Dynamic Sampling and Rendering of Algebraic Point Set Surfaces

Recette

21 Février 2017

William Caisson
Xavier Chalut
Christophe Claustre
Thibault Lejemble

A destination de :
Nicolas Mellado
David Vanderhaeghe
Adrian Basarab
Mathias Paulin

Table des matières

1 Introduction

Avec l'arrivée des méthodes de numérisation 3D telles que le scanner laser, la représentation par nuage de points devient bien plus courante pour représenter les surfaces d'objets en 3 dimensions. Cette représentation de plus en plus facile à obtenir, est assez complexe à visualiser. Les problèmes majeurs étant le bruit généré lors de l'acquisition des points et la quantité de données à manipuler si l'on souhaite représenter avec une grande précision la scène d'origine.

L'article de monsieur Guennebaud et ses collègues publié en 2008 propose une méthode réglant le problème des points bruités et permettant une visualisation pertinente de la scène même avec une plus faible quantité de données pour la représenter.

L'objectif de ce chef d'œuvre est de parvenir à une implémentation de cette méthode de visualisation de nuage de points au sein d'un plugin du moteur de rendu *Radium-Engine* . Dans un second temps, l'objectif sera de rendre l'implémentation la plus efficace possible en se basant sur l'utilisation de structures de données et d'algorithmes optimisés pour GPU par l'utilisation de la technologie CUDA.

Ce document a pour but de présenter le travail qui a été réalisé au cours de ce chef d'œuvre et de définir l'avancement final des différentes tâches initialement prévues.

2 Architecture finale

Cette partie présente les réels choix d'implémentation faits en terme de structures de données et de classes afin de répondre aux exigences de notre client. Les conceptions finales du code de la représentation du nuage de points, de la mise en place du *plugin* Radium et de l'algorithme de l'APSS sont successivement détaillées. Un accent est mis sur les modifications faites par rapport à ce qui était prévu par le document de conception.

2.1 Représentation du nuage de points

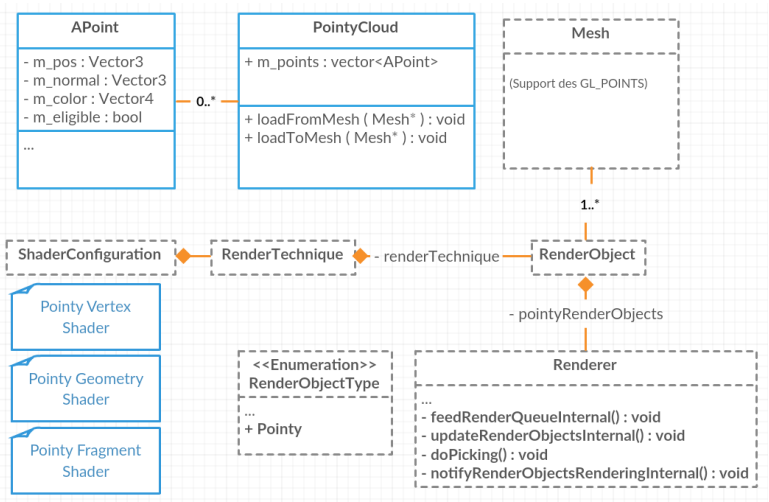


FIGURE 1 – Diagramme associé à la représentation des nuages de points.

Cette partie du code devait originalement s'appuyer sur certaines classes déjà existantes dans Radium telle que la classe **Mesh** pour laquelle les informations de faces du maillage auraient été simplement ignorées. Cependant, l'accès et la modification des sommets, normales et points étant peu aisés, et certains modules de l'APPS nécessitant une représentation de point plus compacte, une nouvelle classe **PointyCloud** a été créée. Cette classe contient ainsi un vecteur de **APoint**, une nouvelle structure qui stocke un sommet, une normale et une couleur. Ce faisant, l'algorithme de l'APSS s'effectuera sur une telle structure de données et enverra le nuage résultant à un **Mesh** correspondant afin d'en afficher le résultat. De plus, un attribut d'éligibilité à l'APSS est ajouté à chaque point. Un point ne satisfaisant pas une certaine condition sur son voisinage ne sera ainsi ni sur-échantillonné ni projeté. Ce cas est rencontré lorsqu'un sommet possède moins de 6 points dans son voisinage, ce qui rend instable la calcul local de l'APSS.

L'affichage du nuage s'effectue sur GPU grâce à trois *shaders*. Le processus de création des *splats* est le suivant. Pour chaque point et normale associée, un *quad* est créé dans le plan tangeant par le *geometry shader*. Des coordonnées 2D sont attribuées aux quatre sommets, puis interpolées par le *fragment shader* qui pourra en déduire si le fragment est localisé dans la *splat* ou non. La taille des *splats*, correspondant à la taille des *quads*, est donc décorrélé du point de vu de la caméra.

2.2 Le plugin Radium

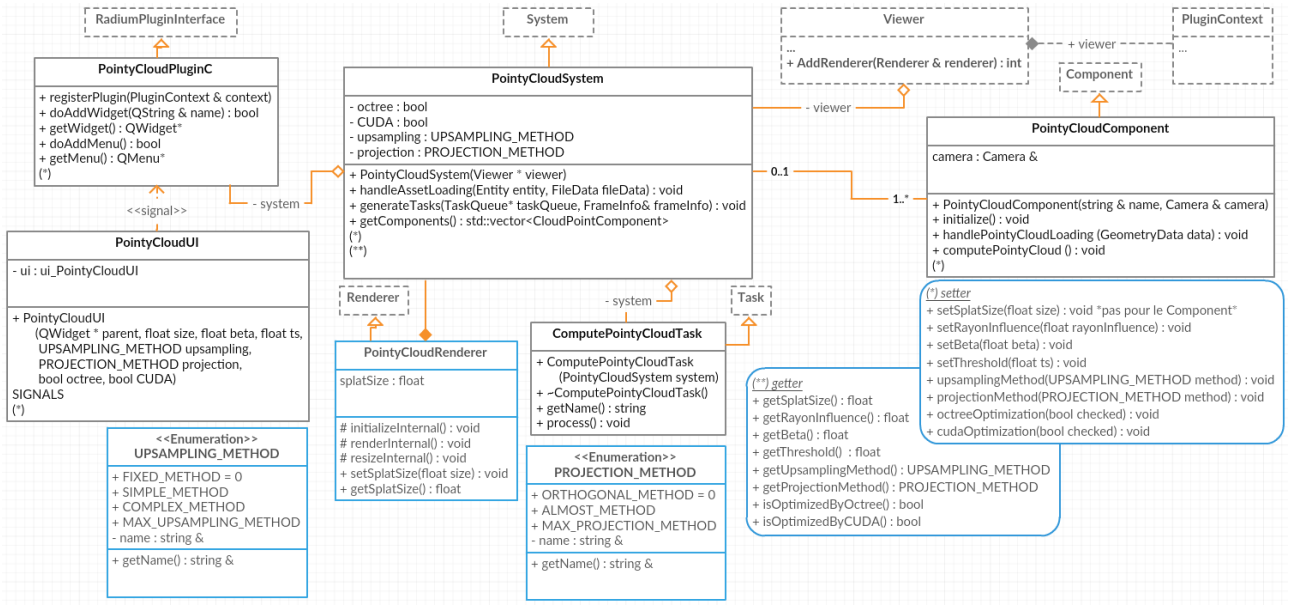


FIGURE 2 – Diagramme associé à la mise en place du *plugin*

La mise en place du *plugin*, nommé **PointyCloud**, s'est effectuée comme prévue par le document de conception. Un aperçu des classes correspondantes est disponible sur la Figure 2. Pour rappel, la classe **PointyCloudPluginC** joue le rôle d'interface avec Radium et permet de créer une interface utilisateur gérée par la classe **PointyCloudUI**. Le *plugin* crée aussi un système de type **PointyCloudSystem** qui va gérer différents **PointyCloudComponent** représentant chacun un nuage de points. A chaque calcul d'image, le système transmet à Radium une **ComputPointyCloudTask** qui effectue le calcul de l'APSS sur tous les **PointyCloudComponent** existants. Enfin, le module de sélection des voisins est constitué d'une classe

NeighborsSelection qui effectue une recherche simple dans tous le nuage, et d'une classe **NeighborsSelectonWithRegularGrid** qui accélère les requêtes de voisinage grâce à une grille régulière qui subdivise l'espace en cellules mémorisant les points contenus dans celles-ci.

2.3 L'APSS

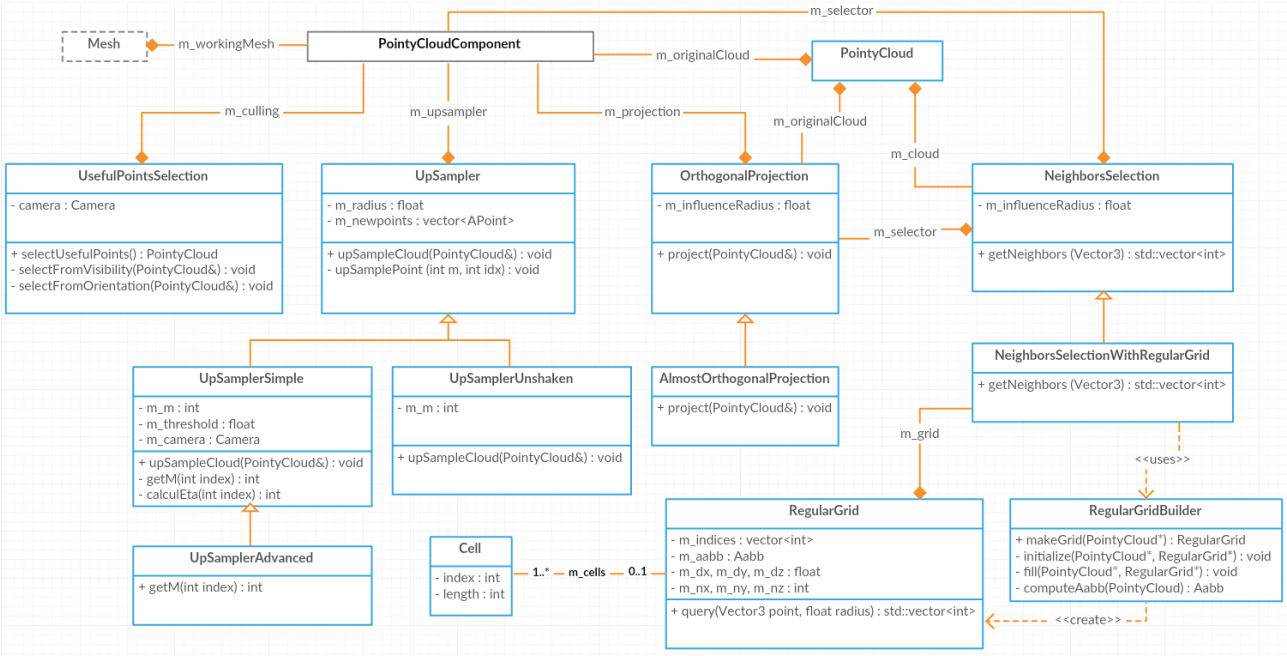


FIGURE 3 – Diagramme de l'algorithme de calcul de l'APSS

Le plupart des classes nécessaires au calcul de l'APSS ont été implémentées en suivant le schéma définit lors de l'étape de conception du projet. La différence notable entre ce qui a été prévu et le code effectivement produit réside principalement dans le changement de structure de données du nuage introduit en Section 2.1. En effet, la librairie Patate utilisée pour la projection des point sur l'APSS requière un concept de point compact contenant sa position et sa normale. La Figure 3 montre le diagramme mis-à-jour des classes liées à l'APSS. L'algorithme est effectué par le **PointyCloudComponent** sur son **PointyCloud** originalement chargé depuis un fichier. Les instances de **UsefulPointsSelection**, de **UpSampler** (ou ses dérivées) et de **OrthogonalProjection** (ou **AlmostOrthogonalProjection**) sont successivement appelées afin de finalement obtenir le **PointyCloud** désiré qui sera transmis au **Mesh** associé pour le visualiser.

3 Présentation du travail effectué

3.1 Différences entre prévisionnel et réalisé

Cette partie reprend une à une les différentes exigences définies dans le dossier de spécifications et indique leur état d'avancement.

3.1.1 Exigences principales

Les exigences principales sont dans l'ensemble respectées. Le programme est bien un *plugin* Radium (P1) qui fonctionne sous une architecture Linux(P2) et qui permet de visualiser un

P1	Le travail effectué doit s'intégrer dans le moteur de rendu *Radium-Engine* sous la forme d'un *plugin*	
P2	Le *plugin* doit pouvoir fonctionner sur un système d'exploitation `Linux` qui comporte une carte graphique `NVIDIA` permettant l'exécution de programmes `CUDA`	
P3	Le *plugin* doit permettre à *Radium-Engine* d'afficher un nuage de points	
P4	Le *plugin* doit permettre à *Radium-Engine* d'extraire un nuage de points des *fichiers .PLY* donnés en entrée	
P5	Le *plugin* doit permettre à *Radium-Engine* d'afficher une scène décrite en nuage de points de moins de **1 million de points** avec une fréquence d'affichage minimum de **50 images par seconde**	
P6	Il devra être fourni lors de la recette, le code source du *plugin*	
P7	Il devra être fourni lors de la recette, une documentation utilisateur complète décrivant comment utiliser le *plugin*	

nuage de points(P3) extrait d'un fichier .ply(P4).

Néanmoins l'exigence P5 n'est pas respectée. En effet, le programme n'est pas encore capable d'afficher 1 million de points à 50 images/seconde. Actuellement le taux de rafraîchissement sur un modèle de 1 million de points est inférieur à 1 image par seconde et s'explique par la complexité du calcul sur CPU du ré-échantillonnage et de la projection.

Concernant les exigences principales P6 et P7 celle-ci ne sont à ce jour pas encore rempli bien entendu, la recette avec notre client n'ayant pas encore eu lieu.

3.1.2 Exigences secondaires

S1	Le *plugin* doit permettre à *Radium-Engine* d'afficher une scène décrite en nuage de points de moins de **2 millions de points** avec une fréquence d'affichage minimum de **50 images par seconde**	
S2	Quelque soit le positionnement et l'orientation de la caméra, la surface des objets dans le champ de vision doivent présenter aucune ouverture de plus que celle dut à la description du nuage de points	
S3	Le *plugin* doit permettre à *Radium-Engine* d'afficher une surface lisse à partir d'un nuage de points	
S4	Le *plugin* ne doit pas empêcher *Radium-Engine* de fonctionner sous Windows et MacOS	

Deux des quatre exigences secondaires sont respectées. Les résultats obtenus montrent bien une surface lisse(S3) des objets dans le champ de vision sans aucune ouverture de plus que celles dues à la description du nuage de points(S4).

L'exigence sur le taux de rafraîchissement n'est logiquement pas atteinte au vu de l'exigence principale P5, et il n'a pas été possible d'effectuer dans les temps les tests du *plugin* sur les systèmes d'exploitation Windows et MacOS.

3.2 Avancement des modules

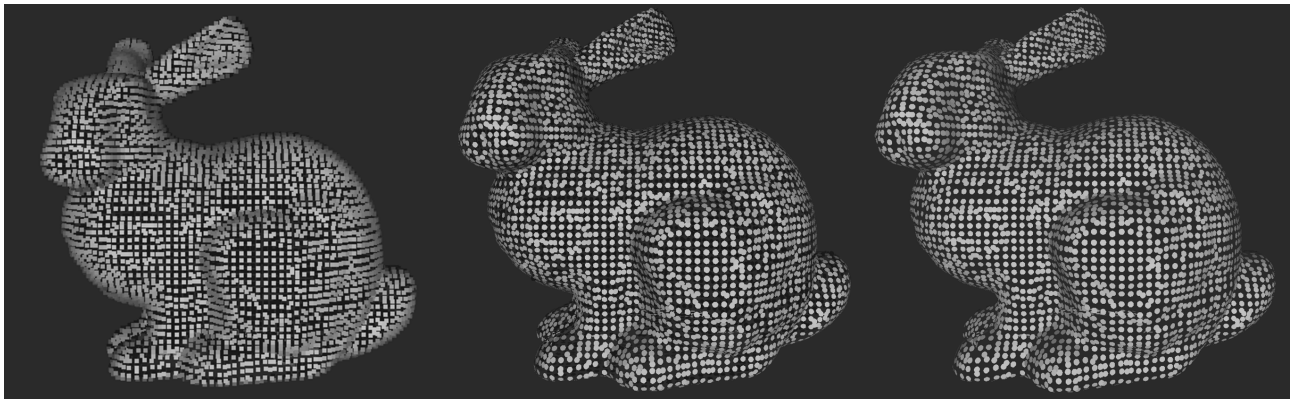
Module	État	Remarque
Récupération des nuages de points	**Complètement implémenté**	Sous réserve que la lecture de fichier de *Radium-Engine* soit fonctionnelle (aucun test complet sur cette fonctionnalité)
Visualisation	**Complètement implémenté**	Utilisation de *quad* orientés générés par une étape de *geometry shader* au lieu de ceux générés automatiquement
Sélection des points à traiter	**Partiellement implémenté :** - Adapter l'implémentation existante pour une optimisation CUDA	
Ré-échantillonnage	**Partiellement implémenté :** - Implémenter la dernière version du ré-échantillonage - Adapter l'implémentation existante pour une optimisation CUDA	Détection d'une divergence entre la première et les deux autres méthodes et de fait ajout d'une classe mère abstraite
Projection	**Partiellement implémenté :** - Implémenter la deuxième version de la projection - Adapter l'implémentation existante pour une optimisation CUDA	–
Sélection des voisins	**Partiellement implémenté :** - Adapter l'implémentation existante pour une optimisation CUDA	–
Construction de l'Octree	**Partiellement implémenté :** - Adapter l'implémentation existante pour une optimisation CUDA	Transformation de l'Octree en grille régulière

3.3 Utilisation du plugin

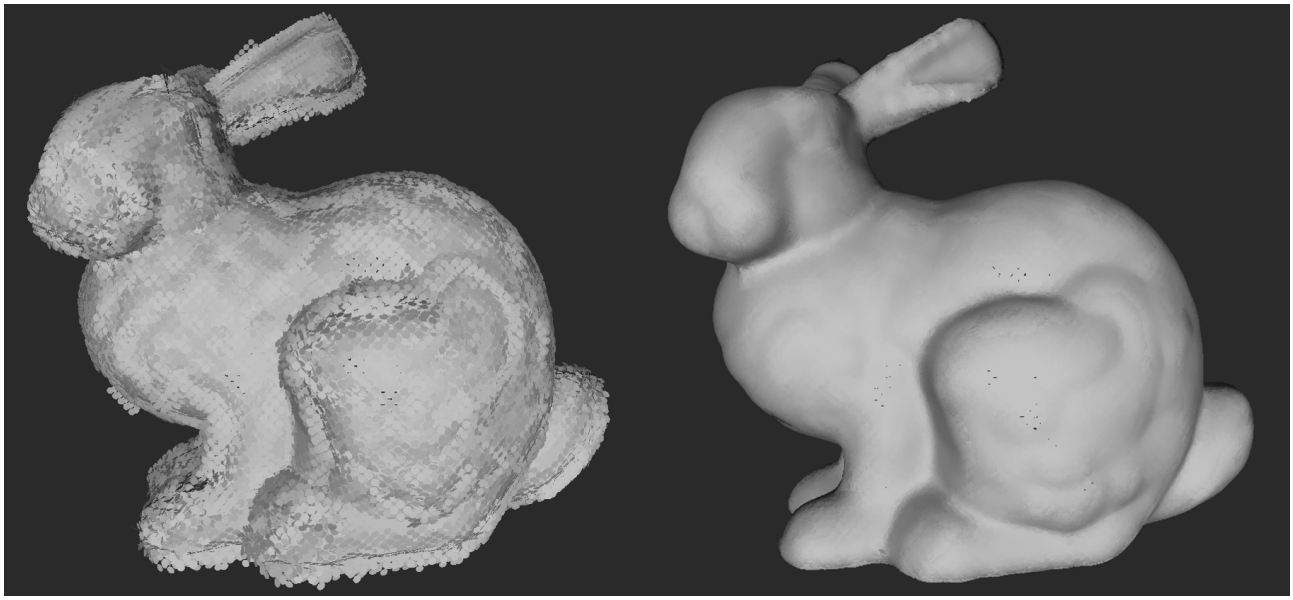
Les étapes nécessaires à l'installation et les explications sur comment utiliser notre plugin sont décritent dans la documentation utilisateur fournit avec le plugin.

3.4 Résultats

Cette partie met en avant des résultats obtenus via l'utilisation du *plugin* sur un modèle de référence. Les résultats présentés dans ce rapport ont été réalisés sur un ordinateur Linux avec un processeur Intel Core i5 2.40Gz, et une carte graphique Intel HD Graphics 520. La Figure 4 propose un aperçu des différentes étapes de l'algorithme de l'APSS.



Nuage de points de référence (splats carrées non-orientées) | Splats elliptiques orientées | Culling



Sur-échantillonnage (fixe) | Projection (orthogonale) sur l'APSS

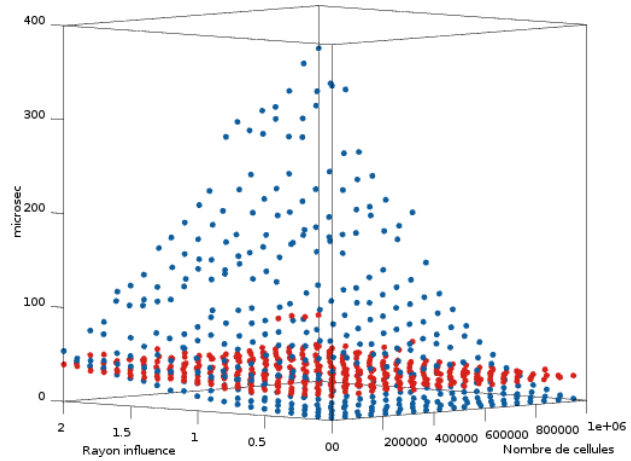
FIGURE 4 – Différentes étapes de calcul de l'APSS pour un nuage de plus de 8000 points

L'étape de ré-échantillonnage du nuage est illustrée par la Figure 5 où l'on peut visualiser le processus de génération de nouveaux *splats* pour un point uniquement.

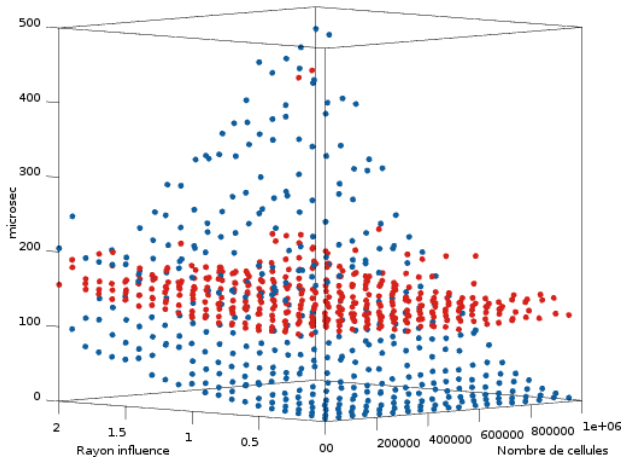
Le fichier testé à l'origine des images de la Figure 4 contient seulement 8146 points et ne permet de visualiser l'APSS qu'à 15 images par secondes, ce qui ne valide pas l'exigence principale P5 du cahier des charges (cf Section 3.1.1). Ces performances relativement faibles par rapport à ce qui était attendu s'expliquent par l'utilisation exclusive du CPU pour le calcul de l'APSS, la technologie CUDA n'ayant pas pu être mise-en-place à temps pour ce rapport.

9 25 49 81

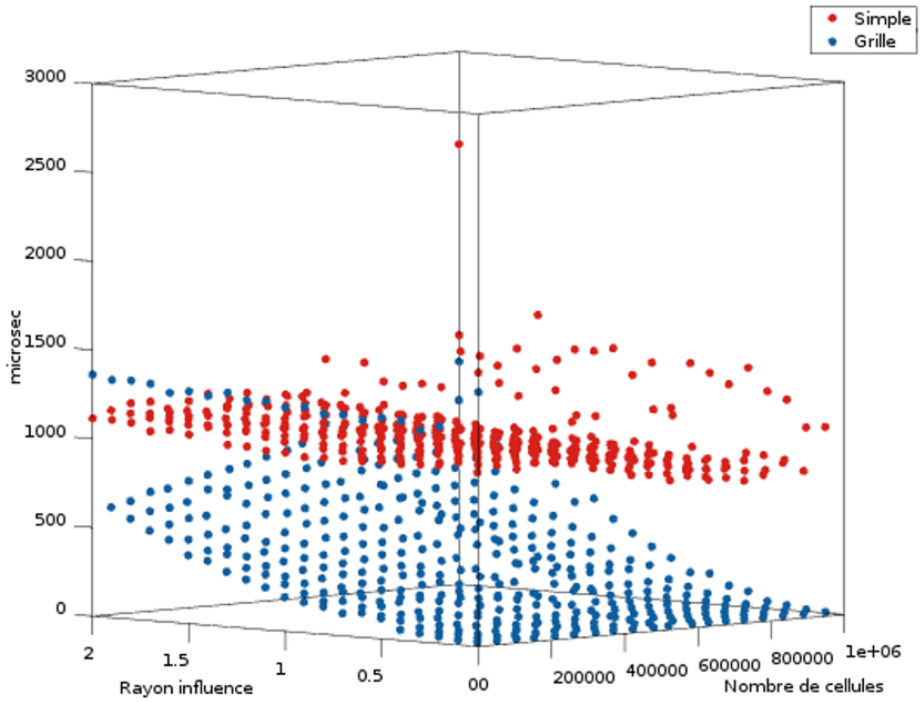
FIGURE 5 – Sur-échantillonnage et projection d'un unique point en fonction du nombre d'échantillons générés (inscrit au-dessous de chaque image)



8K



34K



208K

FIGURE 6 – Temps d'accès aux voisins d'un échantillon selon la méthode de calcul (parcours naïf du nuage en rouge, et utilisation de la grille en bleu) et en fonction du rayon d'influence, du nombre de cellules composant la grille et de la taille du nuage indiquée au-dessous de chaque graphique

3.5 Analyse de performance

Culling	0,113%
Upsampling	0,162%
Projecting	99,680%
Neighbors query	82,930%
Sphere fitting	17,054%
Point projection	0,016%
Loading in mesh	0,045%

TABLE 1 – Temps moyen pour chaque étape en pourcentage pour les paramètres par défaut de l'APSS sur différentes tailles de modèle

On peut remarquer dans la table 1 que se sont les requêtes de voisinage qui prennent le plus de temps et qui de fait ralentissent considérablement le rendu.

De plus, la grille régulière implémentée dans le but d'améliorer la requête de voisinage lors des étapes de sur-échantillonnage et de projection ne permet de gagner significativement du temps que pour des nuages dont le nombre de points dépassent approximativement 10000 points comme le montre la Figure 6.

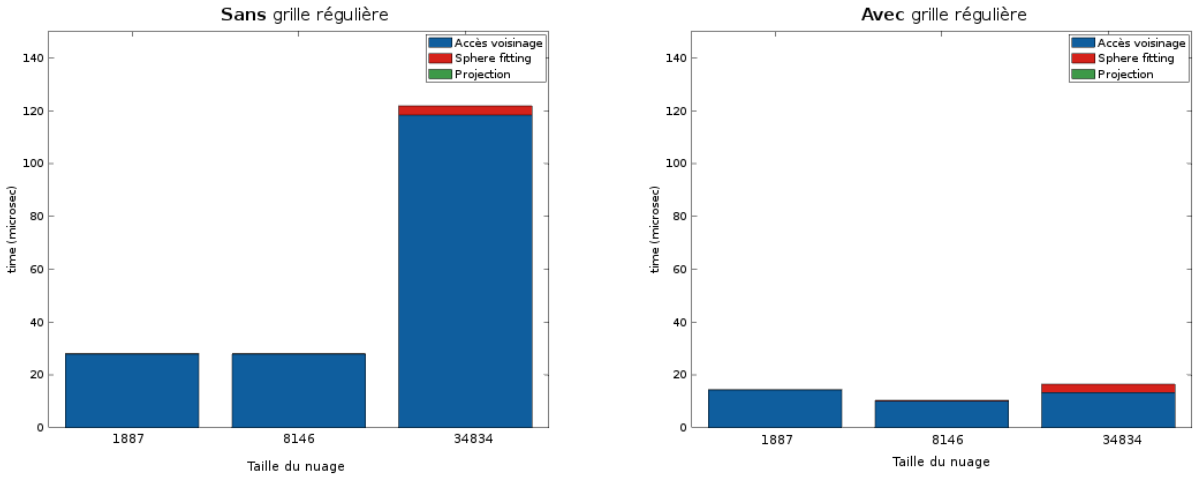


FIGURE 7 – Durées des étapes de la projection. L'accès au voisinage d'un point limite considérablement les performances comparé à l'estimation de la sphère et la projection du point. La grille régulière utilisée comporte 512000 cellules et le rayon d'influence est égal à 1.0

Aussi, on peut voir en figure 7 que le fitting de la sphère prend de plus en plus de temps en fonction de la taille du modèle. Mais cela ne représente toujours qu'une infime partie du temps passé dans la projection comparé au temps consacré aux requêtes de voisinage.

3.6 Évolutions possibles

Bien que l'ensemble des fonctionnalités initialement prévues n'est pu être totalement intégré, le *plugin* **PointyCloud** offre une architecture solide sur laquelle il est relativement facile d'ajouter des méthodes ou d'améliorer les méthodes existantes. Les évolutions possibles que nous proposons sont :

- Toutes les parties de modules prévues qui n'ont pas eu le temps d'être implémentées :
  - Implémenter la deuxième version de la projection
  - Implémenter la dernière version du ré-échantillonnage

- Adapter l'implémentation existante pour une optimisation CUDA

- Gérer le rayon de chaque *splats* indépendamment, afin de permettre au sur-échantillonnage d'adapter la taille des nouvelles *splats* aux besoins.
- L'écriture des nuages de points dans des *fichiers .PLY* afin de pouvoir sauvegarder une version calculée par notre plugin du nuage de points d'origine par exemple.
- Gérer les paramètres de l'APSS indépendamment pour chaque **Component**.

4 Conclusion

Dans le cadre de ce chef d'œuvre, notre équipe n'a pu malheureusement atteindre tous les objectifs fixés. Tout les modules sont présents mais pas dans leur version finale. Et les objectifs de performances ne sont pas atteints en très grande partie dû au fait que nous n'avons pas eu le temps d'adapter nos algorithmes pour une optimisation CUDA.

Cependant, une architecture robuste est présente et devrait pouvoir accueillir les dernières modifications à faire facilement. Les performances obtenues en terme de rapidité ne sont certes pas aussi bonnes que celles attendues mais nous avons particulièrement travaillé sur l'optimisation de nos algorithmes sur CPU et sommes tout de même fiers des résultats. Aussi les performances au niveau de la reconstruction de surface lisse (Figure 4) se rapprochent fortement de ce qui est annoncé dans l'article.