

école —
normale —
supérieure —
paris — saclay —

MASTER MVA

NUAGES DE POINTS ET MODÉLISATIONS 3D
RAPPORT

PointNetLK Revisited

Élèves : Colin COËRCHON & Chloé HABASQUE

Enseignants : François GOULETTE & Jean-Emmanuel DESCHAUD

Correspondant : Jean Pierre RICHA



19 mars 2025

Table des matières

1	Introduction	1
1.1	Contexte	1
1.2	Travaux en lien	1
2	De PointNet à PointNetLK revisité	1
2.1	Rappel sur PointNet	1
2.2	PointNetLK : la transition vers le recalage	2
2.2.1	Formulation mathématique du problème	2
2.2.2	Utilisation d'IC-LK	3
2.2.3	Limites et instabilités du Jacobien	4
2.3	Vers une version revisitée	4
2.3.1	Assemblage final	5
2.4	Voxélisation pour l'application aux scènes réelles	5
3	Implémentation et extension	6
3.1	Dataset	6
3.2	Résultats	6
3.3	Propositions d'améliorations	9
4	Conclusion	10
A	Algorithme LK	12
B	Preuve des lemmes	13

1 Introduction

1.1 Contexte

Ce rapport porte sur la problématique de l'alignement de nuages de point. Le but de l'alignement est d'estimer une transformation (rotation et translation) permettant de faire correspondre un nuage de points *source* à un nuage *cible*.

1.2 Travaux en lien

Ces dernières années, les **méthodes d'apprentissage profond** ont montré des performances impressionnantes dans le traitement de nuages de points. Cependant, elles souffrent souvent d'un **manque de généralisation**, notamment lorsqu'elles sont confrontées à des catégories d'objets différentes de l'apprentissage ou des scènes trop complexes.

Dans ces situations, les méthodes classiques non basées sur l'apprentissage, comme l'Iterative Closest Point (ICP), offrent généralement de meilleurs résultats grâce à leur robustesse intrinsèque.

Depuis 2016/2017, des méthodes comme PointNet [1] ont commencé à donner des résultats très intéressants. Ainsi, dans leur travail, Li, Pontes et Lucey (2020) [2] proposent une amélioration de PointNet. Ils se basent sur une première amélioration du modèle : PointNetLK (2019) [3], une version déjà plus robuste que PointNet, et en proposent une réévaluation en **intégrant un Jacobien analytique**. Cette innovation permet d'**améliorer significativement la capacité de généralisation** tout en conservant les avantages inhérents aux approches d'apprentissage. Le problème de cette technique est le calcul numérique du Jacobien qui bien que donnant de bons résultats, peut demander des ressources de calcul importantes. De plus, son calcul est très instable, les gradients peuvent être très bruité et biaisé, en fonction du pas t_p choisi à chaque itération et du jeu de données.

2 De PointNet à PointNetLK revisité

2.1 Rappel sur PointNet

Le modèle *PointNet*, introduit par [1] en 2017, est l'une des premières approches d'apprentissage profond conçues spécifiquement pour traiter des nuages de points dans $\mathbb{R}^{N \times 3}$. Le principe clé de PointNet repose sur deux idées :

- **Un encodage par réseau multi-couche (MLP)** appliqué point par point, avec des opérations de type convolution 1D pour extraire des *features* de chaque point individuellement.
- **Une fonction de pooling symétrique (souvent un max-pooling)** afin d'agréger les informations ponctuelles en un vecteur de caractéristiques globales invariante

à l'ordre de permutation des points.

En d'autres termes, PointNet modélise chaque point indépendamment, puis fusionne ces informations dans un descripteur $\phi(P)$ de dimension fixe K . Ce modèle a prouvé son efficacité pour des tâches de classification ou de segmentation de nuages de points, mais **il n'est initialement pas conçu pour le recalage**. En effet, l'agrégation globale a tendance à ignorer ou à perdre en partie la structure géométrique locale.

Expérience en TP : Au cours du dernier TP (TP6) sur le dataset *ModelNet40*, nous avons pu entraîner et tester un réseau PointNet pour de la classification, obtenant des performances situées entre 85 % et 90 % de précision. Nous étions ainsi déjà familier avec l'idée d'extraire des *features* globales robustes de nuages de points.

2.2 PointNetLK : la transition vers le recalage

Afin d'exploiter les capacités d'encodage global de PointNet pour le recalage de nuages de points, l'article [3] (sorti en 2019) propose une extension nommée *PointNetLK*. L'objectif est de trouver la **transformation rigide** (rotation + translation) qui aligne un nuage *source* P_S sur un nuage *cible* P_T , tout en profitant du descripteur global $P \mapsto \phi(P)$ fourni par PointNet.

Notre objectif est très clair, il faut minimiser une fonction de coût du type :

$$\|\phi(\text{nuage transformé}) - \phi(\text{nuage fixe})\|^2$$

Pour cela, l'article [3] utilise une des variantes de l'algorithme de Lucas-Kanade, la variante *inverse compositional* IC-LK, présentée et largement détaillée dans l'article [4].

2.2.1 Formulation mathématique du problème

D'un point de vue plus formel, on considère un vecteur $\xi \in \mathbb{R}^6$ décrivant la transformation rigide en 3D : 3 rotations $(\omega_x, \omega_y, \omega_z)$ et 3 translations : (t_x, t_y, t_z) . Les transformations s'expriment dans le groupe $SE(3)$, et on paramètre ξ dans son algèbre de Lie $\mathfrak{se}(3)$ grâce à l'exponentielle de matrice :

$$\mathcal{G}(\xi) = \exp \left(\sum_{p=1}^6 \xi_p T_p \right)$$

où chaque T_p est un générateur d'une rotation ou translation élémentaire.

Ainsi, le nuage P_S transformé devient $\mathcal{G}(\xi) \cdot P_S$. Le but est alors de minimiser l'erreur entre les *features* calculées par PointNet :

$$\boxed{\arg \min_{\xi} \|\phi(\mathcal{G}(\xi) \cdot P_S) - \phi(P_T)\|_2^2} \quad (1)$$

Cette équation est non-linéaire. PointNetLK optimise de manière itérative un paramètre de transformation incrémental $\Delta\xi$ de manière à aligner au mieux les représentations de caractéristiques des nuages de points apprises par PointNet.

2.2.2 Utilisation d'IC-LK

Pour nous aider à déterminer les bons paramètres $(\xi_p)_{p \in \llbracket 1,6 \rrbracket}$, on utilise une variante de l'algorithme LK (*Lucas-Kanade*). En effet, PointNetLK s'appuie sur un schéma d'optimisation inspiré de la méthode [5]. Cette méthode était initialement conçue pour du recalage d'images en 2D, mais elle a ensuite été adapté dans le cadre des nuages de points en 3D. Et dans ce travail, plus précisément, ils emploient la *variante inverse compositionnelle* (IC-LK) [4]. Le pseudo-code de l'algorithme général IC-LK se situe en annexe A.

- Rapidement, dans l'algorithme IC-LK, on cherche $\Delta\xi$ qui aligne le nuage *source* P_S et le nuage *cible* P_T en **modifiant temporairement** P_T . Cela peut paraître paradoxal, mais cela permet d'obtenir une Jacobienne constante, donc pré-calculable, ce qui accélère les itérations.
- Pour mettre à jour notre estimation $\xi^{(i)}$ (qui agit initialement sur P_S), on doit donc "inverser" l'incrément $\Delta\xi$ parce que celui-ci a été calculé dans le référentiel inverse.

Ainsi, la mise à jour du paramètre $\xi^{(i)}$ peut s'écrire :

$$\xi^{(i+1)} \leftarrow \xi^{(i)} \circ^{-1} \Delta\xi$$

Avec le symbole $\circ^{-1}$ qui désigne l'inversion compositionnelle. Cette définition de $\xi^{(i+1)}$ vient directement de l'implémentation de l'algorithme IC-LK. Et donc, notre problème donné dans l'équation (1) est alors transféré dans la recherche de $\Delta\xi$:

$$\arg \min_{\Delta\xi} \left\| \phi(P_S) - \phi(\mathcal{G}^{-1}(\xi^{(i)} \circ^{-1} \Delta\xi) \cdot P_T) \right\|_2^2 \quad (2)$$

Pour résoudre numériquement ce problème, l'algorithme IC-LK fait une approximation de Taylor afin d'estimer un incrément $\Delta\xi$. La preuve de ce résultat est donnée dans le lemme B.1. À l'itération i , on linéarise donc :

$$\boxed{\arg \min_{\Delta\xi} \left\| \phi(P_S) - \phi(\mathcal{G}^{-1}(\xi^{(i)}) \cdot P_T) - J\Delta\xi \right\|_2^2} \quad (3)$$

où J est la matrice Jacobienne par rapport aux 6 paramètres $\Delta\xi$. PointNetLK (version 2019) calcule justement cette matrice Jacobienne via **différences finies** :

$$J_p \approx \frac{\phi(\exp(-t_p T_p) \cdot P_T) - \phi(P_T)}{t_p}$$

avec un petit pas t_p . Cette estimation est correcte si t_p est bien choisi (suffisamment petit $\approx 10^{-2}$), mais peut s'avérer instable ou imprécis. Ce résultat est prouvé dans le lemme B.2.

2.2.3 Limites et instabilités du Jacobien

Le principal inconvénient de cette version *classique* de PointNetLK réside donc dans la sensibilité au choix du pas $\{t_p\}$: trop grand, on crée une erreur d'approximation significative ; trop petit, on risque des annulations numériques. De plus, recalculer cette matrice jacobienne à chaque itération ralentit les mises à jour et peut rendre la méthode peu robuste pour certaines configurations.

2.3 Vers une version revisitée

Pour contourner les limites décrites précédemment (instabilités numériques et sensibilité au choix du pas de perturbation), *PointNetLK revisited* [2] s'appuie sur une **matrice jacobienne analytique** de la fonction

$$\phi(\mathcal{G}^{-1}(\xi) \cdot P_T)$$

où ϕ désigne l'encodage PointNet, et $\mathcal{G}^{-1}(\xi)$ la transformation rigide (en $\mathfrak{se}(3)$) appliquée au nuage P_T . Par la *règle de la chaîne*, ce Jacobien se décompose en deux parties :

$$J = \underbrace{\frac{\partial \phi(\mathcal{G}^{-1}(\xi) \cdot P_T)}{\partial (\mathcal{G}^{-1}(\xi) \cdot P_T)^T}}_{\text{Feature gradient}} \cdot \underbrace{\frac{\partial (\mathcal{G}^{-1}(\xi) \cdot P_T)}{\partial \xi^T}}_{\text{Warp Jacobian}}$$

1) Feature Gradient

Le *gradient des features* obtenues grâce au réseau PointNet. On suppose, pour simplifier, que le réseau PointNet comporte L couches consécutives. Sur chaque couche l , la sortie $\mathbf{z}_l$ est donnée par :

$$\mathbf{z}_l = \text{ReLU}\left(\text{BN}_l(A_l \mathbf{z}_{l-1} + b_l)\right)$$

La dérivation $\frac{\partial \mathbf{z}_L}{\partial \mathbf{z}_0^T}$ s'obtient alors par produit successif des Jacobiennes de chaque couche. Enfin, $\text{Pool}(\mathbf{z}_L)$ consolide les informations en un vecteur global $\phi(P_T) \in \mathbb{R}^K$. Cela donne le *gradient des features* par rapport aux coordonnées de P_T .

2) Warp Jacobian

Le *Warp Jacobian* décrit la dérivée de la transformation rigide $\mathcal{G}^{-1}(\xi) \cdot P_T$ vis-à-vis des paramètres ξ . Pour chaque point, si $\mathbf{p}_i \in P_T$, on écrit $\mathbf{p}'_i(\xi) = \mathcal{G}^{-1}(\xi) \mathbf{p}_i$. Les dérivées

$\frac{\partial \mathbf{p}'_i(\xi)}{\partial \xi^T}$ se calculent analytiquement, car $\mathcal{G}^{-1}(\xi)$ possède des formules explicites pour la rotation et la translation.

2.3.1 Assemblage final

En combinant ces deux parties, on obtient :

$$J = \text{Pool} \left(\underbrace{\frac{\partial \mathbf{z}_l}{\partial \text{vec}(P_T)^T}}_{\text{Feature gradient}} \cdot \underbrace{\frac{\partial (\mathcal{G}^{-1}(\xi) \cdot P_T)}{\partial \xi^T}}_{\text{Warp Jacobian}} \right) \quad (4)$$

Comme on peut le constater dans l'équation (4), le calcul de la matrice jacobienne est totalement **analytique**. *PointNetLK revisited* évite donc l'approximation numérique de *PointNetLK* par la méthode des différences finies et bénéficie normalement d'une **plus grande stabilité** (indépendance au pas d'approximation) et d'une **rapidité accrue** (pas de réévaluation multiple de ϕ).

2.4 Voxélisation pour l'application aux scènes réelles

Dans le cadre d'une matrice Jacobienne déterminée numériquement ou analytiquement, l'algorithme *PointNetLK* peut parfois avoir du mal à traiter des scènes trop complexes. Pour répondre à cette problématique, l'article [3] évoque rapidement une solution de "voxélisation" du nuage de points, mais c'est surtout dans l'article principal de notre projet [2] que cette solution est développée.

La voxelisation consiste à **diviser l'espace 3D en voxels**, c'est-à-dire en petits volumes réguliers, ce qui permet de traiter de manière locale des régions complexes d'une scène. Cette approche est très utile lorsque le modèle est appliqué sur des scènes réelles, notamment pour le jeu de données *3DMatch* [6]. Plutôt que d'extraire une seule caractéristique globale à partir de l'ensemble du nuage de points, **le modèle partitionne la scène en plusieurs voxels** et extrait, pour chacun d'eux, un descripteur local en utilisant le réseau PointNet. Pour chaque voxel, un jacobien local est calculé dans le repère du voxel, permettant ainsi de quantifier comment une modification des paramètres de transformation affecte localement la géométrie. Ces jacobiens locaux J_{V_i} sont ensuite transformés et "rassemblés" pour estimer une transformation rigide globale J_g qui aligne l'ensemble des deux nuages de points.

$$J_g = [J_{V_1}, \dots, J_{V_M}] \begin{bmatrix} \frac{\partial \xi_{V_1}}{\partial \xi^T} \\ \vdots \\ \frac{\partial \xi_{V_M}}{\partial \xi^T} \end{bmatrix}$$

L'équation (3), qui correspond à l'optimisation pour minimiser la différence entre les nuages de points, peut alors être réécrite :

$$\arg \min_{\Delta \xi} \sum_{m=1}^M \left\| \phi(V_{S_m}) - \phi(\mathcal{G}^{-1}(\xi_i) \cdot V_{T_m}) - J_g \Delta \xi \right\|_2^2 \quad (5)$$

où $\phi(V_{S_m})$ et $\phi(V_{T_m})$ sont les caractéristiques des voxels de la source P_S et de la cible P_T .

3 Implémentation et extension

3.1 Dataset

Les deux datasets utilisés sont ModelNet40 [7] et 3DMatch [6]. Le jeu de données *ModelNet3D* est un ensemble de nuages de points 3D synthétiques conçue pour l'entraînement et l'évaluation des algorithmes de reconnaissance de formes 3D. La version simplifiée, *ModelNet40* regroupe 12 311 objets répartis en 40 catégories.

De plus, le jeu de données *3DMatch* se compose de scènes réelles complexes reconstruites à partir de capteurs RGB-D. Il fournit des paires de portions de scènes 3D avec des annotations de transformations rigides entre elles, permettant en particulier d'être utilisé pour l'évaluation des algorithmes de recalage de nuages de points 3D.

3.2 Résultats

Pour l'implémentation de l'algorithme *PointNetLK Revisited* [2], nous nous sommes grandement aidé de l'implémentation déjà proposé par les auteurs de l'article et partagé sur GitHub [8].

En premier lieu, nous avons tout d'abord entraîné le modèle de *PointNetLK Revisited* sur une partie du dataset *ModelNet40* (20 classes). Puis nous avons testé cet entraînement sur différents jeu de données de nuages de points, notamment les nuages de points plus complexes, représentant des scènes 3D du dataset *3DMatch*, tels que les scènes *chess*, *fire*, *heads*, *office*, *pumpkin* et *red-kitchen*, composé de plusieurs nuages de points. On observe ainsi une bonne généralisation aux nuages de points à des scènes 3D, grâce à la voxélisation.

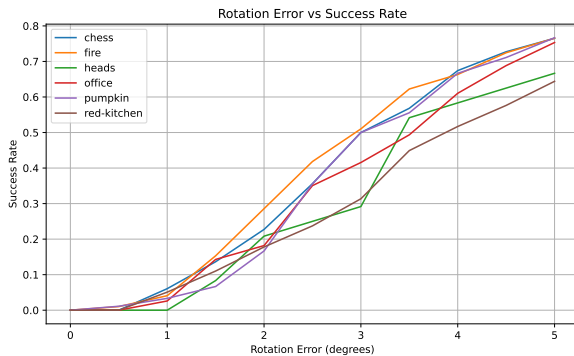


FIGURE 1 – Taux de succès en fonction de l'erreur de rotation

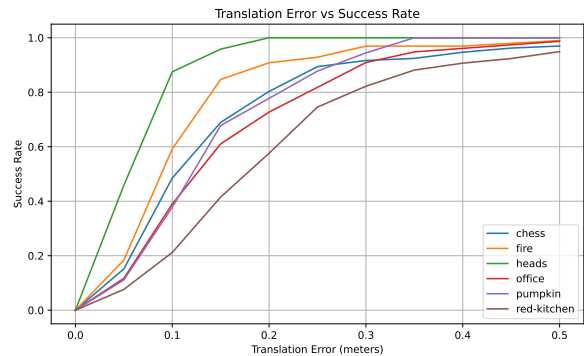


FIGURE 2 – Taux de succès en fonction de l'erreur de translation

Les graphiques 1 et 2 illustrent respectivement l'évolution du taux de succès en fonction de l'erreur de rotation (en degrés) et de l'erreur de translation (en mètres) pour différentes scènes du jeu de données **3DMatch**.

- **Pour la rotation :**

On observe que le taux de succès augmente de manière monotone pour la rotation lorsque les seuils d'erreur admissibles augmentent. Cependant, **il est compliqué d'atteindre les 100% de taux de succès** même en mettant le seuil d'erreur à 5° de rotation.

- **Pour la translation :**

Certaines scènes telles que *heads* et *fire*, présentent **des bonnes performances** avec des taux de succès atteignant plus de 80% pour des faibles erreurs de rotations (inférieures à 0.15). À l'inverse, les recalages de scènes plus complexes comme *red kitchen* ou *office* ont plus d'erreur de translation avec une croissance plus lente du taux de succès, ce qui reflète une difficulté plus importante du recalage de ces scènes.

		Rotation			Translation		
Classe	#Tests	MSE	RMSE	MAE	MSE	RMSE	MAE
Chess	132	98.41	9.92	4.03	0.087	0.296	0.135
Heads	24	20.72	4.55	3.51	0.005	0.069	0.053
Pumpkin	90	16.28	4.03	3.08	0.025	0.157	0.114
Office	77	23.10	4.81	3.44	0.036	0.190	0.135
Fire	98	98.66	9.93	3.67	0.081	0.285	0.108
Red Kitchen	118	65.02	8.06	4.11	0.191	0.437	0.197

TABLE 1 – Résultats moyens de recalage pour chaque scène (rotation et translation)

En plus du taux de succès, nous avons calculé les différentes erreurs globales pour chacune des classes. Le Tableau 1 synthétise ainsi les performances moyennes du modèle en termes d'erreurs globales de recalage, séparées en rotation et en translation. On remarque que la scène *pumpkin* obtient les meilleurs résultats avec les plus faibles erreurs de rotation (MSE = 16.28, MAE = 3.08) et de translation (MSE = 0.025, MAE = 0.114), traduisant une grande stabilité du recalage sur cette séquence. À l'inverse, les scènes *chess* et *fire* présentent des erreurs de rotation très élevées (MSE $\approx$ 98), suggérant une difficulté à estimer correctement les orientations.

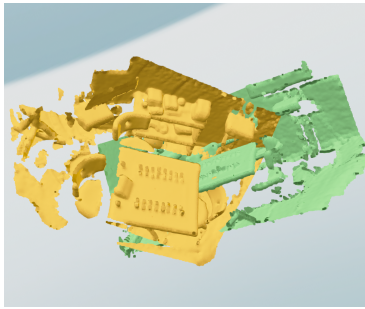


FIGURE 3 – Chess initial

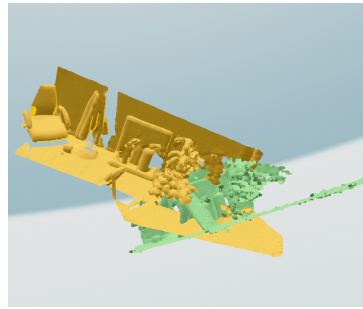


FIGURE 4 – Fire initial

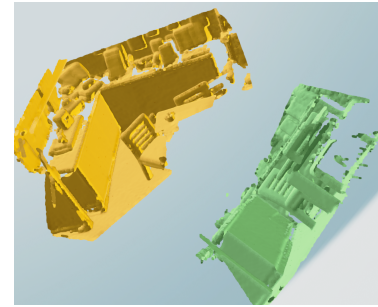


FIGURE 5 – Red-Kitchen
initial

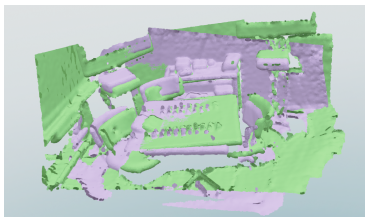


FIGURE 6 – Chess

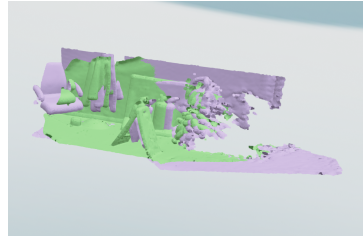


FIGURE 7 – Fire

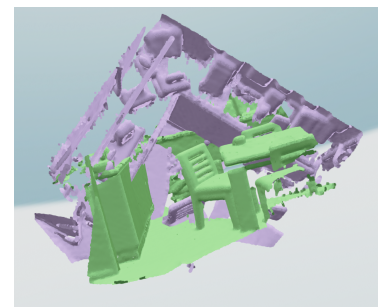


FIGURE 8 – Red-Kitchen

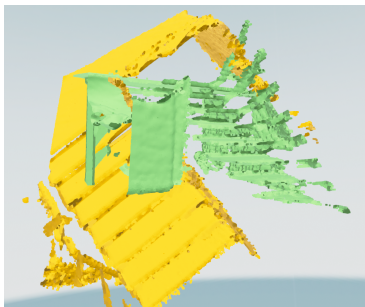


FIGURE 9 – Stairs initial

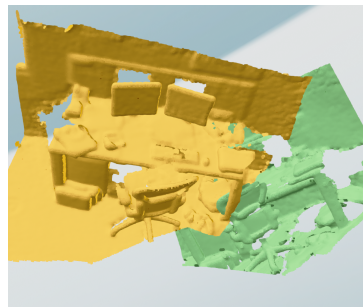


FIGURE 10 – Office initial

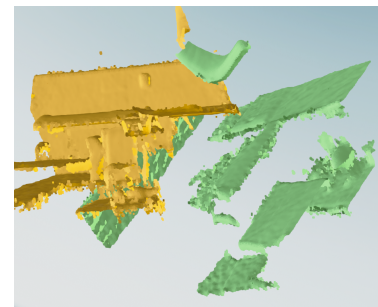


FIGURE 11 – Pumpkin
initial

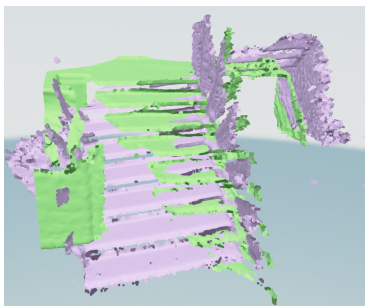


FIGURE 12 – Stairs

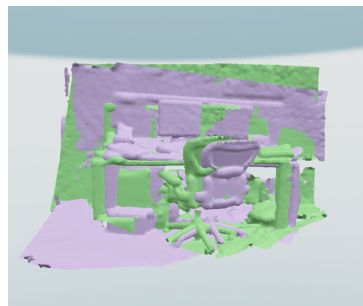


FIGURE 13 – Office

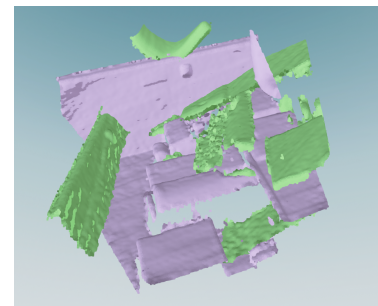


FIGURE 14 – Pumpkin

Nous avons ensuite visualisé certains tests de recalages en prenant différentes scènes 3D. Dans le cas des exemples *chess*, *fire*, *stairs* et *office*, **le recalage est très correct** au bout de peu d'itérations (entre 5 et 10). En effet, la réussite du recalage se remarque bien sur la figure 12 pour les escaliers, ou sur la figure 6 pour la salle avec un échiquier.

Par contre, comme le montre les figures 8 et 14 pour les scènes choisies pour *red-kitchen* et *pumpkin*, l'erreur de rotation et de translation est encore très importante : le recalage n'a clairement pas fonctionné. Cela peut être dû à l'éloignement initial très important du nuage de point cible, notamment en terme de **rotation**. Indéniablement, lorsque les deux nuages de points ont de grandes différences de rotation, **le recalage semble souvent très complexe** pour l'algorithme.

3.3 Propositions d'améliorations

Pour améliorer les résultats présentés dans ce papier, plusieurs améliorations peuvent être explorées :

- **Amélioration de Pointnet** : Utiliser des architectures plus récentes ou plus complexes que PointNet (par exemple, PointNet++, DGCNN ou des réseaux basés sur des mécanismes d'attention) afin de mieux capturer à la fois les structures locales et globales du nuage de points. Malheureusement, cela complexifierait encore davantage notre algorithme qui met déjà beaucoup de temps à s'exécuter.
- **Augmentation de données** : En plus de réaliser des rotations et translations pour recalés 2 objets entre eux. Il est également possible d'augmenter le nombre de nos données à l'aide de rotation, translation et bruit. Cela va rendre l'entraînement plus robuste et aider notre modèle à améliorer la généralisation aux scénarios réels.
- **Optimisation de matrice jacobienne analytique** :
 - ▶ Développer une formulation plus flexible pour le warp Jacobian, par exemple en intégrant des modules d'attention ou des mécanismes adaptatifs pour pondérer l'influence de chaque point ou voxel.
 - ▶ Expérimenter avec des techniques d'apprentissage auto-supervisé ou contrastif pour renforcer la stabilité des gradients extraits.
- **Optimisation computationnelle** : Une dernière idée pourrait être d'optimiser l'implémentation du calcul du jacobien analytique afin de réduire la charge computationnelle, ce qui pourrait permettre d'étendre l'approche à des nuages de points plus volumineux ou d'améliorer la vitesse en temps réel.

Bien évidemment, il existe maintenant dans la littérature de nombreuses variantes utilisant du deep learning pour faire du recalage de nuages de points. Nous avons bien essayé d'appliquer d'autres méthodes à nos jeu de données tel que *PointNetLK* ou *DeepGlobal-Registration* pour comparer nos résultats, mais, au vu de notre faible capacité de calcul, nous n'avons pas obtenu assez de résultats pertinents pour ce rapport.

4 Conclusion

Difficultés rencontrées

Même si la version revisitée de PointNetLK est très prometteuse (notamment grâce à son Jacobien analytique et à la “voxélisation” pour traiter de gros nuages de points), sa mise en œuvre pratique nous a causé quelques difficultés. Le calcul explicite de la matrice jacobien nécessite beaucoup de ressources et nous avons souvent rencontré des problèmes de manque de mémoire, même pour simplement tester le modèle pré-entraîné (exemple d’erreur : « `DefaultCPUAllocator: not enough memory: tried to allocate 84180729856 bytes...` »).

Cependant, nous pensons avoir bien compris les aspects mathématiques (le calcul de la matrice jacobienne, la formulation IC-LK) et à étudier les modules de code gérant la jacobienne analytique (dans `utils.py` et `data_utils.py`). Et du point de vue expérimentation, nous aurions aimé disposer de machines plus puissantes pour tester davantage de configurations et peaufiner l’entraînement.

Perspectives et bilan

Malgré les contraintes matérielles rencontrées, notre exploration de *PointNetLK Revisited* confirme l’intérêt de cette méthode. L’intégration d’une matrice jacobienne analytique renforce considérablement **la stabilité et la précision du recalage**, tandis que la stratégie de voxélisation permet de mieux gérer des scènes complexes de grande taille.

Cette approche hybride, mêlant l’optimisation incrémentale inspirée de Lucas & Kanade et les représentations apprises via PointNet, offre un bon compromis entre rigueur géométrique et expressivité des descripteurs. Elle constitue ainsi une base solide pour des applications variées de recalage 3D, **avec un fort potentiel de généralisation**.

Références

- [1] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet : Deep learning on point sets for 3d classification and segmentation, 2017.
- [2] Xueqian Li, Jhony Kaesemodel Pontes, and Simon Lucey. Pointnetlk revisited, 2021.
- [3] Yasuhiro Aoki, Hunter Goforth, Rangaprasad Arun Srivatsan, and Simon Lucey. Pointnetlk : Robust & efficient point cloud registration using pointnet, 2019.
- [4] Simon Baker and Iain Matthews. Lucas-kanade 20 years on : A unifying framework. *International journal of computer vision*, 56 :221–255, 2004.
- [5] Bruce D Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *IJCAI’81 : 7th international joint conference on Artificial intelligence*, volume 2, pages 674–679, 1981.
- [6] Christopher Choy. Deep global registration - 3dmatch download script. https://github.com/chrischoy/DeepGlobalRegistration/blob/master/scripts/download_3dmatch.sh, 2019.
- [7] Princeton ModelNet Dataset. Modelnet : A large-scale 3d cad model dataset. <https://modelnet.cs.princeton.edu/>, 2015. Accessed : Mar. 23, 2025.
- [8] Lilac Lee. Pointnetlk revisited. https://github.com/Lilac-Lee/PointNetLK_Revisited, 2020.

Annexes

A Algorithme LK

Nous réécrivons ici le pseudo-code de l'algorithme *Inverse Compositional* (IC-LK) tel que présenté dans [4].

Notations :

- $T(\mathbf{x})$: image *template*, définie sur les coordonnées $\mathbf{x}$
- $I(\cdot)$: image *input* à aligner avec T
- $\mathbf{W}(\mathbf{x}; \mathbf{p})$: transformation paramétrée (warp) agissant sur les coordonnées $\mathbf{x}$, avec paramètres $\mathbf{p}$
- $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$: Jacobienne du warp par rapport aux paramètres $\mathbf{p}$
- $\Delta \mathbf{p}$: incrément à appliquer à $\mathbf{p}$ à chaque itération
- ε : critère d'arrêt sur la norme de $\Delta \mathbf{p}$

Algorithm 1 Inverse Compositional Algorithm (IC-LK)

1: **Pré-calcul :**

- 2: (3) Calculer le gradient ∇T du modèle $T(\mathbf{x})$
- 3: (4) Calculer la Jacobienne $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ en $\mathbf{x}, \mathbf{p} = 0$
- 4: (5) Calculer les directions de descente maximale $\nabla T \cdot \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$
- 5: (6) Calculer la matrice Hessienne $\mathbf{H}$ selon l'équation (38)

6: **Itérer :**

- 7: (1) Appliquer le *warp* $\mathbf{W}(\mathbf{x}; \mathbf{p})$ à I pour obtenir $I(\mathbf{W}(\mathbf{x}; \mathbf{p}))$
- 8: (2) Calculer l'image d'erreur $I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - T(\mathbf{x})$
- 9: (7) Calculer :

$$\sum_{\mathbf{x}} \left[\nabla T \cdot \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^\top [I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - T(\mathbf{x})]$$

- 10: (8) Calculer $\Delta \mathbf{p}$ à l'aide de l'équation (37)
- 11: (9) Mettre à jour le *warp* :

$$\mathbf{W}(\mathbf{x}; \mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x}; \mathbf{p}) \circ \mathbf{W}(\mathbf{x}; \Delta \mathbf{p})^{-1}$$

- 12: Répéter jusqu'à convergence : $\|\Delta \mathbf{p}\| \leq \varepsilon$
-

B Preuve des lemmes

Lemme B.1 : Développement de Taylor à l'ordre 1

Soit $\phi : \mathbb{R}^{N \times 3} \rightarrow \mathbb{R}^K$ la fonction d'encodage (par exemple un réseau de type Point-Net) appliquée à un nuage de points. On cherche à linéariser l'expression de l'erreur quadratique entre les *features* de la source et du modèle transformé. Plus précisément, à l'itération i , on définit :

$$\Delta\xi \mapsto \phi\left(\mathcal{G}^{-1}(\xi^{(i)} \circ^{-1} \Delta\xi) \cdot P_T\right)$$

Le développement de Taylor à l'ordre 1 autour de $\Delta\xi = 0$ fournit :

$$\phi(P_S) - \phi\left(\mathcal{G}^{-1}(\xi^{(i)} \circ^{-1} \Delta\xi) \cdot P_T\right) = \left[\phi(P_S) - \phi\left(\mathcal{G}^{-1}(\xi^{(i)}) \cdot P_T\right)\right] - J \Delta\xi + o(\|\Delta\xi\|)$$

où J est la matrice Jacobienne évaluée en $\Delta\xi = 0$.

Démonstration.

Notons

$$f(\Delta\xi) = \phi\left(\mathcal{G}^{-1}(\xi^{(i)} \circ^{-1} \Delta\xi) \cdot P_T\right)$$

On souhaite approximer $f(\Delta\xi)$ pour $\Delta\xi$ petit, autour de $\Delta\xi = 0$. Par la formule de Taylor au premier ordre, on a :

$$f(\Delta\xi) = f(0) + \underbrace{\frac{\partial f}{\partial \Delta\xi} \Big|_{\Delta\xi=0}}_J \Delta\xi + o(\|\Delta\xi\|) = \phi\left(\mathcal{G}^{-1}(\xi^{(i)}) \cdot P_T\right) + J \Delta\xi + o(\|\Delta\xi\|)$$

On en déduit l'expression de la différence $\phi(P_S) - f(\Delta\xi)$:

$$\phi(P_S) - f(\Delta\xi) = \left[\phi(P_S) - \phi\left(\mathcal{G}^{-1}(\xi^{(i)}) \cdot P_T\right)\right] - J \Delta\xi + o(\|\Delta\xi\|)$$

Ce qui prouve le lemme. □

Lemme B.2 : Approximation de J par la méthode des différences finies

Soit

$$\mathcal{G}(\xi) = \exp\left(\sum_{p=1}^6 \xi_p T_p\right)$$

Alors la matrice jacobienne

$$J = \frac{\partial \phi(\mathcal{G}^{-1}(\xi) \cdot P_T)}{\partial \xi^T} \in \mathbb{R}^{K \times 6}$$

peut être approximée par :

$$\forall p \in \llbracket 1, 6 \rrbracket, \quad J_p \approx \frac{\phi(\exp(-t_p T_p) \cdot P_T) - \phi(P_T)}{t_p}$$

Démonstration.

Premièrement, connaissant la définition de $\mathcal{G}$, on en déduit facilement la définition de $\mathcal{G}^{-1}$:

$$\mathcal{G}^{-1}(\xi) = \exp\left(-\sum_{p=1}^6 \xi_p T_p\right)$$

Considérons alors un chemin $\xi(t) \in \mathbb{R}^6$ qui fait varier ξ_p et fige toutes les autres composantes à zéro. Formellement, pour chacune des 6 composantes, on a :

$$\xi(t) = (0, \dots, 0, \underbrace{t}_{p\text{-ème coord.}}, 0, \dots, 0)$$

Alors

$$\mathcal{G}^{-1}(\xi(t)) = \exp\left(-\sum_{q=1}^6 \xi_q(t) T_q\right) = \exp(-t T_p)$$

Partant de ce chemin $\xi(t)$, on note :

$$\gamma_p(t) = \phi\left(\mathcal{G}^{-1}(\xi(t)) \cdot P_T\right) = \phi(\exp(-t T_p) \cdot P_T)$$

Et donc, par la règle de la chaîne, on a

$$\left. \frac{d}{dt} \gamma_p(t) \right|_{t=0} = \left. \frac{\partial}{\partial \xi_p} \phi\left(\mathcal{G}^{-1}(\xi) \cdot P_T\right) \right|_{\xi=0}$$

En effet, quand on fait varier t autour de 0, on revient à l'idée de dériver ξ_p autour de $\xi = 0$, ce qui est exactement ce que l'on cherche pour approximer J_p . Ainsi, pour calculer $\left. \frac{d}{dt} \gamma_p(t) \right|_{t=0}$ numériquement, on utilise la méthode des différences finies en posant un petit



incrément t_p , on a alors :

$$J_p \approx \left. \frac{d}{dt} \gamma_p(t) \right|_{t=0} \approx \frac{\gamma_p(t_p) - \gamma_p(0)}{t_p} = \frac{\phi(\exp(-t_p T_p) \cdot P_T) - \phi(P_T)}{t_p}$$

Assemblage final

En superposant ces vecteurs-colonnes $\{J_p\}_{p=1}^6$, on obtient la matrice Jacobienne

$$J = [J_1, \dots, J_6] \in \mathbb{R}^{K \times 6}$$

ce qui conclut la preuve.

□