Auteurs :
— Alexis ROBACHE
— Elliot VANEGUE
— Gaëtan DEFLANDRE

Encadrants :
— Hazem WANNOUS
— Jean-Philippe VANDEBORRE

Présentation du projet et état de l'art sur la détection, le suivi de la main et l'animation de celle-ci dans une scène 3D

Projet « Hand Kinect », Master IVI

Octobre 2015 à Février 2016

Table des matières

Introduction

Durant notre master IVI [1], nous avons l'occasion de réaliser un projet orienté recherche. Nous avons sélectionné le sujet « Hand Kinect », car nous nous intéressons à la modélisation 3D et aux dernières techniques d'acquisition. De plus, ce projet est en rapport avec chacune des matières de notre master. Nous utilisons des notions de modélisation 3D de la main, de vision par ordinateur à l'aide d'une caméra, de réalité virtuelle et l'interaction avec l'environnement dans lequel la main va se déplacer et de nouvelles interactions homme machine qui est le sujet de ce projet.

L'homme utilise continuellement ses mains de manière naturelle. Nous sommes extrêmement précis et habiles avec elles. Elles nous permettent de ressentir des détails très fins. A ce jour, utiliser les mains pour interagir avec un ordinateur reste encore limité.

Les études sont nombreuses dans le domaine de la détection des mains, de la reconnaissance de leurs poses, dans leur suivi et dans la reconstruction 3D des articulations des mains et de leurs formes. Cela fait plusieurs années que la recherche scientifique et les techniques d'acquisition évoluent. Dans les années 90, la littérature considère principalement les caméras dont la sortie est un flux RGB. Les méthodes proposent généralement la détection de la main, de sa pose et de sa forme. Par la suite, des techniques utilisant plusieurs caméras et des systèmes de marquage sont apparus. Enfin, depuis les années 2005, les caméras de profondeur ont permis l'amélioration et la création de certaines techniques. Maintenant, il existe des méthodes permettant de retrouver les articulations des mains et leurs reconstructions 3D. Les résultats de ces recherches se retrouvent dans les nouvelles IHM [2], afin de contrôler, interagir et communiquer avec une machine. Les enjeux sont d'autant plus intéressants actuellement, avec le développement des interfaces immersives.

Dans ce domaine, les problématiques actuelles sont variées. Il est nécessaire de détecter les articulations de la main de manière précise. Les variations de luminosité sont courantes, la méthode doit être robuste à ces variations. L'application doit être en temps réel, pour fonctionner de manière interactive.

Dans un premier temps, nous décrivons notre projet, afin de comprendre précisément le problème et les enjeux liés à celui-ci. Ensuite, nous réalisons un état de l'art ainsi qu'un prévisionnel afin de déterminer les solutions adéquates pour notre problème et de planifier les tâches qui seront à réaliser pour la réussite de l'application finale. Enfin, nous présentons ce que nous avons réalisé dans le projet afin d'obtenir un modèle de main cohérent qui effectue les gestes de l'utilisateur.

1. Le master Image Vision Interaction est une spécialité du master informatique de l'université de Lille 1
2. Interface Homme-Machine

Chapitre 1

Présentation générale

1.1 Présentation du projet

Aujourd'hui, détecter les points des articulations du corps humain est une technique courante, grâce à certains périphériques tels que la Kinect. Les points des articulations du corps forment le squelette 3D. Cependant, les squelettes 3D ne sont en général pas assez riche en informations concernant les points des articulations des mains.

Pourtant les mains sont habiles et nous pouvons faire de nombreux mouvements, plus ou moins complexes et rapides, avec elles. De plus en plus d'applications nécessitent des IHM plus précises et plus naturelles. Utiliser ses mains pour contrôler, interagir et communiquer avec un ordinateur, devient très intéressant. Ce niveau de précision peut être utile dans divers domaines :

— simulation médicale.
— modélisation et CAO [1].
— manipulation d'objets 3D virtuels.
— jeux vidéo.

Le projet « Hand Kinect » consiste à extraire depuis une caméra Kinect 2, les points des articulations des mains, pour cela nous étudions plusieurs méthodes réalisées jusqu'à maintenant. Ainsi, l'assemblage de ces points nous permet d'obtenir le squelette 3D complet de la personne devant la Kinect.

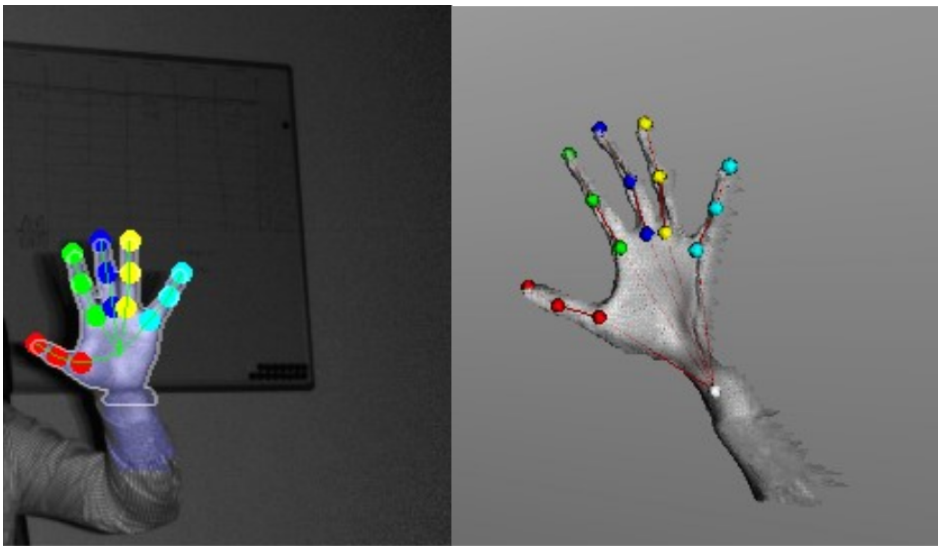

FIGURE 1.1 – Détection des articulations des mains

Ensuite, nous souhaitons animer un modèle 3D, qui peut être un modèle d'humain, de robot ou bien de créature. Les mouvements de ce modèle doivent suivre les points du squelette préalablement détectés, de manière cohérente par

1. Conception Assisté par Ordinateur

rapport aux mouvements de la personne devant la Kinect. Nous utilisons Unity3D, pour cette partie de modélisation.

Unity3D est un moteur de jeu qui permet de développer rapidement des environnements et applications 3D. Il est facile d'obtenir des applications compatibles sur de nombreuses plateformes. Ce moteur propose également une licence gratuite assez complète.

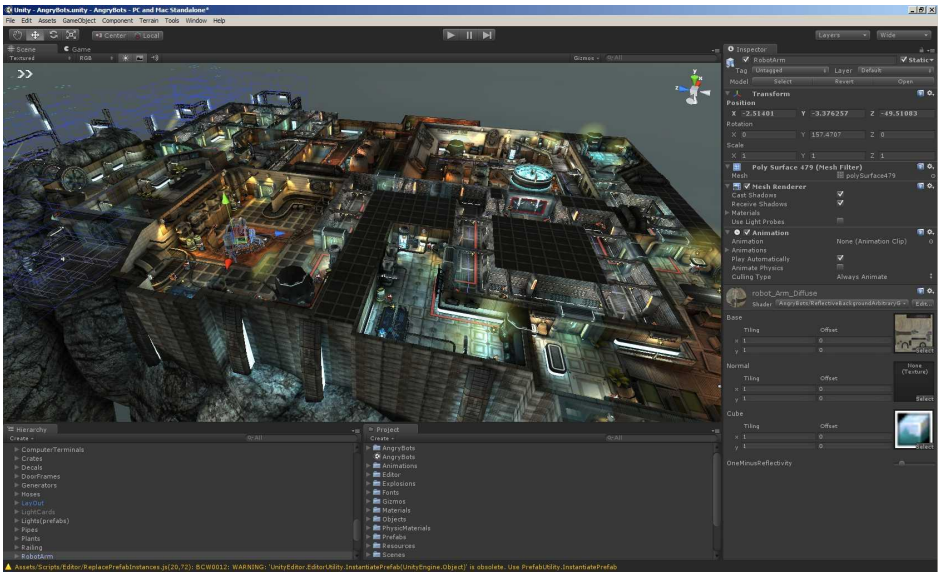

FIGURE 1.2 – Interface de Unity3D

Dans le cadre de ce projet, nous sommes amenés à rédiger un état de l'art. Cette première partie nous permet de synthétiser et sélectionner les solutions. Notre objectif étant de détecter la main d'une personne et ses articulations, afin d'animer en 3D les mouvements de la main de cette personne. Cela doit être réalisable en temps réel et à partir d'une caméra nous fournissant une image RGB et une image contenant l'information de profondeur de la scène filmée.

1.2 Contexte

La réalisation de ce projet se fait avec l'équipe 3D SAM [2]. Cette équipe fait partie du laboratoire CRIStAL [3]. Ce centre est le nouveau pôle de recherche qui regroupe les laboratoires d'informatique, signal et automatique de Lille.

L'équipe 3D SAM est basée à l'école d'ingénieur Télécom Lille. Cette équipe conçoit de nouveaux outils et méthodes d'analyse des formes des objets 3D statiques et dynamiques. Ils travaillent sur l'analyse de formes des objets 3D et la modélisation des variations des formes dans des vidéos 3D.

2. Modeling and Analysis of Static and Dynamic Shapes
3. Centre de Recherche en Informatique, Signal et Automatique de Lille

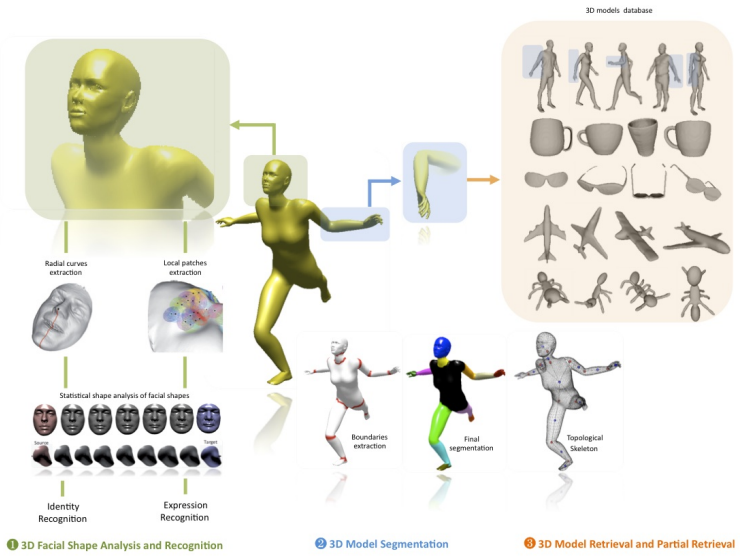

FIGURE 1.3 – Exemples de travaux de l'équipe 3D SAM

Le projet « Hand Kinect »est un nouveau projet proposé par l'équipe. Pour sa réalisation, nous disposons d'une caméra Kinect 2. Pour les représentations et animations 3D, il nous faut un environnement de modélisation. Nous utilisons Unity3D comme montré précédemment. Nous disposons également, d'un module nous permettant d'extraire les points des articulations de la main.

1.3 Problèmatique

D'abord, la Kinect 2 peut déjà suivre six personnes à la fois. Elle détecte un squelette de vingt-six points (voir Fig. **??** et **??**). Cependant, les mains sont représentées seulement par 4 points, qui sont le centre du poignet, le centre de la main, le pouce et le bout du majeur. C'est une amélioration par rapport à la Kinect 1, qui ne détectait que le poignet et le centre de la main.

D'autres outils permettent de détecter les mains des utilisateurs. Par exemple le Leap Motion détecte les articulations des mains. Néanmoins, ce capteur est limité, car il faut garder les mains proches du capteur. La portée et le champ de détection de ce dispositif restent restreint. Leap Motion ne permet pas de détecter le squelette complet d'une personne. Il permet cependant une plus grande précision et une plus grande stabilité que la Kinect dans la détection des articulations de la main.

À partir de ces constats, nous pouvons formuler la problématique suivante. Puisque, les mains sont très habiles, il peut être difficile de détecter leurs articulations dans des positions contraignantes. En supplément, les problèmes liés à la luminosité sont récurrents, il est souhaitable d'utiliser des techniques robustes et invariantes à la luminosité, mais qui restent efficaces. De plus, il existe encore peu de méthodes permettant de détecter la pose et les articulations par caméra, et qui permettraient de retrouver l'ensemble du squelette humain. Enfin, la modélisation de la main dans la scène 3D doit être fidèle à la main de l'utilisateur pour plus de précision et de réalisme.

1.4 Objectif

Dans un premier temps, le but de ce rapport est d'appréhender les différentes méthodes existantes sur la détection des mains, leurs articulations et la reconnaissance de la pose des mains. Nous souhaitons également comprendre les principes utilisés lors de l'animation 3D de celles-ci.

Étudier les techniques de détection des mains et leurs articulations, a pour objectif de bien cerner les méthodes qui peuvent être utilisées par le module. Celui-ci fournit les positions spatiales des articulations des mains.

Ensuite, le but de l'étude des techniques d'animation des mains en 3D est de pouvoir comprendre et utiliser efficacement les techniques pour animer les mains de façon naturelle dans Unity3D.

Le dernier objectif de cette première partie, est de faire une planification des tâches à effectuer. Ceci permet de s'organiser et de voir si les objectifs futurs sont réalisables dans les délais impartis.

1.5 Présentation de la Kinect 2

La Kinect 2 de Microsoft possède une caméra couleur en mode YUV de 1920x1080 pixels. YUV est une combinaison particulière d'informations permettant de restituer la couleur, comme RGB ou HSP. Cette combinaison aussi appelée YCbCr, avec Y la luminance, U/Cb la composante bleue sans luminance et V/Cr la composante rouge sans luminance. En plus de sa caméra couleur, elle possède 3 diffuseurs de rayonnements infrarouges couplés à une caméra infrarouge qui servent à acquérir une carte de profondeur. Les autres fonctionnalités ne nous intéresseront pas pour résoudre notre problématique.

FIGURE 1.4 – Caméra Kinect 2

On récupère donc de Kinect les données brutes suivantes :
— un flux vidéo couleur en YUV
— un flux vidéo d'images de profondeur Fig. **??**
Ces deux flux peuvent servir à détecter la main (méthodes expliquées dans la partie état de l'art). Le SDK de la Kinect fournit des informations déjà traitées, par exemple pour notre projet, les jointures du poignet, du centre de la main, du sommet du pouce et du sommet du majeur.

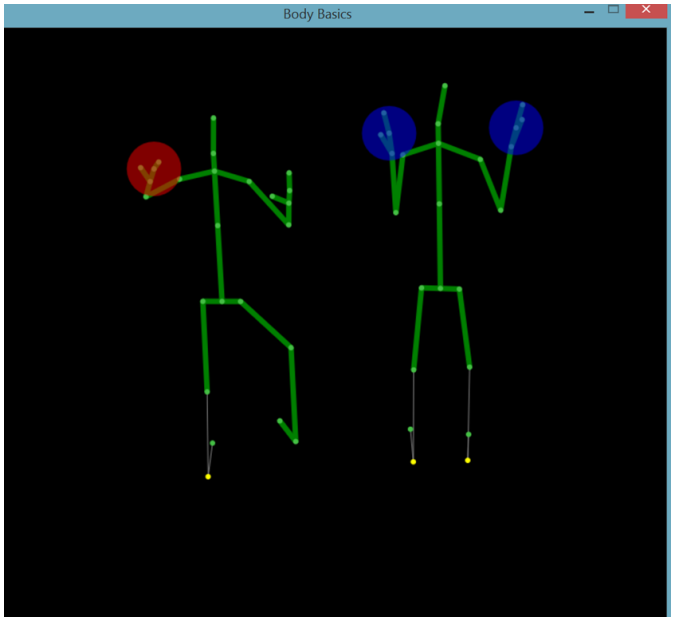

FIGURE 1.5 – Squelette de l'utilisateur détecté par la Kinect 2

1.5.1 Différences entre Kinect V1 et Kinect V2

Comme on peut la voir dans l'image ci-dessous, la Kinect 2 est une amélioration de tous les aspects de la Kinect précédente.

Les améliorations qui nous intéressent pour le projet sont :
— L'augmentation de la résolution de la caméra de profondeur, qui apporte une précision plus importante, mais cause un problème de performance : plus de pixels à traiter.
— L'augmentation des champs de vision horizontaux et verticaux
— Le plus important : l'augmentation du nombre de points détectés, principalement l'ajout de 2 points au niveau de la main

Specifications

	Kinect for Windows v1	Kinect for Windows v2
Color	640 × 480 @ 30fps	1920 × 1080 @ 30fps
Depth	320 × 240 @ 30fps	512 × 424 @ 30fps
Sensor	Structured Light (PrimeSense Light Coding)	Time of Flight (ToF)
Range	0.8~4.0 m	0.5~4.5 m
Angle of View Horizontal / Vertical	57 / 43 degree	70 / 60 degree
Microphone Array	○	○

FIGURE 1.6 – Différence entre Kinect V1 et Kinect V2

FIGURE 1.7 – Depth map 3D de la Kinect 2

Chapitre 2

Etat de l'art

2.1 Différents types de données

Pour réaliser notre projet : animer un modèle de main en se basant sur les mouvements de la main de l'utilisateur, nous avons à disposition une Kinect 2 et un LeapMotion. Pour commencer notre état de l'art, nous allons voir quelles données peuvent être récupérées par ces outils et quels autres moyens auraient pu être utilisés pour réaliser ce projet.

Une liste non exhaustive des périphériques et des données qu'ils fournissent :

— Kinect1 et Kinect 2 de Microsoft, fournissent, en données brutes, une image couleur ainsi qu'une image de profondeur. En utilisant le SDK fourni, on dispose en plus de nombreux outils parmi lesquels une détection du squelette et pour Kinect 2 une abstraction de la main avec 4 points détectés : le centre du poignet, le centre de la main, le sommet du pouce et le sommet du majeur.

— LeapMotion fournit, en données brutes, un squelette 3D plus détaillé mais sur une zone plus restreinte. Avec le SDK, on a accès directement à un squelette très détaillé de la main.

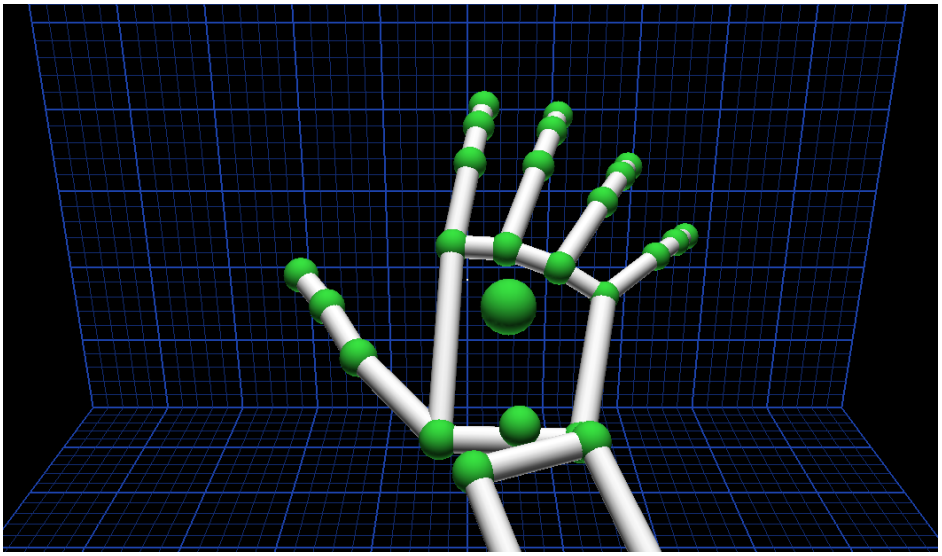

FIGURE 2.1 – Squelette de la main de l'utilisateur détecté par la LeapMotion

— RealSense est un périphérique relativement récent, à mi chemin entre Kinect 2 et LeapMotion. On peut, à l'aide du SDK fourni, récupérer des informations semblables à celles fournies par LeapMotion : 22 points de la main détectés et suivis

On peut utiliser des outils spécifiques (Kinect) ou des caméras classiques pour obtenir des images couleurs. Bien que possédant plus d'informations que les images de profondeur (3 canaux au lieu d'un), les images couleurs sont moins adaptées aux calculs 3D.

Ces données couleurs peuvent être utilisées avec des accessoires colorés pour simplifier la détection des points de la main [?]. Cette méthode permet de détecter plus facilement les jointures et les points d'intérêt. Cependant, cela ajoute des contraintes, notamment le fait de devoir porter un gant standardisé.

On associe également les données couleurs avec des données de profondeur pour améliorer la détection et gérer les cas de superposition [?].

L'autre type de données est la carte de profondeur, image en niveau de gris servant à représenter la profondeur. Cette carte de profondeur est utilisée par un grand nombre de méthodes pour les problèmes de clustering du corps humain, notamment les SDK de Kinect 1 et Kinect 2 se basent principalement dessus pour fournir le squelette [?]. Les SDK de Kinect1/Kinect2 étant prévus pour le suivi du corps entier, ils sont peu performants pour le suivi précis de parties spécifiques (comme la main). L'utilisation directe du SDK semble donc insuffisante pour résoudre notre problème. Cependant, on peut aussi récupérer les données brutes fournies par la Kinect 2 et certaines méthodes se contentent d'une image de profondeur pour la détection de la main [?].

2.2 Détection de la main

La Kinect 2 possède deux types de caméra, une caméra couleur et une caméra de profondeur utilisant la méthode « Time of Flight ». Les caméras de type « Time of Flight »récupérent les rayons envoyés par projecteur qui ont été réfléchis par les objets de la scène, ce qui permet d'obtenir les informations de profondeur. Il existe plusieurs solutions utilisant l'une ou l'autre de ces caméras. Nous allons dans cette partie, détailler plusieurs méthodes réalisées jusqu'à maintenant.

2.2.1 Détection et suivi de la main à partir d'une image couleur

La méthode proposée par S. Bilal et al [?] utilise l'algorithme de Viola et Jones [?] afin de détecter la main et fournir la position de celle-ci en créant une ROI (région d'intérêt) autour d'elle. L'algorithme de Viola et Jones [?] nécessite une base de connaissances composée des caractéristiques de l'objet recherché. Elle est utilisée dans un apprentissage supervisé, c'est-à-dire que l'algorithme a besoin de données représentant l'objet à détecter pour classifier les caractéristiques de celui-ci. Cet algorithme est basé sur des caractéristiques pseudo-Haar qui crée des masques rectangulaires et adjacents dans différentes zones de l'image (voir Fig. **??**). Chaque masque calcule l'intensité des pixels qu'il contient, puis l'algorithme fait la différence entre les masques blancs et les masques noirs.

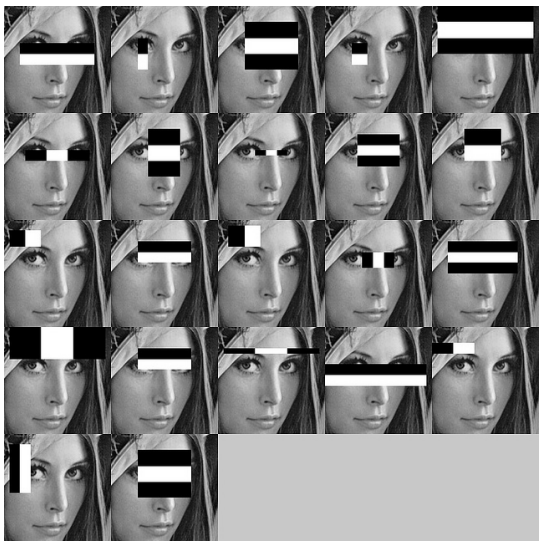

FIGURE 2.2 – Exemple de caractéristiques pseudo-Haar utilisées pour l'algorithme Viola et Jones

Pour améliorer les performances de leur algorithme, Viola et Jones utilisent la méthode Adaboost. Son principe est de sélectionner les caractéristiques les plus performantes pour la détection de l'objet grâce à un calcul de probabilité utilisant l'entropie [1] des données.

1. valeur mesurant l'incertitude

Une fois qu'une ROI est formée autour de la main de l'utilisateur, S. Bilal et al [?] changent d'espace colorimétrique afin de pouvoir différencier la main du reste de l'environnement. Pour cela, ils choisissent d'utiliser l'espace YCbCr qui permet d'obtenir des informations de chromaticité dont la valeur est presque équivalente quelle que soit la couleur de peau de l'utilisateur [?]. Dès que la ROI de la main est binarisée, il est possible de détecter le bout des doigts. Il faut tout d'abord calculer le centroïde de la forme de la main, puis la distance entre ce centroïde et chaque point du contour de la main. Le contour d'une forme dans une image est défini par un changement brutal de couleur. Cette méthode permet d'obtenir un graphe correspondant à la Fig. **??**, où chaque pic représente un doigt.

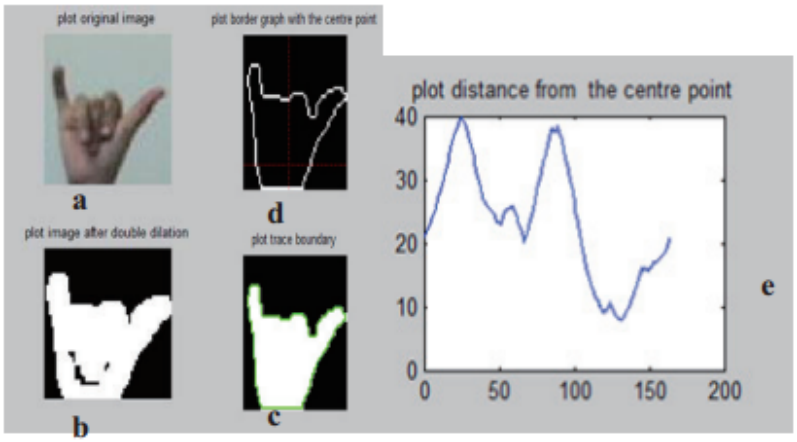

FIGURE 2.3 – Histogramme des distances entre les points du contour de la main et le centroide

Afin de pouvoir suivre la main, S. Bilal et al [?] utilisent l'algorithme de Kalman [?]. Cet algorithme permet de prédire la position d'un objet, ce qui va permettre de suivre la main malgré le bruit présent dans l'image. De plus, cette méthode n'a besoin de connaître que l'état précédent à celui qui est calculé. Etant donné que l'algorithme de Kalman est un algorithme de prédiction, il peut ne pas être très précis lorsque l'utilisateur effectue des changements de direction rapides avec sa main.

Cette méthode nous permet de déterminer la position de la main et de détecter le bout des doigts. Cependant, elle ne nous permet pas de réaliser notre projet entièrement. En effet, nous avons besoin de plus de précisions en détectant les articulations de la main. De plus, cette méthode est sensible à la luminosité de la scène filmée et n'est pas très adaptée à l'animation 3D d'une main.

2.2.2 Détection et suivi de la main à partir d'une image de profondeur

Une seconde solution utilise une image de profondeur plutôt que l'image binarisée avec le changement d'espace colorimétrique. Pour réaliser la détection de la main, il faut dans un premier temps déterminer une ROI autour de celle-ci.

Cette première étape a été expliquée par Sharp et al [?], qui utilisent un classifieur basé sur la méthode développée par J. Shotton et al [?]. Ce classifieur permet d'effectuer de la reconnaissance des parties du corps et permet également de détecter les articulations. Dans ce but, cette méthode utilise l'algorithme de Lepetit et al [?] qui crée un arbre de décision à partir des données d'entraînement et descend dans celui-ci jusqu'à atteindre une feuille (voir Fig. **??**).

Les feuilles de cet arbre sont les différentes parties du corps à reconnaître. Cet arbre est construit en calculant l'entropie de chaque pixel, ce qui permet de déterminer à quelle partie du corps il appartient.

On voit dans la Fig. **??** que l'articulation de la main est détectée, il est alors possible de construire une ROI autour de ce point. Cette méthode a été implémentée dans la caméra Kinect.

La méthode utilisée dans [?] ne nécessite aucune donnée provenant d'images antérieures. L'ensemble des calculs est réalisé sur une image à partir d'une base d'apprentissage contenant différentes postures de la main. Étant donné qu'il est impossible de stocker toutes les postures de la main, la méthode de T. Sharp et al [?] calcule une fonction énergie permettant de déterminer quelle est la posture la plus représentative de celle de la main de l'utilisateur.

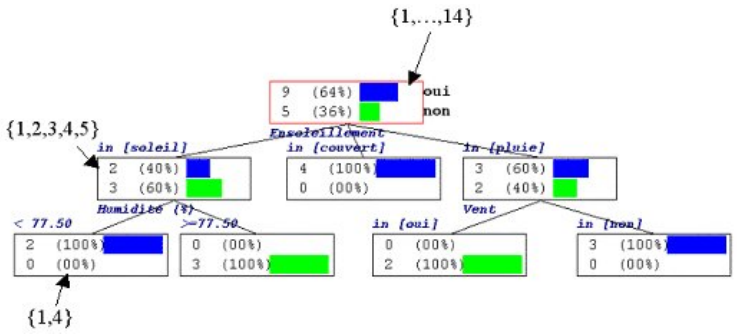

FIGURE 2.4 – Exemple d'arbre de décision

FIGURE 2.5 – Résultat de la détection des parties du corps et des articulations grâce à la méthode de [**?**]

Cette fonction énergie est calculée à partir des pixels du modèle et ceux de l'image de profondeur capturée par la caméra. Elle est définie par le calcul suivant :

$$E^{au}(Z_{roi}, R_{roi}) = \sum_{ij} \rho(\bar{z}_{ij} - r_{ij}) \tag{2.1}$$

Dans cette équation, z_{ij} représente le pixel à la position (i,j) dans le modèle et r_{ij} représente le pixel à la position (i,j) provenant de l'image de profondeur. La fonction $\rho(e)$, quant à elle, est équivalente à $min(|e|, \tau)$ où τ est une valeur fixe permettant d'éliminer le bruit présent dans l'image de profondeur.

2.3 Animation de la main

Pour pouvoir animer une main de façon cohérente et réaliste, il existe plusieurs solutions. Cependant, dans le cadre d'une animation en temps réel et contrôlée par la main de l'utilisateur, des animations pré-calculées et maitrisées entièrement par l'animateur sont à abandonner. L'animation fine et la déformation de modèles 3D nécessitent, le plus souvent, une armature (voir Fig. **??**).

Les armatures permettent de modifier le modèle associé à partir de rotations et translations. Il existe de nombreuses méthodes pour réaliser cette modification. Celle utilisée majoritairement est d'associer les sommets du modèle aux jointures. Cette association peut être difficile : un sommet est associé à une seule jointure, ou floue : un sommet peut être associé à plusieurs jointures par des poids différents.

Un algorithme permettant cette association floue est « bone heat weighting »[**?**]. Il permet de trouver les poids adaptés à l'association entre les os de l'armature et les sommets (voir Tab. **??**). Pour rendre un résultat proche du

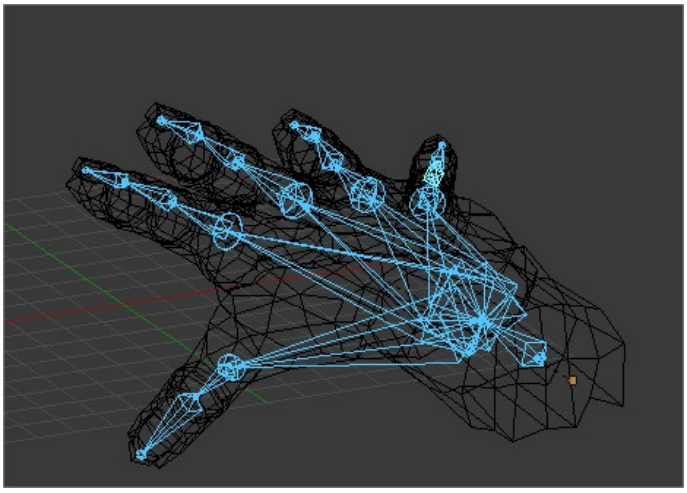

FIGURE 2.6 – Armature de la main droite réalisée sur Blender

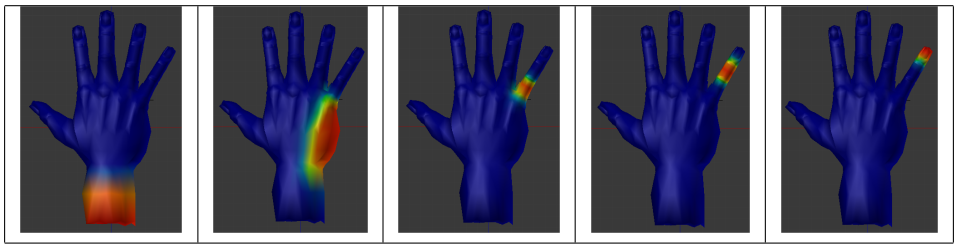

TABLE 2.1 – Répartition des poids à différents endroits de la main

réel, on souhaite que les poids possèdent certaines propriétés :

— Ils devraient être indépendants de la résolution du modèle, un sommet situé à un endroit du modèle devrait avoir le même poids quel que soit le nombre de sommets proches.
— Ils devraient varier de façon homogène avec la surface. Les valeurs de poids ne devraient pas varier fortement si la surface est lisse mais, inversement, si elle est inégale, les valeurs devraient varier plus.
— Pour éviter des artefacts liés aux plis du modèle, il doit y avoir un rapport de proportionnalité entre la largeur des jointures et la distance entre la jointure et le modèle.

Un modèle dont les sommets ont été associés à l'armature à l'aide de cet algorithme peut être « animé »de façon réaliste. Pour cela, on peut créer des contraintes sur les mouvements possibles de l'armature pour correspondre à des actions possibles. Par exemple, pour la main, les doigts ne peuvent pas bouger librement. On peut donc contraindre les degrés de liberté de chaque jointure et même empêcher les translations. Toutes les positions de la main pouvant être exprimées par une série de rotations contraintes au niveau des jointures du squelette.

L'animation par clef permet de créer des animations fluides à partir de, au minimum, deux « Key Frames ». L'animation est réalisée en calculant la position, l'orientation et la posture de l'objet animé pour chaque image entre deux « Key Frame ». La technique utilisée le plus fréquemment est l'interpolation linéaire. Cette méthode est rapide comparée à d'autres techniques d'animation, cela permet d'assurer le temps réel. Cependant dans le cadre de notre projet, l'utilisateur étant celui qui contrôle l'animation du modèle, les « Keyframes » seraient difficilement récupérables par l'outils de capture utilisé.

Une autre solution consiste à n'utiliser que des informations de position obtenues par détection des articulations des doigts de la main *via* un périphérique de capture. Dans cette méthode, on associe les points détectés aux jointures de l'armature. Les positions des jointures du modèle sont donc naturellement contraintes par les postions des articulations de la véritable main.

Enfin, il existe une méthode appelée la cinématique inverse qui utilise un point fixe et un point mobile de l'armature. Elle permet de retrouver les rotations effectuées par les jointures pour que le point mobile soit à sa position

connue. Pour cela, l'algorithme de cinématique inverse minimise la fonction de distance entre le point mobile de l'armature et la position que ce point doit atteindre. La cinématique inverse a besoin de 3 principes pour fonctionner :

— Les membres sont des solides indéformables de longueur donnée, reliés entre eux par des articulations. Ce principe est respecté par l'armature : les os peuvent être indéformables et leur taille peut être fixée, ils sont reliés entre eux par des articulations.
— Les articulations supportent une certaine torsion maximale qui varie selon la direction et la vitesse. Ces deux contraintes peuvent être appliquées aux jointures d'une armature.
— Les mouvements sont supposés suffisamment réguliers pour paraitre naturels. Comme les points mobiles sont obtenus à partir de mouvements d'une main réelle, ils vont nécessairement paraitre naturels.

Les techniques classiques de résolution de la cinématique inverse consistent à inverser les équations de cinématique avant qui retrouvent la position à partir des rotations des articulations. Le problème principal vient de la difficulté de cette inversion et de la possibilité d'un ensemble vide de solution, les techniques classiques utilisent donc des méthodes d'optimisation itératives comme « Iterative Closest Points »ou l'Algorithme Levenberg-Marquardt pour palier ces problèmes.

2.4 Evaluation des solutions envisagées

Parmi les solutions que nous avons envisagées dans cet état de l'art, nous avons vu que la solution avec une image couleur pouvait être limitée. Celle-ci nécessite un algorithme permettant de suivre au mieux la main de l'utilisateur. L'algorithme de Kalman est efficace, mais peu précis lors de changements d'orientation brusques et il est également assez complexe à mettre en place. Il existe d'autres méthodes de suivi qui sont peut-être plus appropriées à la conception d'une nouvelle IHM, mais le suivi n'est pas le seul défaut de cette méthode pour la réalisation de notre projet. Une animation 3D de la main n'est pas possible avec ce genre de méthode. En effet, sans l'information de profondeur de la main, il est difficile de déterminer de nombreuses postures de celle-ci. Cette solution était une première approche de la détection la main ainsi que des doigts qui reste efficace avec une luminosité correcte.

Notre choix pour la réalisation de ce projet se dirige plutôt vers l'utilisation de la caméra de profondeur. La Kinect 2 ayant déjà une implémentation d'une partie des étapes permettant la réalisation de notre projet, nous pourrons nous concentrer sur la partie plus spécifique de détection et de localisation des articulations de la main. Pour cela, nous avons vu que la méthode utilisant l'image de profondeur pouvait être efficace grâce à la détermination de la posture de la main. Cela nous permet ainsi d'ajuster un modèle de main et de fabriquer le squelette en fonction de ce modèle.

Chapitre 3

Travaux réalisés

3.1 Séparation des tâches

Nous constatons assez rapidement dans le projet qu'il est très difficile d'utiliser le SDK de la Kinect et la DLL directement dans Unity3D. Plusieurs projets, trouvés durant nos recherches, utilisent un système de sockets afin de pouvoir envoyer des informations à une application Unity3D à partir d'une application C#. Nous utilisons donc la même méthode afin de pouvoir séparer les différentes tâches de notre projet.

Nous créons donc un client utilisant la DLL et qui effectue l'ensemble des opérations sur les données afin de les envoyer à une application Unity3D qui n'a plus qu'à effectuer les transformations sur le modèle avec les données reçues. En effet, les données fournies pas la DLL ne peuvent être appliquées tel quel sur le modèle. Le repère des coordonnées fournies par la DLL est le repère image, donc un repère en 2D en pixel. Or, notre application doit modèliser la main dans un environnement 3D.

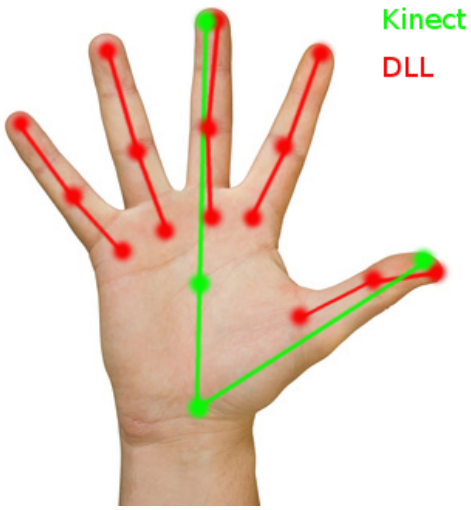

FIGURE 3.1 – Carte des points de la main détectés par la DLL et la Kinect

L'intérêt de cette séparation est que notre méthode de localisation des jointures se repose sur une DLL et que les calculs effectués sur les données sont spécifiques a celle-ci. En cas de changement de technologie, il suffit de modifier le client sans avoir à changer l'application Unity3D. La seule contrainte étant la structure des sockets envoyés. Nous avons donc une application modulable permettant de modifier facilement la méthode de récupération des jointures, mais aussi de modifier le matériel utilisé.

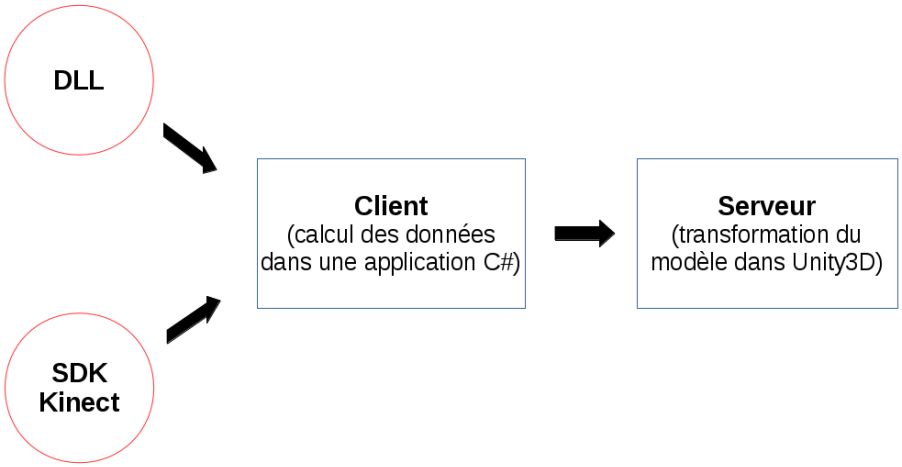

FIGURE 3.2 – Schéma de la structure de l'application

3.2 Récupération de la position des articulations de la main

Le premier problème à résoudre est donc le calcul de la troisième coordonnée de chaque jointure. Grâce à la Kinect, il est possible de récupérer les coordonnées, dans le repère monde, de la jointure du poignet. Elle nous permet également de récupérer une valeur dans l'image de profondeur, valeur représentant la troisième coordonnée d'un point. Nous avons alors la valeur du pixel de l'image de profondeur de la jointure recherchée $p_{recherche}$, la valeur du pixel de l'image de profondeur de la jointure du poignet p_{ref} et la valeur de la troisième coordonnée dans le repère monde de la jointure du poignet z_{ref}. Nous appliquons alors une règle de trois pour obtenir la valeur dans le repère monde de la troisième coordonnée de la jointure courante.

$$z_{recherche} = \frac{z_{ref} * p_{recherche}}{p_{ref}} \tag{3.1}$$

La seconde problématique de notre méthode est la normalisation des coordonnées que nous avons récupérées. Les coordonnées de la kinect sont dans le repère monde, nous avons donc besoin d'utiliser les caractéristiques de la caméra pour normaliser les coordonnées. Nous prenons une distance de 4m pour la profondeur de la caméra, car cela correspond à la zone de détection de la Kinect (voir Fig. **??**).

Enfin la dernière problématique est de mettre en correspondance les points calculés dans la partie client avec les points du modèle de la partie serveur. Cette mise en correspondance se fera côté serveur et sera expliquée dans la partie « modélisation et animation de la main »de ce rapport (*cf.* **??**).

3.3 Structure du message envoyé par sockets

Nous formatons un message contenant les informations de la Kinect et les coordonnées des jointures récupérées précedemment. Ce message est créé par le client pour être envoyé au serveur via le procole UDP. Enfin le message est traité par le serveur, afin d'obtenir les données sous forme de variables exploitables.

Au cours du projet, les informations transmises au serveur ont évolué. Dans un premier temps, nous envoyions un message lorsque la DLL fonctionnait. La permière version du message contenait uniquement les coordonnées des jointures détectées par la DLL avec leurs coordonnées Z normalisées (voir Fig. **??**). Ces coordonnées ne sont pas suffisantes pour l'animation complète de la main. Par exemple, la position du poignet est requise pour l'animer sur Unity.

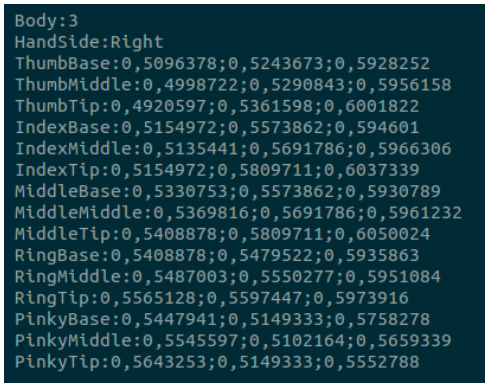

FIGURE 3.3 – Première version du message.

Dans un second temps, nous avons ajouté les informations provenant de la Kinect en plus de celles de la DLL. Dans ce message nous retrouvons les positions relatives aux mains calculées par la Kinect. En plus des coordonnées des jointures de la DLL nous avons les positions :

— du poignet
— du centre de la main
— du bout de la main
— du pouce

Nous transmettons également le statut de la main que la Kinect a assignée. Le message est envoyé lorsque les statuts sont « lasso », « closed »ou « open », (voir Fig. **??**). Pour certaines images acquises par la Kinect, les fonctions de la DLL ne retrouvent aucun résultat. Par exemple, lorsque l'on ferme la main, la DLL ne détecte plus les doigts et aucun message n'est envoyé, alors que la Kinect a détecté les points de la main.

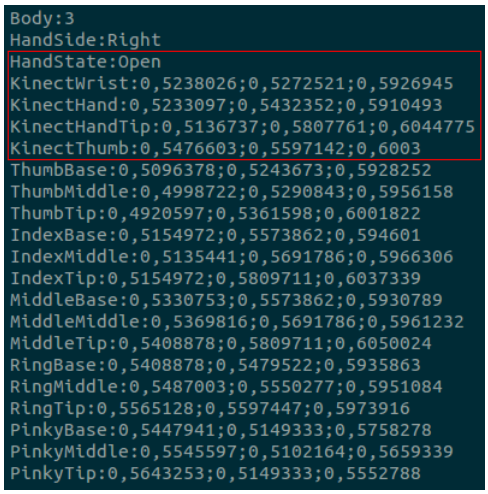

FIGURE 3.4 – Deuxième version du message.

Enfin, nous optons pour deux messages diffèrents. Un message est envoyé quand la DLL n'a pas détecté de point. Et un message différent est envoyé quand la DLL a retrouvé les jointures de la main (voir Fig. **??**). Le champ `AiolosDLLStatus` est ajouté pour différencier les messages.

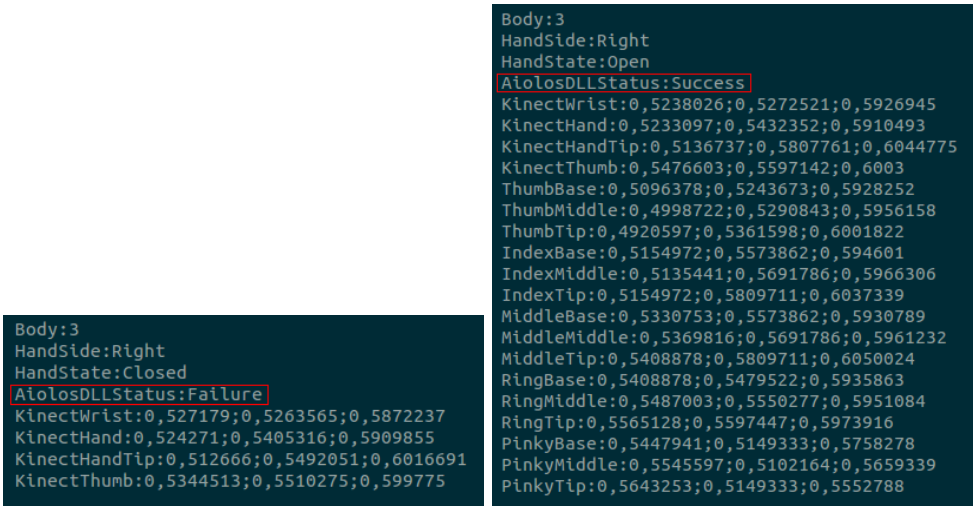

FIGURE 3.5 – Versions actuelles des messages. À gauche, DLL non fonctionnelle. À droite, DLL fonctionnelle.

3.4 Modélisation et animation de la main

Nous avons choisi de ne pas modéliser la main de l'utilisateur, à la place, nous avons récupéré un modèle de main existant. Sous Blender, nous avons appliqué à ce modèle une armature pour pouvoir l'animer de façon réaliste. Cette armature est constituée d'os et de jointures. L'armature est représentée avec une structure d'arbre, avec les noeuds qui représentent les os, et les arcs qui représentent les jointures. Les transformations de chaque fils dépendent des transformations de leurs pères. Ainsi, un mouvement du poignet entraîne le mouvement de chaque os de la main. Cette armature pourra être utilisée pour mimer la main de l'utilisateur.

Pour relier l'armature au modèle, on utilise une méthode de « skinning »(*cf* **??**). Dans Blender, on peut utiliser différents types de skinning, soit « à la main » en créant des groupes de sommets à associer à chaque jointure, soit en utilisant des algorithmes de création des groupes et d'attribution automatique des sommets à ces groupes. Nous avons choisi d'utiliser la méthode « bone heat weighting »[**?**].

Le modèle et l'armature ainsi créés sont importés dans Unity3D tout en conservant les associations effectuées par le skinning. Ensuite, on teste ce modèle en l'animant à l'aide de rotations des jointures, on peut ainsi faire prendre des poses à la main (voir Fig. **??**).

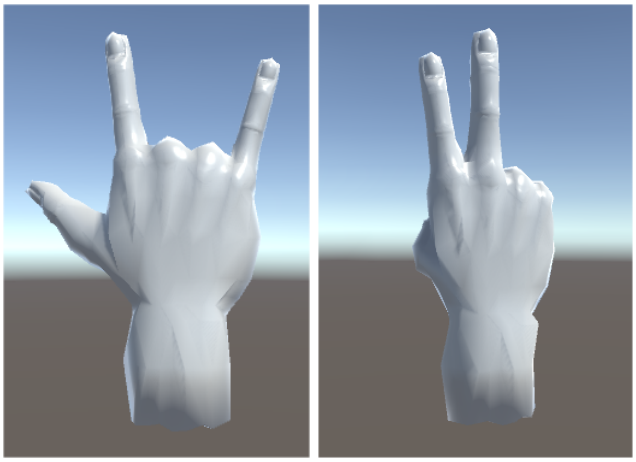

FIGURE 3.6 – Exemple de poses réalisées à partir de rotations des jointures de l'armature dans Unity

Il existe différentes techniques d'animation basées sur une armature (*cf* **??**). Pour notre projet, nous avons choisi la méthode qui utilise les articulations récupérées par la DLL en conjonction avec les points récupérés *via* le SDK de

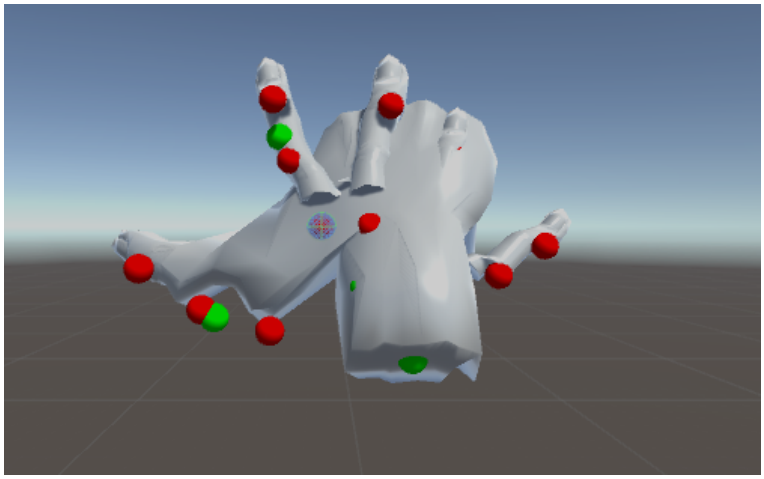

FIGURE 3.7 – Exemple de déformation du modèle avant mise en correspondance

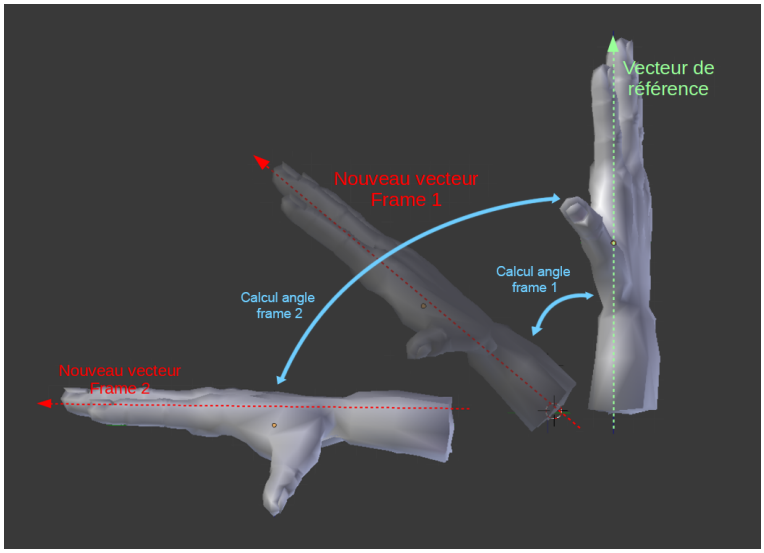

FIGURE 3.8 – Schéma des rotations effectuées par la main entre deux frames

la Kinect. Ces points servent à positionner les jointures de l'armature par translations et/ou rotations. Cependant, comme expliqué précédemment, il faut mettre en correspondance ces points (*cf.* **??**). Pour cela, le plus simple a été de faire correspondre le point du poignet détecté par la Kinect à la jointure équivalente du modèle. Cependant, si aucune rotation n'est appliquée sur la main, la position de la base des doigts ne correspond plus et le modèle est déformé voir Fig. **??**.

Il faut prendre en compte la rotation du poignet pour que les jointures de l'armature correspondent aux articulations de la main. Pour cela, nous gardons en mémoire la rotation du modèle avant l'interaction avec l'utilisateur, puis nous calculons la rotation à partir de cet angle (voir Fig. **??**). Cette méthode permet de ne pas avoir de modification des axes de rotation suite à une succession de rotations.

Une fois le modèle correctement positionné et orienté, il faut s'occuper des doigts. Nous pouvons voir sur la Fig. **??** que le modèle est complètement déformé et ne correspond plus à la main d'un utilisateur. Les points d'un modèle ne sont pas faits pour être manipulés directement dans l'espace, il est préférable de calculer des angles pour chaque jointure afin de leur appliquer des rotations. Une partie complexe a été de décider d'une méthode de calcul des angles des jointures et de l'implémenter. Nous avons choisi de reprendre la même méthode que pour le poignet en prenant la rotation de départ du modèle et en appliquant une rotation avec le nouvel angle.

Conclusion

Durant l'état de l'art, nous avons étudié plusieurs publications portant sur la détection et le suivi d'une main. Nous avons vu une première solution utilisant une image couleur. Cette solution était intéressante quand les caméras de profondeur n'étaient pas très répandues. Au vu des résultats fournis par S. Bilal [?], cette méthode est efficace, mais elle n'est pas adaptée à l'animation d'une main 3D. Elle permet seulement de suivre la main et de reconnaître quelques postures. De plus, les outils que l'on nous fournit nous permettent de gagner du temps sur certaines parties de notre développement et la méthode utilisant une image couleur est assez compliquée à implémenter. La DLL que nous avons utilisé se sert de la seconde méthode utilisant les images de profondeur.

Les résultats que nous avons obtenons dans notre application ne sont pas parfait et manque de précision. L'algorithme utilisé dans la DLL devient très peu précis lorsque des points sont confondus lorsqu'ils sont trop proche les uns des autres. Néanmoins, nos travaux nous ont permis de manipuler un modèle grâce à une armature. Cette méthode nous permet d'obtenir un modèle cohérent lorsque les jointures des doigts sont détectées. La combinaison des informations de la DLL et de la Kinect permet malgrès tout d'effectuer différentes actions et peut déjà servir pour diverses applications.

Ce projet a été très instructive, que ce soit sur l'apprentissage des techniques de détection et de manipulation de modèle 3D, que sur le plan de la collaboration en équipe. Nous avons énormément appris sur la rédaction d'un état de l'art, ainsi que sur la compréhension d'articles de recherche. Nous avons pu pleinement mettre à profit nos cours sur « l'initiation à la recherche et à l'innovation ». Suite à la réalisation de ce rapport, nous avons pu clairement définir les tâches et les méthodes à utiliser pour que ce projet soit un succès. Nous avons appliqué plusieurs notions qui ont été abordé lors de nos enseignements du master IVI, notamment les changements de repère pour le déplacement les articulations des doigts.