Projet de fin d'études
Imagerie Médicale et Réalité Mixte sur Meta Quest 3

Florian Bernadet, Yanis Dubois, Maxime Oçafrain, Coralie Rigaut

27 mars 2024

Encadrants : Fabien Baldacci et Pascal Desbarats

Table des matières

Introduction

La visualisation d'imagerie médicale représente aujourd'hui un véritable défi. Les données, bien que tridimensionnelles par nature, sont souvent présentées et analysées en deux dimensions, ce qui limite la compréhension spatiale et la perception globale des organes étudiés. Cette approche conventionnelle requiert un effort mental considérable pour appréhender la connectivité tridimensionnelle des structures anatomiques, sans parler de l'impossibilité de visualiser ces dernières à leur échelle réelle.

L'arrivée des technologies de réalité mixte (XR), qui englobent la réalité augmentée (AR) [1] et la réalité virtuelle (VR) [2], incarnée notamment par le casque Meta Quest 3, ouvre des horizons prometteurs pour pallier ces contraintes. C'est notamment la technologie du passthrough, qui permet de voir le monde réel à travers les caméras intégrées du casque, tout en y intégrant des éléments virtuels qui nous permet une telle immersion. Toutefois, leur potentiel reste largement sous-exploité dans le domaine de la visualisation, une lacune que ce projet aspire à combler.

Parallèlement, ce projet s'étend au-delà du cadre médical pour répondre à une demande spécifique mais cruciale : le développement d'une solution de visualisation en réalité mixte pour les fichiers au format `OBJ`. Cette initiative répond à un besoin manifeste, puisqu'il n'existe actuellement pas d'outil permettant l'exploration de ces modèles 3D dans un environnement de réalité mixte. Effectivement, la problématique d'appréhension des objets 3D est analogue à celle rencontrée dans la visualisation d'examens d'imagerie médicale, et peut nécessiter l'utilisation d'outil immersif pour être résolu.

Notre projet propose donc deux applications principales : la visualisation d'objets 3D et la visualisation de fichiers d'imagerie médicale, comme des DICOM [19], par communication avec l'application 3DSlicer. Toutes deux sont réalisées pour le casque Meta Quest 3 et construites en utilisant l'environnement de développement Unity, avec une base commune reposant sur `OpenXR`[21] pour assurer une compatibilité étendue. Pour la visualisation des données médicales, le projet s'appuie sur une extension spécifique de 3DSlicer [10], ainsi que sur l'intégration du protocole `OpenIGTLink` [32].

1. La réalité augmentée correspond à l'ajout d'élément virtuel dans le monde réel. Elle est notamment mise en place avec les tablettes et les téléphones
2. La réalité virtuelle est une technologie qui nous immerge dans un monde alternatif. La mise en place d'une telle chose se fait généralement grâce à des casques de réalité virtuelle

I. État de l'art

La réalité virtuelle et la réalitée augmentée sont aujourd'hui deux technologies qui connaissent un véritable essor depuis une dizaine d'années. Cela se traduit par une augmentation des casques de réalitée virtuelle, augmentée ou mixte lancés sur le marché (ainsi en l'espace de 6 mois Meta et Apple ont tous deux sortis un casque de réalité mixte là où, avant 2010 cela était impensable). Cet essor marque aussi le fait que de plus en plus de domaines s'approprient ces technologies, la médecine est l'un d'entre eux. Comme dit dans l'introduction, la visualisation en 3D grâce à la réalité mixte permet une meilleure appréhension des données tridimensionnelles qu'en 2D via une projection sur un écran.

Aujourd'hui, la médecine tire partie de cette observation sur la visualisation des données et des reconstructions qui en émergent pour diverses tâches. L'éducation des étudiants pour enseigner l'anatomie par exemple [4], [20] ou pour les mettre en situation [16]. Le travail avec des patients comme l'explique ces deux papiers [9], [22]. Des études [27], [26] montrent bien que ce domaine s'est emparé de ces technologies, puisqu'elles estiment que d'ici 2030 le taux de croissance annuel composé (TCAC [3]) sera d'environ 20%.

La représentation standard des données liées à des examens médicaux (Scanner CT, IRM, échographie, ...) est aujourd'hui le format "Digital Imaging and Communication in Medicine" (DICOM) [19]. Plusieurs applications permettent déjà la visualisation de ces fichiers DICOM en réalité augmentée ou réalité virtuelle. Le tableau 1 fait un récapitulatif (non exhaustif) de telles applications permettant visualisation et segmentation.

TABLE 1 – Applications permettant la visualisation et la segmentation d'examens médicaux au format DICOM en réalité virtuelle, tableau inspiré du Tableau 1 de l'article [24]

Application	Matériel
Medical Imaging XR	Multi-support (Meta Quest 3 supporté)
HoloDicom	HoloLens
DicomHolo	HoloLens
3D Organon	Multi-support (Meta Quest 3 supporté avec un lien filaire)
Elucis	Multi-support (Meta Quest 3 supporté avec un lien filaire)

D'autres applications existent mais elles ne permettent de faire que de la visualisation : 3Dicom VR, Specto VR. Ou alors si elles sont très complètes, elles fonctionnent avec un matériel propre et souvent plus sophistiqué qu'un simple casque de réalité augmentée et/ou réalité virtuelle (XR) grand public comme True 3D.

Elucis [17] est un logiciel développé par l'entreprise Realize Medical qui a été spécialement conçu pour créer des modèle 3D à partir des données médicales de patients. Comme pour toutes les applications proposées dans cette partie, le format DICOM pour les données d'entrée est supporté. Le logiciel est à installer sur PC et fonctionne ensuite (pour Meta Quest 3) via un lien filaire et le téléchargement de l'application Occulus. L'avantage de ne pas utiliser le casque avec une pratique "Standalone" est

3. Correspond au taux moyen de croissance des revenus, des ventes ou des investissements au fil du temps. Il est calculé par la formule suivante : $TCAC = \frac{VF}{VI}^{\frac{1}{n}} - 1$ où VF est la valeur finale, VI est la valeur initiale, n est le nombre d'années

de pouvoir supporter de lourds fichiers en entrée. Cela permet de travailler "rapidement", de ne pas souffrir de latence lors de la visualisation et de la manipulation en VR de modèle. En revanche, cela impose d'avoir un ordinateur ayant des spécifications minimum (cf. annexe 1). Développée dans le but d'améliorer la compréhension des patients il s'agit d'une proposition qui offre de nombreuses fonctionnalités.

Le tableau 2 donne une vue comparative des différentes fonctionnalités qui peuvent nous intéresser.

Medical Imaging XR [18] est un logiciel qui permet d'importer les données médicales d'un patient pour les visualiser dans un espace XR. Il est développé par Medicalholodeck, une entreprise spécialisée dans le développement d'outil en XR en lien avec l'enseignement de la médecine et de la chirurgie. Sur le marché depuis plusieurs années Medical Imaging XR est disponible sur divers casques XR mais il est recommandé de l'utiliser sur Magic Leap 2, ou Meta Quest (3, 2 ou Pro). Pour l'obtenir il suffit de se rendre sur le store officiel et télécharger l'application Medicalholodeck et acheter une licence parmi quatre proposées qui vont de l'utilisation personnelle à l'utilisation institutionnelle. Cette application a pour but de permettre la planification et l'entraînement chirurgicaux, la collaboration entre professionnels du secteur de la santé, l'enseignement de l'anatomie et la formation médicale en XR. Il s'agit d'une application très complète (cf. annexe 3). Le fait que la visualisation de DICOM en 3D dans un espace de VR ne soit pas sa fonctionnalité principale peut se faire sentir par le possible lag qui peut advenir si les fichiers DICOM sont trop lourds. Ce constat rend Medical Imaging XR, bien que très complète, limitante dans cette utilisation.

3D Organon [2] est aussi une plate-forme éducative médicale spécialisée dans l'enseignement et l'apprentissage de l'anatomie (cf. annexe 2). Comme pour Medicalholodeck, il existe un système de licences payantes. L'utilisation de 3D Organon pour visualiser en 3D des fichiers DICOM ne peut se faire qu'avec l'application premium sur son PC, le casque sert alors à la visualisation. Il est impossible de visualiser des fichiers DICOM en 3D sur la version "Standalone" de l'application. Cela à l'avantage de pouvoir utiliser la puissance d'un ordinateur pour faire ce type de tâche. Ainsi, la latence liée à la visualisation de fichiers DICOM lourds (de haute définition) doit être limitée (surtout en comparaison avec Medicalholodeck - Medical Imaging XR, qui reconnaît ce problème). En revanche, le coeur de ce logiciel n'est pas la visualisation de fichier DICOM mais bien l'apprentissage de l'anatomie à partir de modèle 3D interactifs initialement présent dans l'application. L'utiliser pour visualiser des reconstructions 3D dans une espace VR de fichiers DICOM n'est pas une solution optimum.

HoloDicom [13] est un logiciel développé spécifiquement pour les casques AR Microsoft Hololens qui permet d'afficher et de convertir en modèle 3D tout fichier DICOM. Une fois le modèle ancré dans l'espace il est possible de faire les manipulations suivantes : rotation, translation, segmentations (basées sur diverses méthodes) et mise à l'échelle, le travail collaboratif semble aussi possible. Moins complet que Medicalholodeck - Medical Imaging XR, HoloDicom semble aussi plus difficilement accessible et souffre des problèmes liés à la réalité augmentée par "see-through", comme le fait que la visualisation soit difficile dans des conditions de luminosité importante.

	Elucis	Medical Imaging XR *	3D Organon *	HoloDicom
Manipulations basiques (rotation et translation)	✓	✓	✓	✓
Zoom	✓	✓	✓	✓
Mise à taille réelle	✓	✓	✓	✓
Opacité ajustable	✓	✓	✓	✓
Coupe	✓	✓	✓	✓
Segmentation (seuillage)	✓	✓	✓	✓
Outils de mesures	✓	✓	-	✓
Dessin 3D	✓	✓	✓	X
Exportation du modèle	STL et OBJ	STL	-	-
Travail collaboratif	✓	✓	✓	✓
Enregistrement	✓	✓	-	✓

TABLE 2 – Fonctionnalités proposées par les applications permettant le travail en 3D d'examens médicaux au format DICOM en réalité virtuelle, tableau inspiré du Tableau 2 de l'article [24]
Légende : ✓ → la fonctionnalité est supportée par le logiciel, X → la fonctionnalité n'est pas supportée par le logiciel et - → inconnue
*: Le choix de la licence est important : ces fonctionnalités ne sont pas incluses dans toutes les licences

Hors de la réalité augmentée et virtuelle il existe un logiciel : 3DSlicer[10] qui permet de visualiser et d'analyser de l'imagerie médicale. Il est grandement utilisé, et a notamment été téléchargé plus d'un million de fois et référencé dans 12 000 publications académiques en 2021 [4] . Cette plate-forme, qui a l'avantage d'être disponible sur plusieurs systèmes d'exploitation : Windows, Linux et MacOSX, d'être libre et open-source, permet la création d'extension de fonctionnalité via plug-in. Il existe un plug-in pour travailler sur 3DSlicer en VR : SlicerVR [23].

SlicerVR [23] est une extension open-source de 3DSlicer, elle permet de manipuler une scène 3D de 3DSlicer en réalité virtuelle. Pensée il y a plus de 7 ans elle doit normalement fonctionner sur tous les casques ayant une compatibilité avec Open-VR. Aujourd'hui son code semble permettre l'utilisation de Open-XR. Sur Meta Quest 3 l'utilisation n'est pas triviale, toutes les fonctionnalités de manipulation promises n'ont pas été trouvées.

Voici les fonctionnalités annoncées que nous avons pu tester :

— Afficher tout le contenu des visionneuses 3D dans 3DSlicer
— Aligner la vue du casque pour qu'elle corresponde au point de vue de la vue 3D sélectionnée dans 3DSlicer
— Saisir et repositionner des objets [5]
— Réglage avancé des performances de rendu du volume pour trouver un bon équilibre entre la qualité de l'image et le taux de rafraîchissement

4. Source des chiffres donnés disponible *ici* (accès le 25 mars 2024)
5. Repositionnement d'objets possible mais limité avec les joysticks et saisi avec les manettes très instable

Et voici les fonctionnalités encore inaccessibles :

- Afficher les volumes sous forme de tranches d'image 2D ou de rendu de volume, rendre des surfaces, des points, etc.
- Visualiser n'importe quel jeu de données 4D, en utilisant n'importe quelle technique de rendu (y compris le rendu de volume)
- Déplacer, faire pivoter, mettre à l'échelle le monde (tous les objets)
- Rendre la position des manettes disponible sous forme de transformations dans la scène 3DSlicer

Malgré ces manques, cette application est intéressante puisque son existence prouve la possibilité d'établir une communication entre le logiciel 3DSlicer et le casque de réalité mixte Meta Quest 3. En revanche, comme cette connexion se fait grâce à SteamVR (accessible sur Windows uniquement) SlicerVR ne représente pas une solution envisageable pour répondre aux besoins client.

II. User stories

L'idée de ce projet est de développer, dans un premier temps, un lecteur de fichiers OBJ permettant la visualisation et manipulation d'objet 3D chargé depuis l'application. Puis dans un second temps, de créer un lecteur d'objet 3D issu de 3DSlicer afin de les visualiser et de les manipuler. Chacune des deux applications doivent être disponibles sous forme d'Android Package (APK) afin de pouvoir être utilisées dans le casque de réalité mixte : Meta Quest 3.

Première phase : Lecteur de fichier OBJ

L'utilisateur peut manipuler un objet en 3D dans un environnement en réalité mixte. Il peut le faire tourner, le déplacer, faire varier sa taille, le mettre à taille réelle. L'utilisateur pourra choisir la manière dont il souhaite importer un objet 3D à travers une interface :

- Chargement d'un fichier local, présent sur le casque
- Chargement depuis un URL
- Chargement depuis un serveur SaMBa (SMB)

Le tableau 3 récapitule et priorise l'ensemble des User Stories apparues au cours des discussions avec nos clients.

User Story	Priorité (de 1 à 10)
L'utilisateur peut visualiser un objet 3D dans un environnement en réalité mixte	10
L'utilisateur peut tourner l'objet selon tous les axes de rotation	9
L'utilisateur peut déplacer l'objet dans n'importe quelle direction	9
L'utilisateur peut augmenter ou réduire la taille de l'objet 3D	8
L'utilisateur peut afficher l'objet 3D à sa taille réelle	8
L'utilisateur peut importer un fichier local du casque	7
L'utilisateur peut importer un fichier depuis une URL	7
L'utilisateur peut importer un fichier depuis un serveur SaMBa	7
L'utilisateur peut choisir son mode d'importation via une interface graphique	7
L'utilisateur peut remplir les informations nécessaires à l'importation du fichier via une interface graphique selon le mode d'importation sélectionné	7
L'utilisateur peut visualiser le nombre de sommets et de faces de l'objet importé	6
L'utilisateur peut visualiser plusieurs objet 3D dans une même scène	5
L'utilisateur peut supprimer un objet sélectionné de la scène	5
L'utilisateur peut supprimer tous les objets de la scène	5
L'utilisateur peut visualiser l'objet en nuage de points	2
L'utilisateur peut visualiser l'objet en fil de fer	2
L'utilisateur peut "dessiner" sur l'objet	1

TABLE 3 – User Stories correspondant à l'application de visualisation d'objet 3D type OBJ.

Seconde phase : Lecteur de données issues de 3DSlicer

L'utilisateur manipule des données au format DICOM[19] via 3DSlicer[10] sur son ordinateur, qu'il peut envoyer sur l'application de visualisation en réalité mixte, et ce, quel que soit le système d'exploitation sur lequel il utilise 3DSlicer. Dans le casque, l'utilisateur pourra appliquer au modèle les manipulations de translation, rotation et de mise à l'échelle, de façon similaire à ce qui est énoncé dans la première phase.

Le tableau 4 récapitule et priorise l'ensemble des User Stories apparues au cours des discussions avec nos clients.

User Story	Priorité
L'utilisateur peut charger et visualiser un modèle de type maillage issu de 3DSlicer dans l'application	10
L'utilisateur peut tourner le modèle selon tous les axes de rotation	9
L'utilisateur peut déplacer le modèle dans n'importe quelle direction	9
L'utilisateur peut augmenter ou réduire la taille de l'objet 3D	8
L'utilisateur peut afficher l'objet 3D à sa taille réelle	8
L'utilisateur peut visualiser le nombre de sommets et de faces du maillage qu'il a importé	6
L'utilisateur peut importer plusieurs modèles dans une même scène depuis 3DSlicer	5
L'utilisateur peut supprimer de la scène un modèle sélectionné	5
L'utilisateur peut supprimer de la scène tous les modèles	5
Le modèle se met à jour dans l'application s'il a été modifié sur 3DSlicer	2
L'utilisateur peut importer et visualiser des donnés volumétriques	2
L'utilisateur peut effectuer une coupe sur une donnée volumétrique, selon un des 3 axes (x, y, z)	2
L'utilisateur peut dessiner sur le modèle 3D présent dans la scène	1
L'utilisateur pourra segmenter le modèle	1
La visualisation d'un même objet peut se faire avec plusieurs casque en même temps, de sorte que cela permette le travail collaboratif	1

TABLE 4 – User Stories correspondant à l'application permettant de visualiser des données issues de 3DSlicer.

III. Choix logiciels

À la demande du client, lors de ce projet nous devions travailler en deux phases très distinctes. Comme le montre le Gantt prévisionnel (cf. partie V), nous devions commencer par faire du prototypage et de la prise en main sur Unity pour ensuite passer sur Android Studio pour concevoir notre logiciel.

La phase "Unity" devait permettre de prendre en main le casque et les techniques de programmation sur ce dernier. Ainsi, l'utilisation d'Unity[33] nous a permis d'accéder à un certain nombre de ressources rendant la programmation plus accessible. De plus, dans l'idée de rendre notre application la plus compatible possible, nous avons décidé d'utiliser l'interface de programmation d'application (API) libre de droit `OpenXR`[21] permettant de gérer la communication entre l'application et un grand nombre de matériel de réalité mixte. Ce choix nous a permis de rendre notre application compatible avec un pannel ne se résumant pas uniquement aux casques Meta Quest 3 (comme il nous l'était demandé). En ce qui concerne l'importation des objets, il était d'abord nécessaire de pouvoir charger des données depuis l'application utilisée sur le casque. Pour ce faire, nous avons utilisé des fonctionnalités intégrées à Unity pour la communication via URL, et modifié la bibliothèque `C# SMBLibrary` [12] permettant d'établir une connection avec un serveur SaMBa. Ensuite, nous avons réalisé un lecteur de fichier au format `OBJ` en se basant notamment sur un parser de fichier `OBJ` issu d'un package nommé `Runtime OBJ Importer`[34] qui nous permet la création de *Mesh*[31] lisible par Unity. Il restait alors à rendre manipulable, en réalité mixte, les maillages générés. Pour se faire nous avons travaillé avec le framework `XR Interaction Toolkit` (`XRI`)[15] compatible avec `OpenXR` et permettant de manipuler des objets 3D dans une scène Unity. Enfin, nous avons développé une interface graphique permettant à l'utilisateur de choisir quel fichier `OBJ` il souhaitait visualiser dans le casque. Et pour perfectionner cette interface nous avons aussi fait appel à des assets tels que `Animated Loading Icons`[11] ou encore `MRTK-Keyboard`[5] pour pouvoir ajouter un clavier virtuel par exemple.

Après quatre semaines de développement sur cette phase, nous avons commencé à réfléchir à la seconde phase : travailler sur Android Studio[3] afin de reproduire le viewer et créer une extension de 3DSlicer pour visualiser sur le casque les modèles 3D réalisés avec le logiciel. En commençant les recherches sur cette partie nous avons choisi de changer de stratégie : en effet, `OpenXR` avec Android Studio exige de travailler en `C++`. Nous avons alors pensé que se lancer dans un tel projet (refaire le viewer et faire une extensions pour 3DSlicer) serait difficilement réalisable en `C++` dans le délai imparti d'autant que nous souhaitions/devions produire un code propre, maintenable et extensible. Pour garantir un minimum de résultats nous avons choisi de rester sur Unity pour, d'une part continuer de "perfectionner" l'application viewer, et d'autre part créer une "extension" 3DSlicer. Nous rappelons que SlicerVR n'est pas une extension satisfaisante car, au delà des limites d'utilisation que nous avons rencontrées lors de nos tests, elle ne répond pas à la contrainte d'être disponible sur Linux, Mac et Windows.

Durant la période de transition nous avons cherché comment établir une "connexion" pour communiquer les données depuis 3DSlicer vers le casque. Nos recherches nous ont amené à découvrir le protocole réseau `OpenIGTLink`[32]. Dans l'état de l'art nous discutions de SlicerVR qui utilisait SteamVR pour faire le lien 3DSlicer-Casque et `OpenIGTLink` pour le lien entre collaborateur. Nous nous sommes alors demandé s'il été possible d'utiliser ce dernier protocole pour faire directement le

lien 3DSlicer-Casque. Nous avons alors trouvé le projet `AR Planner - Unity`[25] qui nous a montré qu'une telle communication était possible, validant ainsi notre choix d'utiliser `OpenIGTLink`.

D'après la documentation d'`OpenIGTLink`[32], `OpenIGTLink` est un "protocole réseau à double sens et d'égal à égal ouvert, simple et extensible pour la Thérapie Guidée par l'Image (IGT)[14]. Il est né d'une collaboration de partenaires universitaires, cliniques et industriels dans le développement d'un système robotique intégré pour les interventions sur la prostate guidées par IRM. Il a été conçu pour être utilisé dans la couche application sur la pile TCP/IP, tout en permettant aux développeurs de l'implémenter pour d'autres modèles de réseau." Il existe déjà une extension sur 3DSlicer qui permet, entre autre, de créer un serveur coté 3DSlicer et d'envoyer des données liées à la scène de travail courante. Cette extension se nomme `OpenIGTLink IF`[29] et donne accès à une interface permettant le genre de manipulations évoquées plus haut.

D'un point de vu plus technique `OpenIGTLink`[32] permet de faire transiter des données de différents types tels que (`IMAGE`, `POSITION`, `STATUS`, `...`) mais c'est le type `POLYDATA` qui nous intéresse tout particulièrement puisqu'il doit nous permettre de transférer la géométrie de modèles construit par l'utilisateur dans une scène de 3DSlicer. De plus l'encodage des données au format `POLYDATA` suit ce qui est fait dans la classe éponyme de `VTK`. Enfin, l'aspect collaboratif qu'il permet fait aussi partie des éléments qui nous ont poussés à nous tourner vers cette solution technique.

Au final, nous proposons deux applications distinctes : un lecteur en XR de fichiers `OBJ`[7] et un client XR d'`OpenIGTLink`[6], que nous avons développés avec la version (LTS) 2022.3 de Unity[30].

IV. Architecture

Notre travail se divise en deux sous-projets [7][6] ayant une base de code commune. Cette base inclut l'environnement `OpenXR` et des outils d'interaction qui facilitent l'utilisation en réalité mixte (XR), incluant la prise en charge des contrôleurs. Une interface utilisateur standard, offrant des options telles que le choix des modes d'affichage, un système de log et un clavier virtuel, est également partagée. Cette approche permet une gestion plus efficace des maillages 3D, y compris leur suppression via les manettes. Le diagramme de la figure 1 représente les parties communes entre les projets.

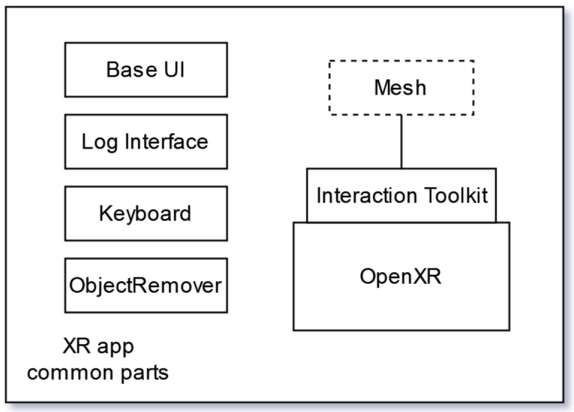


Figure 1 – Diagramme des parties communes entre les projets

Notre code se trouve dans le répertoire `Assets/Script/Script` du projet. Les bibliothèques importées via la plate-forme Unity se trouve dans le dossier `Assets/`, et les autres bibliothèques dans `Assets/Script`.

Lecteur de fichiers OBJ

Ce sous-projet permet aux utilisateurs de charger des fichiers `OBJ` depuis un serveur SaMBa ou directement depuis internet via une adresse URL. Le processus de chargement s'effectue grâce au module `Obj Importer` qui créé un maillage Unity à partir des données chargées.

Pour Samba, nous utilisons la bibliothèque `SMBLibrary`, et pour les sources web, c'est la bibliothèque `Networking` d'Unity qui est mise à contribution. Ces choix technologiques permettent une récupération des données, quelles que soient leurs origines. Le diagramme 2 représente la structure générale de ce projet, les choix de source et le parcours des données au sein de l'application.

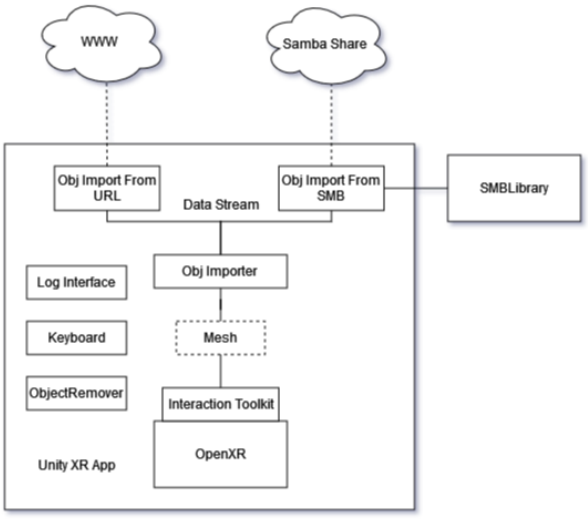


FIGURE 2 – Diagramme de structure du projet de lecteur d'OBJ

Client OpenIGTLink

Le deuxième sous-projet est un client pour le protocole `OpenIGTLink`. Il permet à l'application de se connecter à un serveur `OpenIGTLink` pour recevoir des modèles 3D au format `POLYDATA`, ainsi que pour échanger des informations de transformation des objets. L'utilisateur doit entrer manuellement les coordonnées du serveur pour établir la connexion, permettant ainsi l'affichage et la manipulation des maillages 3D obtenus. Le diagramme 3 généralise la structure de notre application, pour l'échange d'information entre le client et le serveur, et pour le procédé par lequel passe cette information pour finir affichée sous forme de maillage interactif.

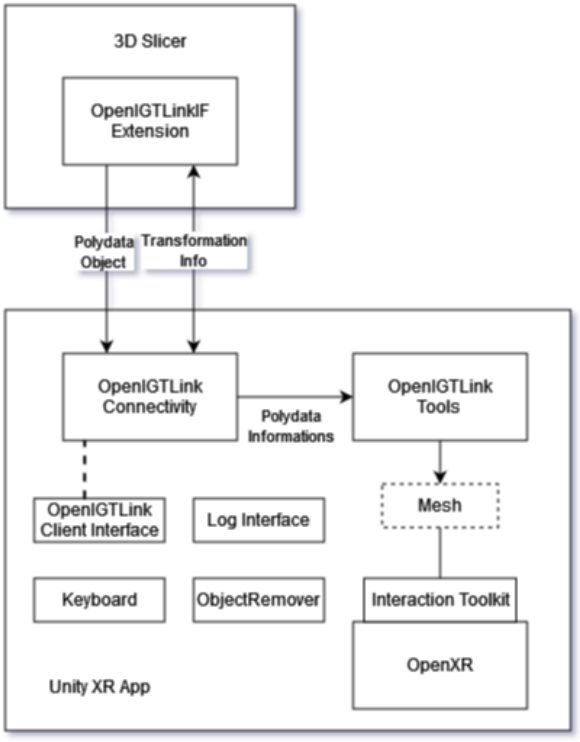


Figure 3 – Diagramme de structure du projet du client OpenIGTLink

V. Gantt prévisionnel

Au début de notre projet nous avons créé un Gantt prévisionnel, disponible sur la page suivante. Bien que le plus parlant soit probablement la comparaison avec le Gantt effectif qui est faite plus loin dans le rapport. Ce Gantt met bien en évidence le fait qu'au départ nous n'avions pas d'idée précise ni de connaissance concernant la partie "3DSlicer", que nous appelions alors "Android Studio". En effet, nous pouvons voir l'absence de détails dans cette phase avec l'utilisation du terme "itération".

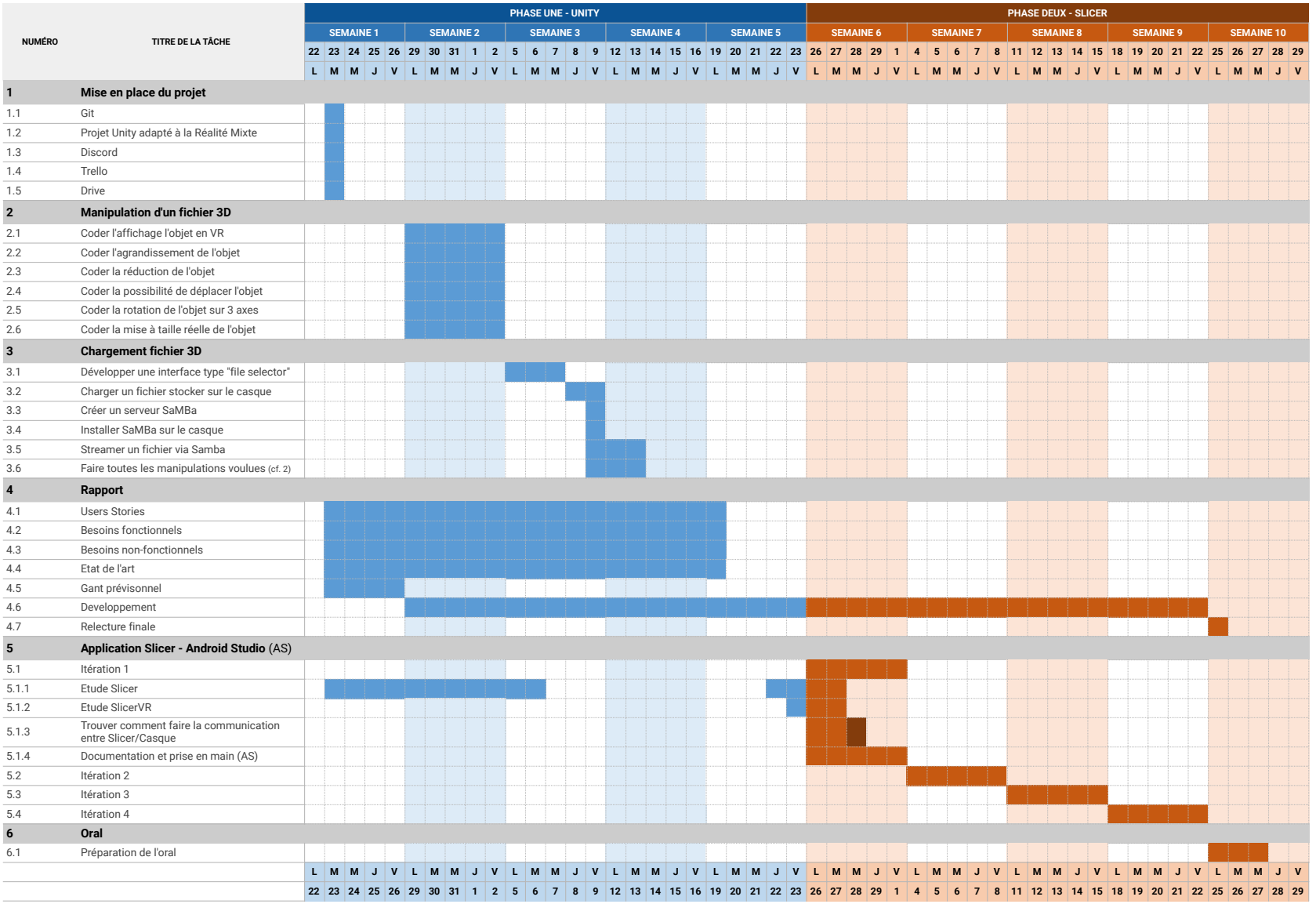
DIAGRAMME DE GANTT PREVISIONNEL

TITRE DU PROJET - Réalité mixte et imagerie médicale

NOM DE L'ENTREPRISE Medimixte

DATE 12/01/24

NUMÉRO	TITRE DE LA TÂCHE
1	**Mise en place du projet**
1.1	Git
1.2	Projet Unity adapté à la Réalité Mixte
1.3	Discord
1.4	Trello
1.5	Drive
2	**Manipulation d'un fichier 3D**
2.1	Coder l'affichage l'objet en VR
2.2	Coder l'agrandissement de l'objet
2.3	Coder la réduction de l'objet
2.4	Coder la possibilité de déplacer l'objet
2.5	Coder la rotation de l'objet sur 3 axes
2.6	Coder la mise à taille réelle de l'objet
3	**Chargement fichier 3D**
3.1	Développer une interface type "file selector"
3.2	Charger un fichier stocker sur le casque
3.3	Créer un serveur SaMBa
3.4	Installer SaMBa sur le casque
3.5	Streamer un fichier via Samba
3.6	Faire toutes les manipulations voulues (cf. 2)
4	**Rapport**
4.1	Users Stories
4.2	Besoins fonctionnels
4.3	Besoins non-fonctionnels
4.4	Etat de l'art
4.5	Gant prévisonnel
4.6	Developpement
4.7	Relecture finale
5	**Application Slicer - Android Studio** (AS)
5.1	Itération 1
5.1.1	Etude Slicer
5.1.2	Etude SlicerVR
5.1.3	Trouver comment faire la communication entre Slicer/Casque
5.1.4	Documentation et prise en main (AS)
5.2	Itération 2
5.3	Itération 3
5.4	Itération 4
6	**Oral**
6.1	Préparation de l'oral



VI. Réalisation

Lecteur de fichier OBJ

Implémentation

La première étape de cette partie de lecture et visualisation de fichier `OBJ` dans un environnement en réalité mixte, a été de mettre en place une base d'application en XR. Nous avons effectivement commencé par créer un projet Unity qui utilise `OpenXR`, et installé la bibliothèque `XR Interaction Toolkit` servant à la manipulation et l'interaction en réalité mixte dans le moteur de rendu Unity. Ces bibliothèques sont nécessaires pour que le matériel de réalité mixte qui nous était fourni (Meta Quest 3, mais aussi d'autres grâce à `OpenXR`) interagisse avec la scène de notre application.

Ensuite, nous nous sommes penchés sur l'interaction des objets, pour se faire nous avons utilisé la bibliothèque `XR Interaction Toolkit` pour configurer les interactions possibles. Nous avons d'abord implémenté la manipulation d'un cube, en permettant à l'utilisateur de l'attraper en maintenant les gâchettes intérieurs des manettes. Une fois l'objet attrapé, en bougeant la manette il est possible de le déplacer sur une sphère centrée sur la manette. Il est également possible de l'éloigner ou le rapprocher de la manette en bougeant le joystick droit vers l'avant ou l'arrière. Par ailleurs, l'objet peut être pivoté sur lui même horizontalement en bougeant le joystick droit vers la droite ou la gauche. De plus, en attrapant l'objet avec les deux manettes, il est possible de l'agrandir en éloignant les manettes, ou de le réduire en rapprochant les manettes. Lorsqu'un objet est saisi par l'utilisateur, il peut le supprimer en appuyant sur la touche 'X'. Enfin, toutes les fonctionnalités énoncées ci-dessus ne nécessitant pas d'utiliser les joystick ou le touche 'X' sont également accessibles en utilisant les mains comme contrôlers.

Le tableau 5, situé plus bas, permet de faire le lien entre des figures de ce rapport et les *User Stories* qu'elles illustrent.

Une autre partie cruciale dans ce lecteur de fichier est la fonctionnalité d'importation des objets. Pour interpréter les fichiers `OBJ` et créer un maillage utilisable par Unity, nous avons utilisé la bibliothèque `Runtime OBJ Importer`. Pour charger les fichiers dans l'application, nous avons d'abord implémenté l'importation via URL qui utilise des fonctionnalités de communication web natif à Unity. Aussi, nous avons rendu cette fonctionnalité asynchrone afin que le chargement du fichier (aussi long soit-il) ne puisse pas bloquer l'application, qui doit avant tout rester interactive. Ensuite, nous avons ajouté la possibilité de charger des fichiers `OBJ` depuis un serveur SaMBa. Pour se faire, nous nous sommes essentiellement basés sur la bibliothèque `SMBLibrary` permettant la communication avec un serveur SMB en `C#`, que nous avons en partie dû rendre asynchrone pour les mêmes raisons que citées précédemment. Cette fonctionnalité d'import de fichier via serveur SMB n'a été testée qu'avec des serveurs autorisant les "invités". Enfin, lorsque les objets sont importés dans la scène, ils sont positionnés à une coordonnée précise proche de l'utilisateur. Si le fichier `OBJ` importé spécifie plusieurs objets, ils seront placés de sorte que leurs positions et tailles relatives soient respectées, comme illustré dans la figure 14.

Pour conclure, en ce qui concerne l'importation des données dans l'application, il est possible, à partir

d'une URL ou d'informations concernant un fichier stocké sur un serveur SMB, de télécharger le fichier OBJ puis de l'importer dans la scène Unity. Chaque objet 3D importé dans la scène obtient également les attributs permettant à l'utilisateur d'interagir avec celui-ci.

Afin de rendre cette application plus accessible, un certain travail a été fourni quant aux interfaces utilisateurs. Inspirées des interfaces issues du projet *template* XR de Unity, les interfaces se veulent simples tout en s'intégrant dans un environnement en réalité mixte comme illustré dans les figures 4, 5 et 6. En étant semi-transparentes, toutes les interfaces permettent de voir son environnement tout en restant facilement lisibles. Pour rentrer plus en détail, deux types d'interfaces sont présentes, d'abord les interfaces de saisi ou d'information ancrées dans la scène, puis l'interface de menu rapide accessible lorsque l'on regarde la paume de sa main (également accessible lorsque l'utilisateur fait la même manipulation avec manette en main).

1. **Premier type d'interface**
   Ces interfaces peuvent être déplacées en bougeant une barre qui leur sert de point d'ancrage, ainsi l'utilisateur les place comme bon lui semble. Par ailleurs, les interfaces font toujours face à l'utilisateur, ce qui permet de ne pas avoir à réajuster en permanence leur orientation si l'utilisateur se déplace dans son environnement. De plus, si l'utilisateur veut saisir des informations dans les champs prévus à cet effet, il peut appuyer sur la gâchette droite ou gauche et un clavier va apparaître sous le point d'ancrage, illustré dans la figure 7. Il est alors possible pour l'utilisateur d'écrire via ce clavier pour rentrer les informations nécessaires. Une fois toutes les informations requises rentrées, l'utilisateur peut appuyer sur le bouton de validation pour commencer le chargement du fichier. Une animation de chargement se lance alors, et il devient impossible pour l'utilisateur d'importer de nouveau objet tant que le premier chargement est en cours. Enfin, une interface contenant les messages de débug est également disponible. Essentiellement utilisée à des fins de debugage, elle permet également à l'utilisateur d'accéder à un certains nombre d'informations, comme la taille des fichiers téléchargés, le nombre de sommets et faces des objets chargés, ou encore des informations liées à la connexion au serveur SMB ou à l'accès internet.
2. **Second type d'interface**
   Pour accéder à ce menu rapide il suffit que l'utilisateur regarde la paume de sa main droite ou gauche. Ce menu permet différents réglages sur l'utilisation de cette application. D'abord, c'est via ce menu que l'on peut choisir d'importer des objets via URL ou SMB, ce qui a pour effet d'afficher l'interface correspondante. Il est aussi possible d'afficher ou de retirer la fenêtre de debug. Ensuite, il est possible de choisir si l'on veut que l'application soit en AR ou VR, ce qui a pour effet d'effectuer une transition entre les deux modes. Par ailleurs, il est également possible de supprimer la totalité des objets qui ont été importés en appuyant sur un bouton de ce menu rapide. Enfin, lorsqu'un objet est importé dans la scène il est automatiquement redimensionné de sorte que sa dimension la plus grande (largeur, logueur ou hauteur) soit égale à 1m, afin que les objets soient de taille similaire et facilement manipulable. Cette fonctionnalité peut être activée ou non depuis le menu rapide. Si elle est désactivée, les objets seront importés avec leur taille originelle.

Pour terminer, certaines fonctionnalités n'ont pas été implémentées ou comportent encore des bugs.

C'est notamment le cas de la fonctionnalité visant à redimensionner les objets selon leur taille d'origine, qui donne des résultats incorrects lorsqu'on l'applique sur des objets qui ont été redimensionnés manuellement. Aussi, les téléchargements des fichiers ont été implémentés de sorte à ce qu'ils soient asynchrones, mais une fois l'objet chargé il est converti au format *Mesh* de Unity de façon synchrone, ce qui peut figer l'écran quelques instants lors de l'import d'objets volumineux. Parmi les fonctionnalités non implémentées, il y notamment le fait de pouvoir importer des maillages depuis les fichiers locaux du casque, car l'OS Android est assez restrictif à ce niveau et nous obligeait à décoder des Identifiants de Ressource Universel (URI), ce qui n'est pas trivial dans un environnement `C#`. De plus, cette fonctionnalité s'est finalement trouvée assez peu intéressante après discutions avec nos clients, puisqu'elle nécessite d'importer des fichiers de son ordinateur vers le casque en amont. Ceci ne fait que rajouter une étape dans le processus d'importation de fichiers dans l'application de lecteur de fichiers `OBJ`. Par ailleurs, par manque de temps nous n'avons pas implémenté les fonctionnalités sensées permettre la visualisation des maillages en fils de fer ou en nuage de points, ou encore la fonctionnalité visant à pouvoir dessiner sur l'objet. Enfin, lorsque l'on applique une rotation à un objet, un seul axe de rotation est disponible, et la rotation est centrée sur le point d'accroche de l'objet ce qui est assez contre intuitif. Nous avons essayé de corriger ce problème, mais il nécessite de réécrire en partie la bibliothèque `XR Interaction Toolkit`, ce que nous n'avons pas eu le temps de faire.

Résultats et screenshots

Voici quelques screenshots illustrant les interfaces présentes dans l'application ainsi que quelques captures illustrant ce qu'il est possible de faire et les propos tenus plus haut :

Figure 4 – Interface permettant de charger un objet depuis un serveur SMB.

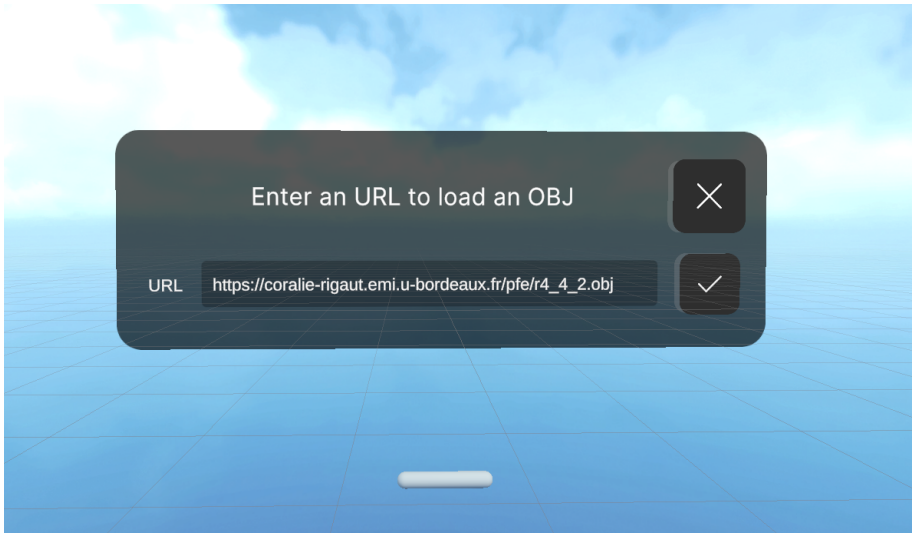


FIGURE 5 – Interface permettant de charger un objet depuis un URL.

FIGURE 6 – Interface du "menu rapide".

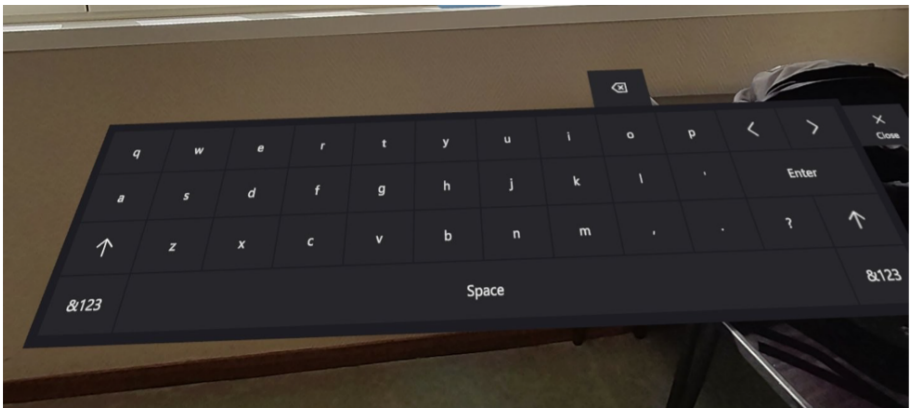


Figure 7 – Clavier virtuel.

Figure 8 – Série de trois captures montrant différents niveaux de "zoom" possibles sur maillage

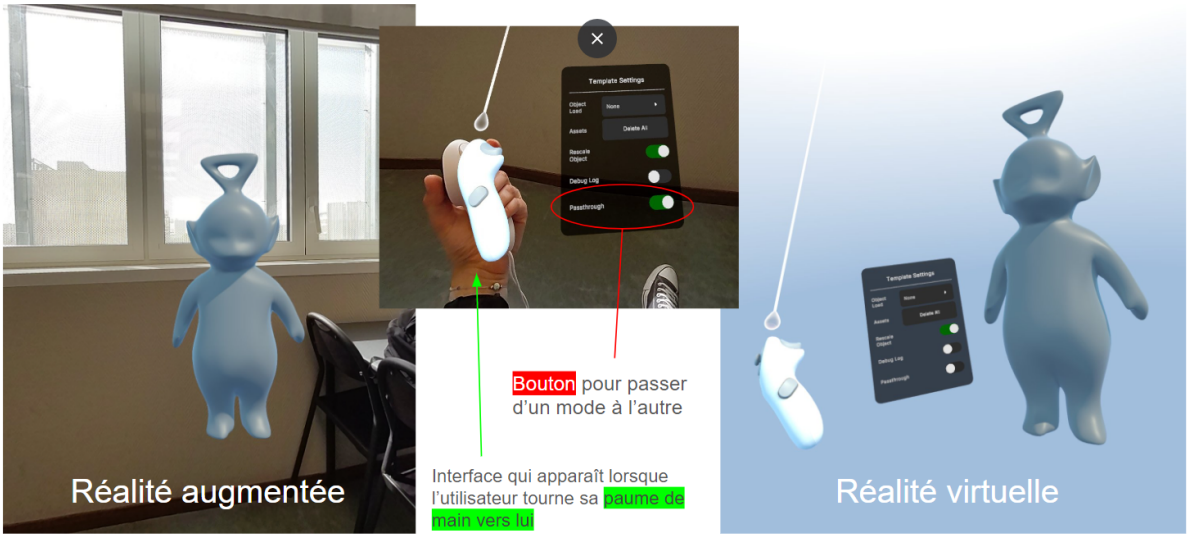


Figure 9 – Captures explicitant le passage de la AR à la VR et vice-versa

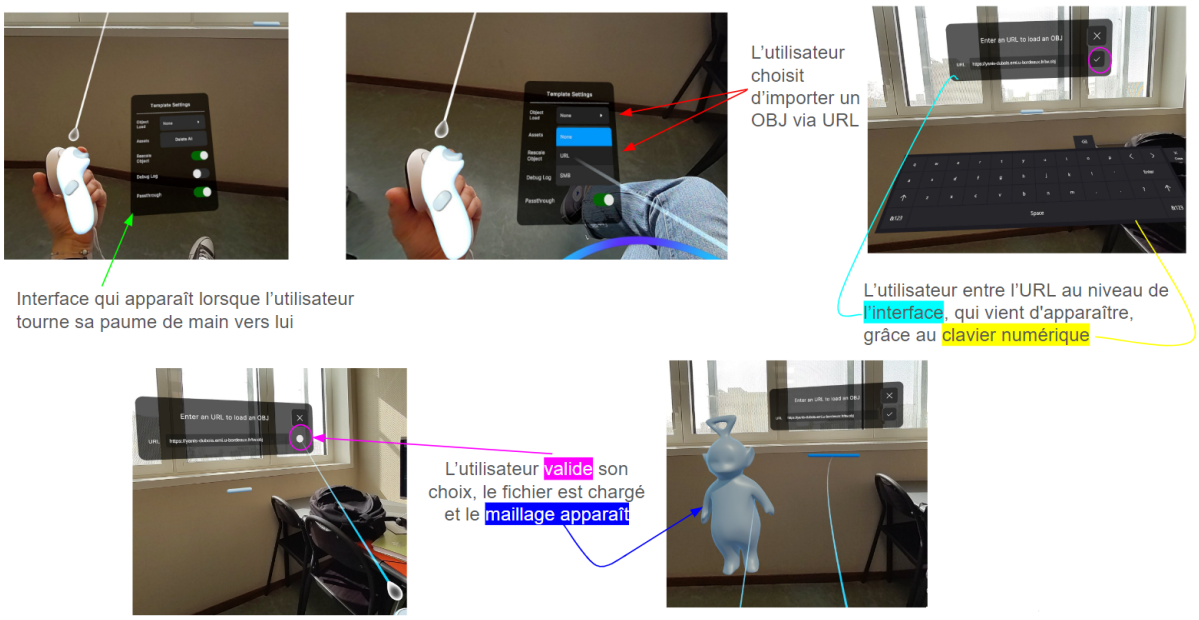


FIGURE 10 – Captures montrant les étapes pour importer un maillage 3D dans le casque en chargeant un fichier OBJ depuis une adresse URL

FIGURE 11 – Captures montrant les premières étapes pour importer un maillage 3D dans le casque en chargeant un fichier OBJ depuis un serveur SaMBa, les dernières étapes sont les mêmes que sur la figure 10

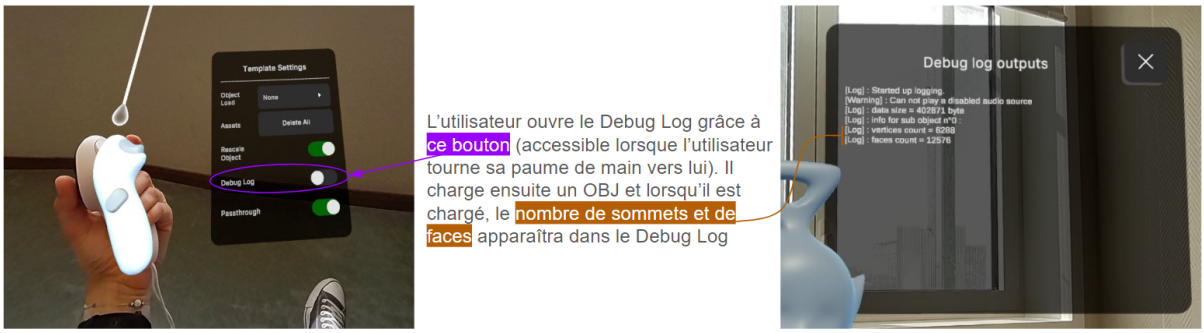


FIGURE 12 – Captures montrant l'affichage d'informations sur le maillage lorsque le Debug Log est ouvert au moment du chargement de ce dernier

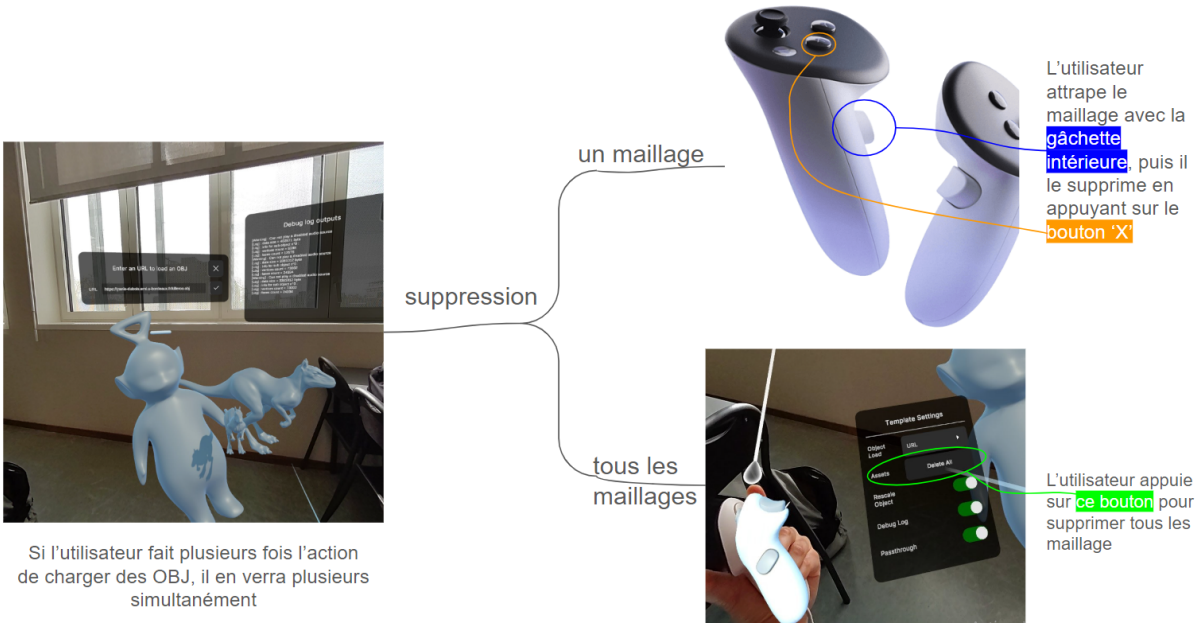


FIGURE 13 – Captures illustrant bien le fait que l'utilisateur peut afficher plusieurs maillages différents et expliquant ensuite les procédures de suppression d'un ou plusieurs maillages

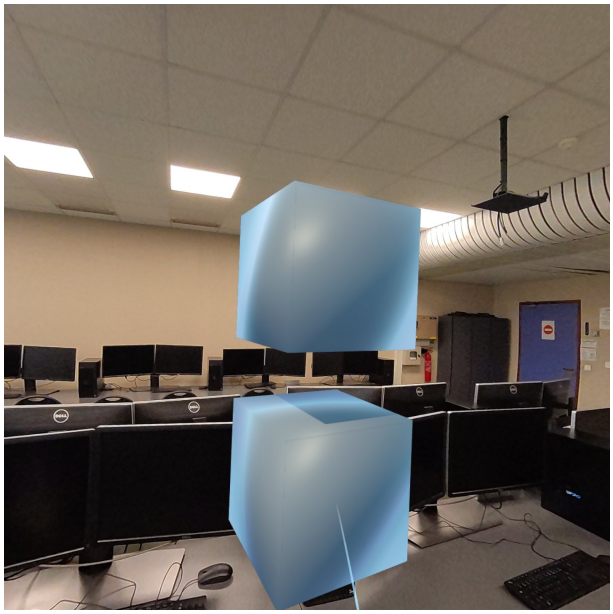


Figure 14 – Captures d'écran illustrant le fait que l'utilisateur peut charger des fichiers OBJ composé de plusieurs maillages

Le tableau 5 récapitule tout ce que nous venons de voir. En effet, la première colonne reprend les *User Stories* données dans le tableau 3. La deuxième colonne donne une indication de la faisabilité de l'action par l'utilisateur et suit la légende suivante :

- Le symbole ✓ signifie que l'utilisateur peut faire ce qu'il a demandé.
- Le symbole ∼ signifie que l'utilisateur peut partiellement faire ce qu'il a demandé.
- Le symbole X signifie que l'utilisateur ne peut pas faire ce qu'il a demandé.

Toutes les manipulations pour permettre à l'utilisateur de réaliser une action sont données plus haut, ces dernières sont aussi accompagnées de justifications et/ou explications de faisabilité ou non. Enfin, la troisième indique le ou les numéros de figures éventuels auxquels se référer pour voir des visuels en lien avec la *User story* donnée en première colonne.

User Story	Réalisée	Figure(s)
L'utilisateur peut visualiser un objet 3D dans un environnement de réalité mixte	✓	9
L'utilisateur peut tourner l'objet selon tous les axes de rotation	✓	-
L'utilisateur peut déplacer l'objet dans n'importe quelle direction	✓	-
L'utilisateur peut augmenter ou réduire la taille de l'objet 3D	✓	8
L'utilisateur peut afficher l'objet 3D à sa taille réelle	~	6
L'utilisateur peut importer un fichier local du casque	X	-
L'utilisateur peut importer un fichier depuis une URL	✓	5 et 10
L'utilisateur peut importer un fichier depuis un serveur SaMBa	✓	4 et 11
L'utilisateur peut choisir son mode d'importation via une interface graphique	✓	6, 10 et 11
L'utilisateur peut remplir les informations nécessaires à l'importation du fichier via une interface graphique selon le mode d'importation sélectionné	✓	10 et 11
L'utilisateur peut visualiser le nombre de sommets et de faces de l'objet importé	✓	12
L'utilisateur peut visualiser plusieurs objet 3D dans une même scène	✓	13
L'utilisateur peut supprimer un objet sélectionné de la scène	✓	13
L'utilisateur peut supprimer tous les objets de la scène	✓	13
L'utilisateur peut visualiser l'objet en nuage de points	X	-
L'utilisateur peut visualiser l'objet en fil de fer	X	-
L'utilisateur peut "dessiner" sur l'objet	X	-

TABLE 5 – Avancement User Stories liées à l'application de visualisation d'objet 3D type OBJ.

OpenIGTLink

Pour cette seconde phase, la demande était la suivante : proposer une extension au logiciel 3DSlicer permettant la visualisation d'examens d'imagerie médicale à l'aide d'un casque Meta Quest 3. Comme nous le disions précédemment, nous avons choisi de nous baser sur le protocole `OpenIGTLink` pour permettre cela. Ainsi, nous avons créé une application qui permet à l'utilisateur de se connecter en tant que client à un serveur de ce protocole pour recevoir des informations au format `POLYDATA`. La partie serveur est assurée par l'extension de 3DSlicer `OpenIGTLink IF`[29]. Notre application décode ces informations pour en faire un maillage qui sera manipulable en XR sur le casque.

Implémentation

Le point de départ du projet de client OpenIGTLink a été la récupération d'une structure de base de code en `C#`, initialement conçue pour une communication partielle via `OpenIGTLink` dans le contexte du projet AR Plannet Unity[28]. Cette base a constitué le fondement sur lequel nous avons construit et étendu nos fonctionnalités.

Une première étape clé a été la réécriture et l'optimisation de la classe de gestion de la connexion réseau. Grâce à l'utilisation des sockets via la bibliothèque native `C# System.Net.Sockets`, nous avons pu concevoir un système capable de gérer efficacement la lecture de messages de grande taille. Cette capacité est cruciale pour le traitement des données `POLYDATA`, qui peuvent atteindre plusieurs gigaoctets. En extrayant la taille du contenu à partir de l'en-tête des messages, notre méthode `Listen` dans le `SocketHandler` permet une récupération complète des données, essentielle pour leur traitement ultérieur par le système.

Nous avons ensuite procédé à une ré-implémentation des fonctions de connectivité, s'inspirant de ce qui avait été réalisé pour l'AR Planner Unity [25] mais en l'adaptant et en l'étendant pour répondre à nos besoins spécifiques de récupération de modèles 3D. La classe `OpenIGTLinkConnect` forme le noyau de notre architecture de communication. Elle gère la réception des informations et coordonne les méthodes de décodage des messages, notamment à travers la méthode `ListenSlicerInfo` qui orchestre la boucle d'écoute et le traitement des messages reçus.

Le décodage des messages, un aspect technique crucial de notre projet, a été réalisé au sein de la classe `ReadMessageFromServer`. Cette classe inclut des méthodes spécialisées dans la transformation des données brutes reçues en contenu structuré, prêtes à être utilisées par notre système.

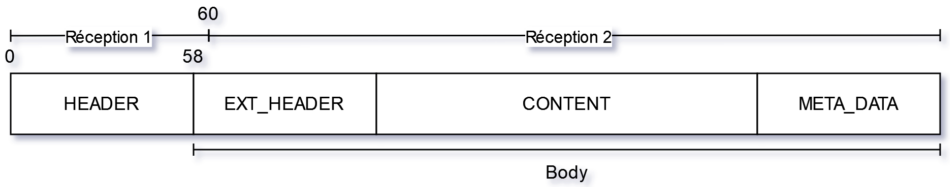


FIGURE 15 – Diagramme de la structure d'un message envoyé via le protocole OpenIGTLink

Afin de pouvoir décoder le message correctement, nous avons modifié la fonction de lecture de l'entête afin qu'elle soit en adéquation avec notre nouvelle boucle de lecture. L'entête, comme décrite dans la spécification du protocole, nous donne plusieurs informations utiles. Les informations qui nous intéressent particulièrement sont le type de message et la taille du corps du message, l'entête entière est décrite sur les 58 premiers octets du message (cf. figure 15).

De plus, le début du corps du message contient un champs qui est une entête étendue (cf. `EXT_HEADER` sur la figure 15). C'est pourquoi afin d'aller chercher directement les informations `CONTENT` du message nous commençons toujours par lire les 60 premiers octets du message. Ainsi ce que nous avons appelé la "Réception 1" sur la figure 15 nous permet de couvrir l'ensemble de l'entête (58 octets) et de récupérer le premier champ du `EXT_HEADER` (qui correspond à sa taille).

Finalement, en récupérant, à l'aide de la fonction `ReadHeaderInfo`, la taille du `HEADER` et du `EXT_HEADER` nous pouvons directement accéder aux données de la partie `CONTENT`, qui est là où se trouve le "cœur" du message.

Selon le type de message reçu, nous n'allons pas utiliser la même stratégie de décodage. En effet, nous avons introduit de nouvelles fonctionnalités pour la gestion des messages de type `STATUS` et `POLYDATA`.

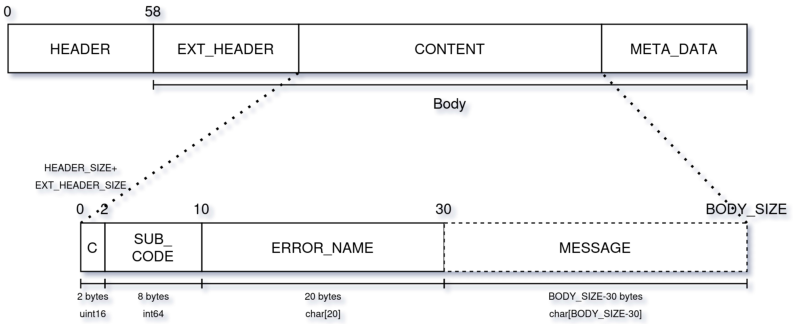


FIGURE 16 – Diagramme de la structure d'un message de type STATUS du protocole OpenIGTLink

L'implémentation des méthodes de lecture et de décodage pour les messages de type `STATUS` (cf. figure 16) constitue une partie fondamentale de notre système de communication via le protocole `OpenIGTLink`. Ces messages jouent un rôle crucial dans le contrôle de la qualité de la connexion et la gestion des erreurs, fournissant des retours essentiels sur l'état de la communication entre les dispositifs. Selon la spécification du protocole `OpenIGTLink`, un message de type `STATUS` se compose de quatre champs, chacun donnant des informations spécifiques concernant l'état de la communication :

- Un champ du *groupe du code d'erreur* de taille `UINT16` soit 2 octets (cf. `C` sur la figure 16)
- Un champ *sous-code* correspondant au code d'erreur de taille `INT64` soit 8 octets (cf. `SUB_CODE` sur la figure 16)
- Un champ correspondant au *nom du message* de taille **`char`**`[20]` soit 20 octets (cf. `ERROR_NAME` sur la figure 16)
- Un champ qui correspond à un tableau de **`char`** de taille variable par rapport à la taille du *contenu du message* (cf. `MESSAGE` sur la figure 16)

A noter qu'un message de type STATUS ne contient pas de partie META_DATA d'où le fait que sur la figure 16 le nombre d'octets visualisé à l'issue du champ MESSAGE correspond à la taille de BODY_SIZE
.

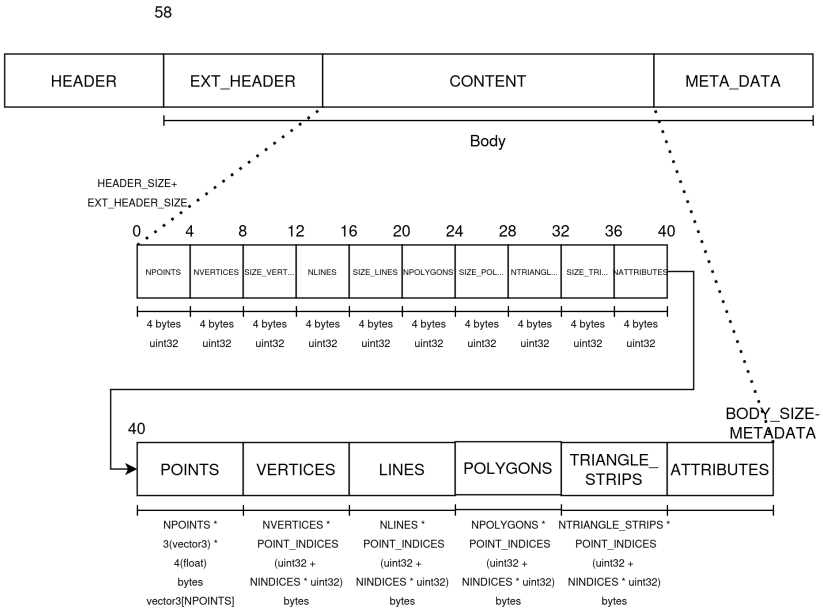


FIGURE 17 – Diagramme de la structure d'un message de type POLYDATA du protocole OpenIGTLink

L'intégration et le traitement des données de type POLYDATA ont nécessité la mise en place d'une structure de stockage adaptée, incarnée par la classe Polydata. Cette dernière contient toutes les informations nécessaires à la reconstruction d'un modèle 3D conformément au protocole OpenIGTLink. Les attributs qui ne sont pas nécessaires à la construction d'un maillage, comme par exemple les attributs de couleur, n'apparaissent pas dans cette classe et ne sont pas pris en compte.

Le message POLYDATA étant structuré en plusieurs champs (cf. figure 17) pour différents formats d'objets 3D (Points, Vertex, Lignes, Polygones, Triangle Strips), il est impératif de procéder étape par étape pour en extraire les informations utiles. Le décodage s'effectue en deux étapes : une pour la lecture du message brut reçu et une autre pour son décodage en structures de données utilisables.

1. **Lecture des données** : le début du message POLYDATA contient des informations sur la taille des différents champs (cf. POINTS, VERTICES, LINES, etc. sur la figure 17). En connaissant les tailles de chaque champs, notre méthode peut itérativement extraire chaque champ du flux de données brut, assurant ainsi que l'intégralité du message soit lue sans perte d'informations.
2. **Décodage en structures de données** : pour chaque champ lu, une méthode de décodage spécifique est appliquée.
   Les points, éléments fondamentaux dans la description d'objets 3D, sont convertis en Vector3, représentant leurs coordonnées dans l'espace 3D.
   Les autres éléments tels que VERTICES, LINES, POLYGONS, et TRIANGLE_STRIPS sont envoyés sous la forme de STRUCT_ARRAY dans le message POLYDATA. Cette structure 18 est une collection de POINT_INDICES, une sous-structure où la première valeur indique le

nombre N d'indices de points et où les valeurs suivantes correspondent aux N valeurs d'indice. Lors du décodage les STRUCT_ARRAY sont transformées en tableaux de structures Point_Indices. La structure Point_Indices est définie dans notre code pour être de format identique à celui décrit dans la spécification du protocole [8]. Elle est constitué d'un premier champ NINDICES de type UINT32 et d'une liste de taille NINDICES aussi de type UINT32.

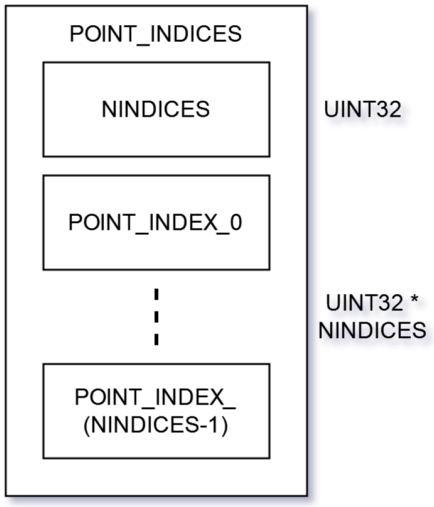


FIGURE 18 – Diagramme de la structure POINT_INDICES

Enfin, la transformation des données stockées dans les attributs de la classe Polydata en un maillage visible et interactif dans l'environnement Unity a été réalisée grâce à la classe PolydataToMesh.

Cette étape a impliqué la conversion des données structurées en un format compatible avec Unity, permettant l'affichage et l'interaction avec les modèles 3D dans une scène de réalité mixte. Le maillage Unity attend une liste de Vector3 pour les vertex, nous pouvons donc directement lui passer notre liste de points, et nous avons implémenté une fonction polygonsToTriangles qui convertit les polygones de la classe Polydata vers un tableau d'index des points comme attendu par l'attribut triangles de la classe Mesh du maillage Unity. Il est important de préciser que le format Polydata peut envoyer des polygones autres triangulaires, auquel cas la classe de maillage d'Unity ne peut pas les gérer. Finalement, l'ajustement de l'échelle des objets et leur préparation pour l'interaction ont été effectués en s'inspirant des méthodes utilisées dans le projet de visualiseur de fichiers au format OBJ.

Résultats et screenshots

Le tableau 6 suit exactement le même principe et la même légende que le tableau 5.

User Story	Réalisée
L'utilisateur peut charger et visualiser un modèle qui est un maillage issu de 3DSlicer dans le casque	✓
L'utilisateur peut tourner le modèle selon tous les axes de rotation	✓
L'utilisateur peut déplacer le modèle dans n'importe quelle direction	✓
L'utilisateur peut augmenter ou réduire la taille de l'objet 3D	✓
L'utilisateur peut afficher l'objet 3D à sa taille réelle	✓
L'utilisateur peut visualiser le nombre de sommets et de faces du maillage qu'il a importé	✓
L'utilisateur peut importer plusieurs modèles dans une même scène depuis 3DSlicer	~
L'utilisateur peut supprimer de la scène un modèle sélectionné	✓
L'utilisateur peut supprimer de la scène tous les modèles	✓
Le modèle visualisé dans le casque se met à jour s'il a été modifié sur 3DSlicer	X
L'utilisateur peut importer et visualiser des donnés volumétriques	~
L'utilisateur peut effectuer une coupe sur une donnée volumétrique, selon un des 3 axes (x, y, z)	X
L'utilisateur peut dessiner sur le modèle 3D présent dans la scène	X
L'utilisateur peut segmenter le modèle	X
La visualisation d'un même objet peut se faire avec plusieurs casques en même temps, de sorte que cela permette le travail collaboratif	X

TABLE 6 – Avancement User Stories liées à l'application de visualisation de modèle 3D venant de 3DSlicer.

Les figures 19 à 25 montrent un exemple d'utilisation que l'application permet.

La figure 27 montre que lors du chargement (avant qu'il ne change sa taille en le manipulant), l'utilisateur peut afficher l'objet 3D à sa taille réelle. En effet, le rapport de taille entre le maillage envoyé depuis le serveur `OpenIGTLink` de 3DSlicer et la taille réel est de 1000. C'est en modifiant les coordonnées lors de la création du *Mesh* que cela est rendu possible.

Si l'utilisateur souhaite importer plusieurs modèles dans une même scène depuis 3DSlicer. Il lui suffit d'envoyer plusieurs modèles en appuyant sur "SEND" (cf. figure 25).

Les *User Stories* suivantes :

- "Le modèle se met à jour dans l'application si il a été modifié sur 3DSlicer."
- "L'utilisateur peut importer et visualiser des donnés volumétrique."
- "L'utilisateur peut effectuer une coupe sur une donnée volumétrique, selon un des 3 axes (x, y, z)."
- "L'utilisateur peut dessiner sur le modèle 3D présent dans la scène."
- "L'utilisateur pourra segmenter le modèle."
- "La visualisation d'un même objet peut se faire avec plusieurs casque en même temps, de sorte que cela permette le travail collaboratif."

N'ont pas été réalisées par manque de temps mais sont théoriquement possibles grâce au protocole de communication `OpenIGTLink`.

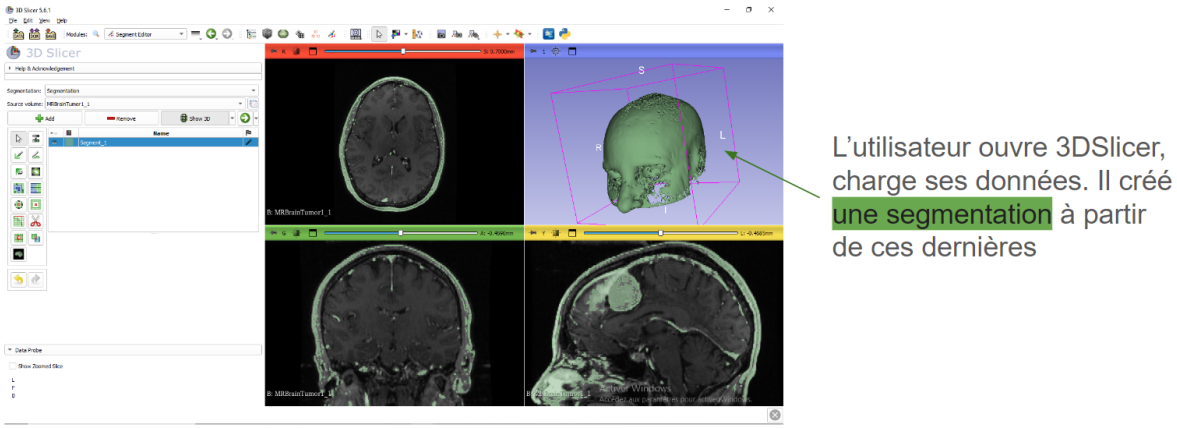


FIGURE 19 – Création d'une segmentation sur 3DSlicer

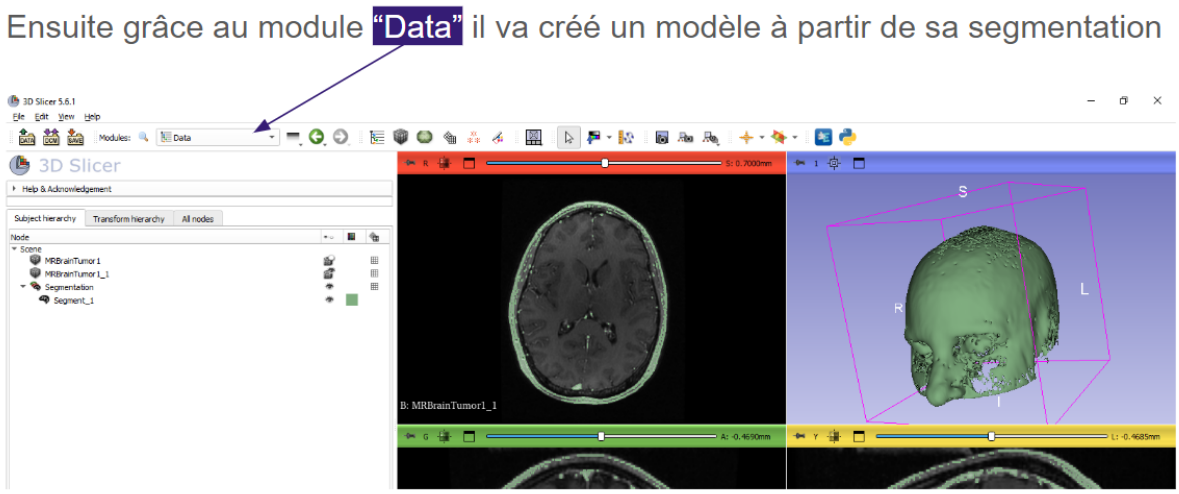


FIGURE 20 – Aperçu du module "Data" de 3DSlicer

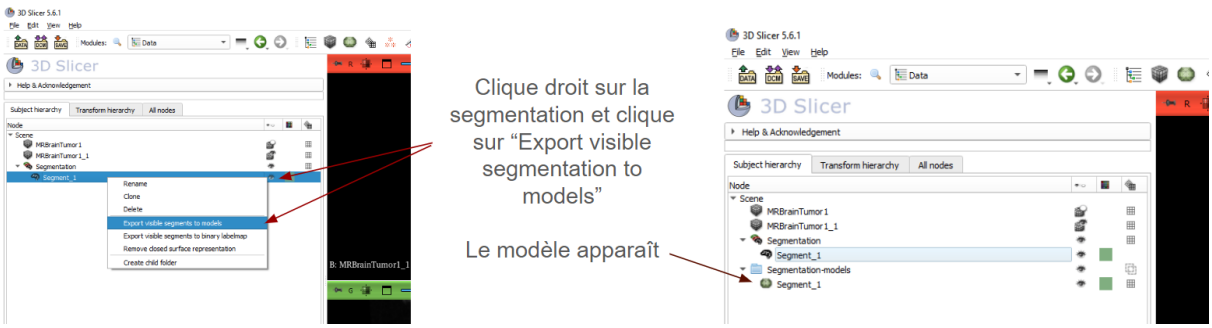


FIGURE 21 – Création d'un modèle 3D sur 3DSlicer à partir d'une segmentation

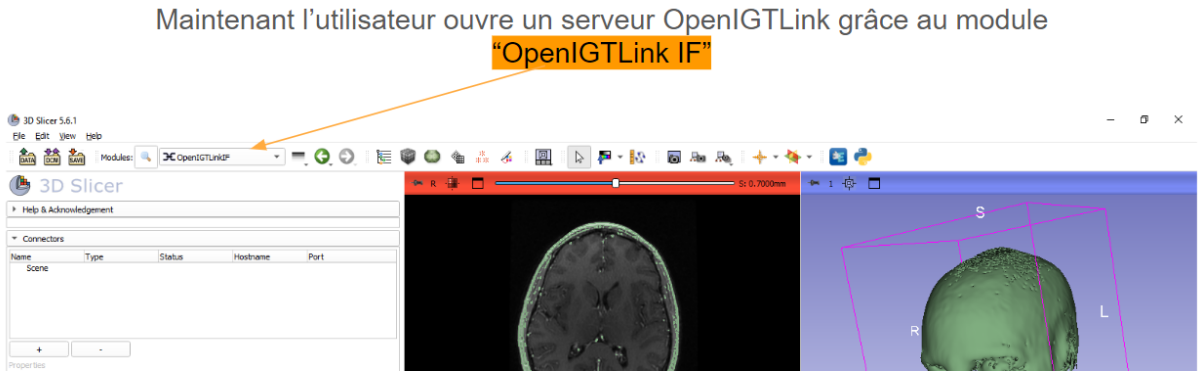


FIGURE 22 – Aperçu du module "OpenIGTLink IF" de 3DSlicer

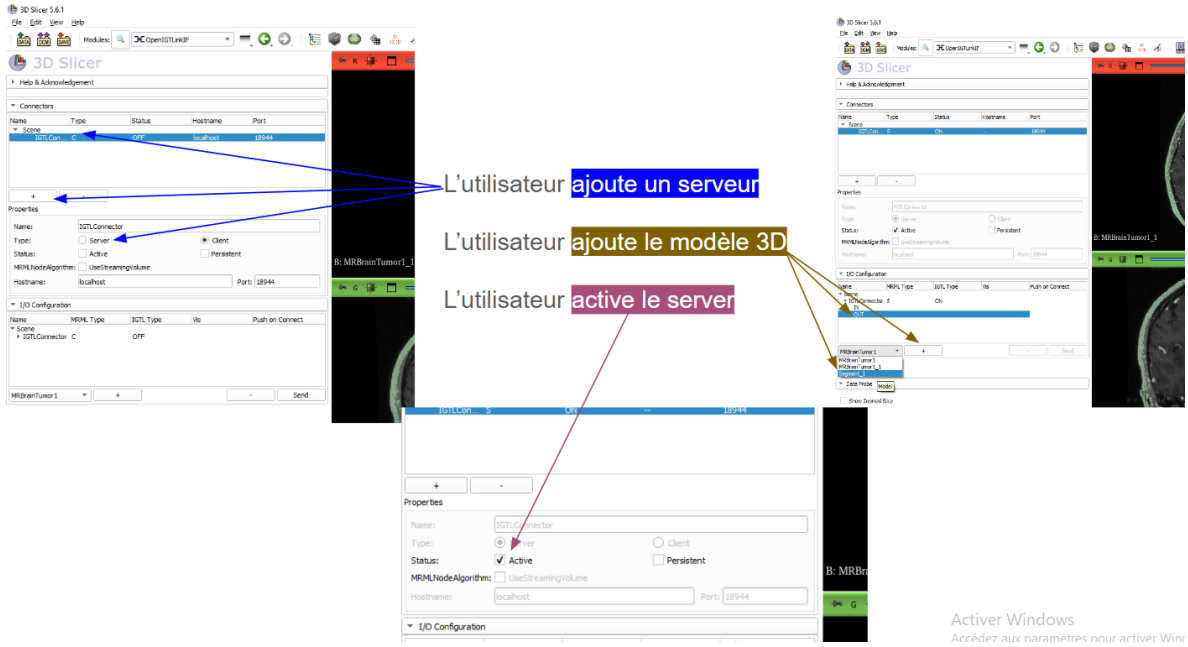


FIGURE 23 – Ouverture d'un serveur avec le module "OpenIGTLink IF" de 3DSlicer

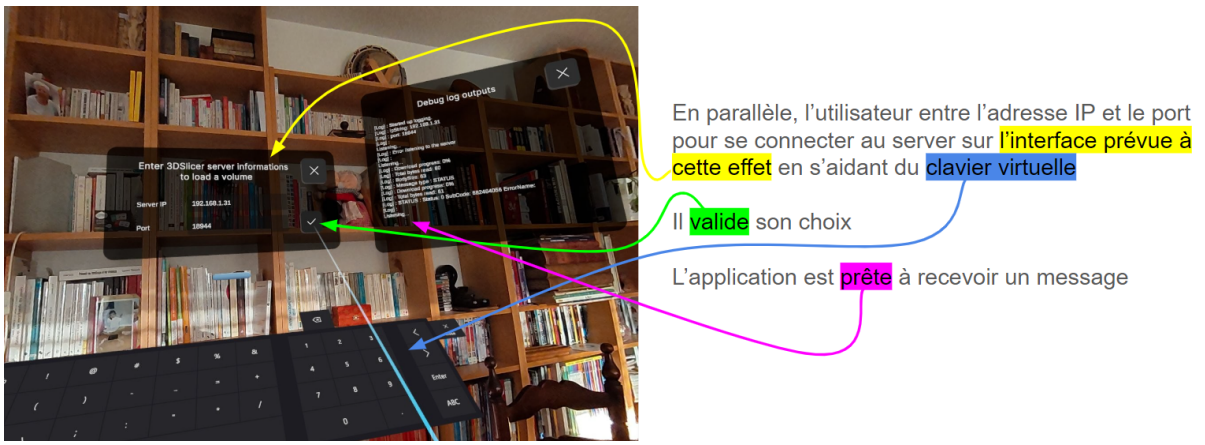


FIGURE 24 – Manipulations à faire pour pouvoir se connecter au serveur dans le casque

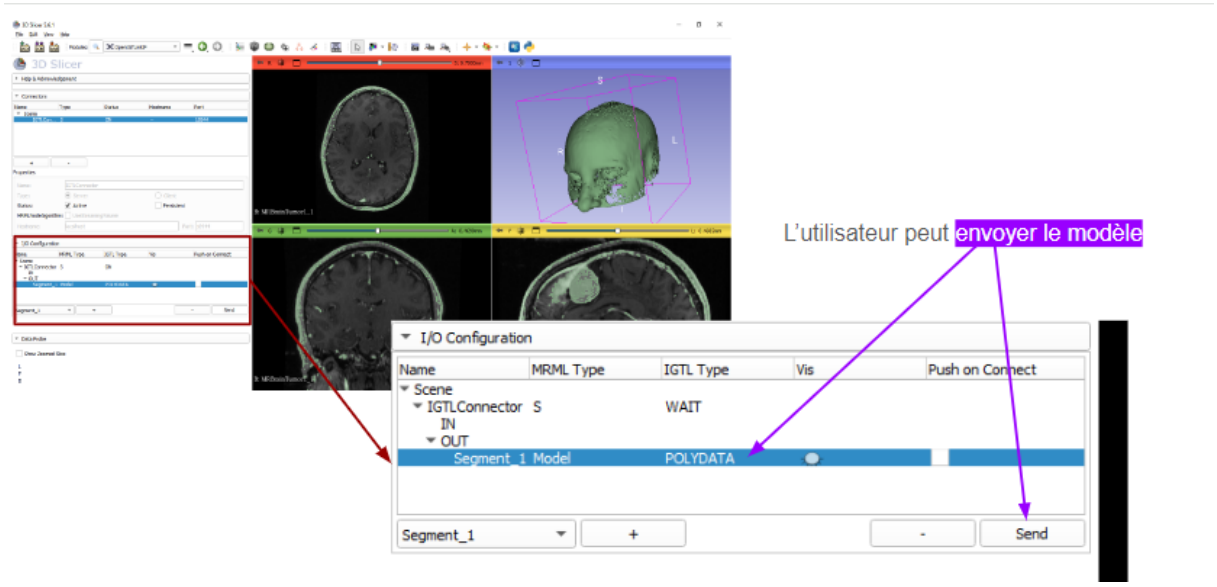


FIGURE 25 – Manipulations pour envoyer un modèle vers le casque connecté au serveur

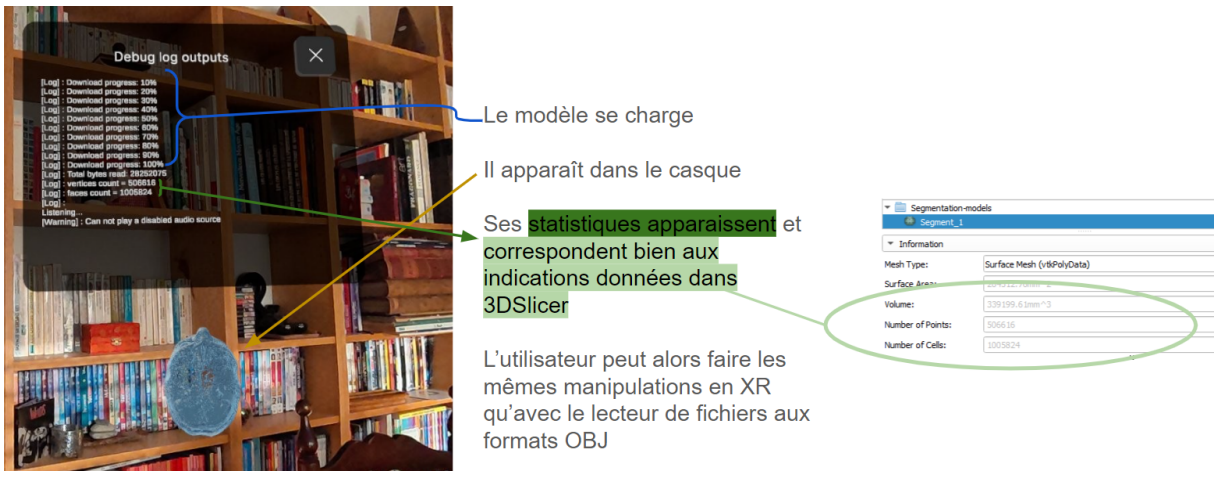


FIGURE 26 – Visualisation du modèle qui apparaît dans le casque

FIGURE 27 – Visualisation du modèle qui charge à taille réelle

VII. Tests

Protocole expérimental

Nous avons, pour notre projet, décidé de réaliser des tests de charge. L'objectif de ces tests est de connaître le nombre de FPS [6] du Meta Quest 3 en fonction du nombre de faces du maillage affichées dans le casque. Pour cette expérimentation nous avons utilisé un maillage avec plusieurs niveaux de résolutions. En effet, nous avons pris un maillage représentant une sphère à laquelle nous avons appliqué des subdivisions de faces successives. Ainsi, nous avons obtenu 28 sphères de dimensions et de formes similaires mais ayant des résolutions différentes (la plus petite et la plus grosse comportent respectivement 1280 et un peu plus de 8 300 000 faces).

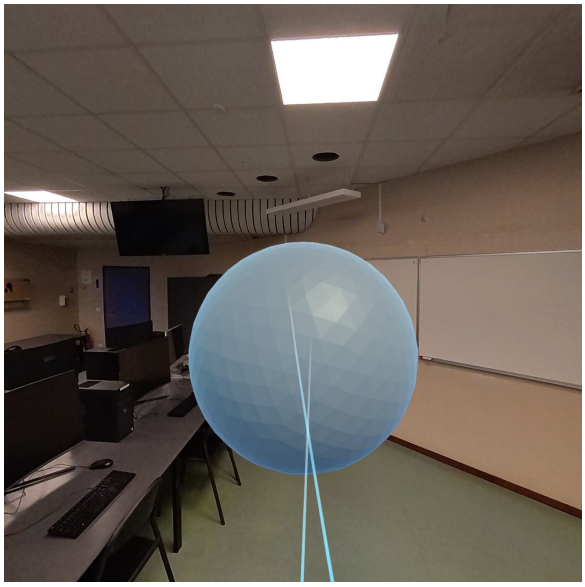


Figure 28 – Capture montrant un exemple de sphère que nous avons utilisé pour les tests

Grace à l'application OVR Metrics Tool[1], qui permet de récupérer différentes statistiques concernant les performances de l'application, nous avons notamment pu récupérer le nombre de FPS moyen. Pour obtenir la courbe du graphe 29 nous avons chargé les sphères une par une et une fois cela fait nous regardions le nombre de FPS donné par OVR Metrics Tool lorsque ces sphères étaient affiché.

6. Il faut noter que FPS est un acronyme qui vient de l'anglais et qui veut dire "Frame Per Second". Cela correspond aux nombres d'images qui sont affichées à l'écran par seconde.

Résultats et Discussion

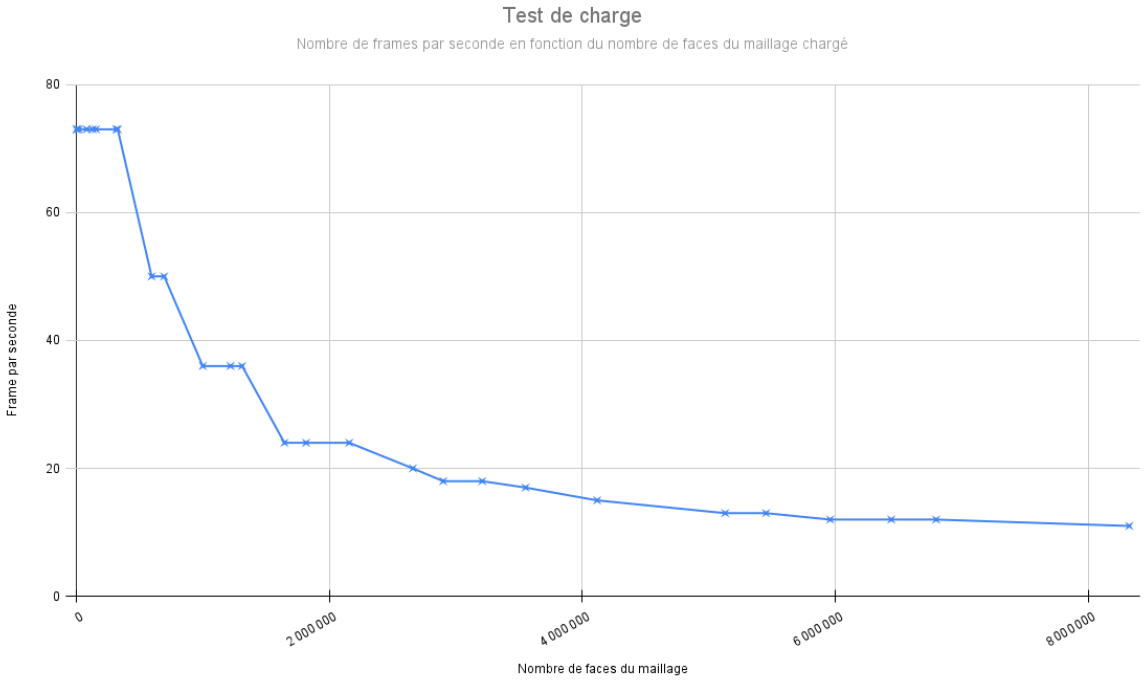


FIGURE 29 – Graphique montrant l'évolution des fps en fonction de l'évolution du nombre de faces

Généralement, pour une application interactive il est conseillé d'avoir au moins 60 FPS. Dans le cas de notre application, nous tombons sous ce seuil lorsque le maillage a plus de 500 000 faces. Toujours dans le cadre d'une application interactive il est possible de s'approcher d'une limite de 30 FPS. Mais il s'agit là d'une valeur minimale, si l'on tombe sous cette valeur le port du casque devient inconfortable (pour ne pas dire désagréable). Cette limite minimale est atteinte avec les maillages dont le nombre de faces apprivoise les 1 500 000. Nous trouvons ces performances satisfaisantes dans le cadre de notre application. Il est également important de noter que pour nos 3 derniers modèles, qui possèdent plus de 6 000 000 de faces, nous pouvons constater des crash de l'application lors du chargement de ceux-ci.

Nous n'avons qu'un test de charge alors que nous avons deux applications. Ce choix est volontaire et s'explique par le fait que toute la partie liée à l'affichage et la manipulation des maillages dans le casque est commune à nos deux applications. Cela signifie qu'une fois le maillage chargé en *Mesh* Unity les deux applications fonctionnent rigoureusement de la même manière. De plus faire un test de charge avec 3DSlicer s'avère assez compliqué : lorsque nous créons un modèle 3D à partir des fichiers DICOM sur ce logiciel il est difficile de contrôler le nombre de faces que le maillage créé a.

Enfin, il ne nous a pas semblé pertinent de réaliser de tests mesurant la vitesse de chargement de maillage car comme nous avons pu le constater durant nos diverses utilisations de nos applications ce temps est intrinsèquement lié à la qualité de la connexion Wi-Fi utilisée.

VIII. Projet : comparaison gantt prévisionnel/effectif

Premièrement, il est intéressant de noter que la décomposition générale du projet est très différente de celle prévue. Dans le diagramme de Gantt prévisionnel nous sommes partis du principe qu'il n'y aurait que deux parties dans le développement, la première phase sur Unity et la seconde phase sur 3DSlicer qui devait être sur Android Studio. Une grosse partie du temps de développement de la partie Unity sera finalement faite sur le chargement et l'affichage d'un maillage sur le casque et l'idée d'un "file selector" sera abandonnée à cause de contraintes matérielles pour laisser place à des alternatives réseau, comme le web. Le développement par Samba aura finalement duré plusieurs semaines au lieu de moins d'une semaine. L'ajout d'une phase de transition de deux semaines dans le diagramme de Gantt effectif nous montre une concentration sur l'aspect recherche d'une solution de communication et de choix logiciels. Finalement, la partie Slicer sera effectuée sur Unity, et le principal du travail sera effectué sur la communication réseau via le protocole OpenIGTLink.

DIAGRAMME DE GANTT EFFECTIF - partie 1/2

TITRE DU PROJET - Réalité mixte et imagerie médicale		**NOM DE L'ENTREPRISE**	Medimixte
		DATE	12/01/24

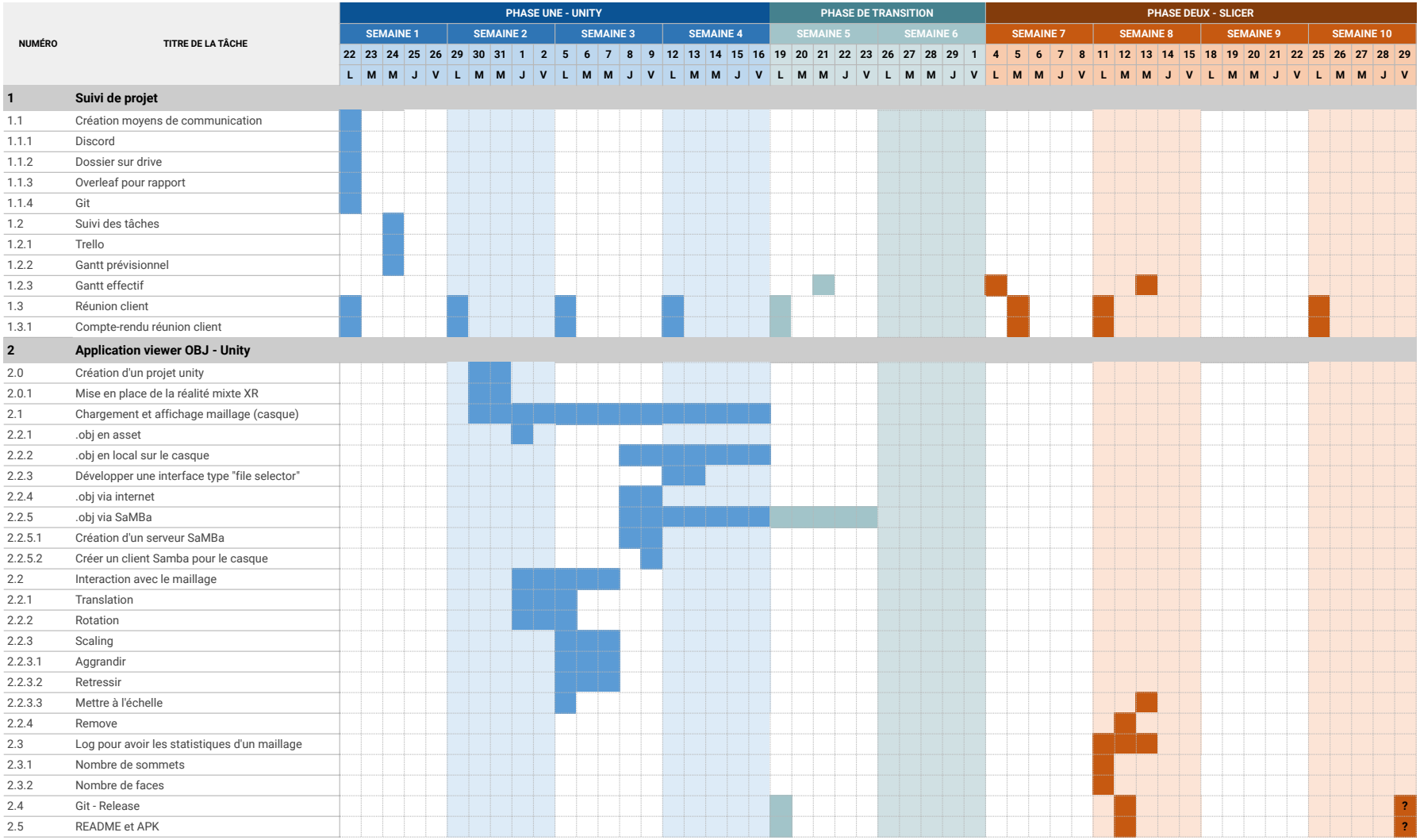
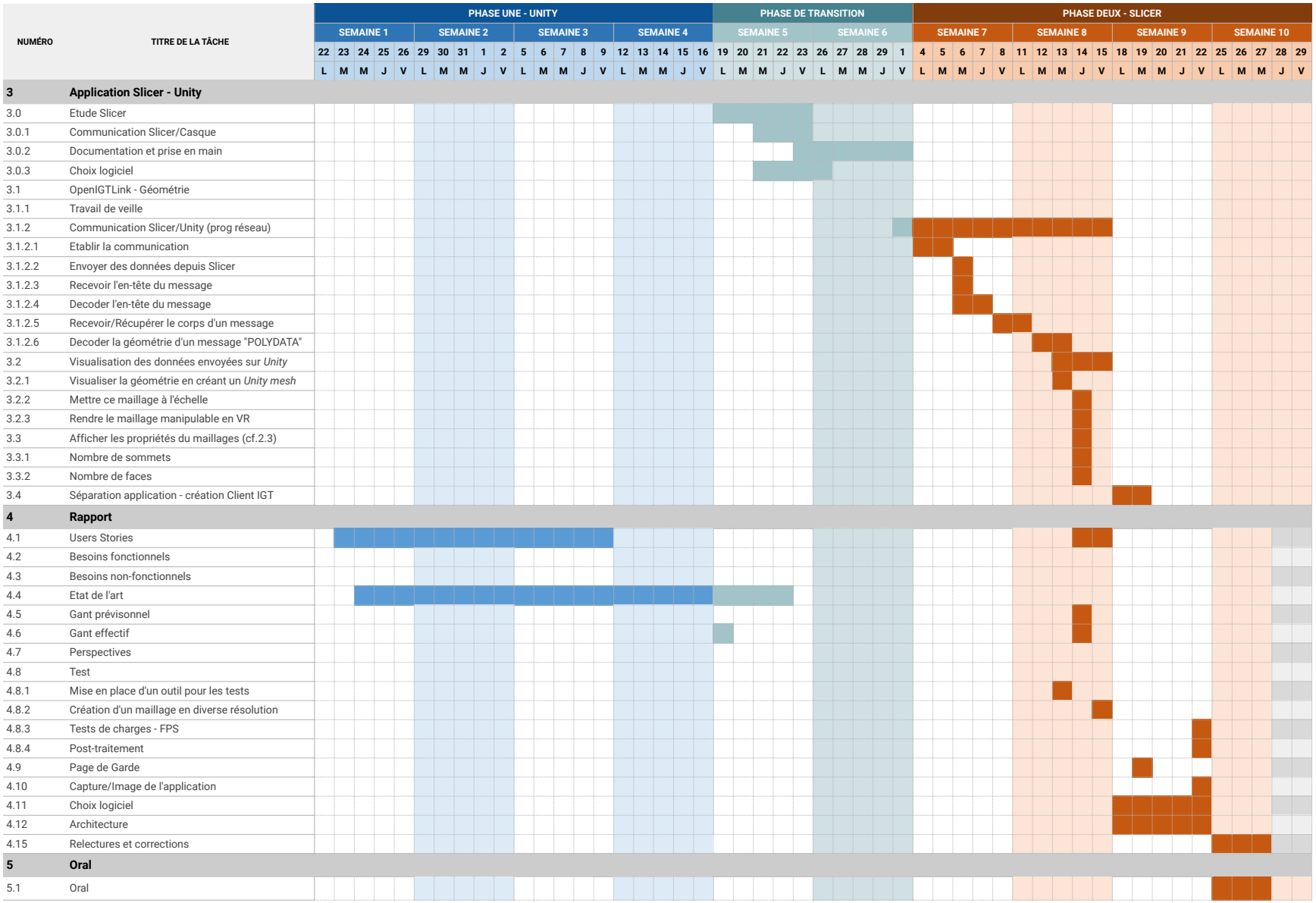


DIAGRAMME DE GANTT EFFECTIF - partie 2/2



NUMÉRO	TITRE DE LA TÂCHE
3	**Application Slicer - Unity**
3.0	Etude Slicer
3.0.1	Communication Slicer/Casque
3.0.2	Documentation et prise en main
3.0.3	Choix logiciel
3.1	OpenIGTLink - Géométrie
3.1.1	Travail de veille
3.1.2	Communication Slicer/Unity (prog réseau)
3.1.2.1	Etablir la communication
3.1.2.2	Envoyer des données depuis Slicer
3.1.2.3	Recevoir l'en-tête du message
3.1.2.4	Decoder l'en-tête du message
3.1.2.5	Recevoir/Récupérer le corps d'un message
3.1.2.6	Decoder la géométrie d'un message "POLYDATA"
3.2	Visualisation des données envoyées sur *Unity*
3.2.1	Visualiser la géométrie en créant un *Unity mesh*
3.2.2	Mettre ce maillage à l'échelle
3.2.3	Rendre le maillage manipulable en VR
3.3	Afficher les propriétés du maillages (cf.2.3)
3.3.1	Nombre de sommets
3.3.2	Nombre de faces
3.4	Séparation application - création Client IGT
4	**Rapport**
4.1	Users Stories
4.2	Besoins fonctionnels
4.3	Besoins non-fonctionnels
4.4	Etat de l'art
4.5	Gant prévisonnel
4.6	Gant effectif
4.7	Perspectives
4.8	Test
4.8.1	Mise en place d'un outil pour les tests
4.8.2	Création d'un maillage en diverse résolution
4.8.3	Tests de charges - FPS
4.8.4	Post-traitement
4.9	Page de Garde
4.10	Capture/Image de l'application
4.11	Choix logiciel
4.12	Architecture
4.15	Relectures et corrections
5	**Oral**
5.1	Oral

PHASE UNE - UNITY
SEMAINE 1 22 L 23 M 24 M 25 J 26 V
SEMAINE 2 29 L 30 M 31 M 1 J 2 V
SEMAINE 3 5 L 6 M 7 M 8 J 9 V
SEMAINE 4 12 L 13 M 14 M 15 J 16 V
PHASE DE TRANSITION
SEMAINE 5 19 L 20 M 21 M 22 J 23 V
SEMAINE 6 26 L 27 M 28 M 29 J 1 V
PHASE DEUX - SLICER
SEMAINE 7 4 L 5 M 6 M 7 J 8 V
SEMAINE 8 11 L 12 M 13 M 14 J 15 V
SEMAINE 9 18 L 19 M 20 M 21 J 22 V
SEMAINE 10 25 L 26 M 27 M 28 J 29 V

Conclusion

Dans son état actuel notre projet propose deux solutions qui partagent une base commune permettant l'interaction dans un environnement en réalité mixte via l'API `OpenXR`. L'interface utilisateur est modulable permettant ainsi de s'adapter au projet, un clavier virtuel est mis à disposition afin de pouvoir remplir les champs nécessaires et finalement, une interface de debug est activable.

La première partie du projet permet de visualiser les modèles 3D de type `OBJ` depuis plusieurs sources (Samba et URL), avec plusieurs types de manipulation possible comme la rotation, la translation et la modification de la taille.

La seconde partie du projet, le client OpenIGTLink permet la communication basique via le protocole avec un serveur. La réception de messages de type `STATUS` et de type `POLYDATA` est implémentée, ainsi que la réception et l'émission de messages de type `TRANSFORM`. Les messages de type `POLYDATA` génèrent un maillage visible et manipulable comme pour la première partie du projet.

Ces deux solutions permettent de répondre à une problématique de visualisation de modèle 3D dans un environnement améliorant considérablement la compréhension spatiale et la perception globale de l'objet. De plus, la communication avec 3DSlicer permet d'afficher des modèles 3D issus de fichiers d'imagerie médicale pouvant ainsi améliorer l'appréhension des structures anatomiques, le tout à échelle réelle.

Les deux solutions sont une réponse à un besoin émis par leurs communautés respectives. Il n'existe aucune application open-source de visualisation de fichier OBJ en réalité virtuelle ou augmentée, ce qui est désormais possible avec notre application. De plus, il existe une certaine demande de la communauté 3DSlicer pour la récupération et le décodage de message `POLYDATA` sur Unity du protocole `OpenIGTLink`.

Toutefois, le chemin vers des applications pleinement opérationnelles et polyvalentes révèle plusieurs avenues d'améliorations et d'expansions. Parmi les perspectives, plusieurs axes de développement se distinguent :

- Pour le lecteur de fichier OBJ :
  - Extension de la compatibilité de format de fichier : élargir la prise en charge à divers formats de fichiers (`OFF`, `STL`, ...) pour accroître la flexibilité et l'universalité de l'application de visualisation d'objet.
  - Importation de fichiers locaux : développer un parser de chemin spécifique à Android pour simplifier l'importation de fichiers dans l'application depuis les données du système.
- Pour le client OpenIGTLink :
  - Travail collaboratif grâce à `OpenIGTLink` : permettre à des utilisateurs de se connecter ensemble et de collaborer en temps réel sur une même visualisation.
  - Visualisation des coupes et volume rendering : exploiter les messages de type `IMAGEDATA` du protocole `OpenIGTLink` et modifier la visualisation simple du modèle 3D obtenu par message `POLYDATA` avec une technique de volume rendering.
- En commun entre les solutions :
  - Gestion améliorée des couleurs, matériaux et rendus : intégrer la prise en compte des fichiers

de matériaux (.mtl) accompagnant les modèles `OBJ` pour un rendu visuel plus fidèle. Et intégrer la prise en compte des `ATTRIBUTES` dans les messages `POLYDATA` pour améliorer le rendu de l'objet 3D.

— Intégration de l'occlusion ambiante (AO) : intégration de l'occlusion ambiante afin d'enrichir le détail du rendu.

— Optimisation de l'interaction utilisateur : ajuster les mécanismes de saisie, de rotation et d'exploration pour une manipulation plus intuitive et précise des objets 3D.

— Développement d'un journal de débogage déroulant : concevoir un système de log déroulant pour faciliter le suivi des événements.

Références

[1] Ovr metrics tool on meta quest. https://www.meta.com/fr-fr/experiences/2372625889463779/. [Accessed 26-03-2024].

[2] 3D Organon VR Anatomy. https://www.3dorganon.com/. Accès le 15-02-2024.

[3] Android Studio. https://developer.android.com/studio. Accès le 18-03-2024.

[4] Djamel Aouam, Nadia Zenati-Henda, Samir Benbelkacem, and Chafiaa Hamitouche. An Interactive VR System for Anatomy Training. In Branislav Sobota and Dragan Cvetković, editors, *Mixed Reality and Three-Dimensional Computer Graphics*, chapter 3. IntechOpen, Rijeka, 2020.

[5] Ayfel. Standalone version of MRTK NonNative Keyboard. https://github.com/Ayfel/MRTK-Keyboard. Accès le 22-03-2024.

[6] Florian Bernadet, Yanis Dubois, Maxime Oçafrain, and Coralie Rigaut. Client openigtlink en xr. https://github.com/Moustakick/OpenIGTLink-Client-Unity-XR. [Accessed 26-03-2024].

[7] Florian Bernadet, Yanis Dubois, Maxime Oçafrain, and Coralie Rigaut. Lecteur de fichier obj en xr. https://github.com/yanis-dubois/XR-Object-Visualizer. [Accessed 26-03-2024].

[8] The OpenIGTLink Community, Brigham, and Women's Hospital 2007-2022. Spécification du protocole openigtlink. http://openigtlink.org/developers/spec. [Accessed 27-03-2024].

[9] Paulo Dias, Ricardo Silva, Paula Amorim, Jorge Lains, Eulalia Roque, Inês Serôdio, Fatima Pereira, and Beatriz Sousa Santos. Using virtual reality to increase motivation in poststroke rehabilitation. *IEEE computer graphics and applications*, 39(1):64–70, 2019.

[10] Andriy Fedorov, Reinhard Beichel, Jayashree Kalpathy-Cramer, Julien Finet, Jean-Christophe Fillion-Robin, Sonia Pujol, Christian Bauer, Dominique Jennings, Fiona Fennessy, Milan Sonka, et al. 3D Slicer as an image computing platform for the Quantitative Imaging Network. *Magnetic resonance imaging*, 30(9):1323–1341, 2012. https://www.slicer.org/.

[11] Infima Games. Animated Loading Icons 2D Icons Unity Asset Store. https://assetstore.unity.com/packages/2d/gui/icons/animated-loading-icons-47844. Accès le 21-03-2024.

[12] SMBLibrary. https://github.com/TalAloni/SMBLibrary. Accès le 22-03-2024.

[13] HoloDicom. http://www.holodicom.com/. Accès le 15-02-2024.

[14] Image Guided Therapy. http://www.imageguidedtherapy.com/. Accès le 22-03-2024.

[15] Installation XR Interaction Toolkit 2.3.2. https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.3/manual/installation.html. Accès le 21-03-2024.

[16] Dr. David Pearlstone Jeremy Nelson, Sarah-Anne Birdsell. Holopatient, Hyperrealistic holographic simulated patients. https://www.gigxr.com/holopatient/. Accès le 15-02-2024.

[17] Realize Medical. Elucis. https://www.realizemed.com/. Accès le 15-02-2024.

[18] Medicalholodeck. Medical Imaging XR. https://www.medicalholodeck.com/en/. Accès le 15-02-2024.

[19] Peter Mildenberger, Marco Eichelberg, and Eric Martin. Introduction to the DICOM standard. *European radiology*, 12:920–927, 2002.

[20] Christian Moro, Charlotte Phelps, Petrea Redmond, and Zane Stromberga. Hololens and mobile augmented reality in medical and health science education: A randomised controlled trial. *British Journal of Educational Technology*, 52(2):680–694, 2021.

[21] OpenXR Unity. https://docs.unity3d.com/Packages/com.unity.xr.openxr@1.10/manual/index.html. Accès le 21-03-2024.

[22] Margarida F. Pereira, Cosima Prahm, Jonas Kolbenschlag, Eva Oliveira, and Nuno F. Rodrigues. Application of AR and VR in hand rehabilitation : A systematic review. *Journal of Biomedical Informatics*, 111:103584, 2020.

[23] Csaba Pinter, Andras Lasso, Saleh Choueib, Mark Asselin, Jean-Christophe Fillion-Robin, Jean-Baptiste Vimort, Ken Martin, Matthew A Jolley, and Gabor Fichtinger. SlicerVR for medical intervention training and planning in immersive virtual reality. *IEEE transactions on medical robotics and bionics*, 2(2):108–117, 2020.

[24] Filipi Pires, Carlos Costa, and Paulo Dias. On the use of virtual reality for medical imaging visualization. *Journal of Digital Imaging*, 34:1034–1048, 2021.

[25] Alicia Pose-Díez-de-la Lastra, Tamas Ungi, David Morton, Gabor Fichtinger, and Javier Pascau. Real-time integration between Microsoft Hololens 2 and 3D Slicer with demonstration in pedicle screw placement planning. *International Journal of Computer Assisted Radiology and Surgery*, 18(11):2023–2032, 2023. https://github.com/BSEL-UC3M/HoloLens2and3DSlicer-PedicleScrewPlacementPlanning/tree/main/AR_Planner-Unity Accès le 22 -03-2024.

[26] Research and Markets. Global Virtual Reality in Healthcare Market by Technology (Full Immersive VR, Non Immersive VR, Semi Immersive VR), Application (Diagnosis of Cognitive Disorders, Education & Training, Phobia Treatment) - Forecast 2024-2030. https://www.researchandmarkets.com/report/healthcare-virtual-reality?utm_source=GNE&utm_medium=PressRelease&utm_code=rl_fd6g9v&utm_campaign=1928068+-+Virtual+Re&utm_exec=jocamsai, 2024. Accès le 15-02-2024.

[27] Verified Market Research. Global Augmented and Virtual Reality in Healthcare Market Size by End User (Government and Defense Institutions, Hospitals, Clinics, and Surgical Centers), by Offerings (Software, Hardware), by Devices Types (Projectors and Display Walls, Virtual Reality in Healthcare Market), by Application (Patient Care Management, Surgery), by Geographic Scope and Forecast. https://www.verifiedmarketresearch.com/product/global-augmented-and-virtual-reality-in-healthcare-market-size-and-forecast/, 2024. Accès le 15-02-2024.

[28] Juan A Sánchez-Margallo, Carlos Plaza de Miguel, Roberto A Fernández Anzules, and Francisco M Sánchez-Margallo. Application of mixed reality in medical training and surgical planning focused on minimally invasive surgery. *Frontiers in Virtual Reality*, 2:144, 2021.

[29] SlicerOpenIGTLink. https://github.com/openigtlink/SlicerOpenIGTLink. Accès le 18-03-2024.

[30] Unity Technologies. Unity - Manual: Unity User Manual 2022.3 (LTS) — docs.unity3d.com. https://docs.unity3d.com/Manual/UnityManual.html. Accès le 22-03-2024.

[31] Unity Technologies. Unity - Scripting API: Mesh. https://docs.unity3d.com/ScriptReference/Mesh.html. Accès le 21-03-2024.

[32] Junichi Tokuda, Gregory S Fischer, Xenophon Papademetris, Ziv Yaniv, Luis Ibanez, Patrick Cheng, Haiying Liu, Jack Blevins, Jumpei Arata, Alexandra J Golby, et al. OpenIGTLink: an open network protocol for image-guided therapy environment. *The International Journal of Medical Robotics and Computer Assisted Surgery*, 5(4):423–434, 2009.

[33] Unity. http://www.unity3d.com. Accès le 18-03-2024.

[34] Runtime obj importer modeling unity asset store. https://assetstore.unity.com/packages/tools/modeling/runtime-obj-importer-49547. Accès le 21-03-2024.

Annexe 1 - Realize Medical - Elucis [17]

Spécification des systèmes minimal recommandé pour le PC :

- NVIDIA GTX 1080 ou plu récent
- Intel Core i7-6700K
- 32 Go de RAM (plus il y en a, mieux c'est)

En entrée		En sortie	
reconstruction 3D à partir de slices	modèle importer pour être édité	déjà disponible	en préparation
DICOM CT DICOM MRI	NifTi image et modèle STL OBJ DICOM RT Structure	STL OBJ DICOM RT Structure	NifTi models DICOM encapsulated STL & OBJ

TABLE 7 – Récapitulatif des dormats des données supportés par Elucis

Annexe 2 - 3D Organon VR Anatomy [2]

	Licence				
	Invité	Etudiante	Educative	Professionelle	Institutionnelle
Régime de licence	Usage personnel	Usage personnel	Licence de présentation	Licence d'éducation des patients	Licence de présentation
Rejoignez des sessions de livraison à distance	✓	✓	✓	✓	✓
Créer des sessions de livraison à distance		✓	✓	✓	✓
Visionneuse DICOM			✓	✓	✓
Modes XR	VR	VR	VR, MR	VR, MR	VR, MR

TABLE 8 – Récapitulatif de fonctionnalités qui nous intéressent proposées par 3D Organon VR Anatomy [2] selon les licences disponibles

Annexe 3 - Medicalholodeck - Medical Imaging XR [18]

	Licence			
	Personnelle	Commerciale	Educative	Professionelle
Données d'entrée supportées				
DICOM 3D depuis CT	✓	✓	✓	✓
DICOM 3D depuis IRM	✓	✓	✓	✓
DICOM 3D depuis CBCT	✓	✓	✓	✓
NIFTI	✓	✓	✓	✓
Échocardiographie animée 4D DICOM		✓	✓	✓
Ultrason 2D DICOM		✓	✓	✓
Echocardiographie 2D DICOM		✓	✓	✓
Angiographie 2D DICOM		✓	✓	✓
STL 3D et OBJ		✓	✓	✓
Video: MP4, M4V, MPEG, AVI		✓	✓	✓
PDF, JPG, PNG		✓	✓	✓
PET			✓	✓
DICOM animé 4D à partir de CT et IRM			✓	✓
Segmentation Automatique				✓
Outils à disposition				
Cutter	✓	✓	✓	✓
Filtre à tissus	✓	✓	✓	✓
Comparateur de données	✓	✓	✓	✓
Adjustement de la qualité DICOM	✓	✓	✓	✓
Création de préréglages	✓	✓	✓	✓
Capture d'écran	✓	✓	✓	✓
Enregistrement de video	✓	✓	✓	✓
Navigateur web en VR	✓	✓	✓	✓
Annuler/Rétablir	✓	✓	✓	✓
Mesure des distances		✓	✓	✓
Mesure des angles		✓	✓	✓
Lumière On/Off		✓	✓	✓
Outils de dessin		✓	✓	✓
Set Markers		✓	✓	✓
Enregistrer et charger des scènes		✓	✓	✓
Mesure d'espace 3D			✓	✓
Mesure de volumes			✓	✓
Masquage et sélection manuels			✓	✓
Filtre de bord			✓	✓
Mesure du cylindre				✓
Exportation des données				
DICOM en STL		✓	✓	✓
Travail collaboratif				
Travail collaboratif en VR	1 User	1 or 2 Users	✓	✓
PACS				
Accès au système PACS			Disponible	Disponible

Table 9 – Récapitulatif de fonctionnalités proposées par Medicalholodeck - Medical Imaging XR [18] selon les licences disponibles