RAPPORT

Projet Spé

Hololens 2: Technique de pointage en Réalité Augmentée

Anas Fadil - Khadfy El Mehdi - Abdellah Belaid

Nada Boukhriss - Chaymae Acherki

Problème de la Sélection en Réalité Augmentée et Techniques existantes:

L'outil classique de la sélection d'une cible en Réalité Augmentée (RA) est la sélection directe par le doigt. Cependant, ce n'est pas toujours la solution optimale. En effet, l'usage prolongé du casque RA (ex Hololens) avec des mouvements fréquents du bras en l'air pour sélectionner cause une fatigue pour l'utilisateur, ce qui va empêcher l'adoption de la RA en contexte professionnel et prolongé.

La solution proposée est d'utiliser un gant avec un capteur de pression pour pouvoir sélectionner la cible sans la toucher. L'utilisateur peut reposer son bras et sa main le long du corps et ainsi éviter la fatigue d'avoir les bras en l'air.

La sélection repose alors sur une détection de micro-mouvements comme la détection de l'inclinaison du doigt (fermé, semi-ouvert et ouvert) et la pression sur le bout du doigt. Ces micro-mouvements sont traduits en événements utiles pour la sélection.

Premier contact avec le projet :

Les premières séances étaient consacrées à l'installation de l'environnement de travail (Unity, MRTK, etc.), ce qui nous a demandé environ 2 séances. Nous avons aussi suivi les différents tutoriels du Mixed Reality ToolKit (MRTK) pour nous familiariser au casque Hololens et au développement d'applications en Réalité Augmentée.

Les séances suivantes avaient pour but de découvrir les techniques que nous devions implémenter, ainsi que le code existant de sélection avec le gant. Ce code nous a causé des problèmes lors de son exécution et son déploiement sur l'Hololens, notamment dûs aux mises à jour de MRTK et de Unity.

Nous nous sommes ensuite répartis les tâches entre nous pour implémenter une technique seul ou en groupe de 2.

Problèmes rencontrés et solutions proposées:

Le premier problème rencontré était la mise à jour de MRTK et de l'environnement, ce qui causait de nombreux problèmes de compilation avec le code existant fourni. Il était difficile de les résoudre car l'architecture du code et le code lui-même étaient complexes. De plus, c'était notre premier contact avec Unity et son langage. Heureusement, Adrien Chaffangeon, auteur du code existant et notre encadrant, était à notre écoute pour toutes nos questions.

Le code existant comportait des sections dont nous n'avions pas besoin. Nous avons donc fait des modifications pour prendre les parties nécessaires et travailler dessus.

Pour détecter les micro-mouvements, nous utilisons un gant avec des capteurs de pression. Ce gant est connecté à un web serveur qui envoie différents événements via une web socket. Le premier problème rencontré était de connecter le gant à Unity. Pour ce faire, nous avions besoin que Unity et le gant soit sur le même réseau wifi. Au début nous étions sur Eduroam, mais pour avoir facilement l'adresse IP de l'ordinateur auquel est connecté le gant, nous avons préféré utiliser nos téléphones pour faire un partage de connexion et connecter les deux. Une fois connecté, l'étape suivante a consisté à analyser et traduire les événements du gant pour pouvoir les utiliser dans Unity.

Enfin, le dernier problème général était le déploiement du code vers l'Hololens. Certains n'arrivaient pas à déployer à cause de problèmes de configuration, d'autres arrivaient à déployer mais la scène était compromise et clignotait dans le casque. Ces problèmes étaient difficiles à résoudre car ni nos encadrants ni les différents forums internet n'avaient d'explication ou de solution. Ces problèmes ont limité nos possibilités pour le projet.

Ces problèmes généraux exposés, nous présentons par la suite la technique d'origine ainsi que nos différentes extensions.

La technique de pointage d'origine :

La technique de pointage permet de sélectionner des cibles en Réalité Augmentée en les pointant du regard et en validant la sélection par des micro-mouvements des doigts, c.-à-d. des mouvements rapides et subtils des doigts.

Plus précisément, pour sélectionner un objet, l'utilisateur commence avec le point fermé (état *Deactivated*). Puis, il entrouvre son index pour pré-selectionner les objets contenus dans le cône dirigé par son regard (état *Preview*). Les objets hors de ce cône sont grisés. Lorsque que l'utilisateur ouvre complètement son doigt , cela sélectionne les éléments du cône et affiche une liste de ces éléments devant l'utilisateur (état *Selected*). L'utilisateur peut soit sélectionner un élément en le touchant, technique Look&MidAir, soit parcourir la liste à l'aide de tap du pouce sur l'index (état *Disambiguation*) et valider l'objet dans la liste en ré-ouvrant son index (état *Validate*), technique Look&Micro.

La Figure 1 représente ces différentes étapes et la Figure 2 représente la machine à état décrite précédemment.

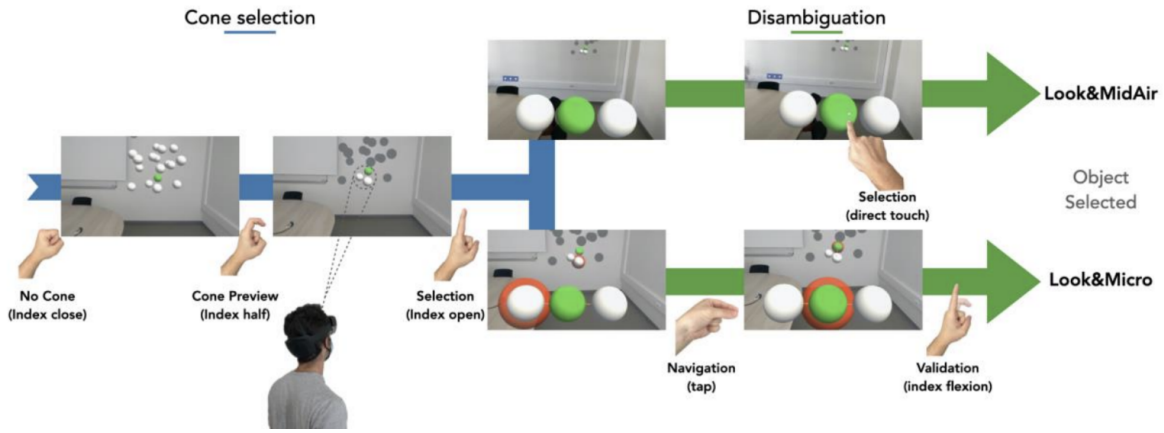

Figure 1: Les étapes de la sélection d'un objet en utilisant la technique de pointage.

Prise en main du code :

Premièrement, nous avons commencé par faire fonctionner cette technique sur Unity seulement, sans la tester sur le casque Hololens. Pour cela nous avons simulé les différents micro-mouvements à l'aide du clavier, pour faciliter le débogage de la technique.

Pour ce faire, nous avons modifié le code du fichier "GloveEvent.cs" pour redéfinir les événements du gant, c'est-à-dire les micro-mouvements nécessaires à la sélection d'un objet (par exemple tap pour avancer/reculer dans la liste et ouverture du doigt pour valider).

Puis dans le fichier "Glove.cs", où la machine à état a été définie, c'est-à-dire les passages entre les états : *Deactivated*, *Preview*, *Selected*, *Disambiguation*, *Validate*. Nous avons fait quelques modifications pour que le code du fichier "Glove.cs" soit compatible avec celui du fichier "GloveEvent.cs". En effet "GloveEvent.cs" définit les événements du gant, ces derniers sont utilisés par "Glove.cs".

Ensuite, pour chaque micro-mouvement, nous avons associé une touche du clavier ce qui nous permet de naviguer dans la machine à états sans utiliser le gant en modifiant le fichier "Keyboard.cs" :

- doigt fermé avec la touche C;
- doigt entrouvert avec la touche H;
- doigt ouvert avec la touche O;
- tap sur le bout de l'index avec la touche F;
- tap sur la base de l'index avec la touche B.

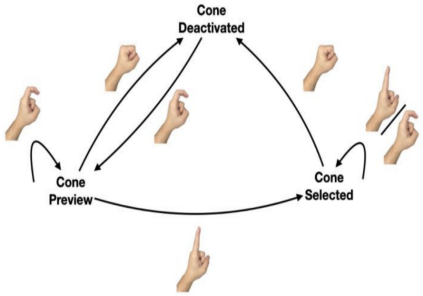

Figure 2 : Machine à état

Finalement, nous avons mis à jour le fichier "GloveClient.cs" pour connecter le gant à Unity.

<u>Problème du contour des éléments sélectionnés et sa solution :</u>

Dans la Figure 1, nous pouvons voir que l'élément pré-sélectionné dans la liste face à l'utilisateur (technique Look&Micro) possède un contour rouge. Lors de la mise à jour du code, ce contour rouge ne fonctionnait plus. La première solution implémentée était de colorier l'élément sélectionné (une balle) en vert. Or cette solution ne sera pas utilisable en pratique. En effet, si on considère un scénario réel et une liste d'objets (par exemple un ordinateur, une lampe, etc.), colorier complètement ces objets semble inadéquate et il convient de les entourer pour indiquer l'objet courant dans la liste d'objets.

Pour atteindre cet objectif, nous avons modifier le code pour afficher un

contour. Pour cela nous avons ajouté un cube nommé “outline”, cet objet est pris comme fils pour chaque objets dans la scène au moment son initialisation par son composante “Target”, ce cube prend une taille relatif à l’objet, lors de sa création il est désactivé par défaut, et lors d’un sélectionnement nous activons son affichage.

Tech 1 Manipulation directe et indirecte d’un objet :

Le but de cette technique est de manipuler un objet après l’avoir sélectionné avec la technique Look&MidAir. Il existe plusieurs façons de manipuler un objet. Par exemple le translater ou le faire tourner directement mais le but de cette technique est de faire ces mêmes manipulations en minimisant les mouvements de la main en l’air pour éviter la fatigue.

Après une séance de recherche sur les techniques de manipulation offerte par l’Hololens, un blog [1] présentait une façon simple d’implémenter la manipulation d’un objet par contrôle direct, c.-à-d. lorsque la main bouge x cm dans une direction, l’objet bouge de f(x) cm dans cette même direction, avec f une fonction continue. C’était notre point de départ.

La deuxième semaine, nous avons réussi à reproduire la technique de manipulation par contrôle direct. Nous nous sommes ensuite demandés s'il était possible de contrôler la rotation de façon indirecte, c.-à-d. lorsque la main tourne d’un angle y° dans une direction, l’objet tourne à la vitesse f(y) dans cette direction.

Nous avons commencé par une première solution. On ajoute cette f(y) à une variable notée ‘RotationSpeed’ qui conserve l’état de rotation de l’objet même si la main qui le maintient s’arrête de tourner tout en posant des limites sur cette vitesse de rotation pour que l'objet tourne toujours avec une vitesse significativement utilisable.

L’avantage de cette première solution apparaît lorsque la personne veut diminuer la vitesse où il suffirait de tourner la main dans le sens inverse à la rotation actuelle. Le programme traite alors cette rotation avec une simple multiplication de deux Quaternions qui crée une diminution de vitesse.

Mais le problème avec cette solution c’est qu’à travers une simple

vibration de la main, l'objet peut gagner une vitesse dans un sens non désiré. Face à ce problème on a essayé de bloquer les petites rotations par des conditions sur les angles d'euler. Içi un autre problème est apparu : les fonctions internes à Unity font des changements sur les angles d'euler pour qu'ils soient toujours positifs donc il ne renvoie pas toujours la vrai valeur de la rotation. A force de chercher on a pu trouver ces transformations et les inverser pour retrouver la vraie valeur de la rotation. Ainsi les conditions marchaient correctement. Ces dernières nous ont permis d'ignorer les vibrations de la main mais aussi d'ignorer les faibles rotations de la main. Si l'utilisateur souhaite lancer la rotation de l'objet il faut que la main fasse une rotation rapide. En effet si la main fait un angle de 90° mais avec 30° par seconde et comme le programme se rafraîchit au moins 30 fois par seconde, les angles reçus seront alors de moins de 1° et seront ignorés comme étant des vibrations.

Après des échanges avec des chercheurs au laboratoire, nous avons décidé d'adopter une autre idée qui consiste à stocker l'angle initial de la main et à chaque itération on ajoute la différence entre l'angle actuel de la main et de celui de l'état initial à la variable qui garde la vitesse de l'objet 3D.

Le problème avec cette technique c'est que même si on donne un angle plus faible que l'état précédent le programme va le traiter comme une augmentation de la vitesse et il ne la diminuera pas. Pour régler ce problème on a ajouté une nouvelle variable qui garde la différence d'angle de l'étape précédente. Si on fournit au programme un angle moins que cette valeur, il interprète que la main a bougé dans le sens inverse et le programme commence à diminuer la vitesse.

Un autre problème est apparu lors des tests : l'objet 3D gagne rapidement une grande vitesse et il devient très difficile de la limiter ou contrôler son sens, puisque à chaque itération on ajoute à la vitesse une différence entre deux mains dont la capture est séparée d'un grand lapse de temps.

En essayant de résoudre ce dernier problème, nous avons identifié une nouvelle solution. La solution consiste à enlever la variable de la vitesse de rotation et ajouter directement la différence entre l'angle actuel de la main et l'angle de la main initiale à la rotation de l'objet 3D. L'angle est normalisé de

façon à limiter la vitesse dans les deux sens ainsi qu'à garder la possibilité de diminuer la vitesse en diminuant l'angle avec la position initiale stockée au moment où l'objet a été attrapé par la main pour la première fois.

Finalement, après avoir réussi à implémenter cette technique de contrôle indirect de la rotation, nous avons implémenté de la même façon le contrôle indirect de la translation d'un objet. Enfin, nous avons créé une scène permettant de tester et comparer la manipulation directe et indirecte d'un objet. Cette scène pourra être utilisée pour mener des expériences utilisateur et comparer les deux approches.

Tech 2 - Sélection par le regard dans la liste de Look&Micro :

Pour le moment, dans Look&Micro, le parcours de la liste se fait par des taps successifs du pouce sur le bout de l'index. Or, ceci est lent lorsque la liste est longue. L'idée est donc de pointer un objet de cette liste à l'aide du regard de l'utilisateur et de le sélectionner en rouvrant son index. Cette solution permet par un accès direct du regard de sélectionner plus rapidement un objet dans la liste.

La première séance de travail sur cette technique était principalement réservée à la recherche du fonctionnement et de l'utilisation de l'*eye-tracking*, ou suivi oculaire, de l'Hololens. À l'aide de la documentation de MRTK, nous avons pu ensuite configurer notre scène de façon à utiliser le suivi oculaire. Il fallait principalement activer la fonctionnalité *GazeInput* dans les profils MRTK.

Les séances suivantes ont été dédiées à la conception du code à implémenter, et plus principalement à la compréhension du code fourni de manière approfondie afin de comprendre ce qu'il faut ajouter et où.

Dans la technique actuelle, à chaque frame, la technique teste pour chaque élément de la scène si cet élément est en contact ou non avec le regard de l'utilisateur. Cette approche est lourde en calcul et impose des contraintes sur la structure de la scène.

Nous avons décidé de fonctionner dans l'autre sens. Chaque cible va déterminer si elle est en contact ou non avec le regard, grâce à l'ajout de la classe "*Eye Gaze Provider*" au fichier de gestion du comportement des balles concernées

(représentant les objets de la liste - voir figure 1), appelées Target dans notre projet. De ce fait, il suffit d'attacher ce fichier à un élément de la scène et de correctement définir les fonctions "OnFocusEnter" et "OnFocusExit" responsables du comportement de l'objet une fois qu'on le regarde ou qu'on le quitte du regard.

Cette approche change radicalement l'approche du code existant. En effet, nous avons non seulement utilisé des fonctions du MRTK au lieu d'en implémenter une propre à nous. De plus, cela facilite le suivi oculaire de n'importe quel objet. En effet, il suffira de lier l'objet à ce script.

Faute de temps, et dans le but de simplifier le projet, nous avons opté pour la création de notre propre projet, assez élémentaire, contenant 6 sphères, chaque sphère pouvant être sélectionnée grâce à notre technique.

Après plusieurs difficultés rencontrées dans le déploiement, et après plusieurs modifications de la configuration du MRTK, nous avons enfin réussi à exécuter notre projet sur l'Hololens. Toutefois, avant chaque utilisation de l'application dans le casque et tout nouvel utilisateur, il fallait étalonner la détection du suivi oculaire. Mais, notre technique fonctionne.

Pour valider la sélection par le regard, notre objectif est d'utiliser un PointerHandler pour écouter tous les événements d'entrée, et agir selon la selectAction qui rentre. Etant donné que le projet était désormais complètement séparé de celui d'origine, nous avons cherché à valider la sélection sans utiliser le gant. Nous avions envisagé le clignotement des yeux ou une commande vocale. Nous avons finalement choisi la commande vocale car plus simple à mettre en place. Cependant, nous n'avons pas eu le temps de finir son implémentation.

Tech 3 - Manipulation d'un objet après sélection par Look&Micro.

Cette technique nécessitait que la technique Look&Micro fonctionne, ce qui a demandé beaucoup de temps et d'efforts pour résoudre les problèmes expliqués au début de ce rapport.

La première séance était centrée sur comprendre ce qui est demandé pour cette extension et planifier les différentes étapes de son implémentation.

La technique était prévue pour sélectionner des objets physiques et interagir avec. Cependant, la détection de ces objets par le casque peut s'avérer complexe et non nécessaire pour tester notre technique. De ce fait, nous avons créé une scène dans Unity simulant des objets physiques comme une télévision, une radio, ou une lampe.

Ensuite, il a fallu adapter Look&Micro à ces différents types d'objets car elle était implémentée initialement pour des sphères, toutes de petites tailles (voir figure 1). De ce fait, il y avait des problèmes d'affichage, par exemple des objets trop gros ou une superposition de plusieurs objets, dans la liste affichée en face de l'utilisateur.
Nous avons adapté le positionnement et redimensionnement de chaque objet pour éviter les collisions et les afficher correctement sans déformation.

Un autre problème rencontré était l'affichage des "Outline", contour rouge des objets. Comme expliqué précédemment, l'outil fourni par la librairie MRTK ne fonctionnait plus et nous n'avons pas trouvé d'explications à ce dysfonctionnement. Notre solution était alors d'utiliser un cube rouge que nous allions utiliser pour entourer chaque objet.

Enfin, après avoir sélectionné un objet, il fallait pouvoir commander ces objets, par exemple allumer la lampe, changer de chaîne de télévision, etc.

La première approche était d'afficher des objets fournis par la librairie MRTK comme des boutons , le PinshSlider (glissière)… , mais après une discussion avec Adrien nous avons conclu que cette approche va diminuer l'intérêt de notre technique de pointage , donc nous avons décidé de ne pas utiliser des objets de la librairie MRTK.

Dans notre deuxième approche, pour chaque objet, nous avons simulé les commandes à l'aide de petits boutons situés sous l'objet, De ce fait, les actions effectuées sur ces boutons influent les propriétés de l'objet, par exemple un changement de couleur, etc, selon nos choix d'implentation.

Et afin de généraliser cette implémentation nous avons créé une classe d'objets qui prend la liste des commandes comme un attribut, cette classe est alors associée à nos objets principaux, le but de cette classe est d'assurer la diversité des commandes d'un objet à l'autre.

Il reste des choses à finaliser dans cette extension, mais l'essentiel est déjà fait.

Malheureusement, le déploiement de cette extension dans l'Hololens n'a pas réussi, apparemment à cause d'un problème de configuration du projet Unity. En effet, tout fonctionne dans Unity mais le suivi oculaire dans l'Hololens ne fonctionne pas.

Conclusion:

Dans l'ensemble, ce projet a été une expérience très enrichissante et mémorable pour nous tous. En effet, cela a été pour nous la meilleure opportunité de découvrir des technologies innovantes qui nous passionnent tous depuis si longtemps. C'était vraiment notre porte d'entrée vers les principes de base de la réalité augmentée sous Unity. Et nous sommes tous extrêmement reconnaissants pour tout l'encadrement reçu à travers toutes les étapes du projet, que ce soit de la part de Pr. Nigay Laurence ou de la part de notre encadrant Chaffangeon Adrien.

Bibliographie

[1] Basic hand gesture recognition with MRTK on HoloLens 2 - Joost van Schaik