3DBLEND

Bouder Ewen, Hervé Maxime, Sadikou Faroud, Van Zelst Jelle

PROJET SAI & MIA
Université Bretagne Sud
5 janvier 2021

Resumé : Les interfaces homme-machine (IHM), aussi appelées interactions personne-machine (IPM ; en anglais : human-machine interfaces [HMI]) sont les moyens et outils mis en œuvre afin qu'un humain puisse contrôler et communiquer avec une machine.**[1]** L'évolution des interfaces d'acquisition à fait apparaître des périphériques comme le contrôleur Leap Motion **[2]**. Ce capteur n'a pas besoin d'être portés et l'utilisateur n'a qu'à placer la main dans l'espace d'acquisition au-dessus du capteur pour permettre la capture des informations tridimensionnelles de la main dans son système de coordonnées. C'est dans ce contexte que nous proposons un système de modélisation 3D en utilisant l'outil Leap Motion comme périphérique d'entrée. L'objectif ici est de proposer un outil de modélisation en utilisant des gestes utilisateurs pour interagir avec le moteur 3D.

mot clé : Modélisation, gestes, classification.

1. Introduction

La modélisation 3D est un domaine de l'infographie visant à représenter une entité à l'aide de constituants (points, arêtes, faces...). L'interface homme machine est également un enjeu important depuis le début de l'utilisation des ordinateurs. La capture de mouvement est quant à elle un domaine assez récent et constitue un paradigme innovant dans l'interaction homme machine.

Ce projet a été proposé dans le cadre de notre deuxième année de Master informatique AIDN de l'université Bretagne sud De Vannes. Le but étant de proposer une façon de modéliser des objets "simples" dans un premier temps à l'aide du mouvement des mains.

Certains travaux existants ont réfléchis à des approches de couplage entre différents périphériques afin de faciliter la modélisation. Xiao Yu et Peng Qingjin de l'université de Manitoba au Canada dans leur papier **[3]** ont proposé une interface de conception basée sur le geste de la main pour l'examen des modèles de la CAD (Computer-aided Design) dans des environnements virtuels. Ils proposent dans leur travail un système permettant d'effectuer des commandes telles que : la translation, la rotation et le scaling.

Jian Cui et Alexei Sourin ont dans leur papier **[4]** réfléchis à comment les interfaces de modélisation 3D existante peuvent être améliorées par le suivi optique des mains. Dans cette optique, ils ont proposé d'utiliser une interaction bimanuelle et séparer les fonctions d'une main. La main dominante pour les commandes manuelles : Position et rotation en 3D, tandis que l'autre main contrôle la préhension et la libération.

Toutes ces approches de solution propose des systèmes plus ou moins limités en terme de commande réalisable ou ne sont pas totalement contrôlables par des gestes (On assiste à un mixage entre les différentes techniques d'IHM).

Nous proposons donc un système avec une approche différente. Le choix de transformation et le passage d'une transformation à une autre se fait entièrement par reconnaissance de geste de l'utilisateur. Pour cela, nous nous sommes basés sur le logiciel Blender fournissant une interface et une API intéressante pour le développement de plugin ainsi que du Leap Motion pour capturer, traiter et exploiter les mouvements des mains.

2. Choix des outils

2.1. Plateforme

Lors du choix du moteur 3D, nous ne comptions pas choisir une plateforme trop bas niveau (openGL pur...) mais s'orienter plutôt vers

des solutions plus développées telles que Unity, Maya, Blender... Nous nous sommes surtout intéressés à Blender, logiciel 3D open source disposant d'une API en python, langage avec lequel nous étions assez familier et surtout disposant également de framework intéressant en reconnaissance de gestes. De la même façon, même si Unity représentait également une bonne alternative en C#, celui-ci venait muni d'outils dispensables pour notre projet (moteur de jeu, prefabs...) faisant d'Unity une solution plus lourde que Blender, une application légère. Concernant le choix de Blender avec les autres applications dites de modélisation 3D (Maya, autodesk cinéma 4D,...), la gratuité, la légèreté, la modularité (API), le regain d'intérêts pour Blender ces dernières année ainsi que de sa communauté en ont fait un atout majeur dans notre choix de plateforme.

2.2. Matériel

Enfin, l'idée du projet étant d'utiliser ses mains pour modéliser, nous nous sommes naturellement tourné vers le Leap Motion étant un matériel puissant, assez abordable et facile d'installation. L'usage du Myo Capteur n'était pas forcément à exclure mais nous avons préféré nous tourner vers le Leap en raison de sa plus grande précision et flexibilité notamment pour extraire et utiliser les caractéristiques propres aux doigts de la main.
L'aspect réalité virtuelle a été également évoqué durant le début du projet. Ceci étant, en vue de la crise sanitaire et l'accès plus compliqué au HTC Vive de l'UBS, cet aspect a été mis de côté.

Une étape du projet pendant la mise en place à consisté à faire communiquer notre matériel (Leap motion) avec notre plateforme. (Blender). En effet nous avons utilisé une version récente de Blender (2.9), et celui-ci utilisait une version également assez récente de python (3.7.7). Il nous était indispensable d'utiliser une version de python 3.X pour l'usage des librairies de reconnaissance de geste.

Le fabricant du Leap motion fournissant un SDK ayant quelque peu laissé tomber Python au profit d'une API en C et C# (pour Unity), il nous a été quelque peu laborieux pour adapter le tout avec Blender, le SDK étant prévu pour python 2.X. Dans un premier temps nous avons pu utiliser l'interface qu'ils mettaient à disposition en utilisant SWIG et visual studio pour exporter et recompiler le SDK Version 3 du Leap en Python 3.7.7 [5]. La version 4 ne disposant pas de documentation pour wrapper et recompiler pour Python 3.7.7. Cependant, après un échange avec le support il nous a également été recommandé d'utiliser la dernière version du SDK (V4) et nous avons pu tant bien que mal réussi à générer une librairie du contrôleur du leap motion pour python 3.7 sur la dernière version du sdk (V4) du Leap motion. Celui-ci comportant des améliorations dans la prélève des caractéristiques de la main. (Précision, occlusion...)

Ceci étant, il est important de préciser que notre projet a été développé et réfléchi dans une optique d'indépendance vis à vis des plateformes (Blender) et du matériel (Leap motion) utilisé afin de pouvoir éventuellement être repris et exporté aisément sur d'autres supports.

3. Module 3D

Le module 3D a donc été développé avec l'API de Blender. Nous avons pu proposer différentes opérations sans forcément tous les intégrer avec la solution finale du Leap Motion. Notre plugin, après son installation dans Blender, se présentera sous la forme d'un panneau accessible dans la partie Propriété de Blender (en bas à droite) dans n'importe quel onglet sous un volet du nom "3DBLEND". Ce volet comportera un bouton pour toutes les actions qu'il est capable d'effectuer, ainsi qu'un bouton "Use Leap" permettant le lancement et l'usage de la solution en mode Leap Motion et reconnaissance de geste. Il est possible de voir les informations de debuggage en activant la console dans Blender. *Les différentes opérations disponibles sont les suivantes :*

3.1. Opérations

Opérations de création d'objets : Il est possible de créer 3 types d'objet pour commencer une modélisation : Un cube, une sphère et un cylindre.

Opération de sélection d'objet : On peut sélectionner et passer d'un objet à un autre.

Opérations de transformations basiques : Il sera bien sûr possible d'effectuer les opérations basiques telles que la Translation (déplacement), le Rotate (Rotation), ainsi que le Scale (mise à

l'échelle) des objets dans l'espace. Il est aussi possible d'effectuer ces opérations sur une sous partie du maillage en ayant préalablement sélectionné la partie désirée (voir opération de sélection ci-dessous).

Opérations de déformation : Il a été intégré également des opérations de déformation du maillage 3D telles que le Bending (plie), le Twist (tordre), Le taper (effiler) et le stretch (étirer) sur les 3 axes différents.

Opération de sélection : Il a été pensé et réfléchi à plusieurs opérations de sélections. Par selections on entendra ici de choisir des constituants d'un maillage (à savoir soit des points, des arêtes ou des faces) afin de leur appliquer des opérations spécifiques (extrusions, transformations...).

Sélection par orientation : Dans ce mode, il suffira de faire tourner l'objet (rotation) avec la main pour en sélectionner les faces (/arêtes/points) les plus proches de l'observateur (point de vue).

Sélection à l'aide d'objet virtuelle : Dans ce mode, un objet représentant un crayon apparaît à l'écran et pourra être manipulé à l'aide des mains (leap motion) pour sélectionner les faces (/arêtes/points) en rentrant en interaction (collisions) avec.**[6]**
C'est le deuxième mode d'interaction qui a été privilégié et intégré dans la solution, celui-ci étant plus intuitif et intéressant à utiliser. D'autant plus que le premier mode de sélection est moins efficace (imprécis) à utiliser lorsque le maillage se complexifie davantage.

Opération d'extrusion : Ensuite, il est possible d'effectuer les opérations d'extrusions du maillage en fonction de la sélection faite par l'outil de sélection. L'extrusion consiste à "développer" une partie du maillage pour poursuivre la modélisation.

Opération de sauvegarde / chargement : Enfin il est possible de sauvegarder un maillage et d'en charger un. Cependant cela nécessite l'utilisation du clavier pour la saisie de l'objet à sauvegarder ou à charger.

Opération de subdivision du maillage : Enfin il est possible de subdiviser toutes les arêtes du maillage afin d'obtenir un maillage plus complexe rapidement afin d'avoir plus de contrôle et de liberté possible sur l'objet.

Enfin, le dernier bouton "Use Leap" servira à lancer l'interaction avec les gestes dans les menus et les différents outils.

La plupart de ces opérations utilisent ce qu'on appelle un "Modal" dans Blender à savoir une classe spéciale qui va prendre en charge l'objet sélectionné et appliquer les opérations désirées. La plupart des opérations exploitent les informations classiques comme la position de la main notamment pour la translation ou bien la distance entre les deux mains pour le scale etc... Ces usages ont été proposés de façon assez intuitive, et il est relativement facile de les éditer dans le code des modals en question en modifiant directement la partie concernant. Une optique du futur possible du logiciel serait de proposer à l'utilisateur de choisir et de pouvoir configurer différents modes d'usage pour un outil comme par exemple pour le scale avoir le choix entre la distance entre les deux mains, ou bien par exemple la hauteur de la main, ou encore l'écart entre ses doigts etc...

Le tutoriel sur l'usage de chaque module est disponible dans les livrables du projet.

Enfin, nous avons rajouté un FeedBack sur l'interface (retour textuel) pour indiquer à l'utilisateur dans quel menu il se trouve, et de quelles opérations il dispose.

4. Reconnaissance des gestes

4.1. Collecte des données

Les expériences ont été réalisées sur un ensemble de données que nous avons collecté avec l'aide de 3 personnes. (Voir les détails des conditions de collecte en annexe). La base de données est constituée de 8 classes de gestes statiques dont 5 proviennent de la langue de signe polonaise. Chaque geste est réalisé par 3 différentes personnes. A chaque réalisation, on récolte les données sur 100 frames. Ce qui fait un total de 300 observations par classes soit $300 * 8 = 2400$ frames au total. Avec une frame représentant la réalisation d'un geste statique à un instant t.

4.2. Extraction des caractéristiques

Avec le contrôleur Leap Motion, on récupère les informations sur les coordonnées des 5 doigts de la main **[7]** et de la paume de la main.

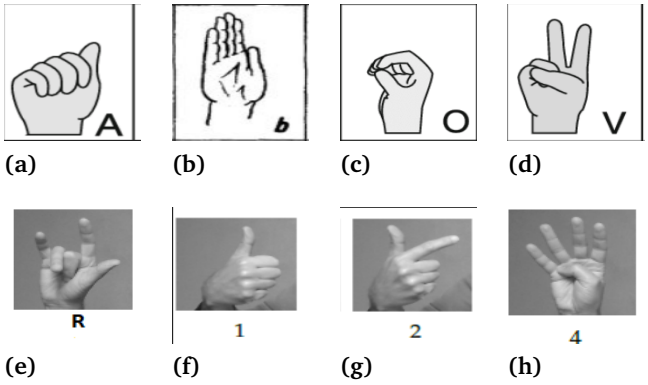

FIGURE 1: Illustration des gestes de la base de donnée

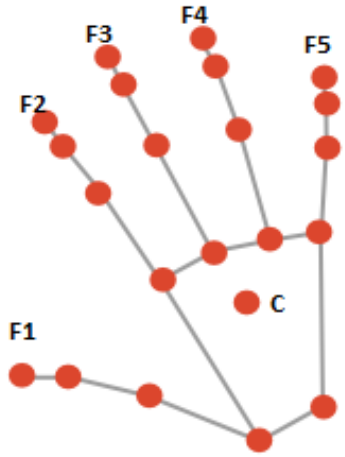

FIGURE 2: Modèle de la main.

Une posture (ou geste) est représentée par une frame. Et une frame peut être considérée comme la combinaison de différentes caractéristiques. Une frame est donc constituée d'un vecteur de distance entre les doigts adjacents.

$$D_{i,i+1} = \|\overrightarrow{F_i F_{i+1}}\|(1,2,3,4)$$

On obtient ainsi par frame un vecteur de caractéristique de longueur 10 qui se présente comme suit :

$$[D_{1,2}, D_{1,3}, ..., D_{4,5}]$$

4.3. Pré-traitement des données

Suite à l'extraction des caractéristiques, on obtient des données avec de grandes variations. Afin de corriger cela, une étape de normalisation a été appliquée aux données. On ramène ainsi les valeurs dans l'intervalle $[0, 1]$.

```
X_std = (X - X.min(axis=0)) /
    (X.max(axis=0) - X.min(axis=0))
X_scaled = X_std * (max - min) + min
```

(min = 0, max = 1 dans notre cas).

4.4. Modèle de classification

Quatres classifieurs ont été testés : un vecteur de support avec un noyau linéaire, un modèle des k plus proches voisins avec $k = 5$ une forêt aléatoire et un réseau de neuronne convolutionnel.

Le tableau suivant présente les paramètres des classifieurs.

TABLE 1: Configuration des classifieurs

Classifieurs	Paramètres	Valeurs
SVM-Lin	noyau	linéaire
	régularisation (C)	1
	méthode multi-classe	one-vs-rest
5-NN	nombre de voisins	5
	distances	Euclidienne
Forêt aléatoire	couches convs.	2 (64 (6,6)) et (32(3,3))
	couche denses	2
	activation	Swish, softmax
	optimiseur	Adam
CNN	Nombre d'arbre	100

4.5. Évaluation des performances des estimateurs et choix de modèle

Afin d'estimer avec une meilleure précision la capacité de nos modèles à généraliser, nous avons fait usage d'une validation croisée. C'est-à-dire utiliser un échantillon limité afin d'estimer les performances générales du modèle lorsqu'il est utilisé pour faire des prédictions sur des données non utilisées pendant l'apprentissage du modèle. Les données d'apprentissage divisées en k = 5

sous ensemble. Pour chacun de ces sous-ensemble k, l'apprentissage s'est donc fait sur 4 sous ensemble le modèle est validé sur la partie restante des données. La mesure de performance rapportée par la validation croisée 5-fois est alors la moyenne des valeurs calculées dans la boucle. Ce qui donne :

Le tableau suivant présente les paramètres des classifieurs.

TABLE 2: Accuracy de la validation croisée par classifieur

Modèle	SVM-Lin	5-NN	Forêt aléa.	CNN
Exactitude	99,22	100	100	87.3

On obtient une précision de 100% dans le cas des k plus proches voisins et de la forêt aléatoire. La figure suivante présente en détail le rapport de classification avec le classifieur forêt aléatoire qui a été choisi pour la suite du projet.

	precision	recall	f1-score	support
1	1.0000	1.0000	1.0000	63
2	1.0000	1.0000	1.0000	83
4	1.0000	1.0000	1.0000	80
A	1.0000	1.0000	1.0000	64
B	1.0000	1.0000	1.0000	85
O	1.0000	1.0000	1.0000	82
ROCK	1.0000	1.0000	1.0000	83
v	1.0000	1.0000	1.0000	60
accuracy			1.0000	600
macro avg	1.0000	1.0000	1.0000	600
weighted avg	1.0000	1.0000	1.0000	600

FIGURE 3: Rapport de classification

4.6. Reconnaissance des gestes par images via CNN

Afin de tester d'autres méthodes pour la reconnaissance des gestes que l'extraction de features sur les coordonnées de la main, nous avons aussi essayé une approche de reconnaissance par réseau de neurones convolutionnel. Le CNN établi est entraîné sur des images venant du dataset Multi-modal Leap Motion dataset for Hand Gesture Recognition **[8]**

Afin de limiter les calculs, les images de ce dataset ont été compressées en taille. En effet la taille d'origine de ces images était de 416 * 275 et après compression seulement de 100

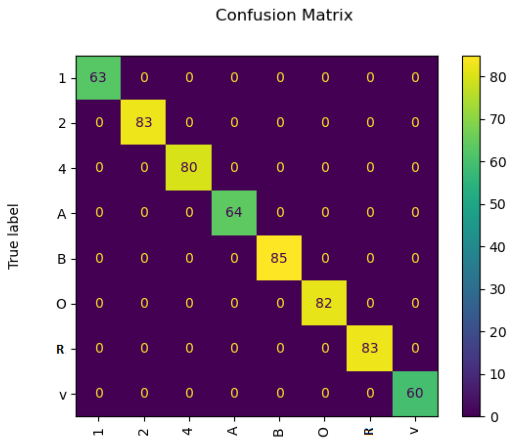

FIGURE 4: Matrice de confusion (Forêt aléatoire)

* 60. Malgré cette compression, ces nouvelles images représentent tout de même 1,5 Go de données sur disque, et limitent la vitesse d'entraînement du réseau. Ce dernier est composé de plusieurs couches convolutionnelles et denses, le tout avec une couche dropout afin de limiter le sur-apprentissage. Toutefois, même après un nombre conséquent d'époques d'entraînement, ce modèle n'obtient une précision que de l'ordre de 80% sur des données de test fournies par le dataset. Ceci est bien inférieur aux méthodes présentées précédemment.

De plus, les calculs pour le traitement des données en temps réel sont bien trop importants pour être utilisés dans notre module. En effet il faut que notre programme soit léger et n'empiète pas sur les performances de Blender car ceci affecterait grandement l'expérience utilisateur. Or lire plusieurs dizaines d'images par seconde, les dimensionner, les faire traiter dans notre CNN pour finalement détecter le mouvement correspondant est bien trop important en termes de puissance de calcul requise pour que l'application se déroule de façon inaperçue pour l'utilisateur. Ceci est accentué davantage par le fait que si l'utilisateur ne possède pas les bons drivers GPU, la version de tensorflow exploitant la carte graphique ne serait pas utilisée, augmentant encore significativement les temps de calcul et rendant probablement l'application inutilisable.

Ce module sert donc de preuve de concept et bien qu'il est possible d'effectuer la reconnaissance de gestes par cette méthode, elle est bien trop gour-

mande et peu précise pour être utilisée dans notre projet.

4.7. Suivi des gestes en temps réel

Le fonctionnement du système proposé nécessite un suivi en temps réel des gestes de la main. Afin de détecter avec une meilleure précision le geste de l'utilisateur et aussi de réduire les erreurs de détection liées à certains gestes intermédiaires lors du passage d'un geste à un autre, nous proposons une méthode se basant sur une fenêtre de temps. En temps normal le leap motion renvoi 30 frames par seconde. En cas de mouvement rapide ou très rapide, on est respectivement à $55fps$ ou $115fps$. Sur une fenêtre de $1/2$ secondes, on collectionne les différentes postures prédite par le modèle par frame. Ce qui donne entre $55/2$ et $115/2$ frames. On détermine pour les différentes classes prédites sur cette durée, le nombre de frames représentant. Ensuite on en déduit la posture la plus prédite avec le pourcentage de présence. Mais avant, il faut que le pourcentage calculé soit supérieur à 60% au moins lorsque le nombre de gestes candidats dans cette fenêtre de 1/2 seconde est inférieur à 3. Sinon on répète la même action sur une autre fenêtre.

5. IHM

Notre projet posait un réel défi d'interaction homme machine. En effet, il existe encore peu de modèles de système se basant uniquement sur le mouvement des mains. L'enjeu consistait donc à proposer une façon fonctionnelle et un minimum ergonomique de l'interaction avec notre solution. Il s'agit plus précisément de la navigation entre les différents modules et outils 3D qui est au cœur de la problématique. Pour cela, nous avons proposé à l'aide d'un diagramme d'interaction un plan possible d'interaction entre les différents outils. Il est possible de le représenter sous forme de Graphes avec un état représentant un outil et une transition étant défini par un geste.

L'idée n'est pas de figer ce graphe, ce qui est géré par un module principal du projet à savoir notre contrôleur. Une optique potentiel d'améliorations du projet consisterait à proposer à l'utiliser de pouvoir non pas seulement choisir les gestes qu'il souhaite utiliser ("V", chiffres. . .) mais également

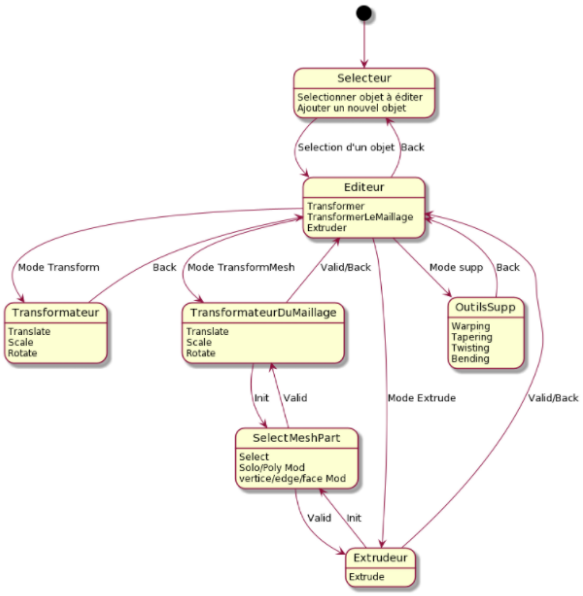

FIGURE 5: Diagramme d'interaction

de pouvoir composer son propre graphe d'outil avec les fonctionnalités qu'il souhaite utiliser ou non et les gestes qu'ils souhaite incorporer comme transition.

Nous avons conçu le programme pour qu'une classe spécifique ayant le rôle de contrôleur gère ces transitions d'états.

Contrôleur : L'idée de la classe du contrôleur va être de relier le module de reconnaissance de geste et des différents outils 3D développés précédemment. Celui-ci créer un sous processus (thread) et écoutera la sortie du module de reconnaissance qui renverra les différents gestes reconnus ("V", "1","A" etc. . .). Enfin selon les gestes reçu il interprètera et activera tel ou tel module en fonction du plan d'interaction défini ci-dessus. Voir le diagramme de classe. Une fois un module 3D en exécution initié par le contrôleur celui-ci aura la main et exécutera et rafraichira sa fonction "d'animation" en recevant les entrée de l'utilisateur par le leap motion (exemple la position de la main pour effectuer une translation) en permanence, ceci étant, le module en cours de progression écoutera aussi les messages que peut lui envoyer le contrôleur, à savoir dans tous les cas un geste de validation ("V") qui mettra fin au module et conservera les modifications apportés, ainsi qu'un geste d'annulation ("1") qui

mettra également fin au module mais annulera les modifications dans la majorité des cas. De plus, le module Contrôleur pourra également envoyer des messages spécifiques dits de “Switch” (voir “Switch2”) permettant de basculer entre différentes options du module en cours. On peut l'exemple du module de déformation, qui pourra basculer entre ses différentes options de méthodes (Blend, twist, taper...) ou bien encore de basculer (avec l'autre commande de switch) entre les différents axes de déformation (X/Y/Z...)

Reconnaissance vocale : Nous avons également eu comme potentielle piste d'amélioration l'ajout d'un module de reconnaissance vocale qui aurait permis une autre interactions utilisateurs, non pas par des gestes de la main via l'utilisation du leap motion, mais par le biais d'un micro.
En effet, il aurait été tout à fait possible de choisir un “mode” directement via des paroles en utilisant des modules python tel que PyAudio et SpeechRecognition qui analysent les paroles afin de générer une chaîne de caractères.

Il aurait ainsi suffit de comparer la chaîne générer avec les options possibles. Si la chaîne correspond à une des options, il y aurait eu alors l'exécution d'un code de validation similaire à ce qui à été fait pour la reconnaissance de gestes par le contrôleur.

6. Perspectives d'amélioration du projet

Comme parfois mentionnés dans les parties précédentes voici les optiques d'améliorations du projet :

- Optimiser les interactions et leur retour dans le système
- Proposer une interface utilisateur pour configurer les gestes désirés
- Proposer une interface utilisateur pour designer soi-même son graphe d'état entre les différents modules qu'on souhaite utiliser ou non
- Proposer des méthodes d'interactions différentes pour l'usage de module 3D au choix de l'utilisateur
- améliorer le crayon de sélection en lui donnant une orientation et éventuellement un laser (pointage) partant de la mine du crayon
- Proposer d'autres fonctions 3D (sculpting, particules, etc...) bien que cela implique de faire des choix dans le graphe d'état
- Incorporer un module de reconnaissance de voix pour lancer des actions indépendamment des gestes
- Exporter sous d'autres plateformes le projet (Unity) ou d'autre matériel (myocapteur...)
- Exporter le projet dans un environnement VR

7. Conclusion

Nous avons donc pu proposer une solution de modélisation 3D à l'aide du mouvement des mains à l'aide de Blender et du Leap motion. Pour cela nous avons divisé le projet en plusieurs modules, à savoir un module gérant les formes 3D, un module de reconnaissance de gestes et enfin après avoir formaliser un plan d'interaction/navigation un module de contrôle. Ce projet nous a permis de mettre en pratique et d'exploiter les connaissances et les notions abordés dans les unités d'enseignement de simulation et application interactives ainsi que de mouvement et intelligence artificielle. De plus, nous avons pu continuer de consolider notre méthodologie de travail acquise durant notre cursus universitaire à travers la gestion de projet grâce aux outils utilisés tels que Trello et GitLab. C'est en autre un projet qui nous a réellement plu durant sa conception.

Références

[1] Wikipédia Interactions homme-machine — Wikipédia, l'encyclopédie libre, [En ligne; Page disponible le 8-décembre-2020], **2020**.

[2] UltraLeep Leap Motion Controller `https://www.ultraleap.com/product/leap-motion-controller/`, (consulté le 08/12/2020).

[3] Xiao, Y.; Peng, Q., et al. In *DS 87-8 Proceedings of the 21st International Conference on Engineering Design (ICED 17) Vol 8: Human Behaviour in Design, Vancouver, Canada, 21-25.08. 2017*, **2017**, 239–248.

[4] Cui, J.; Sourin, A. In, **2017**, 1–4.

[5] Ultraleap Implementation of LeapAPI (1.0-3.0) using LeapC, **2020**.

[6] Foundation, B. BVH tree structures for proximity searches and ray casts on geometry https://docs.blender.org/api/current/mathutils.bvhtree.html#mathutils.bvhtree.BVHTree.overlap, (consulté le 24/11/2020).

[7] Foundation, B. Leap Motion, Finger class https://developer-archive.leapmotion.com/documentation/python/api/Leap.Finger.html, (consulté le 11/12/2020).

[8] GTI MultiModal Hang Gesture Dataset http://www.gti.ssr.upm.es/data/MultiModalHandGesture_dataset.html, (consulté le 08/12/2020).