Simplification rapide de nuages de points avec préservation des caractéristiques vives

H. Benhabiles O. Aubreton

Laboratoire Le2i UMR 6303 CNRS

Université de Bourgogne

{halim.benhabiles, olivier.aubreton}@u-bourgogne.fr

Résumé

Cet article présente une méthode de simplification de nuages de points rapide. La méthode permet de préserver les points appartenant aux arêtes vives du nuage considéré en se basant sur la combinaison de deux approches de simplification, à savoir une approche de clustering *et une approche* coarse-to-fine*. Ayant un nuage en entrée, la méthode de simplification commence par créer un nuage grossier en utilisant un algorithme de clustering. Ensuite, un poids est affecté à chaque point du nuage grossier résultant permettant de quantifier son importance, et de le classifier en un point vif (ie. appartenant à une arête vive) ou en un point simple. Enfin, les deux types de points sont utilisés pour raffiner le nuage grossier et créer ainsi un nouveau nuage simplifié caractérisé par une densité de points qui est respectivement importantes dans les régions saillantes, et faible dans les régions planes.*

Les expérimentations montrent que notre algorithme de simplification est beaucoup plus rapide que le dernier algorithme proposé par Song et Feng [1] qui vise à préserver les points vifs, et donne des résultats de même qualité.

Mots clefs

Nuage de points, simplification, caractéristiques vives.

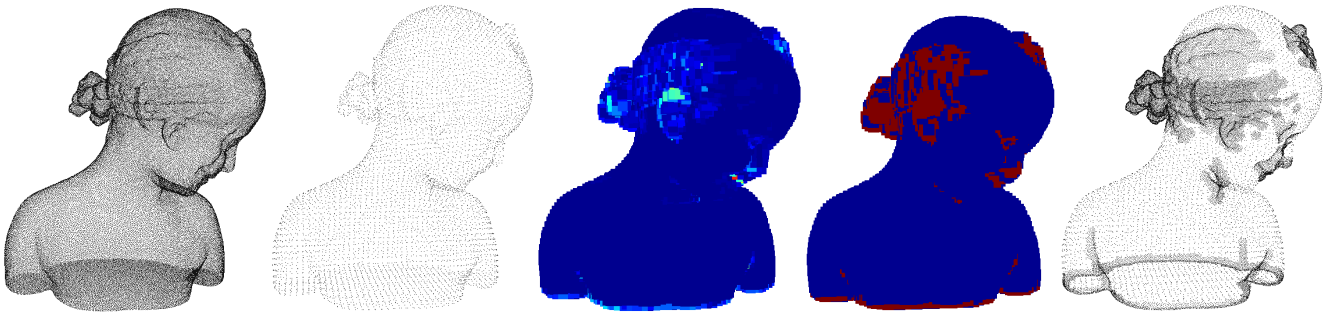
1 Introduction

Les systèmes de numérisation 3D ont connu un fort développement depuis une dizaine d'années. Il est ainsi possible d'obtenir en quelques minutes des nuages contenant des millions de points. Cependant les scanners 3D restent incapables de déterminer la densité de points optimale permettant de représenter fidèlement une surface. Cela conduite à une redondance importante de données, qui doit être supprimée afin de limiter les ressources de calcul nécessaires à l'analyse et la représentation de la forme. Ainsi, dans une chaine 3D typique, le nuage de points est transformé en maillage de triangles à l'aide d'algorithmes de reconstruction de surface [2] ; par la suite le maillage résultant subit une simplification [3, 4] afin de réduire sa taille et rendre possible les traitements ultérieurs. Ces deux dernières étapes (reconstruction de surface et simplification) sont en réalité très couteuses en mémoire et en temps de calcul. Par conséquent, simplifier en premier lieu le nuage de points permettra d'éliminer la redondance de données, d'accélérer significativement le processus de reconstruction, et d'éviter la simplification de maillages.

On trouve dans la littérature un certain nombre de travaux abordant le problème de simplification de nuage de points. Ils peuvent être classifiés en trois catégories : les méthodes de *clustering*, les méthodes *coarse-to-fine*, et les méthodes itératives.

La première catégorie (méthodes de clustering) couvre les algorithmes qui visent à subdiviser le nuage de points en entrée en plusieurs patchs surfacique (des sous ensembles de points), basés sur un certain critère, puis remplacer chaque patch par son point représentatif (le centroid par exemple). Dans ce contexte, Pauly *et al.* [5] ont proposé deux stratégies différentes, inspirées des méthodes de simplification de maillages [6, 7], pour la construction des patchs. La première, incrémentale, utilise un algorithme de croissance de régions. La seconde est hiérarchique, et se base sur un partitionnement binaire de l'espace. Le nombre, ainsi que la taille des patchs, sont contrôlés à l'aide de la courbure. Comme l'expliquent les auteurs, les méthodes de clustering sont rapides et efficaces en matière de ressource mémoires, mais conduisent à des nuages de points présentant une erreur quadratique importante.

Les approches de type coarse-to-fine extraient aléatoirement un sous ensemble de points du nuage original puis font appel à un diagramme de Voronoi 3D pour définir implicitement une fonction de distance. Cette dernière fonction est ensuite utilisée pour raffiner le nuage jusqu'a ce que l'erreur tolérée par l'utilisateur soit atteinte. Moenning et Dodgson [8] ont proposé un algorithme basé sur ce principe. Pour raffiner le sous ensemble de points les auteurs ajoutent de manière répétitive les points qui se trouvent à une distance définie par une fonction géodésique implicite basée sur le diagramme de Voronoi calculé pour le nuage original. Cette approche offre la possibilité de contrôler la densité du nuage de telle sorte



(a) (b) (c) (d) (e)

Figure 1 – *Vue globale de notre méthode de simplification hybride : nuage original (a), nuage grossier (b), estimation de la courbure (c), classification des points (d), nuage simplifié (e).*

que les régions planes soient caractérisées par une densité de points faible, et une densité plus importante dans les régions à forte courbure. Cependant, la distribution des points dans le nuage final reste aléatoire car elle dépend fortement de la distribution du sous ensemble de points initial qui est extrait aléatoirement.

Les méthodes itératives correspondent aux algorithmes dont l'objectif est de réduire de manière itérative le nombre de points du nuage en entrée en utilisant un opérateur de décimation comme cela a été proposé dans Song et Feng [1]. A cette fin, à chaque point du nuage d'entrée est affecté un poids quantifiant son importance parmi ses voisins directs. Le point ayant le poids le plus faible est supprimé, et la liste des voisins des points restants ainsi que celle de leurs poids respectifs sont mis à jour. Cette dernière étape est répétée jusqu'à ce que le nombre de points fixé par l'utilisateur soit atteint. Dans l'algorithme proposé par Song et Feng [1] l'importance d'un point donné est évaluée en vérifiant si ses voisins directs peuvent le représenter. Dans le cas positif, le point considéré est supprimé. Contrairement aux algorithmes issus des catégories précédentes, les algorithmes itératifs ont la propriété de préserver les zones saillantes présentes dans le nuage. Cependant, leurs inconvénient principal est qu'ils sont très couteux en mémoire et en temps de calcul, particulièrement pour les nuages massifs. Comme l'expliquent Song et Feng [1] ces algorithmes requièrent la maintenance d'une structure de données lourde, tel qu'un diagramme de Voronoi, afin de mettre à jour les poids des points et leurs voisins à chaque itération.

Dans cet article, nous proposons une technique de simplification hybride qui combine deux approches de simplification dont l'une est basée sur les méthodes de clustering et l'autre sur les méthodes coarse-to-fine pour identifier les régions contenant les points appartenant aux arêtes vives de manière très rapide. D'une part, le clustering permet d'obtenir un nuage grossier uniforme préservant la forme général de l'objet réel, et d'accélérer le temps de calcul dans les étapes ultérieurs de l'algorithme. D'autre part, la stratégie coarse-to-fine permet de contrôler localement la densité du nuage. La méthode consiste à créer tout d'abord un nuage grossier en utilisant un algorithme de clustering. Ensuite, à chaque point du nuage grossier résultant est affecté un poids quantifiant son importance parmi ses voisins. Ce poids correspond à la courbure (ie. la variation locale de la surface). Puis, tout les points sont classifiés selon leur importances en deux classes à savoir des points simples ou bien des points vifs. Enfin, les deux types de points sont utilisés pour raffiner le nuage grossier, et créer ainsi un nouveau nuage simplifié. Il est à noter que notre méthode est générique puisque c'est possible d'utiliser d'autres critères géométriques plus robustes que la courbure pour évaluer l'importance des points, et détecter plus efficacement ceux qui sont vifs.

Dans la partie suivante nous détaillons notre approche. Le calcul de la courbure sera décris dans la partie 3. Les résultats préliminaires seront présentés et discutés dans la partie 4, avant de conclure et d'expliquer nos perspectives.

2 Notre approche de simplification

Ayant un nuage de point en entrée (figure 1(a)), notre méthode commence par créer un nuage grossier en utilisant un algorithme de clustering basé sur une approche de grille voxélisée (figure 1(b)). A cette fin, nous subdivisons la boite englobante du nuage original en une grille de voxels 3D. Ensuite, tout les points appartenant au même voxel sont représentés par leur centroid. Notre choix d'utiliser une approche de clustering volumétrique est justifié par les raisons suivantes : l'approche permet de garder une distribution uniforme des points du nuage résultant. Cette dernière propriété est très importante puisque initialement nous ne possédons aucune information sur l'importance des point résultants, donc le nuage grossier doit impérativement couvrir toute la surface du nuage original. De plus, l'approche permet de contrôler facilement la densité global du nuage grossier en fixant tout simplement la taille des

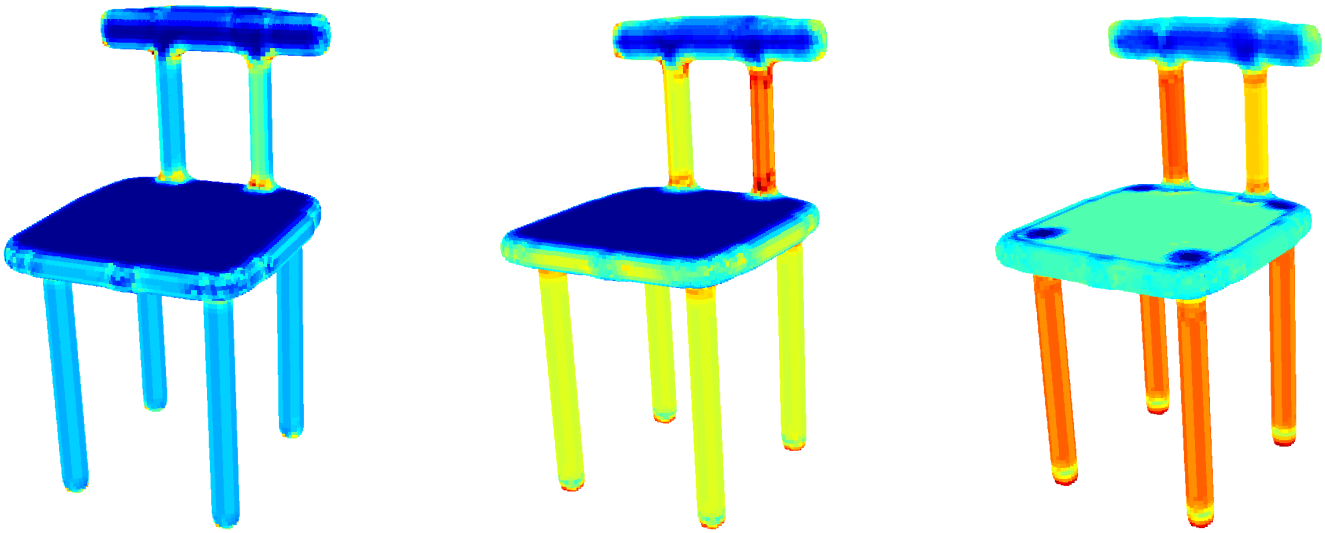


Figure 2 – *De gauche à droite, courbure (ou bien variation locale de la surface) calculée avec un voisinage de taille 20, 50, et 100.*

voxels. Par ailleurs, l'approche est très rapide même pour les nuages massifs.
La courbure est ensuite calculée pour chaque point du nuage grossier dans un voisinage donné en se basant sur le nuage original ; elle fournit une information sur la variation locale de la surface (figure 1(c)) et permet de classifier les points considérés en deux classes, à savoir des points vifs (régions rouges dans la figure 1(d)), et des points simples (régions bleus dans la figure 1(d)) selon un seuil fixé par l'utilisateur suivant la complexité du modèle. Pour créer le nuage simplifié final, les deux types de points (vifs et simples) sont utilisés pour raffiner le nuage grossier en sélectionnant pour chacun d'entre eux un ensemble de point à partir du nuage original (figure 1(e)). Ces ensembles de points correspondent aux voisins directs.

3 Calcul de la courbure

Afin de calculer la courbure de chaque point du nuage grossier, nous utilisons la méthode proposée par Pauly *et al.* [5]. Dans cette dernière méthode, la courbure est vue comme une variation locale de la surface correspondant à la distance des points du plan tangent. Les auteurs ont montré que l'analyse des vecteurs propres et des valeurs propres de la matrice de covariance dans un voisinage local d'un point donné peut être utilisée pour calculer certaines propriétés locales de la surface y compris la courbure.

Plus précisément, ayant un point $p \in \mathbb{R}^3$ ainsi que ses k-voisins les plus proches N_p, la matrice de covariance $C \in \mathbb{R}^{3\times3}$ de p est définie comme suit :

$$C = \begin{bmatrix} p_1 - \bar{p} \\ \dots \\ p_k - \bar{p} \end{bmatrix}^T \cdot \begin{bmatrix} p_1 - \bar{p} \\ \dots \\ p_k - \bar{p} \end{bmatrix}, p_{i=1,k} \in N_p$$

où $\bar{p}$ est le centroid de l'ensemble des voisins N_p, et il est définie par :

$$\bar{p} = \frac{1}{k} \cdot \sum_{i=1}^{k} p_i$$

Il est possible d'estimer la courbure σ d'un point p en analysant le problème de vecteurs propres suivant :

$$C.\vec{v_j} = \lambda_j.\vec{v_j}, j \in \{0, 1, 2\}$$

Par ce que C est symétrique et positive semi-définie (puisque c'est une matrice de covariance), toute les valeurs propres λ_j sont des valeurs réelles, et tout les vecteurs propres $\vec{v_j}$ forment un cadre orthogonal correspondant au composantes principales de l'ensemble des voisins N_p [9].

Si on suppose que $\lambda_0 \leq \lambda_1 \leq \lambda_2$, alors la variation de la surface (ou bien courbure) $\sigma_k(p)$ au point p dans un voisinage de taille k est définie comme suit :

$$\sigma_k(p) = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2}$$

Plus précisément, le rapport entre la valeur propre minimale et la somme de toute les valeurs propres permet d'estimer la déviation des points N_p du plan tangent à la surface. Ce rapport est invariant aux changements scalaires, et les valeurs obtenues sont comprises entre 0 et 1. Une valeur de 0 indique que les points N_p se trouvent sur le plan, et une valeur de 1 indique une déviation maximale du plan. Comme annoncé par Pauly *et al.* [5], la variation de la surface est considérée comme une courbure, cependant, son estimation dépend fortement du choix de la taille du voisinage. La figure 2 illustre un exemple de calcul de courbure pour un modèle *chaise*

(représenté par un nuage de points) en fixant différentes tailles du voisinage local. La figure montre clairement que l'estimation de la courbure est directement influencée par la taille du voisinage. Par conséquent, la méthode utilisée pour déterminer les voisins d'un point donné joue un rôle important dans le calcul de courbure.

Pour déterminer les k-voisins les plus proches des points du nuage grossier, nous utilisons la méthode de recherche rapide kd-tree [10]. Afin d'apporter une certaine robustesse au bruit et à l'échantillionage lors du calcul de la courbure, nous considérons le nuage de points original comme la surface de voisinage sous-jacente utilisée pour obtenir l'ensemble des voisins les plus proches de chaque point du nuage grossier. Comme illustré dans la figure 3, si $Q = \{q_1, q_2\}$ est l'ensemble des points du nuage grossier, et P est la surface de recherche (les points bleus représentant le nuage original) pour Q, alors les voisins de q_1 et q_2 sont déterminés à partir de P.

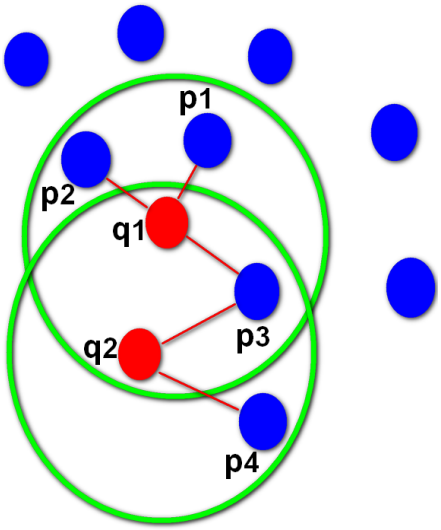


Figure 3 – *Exemple de calcul de voisinage pour les points du nuage grossier (en rouge) basé sur le nuage original (points bleus).*

4 Résultats et discussion

A partir de la courbure calculée précédemment, les points du nuage grossier sont répartis en deux classes (points vifs et points simples) selon un seuil fixé par l'utilisateur sur la valeur de courbure (entre 0.2 et 0.6 selon le modèle). Pour créer le nuage final simplifié, les deux types de points sont utilisés pour sélectionner un ensemble de points du nuage original correspondant aux voisins les plus proches. Selon nos expérimentations à travers différents modèles, nous avons constaté que sélectionner respectivement 20 voisins les plus proches pour chaque point vif est suffisant pour couvrir les régions vives dans le nuage simplifié, et 1 seul voisin le plus proche pour chaque point simple

Nuage	**Vertex**	**Simplification (%)**	**Temps (s.)**
Carter	533,747	95	50.09
Grayloc	460,592	95	57.36
Screw nut	1,656,198	97	150.02

Tableau 1 – *Temps de calcul pour la simplification de quelques nuages de points.*

permets d'éviter l'obtention d'un nuage avec des trous larges qui peuvent être un obstacle pour les méthodes de reconstruction de surface.

La figure 4 montre les résultats de notre méthode de simplification appliquée sur quelques nuages de points représentant des pièces mécaniques. On peut constater que les régions saillantes dans les différents modèles sont généralement bien identifiées et représentées par une densité de points plus importante. Malgré tout, on remarque parfois une densité de points importante sur certaines régions planes en raison d'une présence de bruit localement, qui perturbe la mesure de courbure.

La figure 5 montre une comparaison visuelle de la détection des points vifs entre notre algorithme et celui de Song et Feng [1]. Les résultats obtenus par les deux algorithmes sont similaires. En particulier en ce qui concerne la détection des zones saillantes telles que les frontières des oreilles, le contour de la queue, etc. Toutefois, le temps d'exécution de l'approche que nous proposons semble meilleur. Les expérimentations ont été menées sur un processeur Intel(R) Core(TM) 2 Duo 2 GHz avec une mémoire de 2 Gb. La simplification du modèle *bunny* avec les mêmes paramètres (36k points, et un taux de simplification de 90%) est obtenue en 1.5 secondes alors que Song et Feng annoncent 2093 secondes sur un pentium 4, 3 GHz. Le tableau 1 résume le temps de calcul pour la simplification des modèles présentés dans la figure 4 qui reste plus performant même sur des modèles volumineux.

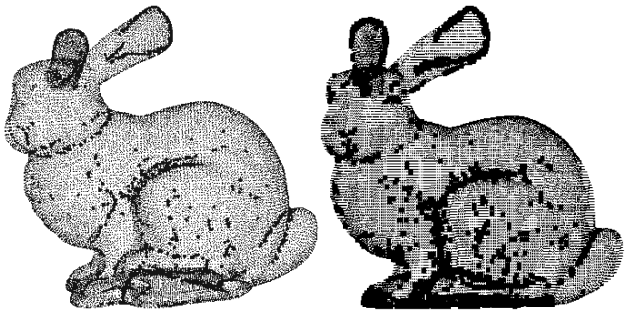


Figure 5 – *Comparaison de la détection des points vifs entre notre algorithme (à gauche), et celui de Song et Feng [1] (à droite).*

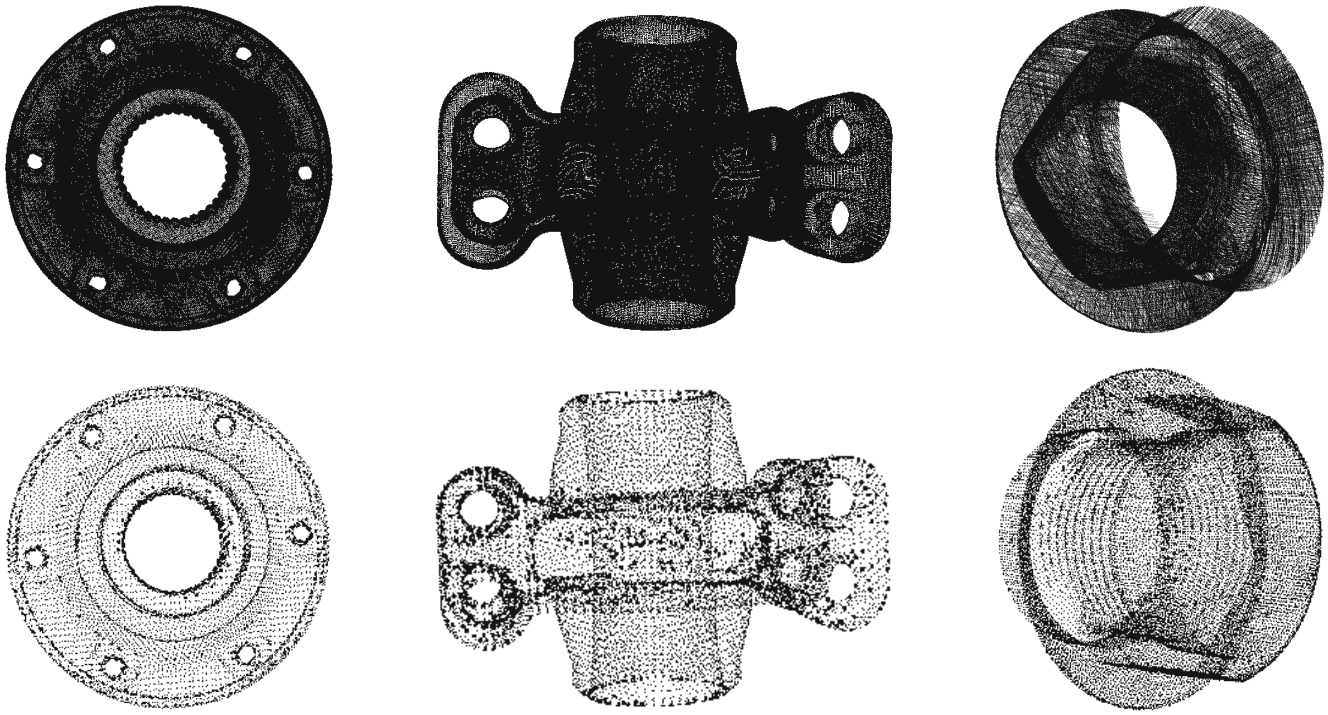


Figure 4 – *De gauche à droite : en haut les nuages denses de différents modèles mécaniques (*carter*,* grayloc*,* screw nut*), en bas les résultats de la simplification avec un taux respectif de 95%,95%, et 97%.*

Pour montrer l'utilité de la préservation des points vifs dans un nuage après sa simplification, nous proposons de reconstruire la surface d'un nuage représentant un modèle *cube* en utilisant l'algorithme de *ball pivoting* [11]. La figure 6 donne les résultats de la reconstruction obtenue basée sur le nuage original ainsi que sa version simplifiée (taux de simplification de 82%) en utilisant notre algorithme et, à titre de comparaison par un simple algorithme de clustering. La figure montre clairement que garder les points vifs permet de reconstruire une surface avec des coins vifs (deuxième colonne de la figure). De plus, la surface obtenue est visuellement identique à celle reconstruite à partir du nuage original.

5 Conclusion

Dans cet article, nous avons présenté une méthode de simplification de nuages de points rapide permettant de préserver les points vifs. La méthode est basée sur une technique hybride qui combine deux approches de simplification à savoir une approche de clustering et une approche coarse-to-fine pour identifier les régions contenant les points vifs. Le nuage résultant est caractérisé par une faible densité de points dans les régions planes, et une forte densité dans les régions vives. Cependant, notre méthode souffre de certaines limites. Par exemple, il est difficile de fixer le seuil de classification des points vifs automatiquement à cause de la sensibilité de la courbure au bruit géométrique présent dans le nuage. Par conséquent, la fixation de ce seuil nécessite l'intervention de l'utilisateur. De plus, la méthode ne fonctionne pas bien sur les données bruitées puisqu'elles sont caractérisées par une forte courbure même dans les régions planes. Une solution possible pour palier à cet inconvénient est d'utiliser un critère ou bien un descripteur géométrique plus robuste pour classifier de manière plus efficace les points vifs. En vue de travaux futurs, nous envisageons d'explorer les différents descripteurs géométriques existants dans la littérature afin de les adapter à notre méthode.

Références

[1] Hao Song et Hsi-Yung Feng. A progressive point cloud simplification algorithm with preserved sharp edge data. *Advanced Manufacturing Technology*, 45 :583–592, 2009.

[2] Michael M. Kazhdan, Matthew Bolitho, et Hugues Hoppe. Poisson surface reconstruction. Dans *Symposium on Geometry Processing*, pages 61–70, 2006.

[3] Martin Isenburg, Peter Lindstrom, Stefan Gumhold, et Jack Snoeyink. Large mesh simplification using processing sequences. Dans *In Visualization'03 Proceedings*, pages 465–472, 2003.

[4] Paolo Cignoni, Claudio Montani, Claudio Rocchini, et Roberto Scopigno. External memory management and simplification of huge meshes. *IEEE Trans. Vis. Comput. Graph.*, 9(4) :525–537, 2003.

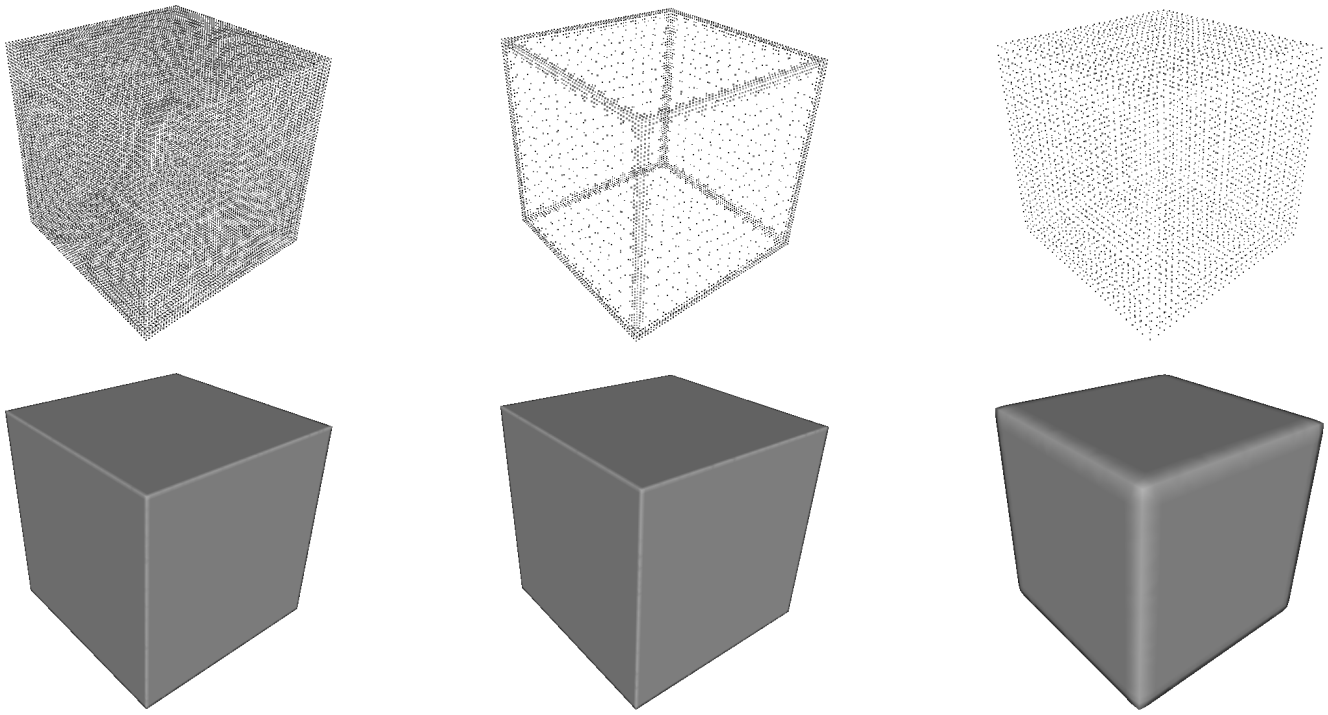


Figure 6 – *En haut, de gauche à droite : nuage original, sa simplification en utilisant notre algorithme, et un simple algorithme de clustering. En bas, la surface reconstruite des nuages correspondants en utilisant l'algorithme de* ball pivoting *[11].*

[5] Mark Pauly, Markus Gross, et Leif P. Kobbelt. Efficient simplification of point-sampled surfaces. Dans *Proceedings of the conference on Visualization '02*, VIS '02, pages 163–170, 2002.

[6] Dmitry Brodsky et Benjamin Watson. Model simplification through refinement. Dans *Graphics Interface*, pages 221–228, 2000.

[7] Eric Shaffer et Michael Garland. Efficient adaptive simplification of massive meshes. Dans *IEEE Visualization*, 2001.

[8] Carsten Moenning et Neil A. Dodgson. Intrinsic point cloud simplification. Dans *Proceedings of 14th GraphiCon '04.*, 2004.

[9] I.T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, 1986.

[10] Kun Zhou, Qiming Hou, Rui Wang, et Baining Guo. Real-time kd-tree construction on graphics hardware. *ACM Transaction On Graphics*, 27 :126 :1–126 :11, 2008.

[11] Fausto Bernardini, Joshua Mittleman, Holly Rushmeier, Claudio Silva, et Gabriel Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 5 :349–359, 1999.