

Rapport de Projet

*Deep Learning Projet libre : "Fast and deep deformation approximations.",
Bailey, S. W., Otte, D., Dilorenzo, P., & O'Brien, J. F. (2018)*

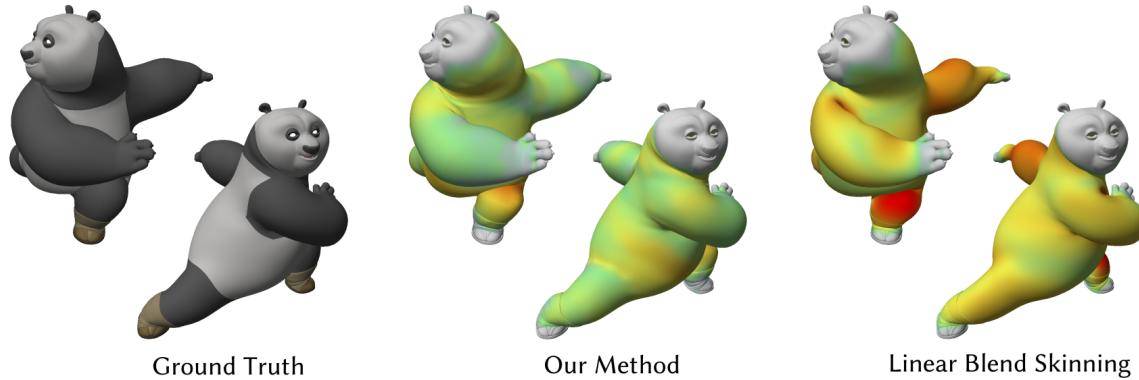


Figure 1. L'image d'illustration du papier implémenté dans ce projet et qui traite d'une méthode d'approximation de déformations non-linéaires reposant sur les réseaux de neurones. De gauche à droite : La déformation à approximer, le résultat de leur méthode et enfin le résultat d'une approximation ne reposant que sur des déformations linéaires. Les deux derniers maillages sont colorés en fonction de leur distance au maillage "ground truth" allant de 0 cm (gris) à 6 cm (rouge).

Thomas Vallentin

14/12/2022

Deep Learning – IGM M2 Sciences de l’Image

INTRODUCTION

Dans les domaines de l'animation 3D, des VFXs et du jeu vidéo, les personnages sont dans l'écrasante majorité des cas représentés par des maillages de faces. Pour donner vie à ces maillages statiques, les artistes et techniciens des studios produisent des "rigs" : des systèmes procéduraux qui calculent la position de tous les points qui composent le maillage pour une pose donnée. On découpe souvent cet ensemble en deux sous-catégories : les rigs dits "temps-réel" principalement utilisés dans le jeu-vidéo et qui doivent s'évaluer en un temps extrêmement court (souvent au prix d'une perte de qualité), et les rigs dits "films" qui n'ont pas cette contrainte et cherchent à produire les déformations les plus riches possible peu importe le coup.

Au sein de ces derniers, on y trouve des sous-systèmes très avancés allant de la conservation de volume physique à du smoothing intelligent ou encore à des systèmes de peau glissant sur des muscles virtuels simulés. Cette complexité peut rendre les rigs de film très lent à évaluer, ce qui peut ralentir la fluidité du travail des animateurs, la qualité des animations qu'ils réalisent ou encore limiter l'agilité du pipeline de production. De nombreuses recherches ont eu lieu au cours des dernières années pour proposer des déformations riches et de qualité à moindre coût, permettant ainsi de profiter des avantages des deux mondes.

Pour ce projet, j'ai choisi d'implémenter le papier de recherche "*Fast and deep deformation approximations*", Bailey et al. [1], au sein duquel les chercheurs de l'université de Berkeley et du studio DreamWorks proposent une méthode permettant d'approximer les déformations non-linéaires de rigs complexes à l'aide des réseaux de neurones.

MATÉRIEL

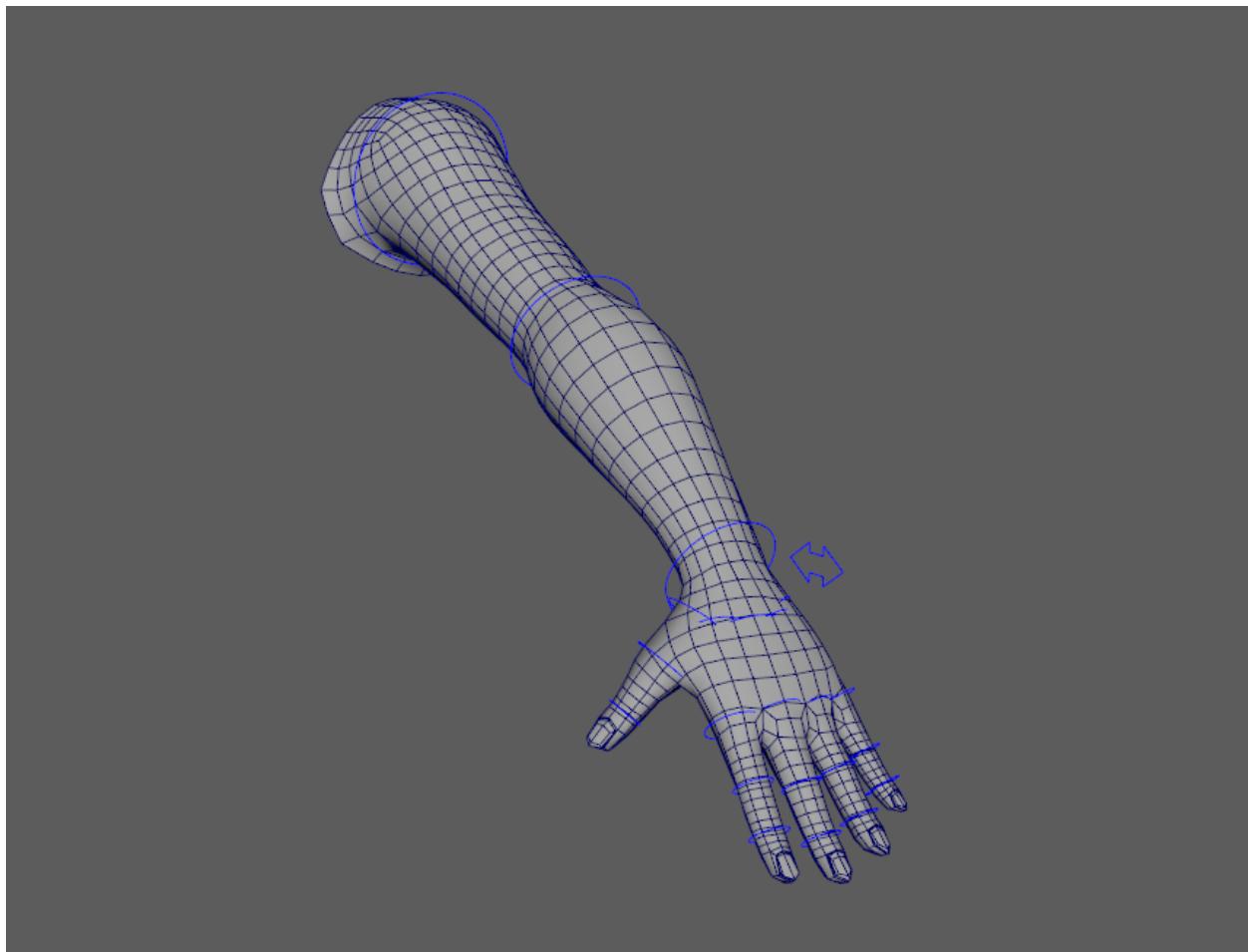
Ressources

Le projet est réalisé à l'aide du logiciel Autodesk Maya [2], un des logiciels d'animation 3D les plus utilisés dans les productions de film d'animation, de VFX et de jeux-vidéos. Ce dernier possède un kit de développement en C++ et en Python permettant d'écrire des plug-ins et d'enrichir ainsi ses fonctionnalités.

Nous utiliserons ici les APIs Python de Maya, ce langage étant idéal pour travailler avec les réseaux de neurones et bien adapté pour des projets de cette envergure. Les réseaux de neurones seront construits à l'aide de Tensorflow / Keras [3] dont il existe un binding Python. Seule ombre au tableau, Python ne brille pas par sa rapidité. Les performances de notre système seront donc bien inférieures à celles reportées dans le papier de recherche. L'objectif de ce projet n'étant pas de produire un résultat pour la production, cet aspect est négligeable.

Le rig

Le rig que nous allons approximer ici est un rig de bras que j'ai réalisé et qui repose spécifiquement sur des déformations non-linéaires, afin de mettre en valeur les capacités du système.



Il est composé de 3 couches :

- Une base de “hard-skinning” : Chaque point du maillage est attaché à un os virtuel, un “joint”. Lorsqu’un joint subit une transformation, il applique également cette transformation aux points qui lui sont attachés. Ici, chaque os virtuel correspond à un os réel (humérus, radius, métacarpes, phalanges, etc...) et entraîne les points qui définissent la partie du bras que l’os représente.
- Une seconde couche de “blendshape” : On sculpte une copie du maillage dans une pose donnée et on déforme le maillage initial pour lui donner la forme sculptée. Dans notre cas, on utilise ce système pour représenter la forme du biceps contracté qu’on pondère en fonction de l’angle entre le bras et l’avant bras. Ainsi, si l’angle est plat, on n’applique aucune déformation : le biceps est relâché ; plus

l'angle devient aigu, plus la déformation est appliquée : le biceps se contracte et prend la forme sculptée.

- Enfin, on applique une couche de “DeltaMush” [4] : Ce composant permet de lisser un maillage ayant subi des déformations en prenant en compte sa forme initiale. On ne perd donc aucun détail qui était présent dans le maillage original, seules les zones ayant subi de fortes déformations locales (les pliures des articulations, etc...) seront lissées. Cette couche assez coûteuse est particulièrement non-linéaire et permet d'obtenir un résultat de bonne qualité malgré la simplicité de notre rig. Ce genre de système est très commun en production pour gagner en qualité à faible coût, contre quelques lenteurs lors de l'évaluation du rig.

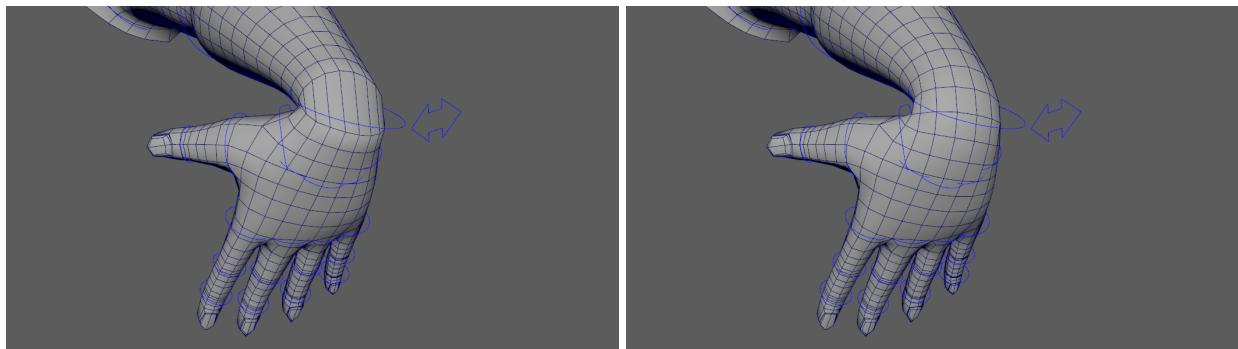


Figure 2. Les déformations du poignet avant et après l'application de la couche “DeltaMush”.

Le rig que nous utiliserons ici n'est pas aussi complexe que ceux que nous évoquions dans l'introduction, son évaluation est donc plutôt rapide. Cependant, il intègre des déformations similaires à celles qui pourraient être générées par de tels rigs, notamment sur le point de la non-linéarité (à l'aide du DeltaMush). Cela en fait donc un bon candidat pour évaluer les procédés introduits dans ce papier.

PROCÉDURE

Déformation non-linéaire

Le système que nous allons construire a donc pour but d'approximer les déformations complexes générées par un rig (en l'occurrence, notre rig de bras). Pour y arriver, les auteurs proposent de considérer ces déformations comme la somme d'une déformation linéaire reposant sur un squelette de joints et d'une déformation non-linéaire :

$$\mathcal{D}_{\text{complex}} = \mathcal{D}_{\text{lin}} + \mathcal{D}_{\text{nonlin}}.$$

Ils proposent que la partie linéaire soit réalisée à l'aide d'une couche de hard-skinning linéaire qui peut être calculée assez facilement au préalable à partir de la déformation finale, à l'aide d'une méthode des moindres carrés par exemple [5]. Notre

rig reposant sur une couche similaire, nous nous baserons directement sur celle-ci.

De son côté, la partie non-linéaire de la déformation n'est pas aussi simple à calculer. Les auteurs proposent donc de faire apprendre cette partie par notre réseau de neurones. Ce résidu non-linéaire est défini comme la différence de position entre des sommets subissant uniquement la déformation de la couche linéaire et ceux subissant la déformation complète : $\mathcal{D}_{nonlin} = \mathcal{D}_{complex} - \mathcal{D}_{lin}$. Comme nous réalisons ce projet pour des maillages 3D, le résultat de notre réseau sera donc un tableau contenant un vecteur pour chaque vertex du maillage déformé.

Les déformations d'un rig sont généralement déclenchées par les transformations des joints du squelette et c'est le cas dans notre rig de test. Les transformations d'un objet 3D sont représentées à l'aide de matrices 4x4 (en coordonnées homogènes) dont les valeurs constituent donc les inputs de notre réseau. Chaque matrice contient 16 éléments, cependant le 4e élément de chaque colonne (pour des matrices de transformation "column-major") est négligeable dans notre cas de figure car il est constant : ce sont les coordonnées homogènes de nos matrices. Nous n'utiliserons donc que 12 éléments par joint, soit un total de $12 * 22 = 264$ éléments.

Il est également possible qu'un rig plus complexe ait d'autres contrôles que les joints du squelette (un attribut contrôlant l'intensité du DeltaMush ou d'une BlendShape par exemple) mais nous ne supporterons pas ce cas de figure dans le cadre de ce projet. Si c'était le cas, il suffirait de lister tous ces attributs et d'ajouter leurs valeurs à la liste des inputs.

Génération du dataset

Maintenant que nous connaissons nos inputs et nos outputs, il est temps de générer le dataset sur lequel nous entraînerons le réseau. Chacun des joints de notre squelette peut subir des transformations de translation et de rotation sur les 3 axes et certaines déformations peuvent survenir grâce à l'action combinée de plusieurs joints. Afin de capturer au mieux la richesse de toutes ces combinaisons, les auteurs proposent de générer des sets de poses à la volée en définissant les transformations de chaque contrôleur d'animation (qui contrôlent les joints du squelette) à l'aide d'une distribution gaussienne. Chaque contrôleur d'animation possède une certaine amplitude de mouvement. Par exemple, un coude peut tourner sur un seul axe et sur un intervalle allant de 0° (coude déplié) à environ 120° (coude complètement plié) alors qu'une épaule peut tourner sur trois axes sur une amplitude allant d'environ -90 à 90° dans chaque axe. Ces limites varient en fonction de la morphologie de chaque personnage et doivent donc être stockés dans le rig lui-même. Les auteurs proposent d'échantillonner les transformations sur chaque axe de rotation et de translation à l'aide d'une gaussienne centrée sur l'amplitude de mouvement du contrôleur et possédant un écart type équivalent à la moitié de l'intervalle.

Les poses générées sont ainsi des combinaisons “probables” des contrôleurs et non des poses réalistes du rig (en témoigne la Figure 3.). Ces dernières permettent de capturer le large panel de possibilités du rig tout en concentrant les poses vers le centre de l’amplitude de mouvement des contrôleurs, qui correspond habituellement à leur état le plus courant.

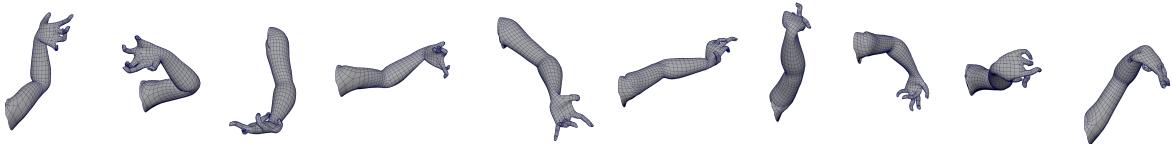


Figure 3. Exemples de poses aléatoires générées à l'aide d'un échantillonnage des transformations basé sur une répartition gaussienne.

D’après les essais que j’ai pu réaliser, générer un dataset de quelques centaines de poses permet de capturer l’entièreté des mouvements de notre rig de bras. Un plus grand nombre d’échantillons ne permettant pas d’obtenir un gain significatif, nous utiliserons donc un dataset de 1000 échantillons.

Structure du réseau

Les auteurs proposent une structure de réseau très simple, composée uniquement de deux couches “Fully Connected” utilisant la fonction d’activation “tanh” et dont la taille est à définir empiriquement (ils conseillent d’utiliser 128 neurones par couches ce qui assez peu, un point sur lequel nous reviendrons plus tard). Un dernier layer linéaire est connecté en sortie du réseau dont la taille est $3 * nbSommets$, un élément pour chaque composante x, y, et z de chaque sommet du maillage déformé.

En suivant les instructions que nous avons évoqué précédemment, on peut approximer un modèle simple composé de quelques centaines de vertex et déformé par un seul joint.

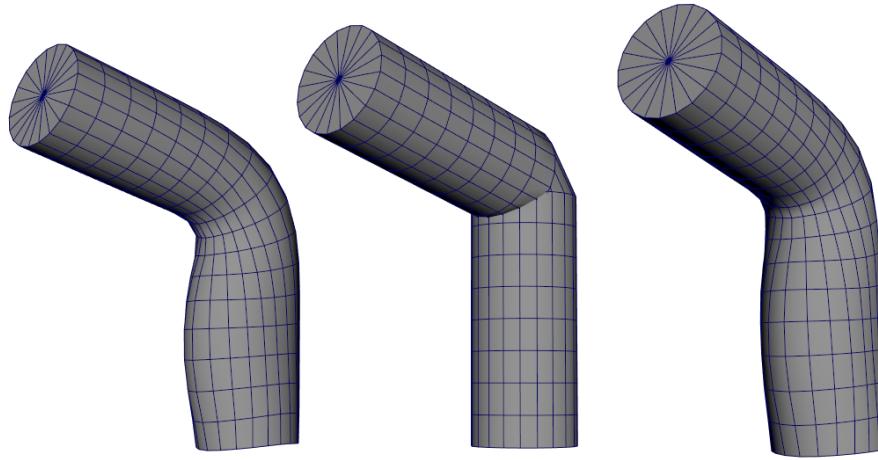


Figure 4. Approximation d'un modèle simplifié du bras composé uniquement d'un coude. De gauche à droite : la déformation de référence, la base de déformation linéaire et enfin la combinaison de la couche linéaire + et de l'approximation non-linéaire.

Cependant, ce modèle ne se généralise pas correctement lorsqu'on le porte sur un rig dont le squelette contient un nombre le nombre de joints plus important et sur des maillages comportant un grand nombre de sommets. En effet, bien que d'un point de vue local, les déformations ne soient impactées que par un faible nombre de joints, les liens de parenté entre ces derniers ajoutent beaucoup de complexité à l'ensemble. Si on prend en exemple le rig très simple présenté dans la Figure 5., on remarque qu'une déformation des points liés au dernier joint de la chaîne, symbolisée par les flèches rouges, est entièrement dépendante des transformations de ses ancêtres, alors même que la déformation est inchangée localement. Ce phénomène est d'autant plus important que la chaîne de joint est grande car les combinaisons augmentent drastiquement le nombre d'états possibles de notre rig. Dans notre exemple, la plus longue chaîne de joint est de 6 joints (de l'épaule au bout de chaque doigts) et il serait sans doute possible d'échantillonner un nombre bien plus important de poses pour contrer ce phénomène, cependant un rig de production peut contenir des chaînes contenant plusieurs dizaines voire centaines de joints, ce qui rend la combinatoire trop importante à échantillonner.

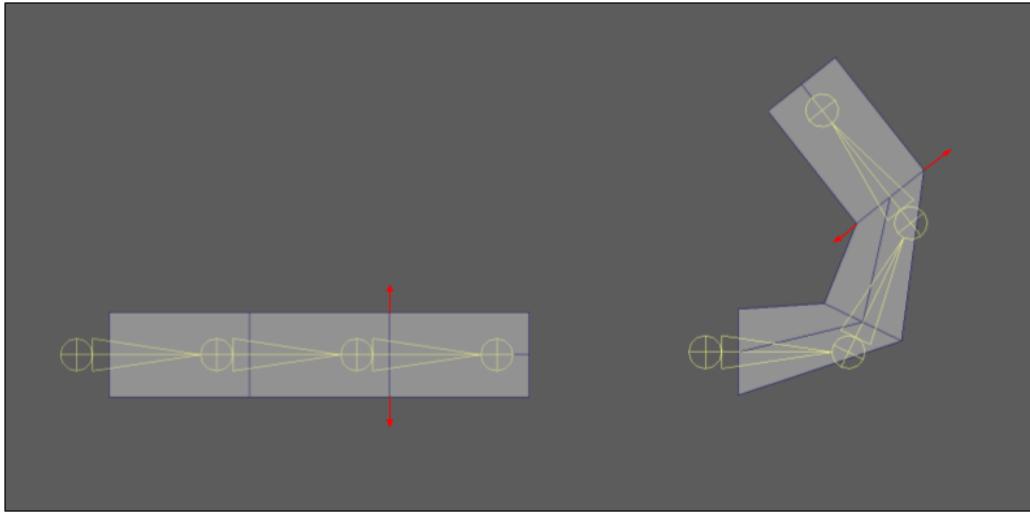


Figure 5. Impact des transformations des ancêtres des joints sur les déformations. Bien que la déformation non-linéaire (symbolisée par les flèches rouges) reste constante localement, elle change en fait drastiquement si les transformations des joints à la base de la chaîne sont modifiées.

Les auteurs proposent donc de segmenter le modèle en plusieurs partitions en se basant sur les assignations sommet-joint définies dans la couche de hard-skinning. Les déformations de chaque sous-ensemble de sommets seront localisées par rapport au joint auquel tous ces sommets sont attachés, appelé le “main joint” ou joint principal. De cette manière, les déformations qu’on cherche à approximer ne sont plus dépendantes des transformations de tous les ancêtres des joints principaux. Si on reprend l’exemple de la Figure 5., la déformation induite par les deux flèches rouge est désormais identique dans les deux cas : un déformation perpendiculaire au dernier joint de la chaîne. On limite ainsi grandement les effets combinatoires extrêmes causés par le nombre de joints qui composent les squelettes (x^{22} dans le cas de notre bras à 22 joints, $x^{[200,1000]}$ pour un rig de personnage de production) et les données sont plus simples à apprendre.

Chaque input (les transformations des joints) sera donc exprimé relativement aux transformation de son parent $T = T_{Parent_{Inv}} \cdot T_{Joint}$ et chaque output sera localisé à son, joint principal $V_{local} = T_{Joint_{Inv}} \cdot V_{Global}$. Il nous suffira d’effectuer l’opération inverse en sortie du réseau afin de rétablir les déformations dans l’espace global et les appliquer sur le maillage.

Au-delà de ces changements d'espace, l'aspect local des déformations d'un rig permet également d'optimiser la complexité du réseau. Les auteurs mettent en avant le fait qu'une déformation est habituellement très localisée spatialement : la déformation d'une phalange n'a aucun impact sur la déformation d'une épaule, et vice versa. Ainsi, plutôt que d'entraîner un énorme réseau recevant $12 * nbJoints$ inputs et produisant $3 * nbSommets$ outputs, ils proposent d'affecter un réseau à chaque partition réalisée

précédemment. On se retrouve donc avec une constellation de petits réseaux très spécialisés dans l'apprentissage des déformations d'un nombre réduit de sommets, et dont les déformations sont toutes exprimées dans le même espace (relative au joint principal de la partition).

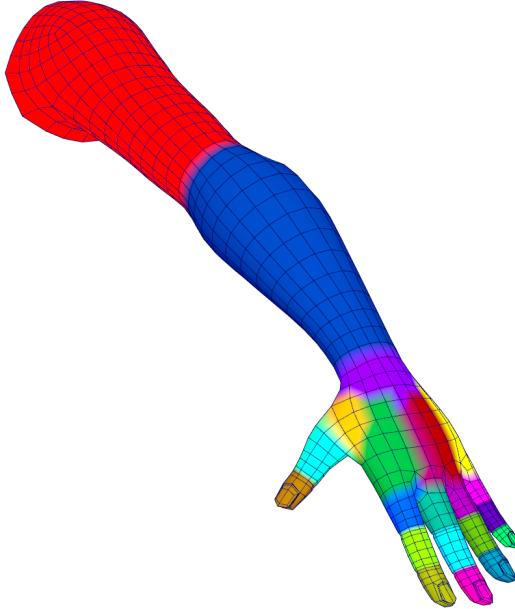


Figure 6. Partitionnement de notre rig de bras, chaque couleur correspond à une partition de sommets.

Enfin, comme les déformations ont tendance à être concentrées localement, cela veut dire que l'on peut réduire drastiquement le nombre d'inputs de nos réseaux. Par exemple, la partition représentant l'humérus et allant du bras au coude ne subit l'impact que deux joints : l'humérus et le radius. On détermine donc pour chaque réseau la liste des joints qui impactent la partition en agitant chaque joint un par un et en regardant si cette transformation a eu un effet sur la déformation des points de la partition. Dans la grande majorité des cas, une dizaine de poses par joint suffit à capturer l'effet de chaque joint sur chaque partition. Notre rig étant très simple, 4 suffisent amplement.

Tout cela simplifie encore plus le problème qui revient à approximer les déformations de quelques dizaines à quelques centaines de points en fonction des transformation d'un nombre réduit de joints en input, ce qui explique les faibles dimensions que les auteurs recommandent d'utiliser dans leurs réseaux (2 couches de 128 neurones chacunes).

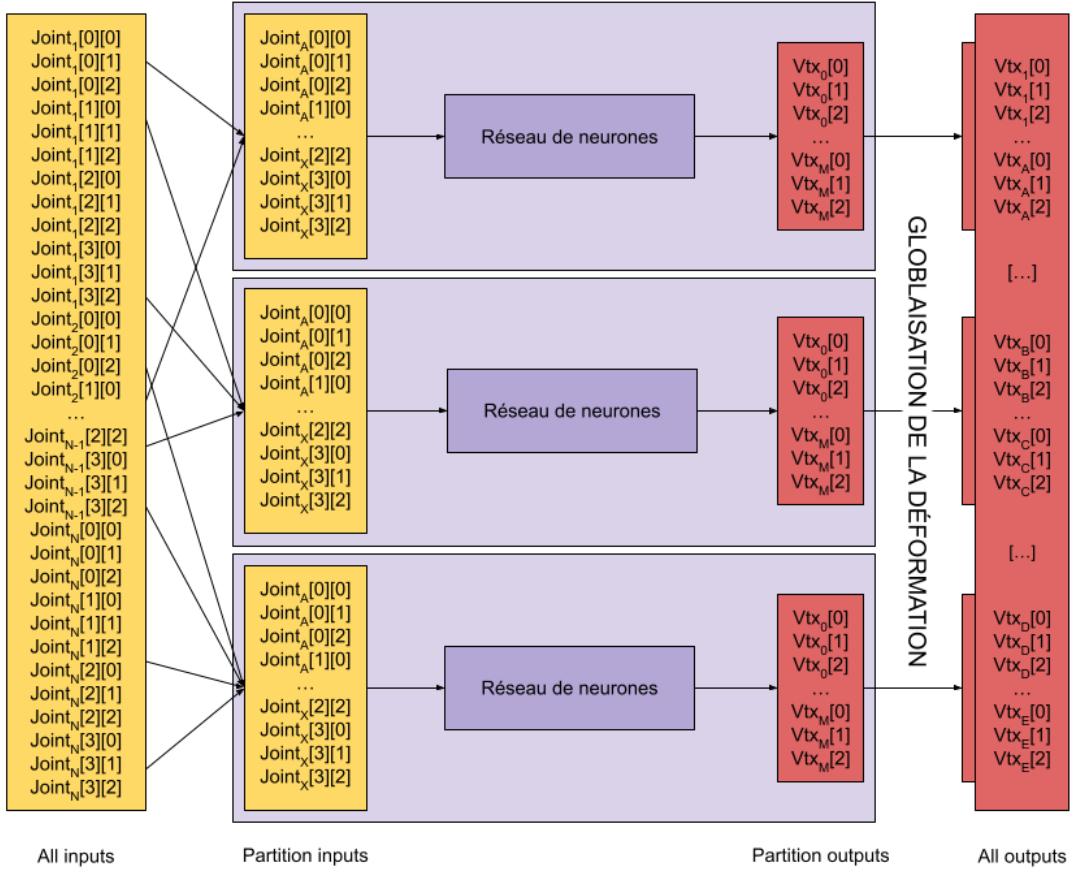


Figure 7. Résumé du processus d'approximation des déformations non-linéaires. Les inputs sont affectés aux différentes partitions en fonction de leurs impacts sur la déformation des sommets de chaque partition. Les réseaux de neurones génèrent les déformations locales de chaque partition qui sont ensuite re-transformées dans l'espace global et injectées dans les sommets du maillage.

MISE EN OEUVRE

Recording

La génération du dataset repose sur la classe Recorder. Cette dernière génère les partitions, appelée Subsets dans le code, contenant une liste de sommets et une liste des joints qui affectent la partition et qu'il faudra prendre en compte pour l'apprentissage du réseau. Elle génère ensuite un ensemble de poses et calcule les déformations non-linéaires globales de chaque pose ainsi que les transformations de tous les joints. Elle passe ensuite cet ensemble de matrices de transformation et de vecteurs aux Subsets qui vont sélectionner les sous-ensembles qui constitueront leurs inputs et outputs à partir de leurs listes de sommets et de joints d'intérêt. C'est ici que vecteurs de

déformation seront localisés au main joint de chaque Subset.

Une description du réseau complet et de chaque Subset est sauvegardée, ainsi que les inputs et les outputs de chaque Subsets.

Le module scripts/recording.py contient le code nécessaire à cette étape.

Training

L'entraînement n'étant plus lié à Maya d'une quelconque façon, j'ai choisi de le réaliser dans un notebook python. Afin d'éviter de devoir importer un nombre important de fichiers (deux par Subset), l'entraînement sera réalisé sur un serveur jupyter local. Le notebook charge la description du réseau et génère un objet SubsetModel pour chaque Subset qu'elle contient. Ces objets représentent la combinaison d'un Subset et du réseau de neurones qui apprendra les déformations du Subset. Il suffit ensuite de charger les inputs et outputs de chaque Subsets et de lancer l'apprentissage.

Tous les inputs et les outputs subissent une normalisation standard lors de la phase d'apprentissage : une division par la moyenne et une soustraction par l'écart type. Cette normalisation devra être inversée en sortie du réseau. Chaque réseau entraîné est sauvegardé sous la forme de 5 fichiers sur le disque : un fichier .h5 contenant la topologie et les poids du réseau et 4 fichiers .csv contenant les écarts type et moyennes de chaque colonne des inputs et des outputs qui nous permettront d'inverser la normalisation. On copie également le fichier .json de description écrit lors du recording et dont le contenu permet de définir la liste des réseaux, ainsi que le mapping entre les partitions et la déformation complète.

Les auteurs de l'article recommandent l'utilisation de l'optimiseur Adam et d'utiliser ses paramètres par défaut ($\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$). Celui-ci ayant produit un bon résultat lors de mes tests, je l'ai utilisé sur l'entièreté du projet sans le comparer avec les optimiseurs disponibles.

Le notebook scripts/training.ipynb et le module scripts/model.py contiennent le code nécessaire à cette étape.

Application de la déformation

Maya est un logiciel nodal, ce qui veut dire que chaque objet présent dans la scène 3D est généré à l'aide d'un graph de nœuds. Chaque nœud contient de la logique qui définit ou modifie les attributs. Les uns après les autres, ces nœuds définissent la nature et le comportement des objets 3D. Les maillages de points sont définis à l'aide de types de noeuds : les mesh qui contiennent les données des maillages et sont affichés dans le viewport du logiciel et les deformers qui altèrent les positions, les normales ou la topologie des maillages afin d'obtenir divers types d'effets. Les "couches" évoquées dans la section portant sur la structure du rig sont toutes des deformers qui modifient le maillage statique du bras pour le rendre animable.

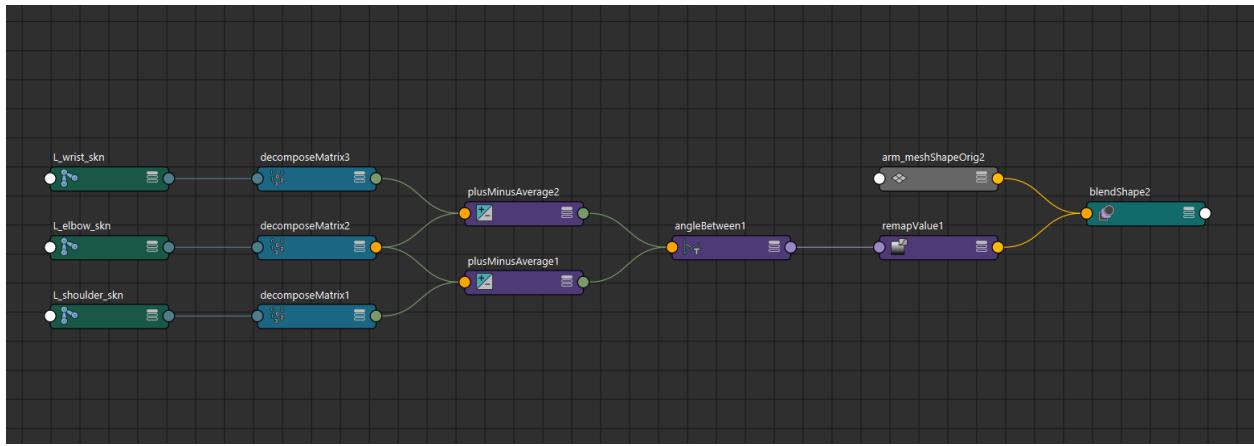


Figure 8. Exemple d'un graph de nodes contenu dans notre rig de bras, les positions de l'épaule, du coude et du poignet définissent un angle qui contrôle l'intensité d'activation d'une BlendShape gonflant le biceps.

Nous allons donc écrire un deformer qui encapsule nos réseaux de neurones et applique les déformations non-linéaires qu'ils génèrent. Ce deformer prend en entrée un fichier .json de description le partitionnement à partir duquel il va générer les réseaux de neurones, le maillage ayant subi la partie linéaire de la déformation en sortie de la couche de hard-skinning et les transformations locales de tous les joints du squelette, les inputs des réseaux. Nous allons également avoir besoin des matrices de transformation globales de tous ces joints afin de pouvoir re-transformer les déformations de tous les points, appris localement par nos réseaux, dans l'espace global.

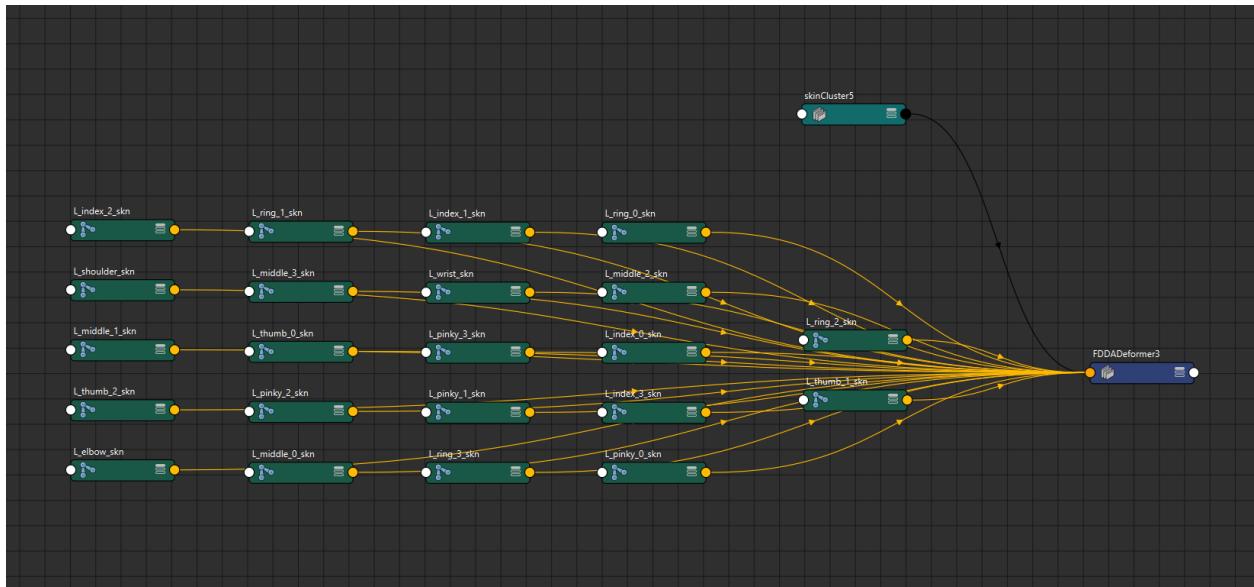


Figure 9. Le graph représentant les entrées de notre deformer : un skinCluster qui contient la partie linéaire de la déformation, et tous les joints du squelette de notre bras.

Le fichier plug-ins/FDDADeformer.py contient le code nécessaire à cette étape. Les

réseaux sont chargés dans la méthode `_load_model_file()` et la déformation est appliquée par la fonction `deform()`.

PROBLÉMATIQUES ÉTUDIÉES

Les déformations de la main

Si on se penche plus en détail sur l'apprentissage des réseaux, on constate que dans les cas les plus simples (coude, épaule, dernières phalanges des doigts), l'apprentissage se passe de manière idéale, en suivant une courbe qui s'aplatit rapidement, converge vers une erreur nulle et ne subit pas de phénomène de surapprentissage.

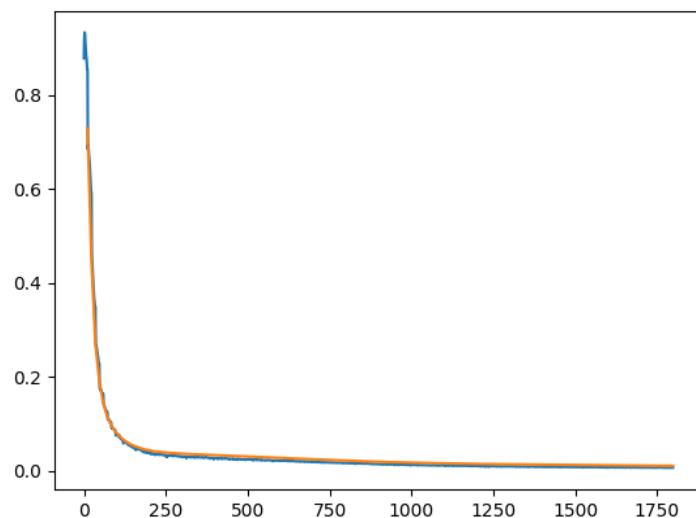


Figure 10. Courbe d'apprentissage de la partition contenant l'épaule et l'humérus. La courbe bleue représente l'erreur, la courbe orange l'erreur lors de la phase de validation.

Cependant, les zones sur lesquelles plus de joints entrent en interaction et dont le résultat est plus difficilement prévisibles ont des courbes d'apprentissage bien plus chaotiques.

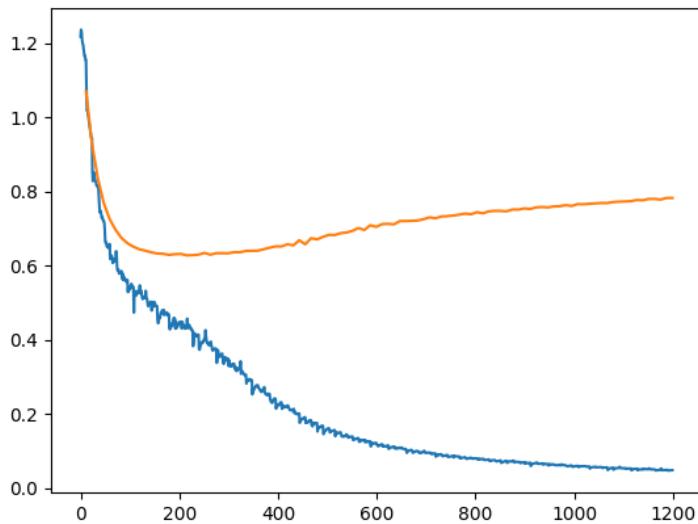


Figure 11. Courbe d'apprentissage de la partition définissant le métacarpe de l'index. On note que la courbe de validation ne converge pas comme la courbe d'erreur, traduisant un surapprentissage.

Pour contrer cet effet de surapprentissage, j'ai tenté plusieurs approches : augmenter la taille du dataset, régulariser les couches à l'aide d'un régularisateur L1 ou L2 ou encore ajouter des couches Dropout pour éliminer des poids au fur et à mesure de l'apprentissage. Cependant, aucune de ces méthodes n'a produit un résultat significativement meilleur qu'une autre, la courbe de validation se stabilisant systématiquement entre 0.6 et 0.8.

Bien qu'il ressemble à un problème de surapprentissage, le problème venait en fait des données d'entraînement. Si on se penche sur une pose générée par le Recorder, on remarque que les positions des articulations simples sont plutôt crédibles (prises séparément les unes des autres), mais que la forme de la main est extrêmement difficile à appréhender (voir figure 12.). Par défaut, on considère que les joints ont une amplitude de mouvement de -90° à 90° dans chaque axe, afin de couvrir un maximum les possibilités offertes par le rig. Cependant lorsque cela s'applique à la main, on obtient des résultats très extrêmes à cause de la proximité des joints. De plus, l'utilisation du DeltaMush de manière si prononcée rend difficile la prédiction de la forme de la main car celui-ci vient annuler une grande partie des déformations causées par les joints. Il les mélange avec les déformations induites par les autres joints et les propage à travers toute la main, ce qui lui donne un aspect très souple, voire flasque, et dilue l'effet des inputs sur les outputs ce qui complexifie énormément l'apprentissage.

Afin de résoudre ce problème, j'ai limité plus sévèrement les amplitudes de mouvement des doigts, et spécialement des métacarpes qui donnent la forme générale de la main. Ces derniers n'ont plus que 40 degrés d'amplitude ($\pm 20^\circ$) dans chaque axe et les doigts ont une grande liberté dans l'axe de l'articulation mais une liberté plus limitée dans les

autres axes. Au-delà de réduire les possibilités du rig, et ainsi de rendre le problème plus simple à approximer, on rend surtout les données d'entrée plus facilement “compréhensibles” par l'apprentissage. Cela se traduit par des courbes de validation plus proches des courbes d'erreur, mais aussi par de meilleures déformations de la main par rapport au dataset précédent, même en dehors de la l'amplitude limitée dans laquelle nous avons entraîné nos réseaux (ce point est évoqué dans la section suivante).

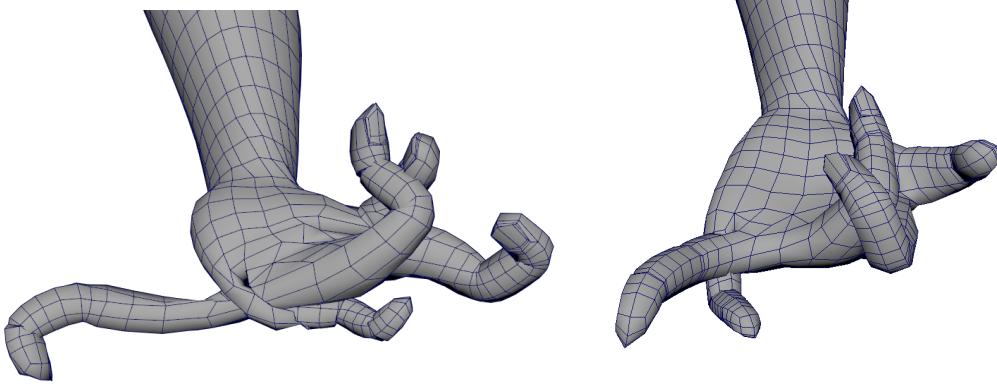


Figure 12. Comparaison d'une pose de main ayant des limites de rotation [-90°, 90°] dans chaque axe avec la même pose contrainte par des limites plus proches de l'amplitude du mouvement humain.

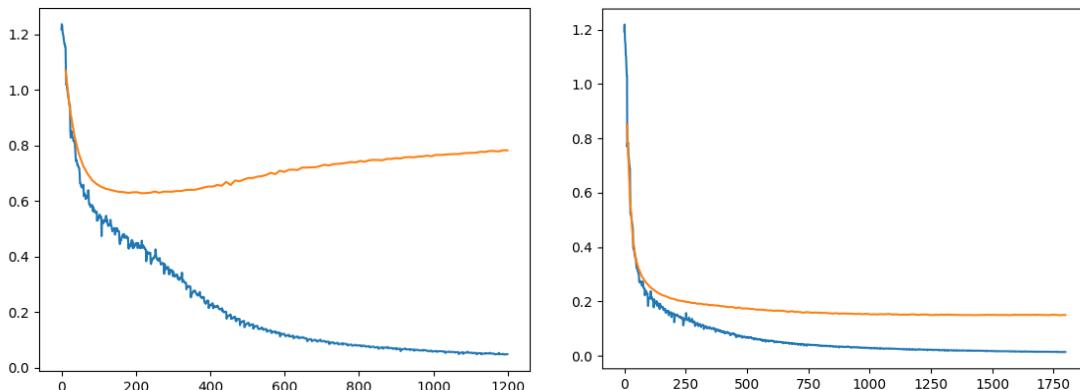


Figure 13. Les courbes d'apprentissage de la partition définissant le métacarpe de l'index avant et après la réduction de l'amplitude de mouvement dans le dataset d'apprentissage. On note que cette zone reste tout de même plus difficile à approximer que des articulations simples, ce qui se traduit par une convergence de la courbe de validation de aux alentours de 0.2.

Évolutions des réseaux

Dans le papier, les auteurs n'exposent qu'une topologie très simple et une unique fonction d'activation pour leurs réseaux. J'ai décidé de tester 3 variantes de ce réseau

afin d'étudier les effets de ces changements sur les résultats. Les variations consistent à augmenter la taille des couches (512 neurones au lieu de 128), à augmenter la profondeur du réseau (3 couches au lieu de 2) et à utiliser une autre fonction d'activation (“relu” au lieu de “tanh”). Voici les résultats de ces variations :

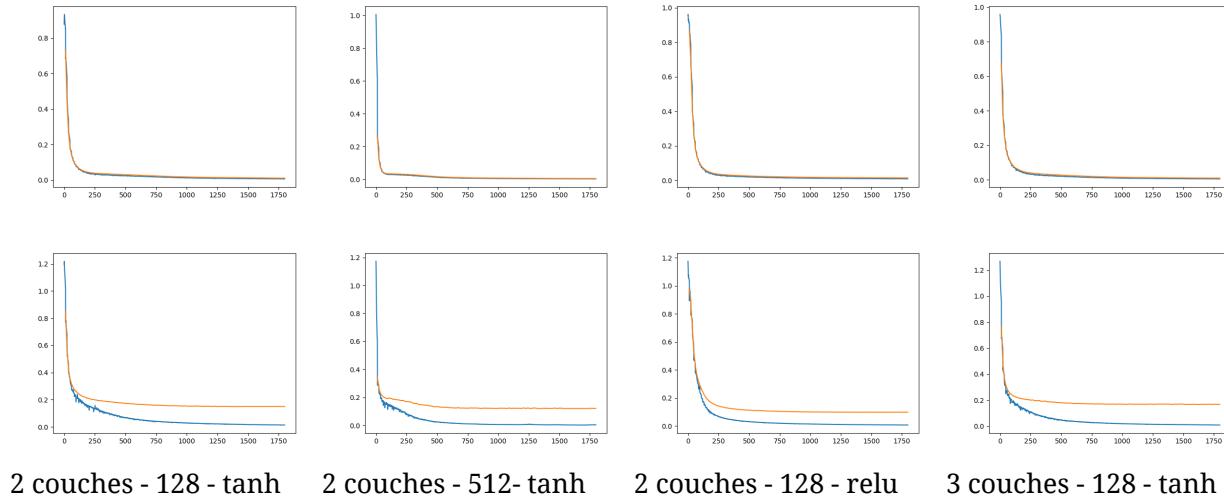


Figure 14. Comparatif des variations de réseau étudiées pour la partition de l'humérus (première ligne) et du métacarpe de l'index (ligne du bas).

Les courbes d'apprentissages ne varient pas de manière significative en appliquant ces variations. On note que l'apprentissage à base de relu permet tout de même d'obtenir une courbe d'erreur de validation plus proche de la courbe d'erreur sur le set d'apprentissage. Cependant, ce gain ne se traduit pas par un gain significatif dans l'aspect visuel des déformations. Il en va de même pour les autres variations du réseau, il est très complexe de noter une amélioration ou une régression dans les performances des réseaux. Il faudra donc se tourner vers d'autres solutions si on souhaite augmenter la qualité du système.

RÉSULTATS

Voici quelques images illustrant les résultats produits par ce projet, ainsi qu'une [vidéo](#) montrant l'effet du système sur une animation de test et un autre [vidéo](#) ayant le rendu heat-map exposé dans la figure X.



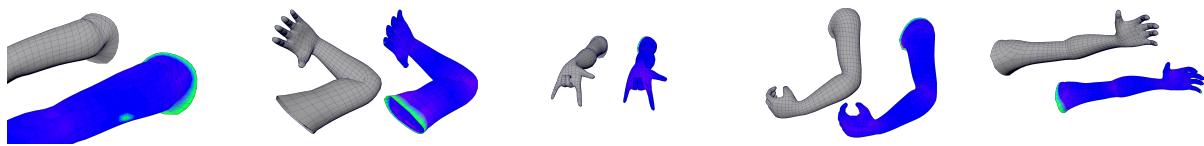


Figure 15. Comparaison des déformations de référence (le bras de gauche) et des déformations approximées. La ligne du bas contient des heat-maps représentant l'erreur absolue produites par l'approximation. Vert=aucune erreur, Bleu: erreur>1e-4 cm, Violet: erreur=0.5 cm, Rouge: erreur=1.0 cm. Bien que le résultat produit ne corresponde pas parfaitement aux déformations de références, notamment lors de poses extrêmes, on note que les erreurs ne dépassent presque jamais 0.3-0.5 cm.

On note également que le système arrive bien à extrapoler des résultats en dehors de la zone d'entraînement. La déformation produite est alors de moins bonne qualité et on note l'apparition de bruit pour des valeurs très différentes de l'amplitude d'apprentissage, cependant le résultat reste cohérent. C'est un point très positif car un animateur a souvent tendance à réaliser des poses extrêmes (notamment lors de mouvements rapides) en dehors de l'amplitude de mouvement habituelle. Si une pose non-prévue par le système provoquait une explosion du maillage, il serait alors nécessaire de ré-entraîner tout le réseau, ce qui causerait de nombreux ralentissements dans la chaîne de production.

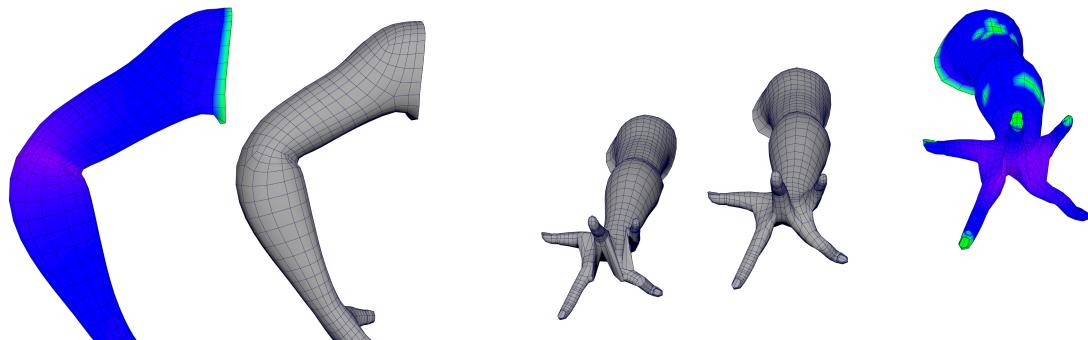


Figure 16. Exemples de déformations en dehors de l'amplitude de mouvement dans laquelle le réseau a été entraîné. Le coude n'est pas plié dans son axe habituel et subit une transformation 3x plus importante que son amplitude normale dans cet axe. Dans la deuxième image, les métacarpes subissent des transformations 2.5x plus importantes que leur amplitude normale. On note de plus grandes différences mais les réseaux de neurones produisent un résultat suffisant pour une étape d'animation.

Toutes les courbes d'apprentissage, ainsi que fichiers d'entraînement et les captures du viewport sont disponibles dans les dossiers Arm et Démo de l'archive en annexe et sur le [dépot github du projet](#).

RÉFÉRENCES

- [1.] "Fast and deep deformation approximations.", Bailey, S. W., Otte, D., Dilorenzo, P., & O'Brien, J. F. (2018)
- [2.] [Autodesk Maya \(2023\)](#)
- [3.] [Keras](#)
- [4.] "Delta mush: smoothing deformations while preserving detail.", Mancewicz, J., Derksen, M. L., Rijpkema, H., & Wilson (2014)