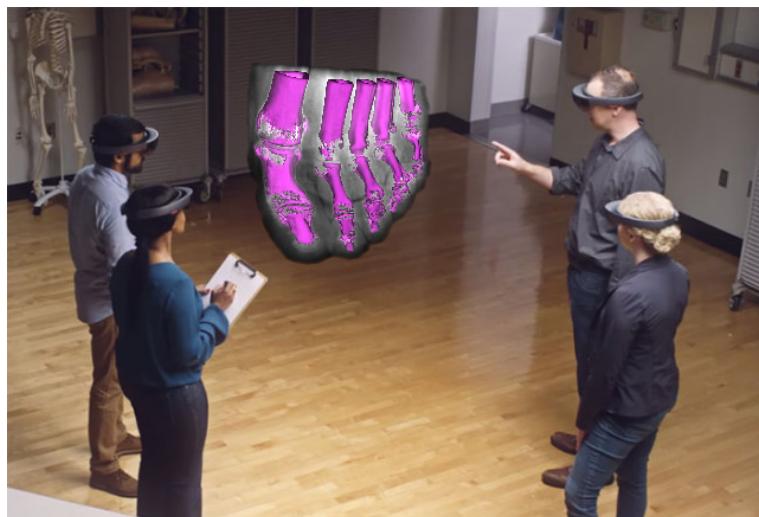


RAPPORT DE TRAVAIL DE BACHELOR

Lancer de rayon GPGPU simple sur serveur
dédié et transmission d'images
stéréoscopiques sur HoloLens pour le projet
HoloM3D



Auteur: Arnaud Droxler
Encadrant: Stéphane Gobron
Expert: Patrick Salamin, Logitech
Mandant: Prof. Christian Simon, CHUV
Date: September 9, 2017
Version: V2.0

Résumé

L'imagerie médicale est un domaine qui réunit la médecine et l'infographie. En effet, après que les médecins aient procédé à des examens, ils obtiennent des données sur le corps du patient. Pour traiter celui-ci, ils ont besoin de visualiser ces données. On va pouvoir afficher à l'écran des volumes 3D de données à l'aide de l'infographie. L'HoloLens est une nouvelle technologie développée par Microsoft. Il s'agit d'un casque de réalité augmentée. Il permet d'afficher des hologrammes qui s'intègrent dans le champ de vision de l'utilisateur. Les hologrammes peuvent être disposés dans l'espace et visualisés par plusieurs personnes en même temps. L'idée directrice du projet HoloM3D est de permettre aux chirurgiens de visualiser des matrices 3D dans plusieurs casques HoloLens simultanément, dans le cadre de séances préopératoires. Ce rapport présente le travail effectué lors de la création du préprototype du projet HoloM3D.

Abstract

Medical imaging is an area that combines medicine and computer graphics. Indeed, after doctors have carried out examinations, they obtain data on the body of the patient. To process this, they need to view this data. We will be able to display 3D volumes of data on the screen using computer graphics. HoloLens is a new technology developed by Microsoft. This is an augmented reality helmet. It allows you to display holograms that integrate into the user's field of vision. Holograms can be arranged in space and viewed by several people at the same time. The main idea behind the HoloM3D project is to allow surgeons to visualize 3D matrices in several HoloLens helmets simultaneously during preoperative sessions. This report presents the work done during the creation of the pre-prototype of the HoloM3D project.

Remerciements

Je tiens à remercier toutes les personnes qui ont contribué à la réussite du projet et qui m'ont aidé lors de la rédaction de ce rapport.

Tout d'abord, j'adresse mes remerciements au professeur Dr Simon et son assistante Dr Kokje pour leur accueil et l'opportunité de réaliser un projet avec eux et le CHUV.

Je remercie mon professeur encadrant, M Gobron, qui est venu me voir pour me proposer ce projet et qui m'a accordé sa confiance pour le mener à bien. De plus, je le remercie de m'avoir laissé l'opportunité de pouvoir fournir une nouvelle version de ce rapport. Je remercie aussi mon expert Dr Salamin pour la lecture du rapport.

Je remercie également la He-Arc et le domaine Ingénierie, ainsi que le groupe imagerie de l'ISIC. Je tiens aussi à remercier M Christophe Bolinhas pour son aide quotidienne sur le projet, ses conseils et son soutien.

Enfin, je tiens à remercier Aurélie et Sylvie qui m'ont conseillé et relu lors de la rédaction de ce rapport.

Table des matières

1	Introduction	4
1.1	Contexte	4
1.2	Présentation du projet HoloM3D	4
1.3	But du travail de Bachelor	7
2	Etat de l'art	8
2.1	Imagerie médicale	8
2.1.1	Format des données	8
2.1.2	Volume rendering	9
2.2	HoloLens	9
2.2.1	Réalité virtuelle vs réalité augmentée	10
2.3	Remote Computing	10
2.4	3DvNCA	11
2.5	Solutions existantes	12
2.5.1	HoloM3D	12
2.5.2	Travail de Bachelor	12
3	Concepts théoriques	13
3.1	Ray tracing	13
3.2	Volume ray casting	14
3.3	API graphique	14
3.3.1	Shader	15
3.3.2	Render to texture	16
3.3.3	Face culling	16
4	Analyse	17
4.1	Volume ray tracing - Direction des rayons	17
4.2	Calcul du gradient	18
4.3	API graphique	19
4.3.1	OpenGL	19
4.3.2	DirectX	19
4.3.3	Utilisation	20
5	Implémentation	21
5.1	Structure	21
5.1.1	OpenGL	21
5.1.2	DirectX	22
5.2	Matrice 3D	23
5.2.1	OpenGL	23
5.2.2	DirectX	23
5.3	Fonction de dessin	24
5.3.1	OpenGL	24
5.3.2	DirectX	24
5.4	Shader	25
5.4.1	OpenGL	25
5.4.2	DirectX	25
5.5	Oversampling	26

5.5.1	OpenGL	26
5.5.2	DirectX	26
5.6	Modes de rendu	26
5.6.1	OpenGL	26
5.6.2	DirectX	26
5.7	Remote computing	27
6	Résultats	28
6.1	Mode Seuil	28
6.2	Mode X-Ray	30
6.3	Mode Cumulé	31
6.4	Oversampling	32
6.5	Choix de la matrice	33
6.6	Autres résultats	33
7	Problèmes rencontrés	34
7.1	Gestion du projet	34
7.2	Bibliothèques C++	34
7.3	DirectX	34
7.4	Programme de remote computing	35
8	Discussion	36
8.1	Conclusion	36
8.2	Perspective d'avenir	36

Chapitre 1

Introduction

Ce document fait office de rapport de travail de Bachelor. Il permet de comprendre le contexte de celui-ci et de décrire les éléments qui le composent. Il présente plusieurs concepts théoriques inhérents au projet et à la résolution de sa problématique, ainsi que l'implémentation qui a été effectuée.

1.1 Contexte

Ce projet s'inscrit dans le cadre du travail de Bachelor d'ingénieur en informatique en option développement logiciel et multimédia de la He-Arc. Le projet est effectué pour le CHUV dans le cadre du projet HoloM3D. Ce chapitre vise à expliquer le contexte du projet HoloM3D, et la problématique dans laquelle s'inscrit ce projet.

1.2 Présentation du projet HoloM3D

L'imagerie médicale permet de visualiser les informations extraites d'examens médicaux. Elle est utilisée par les chirurgiens pour détecter les maladies et pour mieux connaître la disposition des organes dans le corps du patient. En effet, les organes ne sont jamais disposés de la même manière, cela peut varier d'une personne à une autre. Les chirurgiens ont notamment du mal à repérer les vaisseaux sanguins comme les veines ou les artères. Avoir une meilleure connaissance du patient, les aiderait lors de l'opération et réduirait le risque d'accident. Les machines qui effectuent les examens médicaux comme une IRM ou un CT-Scan fournissent en sortie une matrice 3D de points, chacun de ces points représente une intensité. La représentation de cette matrice a une grande importance. En fonction de la méthode utilisée pour afficher cette matrice, les informations extraites par les chirurgiens peuvent varier. Il faut donc proposer une nouvelle façon de visualiser les données des examens médicaux en utilisant les nouvelles technologies à disposition. L'HoloLens est un casque de réalité augmentée développé par Microsoft. Il permet de superposer à la réalité des hologrammes 3D. Ces hologrammes sont uniquement visibles par le porteur du casque. L'utilisateur peut interagir avec ces hologrammes et même les partager avec d'autres casques HoloLens. Le projet HoloM3D vise à utiliser des casques HoloLens pour fournir une nouvelle façon de visualiser une matrice 3D aux chirurgiens lors des séances préopératoires.

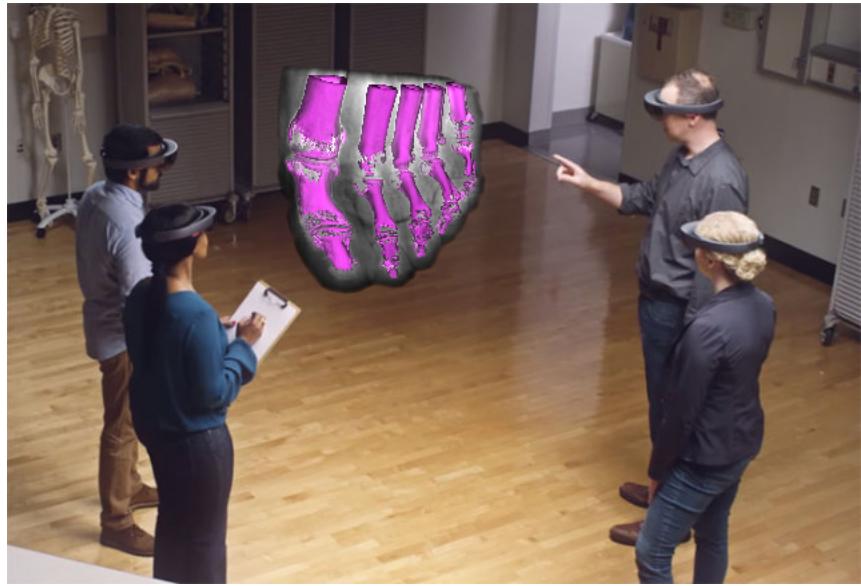


FIGURE 1.1 : Illustration représentant une vue d'esprit du projet HoloM3D

L'image ci-dessus est une vue d'esprit d'une séance entre chirurgiens utilisant le projet HoloM3D. On peut voir que chacun des spécialistes possède un casque HoloLens et que l'hologramme est partagé entre chaque casque. Ce projet est en collaboration avec le service d'oto-rhino-laryngologie et de chirurgie cervico-faciale du CHUV, la HES-SO Valais et la He-Arc. Plusieurs domaines de compétence sont donc mobilisés dans ce projet. L'équipe du Valais va effectuer un traitement sur les matrices fournies par le CHUV. Ces matrices sont composées de voxels (pixels 3D) qui ont des valeurs entre 0 et 255, et qui sont donc codés sur huit bits. L'équipe du Valais va commencer par aligner les matrices pour qu'elles soient contenues dans une matrice 256^3 . En effet les matrices résultant des examens médicaux ne sont pas toujours cubiques avec une découpe régulière. Une fois que la matrice est fusionnée, ils vont réaliser une segmentation avec des seuils sur les données pour faire apparaître les différents organes. La matrice est ensuite découpée en 256 slices de 256^2 , pour être stockée sous la forme d'une seule image de 4096^2 pixels, ou toutes les slices sont mises bout à bout. Chaque pixel doit donc contenir plusieurs informations, les données du Pet scan, celles du CT scan et le résultat de la segmentation. Les canaux RGBA (rouge-vert-bleu-alpha) sont utilisés pour coder ces informations. Chaque pixel de l'image finale contient 32 bits d'information répartie en huit bits sur chaque canal. Le schéma 1.2 ci-dessous montre la répartition des informations sur les différents canaux ainsi que le traitement effectué sur les matrices.

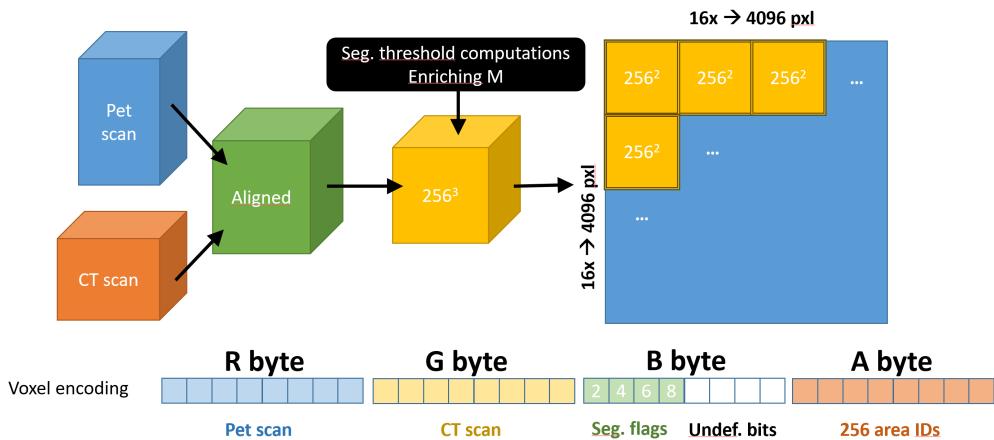


FIGURE 1.2 : Illustration de la première version du pipeline des données et de l'encodage de chaque voxel de la matrice

Il s'agit de la première version du pipeline de données et de l'encodage, c'est la version utilisée au stade actuel du projet. La version définitive contient l'ajout de l'IRM au pipeline de données, ainsi qu'une matrice de 1024³. Pour afficher une matrice 3D on va utiliser un algorithme de la famille du *volume rendering*. Il s'agit d'une catégorie d'algorithme qui permet d'afficher une projection 2D d'un volume 3D de données(voir le chapitre 2.1.2). Le *volume ray tracing* est la technique de *volume rendering* qui fournit les meilleurs résultats, mais il est couteux en temps et en ressources(voir le chapitre 3.2). Malgré cela c'est l'algorithme choisi pour afficher la matrice 3D. Un problème se pose alors, car calculer le rendu de la matrice qui contient plus de 16 millions de voxels avec cet algorithme est extrêmement couteux. Il est donc impossible d'effectuer ce rendu directement sur l'HoloLens, il n'est pas assez puissant. Pour régler ce problème , on va utiliser le *remote computing* ou calcul distant. Le rendu de la matrice va être effectué sur un serveur avec un GPU dédié. Chaque *frame* calculée sera envoyée au casque pour être affiché grâce au Wifi. Le serveur va aussi s'occuper du partage de l'hologramme entre les différents HoloLens. Pour résumer, le projet HoloM3D est composé de cinq étapes :

1. L'équipe médicale du CHUV réalise des examens médicaux sur un patient. Plusieurs types d'examens sont effectués, IRM, PET-Scan, CT-Scan. Il en ressort trois matrices 3D de données.
2. Les matrices sont traitées par l'équipe de la He-Vs. Ils vont d'abord fusionner les trois matrices en une seule, puis effectuer une segmentation 3D pour faire apparaître des zones d'intérêt.
3. Cette matrice est transmise à l'équipe de la He-Arc pour l'afficher. Un programme sur un serveur dédié va charger cette matrice et faire le rendu avec l'algorithme de *volume ray tracing*. Le rendu est envoyé à l'HoloLens avec le Wifi grâce au *remote computing*.
4. HoloLens affiche le rendu reçu depuis le serveur et envoie sa position dans l'espace au serveur. Cette opération est réalisée au minimum 60 fois par seconde.
5. Les chirurgiens utilisent les casques HoloLens durant les séances préopératoires, dans le but d'avoir une meilleure vision des données du patient et de pouvoir lui fournir un *feedback*.

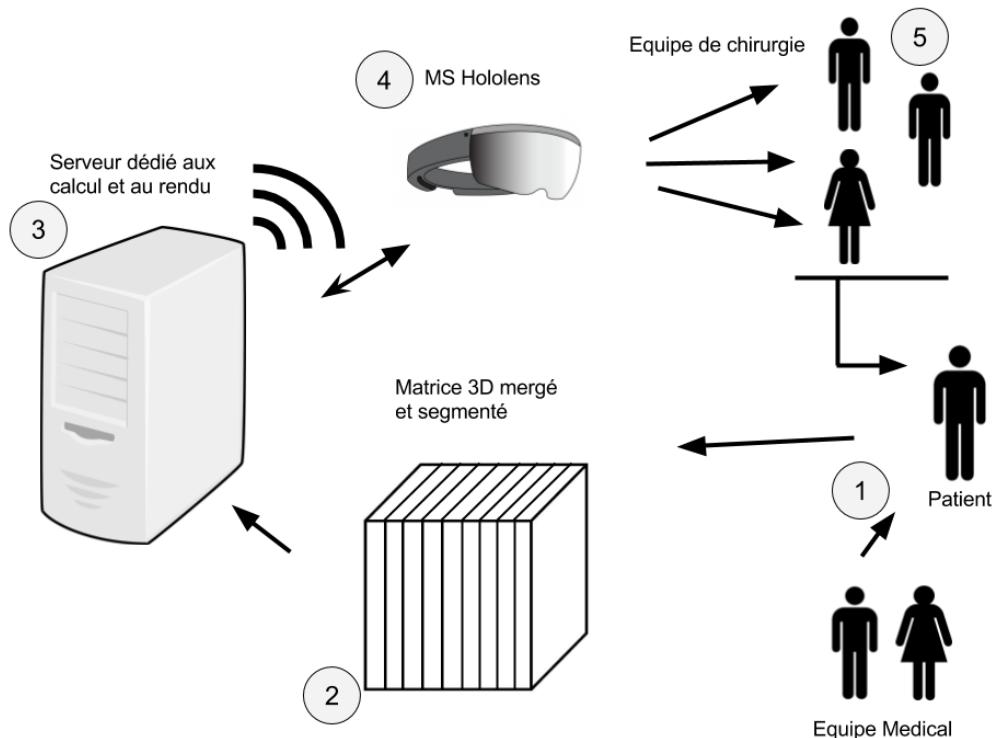


FIGURE 1.3 : Illustration résumant les cinq étapes du projet HoloM3D

1.3 But du travail de Bachelor

L'équipe de la He-Arc avait deux étudiants pour réaliser le programme du serveur dédié. Un étudiant en Master et un autre en Bachelor. Le programme a donc été coupé en deux parties. Ce travail de Bachelor portait donc sur le rendu de la matrice grâce à l'algorithme de *volume ray tracing* sur le serveur dédié. L'étudiant de Master devait quant à lui s'occuper de la communication entre le serveur et les HoloLens et du partage de la matrice entre les casques grâce au *remote computing*.

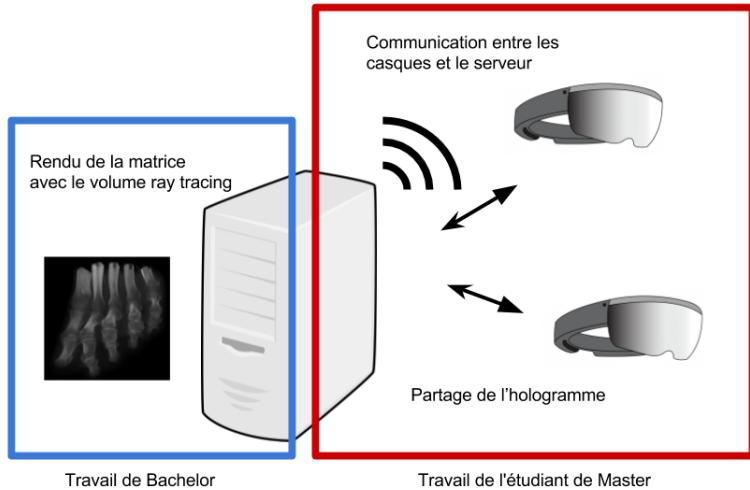


FIGURE 1.4 : Illustration résumant la répartition du travail entre ce travail de Bachelor et le travail de l'étudiant de Master

Le cahier des charges initial était concentré sur le rendu en *volume ray tracing*. Les objectifs principaux du TB étaient de réaliser un *volume ray tracing* avec un seuillage en fausses couleurs en fonction des densités tout en maintenant un taux de rafraîchissement supérieur à 60 images par seconde. Les objectifs secondaires se concentraient sur la qualité du rendu, avec notamment l'ajout de texture et de matériaux réalistes, et l'optimisation de l'algorithme. Une fois que les deux étudiants avaient fini leur projet respectif, ils auraient dû fusionner leurs travaux pour obtenir un seul programme fonctionnel. Mais plusieurs problèmes ont modifié cette planification. L'étudiant de Master a eu des problèmes personnels d'ordre majeur et il n'a pas pu fournir le travail qui lui était demandé. Ce travail de Bachelor a donc été modifié pour intégrer la partie de l'étudiant de Master. En plus de réaliser le rendu de la matrice, il a donc fallu ajouter l'intégration au programme de *remote computing* avec HoloLens.

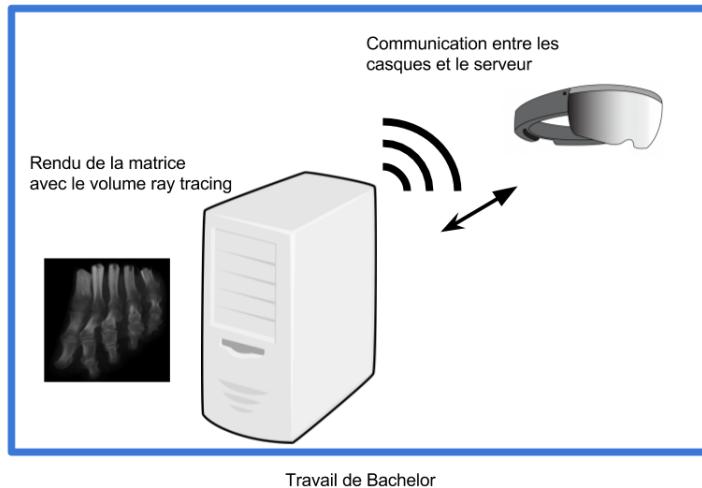


FIGURE 1.5 : Illustration résumant le travail effectué pour ce travail de Bachelor au final

Chapitre 2

Etat de l'art

Avant de commencer la conception et l'implémentation d'un projet, il est utile d'effectuer un état de l'art des technologies disponibles sur le marché. Étant donné que le projet HoloM3D avait déjà été conceptualisé au début de ce travail de Bachelor. Il a donc fallu apprendre les technologies et concepts présents dans le projet. Ce chapitre détaille donc cet apprentissage ainsi que les recherches effectuées pour trouver si des solutions existantes étaient disponibles.

2.1 Imagerie médicale

L'imagerie médicale a pour but de représenter visuellement les informations médicales. Cela peut être représenté par une image 2D ou 3D et même une animation montrant l'évolution au cours du temps. La représentation qui sera faite dépend essentiellement des données que l'on utilise et du rendu que l'on souhaite obtenir. Il existe de nombreux moyens d'acquisition :

- Champs magnétiques : IRM
- Radioactivité : TEP ou PET
- Rayons X : Radiographie, scanner ou CT-Scan
- Ultrasons : Echographie

Cette liste est non exhaustive, car il existe de nombreuses variantes (il s'agit des termes les plus connus). On comprend aisément qu'avec toutes ces méthodes d'acquisition différentes, utiliser une seule méthode pour afficher ces données est impossible. Il existe donc différentes méthodes pour stocker et afficher ces données.

2.1.1 Format des données

Les machines qui effectuent les IRM ou les scanners réalisent généralement des acquisitions volumétriques. À la place d'avoir des échantillons comme des pixels sur une image bidimensionnelle, l'échantillonnage volumique ajoute une troisième dimension. Les pixels acquièrent une épaisseur et deviennent alors des voxels. On peut donc considérer ce volume comme une matrice 3D et associer des coordonnées (x , y , z) à chaque position. Lorsque l'on regarde une image 2D, l'échantillonnage est toujours régulier et uniforme, c'est-à-dire que les pixels de l'image sont toujours carrés et de même taille. Cette question se pose aussi pour les voxels d'un volume. Si l'échantillonnage est régulier on le qualifiera d'*isotrope*, dans le cas contraire on le qualifiera d'*anisotrope*. Ces notions sont directement liées à la technique d'acquisitions utilisée et à l'information que le médecin veut recueillir. L'*isotropie* d'un volume a aussi de l'importance pour l'affichage de celui-ci.

2.1.2 Volume rendering

Il existe deux approches pour afficher un volume/matrice, soit une approche volumique soit une approche surfacique. Cette dernière s'appuie sur la surface des organes pour créer un maillage de points. Cette méthode est compliquée, car elle demande beaucoup de prétraitements sur la matrice pour déterminer les surfaces que l'on veut afficher. L'approche volumique utilise toutes les données de la matrice, mais ces méthodes ont un coût plus élevé, car il faut stocker et traiter tous les voxels. La puissance des ordinateurs actuels nous permet d'afficher des matrices avec ces méthodes en temps réel, mais cela n'a pas toujours été le cas. Il existe donc plusieurs méthodes qui sont adaptées à plusieurs situations, elles sont détaillées dans un article complet sur l'imagerie médicale[3]. Celle qui nous intéresse s'appelle le *volume ray casting*, on va appliquer un lancer de rayon sur la matrice. Cette méthode permet la gestion de la densité par la transparence et fournit les meilleurs résultats.

2.2 HoloLens

L'HoloLens est une nouvelle technologie développée par Microsoft. Il s'agit d'une paire de lunettes de réalité augmentée permettant d'afficher des hologrammes qui s'intègrent dans le champ de vision de l'utilisateur. Il est disponible depuis le premier trimestre de 2016 au prix de 3000 \$.



FIGURE 2.1 : Image de l'HoloLens vue de côté

Le casque est ajustable grâce à un arceau réglable, il s'adapte à tous les utilisateurs. Sur le côté se trouvent deux haut-parleurs rouges qui fournissent un son spatialisé. Pour commander le casque, on peut utiliser des gestes ou la voix. À l'avant du casque se trouvent le centre inertiel (IMU) et une caméra de profondeur qui permettent au casque de connaître sa propre position dans l'espace. Pour traiter toutes ces données, l'HoloLens intègre un véritable ordinateur avec une carte graphique. En plus de cela, Microsoft a développé un HPU (Holographic Processing Unit). Il permet de gérer les données qui viennent des capteurs, il s'occupe de la cartographie spatiale, de la reconnaissance des gestes et des voix. Le casque possède une autonomie de deux heures.

Même si la fiche technique du casque est disponible sur internet [22], il est difficile de calculer sa puissance et de la comparer à un ordinateur classique. L'HoloLens reste un ordinateur portable et autonome, il ne doit pas gérer la même chose. Quand un PC doit rendre une image à l'écran, il doit calculer tous les pixels de l'écran, alors que l'HoloLens doit juste afficher les hologrammes présents dans le champ de vision. En plus de cela, Microsoft est avare en information concernant la puissance de son casque. Cet article[19] détaille les limitations du casque qui ne sont pas explicites dans la documentation de l'HoloLens.

2.2.1 Réalité virtuelle vs réalité augmentée

Le casque HoloLens fait partie de la catégorie des appareils de réalité augmentée, il ne faut donc pas confondre avec la réalité virtuelle. Ces deux termes sont des fondamentaux, il est donc important de les définir.

La réalité augmentée est la superposition de la réalité et d'éléments (sons, images 2D, 3D, vidéos, etc.) calculés par un système informatique en temps réel. [30]

La réalité virtuelle renvoie typiquement à une technologie informatique qui simule la présence physique d'un utilisateur dans un environnement artificiellement généré par des logiciels, environnement avec lequel l'utilisateur peut interagir.[31]

Comme le montrent ces deux définitions, l'une ne fait qu'augmenter la réalité alors que l'autre plonge l'utilisateur dans un monde entièrement virtuel. Utiliser la réalité virtuelle pour le projet HoloM3D aurait pu être une bonne idée. Mais le projet vise à être utilisé par plusieurs chirurgiens dans une même pièce. S'ils utilisent un casque de VR, ils seront immergés dans le monde virtuel et ils ne se verront plus dans la vie réelle. Cela pose des problèmes de communication lors des réunions. De plus, la réalité virtuelle a un côté invasif que la réalité augmentée n'a pas.

2.3 Remote Computing

Pour résoudre le problème de puissance du casque, on va utiliser le *remote computing* ou calcul distant. Microsoft a directement mis en place un système pour réaliser ce calcul distant entre le casque et un serveur. Il est composé de deux applications, une sur le serveur et une sur l'HoloLens. Sur le *store* de Microsoft disponible sur l'HoloLens, on peut trouver une application appelée Holographic Remoting Player[15]. Elle est uniquement disponible avec l'HoloLens, et elle est gratuite. C'est cette application qui va s'occuper de réceptionner le flux d'image qui est *streamer* depuis le serveur. Quand on la lance, elle nous indique l'adresse IP du casque, pour pouvoir la rentrer dans l'application sur le serveur. Cette application va aussi se charger de transmettre la position du casque, ainsi que d'éventuelles actions effectuées par l'utilisateur. Sur le serveur, Microsoft propose deux solutions pour effectuer le *remote computing*. Soit on utilise le *plug-in* Unity soit on utilise un projet C++. Pour développer une application native sur HoloLens, on utilise le moteur Unity. Microsoft et Unity ont travaillé ensemble pour fournir un environnement de développement simplifié[27]. Le *remote computing* fait partie des fonctionnalités disponibles dans ce *plug-in*. L'autre solution provient du *MixedRealityCompanionKit*[16], il s'agit d'applications qui ne sont pas destinées à être exécutées sur l'HoloLens, mais qui sont associées à lui pour certaines expériences de réalité augmentée, et le *remote computing* en fait partie.

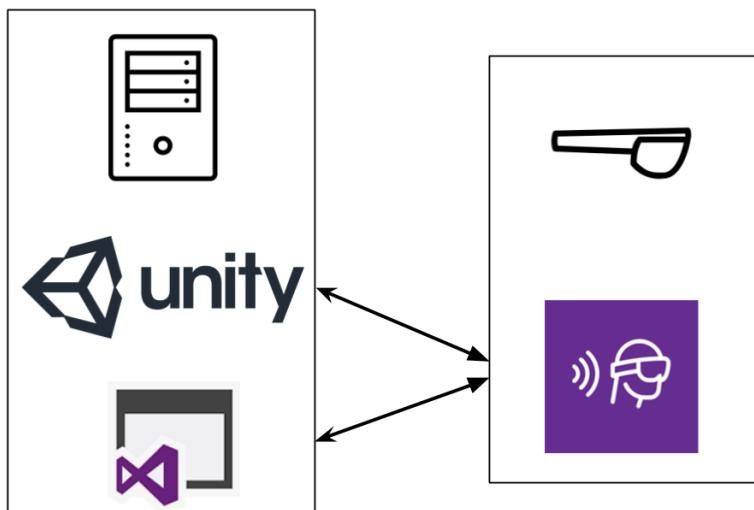


FIGURE 2.2 : Schéma représentant les différentes solutions fournies par Microsoft pour effectuer du *remote computing* entre le serveur et l'HoloLens

L'application de *remote computing* est un code d'exemple qui effectue les quatre fonctionnalités essentielles au fonctionnement du *remote computing*.

- Établir une connexion avec l'HoloLens
- Recevoir le flux de données des inputs de l'HoloLens
- Rendre le contenu dans une *virtual holographic view*.
- Envoyer les *frames* calculées vers HoloLens en temps réel

Ce programme d'exemple permet de visualiser un petit cube de couleur qui tourne. On peut déplacer le cube avec les commandes gestuelles et changer sa couleur avec les commandes vocales. La grande majorité des composants et des classes présents dans ce programme sont nouveaux et ont été développé exprès pour cela. Pour faire le rendu 3D du cube, l'API 3D de Microsoft DirectX 11 est utilisée. Ce programme d'exemple devait servir de base pour le travail de l'étudiant de Master afin de comprendre le fonctionnement du *remote computing*. Vu que l'étudiant de Master n'a pas fourni d'application, ce programme d'exemple a donc été utilisé pour effectuer le prototype du projet.

2.4 3DvNCA

Il s'agit un programme réalisé lors de la publication - GPGPU Computation and Visualization of Three-dimensional Cellular Automata - de M Gobron ainsi que Mme Çöltekin, M Bonafos, M Thalmann [10]. Le but de ce programme est de calculer et visualiser un automate cellulaire 3D grâce à un GPU. Pour afficher cette matrice 3D, ils ont utilisé la méthode du *volume ray tracing*, et c'est en cela que ce programme est intéressant. Ce programme a fourni les bases de compréhension pour l'algorithme de *volume ray tracing*. Mais au vu de l'ancienneté du code et de la version d'OpenGL utilisée, peu de code a été réutilisé. Seulement quelques parties du code des *shader* ont été réutilisées(calcul de la normale et multisampling).

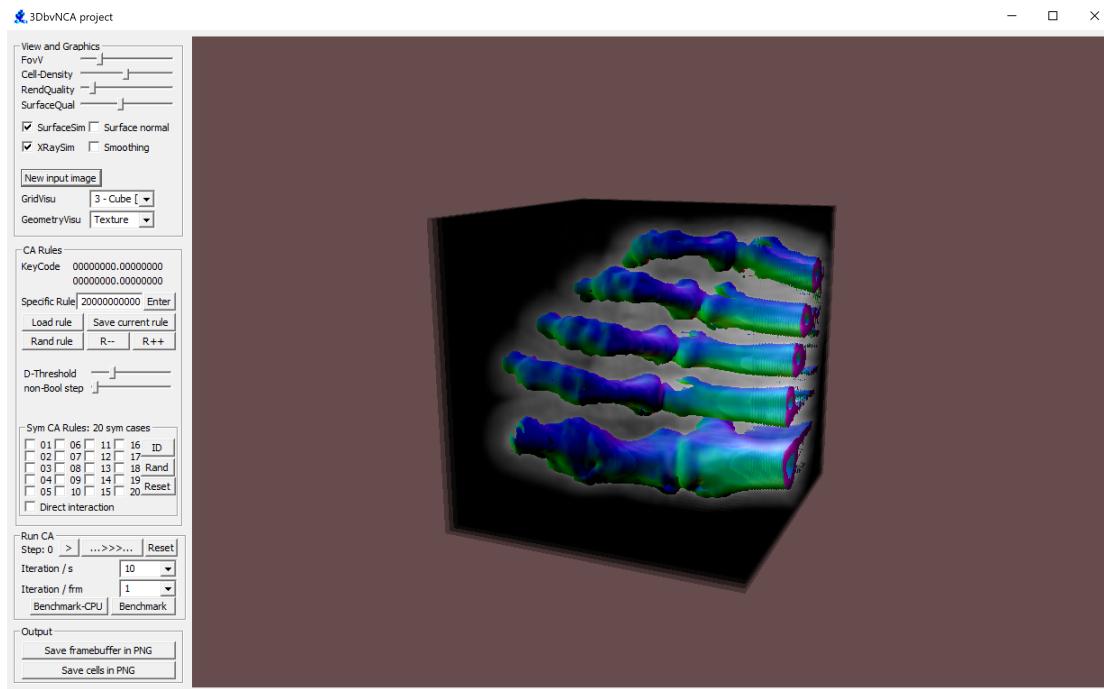


FIGURE 2.3 : Capture du programme 3DvNCA [10]

2.5 Solutions existantes

Cette section va détailler les recherches qui ont été effectuées afin de trouver des solutions existantes. Elles concernent le projet HoloM3D ainsi que le travail de Bachelor.

2.5.1 HoloM3D

Après avoir passé plusieurs heures à rechercher sur internet si des solutions existaient déjà sur le marché. Seulement deux vidéos se sont portées dignes d'intérêt, elles se trouvent sur la chaîne YouTube d'Henrique Debarba [5]. Peu d'informations sont données concernant la méthode utilisée pour effectuer le rendu dans le casque HoloLens. Il s'agit des seules sources qui font état d'un lancer de rayon dans un casque HoloLens. Sur le Windows dev center[2], on peut trouver un article qui liste les solutions à adopter pour réaliser un *volume rendering* sur l'HoloLens. Les solutions que Microsoft propose sont simples, diminuer la résolution du rendu pour avoir moins de calculs ou afficher le volume *slice* par *slice*. Ces solutions sont efficaces, mais ne permettent pas d'avoir un rendu de haute qualité sur HoloLens. Ces deux solutions ne sont pas suffisantes, car elles ne montrent pas des résultats de grande qualité et elles ne prennent pas en compte une fonctionnalité importante du projet, le partage de l'hologramme entre les casques.

2.5.2 Travail de Bachelor

Concernant les recherches effectuées pour le travail de Bachelor en lui même, elles portaient sur le rendu en *volume ray tracing*. Cette méthode existe depuis plus de 20 ans, il existe donc une pléthore de publications sur ce sujet, mais paradoxalement peu d'exemples de code. La première source de code a été le programme 3DvNCA fourni par M. Gobron, mais comme expliqué plus tôt, il est désuet dû à sa version d'OpenGL. Donc peu de choses ont pu en être tirées. Plusieurs articles sur internet ont permis la compréhension de certains points techniques concernant la texture 3D [1] ainsi que sur la technique pour effectuer le lancer de rayon [11]. Mais la principale source d'aide est venue d'un dépôt sur GitHub[26] qui est une implémentation d'un *volume rendering* à l'aide de l'algorithme *ray tracing* réalisée en OpenGL. C'est donc grâce aux sources mentionnées ci-dessus que le projet a pu être réalisé.

Chapitre 3

Concepts théoriques

Ce chapitre va détailler certains concepts théoriques nécessaires à la compréhension des phases d'analyse et d'implémentation. Les explications de ce chapitre vont rester théoriques et ne vont pas être techniques. Elles expliquent le fonctionnement global de plusieurs algorithmes, ainsi que le fonctionnement de certaines fonctionnalités des API 3D.

3.1 Ray tracing

Dans la nature, une source de lumière émet un nombre incalculable de rayons qui rebondissent sur des objets jusqu'à atteindre la surface de l'œil. Simuler ces rayons est quasiment impossible dû au nombre de rayons et au nombre de reflets que la lumière peut faire avant d'atteindre l'œil. Le *raytracing* ou lancer de rayon est une technique d'infographie qui consiste à simuler le parcours inverse de la lumière. On va calculer la trajectoire de la lumière depuis la caméra vers les objets, puis vers la lumière. Cette technique permet de simuler des effets physiques comme la réfraction et la réflexion, mais aussi des phénomènes optiques plus complexes tels que les reflets caustiques, l'illumination globale ou encore la dispersion lumineuse. Le *raytracing* permet d'obtenir des images d'une grande qualité qui sont presque photoréalistes, mais il peut requérir un temps de calcul très important, en fonction de la complexité de la scène 3D.

Pour chaque pixel de l'image finale, on tire un rayon dans la scène. La direction de ce rayon est obtenue en traçant une droite depuis la caméra jusqu'au centre de ce pixel (rayon rouge sur le schéma 3.1 ci-dessous). Une fois que la direction du rayon a été définie, l'algorithme va itérer sur tous les objets de la scène pour voir si elle croise l'un d'entre eux. Si le rayon intercepte plus d'un objet, on va sélectionner l'objet dont le point d'intersection est le plus proche de la caméra. Une fois que le point d'intersection a été déterminé, on va lancer un rayon depuis ce point en direction de la source de lumière (rayon gris sur le schéma 3.1 ci-dessous). Si le rayon arrive directement à la source de lumière, le pixel sera éclairé. Si le rayon croise un objet, le pixel sera ombré, car un objet obstrue la source de lumière. On va répéter cette opération pour tous les pixels de l'image.

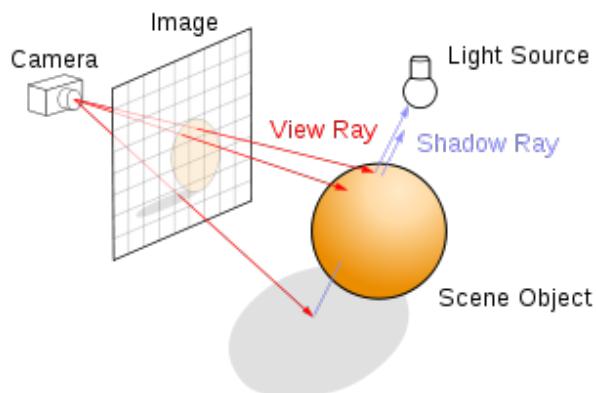


FIGURE 3.1 : Schéma illustrant le fonctionnement du *raytracing* [32]

3.2 Volume ray casting

La technique de *volume ray tracing* ou *volume ray casting* permet de visualiser une image 2D d'une matrice ou d'un volume 3D de données. Il ne faut pas la confondre avec le *raytracing* classique. Dans cette variante, le rayon ne s'arrête pas à la surface du volume, mais le traverse en récupérant des échantillons le long du rayon. Cette technique ne génère pas de rayon secondaire. L'algorithme est composé de quatre étapes détaillées sur le schéma 3.2 ci-dessous.

- **Ray tracing(1)** Pour chaque pixel de l'image, on lance un rayon à travers le volume, de la même manière que pour le *raytracing*. En général, on considère que le volume est un cube pour pouvoir arrêter le rayon quand il dépasse du cube.
- **Sampling(2)** Lorsque l'on fait avancer le rayon dans le volume, on va établir des points d'échantillonage. Ces points sont pris de manière régulière, cet intervalle influe sur la qualité du rendu final. Généralement ces points ne sont pas forcément alignés avec les voxels du volume. Il faut donc faire une interpolation pour récupérer la valeur de l'échantillon.
- **Shading(3)** Pour chaque point échantillonné, on va utiliser un seuil ou une fonction de transfert pour calculer la couleur du pixel. On va aussi calculer le gradient qui représente l'orientation de la surface dans le volume. On va déterminer la couleur finale du point en calculant l'éclairage diffus grâce au gradient.
- **Compositing(4)** Après avoir calculée la couleur de tous les échantillons, on va calculer la valeur finale du pixel en additionnant les différentes couleurs des échantillons. En fonction du rendu que l'on veut obtenir, on va changer la manière dont on va additionner les couleurs.

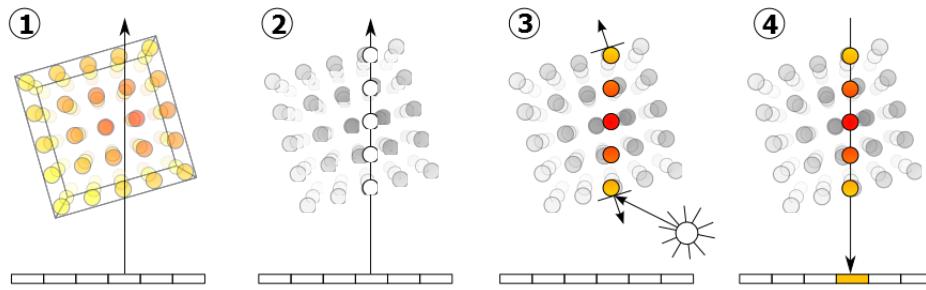


FIGURE 3.2 : Schéma illustrant les quatre étapes du volume ray tracing (1) Ray casting (2) Sampling (3) Shading (4) Compositing [33]

3.3 API graphique

Pour commencer, qu'est-ce qu'une API (*Application Programming Interface*) ou interface de programmation. Une API est un ensemble normalisé de classes, de méthodes ou de fonctions permettant aux développeurs de faire facilement le lien entre deux éléments. Généralement, une API existe sous la forme d'une bibliothèque logicielle. Elle fournit une façade à un problème en faisant abstraction de son fonctionnement.

Une API graphique est une interface de programmation spécifique, qui permet au développeur de créer des rendus 2D ou 3D d'un objet 3D. L'API va utiliser l'accélération matérielle sur GPU pour effectuer ses calculs, elle fait l'intermédiaire entre le programme et le *driver* de la carte graphique.

L'API permet au développeur de déclarer la géométrie d'une scène 3D sous forme de points et de polygones avec des textures. L'API effectue ensuite des calculs de projection en vue de déterminer l'image à l'écran, en tenant compte des distances, de la lumière, des ombres, de la transparence.

3.3.1 Shader

Les *shaders* sont de petits programmes qui sont utilisés pour paramétriser le processus de rendu réalisé par la carte graphique. Il permet de décrire le comportement de chaque étape du pipeline graphique. On va pouvoir calculer la diffusion de la lumière, appliquer des textures sur un objet, réaliser de l'ombrage ainsi que de nombreux effets de post-traitement sur l'image finale.

Les *shaders* sont écrits dans un langage spécifique qui a une syntaxe proche du C. Le code écrit doit être compilé par l'API avant d'être envoyé à la carte graphique. À chaque fois qu'un *draw call* sera exécuté, c'est-à-dire que l'on ordonne à la carte graphique de dessiner les points qui sont stockés dans sa mémoire. Les *shaders* seront exécutés pour calculer la position des points et la couleur du pixel. Il existe principalement deux types de *shaders* :

- Le Vertex Shader (VS) doit calculer la projection des coordonnées des sommets des objets depuis l'espace 3D dans l'espace 2D de l'écran. Pour chaque sommet, un *vertex shader* lui est associé.
- Le Pixel ou Fragment Shader (PS) doit calculer la couleur de chaque pixel individuellement. Il prend en entrée les données de chaque pixel de l'image comme la position, les coordonnées de texture, la couleur et renvoie la couleur de celui-ci.

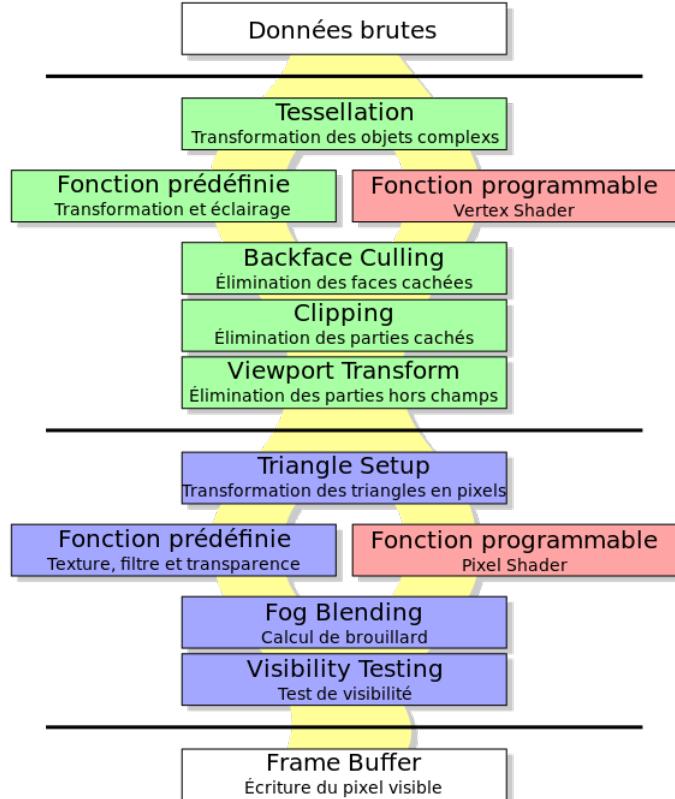


FIGURE 3.3 : Pipeline avec indication du lieu d'intervention des pixels et des vertex *shaders* (en rouge).
[29]

3.3.2 Render to texture

Le *render to texture* ou rendu de texture est une technique avancée des API graphiques. Quand on exécute un *drawCall*, le résultat est stocké dans le *back buffer* avant d'être présenté à l'écran. Le *render to texture* nous permet d'effectuer un *drawCall* sans l'afficher à l'écran. Le rendu sera alors stocké dans une texture. Cette texture est donnée en tant que ressource au prochain *drawCall* et elle pourra être utilisée dans le pixel *shader* comme s'il s'agissait de la texture d'un modèle.

3.3.3 Face culling

Si on visualise un cube en trois dimensions, on peut voir au maximum trois faces en même temps. Pourquoi fournir un effort pour calculer la couleur des trois faces cachées si on ne les voit pas. Le *face culling* vérifie si les faces de l'objet sont à l'avant ou à l'arrière afin de supprimer celles qui ne nous intéressent pas. En effet, on peut choisir si on veut voir les faces de derrière ou de devant en l'indiquant à l'API graphique avec une commande qui change le *cull mode*.

Par exemple dans la figure 3.4 ci dessous le cube de gauche a été rendu avec le *cull mode* en *backface*, les faces de derrière ont donc été supprimées. Alors que le cube de droite a été rendu avec le *cull mode* en *frontface*, les faces de devant ont donc été supprimées.

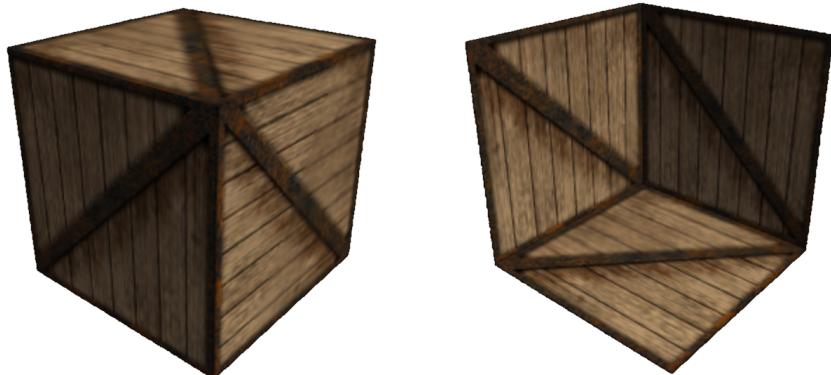


FIGURE 3.4 : Capture d'un cube texturé rendu avec les deux modes de *face culling*, à gauche en mode *backface* et à droite en mode *frontface*

Chapitre 4

Analyse

Ce chapitre va détailler la partie d'analyse de ce projet. La phase d'analyse comporte plusieurs thèmes qui ont demandé des recherches et des tests afin d'être implémentés. Cette partie va donc parler des concepts techniques vus au chapitre précédent. Ce chapitre présente la méthode qui est utilisée pour effectuer le *volume ray tracing*, ainsi que la méthode utilisée pour calculer le gradient. Une section est aussi consacrée aux API graphiques, on y trouve des explications sur les choix qui ont été faits les concernant.

4.1 Volume ray tracing - Direction des rayons

A la base, pour réaliser le *volume ray tracing* l'algorithme du programme 3DvNCA devait être utilisé. Mais après quelques tests et recherches, une méthode plus optimisée et plus simple a été trouvée. Cette section détaille son fonctionnement. Comme vu dans le chapitre précédent, le *volume ray tracing* est composé de quatre étapes. La méthode qui va être présentée concerne la première étape : le *ray tracing*. Elle n'a pas de nom officiel, mais elle est explicitée dans plusieurs articles trouvés sur internet [26] [11].

Cette méthode est basée sur le fait que la couleur du pixel du cube est aussi sa position dans l'espace. En effet les huit points qui composent le cube ont une position comprise entre $(0,0,0)$ et $(1,1,1)$ comme sur le premier cube de l'image 4.1. Quand on va afficher ce cube à l'écran, tous les pixels de couleurs qui composent le cube représentent aussi leur position dans l'espace. Ce cube représente la matrice dans l'espace.

La première chose à faire pour réaliser le *ray tracing* est de calculer la direction du rayon. Pour ce faire, on va utiliser les deux concepts expliqués au chapitre précédent, le *render to texture* et le *face culling*. On va commencer par rendre le cube en définissant le *cull mode* en *backface* et stocker ce rendu dans une texture grâce au *render to texture*. Puis, on va rendre le même cube en ne changeant que le *cullmode* en *frontface*. Ce rendu sera aussi placé dans une texture. Une fois que les deux textures (*backface* et *frontface*) sont rendues, on peut les utiliser pour calculer la direction du rayon dans le volume.

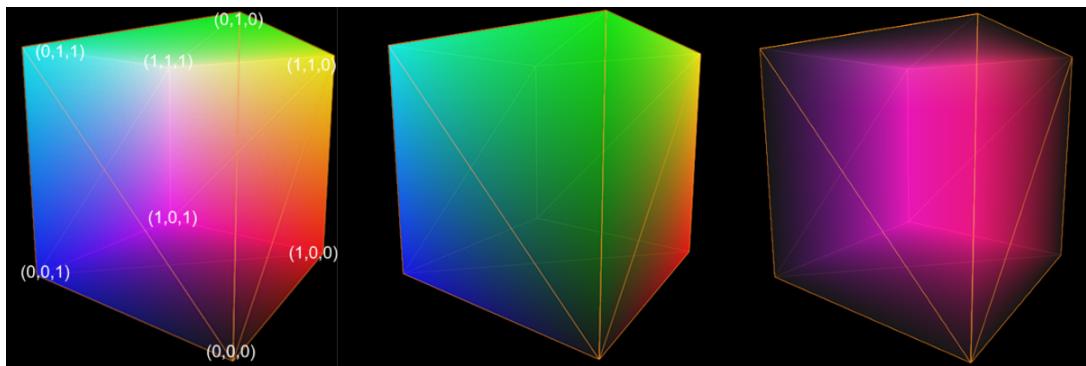


FIGURE 4.1 : Image représentant les trois textures calculées (1) backface (2) frontface (3) direction [11]

Pour calculer la direction, on va simplement soustraire les deux textures que l'on vient de calculer. En effet, vu qu'elles contiennent les positions de chaque pixel dans l'espace (donné par la couleur du pixel). On peut soustraire les deux positions pour déterminer le vecteur directeur entre ces deux points. Comme dans le schéma 4.2 où la texture bleue représente la *frontface texture* et la verte, le *backface texture*. La soustraction de deux points nous donne bien la direction du vecteur en rouge sur le schéma 4.2. La texture qui résulte de ce calcul 4.1 contient la direction de chaque rayon pour chaque pixel du cube.

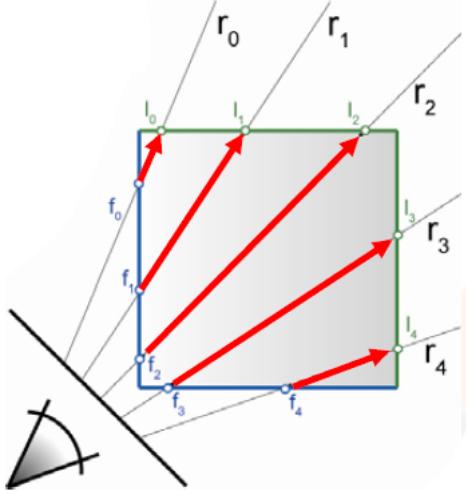


FIGURE 4.2 : Schéma illustrant le calcul de la direction des rayons[11]

Maintenant que l'on a la direction de chaque rayon, on va pouvoir les faire avancer dans la matrice. On va normaliser le vecteur directeur et le multiplier par un coefficient pour déplacer le point d'échantillonnage le long du rayon. Ce coefficient va définir la précision du rendu que l'on va obtenir. Pour arrêter le rayon, on va calculer la longueur du vecteur directeur et celle du vecteur directeur normalisé avant de commencer à se déplacer sur le rayon. À chaque nouveau point, on va additionner le vecteur directeur normalisé pour déterminer la nouvelle position de l'échantillon, et additionner la longueur du vecteur directeur normalisé. Quand celle-ci dépasse la longueur du vecteur directeur, c'est que le rayon est sorti du cube. On peut donc arrêter l'algorithme de *volume ray tracing*.

4.2 Calcul du gradient

Le calcul du gradient intervient dans la troisième étape du *volume ray tracing* lors que l'on doit calculer la couleur du pixel.

Pour déterminer la couleur du pixel, on va réaliser un *shading* simple (lumière ambiante et diffuse). La valeur de la couleur ambiante est une constante du *pixel shader*, mais pour calculer la couleur diffuse on a besoin de la normale à la surface. C'est là que le calcul du gradient intervient, en effet on ne possède pas une surface, mais une matrice 3D de valeur, que l'on peut considérer comme un champ scalaire. Calculer la normale s'apparente alors à calculer le gradient du point d'échantillonnage.

Pour calculer ce gradient des recherches ont été faites afin de trouver la méthode fournissant le meilleur compromis entre le temps et la qualité du rendu. Dans un article trouvé sur internet, l'auteur détaille les différentes méthodes existantes pour calculer le gradient et compare les résultats qu'il a obtenus[20]. Les quatre opérateurs présentés sont :

- l'opérateur de différence intermédiaire
- l'opérateur de différence central
- l'opérateur de Zucker-Hummel
- l'opérateur Sobel 3D

Les deux premiers opérateurs ont été testés, mais ils ne fournissaient pas un résultat suffisant. Ces deux opérateurs sont simples, on regarde uniquement les six voxels voisins du voxel courant. L'opérateur de Sobel est celui qui fournit le calcul le plus correct et qui filtre aussi le bruit de la matrice. Mais mettre en place un tel opérateur a posé quelques problèmes, son implémentation n'a pas pu être réalisée. Le choix a donc été d'implémenter l'opérateur de Zucker-Hummel qui regarde les 27 voisins du voxel courant. Il permet un bon compromis entre performance et complexité. Cet opérateur était aussi utilisé dans le programme 3DvNCA, il a donc été réutilisé.

4.3 API graphique

Cette section va détailler les deux API graphiques qui ont été utilisées, OpenGL et DirectX. Après avoir présenté chacune de ces deux API, on verra pourquoi elles ont été utilisées dans la réalisation de ce projet.

4.3.1 OpenGL

Silicon Graphics Inc. a commencé à développer l'API OpenGL en 1991. Elle est maintenant gérée par le groupe Khronos. La version actuelle d'OpenGL est la 4.6 qui est sortie en juillet 2017. Elle est utilisée dans de nombreux domaines comme pour les applications de CAO, de visualisation scientifique et médicale ainsi que pour les jeux vidéo.

OpenGL est une spécification abstraite, c'est-à-dire qu'elle est définie comme un ensemble de fonctions qui peuvent être appelées par le programme. OpenGL est indépendante du langage, elle peut être implémentée pour différents langages comme pour le WebGL avec JavaScript. Dans la majorité des cas, on va utiliser l'implémentation en langage C. En plus de cela, OpenGL est multi-plateforme. En effet, la spécification ne précise pas comment gérer le contexte OpenGL. Cette tâche est laissée au programme et au système d'exploitation. OpenGL est uniquement concentré sur le rendu. En plus des fonctionnalités de base fournies par OpenGL, les constructeurs de cartes graphiques peuvent fournir des fonctionnalités supplémentaires sous forme d'extensions. Les extensions peuvent introduire de nouvelles fonctions et de nouvelles constantes et peuvent supprimer des restrictions sur les fonctions OpenGL existantes. Ces extensions augmentent considérablement la flexibilité d'OpenGL. Le groupe Khronos s'occupe d'enregistrer et de valider les extensions proposées par les constructeurs. Elles sont alors identifiées et ajoutées à la liste des extensions. Le langage utilisé pour écrire un shader en OpenGL s'appelle GLSL.

4.3.2 DirectX

DirectX est développé par Microsoft pour ses plateformes (Xbox, Windows) depuis Windows 95, elle est dédiée à la création de jeux et d'applications ultra optimisées qui demandent de hautes performances. À l'origine DirectX est le nom qui abrège une collection de différentes API comme Direct3D, DirectDraw, DirectMusic, DirectPlay, DirectInput. Le nom DirectX est souvent utilisé pour parler de Direct3D qui est l'API qui sert à faire des rendus 3D. Cette confusion vient du fait que Direct3D est le composant le plus utilisé de DirectX. DirectX est implémenté en C et disponible uniquement sur Windows. Le langage utilisé pour écrire les *shaders* s'appelle le HLSL.



FIGURE 4.3 : Logo des deux API, OpenGL et DirectX11

4.3.3 Utilisation

Pour réaliser la première version du rendu de la matrice, il a été décidé d'utiliser OpenGL. Cette API était connue par l'équipe de la He-Arc et c'était aussi l'API utilisée par M Gobron dans le programme 3DvNCA. Il était donc plus simple de commencer à utiliser OpenGL. Il a fallu apprendre OpenGL. En effet, seule la spécification d'OpenGL pour le web, WebGL, a été vue en cours. Les concepts sont les mêmes entre les deux versions, mais la syntaxe change entre le C++ et le JavaScript. Beaucoup de tutoriels sont disponibles sur internet, le plus complet est *Learn OpenGL* [4]. Il fournit les bases pour comprendre OpenGL ainsi que des concepts très avancés en OpenGL et en infographie.

Pour réaliser la première version du rendu, la version 4.3 d'OpenGL a été utilisée. Ce choix a été influencé par l'utilisation de *debugger* graphique uniquement compatible avec les dernières versions d'OpenGL. En soi, le choix de la version n'avait que peu d'importance, car depuis la version 3.0 d'OpenGL le cœur de l'API n'a pas changé. Les versions supérieures ne font qu'ajouter des fonctionnalités avancées.

Le programme de rendu de la matrice a donc été réalisé une première fois en OpenGL 4.3. Mais un problème est alors apparu. Il s'avérait difficile voir impossible de partager le rendu du programme OpenGL avec le programme de *remote computing* qui lui fonctionnait avec DirectX. Des solutions avaient été trouvées, mais elles étaient toutes compliquées à mettre en place. La décision a été prise de passer le rendu d'OpenGL à DirectX pour simplifier les choses. C'est aussi à ce moment que l'étudiant de Master a rendu son travail et qui n'était malheureusement pas fonctionnel. La décision de porter sous DirectX le rendu était donc la solution la plus sûre afin d'obtenir un prototype rapidement. Le risque principal était que personne ne connaissait DirectX et qu'il a fallu apprendre à utiliser cette API.

En effet, personne au sein de l'équipe de la He-Arc n'avait d'expérience avec DirectX. Il a donc fallu rechercher des livres et des tutoriels. Il existe peu d'ouvrage complet disponible sur internet pour apprendre DirectX tout seul. Deux ouvrages ont été utilisés pour apprendre DirectX, le premier est un livre *Introduction to 3D Game Programming with Direct3D 11.0* par Frank D. Luna [12] et l'autre un tutoriel *DirectX 11 Tutorial* par rastertek[21]. Le premier a surtout servi pour comprendre les concepts théoriques de l'API, alors que l'autre a servi comme exemple pour le code. Ces deux ouvrages ont permis d'avoir deux visions différentes de l'API et de voir plusieurs manières de réaliser une même action. Le choix de la version a été imposé par les deux tutoriels qui utilisent la version 11 de DirectX et par le fait que l'HoloLens ne supporte pas DirectX 12. DirectX 11 reste la version la plus utilisée de DirectX malgré le fait que DirectX 12 soit sortie depuis quelques années. Le programme de rendu de la matrice a donc été réalisé une deuxième fois en DirectX 11. Ce programme n'a pas d'intérêt en soi, il a juste servi de base de compréhension au fonctionnement de DirectX 11 dans le but d'intégrer le rendu dans le programme de *remote computing*.

Chapitre 5

Implémentation

Ce chapitre va détailler l'implémentation réalisée lors de ce travail de Bachelor. Chaque section est divisée en deux sous-sections pour les deux API utilisées. Chaque thème abordé sera vu du point de vue des deux API.

5.1 Structure

Les deux programmes ont deux structures complètement différentes, cela est dû au contexte différent pour chaque programme. La seule chose qu'il partage et qu'ils sont écrits tous les deux en C++ sur Windows avec Visual Studio 2017.

5.1.1 OpenGL

Au début du projet, le programme OpenGL devait être intégré au programme de l'étudiant de Master. Il a donc été décidé d'avoir une structure simple pour simplifier le travail à effectuer. Le programme OpenGL à une structure simple puisqu'il est écrit dans un style C, c'est-à-dire qu'il n'y a pas de classe, seulement des fonctions qui s'appellent entre elles. Tout le code est écrit dans un seul fichier *main*. Deux classes externes sont utilisées pour gérer facilement la caméra et les shaders, ces classes proviennent du tutoriel *LearnOpenGL*. En plus de cela le programme utilise plusieurs bibliothèques C++.

- GLEW est une bibliothèque qui détermine quelles extensions OpenGL sont prises en charge sur la plate-forme cible[7].
- GLFW est une API pour créer des fenêtres, gérer des contextes et des surfaces, recevoir des entrées et des événements.[8].
- SOIL est une bibliothèque C utilisée pour charger des textures dans le contexte OpenGL[24].
- GLM est une bibliothèque mathématique en C ++ pour OpenGL[9].

Ces bibliothèques sont essentielles quand on veut créer un programme OpenGL. Ces bibliothèques étaient présentées dans le tutoriel *LearnOpenGL*, c'est pour cela qu'elles ont été choisies. Le fichier *main* est découpé en quatre parties :

- Les fonctions d'initialisation, ces fonctions vont créer tout ce dont on va avoir besoin pour effectuer la boucle de rendu (création de la fenêtre, contexte OpenGL, création et compilation des *shaders*, buffer contenant le cube, texture 3D, *render to texture*).
- La fonction de dessin, elle est appelée à toutes les *frames* pour effectuer les *drawCall* sur le *viewport*.
- Les fonctions de *call back*, elles sont appelées par la fenêtre quand une touche est enfoncee ou quand la souris bouge.
- La fonction main, elle commence par appeler les fonctions d'initialisation puis elle rentre dans la boucle de rendu principale, c'est aussi elle qui gère les *inputs* du clavier.

5.1.2 DirectX

N'ayant aucune expérience avec DirectX, une structure existante était la bienvenue. En plus, peu de temps pouvait être consacré à la réflexion d'une structure. Il a donc été décidé d'utiliser la structure fournie par le tutoriel Rastertek[21] qui avait servi pour apprendre DirectX 11. Le programme DirectX a une structure plus complexe que le programme OpenGL. Cela est dû au fait que DirectX demande au développeur de tout gérer lui-même. Contrairement à OpenGL qui s'utilise avec des bibliothèques, DirectX fournit tous les outils pour créer le contexte, et la fenêtre, gérer les inputs, charger les textures et effectuer des calculs matriciels. De ce fait, aucune autre bibliothèque n'a été utilisée à part DirectX 11. Mais en contrepartie, c'est au développeur de faire tout le travail que les bibliothèques effectuent en OpenGL.

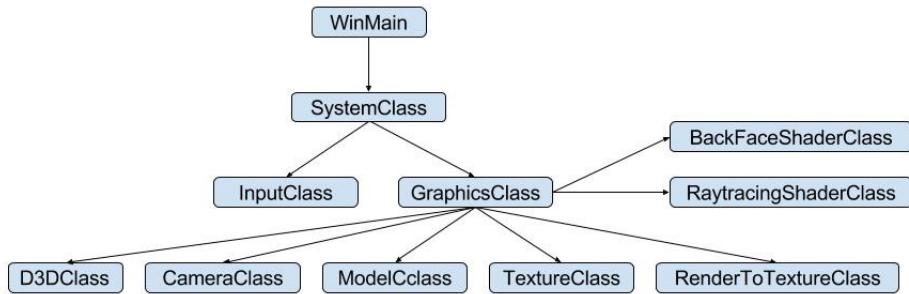


FIGURE 5.1 : Diagramme de classe du programme DirectX

- `WinMain` est le point d'entrée pour un programme Windows. Elle instancie la classe `System` qui contient le programme.
- `SystemClass` va créer la fenêtre du programme, elle instancie la classe `Graphics` et la classe `Input`. La classe `System` contient la boucle principale de rendu et fait le lien entre les inputs et les changements dans la scène.
- `InputClass` stocke l'état des touches du clavier.
- `GraphicsClass` contient toute la gestion de la scène et la fonction de dessin principale. Elle va instancier toutes les autres classes du programme et les utiliser pour faire le rendu.
- `D3DClass` va créer le contexte et le *device* DirectX qui sont utilisés dans le reste du programme. On va aussi créer tous les objets DirectX qui sont nécessaires pour afficher le rendu sur l'écran comme le *backBuffer* et *depthsentilBuffer*.
- `CameraClass` contient la caméra du programme.
- `ModelClass` déclare le buffer qui contient le cube.
- `TextureClass` charge la texture de la matrice en mémoire.
- `RenderToTextureClass` créer la texture et le *renderTarget* pour effectuer le *render to texture*.
- `RaytracingShaderClass` compile les *shaders* utilisés pour le *raytracing* et envoie les bons paramètres à chaque *shader*.
- `BackFaceShaderClass` compile les *shaders* utilisés pour le *render to texture* et envoie les bons paramètres à chaque *shader*.

Même si cette structure peut paraître compliquée, elle a permis de simplifier la compréhension et la rédaction du code DirectX. Une partie de cette structure a été réutilisée dans le programme de *remote computing*.

5.2 Matrice 3D

La matrice 3D est stockée sous la forme d'une texture 2D où les différentes tranches de la matrice sont mises bout à bout. OpenGL et DirectX fournissent tous les deux une ressource pour créer une texture 3D. La texture 3D stocke tranche par tranche la texture 2D pour créer un volume dans la carte graphique. On va ensuite pouvoir l'utiliser dans les *shaders* en y accédant avec des coordonnées 3D (x,y,z). C'est ce type de texture qui a donc été utilisé.

5.2.1 OpenGL

Un problème est apparu lors du chargement de cette texture 3D. Il est dû à un manque de compréhension de la documentation de la texture 3D d'OpenGL. Le problème venait de l'ordre des *bytes* chargés dans la texture. En effet, la texture 3D a besoin de récupérer la texture 2D tranche par tranche. Or, la texture de la matrice était chargée à l'aide de SOIL comme s'il s'agissait d'une texture 2D classique. Les *bytes* étaient chargés comme dans le schéma 5.2 de gauche ci-dessous. Il a donc fallu réorganiser les *bytes* qui composent l'image comme dans le schéma 5.2 de droite en suivant l'ordre indiqué par les flèches. Une fois que cette texture était chargée comme il faut, on a pu l'utiliser dans le *shader* pour récupérer les valeurs de la matrice.

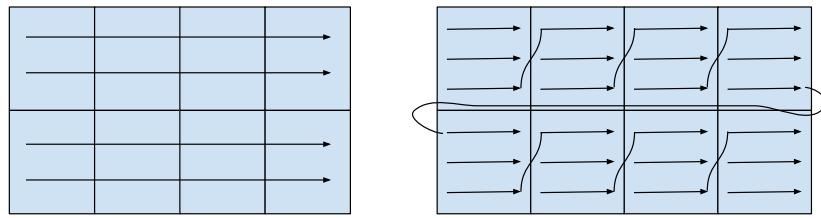


FIGURE 5.2 : Schéma représentant deux façons d'ordonner les bytes pour les charger dans la texture 3D

5.2.2 DirectX

Le problème précédent a été évité pour la version DirectX puisque le même algorithme a été appliqué. Mais un autre problème est apparu. Cette fois le problème est venu des *shaders*. Les valeurs récupérées par le *shader* ne correspondaient pas du tout avec celles stockées sur la texture. Beaucoup de temps a été perdu pour essayer de comprendre ce problème, mais aucune solution n'a été trouvée. Il a donc été décidé de ne plus utiliser une texture 3D pour stocker la texture de la matrice dans la carte graphique, mais d'utiliser une texture 2D classique. La texture a été chargée dans la carte sans problème. Il a donc fallu écrire une fonction dans le *shader* qui convertit des coordonnées 3D d'un voxel dans la matrice en coordonnées 2D sur une texture. Avec cette méthode le *volume ray tracing* a pu être effectué en DirectX. Il faut avoir conscience que cette méthode est moins performante que d'utiliser la ressource de la texture 3D.

5.3 Fonction de dessin

La fonction de dessin est appelée à chaque *frame* pour dessiner ce qui est stocké dans la carte graphique. Ce chapitre va détailler les étapes qui composent cette fonction pour les deux versions du programme.

5.3.1 OpenGL

La fonction de dessin d'OpenGL est contenue au sein d'une seule fonction *draw*. Elle est composée de trois étapes principales.

- On met à jour les matrices de Modèle, Vue, Projection.
- On va rendre le cube avec le *cull mode front* et le stocker dans une texture grâce au *render to texture*.
 - On lie le *framebuffer*, qui va contenir la texture du *render to texture*, à la carte graphique.
 - On nettoie les *buffers* pour dessiner la nouvelle image.
 - On active le programme qui contient les *shaders* qui réalisent le *render to texture*.
 - On active le *cull mode front*.
 - On envoie à la carte graphique les matrices et le *buffer* du cube.
 - On effectue le *drawcall*.
- On va rendre le cube avec le *cull mode back* et utiliser la texture calculée avant pour effectuer le *volume ray tracing*.
 - On va nettoyer les *buffers* pour dessiner la nouvelle image.
 - On active le programme qui contient les *shaders* qui réalisent le *volume ray tracing*.
 - On active le *cull mode back*.
 - On envoie à la carte graphique les matrices, le *buffer* du cube, les paramètres du *volume ray tracing*, la texture de la matrice et la texture du *render to texture*.
 - On effectue le *drawcall*.

A la fin, le rendu est affiché à l'écran.

5.3.2 DirectX

La fonction de dessin du programme DirectX est répartie dans plusieurs classes, mais la classe Graphics-Class contient la boucle de rendu principale. Les étapes pour le programme DirectX sont les mêmes que pour le programme OpenGL puisqu'ils utilisent la même technique de *volume ray tracing*. Les seules différences entre les deux programmes viennent des variations syntaxiques entre les deux API.

5.4 Shader

Les *shaders* sont le cœur du programme, c'est eux qui vont décider, de comment vont être traiter les données envoyées à la carte graphique avant le *drawcall*. Les quatre *shaders* utilisés dans chaque programme vont être détaillés.

5.4.1 OpenGL

Les *shaders* vont au moins toujours par deux, il y a le *vertex shader* et le *pixel shader*. Dans le programme deux couples de *shaders* sont utilisés. Les premiers vont créer la texture du *render to texture* et les deuxièmes effectuer le *volume ray tracing* à l'aide de la texture calculée à la première passe.

Le premier couple de *shaders* va créer la texture de *backface* voir la figure 4.1. Ces deux *shaders* utilisent uniquement les *buffers* du cube et les matrices MVP (Modèle, Vue, Projection). Le *vertex shader* va effectuer la projection des *vertex* du cube avec la matrice MVP et va envoyer au *pixel shader* la position des *vertex*. Le *pixel shader* va utiliser cette position comme couleur du pixel. Le rendu que ces *shaders* fournit est stocké dans la texture *backface*.

Le deuxième couple est plus complexe, c'est là que se trouve l'algorithme de *volume ray tracing*. Ces deux *shaders* utilisent les *buffers* du cube et les matrices MVP, mais aussi la texture de la matrice, la texture du premier rendu et les paramètres nécessaires au *volume ray tracing* comme le seuil, la position de la caméra ainsi que des booléens pour activer les différents modes de rendu. Le *vertex shader* va effectuer le même travail que le précédent, il va projeter les *vertex* du cube avec la matrice MVP et va envoyer au *pixel shader* la position des *vertex*. C'est grâce à cela que l'on va créer la texture *frontface* voir la figure 4.1. Le *pixel shader* contient donc tout l'algorithme de *volume ray tracing*. On commence par récupérer la position du point dans la texture *backface* et *frontface*. On soustrait ces deux valeurs pour obtenir la direction du rayon et on calcule le vecteur directeur normalisé. À partir de là, on peut lancer la boucle qui fait avancer le rayon. En fonction du mode de rendu choisi, l'algorithme ne va pas se dérouler de la même manière. À la fin de l'algorithme, la couleur du pixel est déterminée et envoyée à la fin du *pixel shader*.

Voilà comment fonctionnent les *shaders* dans le programme OpenGL.s

5.4.2 DirectX

Pour le programme DirectX le fonctionnement est exactement le même, il possède aussi deux couples de *shaders*, l'un calcule la texture *backface*, l'autre calcule le *volume ray tracing*. Les seules différences se trouvent au niveau de la syntaxe, mais c'est normal, car les deux programmes utilisent deux langages de *shader* différents.

5.5 Oversampling

L'*oversampling* ou suréchantillonnage intervient dans le *volume ray tracing* quand on doit récupérer la valeur du point d'échantillonnage.

5.5.1 OpenGL

Une fois que l'on connaît les coordonnées du point, on va aller récupérer la valeur dans la matrice. Avec l'*oversampling* on va récupérer les valeurs voisines et faire une moyenne avec celles-ci. Cette moyenne va remplacer la valeur située aux coordonnées du point. Grâce à cette méthode, on lisse le rendu. En effet, les bruits contenus dans la matrice sont effacés par la moyenne.

Cet algorithme est implémenté dans le *shader* qui effectue le *volume ray tracing*. Cette méthode était déjà présente dans le programme 3DvNCA et le code a été repris pour être utilisé dans le programme OpenGL. Pour calculer la moyenne, on va regarder sur les huit voxels voisins du voxel courant. La moyenne est réalisée à l'aide de la fonction mix disponible en GLSL qui réalise une interpolation linéaire entre deux valeurs. L'inconvénient de cette méthode vient des accès répétés à la texture. Effectuer cette opération demande beaucoup de temps à la carte graphique, cela fait perdre beaucoup de performance au programme.

5.5.2 DirectX

Comme expliqué avant cet algorithme est très coûteux, il n'a pas été réimplémenté dans le rendu DirectX ainsi que dans le programme de *remote computing*. En plus de cela, cet algorithme n'était pas une priorité pour faire fonctionner le pré-prototype. C'est pour cela qu'il a été mis de côté, mais il serait intéressant de l'intégrer dans la future version du programme, car il améliore grandement le rendu obtenu.

5.6 Modes de rendu

Plusieurs modes de rendu ont été implémentés, ils permettent de visionner la matrice de différentes manières. Les modes de rendu implémentés sont inspirés des modes présents dans le programme 3DvNCA. Il s'agit du mode seuil et du mode X-Ray.

5.6.1 OpenGL

Le mode seuil permet d'afficher la densité de la matrice. Une valeur de seuil définie entre zéro et un, cette valeur va être utilisée dans le *volume ray tracing*. Lorsque l'on récupère la valeur de la matrice au point d'échantillonnage, on va comparer cette valeur au seuil. Tant que la valeur est inférieure au seuil, on continue l'algorithme. Si la valeur est supérieure, on arrête l'algorithme et on calcule l'éclairage à ce point. Pour calculer la couleur du pixel, on va effectuer un *shading* simple avec la lumière ambiante et diffuse. La lumière ambiante est une constante du *shader* et la lumière diffuse est calculée en utilisant le gradient et la position de la caméra dans la scène. Le gradient représente la normale, et la position de la caméra représente la position de la lumière. De ce fait, les surfaces qui font face à l'utilisateur sont toujours bien éclairées. On peut faire varier la valeur du seuil afin de voir les différences de densité de la matrice.

Le mode X-Ray permet d'afficher la matrice à la manière d'une radio. Ce mode de rendu est plus simple que le précédent. On va simplement additionner toutes les valeurs des points d'échantillons à la suite. Quand le rayon a fini de traverser la matrice, on divise la somme obtenue par le nombre de points traversés pour obtenir un rendu cohérent.

Ces deux modes de rendu sont implémentés dans le programme OpenGL, on peut changer de mode de rendu pendant l'exécution du programme.

5.6.2 DirectX

Le programme DirectX possède les deux mêmes modes de rendu et ils sont implémentés exactement de la même manière qu'en OpenGL.

5.7 Remote computing

Vu que l'étudiant de Master n'a pas fourni de programme auquel intégrer le rendu DirectX, le programme d'exemple du *remote computing* a été utilisé. Cette intégration a été réalisée grâce au concours de M Bolinhas, il a remplacé l'étudiant de Master pour cette partie. La première étape a donc été de comprendre le fonctionnement du programme existant. Il est écrit en C++/CLI qui est une spécification du C++ standard créée par Microsoft. De nombreuses classes présentes dans ce programme ont été écrites spécifiquement pour le *remote computing*, il y a donc très peu d'informations les concernent. Tout cela a rendu la compréhension du programme difficile.

Une fois que le fonctionnement du programme a été plus clair, l'intégration du rendu a pu commencé. Mais avant il a fallu expliquer à M Bolinhas le fonctionnement du rendu et de DirectX. La première chose que l'on a faite, a été de créer toutes les classes utilisées pour faire le rendu, la classe qui contient le modèle, celle de la texture et du *render to texture*. Heureusement la structure du programme de *remote computing* ressemblait à celle du programme DirectX, ce qui a facilité l'intégration. Une fois que tous les éléments ont été implémentés, on a pu commencer à ajouter le rendu en *volume ray tracing* au programme.

Pour réaliser le rendu qui est envoyé au casque, il faut créer deux images pour avoir un rendu stéréoscopique, un pour chaque œil. Ces deux images ont une matrice de projection légèrement différente, c'est grâce à cela que la stéréoscopique est créée. Il faut donc implémenter deux fois la fonction de dessin, une fois pour chaque matrice. Le rendu s'affiche alors correctement dans les deux yeux.

La dernière chose qui a été rajoutée au programme de *remote computing*, c'est de pouvoir modifier le rendu qui est fait en appuyant sur une touche du clavier. On peut donc passer du mode X-ray à seuil et modifier la valeur du seuil.

Chapitre 6

Résultats

Ce chapitre va détailler les résultats obtenus pour les trois programmes réalisés. Vu qu'il s'agit d'applications graphiques, les résultats sont principalement des captures du rendu affiché à l'écran. HoloLens permet de faire des captures, on peut donc voir l'hologramme dans l'espace. Il faut avoir conscience que ces captures ne reflètent pas du tout l'effet 3D de la réalité augmentée.

6.1 Mode Seuil

Cette section va présenter les résultats obtenus avec le mode seuil sur le programme OpenGL, DirectX et HoloLens. Le mode seuil permet d'afficher la densité de la matrice. Le seuil est défini par une valeur qui peut être changée afin de voir plusieurs niveaux de densité. Voici des captures du programme OpenGL où le seuil a été changé à trois reprises.



FIGURE 6.1 : Capture du programme OpenGL en mode seuil. De gauche à droite le seuil vaut 0.8, 0.5, 0.2

Voici la capture du rendu obtenue en avec DirectX11. On peut voir que le résultat est identique à la version OpenGL, la seule différence est dans la couleur de la surface, la version DirectX est à peine plus rose. Cela est dû à une petite erreur dans le *shader*.

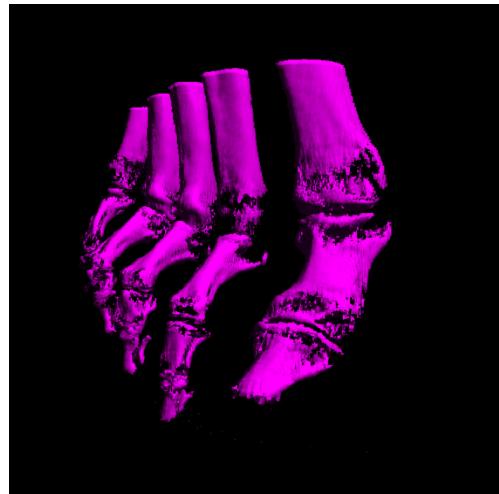


FIGURE 6.2 : Capture du programme DirectX en mode seuil, la valeur du seuil est de 0.5

Voici la capture du rendu tel qu'il est affiché sur HoloLens. On peut voir dans le fond de l'image le lieu où la capture a été réalisée.

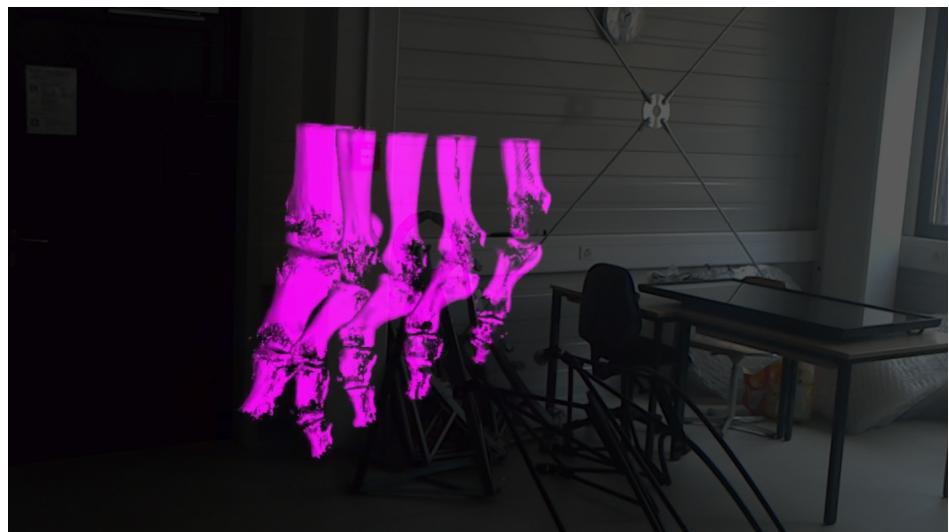


FIGURE 6.3 : Capture du rendu sur HoloLens en mode seuil, la valeur du seuil est de 0.5

6.2 Mode X-Ray

Cette section va présenter les résultats obtenus avec le mode X-Ray sur le programme OpenGL, DirectX et HoloLens. Le mode X-Ray permet d'afficher la matrice à la manière d'une radio. Ce mode gère la transparence et la densité de la matrice. Les captures ci-dessous montrent les résultats pour les programmes OpenGL et DirectX. On peut voir que les résultats sont identiques entre les versions.

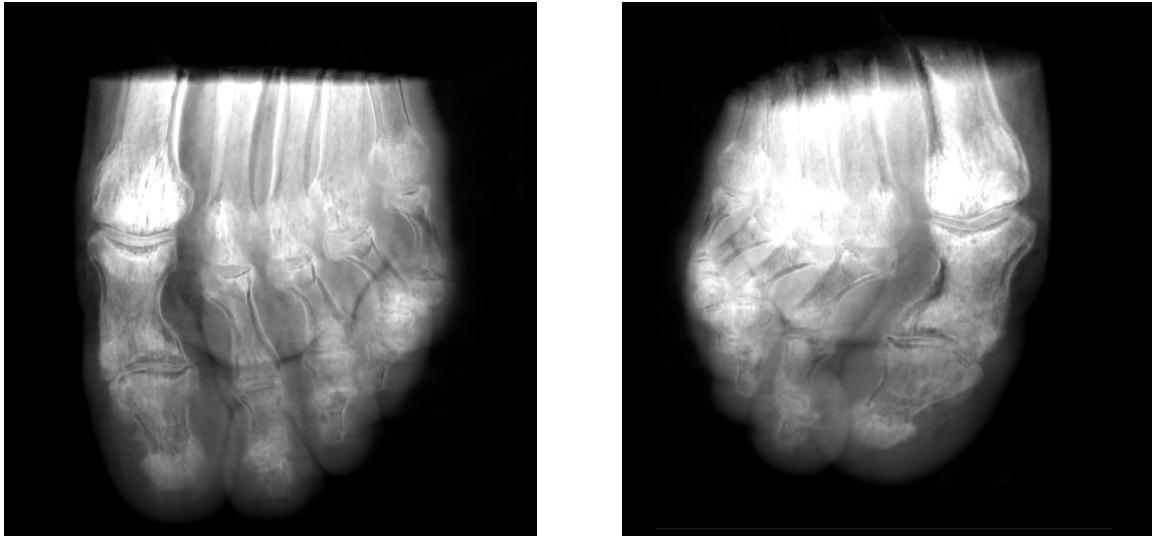


FIGURE 6.4 : Capture des programmes OpenGL et DirectX en mode X-Ray. A gauche la version OpenGL et à droite la version DirectX

Voici la capture du rendu en mode X-Ray tel qu'il est affiché sur HoloLens.



FIGURE 6.5 : Capture du rendu sur HoloLens en mode X-Ray

6.3 Mode Cumulé

Le mode cumulé superpose les modes vus précédemment, les modes X-Ray et seuil. Ce mode n'est utile que pour montrer les possibilités permises par le *volume ray tracing*.

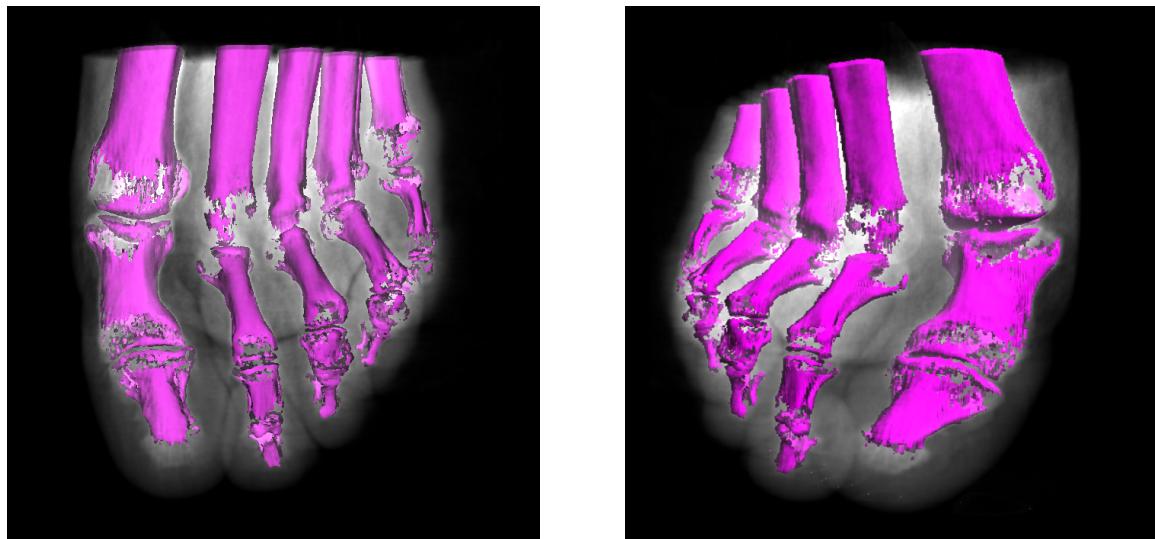


FIGURE 6.6 : Capture des programmes OpenGL et DirectX en mode cumulé. A gauche la version OpenGL et à droite la version DirectX

Voici la capture du rendu en mode cumulé tel qu'il est affiché sur HoloLens.

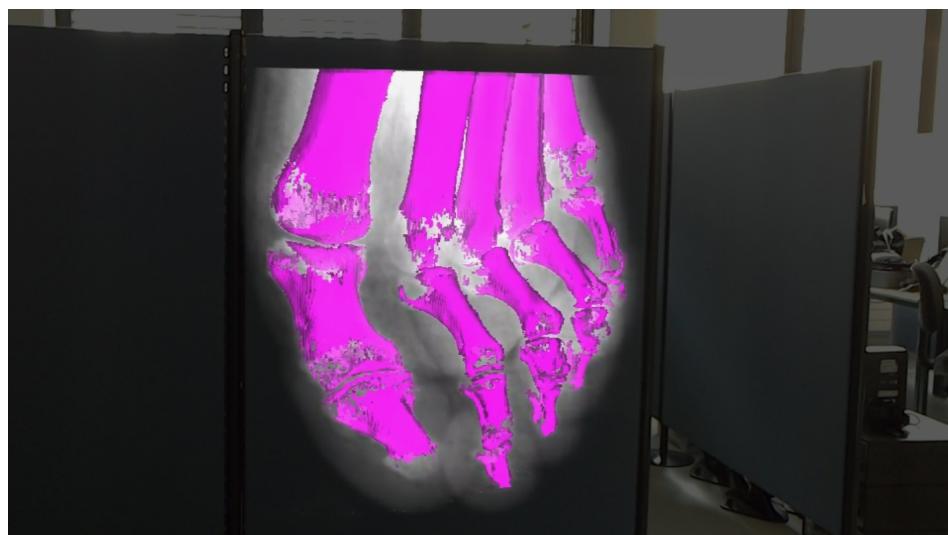


FIGURE 6.7 : Capture du rendu sur HoloLens en mode cumulé

6.4 Oversampling

L'*oversampling* a pour but d'enlever les défauts de la matrice et de lisser le rendu final au détriment des performances. En effet, cette fonctionnalité cause une baisse de *framerate* notable. C'est pour cela que cette fonctionnalité est uniquement présente sur le programme OpenGL.

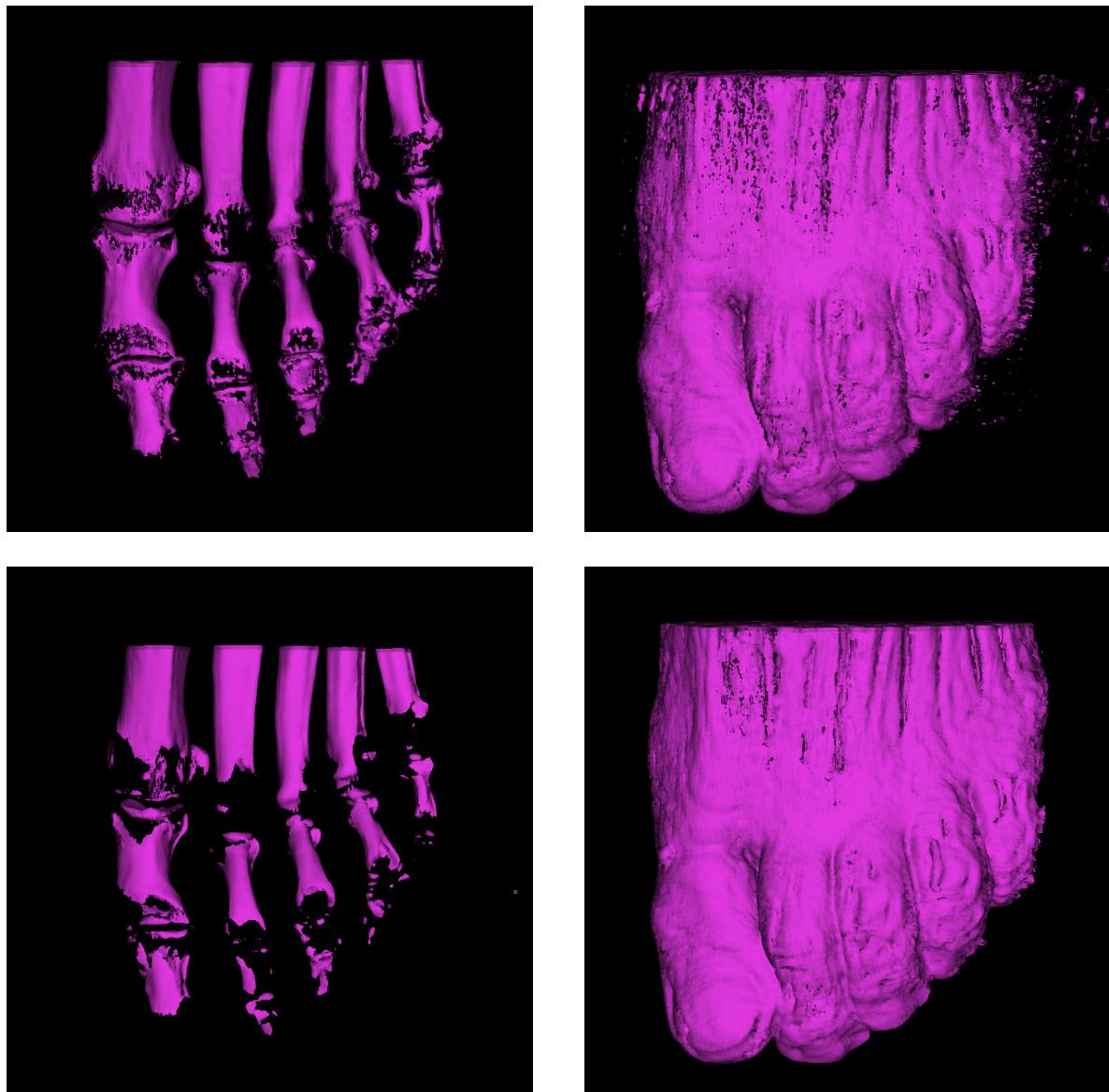


FIGURE 6.8 : Capture du programme OpenGL en mode seuil avec le mode *oversampling* activé pour les deux images du bas

6.5 Choix de la matrice

Toutes les captures effectuées précédemment ont été réalisées avec la matrice qui représente un pied. Cette texture était utilisée dans le programme 3DvNCA. L'avantage de cette matrice c'est qu'elle fournit un résultat qui est visuel, c'est pour cela qu'elle a été utilisée pour les captures. Avec la matrice du projet HoloM3D, le résultat est beaucoup moins visuel. Le programme OpenGL permet de voir cette matrice et de changer de type de scan entre CT et PET. Cette fonctionnalité n'est pas disponible sur le programme DirectX et sur l'HoloLens, car ce n'était pas une priorité et que le temps a manqué pour l'implémenter. Les deux captures ci-dessous montrent le rendu de la matrice en mode cumulé avec les deux types de scan.

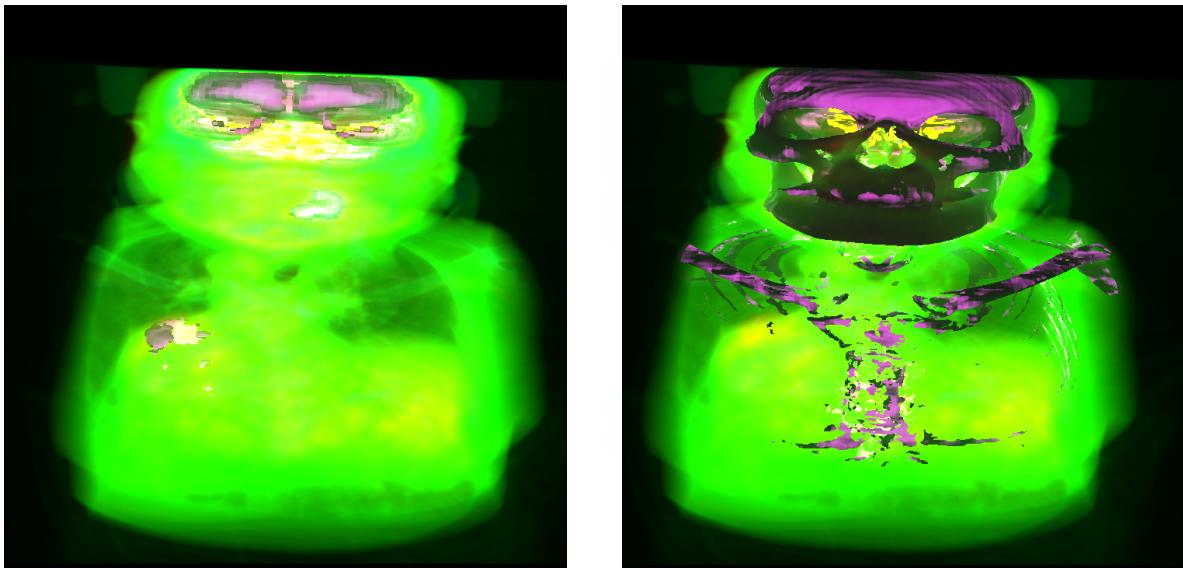


FIGURE 6.9 : Capture du programme OpenGL en mode cumulé avec la matrice du projet HoloM3D. A gauche les données du PET scan stockées dans le canal rouge et à droite les données du CT scan stockées dans le canal vert.

6.6 Autres résultats

Cette section va détailler des résultats annexes qui ont été observés. Concernant la communication entre le casque et le serveur, plusieurs tests ont été réalisés. Lors des premières communications, le réseau wifi du bâtiment a été utilisé, les performances étaient aléatoires. Le PC qui servait de serveur avait une carte Wifi, il a donc pu être utilisé comme routeur pour faire communiquer uniquement le serveur et l'HoloLens. Dans cette configuration, le flux était beaucoup plus fluide. Ces résultats sont totalement normaux, mais ils prouvent bien que pour faire fonctionner correctement le projet il faudra une infrastructure Wifi dédiée. Un test a été réalisé concernant les multiples rendus, quand le partage de l'hologramme sera fonctionnel le serveur devra rendre plusieurs rendus en même temps. Le test a donc été de lancer deux fois le programme de *remote computing* et les connecter à deux HoloLens pour voir si le serveur tenait la charge. Le résultat fut plutôt concluant, les deux programmes affichaient chacun leur matrice en temps réel. Le programme plantait de temps en temps, mais cela est plus dû à des exceptions mal gérées qu'aux limitations du serveur.

Chapitre 7

Problèmes rencontrés

Ce chapitre détaille les problèmes qui ont été rencontrés lors de la réalisation de ce travail de Bachelor. Les problèmes sont d'ordre organisationnel et technique.

7.1 Gestion du projet

Le projet a été mal géré, il n'aurait pas fallu lancer des travaux d'étudiant directement sur le projet. Une étude préliminaire sur HoloLens aurait permis d'éviter de perdre du temps. Il aurait fallu étudier toutes les possibilités que fournit l'HoloLens et Microsoft pour effectuer le *remote computing* et le partage d'hologramme. Il aurait fallu tester ces deux fonctionnalités et comprendre leur fonctionnement. Par exemple, avec le rendu OpenGL, si une étude avait été faite en amont, le rendu aurait pu être directement fait en DirectX.

7.2 Bibliothèques C++

Pour réaliser le programme OpenGL, plusieurs bibliothèques C++ ont été utilisées. Le premier problème vient des délivrables que fournissent les bibliothèques. Des fois, il s'agit de fichiers sources qu'il faut recompiler et des fois d'une dll. En effet, les bibliothèques C++ doivent être adaptées à leur plateforme d'exécution. C'est pour cela que les bibliothèques fournissent souvent les sources pour que les développeurs les utilisent dans leur programme.

Le deuxième problème vient de Visual Studio et de l'ajout d'une bibliothèque à un projet C++. Il faut ajouter le chemin de la bibliothèque dans les paramètres du projet. Cette manipulation n'est pas compliquée, mais quand on doit tester des bibliothèques elle devient très rébarbative. En plus, si on oublie d'ajouter ou d'enlever les chemins dans le fichier de paramètres le projet ne va plus se compiler.

7.3 DirectX

Le gros problème avec DirectX vient du manque d'information. Cela vient du fait que DirectX est surtout destiné à la création de jeux vidéo par des professionnels. Contrairement à OpenGL qui est utilisé dans le domaine universitaire et par de nombreux indépendants. Du coup, quand on fait une recherche sur un problème induit par DirectX, on va tomber sur peu de réponses. Quelques forums sont spécialisés là-dedans et c'est tout. En plus de cela, la documentation de DirectX n'est pas très complète, elle manque de code d'exemple, contrairement à celle d'OpenGL. Tous ces facteurs font qu'il est difficile de se lancer tout seul dans l'apprentissage de DirectX. Surtout quand on doit utiliser des techniques avancées que peu de personnes connaissent. La seule source d'aide pour DirectX vient du forum GameDev [6] où l'on trouve des *posts* concernant DirectX.

7.4 Programme de remote computing

C'est le programme de *remote computing* qui a posé le plus de difficultés durant ce travail de Bachelor. Il n'est jamais facile de reprendre le code que quelqu'un d'autre a écrit, mais c'est encore plus dur quand on ne connaît pas les technologies utilisées. Le programme d'exemple, fourni par Microsoft, qui effectue le *remote computing* est écrit en C++/CLI qui est une spécification du C++ standard. Cette spécification a été créée par Microsoft. Il y a de nombreux changements syntaxiques entre les deux langages. Cela n'arrange donc pas la compréhension du code de *remote computing*. Pour permettre au programme et à l'HoloLens de communiquer, les développeurs de Microsoft ont créé de nouvelles classes spécifiquement dédiées à cela. Comprendre leur fonctionnement quand la seule aide vient de la documentation, qui liste seulement les attributs et les méthodes de la classe, n'est pas chose aisée.

En plus pour effectuer le rendu, le programme utilise des fonctionnalités très spécifiques de DirectX. Pour obtenir un rendu stéréoscopique, que l'on peut envoyer au casque, il faut calculer deux fois la même image avec un petit décalage. Ce problème est résolu en utilisant un identifiant pour différencier les deux rendus.

Tous ces problèmes font qu'il a été difficile de comprendre le fonctionnement du programme dans le but d'y intégrer le rendu de la matrice.

Chapitre 8

Discussion

Ce chapitre termine ce rapport de travail de Bachelor, il comprend une conclusion concernant le travail réalisé et le projet HoloM3D, ainsi qu'une évocation des perspectives d'avenir du projet réalisé.

8.1 Conclusion

Le but originel de ce travail de Bachelor était de réaliser un rendu en *volume ray tracing* d'une matrice 3D pour le projet HoloM3D. Le projet HoloM3D a pour but de fournir aux chirurgiens du CHUV une nouvelle façon de visualiser une matrice 3D grâce au casque HoloLens pour les aider lors des séances préopératoires.

Suite aux problèmes de l'étudiant de Master et à certains problèmes techniques, les objectifs du travail de Bachelor ont changé pour ajouter l'intégration au programme de *remote computing*. Au final, les objectifs initiaux ont été largement dépassés puisque le rendu de la matrice a été effectué en OpenGL, en DirectX et intégré à HoloLens avec le *remote computing*. En plus de cela, il a fallu apprendre DirectX et les technologies associées à l'HoloLens au cours du projet.

Le pré-prototype actuel fonctionne, l'utilisateur peut visualiser la matrice 3D en temps réel dans le casque. Grâce aux hologrammes de l'HoloLens, les chirurgiens devraient pouvoir améliorer leur analyse des données du patient.

Ce pré-prototype valide la faisabilité du projet. Grâce au travail effectué, le projet HoloM3D est devenu plus concret et d'autres personnes vont pouvoir continuer à travailler dessus.

8.2 Perspective d'avenir

Ce projet a beaucoup d'avenir à court et à moyen terme. Dans un premier temps, pour le prototype en lui-même, qui n'est pas terminé. Et dans un deuxième temps, pour le projet HoloM3D en général. Plusieurs éléments sont à améliorer sur le prototype actuel.

- Pouvoir sélectionner l'adresse IP du casque et la matrice à afficher.
- Rajouter de l'interaction avec l'hologramme, pouvoir l'agrandir, le pivoter et le déplacer. Utiliser les commandes vocales pour changer les modes de rendu.
- Récupérer les informations sur la communication entre le casque et le serveur (état du réseau Wifi, *frame* perdue) pourrait être utile. Pour l'instant, la communication est parfois instable, ces informations permettraient de comprendre ces baisses de performance.
- Les deux modes de rendu actuels ne sont pas forcément ceux qui intéressent le plus les chirurgiens. Avoir un feedback du CHUV concernant le rendu actuel pour pouvoir l'améliorer afin qu'ils puissent en tirer le maximum d'informations.
- Le partage des hologrammes sur plusieurs casques reste le plus gros problème. Cette fonctionnalité est centrale pour le projet et elle n'a pas encore trouvé de solutions.

Ce prototype n'est donc pas du tout terminé, beaucoup d'améliorations peuvent encore y être ajoutées. Le projet HoloM3D a beaucoup d'avenir, mais du travail reste à faire pour obtenir un véritable prototype.

Bibliographie

- [1] Divine Augustine. Getting started with volume rendering using opengl. <https://www.codeproject.com/Articles/352270/Getting-started-with-Volume-Rendering>, 2013. [Online; accédé 25 Juillet, 2017].
- [2] Windows Dev Center. Volume rendering. https://developer.microsoft.com/en-us/windows/mixed-reality/volume_rendering, 2017. [Online; accédé 20 Juillet, 2017].
- [3] Bruno Nazarian – CNRS. Imagerie médicale 3d visualisations, segmentations et reconstructions. <http://bnazarian.free.fr/MyUploads/ImagerieMedicale3D.pdf>, 2002. [Online; accédé 25 Juillet, 2017].
- [4] Joey de Vries. Learnopengl. <https://learnopengl.com/>, 2017. [Online; accédé 25 Juillet, 2017].
- [5] Henrique Debarba. Volume rendering with the hololens. https://www.youtube.com/watch?v=o47QA-_CQuk, 2017. [Online; accédé 20 Juillet, 2017].
- [6] Forum gamedev. <https://www.gamedev.net/>, 2017. [Online; accédé 25 Juillet, 2017].
- [7] glew. <http://glew.sourceforge.net/>, 2017. [Online; accédé 09 Septembre, 2017].
- [8] GLFW. <http://www.glfw.org/>, 2017. [Online; accédé 25 Juillet, 2017].
- [9] glm. <https://glm.g-truc.net/0.9.8/index.html>, 2017. [Online; accédé 25 Juillet, 2017].
- [10] Gobron S. Cöltekin A. Bonafos H. and Thalmann D. Gpgpu computation and visualization of three-dimensional cellular automata. http://www.stephane-gobron.net/Core/Publications/Papers/2011_TVCJ_3DCA.pdf, 2011. [Online; accédé 25 Juillet, 2017].
- [11] Kyle Hayward. Volume rendering 101. <http://graphicsrunner.blogspot.ch/2009/01/volume-rendering-101.html>, 2009. [Online; accédé 14 Juillet, 2017].
- [12] Frank D. Luna. Introduction to 3d game programming with direct3d 11.0. <http://www.d3dcoder.net/d3d11.htm>, 2012.
- [13] Microsoft. Add holographic remoting. https://developer.microsoft.com/en-us/windows/mixed-reality/add_holographic_remoting, 2017. [Online; accédé 26 Juillet, 2017].
- [14] Microsoft. Add holographic remoting. https://developer.microsoft.com/en-us/windows/mixed-reality/add_holographic_remoting, 2017. [Online; accédé 31 Août, 2017].
- [15] Microsoft. Holographic remoting player. https://developer.microsoft.com/en-us/windows/mixed-reality/holographic_remoting_player, 2017. [Online; accédé 31 Août, 2017].
- [16] Microsoft. Remotingghostsample. <https://github.com/Microsoft/MixedRealityCompanionKit/tree/master/RemotingHostSample>, 2017. [Online; accédé 31 Août, 2017].
- [17] Mircosoft. Hololenscompanionkit. <https://github.com/Microsoft/HoloLensCompanionKit>, 2017. [Online; accédé 20 Juillet, 2017].
- [18] John Pawasauskas. Volume visualization with ray casting. <http://web.cs.wpi.edu/~matt/courses/cs563/talks/powwie/p1/ray-cast.htm>, 1997. [Online; accédé 28 Juillet, 2017].

- [19] Pcworld. We found 7 critical hololens details that microsoft hid inside its developer docs. <http://www.pcworld.com/article/3039822/consumer-electronics/we-found-7-critical-hololens-details-that-microsoft-hid-inside-its-developer-docs.html>, 2017. [Online ; accédé 04 Septembre, 2017].
- [20] CS 788H S01 Project. Gradient estimators. http://www.aravind.ca/cs788h_Final_Project/gradient_estimators.htm, 2005. [Online ; accédé 18 Juillet, 2017].
- [21] rastertek. Directx 11 tutorials. <http://www.rastertek.com/tutdx11.html>, 207. [Online ; accédé 25 Juillet, 2017].
- [22] Daniel Rubino. Microsoft hololens - here are the full processor, storage and ram specs. <https://www.windowcentral.com/microsoft-hololens-processor-storage-and-ram>, 2017. [Online ; accédé 04 Septembre, 2017].
- [23] Braynzar Soft. Directx 11 - braynzar soft tutorials. <https://www.braynzarsoft.net/viewtutorial/q16390-braynzar-soft-directx-11-tutorials>, 2015.
- [24] SOIL. <http://www.lonesock.net/soil.html>, 2017. [Online ; accédé 25 Juillet, 2017].
- [25] Forum stackoverflow. <https://stackoverflow.com>, 2017. [Online ; accédé 25 Juillet, 2017].
- [26] toolchainX. Volume rendering using glsl. https://github.com/toolchainX/Volume_Rendering_Using_GLSL, 2013. [Online ; accédé 17 Juillet, 2017].
- [27] unity3d. Développez des applications de réalité mélangée avec unity pour microsoft hololens. <https://unity3d.com/fr/partners/microsoft/hololens>, 2017. [Online ; accédé 04 Septembre, 2017].
- [28] Wikipedia. Lancer de rayon. https://fr.wikipedia.org/wiki/Lancer_de_rayon, 2017. [Online ; accédé 14 Juillet, 2017].
- [29] Wikipedia. Pipeline 3d. https://fr.wikipedia.org/wiki/Pipeline_3D, 2017. [Online ; accédé 25 Juillet, 2017].
- [30] Wikipedia. Réalité augmentée. https://fr.wikipedia.org/wiki/R%C3%A9alit%C3%A9_augment%C3%A9e, 2017. [Online ; accédé 19 Juillet, 2017].
- [31] Wikipedia. Réalité virtuelle. https://fr.wikipedia.org/wiki/R%C3%A9alit%C3%A9_virtuelle, 2017. [Online ; accédé 19 Juillet, 2017].
- [32] Wikipedia. Raytracing. [https://en.wikipedia.org/wiki/Ray_tracing_\(graphics\)](https://en.wikipedia.org/wiki/Ray_tracing_(graphics)), 2017. [Online ; accédé 14 Juillet, 2017].
- [33] Wikipedia. Volume ray casting. https://en.wikipedia.org/wiki/Volume_ray_casting, 2017. [Online ; accédé 14 Juillet, 2017].

Annexes

Cahier des charges

Cahier des charges pour travail de **diplôme/semestre**

Titre :	Rendu IRM-3D - HoloLens
N° projet :	219
Étudiant :	Droxler Arnaud
Encadrant :	Stéphane Gobron

Situation initiale

Les chirurgiens utilisent l'imagerie médicale pour détecter les maladies et pour connaître mieux la disposition des organes dans le corps du patient. En effet les organes ne sont jamais disposés de la même position, cela dépend du patient. Les chirurgiens doivent souvent « improviser » au moment de l'opération, s'il pouvait mieux connaître l'intérieur du patient cela les aiderait lors de l'opération et cela réduirait le risque d'accident.

En imagerie médicale, une IRM permet d'observer la résonance magnétique nucléaire des protons d'eau contenus dans l'organisme. Il est donc possible d'observer des altérations des tissus grâce aux différences de densité d'eau dans le corps. Pour représenter une matrice de voxel (pixel 3D qui compose une IRM), il existe plusieurs méthodes comme MIP, *Marching Cubes*, ray-tracing. Le raytracing est la méthode la moins contraignante et qui permet d'obtenir le rendu le plus réaliste.

Hololens est une nouvelle technologie développée par Microsoft, il s'agit d'un casque de réalité augmentée. Il permet de simuler des hologrammes qui s'intègrent dans le champ de vision de l'utilisateur. Les hologrammes peuvent être disposés dans l'espace et visualisés par plusieurs personnes en même temps.

Dans ce contexte, l'idée directrice du projet est de permettre aux chirurgiens de visualiser des IRM et autres matrices HR dans n casques Hololens à plusieurs spécialistes simultanément et ce, dans le cadre de séances préopératoires.

Buts du projet

Le but du projet est de réaliser un double rendu x n de type *raytracing* (i.e. lancé de rayons) d'une matrice 3D en temps réel dans un casque Hololens. Ce projet se concentre uniquement sur le rendu des matrices 3D. La capacité de calcul des Hololens étant restreinte, il faut utiliser un serveur pour calculer les différents rendus. Toute la partie de la gestion du serveur et des différents Hololens est réalisée par un autre étudiant de master.

Objectifs principaux

Ce projet se concentre sur le rendu en *raytracing*, la qualité de ce rendu est donc primordiale.

- Raytracing simple passe x $2n$;
- Seuillage fausse couleur en fonction des densités et des zones ;
- Réaliser un rendu « réaliste » ;
- Maintenir un *framerate* constant de 60 FPS.

Pour maintenir un tel framerate, il faudra mettre en place un serveur avec une grosse puissance de calcul.

Objectifs secondaires

La méthode de raytracing permet de rendre de manière réaliste le comportement de la lumière en effectuant plusieurs passes on obtient un résultat plus réaliste.

- Effectuer plusieurs passes de raytracing ;
- Augmenter le réaliste avec des matériaux (PBR) ;
- Ajouter des textures de couleurs
- Optimiser l'algorithme de *raytracing*
- Recherche sur l'Hololens et la visualisation de volume

St-Imier, le 2017

Les parties ci-dessous déclarent accepter le contenu du présent cahier des charges.

Un exemplaire est remis à chaque partie.

Étudiant : Encadrent

Distribution

Étudiant concerné
Filière informatique