact in real time with the corresponding visualizations. In recent years, the field of *immersive analytics*, combination of virtual technologies and data analysis, has offered intuitive and more flexible interactivity with data than computer softwares. However, they rarely exploit the possibility of a group experience.

This internship allowed me to implement a networked solution and to make improvements concerning the perception of the data in an immersive environment, by working on the graphic rendering in particular. The end of the internship will be fully dedicated to the integration of my work into an existing data visualization tool.

Keywords : *immersive analytics*, virtual technologies, networked solution, rendering

Table des matières

Remerciements	ii
Table des illustrations	vi
Résumé	viii
Abstract	viii
Table des matières	xii
Présentation de l'organisation	1
Introduction	4
1 Etat des Lieux et premiers apports	7
1.1 Présentation des concepts et technologies de Réalité Virtuelle	7
1.1.1 La Réalité Virtuelle et Immersive	7
1.1.2 Bref aperçu de la Réalité Augmentée	9
1.1.3 La Réalité Mixte : entre Réalité Augmentée et Mixte	9
1.2 Présentation des outils de développement et de modélisation	11
1.2.1 Unity, moteur de jeu multi-plateforme	11
1.2.2 Les outils et packages de la Réalité Virtuelle	13
1.2.3 Les outils et packages de la Réalité Mixte	14
1.3 Etat de l'Art des outils de visualisation et d'exploration de données . . .	15
1.3.1 ImAxes : outil interactif d'exploration de données multivariées . .	15
1.3.1.1 Présentation de l'outil	15
1.3.1.2 Contributions	18

1.3.2	MapsSDK-Unity : outil de visualisation de données sur une carte du monde	19
1.4	Les jeux de données à disposition	20
1.4.1	AREMI, visualisation de données en Australie	20
1.4.2	Le programme NEAR	21
2	Construction d'un environnement collaboratif en Réalité Virtuelle	23
2.1	Développement d'une application réseau sous Unity	23
2.1.1	Tour d'horizon des différentes possibilités réseau	23
2.1.2	Concepts fondamentaux de PUN, API de développement réseau pour Unity	24
2.2	Architecture utilisée et fonctionnalités implémentées	26
2.2.1	Enchaînement des scènes	26
2.2.2	Interface graphique de communication réseau	27
2.2.3	Modélisation et comportement des différents utilisateurs	28
2.2.4	Interaction avec les GameObjects de la scène	30
2.3	Difficultés rencontrées inattendues	32
2.3.1	Simulation physique des différents GameObjects	32
2.3.2	Correspondance entre le monde 3D de chaque utilisateur	34
2.3.3	Les problèmes causés par le dispositif Wi-Fi	36
3	Rendu graphique pour la visualisation de données collaborative	38
3.1	Fonctionnement du <i>pipeline</i> graphique sous Unity	38
3.1.1	Processus de rendu graphique et principe des <i>shaders</i>	38
3.1.2	Design des <i>shaders</i> en Unity et présentation des principaux	39
3.2	Représentation graphique des données	42
3.2.1	Création d'un nuage de points et rendu	42
3.2.2	Représentation des données par ce nuage	44
3.3	Amélioration de l'exploration visuelle des données	45
3.3.1	Exploration de données par plans coupants	46
3.3.2	Plans coupants finis et problèmes rencontrés	49
3.3.3	Implémentation d'un prototype d'Highlight Planes	53
3.3.4	Perception de la vision d'un utilisateur par tous	57

3.4 Objectifs et perspectives d'évolution pour la fin du stage	57
Conclusion	59
Bibliographie	xiii
Annexes	xv

Présentation de l'organisation

Ce stage m'a été confié par le CSIRO (Commonwealth Scientific and Industrial Research Organisation), l'agence du gouvernement fédéral australien en charge de la recherche scientifique, dont la mission est de relever les plus grands défis d'innovation scientifiques et technologiques pour le territoire australien. Son siège social se situe à Canberra, la capitale, dans le quartier d'Acton. Cette ville accueille un très faible nombre de sièges sociaux industriels, principalement dû au fait que l'économie du pays est davantage présente dans les villes telles que Sydney, Melbourne ou encore Brisbane. Le CSIRO a son propre campus, accessible à tous, les bâtiments principaux étant *Synergy* et *Discovery*. La particularité de ce campus est de mélanger les bâtiments spécialisés pour la culture d'espèces vertes, la chimie et le développement informatique.



FIGURE 1 – Bâtiment *Synergy* du CSIRO à Canberra

La première forme du CSIRO est née en 1916 sous la forme d'un conseil consultatif pour la Science et l'Industrie à Melbourne. L'Institut a évolué durant les dix années suivantes pour devenir le CSIR (Council of Scientific and Industrial Research). Toutes ces années, la vision et les objectifs sont restés les mêmes : apporter un appui par la recherche

au développement des industries fermières, minières et de manufactures dans toute l'Australie. Les champs d'application ont évolué au fil des années, se tournant davantage vers la vie animale et florale, mais également le traitement de matériaux à la suite de la Seconde guerre Mondiale. Durant cette guerre, le CSIR a soutenu les forces défensives australiennes, jusqu'en 1949 où le CSIR a été renommé CSIRO (Commonwealth Scientific and Industrial Research Organisation), se recentrant sur les préoccupations environnementales du pays. Au fil des années, les domaines de recherche de l'Institut se sont élargis, conduisant des recherches sur l'urbanisation, l'eau ou encore la nutrition. Depuis 2014, afin de clarifier la lecture stratégique du laboratoire, le CSIRO s'est architecturé autour de trois axes principaux : la recherche scientifique (contenant neuf unités regroupant les spécialités abordées précédemment), les infrastructures biologiques et les services tertiaires.

Tout au cours de l'histoire, le CSIRO a su apporter des évolutions scientifiques et technologiques considérables, dont le rayonnement a largement dépassé les frontières australiennes. L'invention la plus célèbre de l'Institut, et que peu d'articles soulignent comme étant issue du CSIRO, est la Wi-Fi. Maintenant utilisée en masse dans de nombreux endroits du monde, ses travaux de recherche se sont basés sur la transformée de Fourier et sur l'étude du comportement des ondes radio selon l'environnement dans lequel elles se propagent. Une autre invention, utilisée partout en Australie, est le billet de banque en plastique, conçu par le laboratoire. Ces billets ont la particularité d'avoir une durabilité bien plus importante que les précédents, sont faits à partir de polymères (donc plus écologiques), et sont moins susceptibles de transporter la saleté. Au début des années 1990, des lentilles de contact à durée de vie d'un mois ont été conçues. D'autres découvertes, dont quelques vaccins, ont illustré à maintes reprises le talent des personnes travaillant dans l'organisation.

Our track record: top inventions



FIGURE 2 – Inventions les plus marquantes du CSIRO

Le CSIRO est composé de multiples organisations, dont Data61, souhaitant relever les défis *big data* d'aujourd'hui par une maîtrise de la Data Science en lien avec les technologies. Data61 est née en 2015, d'une volonté forte du CSIRO de favoriser une collaboration entre industries, laboratoires et universités. De la Cybersécurité à l'Intelligence Artificielle, en passant par le Traitement d'Images, cette association ingénieuse de domaines a permis d'apporter une aide précieuse dans l'analyse et la prédition de données pour les milieux médical, agricole et des télécommunications.

J'ai effectué mon stage au laboratoire d'environnements immersifs à Canberra, faisant partie intégrante de Data61. Ce laboratoire est spécialisé dans les domaines de la Réalité Virtuelle et Augmentée. Cet endroit est un véritable catalyseur d'innovation qui dispose d'un financement important lui permettant d'obtenir les technologies dernier cri, afin de stimuler l'envie des chercheurs de maîtriser les technologies de demain. L'équipe du laboratoire accueille régulièrement le personnes provenant du milieu industriel et le public amateur pour des événements, des manifestations et des démonstrations.



FIGURE 3 – Logo du laboratoire

Dernièrement, le laboratoire d'environnements immersifs s'est concentré sur les possibles moyens d'améliorer la lisibilité de masse de données, par de nouveaux analytics et de nouvelles visualisations dans des mondes virtuels 3D, qu'on englobe sous le terme d'*analytics immersifs*. Ce travail s'effectue en coopération avec d'autres universités australiennes, le gouvernement et certaines entreprises intéressées. Le laboratoire, de part le matériel qu'elle dispose dans le domaine de la Réalité Virtuelle et Augmentée et ses locaux récents, permet chaque année à quelques étudiants, de se familiariser avec ces technologies, et j'ai la chance, cette année, de pouvoir en faire partie.

Introduction

Dans une époque où la donnée est de plus en plus disponible, hétérogène et complexe à décrire, il est nécessaire de pouvoir traiter et visualiser cette donnée de différentes manières afin d'en faire ressortir une information de valeur. Pour ce faire, des représentations graphiques telles que les histogrammes ou encore les nuages de points sont utilisées afin de déceler des tendances générales après application de méthodes statistiques multivariées ou d'apprentissage par exemple. Toutefois, les jeux de données servant à construire ces représentations sont de grande taille et ont de multiples formes, impliquant l'utilisation de techniques de fusion de données afin de pouvoir les comparer. La question qui se pose est alors de savoir si cette fusion peut être automatisée et fiable ou si l'Homme devra, à un moment ou un autre, intervenir.

D'autre part, les technologies de Réalité Virtuelle et de Réalité Augmentée ne cessent de se développer, entre les opportunités qu'elles offrent pour augmenter la précision d'opérations chirurgicales ou permettre une immersion totale dans un jeu vidéo. L'interactivité simple d'esprit (utilisation de manettes ou gestuelle) et l'immersion que ces technologies apportent, permettent de profiter de fonctionnalités d'un tout nouveau genre.

Combinées avec la visualisation, l'exploration et l'analyse de données, ces technologies permettent de franchir un nouveau cap dans l'interactivité : il est désormais possible, par un simple geste de la main, de manipuler facilement des graphiques ou tout autre objet virtuel ancré sur le monde réel, comme le montre la FIGURE 4. Toutefois, l'utilisation de ces nouvelles technologies amènent de nouvelles problématiques. La visualisation collaborative de données est compliquée, notamment en Réalité Virtuelle car le monde de chaque utilisateur est généré à partir de différents ordinateurs, et n'ont ainsi pas forcément le même repère. D'autre part, chaque utilisateur dispose de sa propre visualisation d'une scène de son point de vue précis, et ne peut être partagée avec le même angle aisément aux autres joueurs. Aussi des problématiques d'*Infographie* intéressantes à creuser se posent à ce moment.

Le CSIRO étant très orienté dans les applications environnementales liées au pays, le projet s'est orienté autour d'une préoccupation concernant le domaine énergétique. Consi-

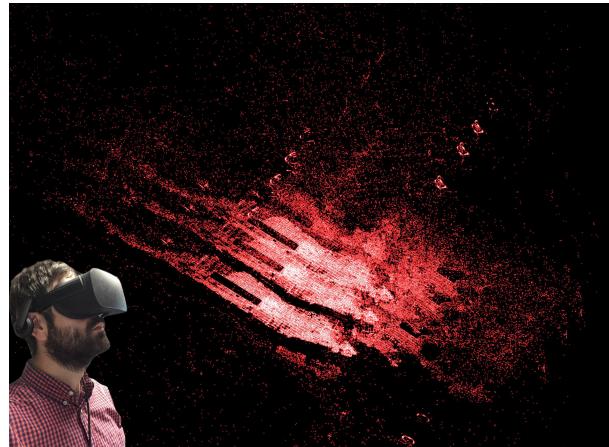


FIGURE 4 – Exemple d'une visualisation de données immersive [1]

dérons le domaine de la construction par exemple : selon le bâtiment considéré (maison, appartement), son ancienneté, sa location sur le territoire, comment est-il possible, à l'aide de jeux de données, de mesurer interactivement l'impact énergétique ces dernières années et à quel point la lecture de l'utilisateur peut être facilitée ? Cet outil d'aide à la décision se basera sur une carte de l'Australie, des jeux de données complets ainsi que des outils de visualisation, et fera apparaître, selon la localisation, une visualisation de données (diagrammes en bâtons, séries temporelles,...).

La finalité de ce projet est, à partir de jeux de données relatifs à la consommation énergétique, de visualiser sur une carte 3D de l'Australie l'impact de cette utilisation. Elle analysera les données récoltées afin de trouver des moyens d'optimiser cette consommation, et éventuellement de la prédire. Ce projet, à visée collaborative, fait émerger de nouvelles problématiques quant à la visualisation multi-utilisateur. Il s'agit donc aussi de trouver des solutions innovantes afin d'améliorer la qualité des visualisations, en interagissant directement avec la carte graphique. Ce projet dense et à grande échelle, intitulé "*Livraison d'un asset de données intégrées pour l'avenir de la prédition énergétique*", se base sur le projet NEAR, une plate-forme qui a permis de renforcer nos connaissances quant au changement de comportement énergétique en Australie (son fonctionnement est décrit FIGURE 5). Il implique le travail de plusieurs équipes du CSIRO dans toute l'Australie.

Dans un premier chapitre, il sera présenté l'ensemble des logiciels et outils utilisés pour le projet, permettant l'exploration et l'analyse interactive de données multivariées en environnement virtuel. Il a ensuite été décidé qu'apporter une dimension collaborative à l'un de ces outils, considéré prometteur et réglé pour la Réalité Virtuelle, serait intéressant. Enfin, des recherches ont été menées afin d'améliorer les possibilités d'analyse de la donnée ainsi que le rendu de visualisation de la donnée pour chaque joueur.

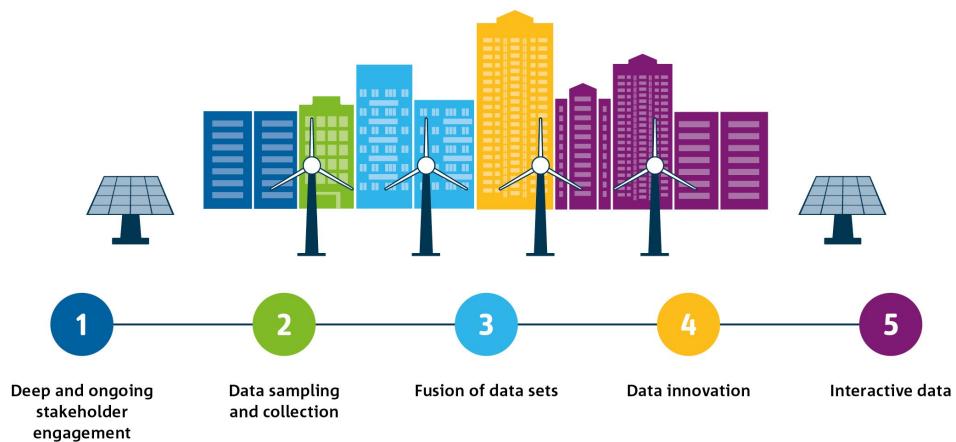


FIGURE 5 – Système unifié créé par le CSIRO pour le programme NEAR [2]

Les attentes du laboratoire n'ayant pas été fixées dès le début, le *Cahier des Charges* établi en collaboration avec mon tuteur s'est adapté en fonction des priorités et de mes objectifs professionnels sur toute la longueur du stage. Le tracé d'un diagramme de Gantt prévisionnel n'est donc pas possible. En revanche, voici le diagramme de Gantt réel (FIGURE 6) :

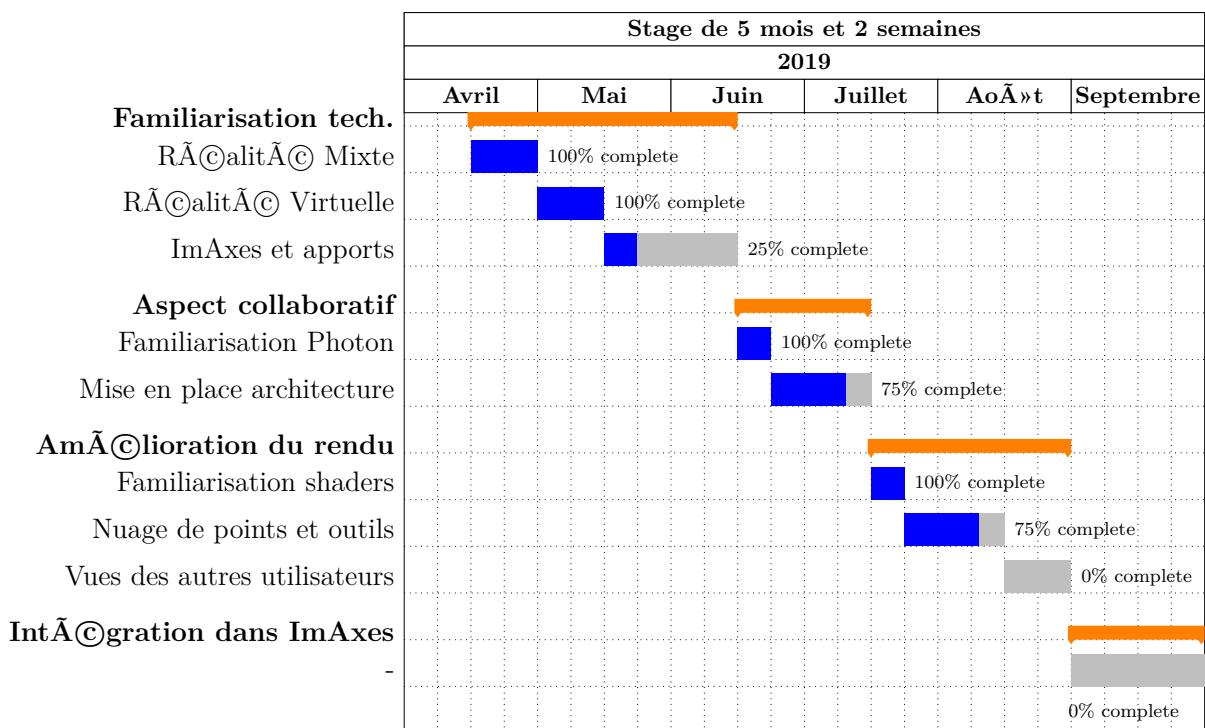


FIGURE 6 – Diagramme de Gantt réel

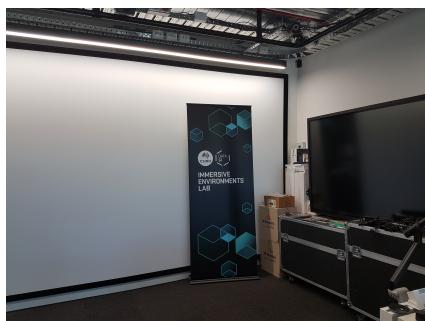
Chapitre 1

Etat des Lieux et premiers apports

1.1 Présentation des concepts et technologies de Réalité Virtuelle

1.1.1 La Réalité Virtuelle et Immersive

La Réalité Virtuelle (RV) peut être vue comme le terme générique qui rassemble toute expérience immersive, rendue possible par la création d'un contenu exclusivement artificiel, réel, ou d'un hybride des deux. La RV se caractérise principalement par le fait de créer l'illusion de plonger l'utilisateur dans un monde artificiel 3D, qui n'était visible que d'extérieur par écran interposé auparavant. Ainsi, lorsque l'utilisateur porte le casque de RV, son champ de vision est substitué par un autre permettant de visualiser le monde virtuel 3D (comme illustré en FIGURE 1.1). Le joueur peut interagir via l'utilisation de manettes (appelées *motion controllers* en anglais).



(a) Monde réel



(b) Substitution par le monde 3D

FIGURE 1.1 – Correspondance entre mondes réel et virtuel

Tout au long du projet, les casques VIVE PRO conçus par Valve Corporation et HTC ont été ceux utilisés (FIGURE 1.2). D'autres casques de RV existent, les concurrents les

plus connus étant Oculus et Microsoft.



FIGURE 1.2 – Exemple d'un kit complet de RV VIVE PRO

La plupart des technologies offrant des expériences de RV impose la connexion et l'initialisation de différents appareils représentés en FIGURE 1.3 :

- des capteurs (appelés *base stations* ou lighthouses en anglais) (A) aidant à la mise à l'échelle de l'espace 3D dans lequel l'expérience va se dérouler, traquant la position exacte des contrôleurs et du casque,...
- des contrôleurs (B), seul outil d'interaction avec le monde virtuel, contenant divers boutons, gâchettes et pad analogique variant selon les modèles.
- le branchement de câbles (display port, usb port et power) du casque (C) à l'ordinateur (E) via une boîte (D).

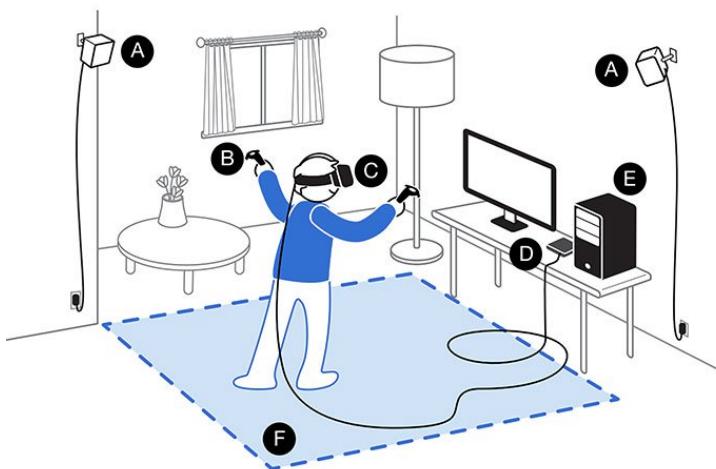


FIGURE 1.3 – Environnement de jeu

Mise à part la RV, d'autres technologies peuvent altérer la réalité sans pour autant immerger le joueur dans un autre monde.

1.1.2 Bref aperçu de la Réalité Augmentée

La Réalité Augmentée (RA), comme son nom l'indique, ne substitue pas le champ de vision, contrairement à la RV, mais l'enrichit avec des objets virtuels, comme un écran virtuel informant la météo en temps réel,...

Les technologies permettant d'offrir des expériences de RA sont bien moins coûteuses que celles de RV. Un simple smartphone contenant les applications dédiées peut lancer des jeux RA, PokemonGO et HoloChess (FIGURE 1.4) étant probablement les applications de RA les plus connues à ce jour. Des lunettes 3D peuvent également être utilisées, comme celle de Google par exemple. En cela, une séance de cinéma 3D (par port de lunettes stéréoscopiques) peut être considérée comme une expérience de RA.



(a) Jeu HoloChess conçu via ARKit

(b) Jeu lancé sur Smartphone

FIGURE 1.4 – Jeu AR HoloChess tiré de la Guerre des Etoiles

Enfin, Microsoft a été capable d'étendre ce qui existait entre RV et RA auparavant.

1.1.3 La Réalité Mixte : entre Réalité Augmentée et Mixte

La Réalité Mixte (RM ou RX) est souvent considérée comme la résultat de la fusion entre Réalité Virtuelle et Augmentée, dans le sens où elle permet l'interaction avec des objets virtuels couvrant le monde réel. Leur placement, leur orientation et leur physique est basée sur le monde réel. Cela peut uniquement être permis avec des rendus haute-performance dépassant les besoins en RA. L'ANNEXE 1 aide à clarifier les différences importantes entre RV, RA et RM, notamment grâce au *continuum Réalité-Virtualité*.

Les technologies Microsoft HoloLens (FIGURE 1.5) appartenant à la WMR (pour *Windows Mixed Reality* en anglais) a mis l'accent sur le potentiel de la RM. Elles sont davantage destinées aux développeurs car la technologie nécessite encore quelques évolutions pour que le grand public puisse suffisamment les maîtriser. Les travaux effectués ont

été principalement effectués sur Microsoft HoloLens 1. L'un des plus gros avantages des casques HoloLens est de fonctionner comme appareil *standalone* (autonome), offrant ainsi une expérience inédite.

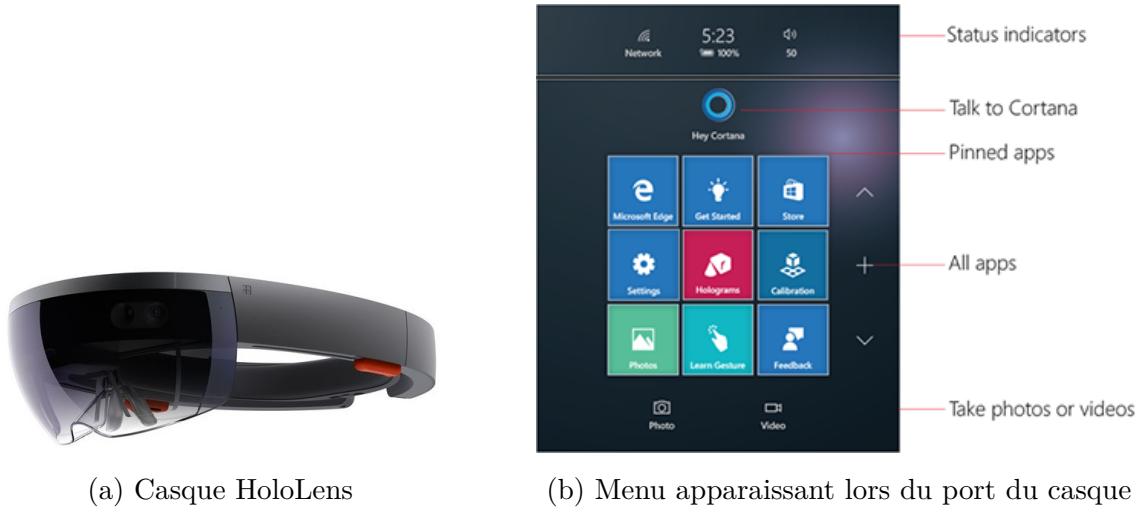


FIGURE 1.5 – Casque HoloLens et Menu d'accueil

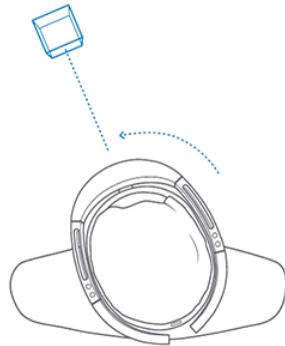
Les applications RM peuvent aussi être lancées sur les casques immersifs WMR, et nécessite juste un supplément de code pour rendre les contrôleurs fonctionnels avec elles.

La fonctionnalité primaire guidant l'utilisateur dans ses interactions est une visée appelée *gaze*. Elle permet de simuler un point suivant les mouvements de la tête de l'utilisateur portant le casque dans le monde réel. Le gaze a été conçu comme un laser pointant droit devant le joueur.

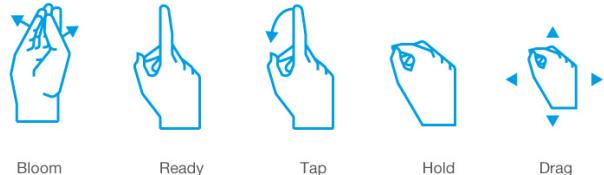
C'est au moment où le *gaze* intersecte un hologramme qu'une action peut être entreprise. Trois possibilités s'offrent alors à nous :

- la gestuelle *bloom*, réservée exclusivement pour revenir au menu principal HoloLens,
- la gestuelle *air-tap*, permettant la sélection (assimilable au clic sur l'ordinateur),
- la commande vocale, offrant des possibilités infinies puisque la reconnaissance vocale HoloLens est avancée. Il suffit de prononcer un mot en langue anglaise et de l'associer à un code pour que l'action soit traduite lorsque l'application est lancée.

Il est possible de créer ses propres gestuelles, qui seraient une combinaison des gestuelles primaires, mais Windows le déconseille fortement car viole l'encapsulation de la technologie. Elle recommande les gestuelles composites prédéfinies comme le *tap and hold* (équivalent au *clic and drag* sur un ordinateur), ou la manipulation pour la rotation ou la mise à l'échelle d'un hologramme, comme montré en FIGURE 1.6.



(a) Le *gaze* HoloLens



(b) Gestuelles classiques HoloLens

FIGURE 1.6 – Interactions basiques HoloLens

Quelques outils vont s'avérer utiles ou fortement conseillés pour développer des applications.

1.2 Présentation des outils de développement et de modélisation

1.2.1 Unity, moteur de jeu multi-plateforme

Unity [1] est un moteur de jeu développé par Unity Technologies, dont la première version est sortie courant 2005. Sa rapidité de prise en main et sa facilité à importer un jeu sur plus d'une vingtaine de plateformes font de lui l'un des moteurs de jeu les plus appréciés. Il permet également aux développeurs d'implémenter des applications pour les casques de RV et RM principalement en C# [3].

La particularité d'un jeu sous Unity est d'être conçu à partir de briques fondamentales :

- les *GameObjects*, correspondant aux éléments rendus dans le jeu ;
- les *Components*, correspondant aux caractéristiques des *GameObjects*, soit préfabriqués, soit conçus par nos propres soins via les scripts.

Le fonctionnement de Unity repose sur quelques composants fondamentaux que voici :

- pour translater, faire tourner ou modifier l'échelle d'un *GameObject* dans l'espace 3D ou accéder aux informations du parent ou d'un enfant relatifs, le component *Transform* est indispensable ;
- pour qu'un *GameObject* soit sensible à la Physique dans le monde 3D, le Component *Rigidbody* contient les lois basique de la Physique gérées par Unity ;

- pour associer un maillage à un GameObject, lui appliquer une texture ou obtenir aisément ses frontières, *Mesh filter*, *Mesh Renderer* ou *Mesh Collider* sont utilisés, respectivement.

Il est possible de sauvegarder la configuration d'un GameObject avec l'ensemble de ses components et réglages : ces objets sont appelés *prefabs*.

La création d'un nouveau projet Unity fait apparaître l'éditeur qui se décompose en 7 fenêtres (FIGURE 1.7), décrit ci-dessous :

- la barre de menus : permet d'accéder à l'ensemble des réglages du moteur de jeu ;
- Hierarchy : permet de visualiser l'arborescence de l'ensemble des GameObjects présents dans la scène ;
- Project : affiche le répertoire du projet contenant l'ensemble des dossiers, les obligatoires étant Assets (contenant tous nos modèles 3D, les scripts de code,...), Packages et Project Settings ;
- Inspector : précise tous les paramètres modifiables ainsi que les Components relatifs à chaque item (GameObjects, Matériaux,...),
- Scene : affiche l'espace 3D construit pour le jeu ;
- Game : rend graphiquement la scène du point de vue de la caméra.

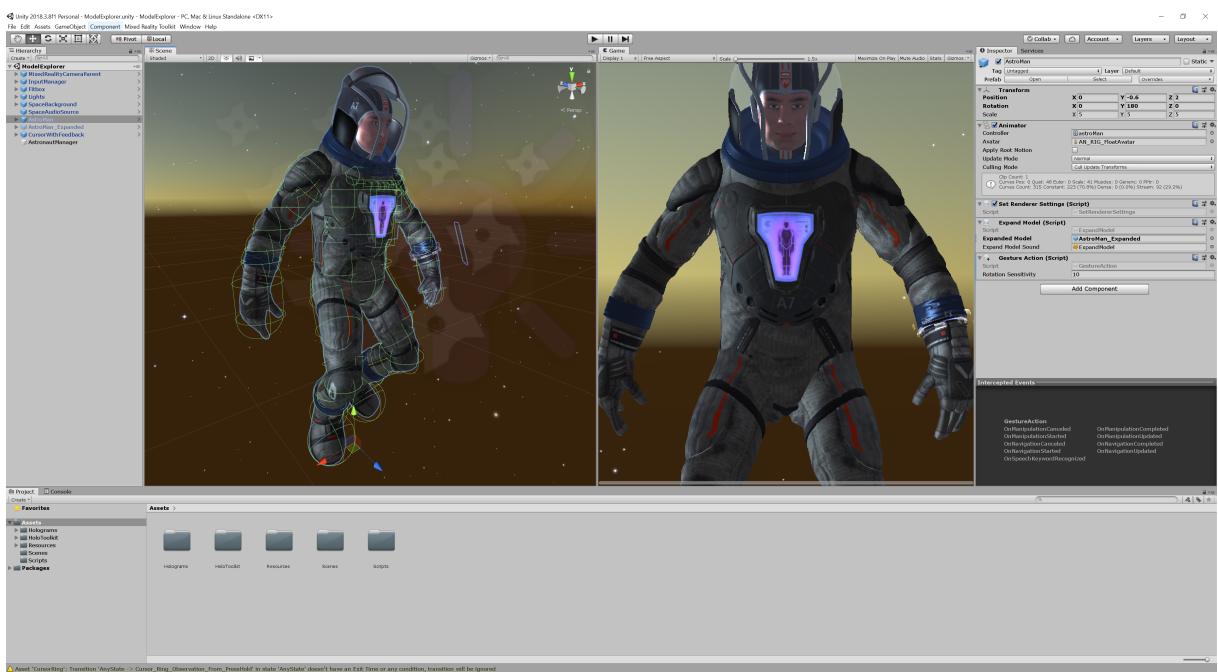


FIGURE 1.7 – Fenêtres du moteur de jeu Unity

Pour maintenir des versions de code fonctionnelles, Unity Collaborate (FIGURE 1.8) a été utilisé, au détriment des services de gestion de versions Git, utilisé uniquement pour

partager du code avec l'ensemble de l'équipe au sein du CSIRO via BitBucket et les pages Confluence.

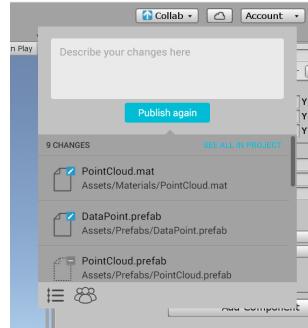


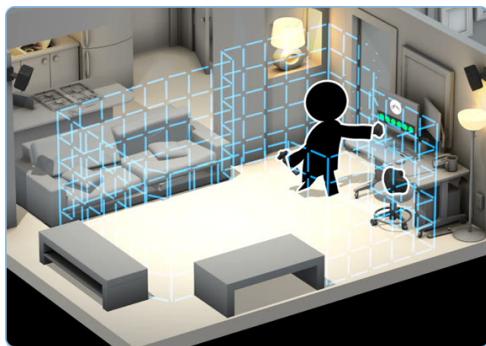
FIGURE 1.8 – Fenêtre pour *push / pull* des modifications

C'est la version 2017 de Visual Studio qui a été utilisée pour coder les modules supplémentaires ajoutés à *ImAxes*. Il dispose de nombreux outils de qualité, dont un débogueur, indispensable pour des codes d'une telle ampleur.

Développer une application de RV sous Unity nécessite l'installation de différents éléments.

1.2.2 Les outils et packages de la Réalité Virtuelle

Pour commencer, l'application SteamVR est nécessaire si l'utilisateur souhaite vivre une expérience de RV sur un hardware Valve. Cela inclut l'initialisation de la technologie VIVE et la zone de jeu doit être tracée (FIGURE 1.9(a)). Les différentes composantes de la technologie apparaissent en vert dans une petite fenêtre s'ils sont détectés par l'ordinateur (FIGURE 1.9(b))



(a) Etape de délimitation de la zone de jeu



(b) Fenêtre SteamVR

FIGURE 1.9 – SteamVR

L'un des autres outils clé est OpenVR, une API libre de droits proposant un ensemble de méthodes virtuelles pures C++ à surcharger. Elle est largement utilisée car son grand avantage est d'offrir une interaction simple avec un casque de RV, et ce quelque soit sa marque. Un code utilisant les fonctionnalités d'OpenVR peut donc être lancé sur n'importe quel casque, sans devoir tenir une version de code par hardware.

Enfin, le projet VRTK (pour *Virtual Reality ToolKit* en anglais), également libre de droits, met à disposition un ensemble de scripts et de prefabs permettant le développement d'applications de RV plus rapidement, en fournissant notamment des solutions d'interactivité (avec les *contrôleurs* et via des boutons) ou encore la physique.

Si l'on souhaite développer une application MR, nous devons importer différents *packages* et initialiser différents paramètres dans la barre des menus afin de ne pas s'exposer à un nombre d'erreurs considérable.

1.2.3 Les outils et packages de la Réalité Mixte

Seule l'adresse IP associée au hardware doit être fournie à Visual Studio pour builder le code sur l'appareil. Unity a implémenté une fonctionnalité de simulation de vue holographique fonctionnant soit sur l'écran d'ordinateur, soit sur l'appareil en fournissant l'adresse IP de l'appareil. C'est un gain de temps considérable lors d'une phase de tests.

L'outil central pour le développement MR est un projet libre de droits appelé MRTK (pour Mixed Reality ToolKit) conçu par Microsoft. Il met à disposition un ensemble de méthodes, divers prefabs (avec des Components utiles, des shaders,...) et fonctionnalités (FIGURE 1.10) permettant le développement rapide d'applications Unity sur HoloLens ou tout autre casque WMR.

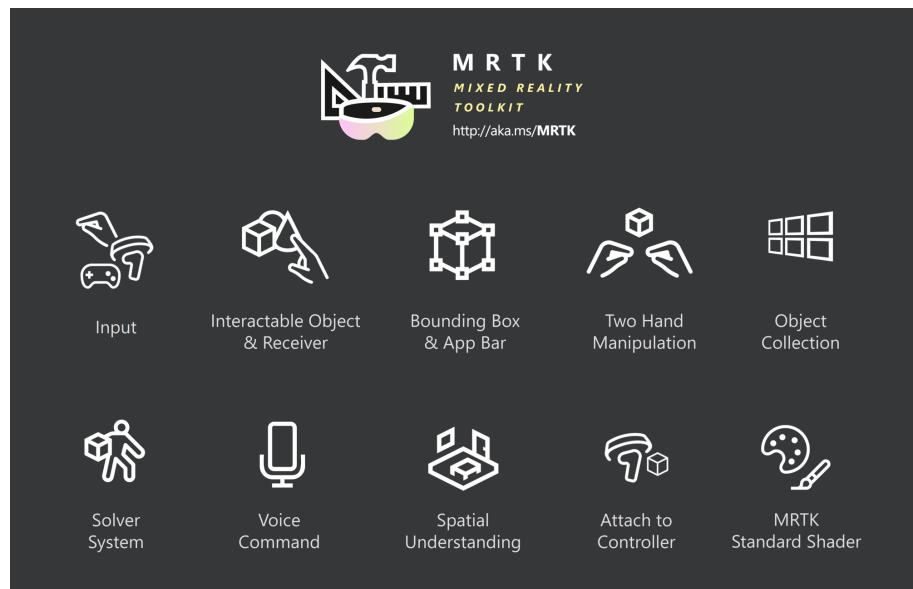


FIGURE 1.10 – Panorama des fonctionnalités de MRTK

Windows a lancé HoloToolkit Academy [4], une série de projets utilisant Unity, sous la forme d'exercices "*code à trous*". Ces tutoriels offrent un premier pas de qualité pour se familiariser avec l'implémentation de fonctionnalités MR de base pour HoloLens v1.

Abordons dès à présent les outils de visualisation sur lequel le projet se base.

1.3 Etat de l'Art des outils de visualisation et d'exploration de données

1.3.1 ImAxes : outil interactif d'exploration de données multivariées

1.3.1.1 Présentation de l'outil

ImAxes [5] est un outil de visualisation et d'exploration de données multivariées en environnement RV. La motivation de ce projet a été de se mettre à la place d'un utilisateur lambda souhaitant analyser un objet par un ensemble de dimensions qui le caractérise. A partir d'une base de données, l'application construit des *histogrammes* associés à chaque grandeur renseignée, comme on peut le voir en FIGURE 1.11.

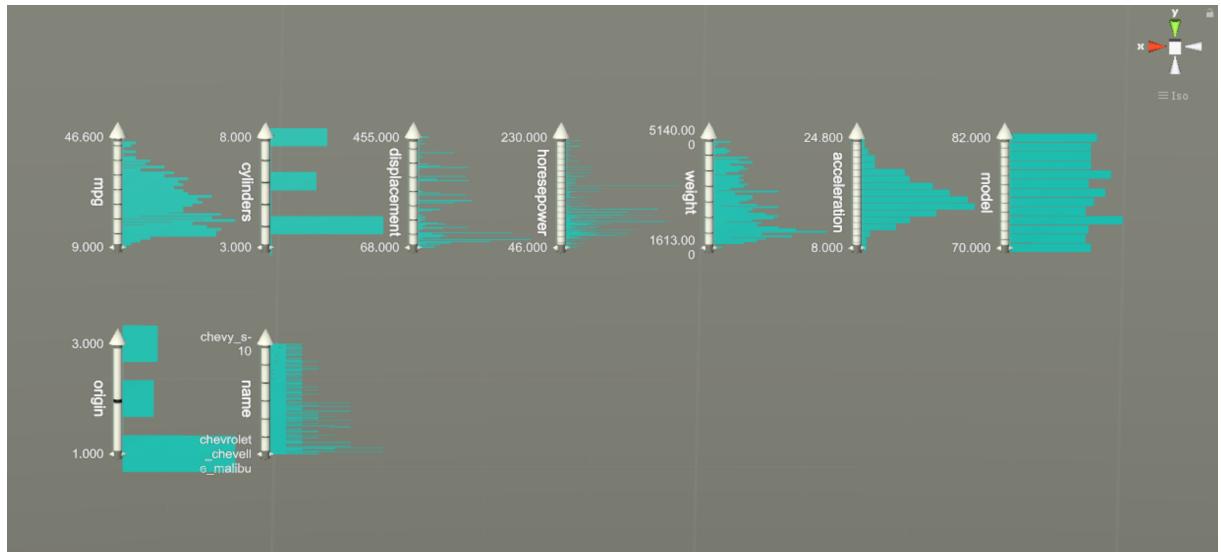


FIGURE 1.11 – Ensemble de graphiques visualisés dès l'ouverture de l'application [5]

ImAxes est outil qui se veut interactif : en appuyant sur la gâchette d'un contrôleur, il est possible d'attraper un histogramme, qui se dédouble alors pour l'analyse. Il est possible de manipuler facilement le graphique en le tenant en son centre, de le jeter,... Si l'on porte à cet histogramme un second que l'on place perpendiculairement à ce dernier, un *nuage de points 2D* est formé (FIGURE 1.12(a)). Si on dispose un troisième axe orthogonal au plan formé par les 2 axes précédents, on peut visualiser un *nuage de points 3D* (FIGURE 1.12(b)).

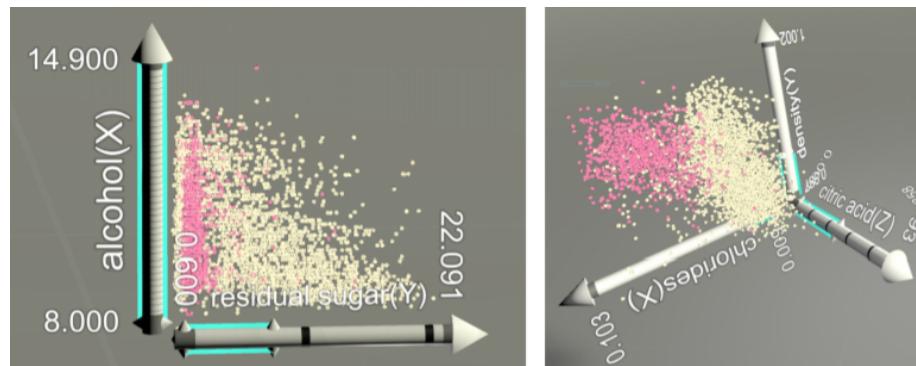


FIGURE 1.12 – Construction de nuages de points [5]

La fusion des axes au point d'accroche et les calculs effectués permettant la construction de nouvelles visualisations amènent à des calculs orthogonaux. Lorsqu'un axe touche un autre, un "squelette physique" doit être codé de sorte que la nouvelle visualisation de données puisse être construite en conséquence. Cela a été traduit en une liste de règles appelée (qualifiée de grammaire) décrivant chaque configuration perpendiculaire ou parallèle d'axes.

L'autre type de visualisation proposée est la *représentation par coordonnées parallèles*. En portant un axe à proximité d'un autre de manière parallèle, des connections par traits droits fins sont créées pour chaque correspondance dans le plan formé par les deux axes (FIGURE 1.13).

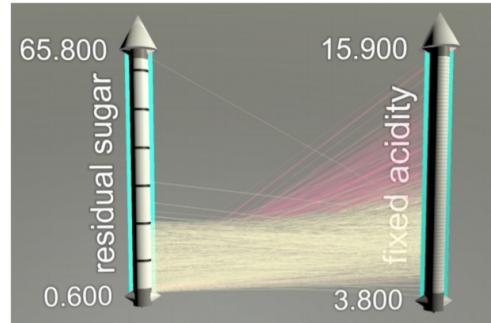


FIGURE 1.13 – Représentation par coordonnées parallèles

Cette visualisation moins connue a permis, combinée de manière circulaire (FIGURE 1.14(a)) ou avec des nuages de points 3D (FIGURE 1.14(b)), de faire émerger de nouvelles visualisations non anticipées par les chercheurs.

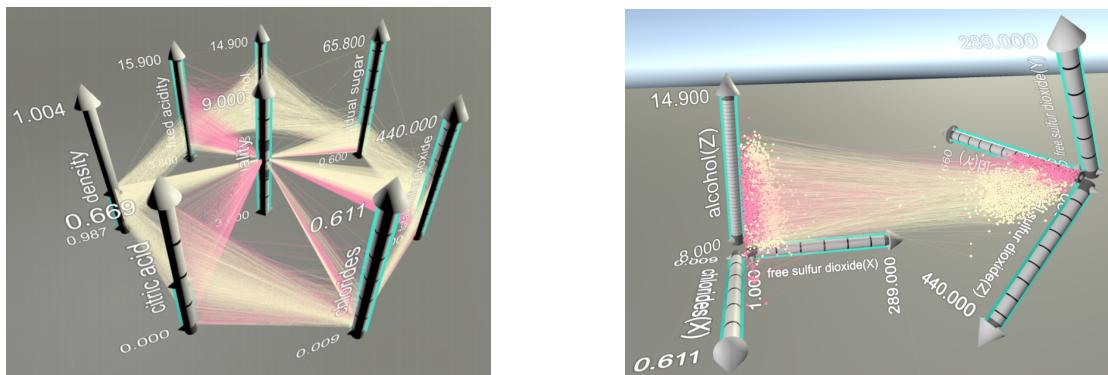


FIGURE 1.14 – Représentations émergentes [5]

Enfin, approcher un contrôleur à proximité d'un histogramme fait apparaître deux types de curseurs : le premier en forme de losange permet d'ajuster l'échelle de chaque axe tandis que la seconde agit comme ligne coupante (FIGURE 1.15).

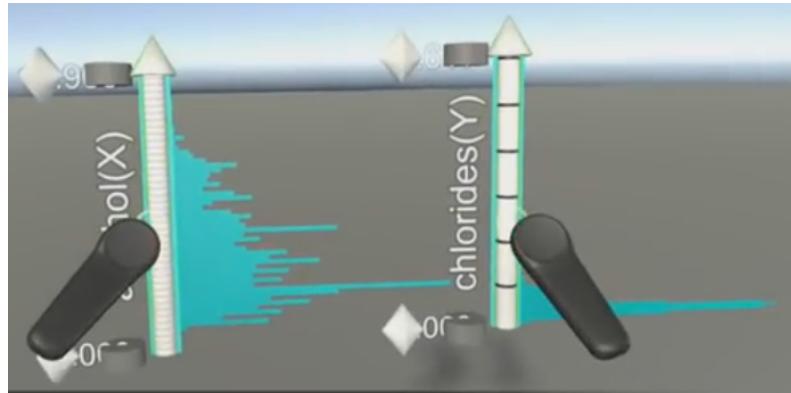


FIGURE 1.15 – Les deux types de curseurs existants

1.3.1.2 Contributions

Afin de prendre en main l'outil, quelques méthodes C# basiques ont été implémentées afin de calculer, à partir d'une série de données extraite de la base de données, la variance et l'écart-type. La série de donnée correspond en réalité à l'abscisse de l'histogramme d'une dimension donnée, dont la hauteur des barres représente le nombre d'occurrence de la dimension dans l'intervalle stipulé.

Ensuite, ces valeurs ont été affichées via un panneau Unity basique, apparaissant lorsqu'un bouton latéral du *contrôleur* (pour la VR) a été appuyé ou une gestuelle *air-tap* (pour la MR) a été effectuée sur l'histogramme en question. La durée d'affichage du panneau a été réglé à 3 secondes pour chaque panel qui apparaît, par utilisation de la méthode Unity *WaitForSeconds* permettant de déclarer et démarrer un minuteur. L'une des difficultés rencontrée a été d'implémenter la gestion de la position des différents panneaux si plusieurs avaient été demandés, sans que les différents panels ne se chevauchent. la solution trouvée a utilisé les *colliders* des panels.

Initialement, l'objectif était d'adapter cet outil pour de la RM. Or, quelques semaines ont suffit pour constater la difficulté d'une telle conversion, car l'ensemble des fonctionnalités disponibles par l'utilisation de contrôleurs devait être traduites en fonctionnalités à déclencher lorsque certains gestes étaient effectués et certaines paroles prononcées. Un import rapide nous a permis d'obtenir une image statique, i.e. une image sur laquelle aucune interaction n'était possible. D'autre part, l'utilisation des casques HoloLens1 étant limitée à une gestuelle, sa puissance de rendu et de calcul étant limitée et la technologie HoloLens2 n'étant pas disponible à cette date, nous avons préféré ne pas s'attarder davantage sur cet objectif.

Regardons dès maintenant comment lier ce projet à une carte de l'Australie.

1.3.2 MapsSDK-Unity : outil de visualisation de données sur une carte du monde

MapsUnitySDK [6] est un outil conçu par Microsoft permettant de visualiser des données sur des cartes. Selon la démo que l'on fait tourner (FIGURE 1.16), les fonctionnalités sont différentes.

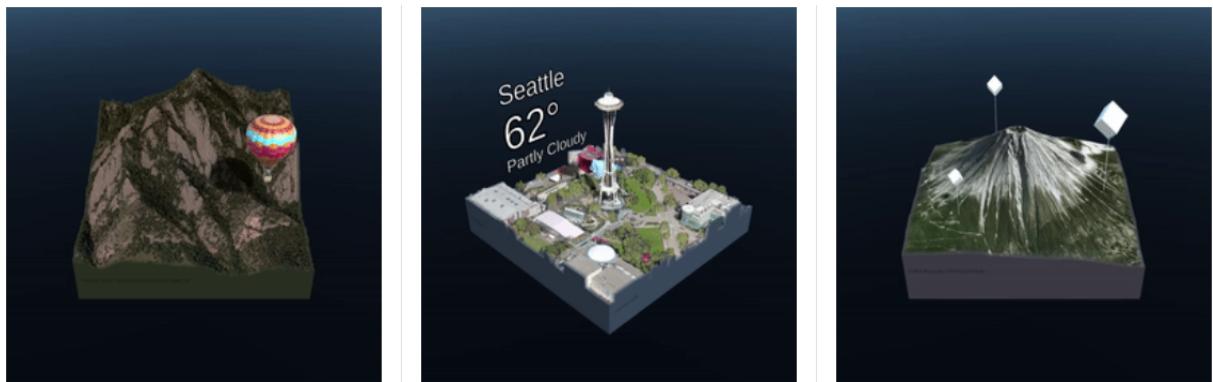


FIGURE 1.16 – Les différentes démos de l'outil [6]

La partie de l'outil qui nous intéresse utilise des Bing maps 2D, dont la localisation précise en longitude / latitude est à fournir. La localisation souhaitée est rendue avec un degré de détail relativement élevé, mais le rendu n'est pas immédiat (le streaming peut mettre quelques secondes avant la mise à jour). A l'aide des contrôleurs, il est possible d'effectuer une rotation de la carte 2D, mais également d'effectuer un zoom sur une zone particulière. Une illustration est fournie FIGURE 1.17.

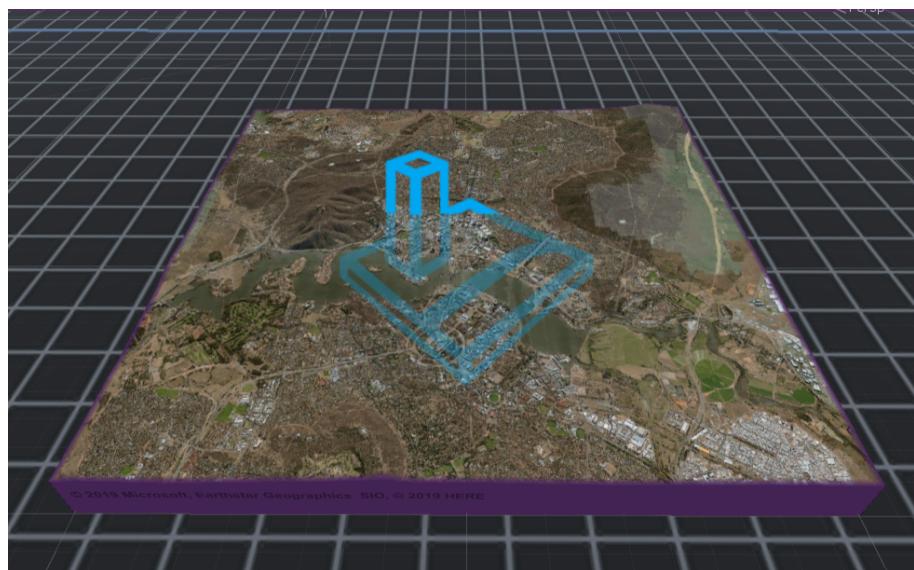


FIGURE 1.17 – Scène obtenue après indication de la Bing key, longitude et latitude de Canberra

Les punaises sont placées selon les localisations lues dans la base de données. Elles sont affichées dans une quantité allant de pair avec le degré de zoom utilisé (on aura des informations nationales si on visualise le pays entier, et régionales si on visualise seulement un état). Cependant, aucun détail n'est fourni par quelque punaise que ce soit, et peuvent également être à l'origine d'une mise à l'échelle mondiale de la carte. En effet, si on a une grande base de données, il sera fortement probable qu'au moins l'une d'entre elle soit localisée dans un autre pays, causant une mise à l'échelle erronée.

Afin de se familiariser avec le projet et pour palier au manque de cette démo, une base de données (variations de températures) a été fournie à l'application. Il a donc été affiché un ensemble de panels analogues à ceux implémentés pour ImAxes, faisant apparaître le nom du site ainsi que la valeur caractéristique de la dimension choisie.

Dans l'optique d'étudier les données énergétiques, quelques bases de données disponibles sur le net a été fournie aux projets précédents.

1.4 Les jeux de données à disposition

Bien que les éléments présentés ci-dessous soient des outils de visualisation, ce sont bien les données qu'ils contiennent qui importent le plus ici.

1.4.1 AREMI, visualisation de données en Australie

Comme indiqué à l'entrée du site, AREMI (pour Australian Renewable Energy Mapping Infrastructure en anglais) [7] est une plateforme répertoriant une quantité considérable de données spatiales relatives aux énergies renouvelables en Australie. AREMI est financé par l'Agence Nationale des Energies Renouvelables. Quant à sa conception, elle a été menée en partie par Data61, se basant sur le projet NationalMap qu'avait initié le gouvernement australien auparavant. Un exemple FIGURE 1.18 vous donne une idée de l'interface.

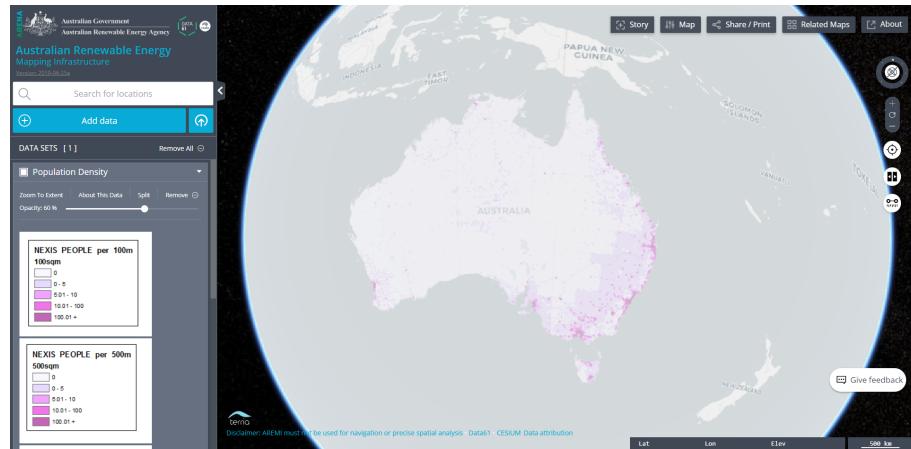


FIGURE 1.18 – Densité de population en Australie (étude fournie par AREMI) [8]

Il suffit d'indiquer une localisation en Australie, puis d'ajouter autant de base de données du catalogue que l'on désire afin de visualiser (par points, gradients de couleurs,...) la dimension sur la carte en vue spatiale. La banque de données est riche, et triée par thèmes (infrastructures énergétiques, topographie, population,...). Pour chaque base de données est indiquée une brève description thématique et technique, l'organisme qui héberge ces données auxquelles la plateforme accède, et éventuellement les adresses url ou les fichiers Excel le cas échéant.

Le programme NEAR sera complémentaire dans le sens où il permettra d'offrir des bases de données différentes et des études sur celles-ci.

1.4.2 Le programme NEAR

Le programme NEAR (pour Australia's National Energy Analytics Research Program) ?? est une plateforme prototype qui a pour ambition de rassembler des données énergétiques, recherches et rapports de tous les secteurs et de les rendre disponibles dans toute l'Australie. Son principe a déjà été exposé en Introduction FIGURE 5. C'est avant tout un outil de visualisation interactive de données, permettant d'alléger considérablement la masse de données que l'on peut trouver usuellement dans des fichiers Excel pour le *Big Data*. Lorsqu'on entre sur la plateforme, une liste d'études est consultable, basée sur les jeux de données, et fournit différentes études sous la forme de graphiques principalement, comme en FIGURE 1.19.

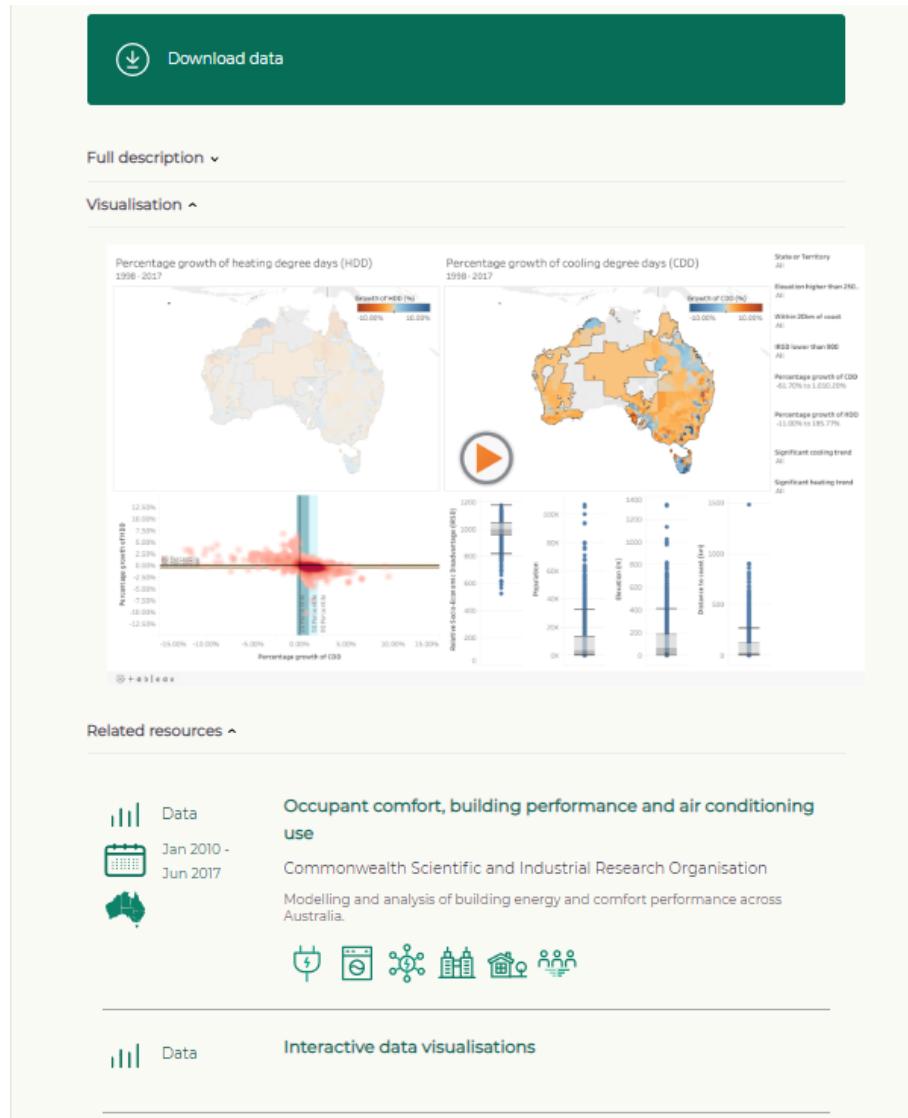


FIGURE 1.19 – Besoin de chauffage et de climatisation en Australie (étude fournie par NEAR) [9]

Les données relatives à la consommation énergétique sont prédominantes, et à cet égard, le NEAR a développé des outils de prédiction afin d'estimer la consommation électrique selon le bâtiment. Pour ce faire, des études concernant l'efficacité des panneaux solaires, ou concernant les taux de demande selon les régions ont été conduites afin de compléter la plateforme. L'objectif de ces études était de déduire le profil type d'un consommateur.

Regardons maintenant comment une solution collaborative a pu être mise en place dans Unity.

Chapitre 2

Construction d'un environnement collaboratif en Réalité Virtuelle

2.1 Développement d'une application réseau sous Unity

2.1.1 Tour d'horizon des différentes possibilités réseau

Le critère de sélection pour choisir la solution multi-joueurs qui convient le mieux pour notre application est sa complexité. En général, les méthodes basiques telles que la connexion, la déconnexion, l'envoi ou la réception de messages utilisent seulement l'API bas niveau (LLAPI). Mais dans notre cas, on souhaite assigner des messages précis à des joueurs précis ou synchroniser des états, ce qui fait croître la complexité, rendant indispensable une API haut niveau (HLAPI). Toutes les méthodes HLAPI utilisent celles de LLAPI pour fonctionner. En vue de ce qu'on a besoin, les solutions HLAPI les plus connues sont :

- UNet, basée sur Unity LLAPI et le Cloud Unity. Jugée insuffisante en termes de performance par la communauté des développeurs, une toute nouvelle couche verra le jour prochainement, ce qui a déprécié UNet ;
- PUN (pour *Photon Unity Networking* en anglais) [10], basée sur l'LLAPI Realitme et utilisant le Cloud Photon. Photon est un moteur de jeu réseau indépendant et multi-plateformes. Une version 2 est sortie récemment, ce qui a déprécié la 1 ;
- développer sa propre architecture réseau indépendamment de Unity, mais le temps considérable qu'il aurait fallu y consacrer a été jugé déraisonnable pour un stage de 6 mois.

PUN a été choisi à des fins de performance. En effet, bien que les deux solutions utilisent une architecture client / serveur, PUN supporte d'autre part l'envoi de messages

peer-to-peer. Ainsi deux joueurs peuvent échanger des données en passant uniquement par un serveur relai au lieu de passer forcément par l'hébergeur comme l'impose UNet, l'illustre la FIGURE 2.1.

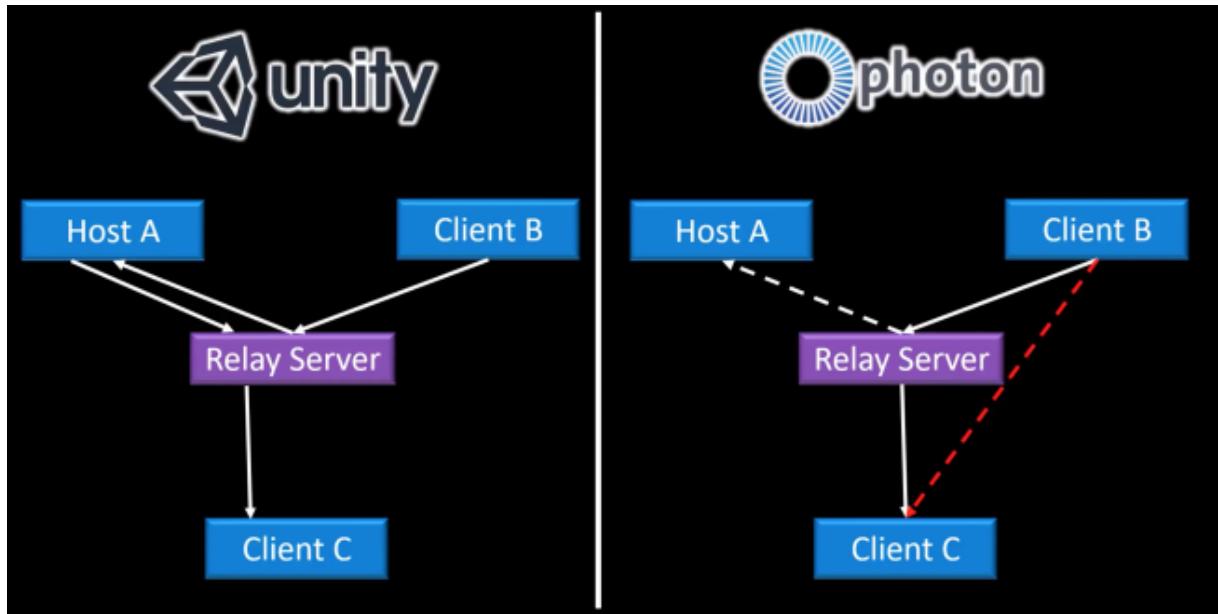


FIGURE 2.1 – Comparaison de l'architecture réseau entre UNet et PUN

Photon v1 a été choisie pour sa facilité de prise en main et les tutoriels qui affluent sur le web. Concernant la v2, la mécanique de Photon a été profondément modifiée et la documentation est plus pauvre pour l'instant.

Tout au long du projet, il a été nécessaire de regarder plus en détail les fondamentaux de PUN.

2.1.2 Concepts fondamentaux de PUN, API de développement réseau pour Unity

L'élément fondamental que tout GameObject doit porter afin d'être visualisé de tous les joueurs est le Component *Photon View*. Il permet, lorsqu'accroché à un GameObject, de lui associer un identifiant unique sur tout le réseau. Cet identifiant est associé au GameObject dès le lancement de l'application faisant appel au serveur Photon.

Différents paramètres sont disponibles dans ce Component, les plus importants étant :

- Owner, réglant le degré de transfert d'appartenance sur l'objet portant le Component. Parmi les options possibles, il y a Fixed (aucun transfert d'appartenance),

- Takeover (transfert via utilisation de méthodes Photon) et Request (transfert via surcharge de méthodes Photon) ;
- Observe, offrant différents modes de transmission de données en passant par les pipelines. Seule l'option OnReliableOnChange sera utilisée, permettant l'envoi de données uniquement lorsqu'il y a une modification entre deux mises à jour.

Détaillons les étapes clés en amont du jeu en multi-joueurs sous Photon (voir FIGURE 2.2). La première est la connexion à un *salon* (*lobby* en anglais) répertorié sur le Cloud. Il permet de consulter l'ensemble des sessions disponibles que l'on peut rejoindre. Photon a des serveurs dans plusieurs régions (l'Australie en faisant partie) distribuée sur de multiples centres d'hébergement de sorte que la latence soit minimisée. Une fois cette connexion réalisée, la création d'une *session* (*room* en anglais) est nécessaire. Son rôle est d'héberger la partie multi-joueurs se déroulant dans une scène Unity où les joueurs peuvent interagir avec les GameObjects. Ces derniers sont qualifiés de *Scene Objects* car le joueur les contrôlant est la scène initialement.

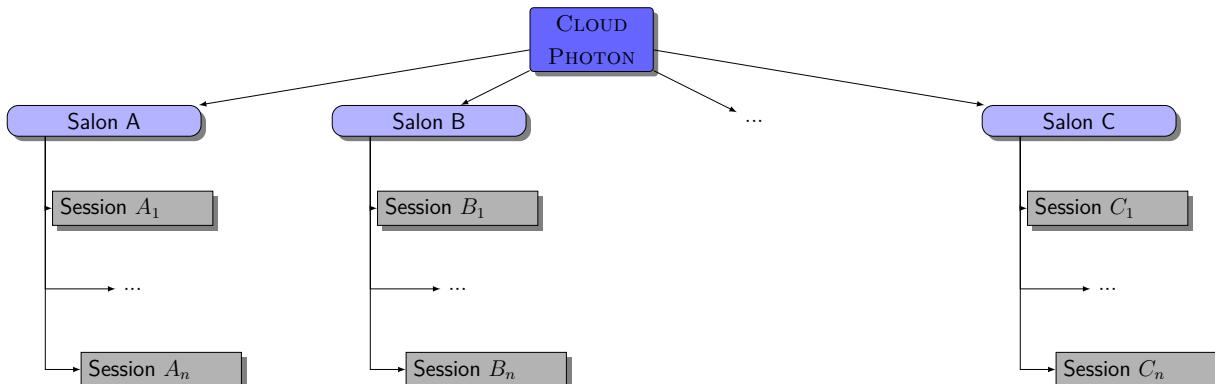


FIGURE 2.2 – Salons et sessions en Photon PUN

Le premier à être présent physiquement dans la scène est a fortiori l'utilisateur qui créé la session. Il est appelé *Master Client* car le seul moyen que les clients se connectant à la session après lui peuvent envoyer des informations au Cloud passera inévitablement par lui, comme illustré en FIGURE 2.3. Il y en a naturellement un par session, et lui est conféré la logique réseau du jeu. L'information envoyée sera des coordonnées de mouvement par exemple, et doit être envoyée au serveur de telle sorte que Photon puisse envoyer des mises à jour aux clients. Selon la fréquence de mise à jour souhaitée, des messages peuvent être envoyés au Cloud par l'utilisation de :

- la méthode `OnPhotonSerializeView` pour des informations dont la mise à jour doit être permanente (simulation de mouvements continu,...) ;
- une fonction callback appelée `RPC` pour des informations ponctuelles ou constantes pendant toute la durée du jeu (nom de joueur, couleur fixe,...). Une `RPC` doit

obligatoirement s'appliquer sur un `GameObject` particulier, qu'elle vise grâce au Component *Photon View*;

- la méthode `SetCustomProperties` permettant l'ajout d'une propriété au cube via une `HashTable`.

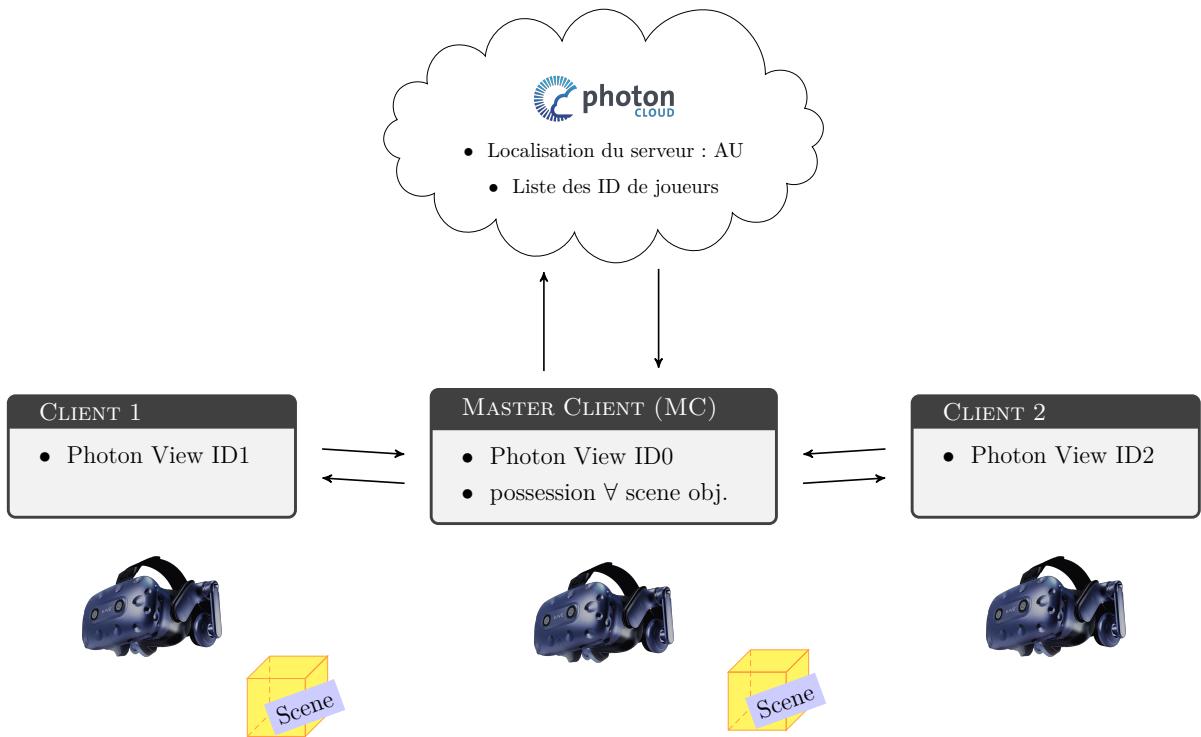


FIGURE 2.3 – Droit de possession et mises à jour sur le Cloud en Photon PUN

D'autres éléments seront abordés dans la section ultérieure, au besoin. Abordons maintenant l'architecture globale Photon qui a été utilisée.

2.2 Architecture utilisée et fonctionnalités implémentées

Rappelons que l'objectif est d'apporter une solution collaborative à ImAxes mais que vouloir convertir directement ce projet s'avère très compliqué, car il n'a pas été conçu pour cela à la base. Un tout nouveau projet a donc été ouvert

2.2.1 Enchaînement des scènes

De manière analogue à un jeu vidéo, l'application peut se décomposer en deux parties, comme montré en FIGURE 2.4. La première étape est la mise en place du réseau Photon,

avec une scène dédiée à son initialisation. Chaque joueur doit renseigner un nom de joueur puis joindre ou créer une session de jeu. Cette scène est locale à chaque ordinateur qui va faire tourner l'application. De plus amples précisions se trouvent en section 2.2.

La scène suivante est celle de jeu, qui contiendra un ensemble de GameObjects constituant le décor. Il est ensuite très courant de charger une scène Unity à part via la méthode `PhotonNetwork.LoadLevel`, permettant à tous les utilisateurs de charger la même scène dans laquelle ils vont interagir. Cette scène n'existera que si au moins un joueur crée une session. Dans une architecture PUN, quand l'hébergeur quitte la session, son pouvoir est automatiquement déferré à un autre client ; sa durée de vie sera donc celle du dernier client quittant la session. Cette scène est détaillée dans les sections 2.3. et 2.4.

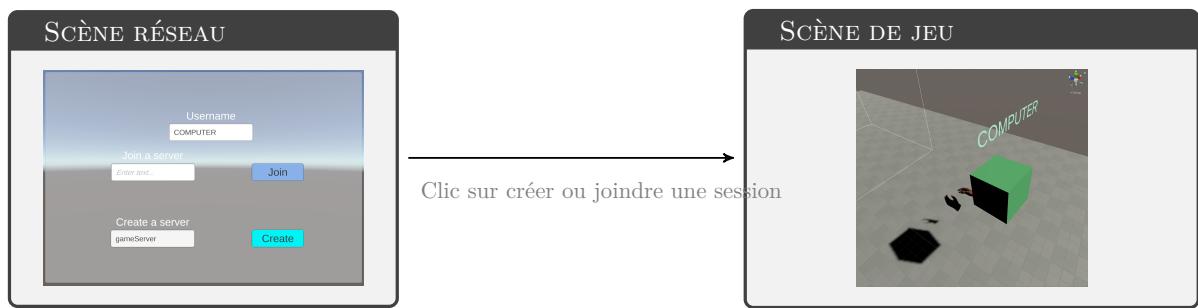


FIGURE 2.4 – Diagramme Bloc de l'enchaînement des scènes

2.2.2 Interface graphique de communication réseau

La première étape, afin de converger vers une application collaborative, est La connexion à une session. Elle consiste à informer l'utilisateur de l'évolution de la connexion au serveur Photon par une série de panels (FIGURE 2.5). Pour faire enchaîner l'ensemble de ces différentes vues, il est nécessaire d'activer ou désactiver les panels correspondants, par la commande `SetActive` en renseignant true ou false comme unique argument.

Apparait alors une interface (FIGURE 2.6(a)) une fois la connexion accomplie, permettant au client d'indiquer le nom d'utilisateur qu'il souhaite, puis soit de créer sa propre session (en lui donnant un nom), soit de rejoindre une déjà existante. Les noms fournis devront être supérieurs à 1. Si ce n'est pas le cas, le sous-texte "enter text" apparaîtra en rouge (FIGURE 2.6(b)) afin d'informer le joueur de sa maladresse. Enfin, tous les textes sont affichés via des Components *TextMeshPro*, de plus grande qualité que les Components *TextMesh* dans Unity par défaut.

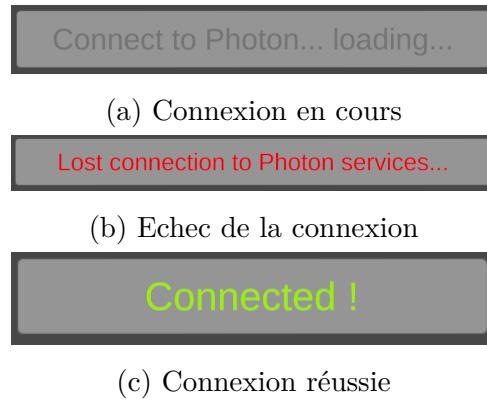


FIGURE 2.5 – Informations de connexions possibles

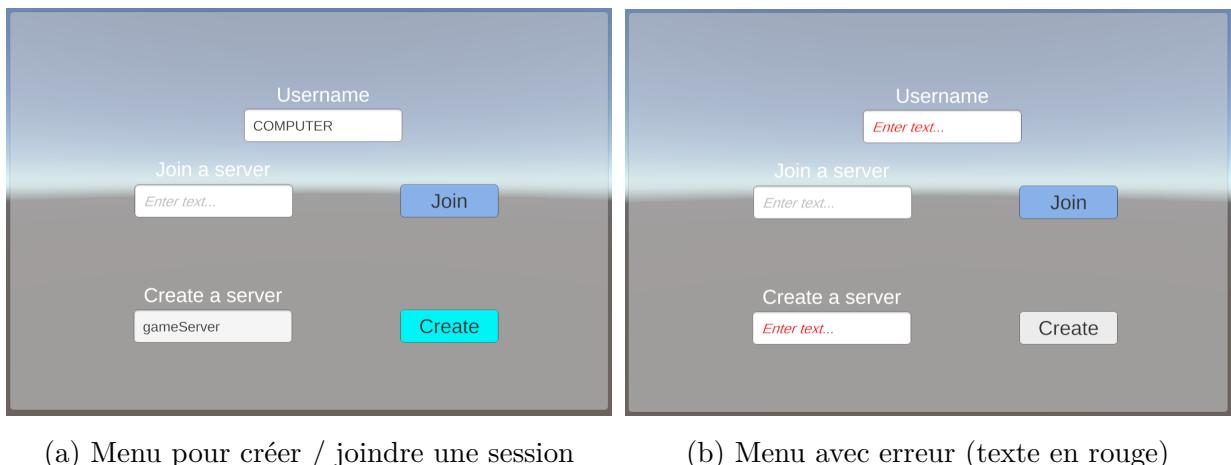


FIGURE 2.6 – Menu initial et erreurs indiquées

L'un des bémol est le rendu quelque peu flou lorsque l'application est lancée, qui n'a pas été encore tout à fait résolu. Un autre objectif sera de créer un menu déroulant afin de choisir la session que nous souhaitons rejoindre parmi celles existantes.

Une fois la session créée ou rejointe, le joueur entre en scène dans le monde 3D sous la forme d'une modélisation 3D, qui se doit d'être explicitée tout autant que les mises à jour de mouvements.

2.2.3 Modélisation et comportement des différents utilisateurs

L'objectif est de visualiser physiquement l'ensemble des joueurs se trouvant dans la scène lorsqu'une session a été créée. Ainsi, lorsqu'un joueur se connecte à une session, une instanciation des principales parties du corps (qui sont des prefabs), représentées sommairement par un cube à la place d'un casque et les modèles 3D de mains pour les

contrôleurs, est effectuée (FIGURE 2.7). Cette instantiation passe nécessairement par la méthode PUN `PhotonNetwork.Instantiate` afin que le Cloud puisse les instancier sur le réseau et transmettre cette information aux autres joueurs. Ces prefabs doivent être placés dans le dossier *Resources*.

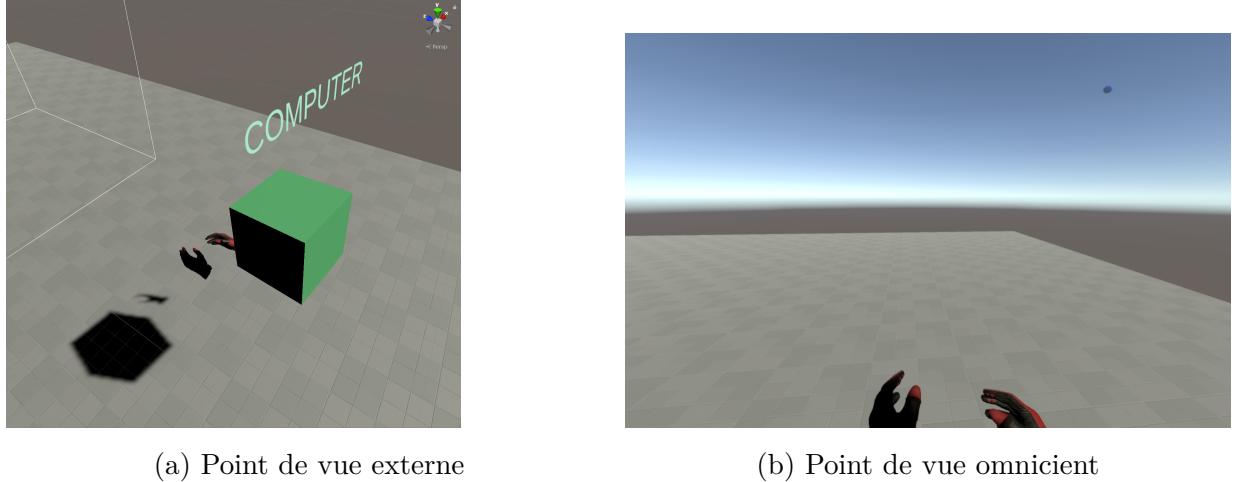


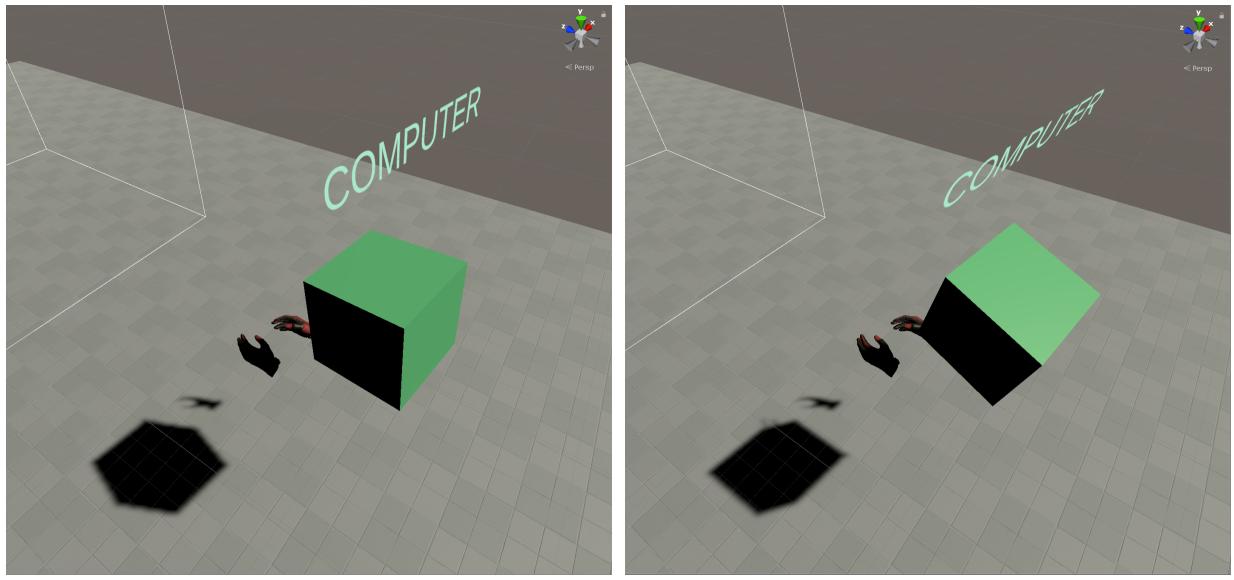
FIGURE 2.7 – Différents points de vue pour le joueur

Le mouvement d'un joueur (se traduisant par l'évolution d'une position et d'une rotation dans le temps) se transmet facilement aux autres en attachant le Component PUN *Photon Transform View* au GameObject correspondant, permettant l'envoi continu de la position / rotation / mise à échelle au choix au serveur (il est tout à fait possible de le coder nous-même via la méthode `OnPhotonSerializeView`).

Afin de distinguer les joueurs, une couleur aléatoire leur a été associée. Cette couleur étant identique pendant toute la durée de la session, un simple RPC a été suffisant pour transmettre l'information. Dans ce même RPC a été communiqué le nom de l'utilisateur fourni dans l'interface graphique. La commande `isMine` est fondamentale pour coder ce type de fonctionnalités. En effet, ce mot-clé permet de distinguer si le code est initié par l'ordinateur du joueur ou celui d'un autre. Ainsi, pour initialiser la couleur des autres joueurs dans notre monde, il faut s'assurer que le cube que l'on colore ne soit pas le nôtre car le code tournant sur les deux ordinateurs relié aux deux casques est identique. De même pour le nom de joueur qui, pour pouvoir être transféré, nécessite un GameObject dans chaque scène.

Enfin, l'orientation du nom de joueur dans le jeu a été codée de telle sorte à ce que l'avatar puisse lire le nom de tous ses collègues (FIGURE 2.8(a)), ainsi que son propre nom en levant la tête vers le ciel (FIGURE 2.8(b)). L'orientation se calcule en introduisant une différence de Quaternions (objet mathématique représentant une rotation en Unity) entre la position du nom de joueur et celle de la camera dans la méthode Unity `LookRotation`.

L'introduire dans la méthode `Update` permet d'avoir l'information pour chaque frame. Ces deux `GameObjects` sont trouvés dans le code grâce à la commande `FindObjectWithTag`, bien moins coûteuse que la commande `Find`, son utilisation étant un mauvais réflexe à prendre.



(a) Le joueur regarde devant lui

(b) Le joueur relève la tête

FIGURE 2.8 – Orientation du nom du joueur selon le regard du joueur

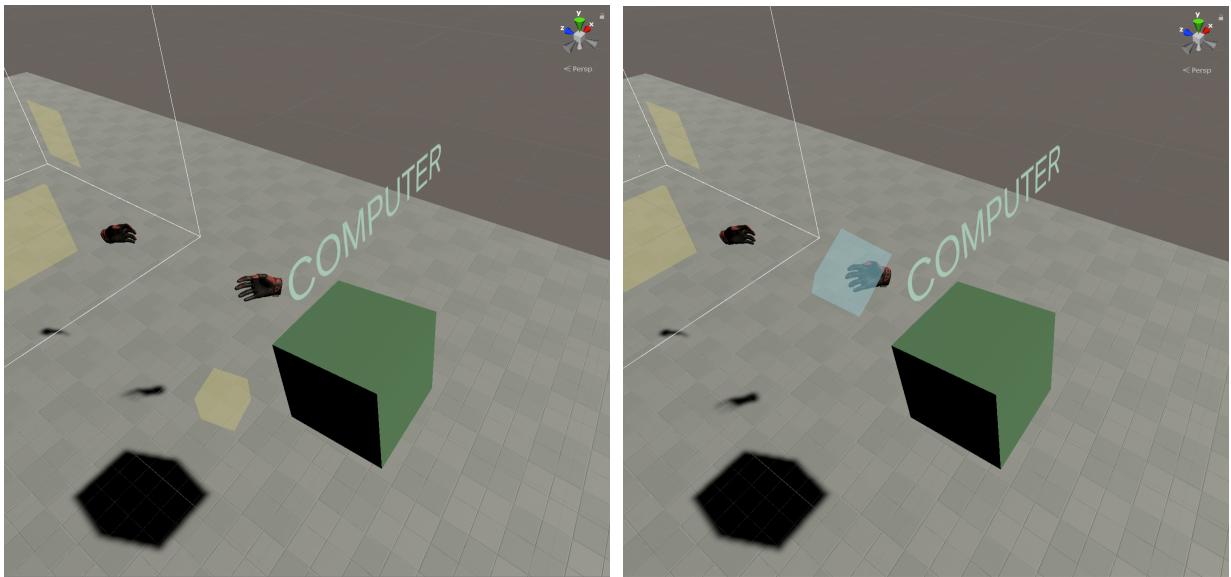
Tout l'intérêt de coder une solution collaborative est de voir comment chaque joueur peut interagir avec des `GameObjects`.

2.2.4 Interaction avec les `GameObjects` de la scène

Afin d'éviter toute cacophonie, le *Master Client* sera le possesseur de tous les `GameObjects` dès son entrée dans la scène qu'il aura créée. Si un client arrivant par la suite souhaite interagir avec l'un d'entre eux, il devra effectuer une demande au possesseur via la méthode `RequestOwnership`, automatiquement acceptée par transfert de son droit de possession via la méthode `TransferOwnership`. L'option *Takeover* a été choisie pour chaque Component *Photon View*, ce qu'un joueur ne peut refuser de perdre son droit de possession sur un `GameObject` si un autre le demande.

Interagir avec un `GameObject` (ici un cube) signifie pouvoir le prendre avec la main et le manipuler comme bon nous semble à l'aide des *contrôleurs*. Le cube est coloré différemment dès lors que les *colliders* du cube et des *contrôleurs* entrent en contact (FIGURE 2.9). Le *collider* de chaque *contrôleur* est une sphère située à leur extrémité respective, et seuls les `GameObjects` taggés *CubeInteractable* sont réceptifs aux *contrôleurs*.

dans le monde partagé.



(a) Non intersection des *colliders* du cube et (b) Intersection des *colliders* du cube et d'un contrôleur

FIGURE 2.9 – Changement de couleur du cube selon les *colliders*

Cependant, le point d'accroche du cube était toujours identique (son centre), et son mouvement était très légèrement saccadé à certains moments, parfois partant de notre main vers l'infini durant une à deux secondes. L'origine de ce désagrément était le fait d'initialiser dans la méthode `Update` la position du parent de l'objet porté à celle du contrôleur. L'utilisation de la méthode `MoveTo` a permis une manipulation bien plus fluide de l'objet porté et une gravité plus réaliste lorsque l'objet était lancé. De plus, les possibilités de points d'accroche sont désormais infinis et bien plus précis (FIGURE 2.10(b) et (c)), au lieu d'un unique point d'ancre auparavant (FIGURE 2.10(a)). Bien souvent, on s'aperçoit qu'une méthode Unity a déjà implémentée pour de nombreuses situations, et bien meilleure que ce qu'on aurait pu faire en de nombreuses lignes.

Afin d'améliorer la facilité de manipulation des `SceneObjects` visés pour l'analyse (plans, ceux décrits en Chapitre 3), lorsqu'un joueur attrape l'un d'eux, sa gravité est activée et sa cinématique désactivée, et le contraire est effectué lorsque le même joueur le laisse. Les plans ne vont ainsi pas automatiquement tomber sur le sol lorsqu'un utilisateur le laisse tomber, et ne va pas résulter des collisions entre plans analytiques des réactions étranges.

Le cube est également soumis aux lois de la gravité de Unity, ces dernières étant à l'origine d'une longue mésaventure.

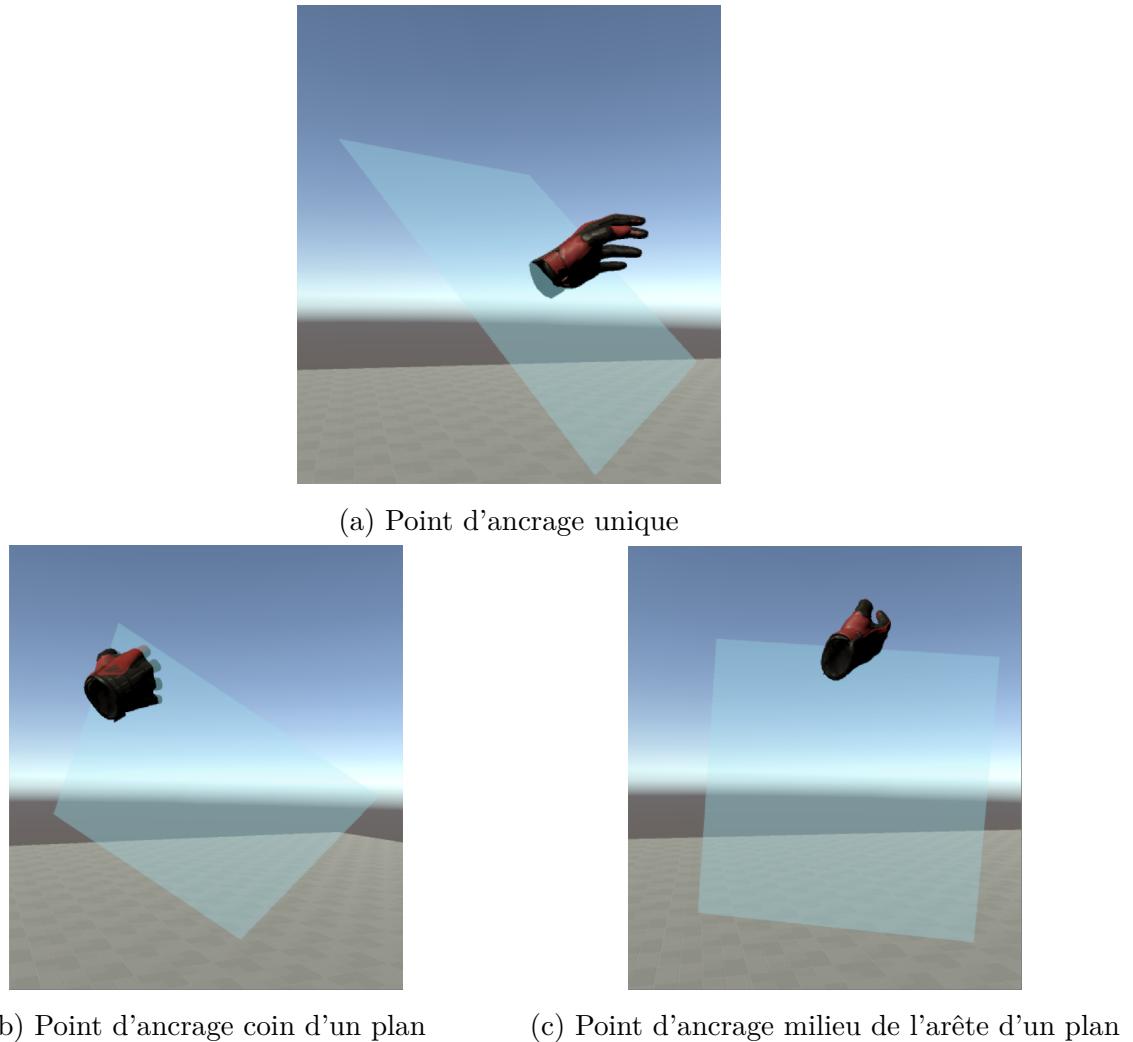


FIGURE 2.10 – Différence des points d’ancrage selon la fonction utilisée

2.3 Difficultés rencontrées inattendues

2.3.1 Simulation physique des différents GameObjects

L'une des limites majeure actuelle de Photon est de ne pas gérer la physique des divers SceneObjects sur le serveur même (ce qui aurait permis à tous les utilisateurs de profiter d'une physique stable, unique et mise à jour instantanément). C'est donc le module Physique de Unity qui est utilisé. Seulement, chaque ordinateur faisant tourner localement la même application Unity, la gestion de la physique est elle aussi locale, avec sa propre temporalité, construite sur le moteur Nvidia PhysX du GPU. Le moteur est donc qualifié de *non-déterministe*. Or, lorsqu'un utilisateur souhaite porter un cube, le mouvement de ce dernier est soumis aux lois physiques de la gravité, qu'il convient de mettre à jour aux autres utilisateurs. Et il n'est pas possible de traiter cette information comme celle du mouvement d'un joueur par exemple, car non soumis à une physique

particulière.

Un Component analogue à *On Photon Transform View* existe (*On Photon Rigidbody View*) mais trop de lacunes sont visibles si on l'utilise, produisant parfois d'étranges phénomènes physiques. Malgré une certaine fluidité, on ne parvenait pas à voir le déplacement en hauteur d'un cube porté, seuls de grands à-coups en hauteurs permettant de voir une évolution.

La première stratégie testée a été de construire un objet 3D n'ayant pas de Component *Rigidbody*, puis de lui construire un object enfant qui en avait un. Son mouvement aurait été transmis indirectement par le biais de l'objet sans Component *Rigidbody*, distribué sur le réseau. Elle s'est avérée être non concluante, aucun mouvement ayant été visible.

La seconde stratégie a été de transmettre les valeurs numériques qui définissent un Component *Rigidbody* (que sont la vitesse et la vitesse angulaire) sous forme d'une liste de Vector2D aux autres utilisateurs, puis de les mettre à jour localement pour chaque joueur. Pour ce faire, une surcharge de la méthode `OnPhotonSerializeView` (aperçu donné en FIGURE 2.11) a été réalisée. Par suite, chaque joueur a été capable de visualiser des mouvements de cube continus. La mise à jour manque cependant d'un peu de fluidité, ce qui ne permet pas de s'échanger un cube comme un ballon par exemple.

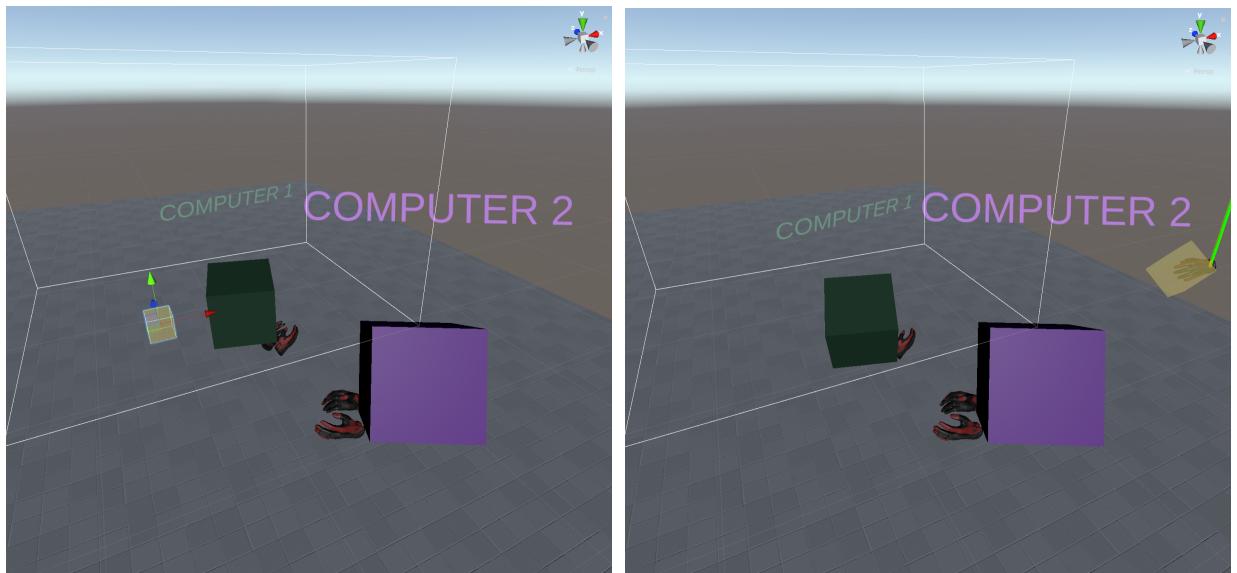


FIGURE 2.11 – Cube visualisé sur l'écran vue Scène du computer 2

Hormis cela, SteamVR a toujours été au centre de nombreux problèmes, en particulier le décalage entre le monde de chaque joueur.

2.3.2 Correspondance entre le monde 3D de chaque utilisateur

L'autre principale difficulté a été de simuler pour tous les utilisateurs un monde à géométrie unique. En effet, comme abordé en début de chapitre, chaque joueur a son propre casque HTC VIVE tournant sur un ordinateur dédié. La scène 3D chargée est donc identique, mais sa délimitation est propre à chaque joueur car construite sur une géométrie relative à l'initialisation de la zone de jeu tracée pour le casque. En résulte des décalages importants entre la visualisation rendue pour les différents joueurs, où un joueur peut voir un autre devant lui alors qu'il est en réalité derrière. En somme, pour cette application immersive, une fonctionnalité de type RA était souhaitée (visualiser la position précise d'un autre joueur dans le monde avec port du casque et réciproquement).

L'une des solutions ayant permis de résoudre en grande partie le problème a été de tracer des zones de jeu (exigé lors de l'installation de SteamVR) très similaires pour l'ensemble des casques impliqués dans la solution collaborative. SteamVR, après que l'on ait fermé la boucle (FIGURE 2.12(a)), effectue une approximation de la zone de jeu à une forme géométrique grossière (FIGURE 2.12(b)) : il suffit donc de quelques centimètres de différence pour aboutir à des zones dont la différence a été accentuée par l'approximation maladroite.

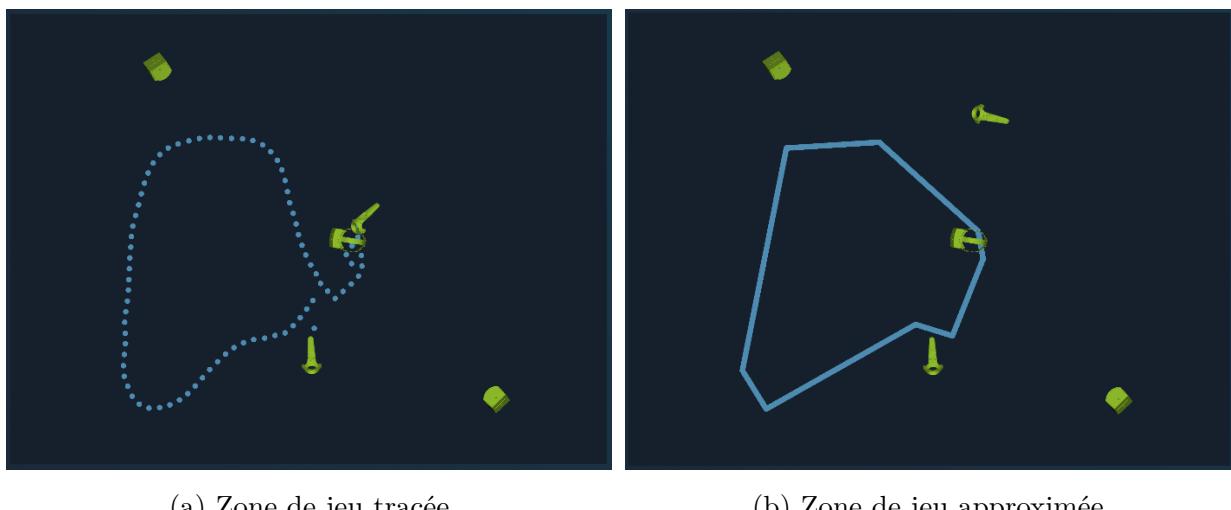


FIGURE 2.12 – Zone de jeu demandée par SteamVR

Cette solution a permis l'obtention de géométries proches pour chaque casque, mais un décalage était parfois visibles entre les *stations* comme illustré en FIGURE 2.13. Par conséquent, un joueur ne pouvait toucher un casque en posant sa main sur sa modélisation correspondante dans le monde virtuel.

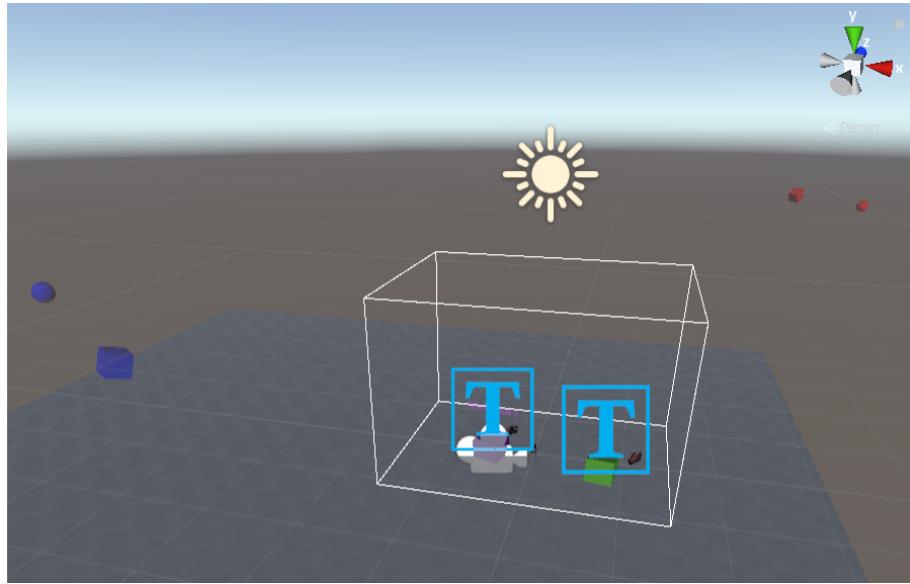


FIGURE 2.13 – Décalage persistant entre les cubes représentant les *stations*

Des recherches ont déjà été effectuées sur ce sujet [11]. L'une d'entre elle a été de chercher une solution adaptée à un jeu se déroulant dans un espace fermé confiné. Ils ont travaillé sur un algorithme qui pouvait prédire une future collision, et modifier le monde virtuel 3D des deux joueurs de manière appropriée, de telle sorte à ce qu'ils deviennent de leur trajectoire initiale. Ces recherches n'étant pas encore totalement abouties, elles ont été rapidement écartées.

Une solution plus simple a été choisie : elle a consisté à implémenter un programme de correspondance spatiale entre les différentes géométries locales sur lesquelles la représentation de la scène commune se base. Le monde du *Master Client* est celui pris par défaut (car le premier à visualiser la scène) et sa géométrie devra être épousée par chaque client. Toutefois, ces calculs ne peuvent s'effectuer qu'à partir d'un point de référence, i.e. qu'en présence d'un élément physique ayant une position fixe dans l'espace réel, quelque soit le casque utilisé. Il est possible de le créer soit même mais, dans notre cas, il est plus aisés d'extraire de SteamVR l'information de position des deux *stations* fixées au plafond selon la géométrie tracée, comme le témoigne la FIGURE 2.12. Tous les casques HTC VIVE du laboratoire utilisent les deux mêmes *stations* du laboratoire. Cette information n'est accessible que via le numéro de série des deux stations, que SteamVR initialise aléatoirement, sauf pour un qui ne varie pas (base pour la géométrie). Ainsi, ladite station ne va pas être référencée par le même numéro de série selon le casque considéré et selon la session SteamVR lancée. Notre seule chance est alors de comparer le numéro de série changeant dans chaque monde : si deux numéros de série sont les mêmes pour une *station*, cela signifie que les *stations* ont été identifiées de la même façon pour les deux mondes (cas montré en FIGURE 2.14 avec les modèles 3D pour différencier à qui une paire de

stations appartient, et les couleurs pour savoir si les paires sont inversées), sinon elles sont opposées, et le décalage de monde 3D est effectué en conséquence.

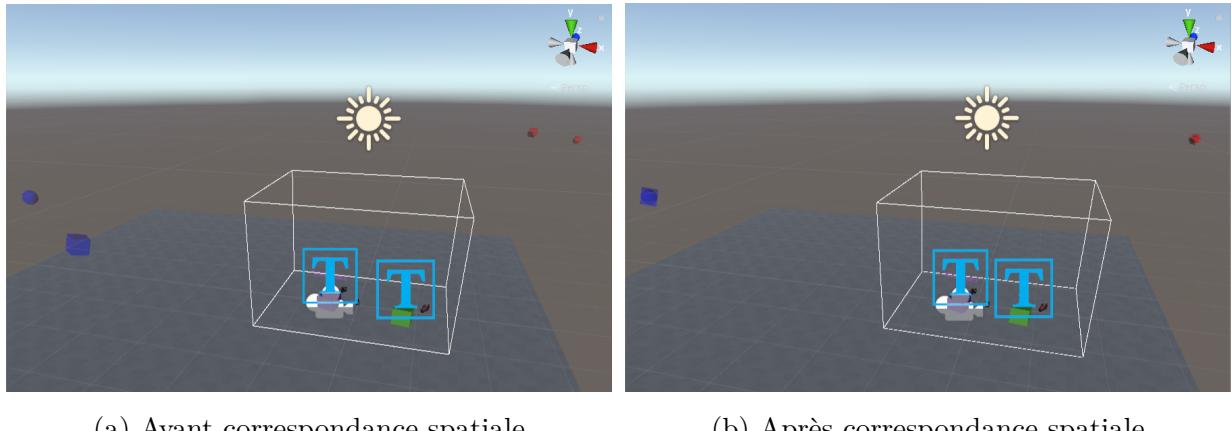


FIGURE 2.14 – Evolution de la position des *stations*

Pour finir, un dispositif a été initialisé au cours de ce projet, mais a apporté plus de problèmes qu'autre chose.

2.3.3 Les problèmes causés par le dispositif Wi-Fi

L'utilisation de fils limitaient l'aisance de l'utilisateur dans ses gestuelles lors de l'immersion. Depuis quelques mois, un dispositif (montré FIGURE 2.15) est vendu par VIVE, permettant de se passer de fils. Voici de quels parties il est composé, outre le logiciel Wi-Fi à installer sur l'ordinateur :

- un adaptateur Wi-Fi à fixer sur le casque,
- un dispositif comprenant une boîte WiGig permettant de communiquer sans fil des données à une vitesse de plusieurs Gigabits et une carte PCIe (pour *Peripheral Component Interconnect* en anglais) à insérer dans l'une des fentes de la carte mère afin que l'ordinateur puisse disposer d'un port additionnel ;
- des câbles reliant le casque et la batterie à l'adaptateur.

Pendant un mois, des tests ont été effectués sur des casques VIVE PRO et VIVE PRO EYE mais le dispositif a vite été abandonné. Malgré une bonne fluidité, les capteurs mettaient souvent plus de dix minutes avant d'être détectés et la durée de la batterie frôlait le ridicule (2 heures tout au plus), le redémarrage de l'ordinateur étant très souvent la meilleure solution à tous les problèmes.



FIGURE 2.15 – Composants d'un adaptateur sans fil Wi-Fi VIVE

La dernière section va exposer la manière dont la visualisation de donnée a été améliorée, en particulier en environnement multi-joueurs.

Chapitre 3

Rendu graphique pour la visualisation de données collaborative

3.1 Fonctionnement du *pipeline* graphique sous Unity

3.1.1 Processus de rendu graphique et principe des *shaders*

Un GPU (pour *Graphics Processing Unit* en anglais) est responsable de la disposition graphique de chaque pixel qui compose l'écran. Il passe par différentes étapes pour rendre une image, ce qui constitue le *pipeline graphique*. Par défaut, le GPU identifie chaque objet à un ensemble de minuscules polygones accolés souvent appelés *primitives* couvrant un ensemble de pixels. Une primitive est une connexion de points appelés *vertices*, et peuvent être décrits par leur position, leur normale,... On ne va considérer que les triangles car c'est la primitive la plus largement utilisée.

Afin d'avoir un contrôle plus direct sur le pipeline graphique et pouvoir intervenir sur le rendu par défaut, un programme codé pour tourner spécifiquement sur le GPU appelé un *shader* peut être utilisé. Il permet, entre autre, de modifier l'apparence géométrique du maillage, modifier les réflexions de lumière ou créer des effets spéciaux. Un shader est généralement utilisé par un *materiau* appliqué sur un objet. Et un matériau requiert une *texture* pour être définie : une image 2D souvent appelée *sprite* et fréquemment exprimée dans le repère relatif au modèle 3D, dont les coordonnées sont qualifiées de *données UV*.

Un shader peut être écrit en plusieurs langages, et supporté par des APIs fournissant un ensemble standardisé de fonctions permettant une grande précision dans la communication de données à la carte graphique ; Ces langages sont :

- GLSL (pour *OpenGL Shading Language* en anglais) supporté par l'API OpenGL,
 - HLSL (pour *High Level Shading Language* en anglais) utilisé par défaut dans la plupart des moteurs de jeu comme Unity ou encore UnrealEngine4, supporté par le sous-ensemble Direct3D de l'API DirectX (dont le pipeline est illustré FIGURE 3.1),
 - Cg (*C for Graphics* en anglais), maintenant déprécié, qui, une fois compilé, sort des programmes shaders DirectX ou OpenGL.

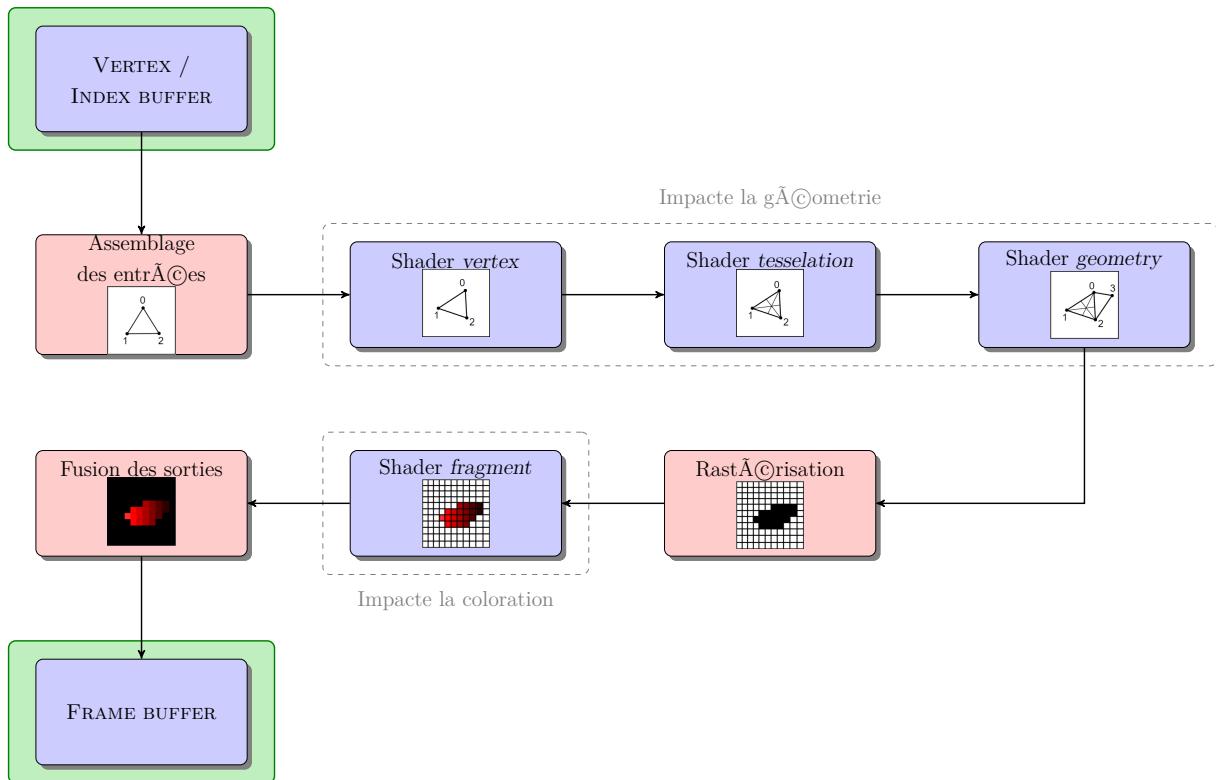


FIGURE 3.1 – Pipeline graphique de Direct3D

Il est maintenant temps d'introduire des shaders particuliers et le langage qui formalise leur connexion avec Unity.

3.1.2 Design des *shaders* en Unity et présentation des principaux

Pour être rendu, un GameObject a un Component *Mesh Renderer* contenant au minimum un matériau, pouvant utiliser un shader, comme expliqué précédemment. Les matériaux peuvent utiliser au plus un shader, mais un shader peut être utilisé par autant de matériaux que l'on souhaite. La FIGURE 3.2 illustre les spécificités de Unity3D.

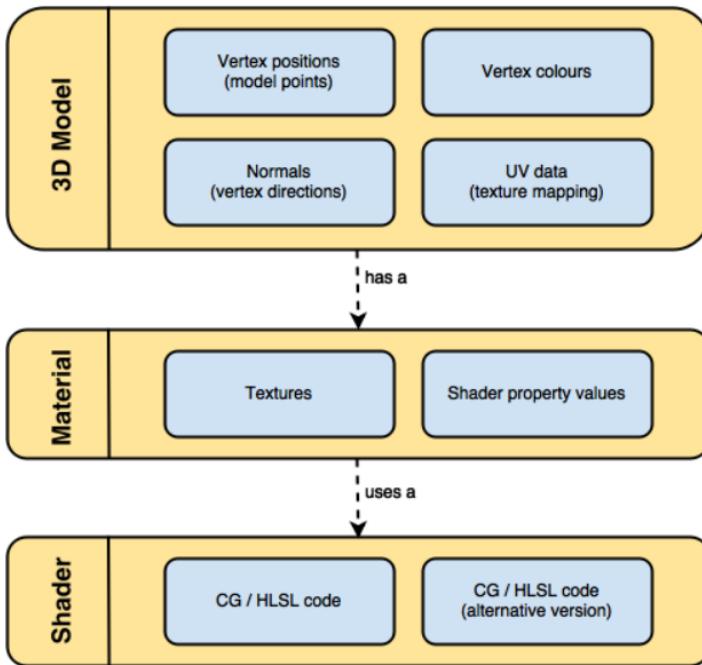


FIGURE 3.2 – Principe de fonctionnement du rendu en Unity3D [12]

Tout shader en Unity doit être écrit en **ShaderLab**, un langage déclaratif qui consiste en un emballage permettant Unity de prendre en compte les shaders plus facilement et gérer leur comportement. **ShaderLab** impose une structure très spécifique :

- une section **Properties** contenant l'ensemble des variables modifiables en temps réel pour un matériau donné, visible dans la fenêtre Inspector, au même titre que les attributs public de classes pour des scripts ;
- une section **Subshader** contenant une première ligne de tags renseignés sous forme de paire clé-valeur, puis l'enchaînement des vagues de rendu graphique successives appelées passes calculées sur le GPU.

Les passes sont très spécifiques à Unity et doivent être comprises en tant que conteneurs de shaders classiques (comme définis précédemment). Un shader Unity peut donc accueillir plusieurs shaders.

Chaque passe contient un programme shader entre balises `CGPROGRAM`, contenant les directives de pré-compilation, des *includes*, et les variables/structures/fonctions utiles pour l'implémenter. Pour implémenter un shader, une ou plusieurs fonctions doivent être stipulées dans une macro au départ, et varient selon le type de shader que l'on souhaite implémenter. Unity supporte plusieurs types de shaders, préconisés selon des objectifs précis, les plus utilisés étant :

- le *shader surface* si on souhaite se focaliser sur les effets de lumière réaliste d'une

texture,

- le *shader vertex & fragment* si le but est de s'occuper d'effets spéciaux non réaliste.

C'est ce dernier type que nous allons utiliser majoritairement, illustré en FIGURE 2.3. Comme son nom l'indique, ce type de shader se compose de deux fonctions principales :

- *vertex* (en charge de la définition des points) va indiquer au GPU à quelle position chaque point va apparaître. Elle prend en entrée une instance de structure contenant les variables (position, normale et coordonnées UV dans notre cas) caractérisant un point, et nous retourne une instance de structure contenant l'ensemble des variables (position clippée à l'écran, couleur, coordonnées globales,...) calculées à partir des entrées.
- *fragment* (en charge de la définition des pixels) va indiquer au GPU quelle est la couleur que chaque pixel doit prendre. Elle prend en entrée la sortie que lui offre la fonction *vertex*, et retourne un vecteur à quatre composantes représentant une couleur RGBA.

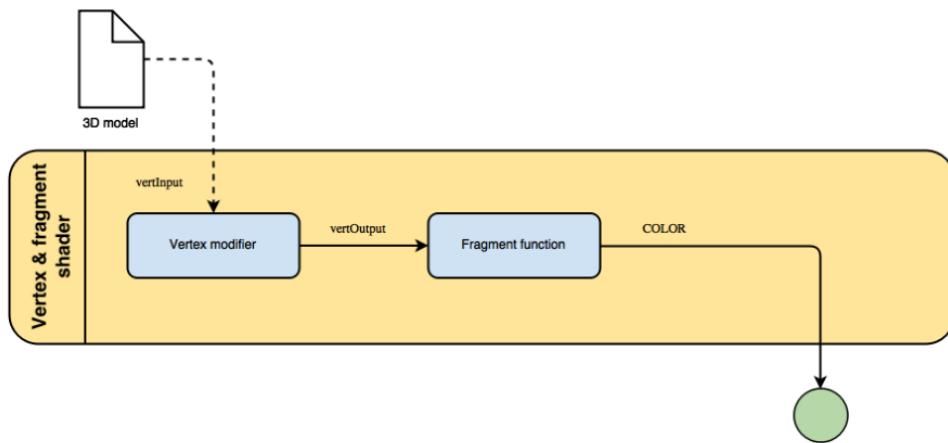


FIGURE 3.3 – Principe de fonctionnement du *shader vertex & fragment* en Unity [12]

La pipeline graphique permet aussi l'implémentation de shaders plus complexes tels que les *shaders de tessellation* permettant de subdiviser une primitive ou encore le *shader geometry* dont le principe est de générer de nouvelles primitives en ajoutant à des points, lignes ou triangles d'autres respectivement [13]. Nous allons rentrer plus en détail sur ce dernier shader dans la section qui suit.

Des moyens doivent ensuite être trouvés pour fournir les données d'énergie aux shaders.

3.2 Représentation graphique des données

3.2.1 Crédit d'un nuage de points et rendu

Avant de se lancer tête baissée dans le rendu d'une scène où plusieurs graphiques d'ImAxes apparaîtraient, il a été décidé dans un premier temps d'appliquer à la texture d'une sphère un shader construisant un nuage de points. Cette sphère est un SceneObject non portatif dans un premier temps. L'objectif, à terme, est de fournir en entrée de ce shader des coordonnées qui seraient calculées selon une base de données pour qu'il construise de lui-même la forme graphique du nuage.

Par défaut, Unity utilise des triangles pour représenter des objets 3D, alors que les hardware graphiques modernes peuvent principalement supporter triangles, lignes et points (qui nous intéresse ici) comme primitives basiques. Le problème est que l'apparence des points n'est pas configurable en HLSL (en particulier leur taille, proche de la taille d'un pixel). C'est pourquoi un *geometry shader* est souvent implémenté afin de convertir chaque point de l'objet concerné en une concaténation de primitives triangulaires couvrant bien plus de pixels, alors visibles par le joueur. Son implémentation consiste à calculer dans une *boucle pour* le futur triangle adjacent au précédent grâce au cercle trigonométrique (triangles représentés en rouge FIGURE 3.4).

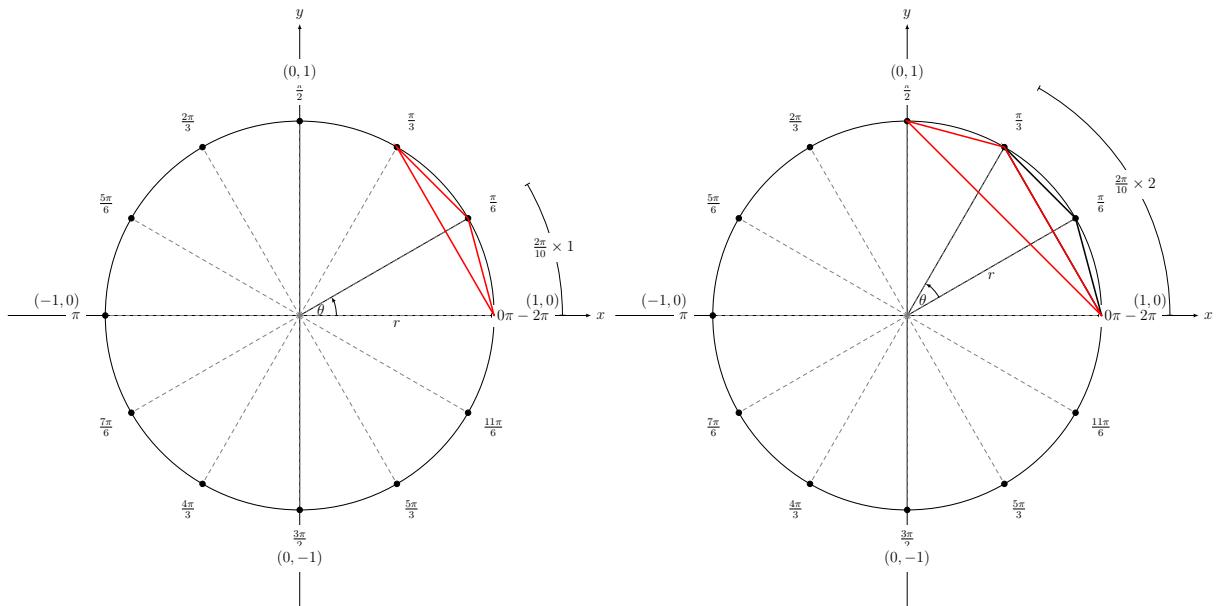


FIGURE 3.4 – Calcul des deux premiers points du polygone

La formule utilisée pour calculer la position des points du futur polygone est :

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} + \begin{pmatrix} \cos(\theta) \times xAspect \\ \sin(\theta) \times yAspect \\ 0 \\ 0 \end{pmatrix} \times polygonSize$$

where :

- $(x' \ y' \ z' \ w')^T$ sont les coordonnées du point à calculer,
- $(x \ y \ z \ w)^T$ sont les coordonnées du point précédent,
- θ est l'angle ($\equiv 0 \left[\frac{2\pi}{polygonNbSides} \right]$) formé par les deux points inscrit dans le cercle trigonométrique,
- $xAspect$ et $yAspect$ sont des paramètres relatifs à la taille de l'écran,
- $polygonSize$ est un facteur d'agrandissement.

Dans notre cas, la solution la plus simple est de construire un polygone ayant suffisamment d'arête pour que l'il humain assimile chaque point du nuage décrit juste précédemment (FIGURE 3.5(a)) à une forme ronde (FIGURE 3.5(b)). Le nombre des arêtes pour chaque polygone ainsi que leur taille est réglable.

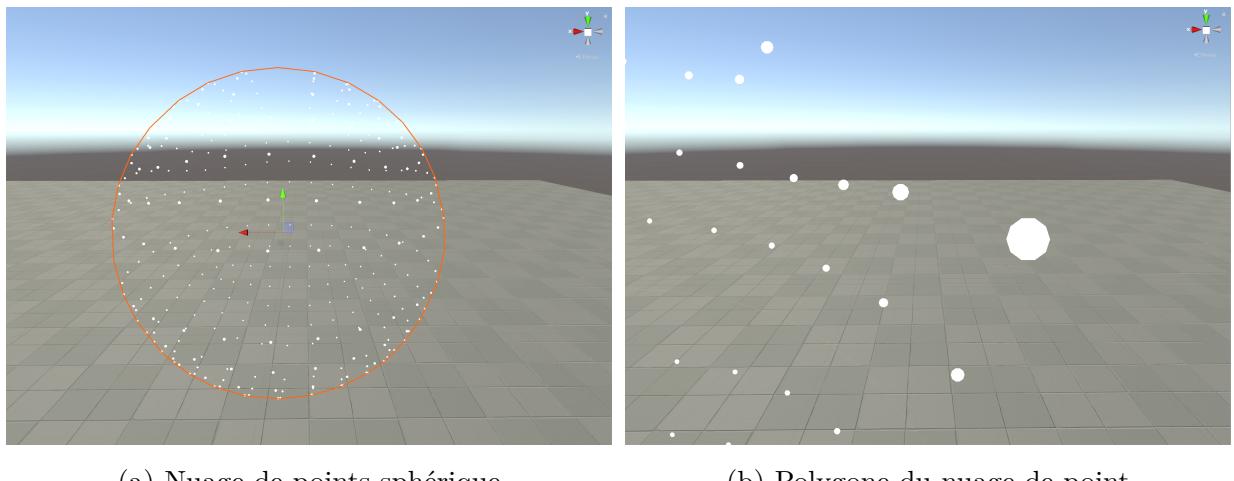


FIGURE 3.5 – Nuage de points sphérique et ses caractéristiques

En guise de conclusion, le *shader geometry* a modifié l'entité considérée par défaut par la structure *vertex*. Il prend la primitive la plus commune (point) en entrée pour calculer une liste de triangles, appelée une *bande de triangles*, formant le polygone. Enfin, le *shader fragment* fournit la couleur du point à visualiser (blanc dans ce projet) sur le rendu final de l'image en fonction de la texture fournie.

Toutefois, ce travail ne permet pas au jeu de données d'être représenté. La section suivante introduit la méthode.

3.2.2 Représentation des données par ce nuage

Afin de se débarrasser de cette sphère, nous devons fournir en entrée de la pipeline graphique la position des points que l'on souhaite visualiser dans le monde immersif. Un point sera logiquement représenté par une sphère, modèle de base Unity. La texture qui sera appliquée à cette sphère sera celle du shader. L'il humain verra un polygone en lieu et place d'une sphère composée de petits polygones trop petits à voir à l'échelle humaine.

Dans un premier temps, une classe basique contenant une méthode de lecture de fichiers Excel a été conçue. Chaque ligne renseigne la position de la donnée dans l'espace (les trois premières colonnes), puis quelques dimensions caractéristiques selon la base de données traitée (les colonnes suivantes). Les délimiteurs ont été définis par des expressions régulières mais il est possible de s'en passer.

Ensuite, un autre script permet, à partir de la lecture d'un fichier utilisant la classe précédente, d'instancier pour chacune de ses lignes une petite sphère dont la position sera celle indiquée dans le fichier. On normalise ensuite avec les valeurs minimales et maximales de chaque axe afin d'éviter l'obtention d'un nuage trop éparsé (FIGURE 3.6(a)). Au dessus de chaque sphère sera indiquée, si on le souhaite, l'une des dimensions, qui sera alors stockée en chaîne de caractères dans le Component *TextMeshPro* d'un GameObject fille de la sphère (FIGURE 3.6(b)). Sa rotation est implémentée par le même code que pour le nom de joueur. La principale différence avec la partie précédente est de devoir tenir à jour une liste de Components *Renderer* contenant celui de chaque sphère, qui profiteront tous de l'implémentation du shader.

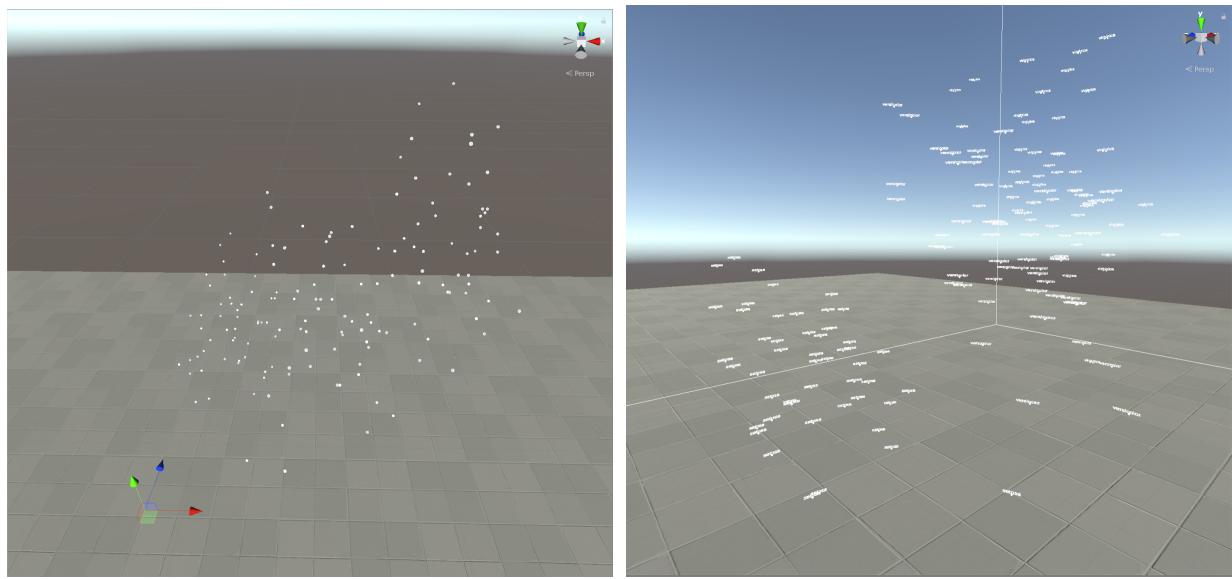


FIGURE 3.6 – Nuage de points de données et ses caractéristiques

Des méthodes vont maintenant devoir être trouvées afin de trouver des motifs ou des couches au sein du nuage de points grâce à quelques outils, largement utilisés en *Infographie*.

3.3 Amélioration de l'exploration visuelle des données

Le code MRTK concernant les shaders a été une base solide et une aide précieuse pour les sections suivantes. Voici une partie des shaders proposés par MRTK (FIGURE 3.7) :

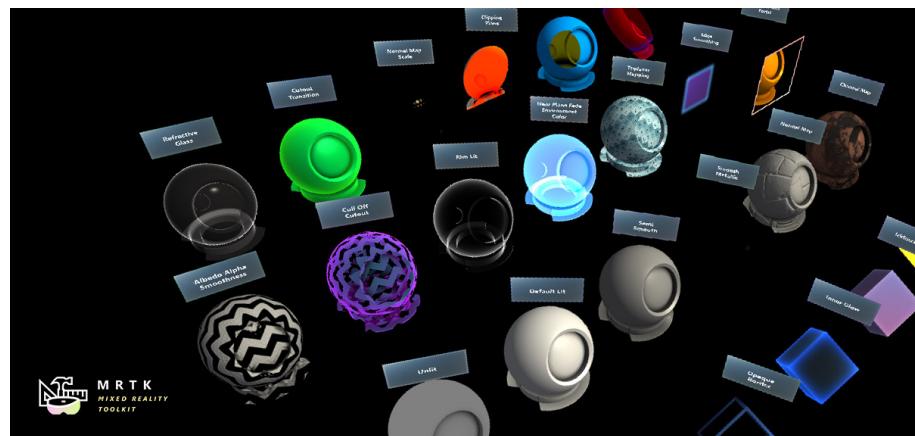


FIGURE 3.7 – Différents shaders disponibles sous MRTK

3.3.1 Exploration de données par plans coupants

L'une des premières idées que l'on a lorsque l'on souhaite explorer un nuage complexe de points est de faire du *Clustering*, i.e. de regrouper un ensemble de points par paquets, permettant de dégager un sens. En *Infographie*, l'une des méthodes les plus célèbres est le *clipping*. Son principe (montré en FIGURE 3.8) est de visualiser différentes couches de points au sein du nuage, par translation d'un plan sur celui-ci. D'ordinaire, les points du nuage se situant entre le plan et son porteur ne sont pas rendus (qualifiés de *clipped*), et ceux situés de l'autre côté restent inchangés jusqu'à ce qu'ils l'intersectent, d'où le nom de plan coupant (ou *clipping plane* en anglais) que l'on donne usuellement. Afin de mettre en évidence les points lors de l'intersection, une couleur orange leur est donnée, et transmise dès le début de la pipeline.

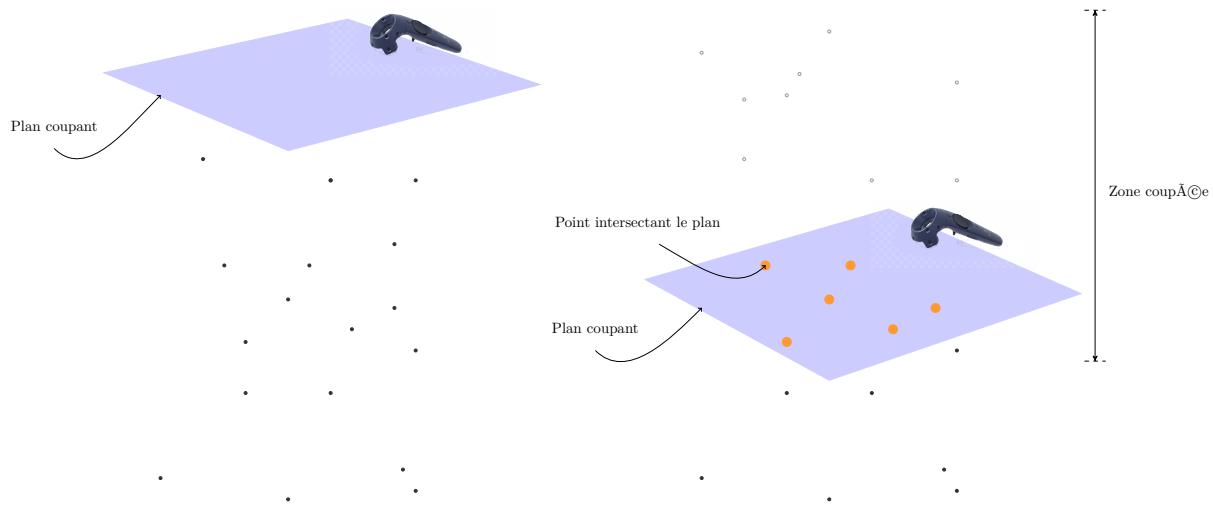


FIGURE 3.8 – Illustration du principe de clipping

Le shader attaché au matériau du plan a seulement pour but de le rendre transparent, de manière à voir les points derrière ce dernier. Quelques lignes réglant le taux de brume a été ajouté à un *shader vertex & fragment* classique et des tags pour la transparence a été fournie. La propriété de clipping est en réalité implémentée dans le shader associé au nuage de points. En effet, le plan n'est qu'un outil entraînant ou non le rendu d'un point, il est donc logique de coder cette propriété dans le shader précédent.

Pour ce faire, la distance de chaque point du nuage au plan est calculée en projetant ce point sur le plan (FIGURE 3.9).

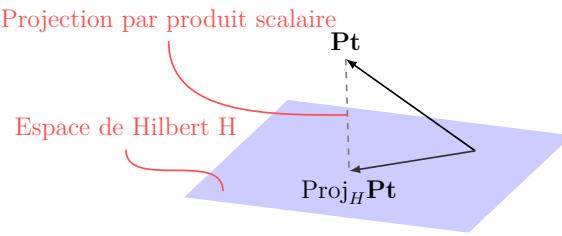


FIGURE 3.9 – Distance du point au plan par projection

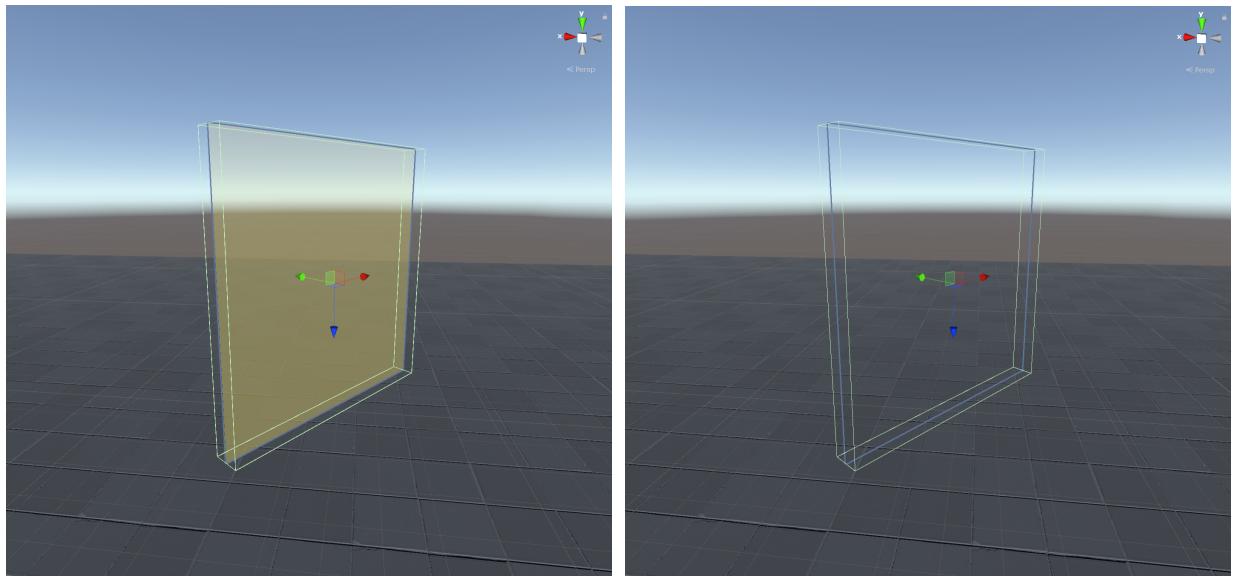
La formule suivante permet de calculer la distance d entre un point \vec{p} du nuage et le centre du plan \vec{c} , grâce à sa normale \vec{n} , sans normalisation :

$$\forall \vec{p} \in cloud, d = (\vec{p} - \vec{c}, \vec{n})_{\mathbb{R}^3}$$

Une fois l'implémentation de ce plan effectuée, la conséquence d'une méthode d'optimisation de rendu utilisée par Unity (comme dans une écrasante majorité des logiciels) a été constatée, communément appelée le *backface culling* (FIGURE 3.10). Cela consiste à rendre uniquement les faces de l'objet que l'utilisateur peut voir. Plusieurs méthodes existent pour rendre les deux faces du plan, parmi laquelle celle, plus astucieuse et peu coûteuse, de coller deux plans de telle sorte que leur rendu respectif soit opposé (selon la normale sortante des plans). On ne lui attache pas de Component *Rigidbody*, un simple Component *Fixed Joint* suffit afin qu'il suive le mouvement de l'autre plan soumis à la gravité. Il devra également porter un Component *Photon View* pour pouvoir être rendu dans l'application multi-joueurs.

Pour que cet objet soit plus intéressant à l'analyse, les points intersectant le plan ont été grossis et mis en surbrillance (FIGURE 3.11(a)). Mais en théorie, un plan n'a pas d'épaisseur ; on ne peut donc espérer une distance entre les points et le plan parfaitement nulle. Ainsi, il est nécessaire de laisser une certaine marge le long de l'axe de la normale au plan, la taille d'un polygone étant prise par défaut, à savoir 0.002 cm. Cette valeur est réglable : elle représente finalement l'épaisseur de la couche que l'on souhaite faire apparaître, calculé à partir du plan coupant.

Afin d'avoir une meilleure lecture des données, la dimension que l'on souhaite faire apparaître pour chaque point (évoquée en section 3.2.2.) a été limitée aux points inclus dans le plan coupant. Concernant le temps d'apparition du texte relatif au point, on ne peut tenir un raisonnement du type : "si le point n'est plus rendu (donc qu'il se trouve derrière le plan coupant), alors on n'affiche plus le texte" car ce code et la shader ne sont pas liés, et fournir la chaîne de caractère de chaque point au shader serait déraisonnable. Ainsi, une alternative est de vérifier si le temps écoulé depuis que le texte est apparu est

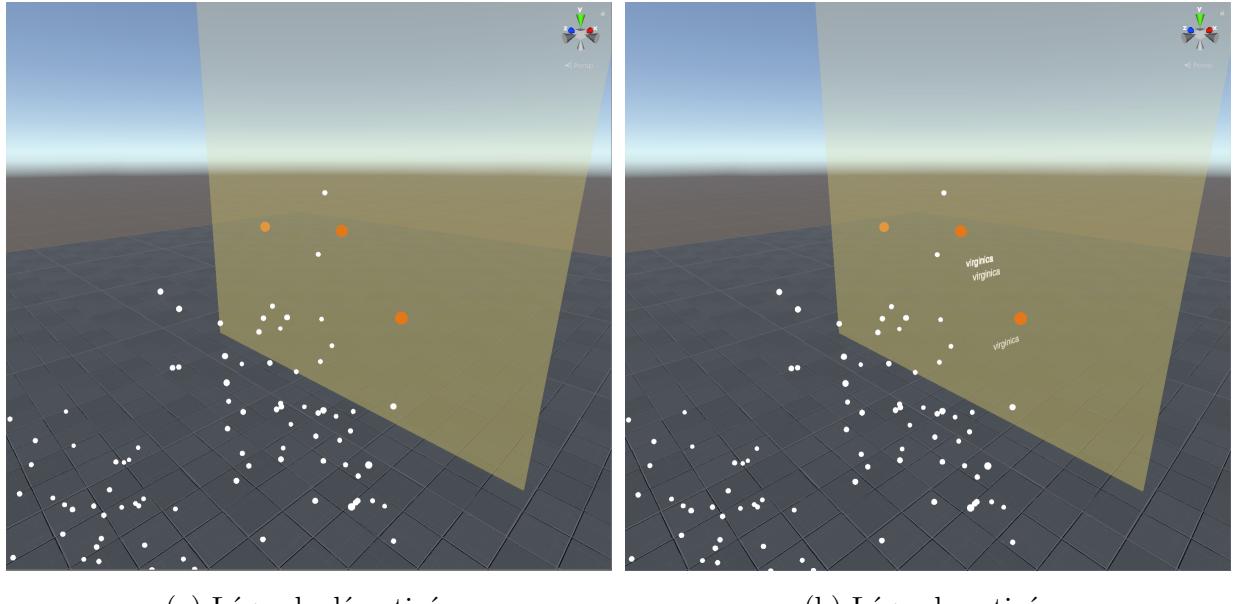


(a) Face du plan visible par le joueur

(b) Face opposée non rendue

 FIGURE 3.10 – Illustration du *backface culling*

supérieur à zero secondes par un script Timer. Les résultats FIGURE 3.11(b) montrent ce que l'on a obtenu.



(a) Légende désactivée

(b) Légende activée

FIGURE 3.11 – Résultats pour le plan coupant

La première version de ce code fonctionnel ne permet pas en revanche de prendre en compte la taille du plan coupant.

3.3.2 Plans coupants finis et problèmes rencontrés

Après maints tests effectués sur le nuage de points, force est de constater qu'une translation du plan à plusieurs mètres du nuage de points permettait également la disparition de points, ce qui n'est pas souhaité (FIGURE 3.12). Nous voulons idéalement fournir en entrée du shader les coordonnées frontières du plan, de manière à vérifier si les coordonnées d'un point appartient ou non au plan.

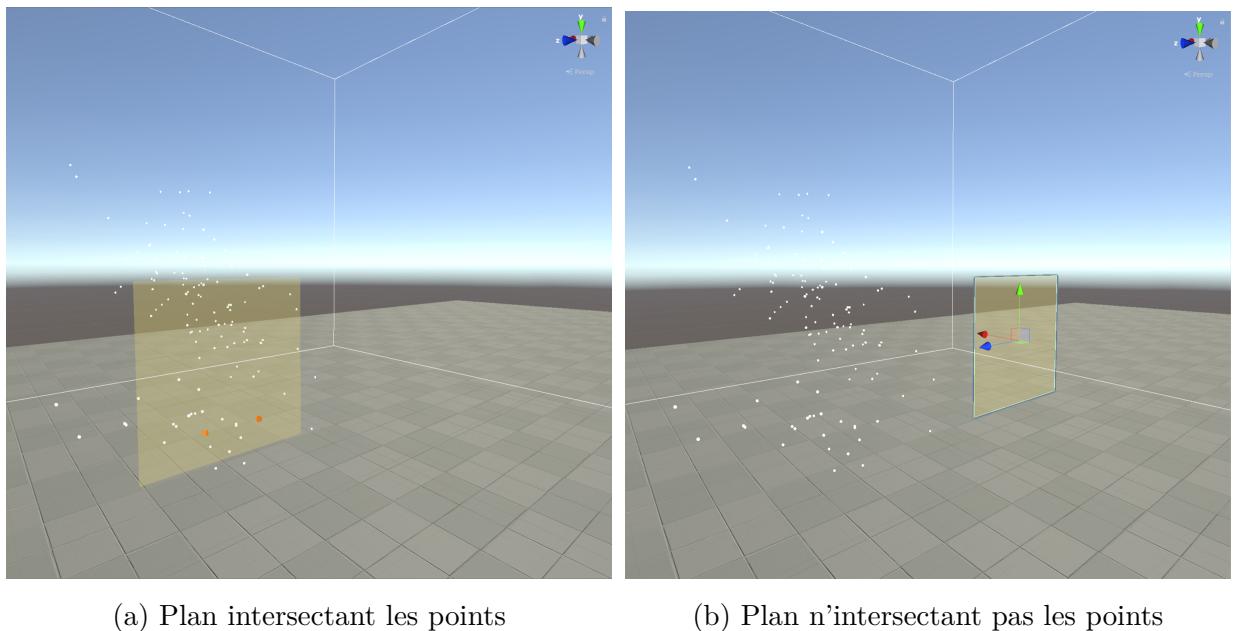
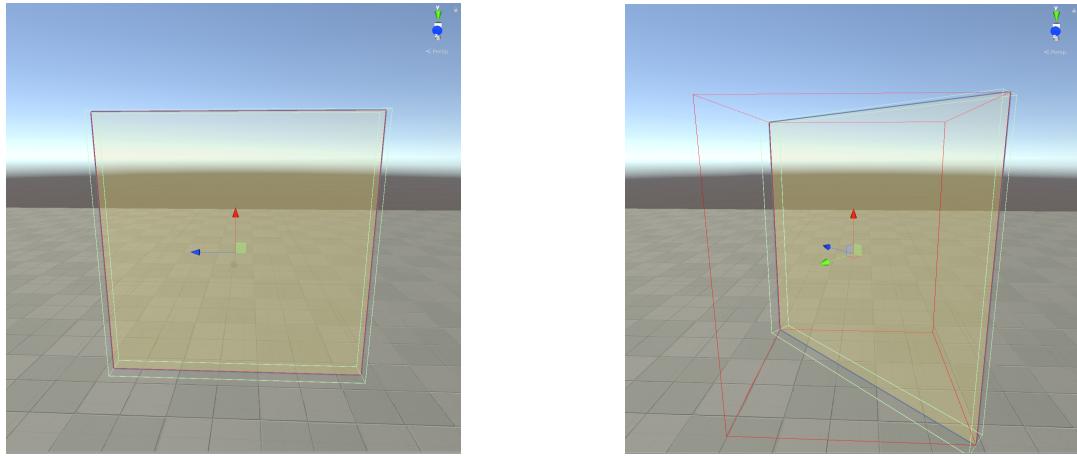


FIGURE 3.12 – Problème remarqué avec des plans coupants infinis

La première idée a été de se baser sur les Components *MeshCollider*, *MeshRenderer* et *MeshFilter* permettant chacun d'extraire les frontières du modèle 3D de base sur lequel il est attaché en utilisant l'élément *bounds*. Aucun résultat significatif a été obtenu. D'une part, les scripts ne pouvaient pas modifier les frontières en conséquence lorsque le plan subissait une rotation ou une mise à l'échelle, comme illustré ci-dessous (FIGURE 3.13). Un tel comportement est souvent référé dans la littérature à AABB (pour *Axis-Aligned Bounding Box* en anglais). C'est efficace tant du point de vue du calcul que de la mémoire car c'est une première approximation grossière, mais ne permet pas de bien épouser la forme de l'objet, donc cela ne va pas nous aider outre mesure. D'autre part, les frontières étaient bien souvent plus grandes que le plan lui-même.



(a) Plan modifié uniquement par translation (b) Plan modifié uniquement par rotation

FIGURE 3.13 – Evolution des frontières d'un plan selon la modification apportée

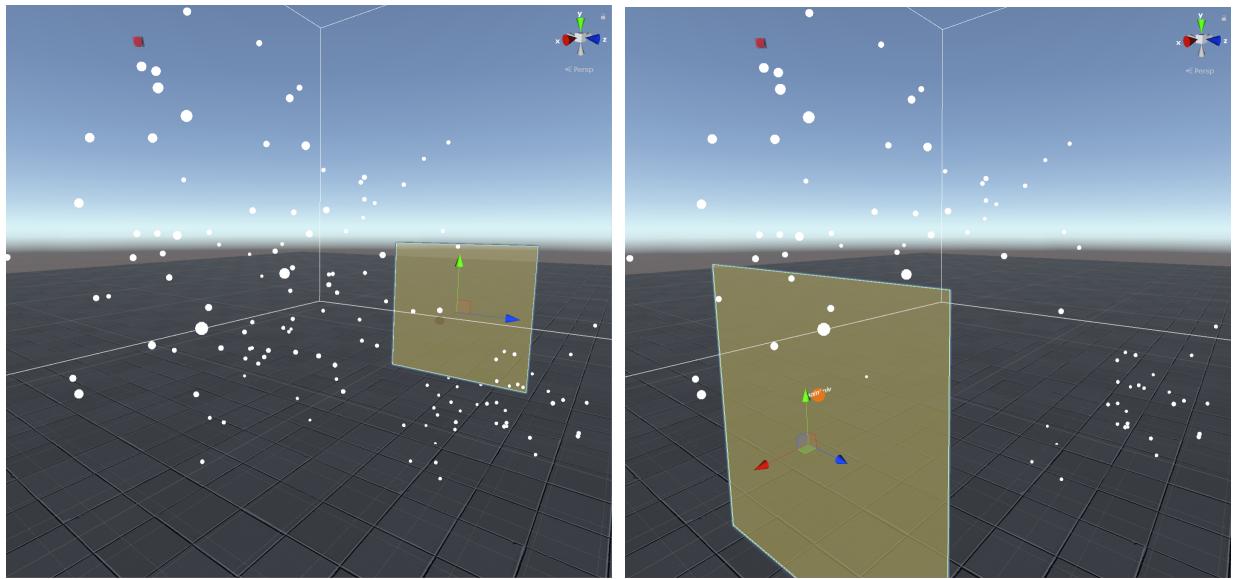
Après avoir tenté bon nombre de méthodes permettant de calculer les frontières d'un plan, aucune ne permettait de les obtenir lorsqu'une rotation était menée. Ainsi, plutôt que d'avoir des plans, des cubes d'épaisseur très fines ont été modélisés afin de contourner la difficulté. Les frontières obtenues ont été correctes, mais ne se modifiaient toujours pas en fonction de la rotation, ceci étant dû à l'élément *bounds*, peu adaptatif. Ainsi, il a été nécessaire de recalculer un par un les huit coins du pavé, en prenant en compte les possibles rotations qui allaient l'animer, pour obtenir des frontières précises. La formule générale pour calculer un coin est :

$$\boxed{\text{corner} = R \times \begin{pmatrix} sgn(width/2) \\ sgn(height/2) \\ sgn(depth/2) \end{pmatrix} \times \begin{pmatrix} x \\ y \\ z \end{pmatrix}}$$

where :

- *sgn* is la fonction signe offrant $2^3 = 8$ combinaisons possibles, nous donnant les huit coordonnées des coins,
- *R* est la matrice de rotation de la matrice calculée par la méthode Unity `SetLookRotation`,
- *width*, *height* and *depth* correspondant aux valeurs du plan en utilisant le paramètre *localScale*,
- $(x \ y \ z)^T$ les coordonnées du centre du plan.

Fournir ensuite les valeurs extrêmales pour chaque axe du repère global a permis d'obtenir le résultat de clipping escompté, avec la condition discriminante stipulant que la normale au plan doit être colinéaire à l'un des vecteurs du repère global (voir FIGURE 3.14).



(a) Plan derrière le nuage de points

(b) Plan clippant les points sur sa trajectoire

FIGURE 3.14 – Plan coupant selon la normale au plan colinéaire à l'axe x

Des conditions basées sur le produit scalaire pour savoir si un point était devant ou derrière le plan ne furent pas suffisantes pour obtenir un plan coupant fini rotatif. En effet, il fallait uniquement supprimer les points se trouvant dans la zone située derrière le plan, zone de direction principale la normale sortante au plan. L'astuce a donc été d'avoir une boîte coupante (*clipping box* en anglais) parent du plan, rendant uniquement une face du cube. La profondeur de la boîte (correspondant à l'axe normal au plan) sera très importante et dépassera largement la position du casque. Le joueur ne s'apercevra pas de l'astuce dans l'environnement immersif.

Implémenter une boîte coupante est analogue à un plan coupant, à ceci près que la condition de clipping (appliquée à l'intérieur ou à l'extérieur de la boîte) est différente, impliquant une nouvelle formule pour calculer la distance :

$$d = \left\| \max(\vec{0}, \vec{d}) \right\|_{\mathbb{R}^2} + \min(\max(d_i)_{0,i \in [|1,3|]})$$

avec $\forall \vec{p} \in cloud, \vec{d} = |R \times (\vec{p}, 1)| - boxSize$ le vecteur distance de \vec{p} au plan avec R sa matrice de rotation.

Voici les résultats qui ont été obtenus, en FIGURE 3.15 pour la boîte coupante, et en FIGURE 3.16 pour la simulation du plan coupant.

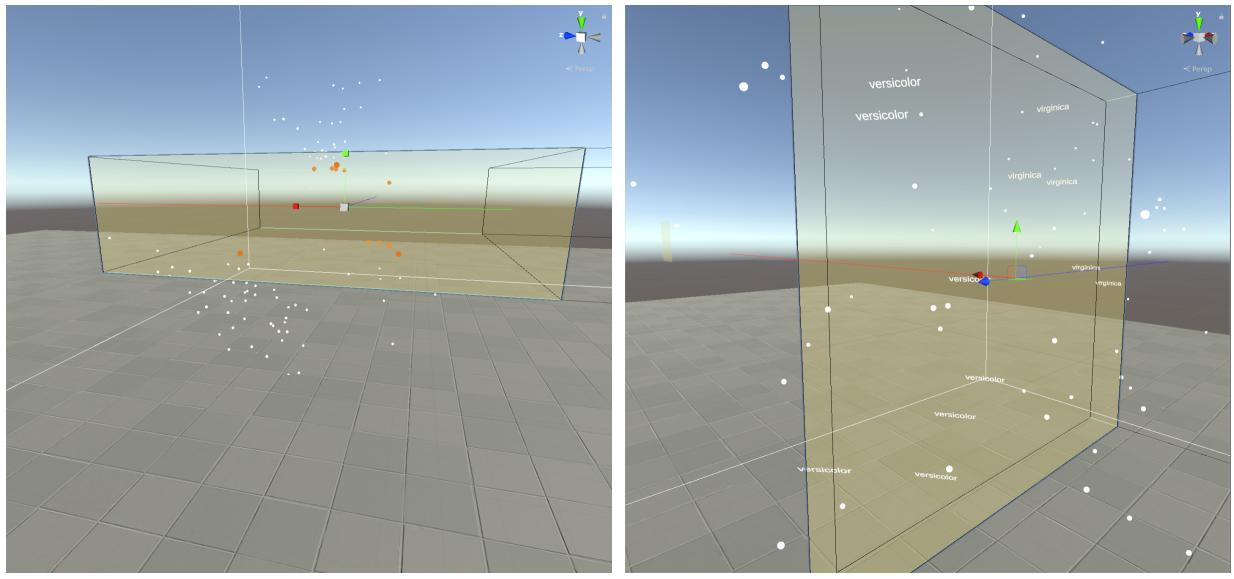


FIGURE 3.15 – Résultats obtenus pour la boîte coupante

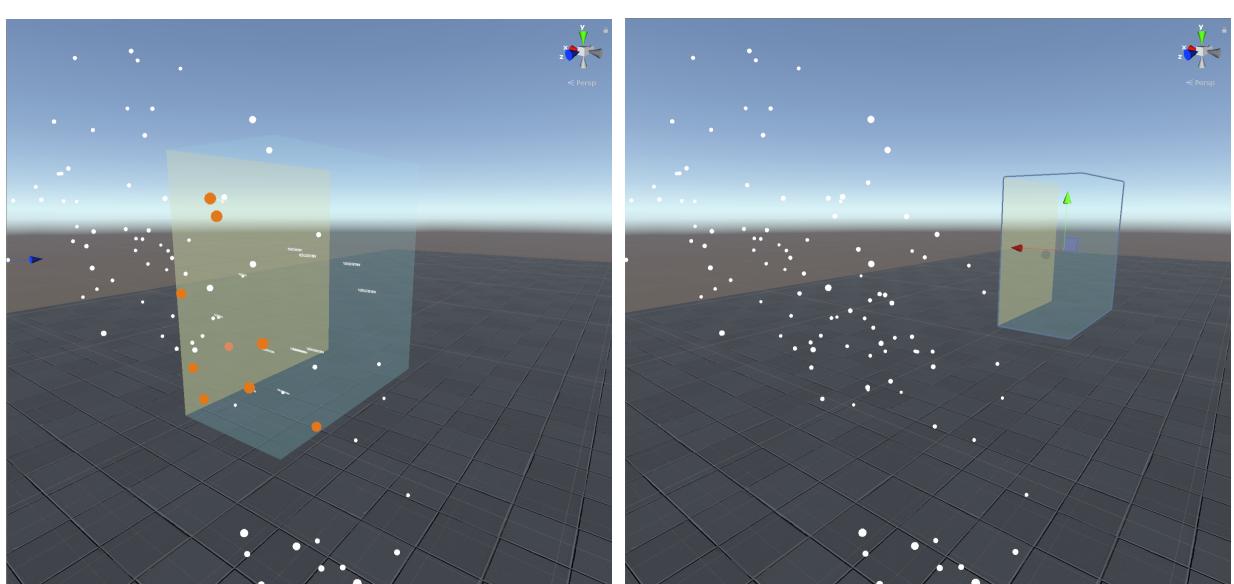


FIGURE 3.16 Résultats du plan coupant avec boîte accolée

Maintenant qu'un plan coupant a été implémenté, réfléchissons à la manière par laquelle on pourrait en utiliser deux pour l'analyse.

3.3.3 Implémentation d'un prototype d'Highlight Planes

Le concept d'Highlight Planes est issu d'un papier de recherche [12], écrit en majorité par les mêmes personnes que celui de ImAxes. Ils ont tenté d'explorer divers moyens d'interagir facilement avec les données en environnement immersif. Parmi les techniques intéressantes qu'ils ont travaillé, les *scaptics* (venant de *haptics* en anglais) consistent en bref en des retours de vibrations selon la densité locale du nuage de points (FIGURE 3.17(a)). Une autre idée a été de s'inspirer des plans coupants afin de faire apparaître des motifs dans le nuage (détecter un ensemble de points intéressant intersectant le plan, ou des trous comme en FIGURE 3.17(b) et (c)).

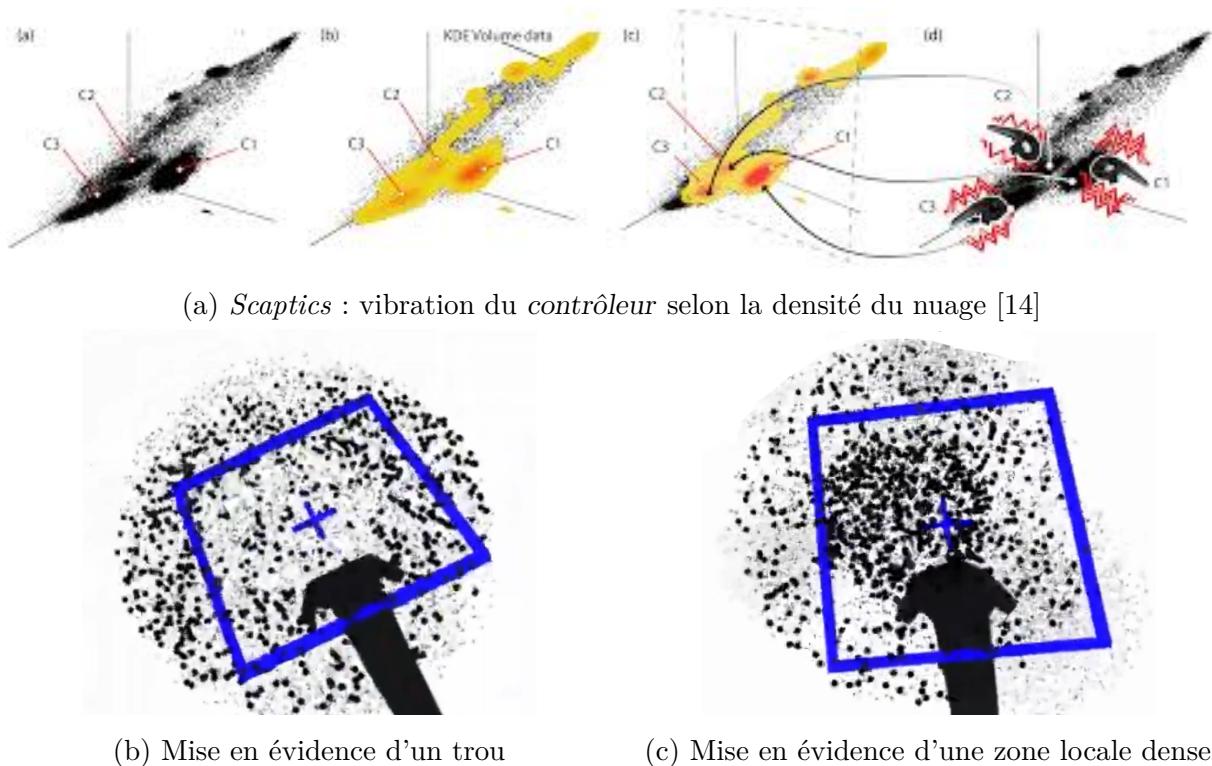


FIGURE 3.17 – Illustration des scaptics et des *highlight planes* [14]

De ces deux techniques a été imaginé un hybride intéressant, consistant à mettre en lumière un certain nombre de points du nuage souhaité en disposant parallèlement deux plans coupants. Les points se situant dans la zone entre ces deux plans seraient donc mis en surbrillance, comme illustré en FIGURE 3.18.

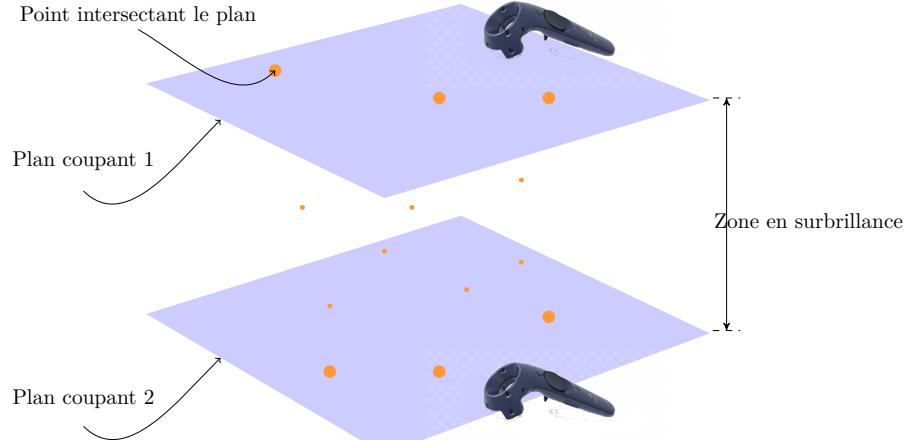


FIGURE 3.18 – Illustration de la fonctionnalité de mise en surbrillance

Cette surbrillance s'est d'abord réalisée en utilisant la texture orangée évoquée précédemment tout au long du pipeline graphique, comme illustré en FIGURE 1.19.

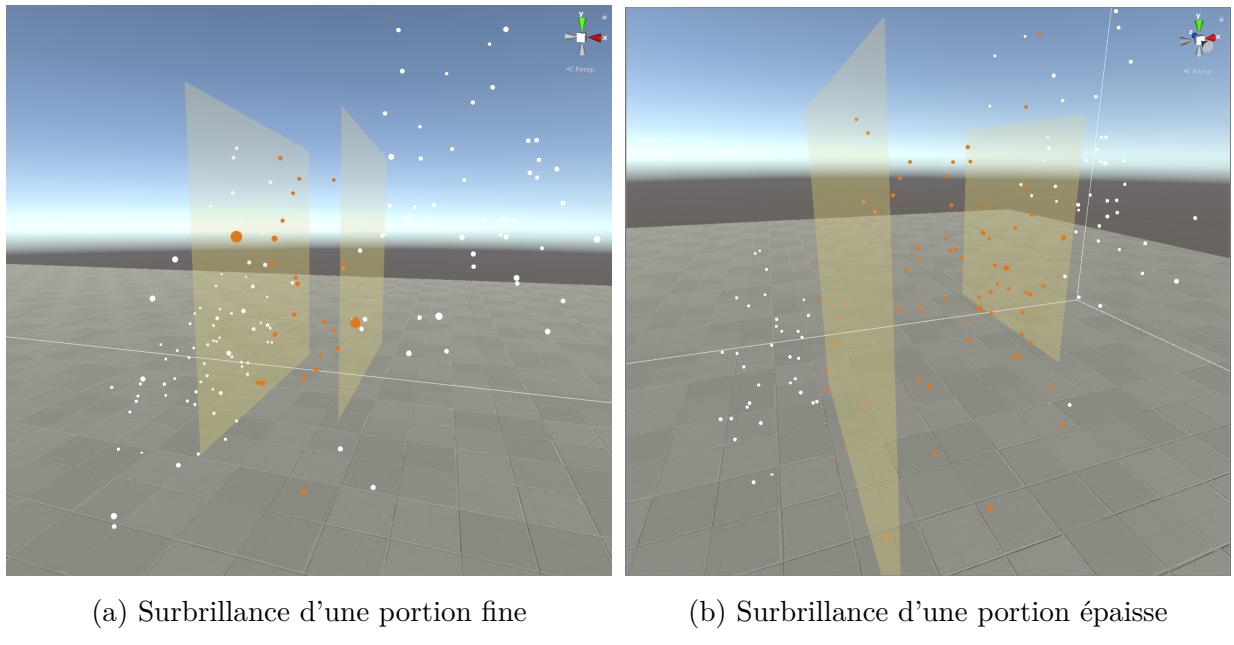


FIGURE 3.19 – Premier prototype des Highlight planes

Ensuite, l'implémentation de modèles de réflexion lumineuse classique s'est vite imposée pour un meilleur rendu graphique. Le premier qui a été implémenté est le *modèle de réflexion Lambertienne* (FIGURE 3.20). Il est accessible et permet de donner aux ma-

tériaux un aspect mat, dû à la réflexion *diffuse*. La loi de Lambert affirme que l'intensité lumineuse d'un rayon lumineux incident sur une surface réfléchissante suit une loi cosinus, quelque soit le point de vue du joueur.

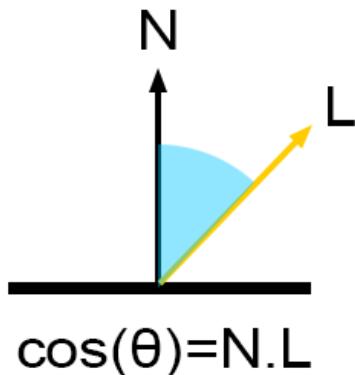


FIGURE 3.20 – Modèle de réflexion lambertienne et vecteurs utiles

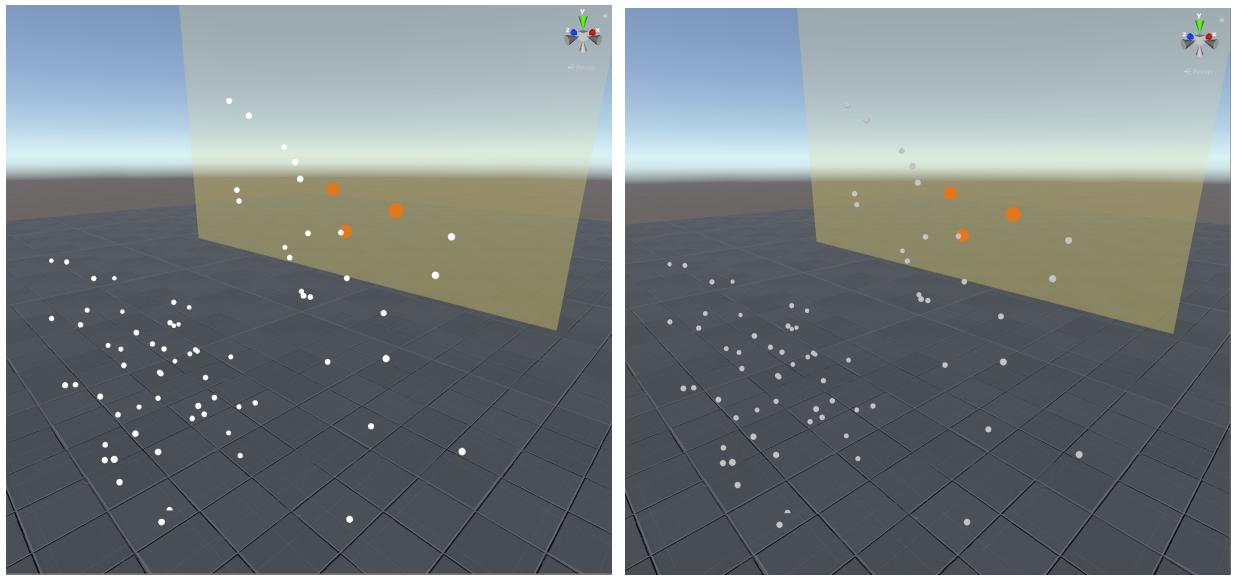
Ainsi, un effet de surbrillance a été indirectement créé par application du modèle de Lambert à tous les points qui sont par conséquent atténusés, excepté ceux orange. La formule utilisée a été conditionnée pour ne jamais passer sous la barre de zéro :

$$I_{refl} = \max \left\{ 0, (\vec{l}, \vec{n})_{\mathbb{R}^3} \times c \times I_{incid} \right\}$$

avec :

- \vec{l} le vecteur de direction normalisée de la lumière,
- \vec{n} est la normale au point appartenant à la surface,
- c est la couleur de la surface (valeur),
- I_{incid} est l'intensité de la lumière incidente et I_{refl} celle de la diffuse.

Vous trouverez ci-dessous les résultats obtenus (FIGURE 3.21) :



(a) Lumière diffuse désactivée

(b) Lumière diffuse activée

FIGURE 3.21 – Resultats pour le modèle de réflexion lambertienne

Pour donner aux points orange la surbrillance souhaitée, le *modèle de réflexion de Blinn-Phong* a dû être implémenté. Il donne aux matériaux une lumière spéculaire (FIGURE 3.22). Ce modèle est dépendant du point de vue du joueur mais également de l'angle de la lumière incidente.

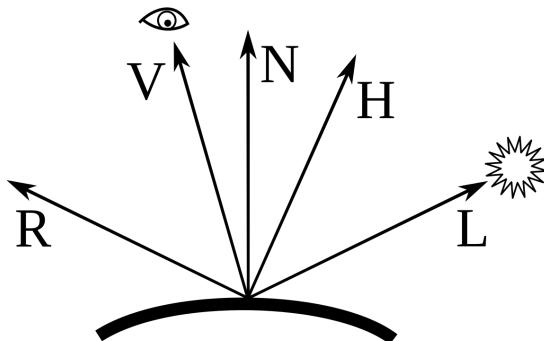


FIGURE 3.22 – Modèle de réflexion Blinn-Phong et vecteurs utiles

Toutefois, il est plus difficile à implémenter dans un shader que le modèle de Lambert. Ce travail est en cours.

En conclusion, nous pouvons résumer par le diagramme ci-dessous (FIGURE 3.23) la pipeline graphique codée pour cette application :

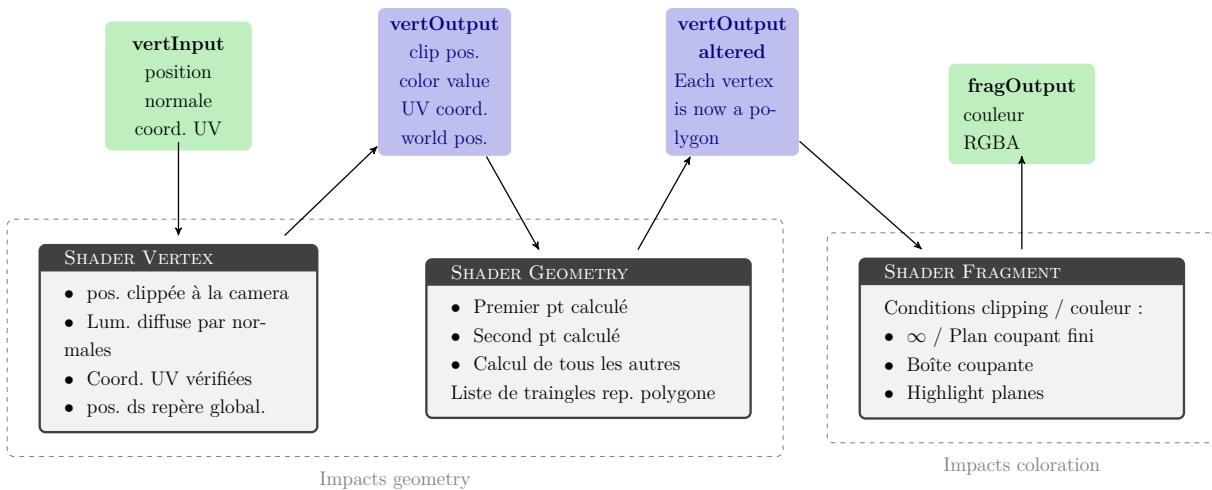


FIGURE 3.23 – Pipeline graphique de l'application

Penchons-nous maintenant davantage sur la manière avec laquelle chaque joueur peut interagir entre eux afin de partager quelques informations intéressantes.

3.3.4 Perception de la vision d'un utilisateur par tous

Cette section est l'origine du chapitre. Lorsque l'équipe a réfléchi à une solution collaborative d'une visualisation immersive de données, la première préoccupation a été de savoir comment un utilisateur pourrait, à un instant donné, profiter de l'exakte image d'un autre utilisateur en temps réel. Imaginons qu'un utilisateur ait déduit une information de valeur par une configuration précise de plans dans un nuage, uniquement visible de son point de vue. Comment tous les autres utilisateurs pourrait profiter de cette vue sans avoir à se déplacer à son exacte position ? Ce travail est en cours.

Le stage se finissant fin Septembre, il restera donc un mois pour finir le travail entamé tout au long de ce stage.

3.4 Objectifs et perspectives d'évolution pour la fin du stage

Ce sera l'occasion d'approfondir et de perfectionner les différents points abordés lors des sections 2 et 3. Les priorités seront les suivantes :

- implémenter une solution permettant de visualiser l'écran d'un autre joueur, à un instant t, de telle manière à profiter de sa vision de données, avec son angle de vue ;

- s'intéresser plus amplement à d'autres outils comme IATK, et de s'en inspirer afin d'importer des *analytics* supplémentaires pour le projet. Cela pourrait faire intervenir des problématiques de rendu graphique volumique, intéressantes pour trouver des motifs sujet à l'analyse dans les données ;
- affiner la communication entre les différents utilisateurs, en permettant notamment qu'un joueur refuse d'accorder à un autre la propriété d'un objet qu'il possède. Pour le moment, les transferts d'appartenance sont automatiques dès que l'on appuie sur la gâchette d'un *contrôleur* lorsque les *colliders* d'un *contrôleur* et de l'objet sont en contact ;
- importer le projet dans l'outil ImAxes. Quelques collègues ont importé des améliorations de leur côté. Jonathon HART a réalisé une fusion entre les outils ImAxes et MapsSDK-Unity. Son travail permet de pouvoir visualiser, sur une carte 3D de l'Australie, des diagrammes en bâtons émergents de locations précises lorsqu'une visualisation d'ImAxes est suffisamment proche de la carte. Et Simon MALNAR a apporté une solution multi-plateforme pour le projet. ImAxes n'ayant pas été conçu pour être utilisé en multi-joueurs initialement, intégrer pleinement la dimension multi-joueurs sera difficile. Et à cet égard, mes contributions concernant la compréhension de certains problèmes et l'apport de solutions potentielles dans la visualisation réseau de données est une première étape cruciale pour y parvenir.

Conclusion

L'objectif initial du stage était d'implémenter une solution pour importer l'outil de visualisation et d'analyse de données ImAxes, fonctionnant uniquement en RV pour le moment, pour de la RM. En théorie, elle devait permettre la manipulation d'un ensemble de visualisations par les gestuelles HoloLens et la commande vocale, et idéalement une utilisation collaborative de l'application. Toutefois, le temps passé n'a pas été suffisamment fructueux, l'objectif du stage n'a pas été clairement fixé dès le début, et les casques HoloLens2 n'ont pu être obtenu à cette époque. Pour toutes ces raisons, il a été décidé de réorienter plus explicitement le stage sur le développement d'une solution multi-joueurs, avec un rendu graphique de qualité, fonctionnelle pour la RV. En ce sens, de nettes améliorations ont été apportées.

Le perfectionnement de ma maîtrise en Unity ainsi que l'appréhension des technologies virtuelles, alors totalement inconnues pour moi jusqu'alors, a réclamé un temps considérable, ce qui a ralenti la progression en début de stage. C'est après une concertation avec l'équipe responsable qu'un objectif a été fixé, répondant à la fois aux besoins du laboratoire et à mes objectifs professionnels.

L'état actuel du projet permet d'entrevoir de nombreuses perspectives d'améliorations. Quelques papiers de recherche complémentaires avec ce qui a déjà été codé n'a pas encore été exploité. En particulier, il aurait été possible, par une compréhension profonde de structures algorithmiques d'autres projets, d'obtenir un nombre plus important d'*analytics immersifs*. D'autre part, les capacités d'analyse de données (en particulier la mise en surbrillance) pourrait être améliorée par un travail sur des effets de lumières plus élaborés.

J'ai pour ma part augmenté considérablement ma maîtrise en programmation en Unity, mais également appris à trouver des solutions algorithmiques et géométriques comblant les quelques lacunes entrevues du moteur de jeu. J'ai également été conscient de l'importance ainsi que du potentiel des technologies de Réalité Virtuelle et Mixte, les ayant étudiées sous beaucoup d'aspects. Il est certain que ce stage sera une plus-value indéniable afin de me familiariser avec des techniques largement utilisées dans le monde des jeux vidéo et des effets spéciaux que j'ai pour ambition d'intégrer.

Bibliographie

Articles scientifiques

- [1] Benjamin Bach Christophe Hurter MAXIME CORDEIL Andrew Cunningham. “IATK : An Immersive Analytics Toolkit”. Thèse de doct. Monash University, University of South Australia, University of Edinburgh, Ecole Nationale de l’Aviation Civile, 2019.
- [5] Akira Utsumi Fumio Kishino PAUL MILGRAM Haruo Takemura. “Augmented Reality : A class of displays on the reality-virtuality continuum”. Thèse de doct. ATR Communication Systems Research Laboratories, 1994.
- [9] Tim Dwyer Bruce H. Thomas Kim Marriott MAXIME CORDEIL Andrew Cunningham. “ImAxes : Immersive Axes as Embodied Affordances for Interactive Multivariate Data Visualisation”. Thèse de doct. Monash University, University of South Australia, 2017.
- [11] Evan Suma Rosenberg MAHDI AZMANDIAN Timofey Grechkin. “An Evaluation of Strategies for Two-User Redirected Walking in Shared Physical Spaces”. Thèse de doct. USC Institute for Creative Technologies, 2017.
- [14] Clement Robin Barrett Ens Bruce Thomas Tim Dwyer ARNAUD PROUZEAU Maxime Cordeil. “Scaptics and Highlight-Planes : Immersive Interaction Techniques for Finding Occluded Features in 3D Scatterplots”. Thèse de doct. Monash University, University of South Australia, 2019.

Pages web

- [2] CSIRO TEAM. NEAR project. <https://near.csiro.au/assets/46008de4-56b8-64e4-aece-b7970f28e91a>.
- [3] MICROSOFT. Csharp documentation. <https://docs.microsoft.com/en-us/dotnet/csharp/>.
- [4] MICROSOFT. Mixed Reality Academy. <https://docs.microsoft.com/en-us/windows/mixed-reality/tutorials>.

- [6] Microsoft TEAM. Maps-SDK Unity. <https://github.com/microsoft/MapsSDK-Unity>.
- [7] Geoscience Australia DATA61 et Clean Energy Council TEAMS. AREMI plateform. <https://www.nationalmap.gov.au/renewables/>.
- [8] Unity TECHNOLOGIES. Unity documentation. <https://docs.unity3d.com/Manual/index.html>.
- [10] PHOTON. PUN documentation. <https://doc.photonengine.com/en-us/pun/v1/getting-started/pun-intro>.
- [12] Alan ZUCCONI. Shaders tutorial. <https://www.alanzucconi.com/2015/06/10/a-gentle-introduction-to-shaders-in-unity3d/>. 2015.
- [13] Anthony BLECHET. Shaders Laboratory. <http://www.shaderslab.com>. 2017.

Annexe 1 - Différences entre RV, RA et RM

Ce concept de RM est très souvent confondu avec la RA car les technologies associées offrent toutes deux la vision d'un monde enrichi. La différence clé est que seule la Mixte permet une interaction avec les objets virtuels qui ont enrichi le monde. La digitalisation est parfois si importante qu'elle peut parfois se confondre avec la RV, mais le principe de la RM est de fixer la position des hologrammes dans l'espace des objets réels. Vous trouverez une infographie en FIGURE 1.

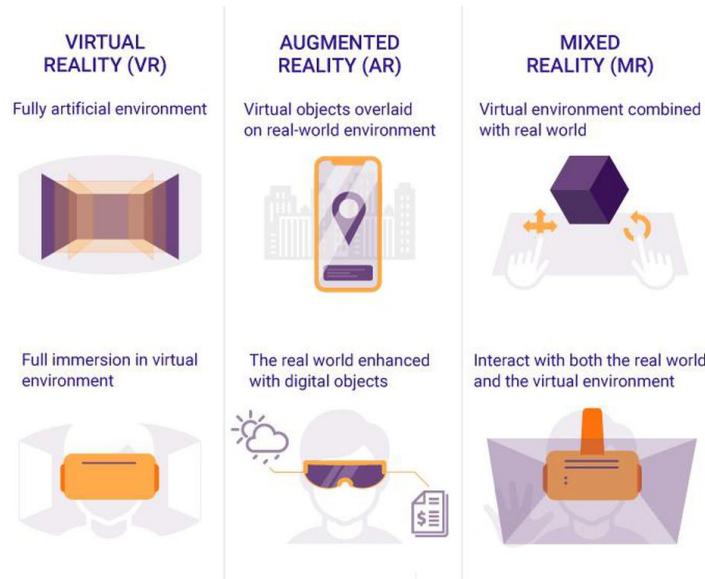


FIGURE 1 – Différences entre RV, RA et RM

Pour une meilleure assimilation des limites entre RV, RA et RM, Milgram a introduit en 1994 [5] le concept de *continuum Réalité-Virtualité* illustré FIGURE 2, apportant une définition plus large à la RM, la qualifiant d'expérience se situant sur le continuum entre Réalité à l'extrême gauche et Virtualité à l'extrême droite.

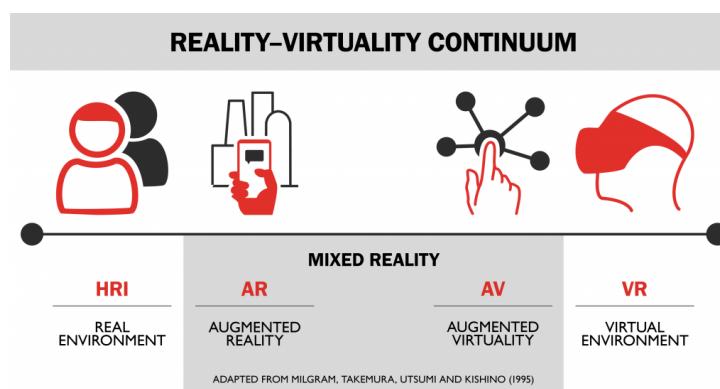


FIGURE 2 – Continuum de Milgram

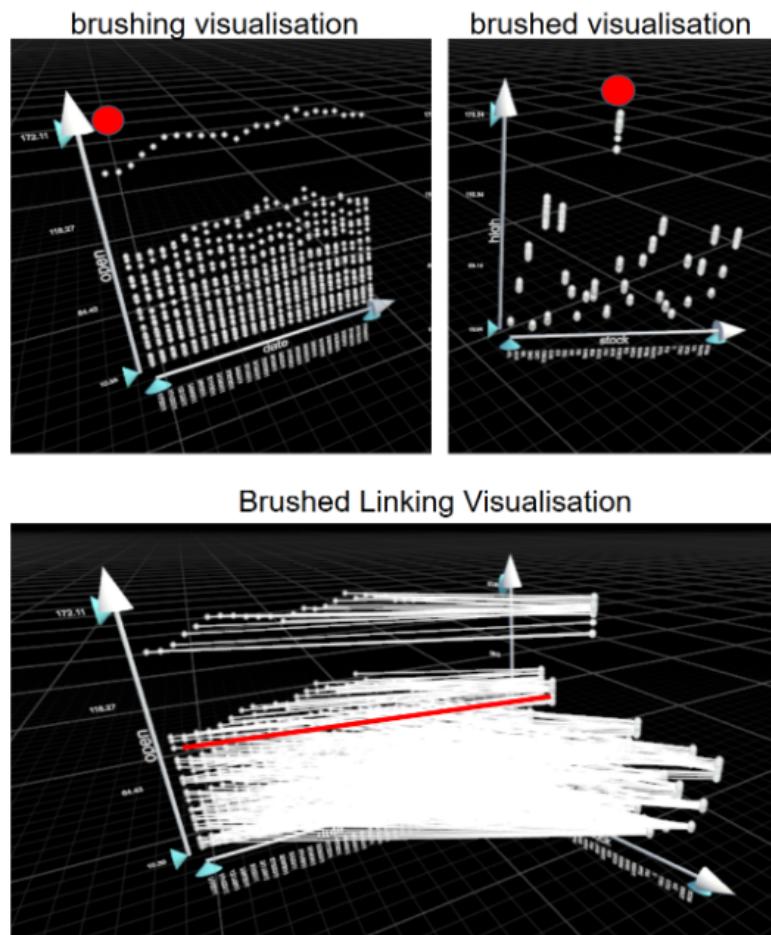
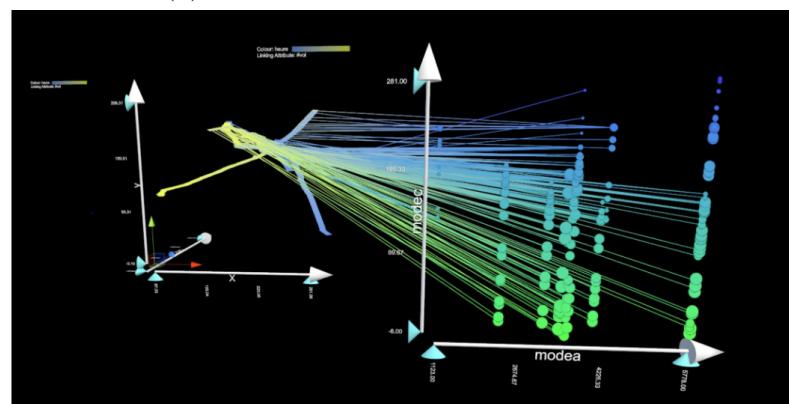
Annexe 2 - IATK : la version évoluée de ImAxes

IATK (pour Immersive Analytics Toolkit) [1] est un projet Unity permettant la création de visualisation de données en RV. Cet outil libre de droits a été conçu par l'équipe de ImAxes et quelques autres chercheurs. Le package VRTK est requis si l'on souhaite faire tourner l'application.

IATK peut être vu comme une version évoluée d'ImAxes, dont les principales améliorations sont :

- une possibilité de fournir en entrée de l'outil une base de données type Big Data (de l'ordre de grandeur du million d'entrée), et la possibilité de mise à l'échelle en fonction de ce sur quoi on souhaite se focaliser ;
- une plus large palette d'outil d'analyse, comme un *brushing and linking* abouti (FIGURE 1.3(a)) permettant la surbrillance de points, des liaisons de visualisations plus précises (FIGURE 1.3(b)),... ;
- une diversité de visualisation plus grande (FIGURE 1.4) que ImAxes comme les histogrammes 3D, la superposition d'histogrammes de différentes couleurs,... ;
- une interface GUI pour une meilleure gestion de son expérience d'*immersive analytics*.

Lorsqu'un graphique est construit, une grande maîtrise des paramètres de la visualisation est possible dans la fenêtre Hierarchy. On peut ensuite lancer l'application pour visualiser Le graphique construit.

(a) Principe du *brushing and linking*

(b) Principe des représentations liées

FIGURE 3 – Améliorations de l'analyse disponible dans ImAxes [1]

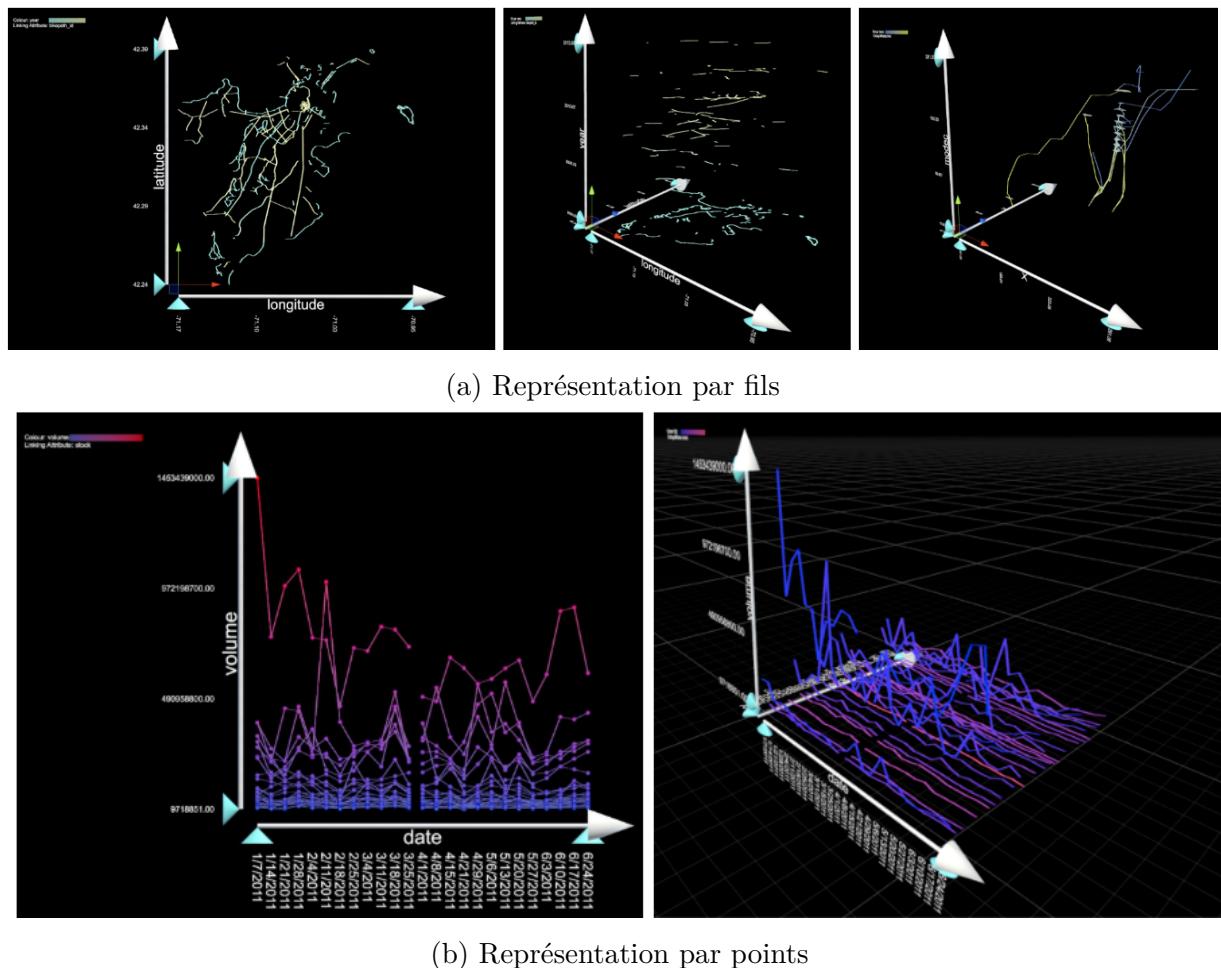


FIGURE 4 – Représentations innovantes, non présentes dans ImAxes [1]

Annexe 3 - Couches réseau en UNet

L'image ci-dessous FIGURE 5 illustre comment la HLAPI Unity UNet est construite : la racine est naturellement la LLAPI, puis ensuite des couches successives sont apportées pour additionner de nouvelles fonctionnalités, permettant principalement la synchronisation d'états entre joueurs incluant une sérialisation haute-performance, des classes réseau pour une meilleure maîtrise des notions,...

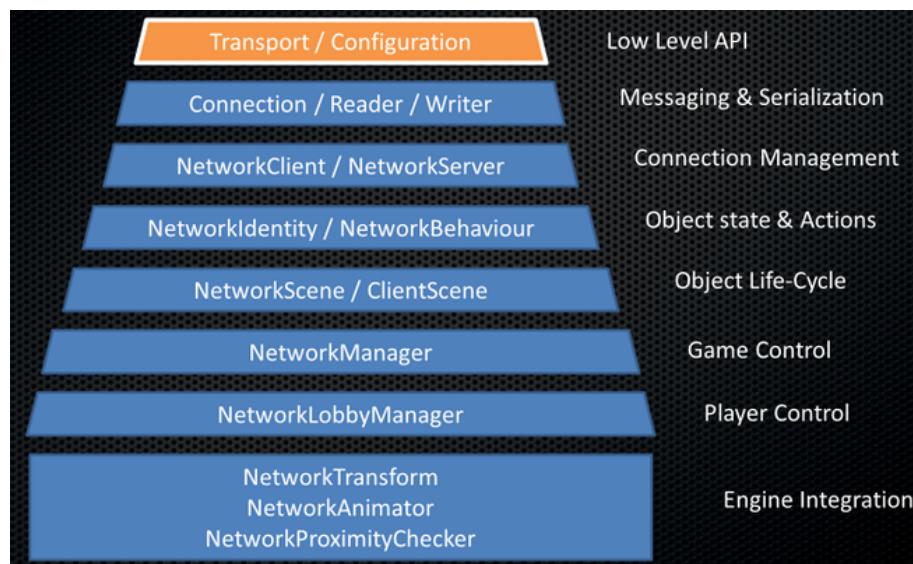


FIGURE 5 – Couches HLAPI réseau UNet

Pour indiquer à Unity que l'on souhaite utiliser son HLAPI, il suffit simplement d'utiliser le namespace `UnityEngine.Networking`