UNIVERSITÉ TOULOUSE III PAUL SABATIER — Université de Toulouse

IRIT — Institut de Recherche en Informatique de Toulouse

Dynamic Sampling and Rendering of Algebraic Point Set Surfaces

Spécifications

5 Décembre 2016

William Caisson
Xavier Chalut
Christophe Claustre
Thibault Lejemble

A destination de :
Nicolas Mellado
David Vanderhaeghe

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 à pour but de présenter l'essentiel des exigences de notre client, monsieur Nicolas Mellado chercheur à l'IRIT. En plus des exigences, nous détaillerons ici l'architecture du système à implémenter et l'organisation que nous espérons suivre tout au long du projet.

2 Exigences

2.1 Prioritaires

Ici est répertorié l'essentiel des exigences de notre client qui doivent être atteintes à la fin du chef d'œuvre.

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 millions 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

2.2 Secondaires

Ici est répertorié l'essentiel des exigences de notre client qu'il souhaiterait voir être atteintes à la fin du chef d'œuvre.

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

3 Le système

3.1 Vue générale

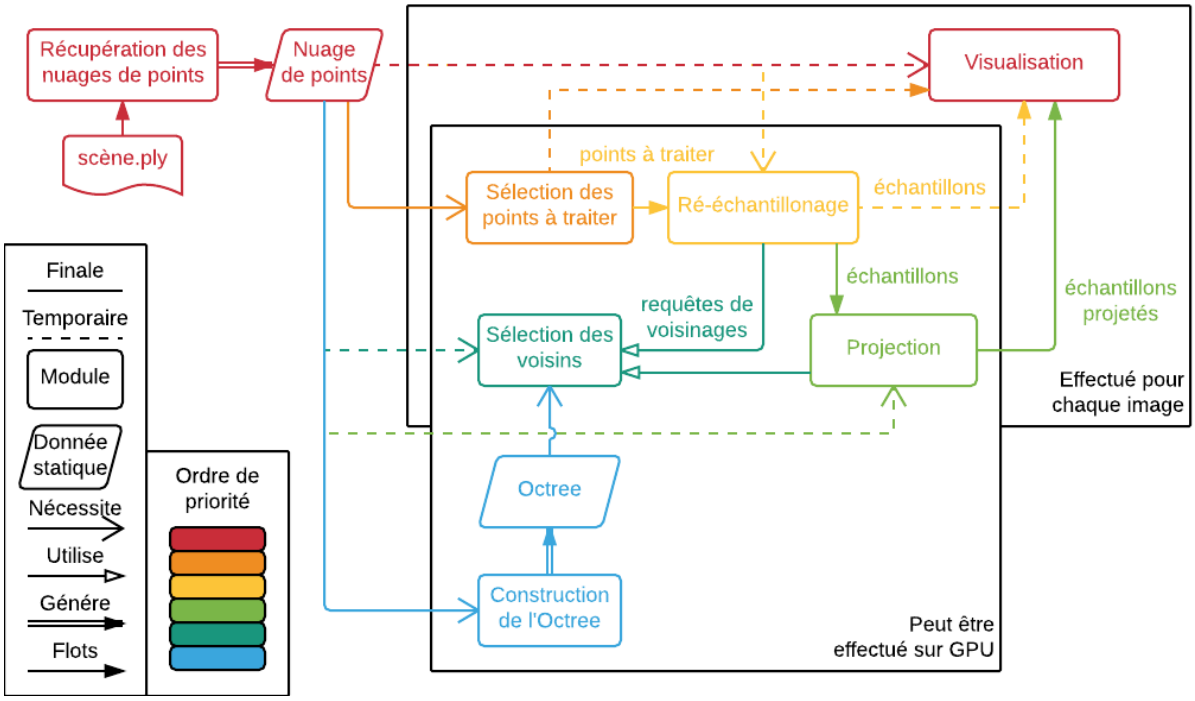

FIGURE 1 – Vue d'ensemble des différents modules

3.2 Module : Récupération des nuages de points

3.2.1 Description

Ce module doit permettre de lire et d'écrire un fichier au format .PLY et d'en extraire un nuage de points. Ce module doit pour cela définir une structure de représentation du nuage de points. Structure qu'il utilisera pour retourner le nuage de points aux autre modules. La lecture et l'écriture de fichiers au format .PLY se feront en utilisant la bibliothèque ASSIMP, déjà utilisée dans *Radium-Engine* .

Entrée	Un *fichier .PLY*
Sortie	Un nuage de points
Objectif	Extraire d'un *fichier .PLY* un nuage de points

3.2.2 Tâches associées

Représentation des nuages de points dans Radium Création de différentes classes pour représenter et manipuler un nuage de points.
Lecture
écriture des *fichiers .PLY* en entrée de Radium En association avec ASSIMP, lecture et écriture des *fichiers .PLY* .

3.3 Module : Visualisation

3.3.1 Description

Ce module doit permettre de gérer l'affichage d'un nuage de points dans le visualiseur de *Radium-Engine* . Les points devront être correctement placés dans l'espace 3D de la scène. Afin de bien les visualiser, l'affichage des points se fera par l'affichage de disque correctement orienté suivant leur normale. Ce module doit également gérer correctement l'occultation possible entre les disques.

Entrée	Des points
Sortie	Visualisation correcte de ces points
Objectif	Afficher correctement des points dans un espace 3D du point de vue de la caméra

3.3.2 Tâches associées

Visualisation des points Placement des points dans l'espace 3D et affichage de ceux-ci par un disque correctement orienté.
Gestion de l'occultation Détection des disques occultés par un autre disque et respect de l'occultation lors de l'affichage.

3.4 Module : Sélection des points à traiter

3.4.1 Description

Afin de réduire les temps de calculs, ce module doit permettre de sélectionner les points qui seront réellement visibles lors de la visualisation. Ceci dépend de la normale des points ainsi que de la position et de l'orientation de la caméra.

Entrée	Un nuage de points
Sortie	Des points à traiter
Objectif	Sélectionner uniquement les points qui ont besoin d'être traités/affichés

3.4.2 Tâches associées

Sélection des points à traiter Sélection des points visibles par la caméra.

3.5 Module : Ré-échantillonnage

3.5.1 Description

Ce module doit permettre le ré-échantillonnage des points du nuage de points. Ce module fait varier le nombre d'échantillons en fonction de la courbure de la surface décrite par le nuage de points en ce point, et en fonction de l'angle de vue de la caméra en ce point et de la position et de l'orientation actuelle de la caméra.

Entrée	Un point, ses voisins
Sortie	Plusieurs échantillons
Objectif	Générer un nombre correcte d'échantillon correspondant au point reçu en entrée en fonction de ses voisins

3.5.2 Tâches associées

Génération d'un nombre fixé d'échantillon Création de l'outil de génération des échantillons.
Calcul simple du nombre d'échantillon Création de l'outil de calcul du nombre d'échantillon à créer.
Calcul complexe du nombre d'échantillon Amélioration de l'outil de calcul du nombre d'échantillon à créer.
Génération des échantillons sur GPU Adapter l'outil de génération des échantillons et l'outil de calcul du nombre d'échantillon pour une utilisation sur le GPU via la technologie CUDA.

3.6 Module : Projection

3.6.1 Description

Via la *bibliothèque Patate* , ce module doit permettre la projection d'un ensemble de points sur l'approximation par des sphères algébriques d'une surface calculée à partir d'un nuage de points. La *bibliothèque Patate* permet déjà d'effectuer la projection d'un point sur une surface décrite par des sphères algébriques, le module doit donc permettre d'étendre cette utilisation à tout un ensemble de points. Aussi la *bibliothèque Patate* n'implémente actuellement que la méthode de projection orthogonale pour projeter les points sur la surface, ce module devra donc également permettre de projeter les points en suivant une projection presque orthogonale.

L'API Patate étant déjà utilisable sur GPU, il faudra s'assurer que ce module n'altère pas cette propriété afin de pouvoir facilement l'adapter pour la projection sur GPU.

Entrée	Un point, ses voisins
Sortie	Un point projeté
Objectif	Projeter correctement le point reçu en entrée sur la surface décrite par ses voisins

3.6.2 Tâches associées

Projection orthogonale en utilisant la *bibliothèque Patate* Implémentation de la projection des points en utilisant la *bibliothèque Patate* .
Projection presque orthogonale Surcharge de la *bibliothèque Patate* afin d'implémenter la projection **presque** orthogonale.

3.7 Module : Sélection des voisins

3.7.1 Description

Afin d'accélérer le calcul de la projection, ce module s'occupera de l'optimisation des requêtes de voisinage nécessaires au module *Projection* et à la tâche *Calcul complexe du nombre d'échantillon* .

Entrée	Un point
Sortie	Ses voisins
Objectif	Sélectionner efficacement les voisins du point reçu en entrée

3.7.2 Tâches associées

Sélection simple des voisins Implémentation de l'outil de sélection des voisins.
Utilisation de l'Octree Amélioration de l'outil afin qu'il utilise l'Octree généré par le module décrit en Section 3.8.

3.8 Module : Construction de l'Octree

3.8.1 Description

Ce module doit permettre la construction d'une structure de donnée nommée Octree à partir d'un nuage de points. L'Octree permettra d'accélérer les requêtes de voisinage (voir Section 3.7). **L'Octree doit être reconstruit à chaque modification du nuage de points.**

Entrée	Un nuage de points
Sortie	Un Octree
Objectif	Générer efficacement l'Octree décrivant le nuage de points en entrée

3.8.2 Tâches associées

Construction de l'Octree sur CPU Implémentation de l'Octree et de la conversion nuage de points vers Octree
Construction de l'Octree sur GPU Adaptation pour utilisation sur GPU

4 Organisation

4.1 Planning prévisionnel

Notre cycle de développement, illustré par le diagramme en Figure 2, suivra les méthodes agiles et sera découpé en quatre *sprints*. Chaque *sprint* sera ponctué par une phase de tests, puis une réunion avec le client afin de montrer les derniers incréments.

Une étape finale de tests d'intégration est prévue en fin de chef d'œuvre. Une période relativement longue lui est attribuée afin de disposer d'une marge sur le développement du projet en cas de problèmes. Enfin, puisque le mois de Février est entièrement consacré à la réalisation du chez d'œuvre, les *sprints* se déroulant lors de cette période passeront de deux à une semaine.

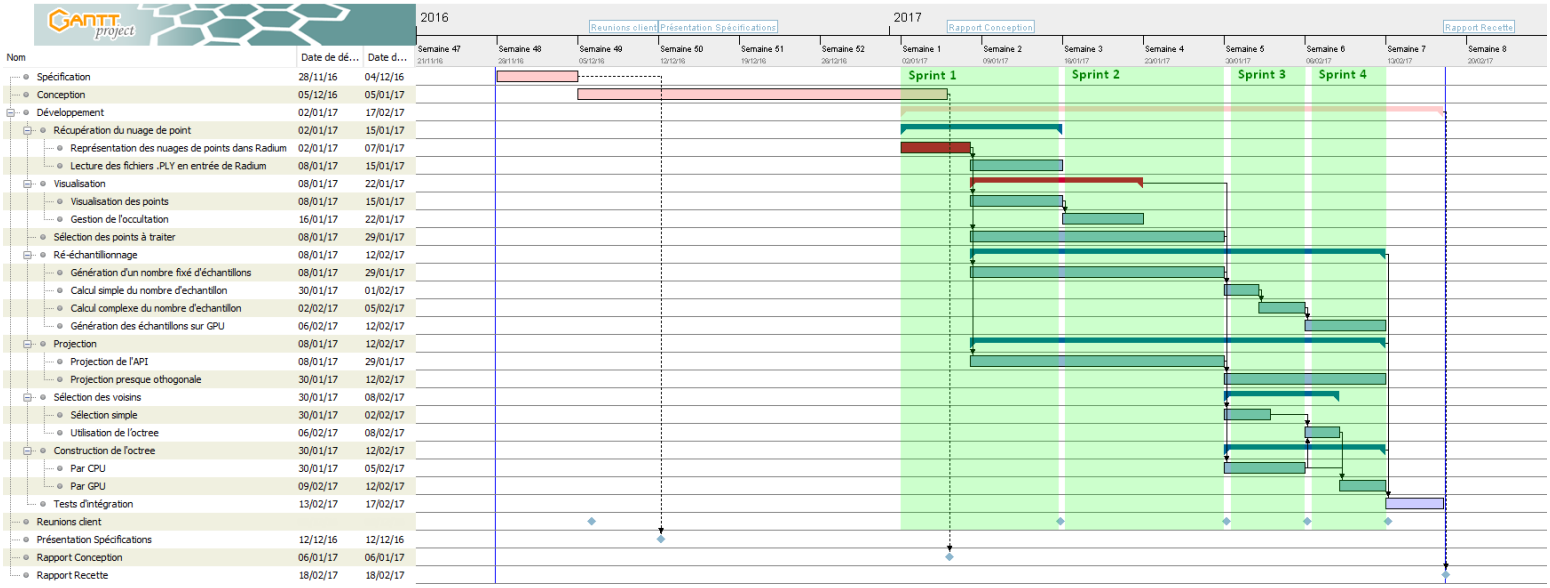

FIGURE 2 – Planning prévisionnel. En *rose* sont représentés les phases du projet, en *bleu* les tâches du développement, en *rouge* les tâches/modules critiques du développement et en *bleu clair* les tests d'intégration (servant de marge au projet)

4.1.1 Analyse des tâches

L'une des grandes priorités de ce projet va être d'avoir le plus rapidement possible un rendu visuel des nuages de points afin de pouvoir montrer l'avancement du projet à notre client. Ensuite nous devrons nous consacrer aux tâches servant à l'optimisation temporelle ou à l'esthétique.

Modules/Tâches clés :

- Représentation des nuages de points dans Radium : **Tâche critique**. Tant que cette tâche n'es pas effectuée, nous ne pouvons stocker les points et nuage de points.
- Visualisation des points : **Module critique**. C'est ce module qui nous permettra de visualiser la surface lorsque tous les autres modules seront terminés.
- Sélection des points à traiter : **Module très important**. L'application est en temps réel, ce pré-traitement sera à priori celui qui allégera le plus les temps de calculs.

- Génération d'un nombre fixé d'échantillons : **Tâche importante**. Sans cette fonctionnalité implémentée aucune raison d'effectuer de la reprojection.
- Projection en utilisant l'API : **Tâche importante**. La visualisation d'une surface lisse calculée à partir du nuage de points ne pourra se faire sans cette étape.
- Lecture des fichiers .PLY en entrée de Radium : **Tâche assez importante**, cette tâche est d'une importance plus faible mais permettra de visualiser n'importe quelle scène et permettra de tester plus en profondeur nos modules.

Les autres tâches se composent en 2 grandes catégories :

- Tâches améliorant la qualité visuelle :
  - Module Projection : Projection presque orthogonale
  - Module Ré-échantillonnage : Les autres tâches associées à ce module
- Tâches améliorant le temps de calcul :
  - Module Sélection des voisins : Toutes les tâches associées à ce module
  - Module Construction de l'Octree : Toutes les tâches associées à ce module

4.2 Outils

Comme nous l'avions évoqué dans notre précédente revue, le code sera disponible sur GitHub. La gestion des versions se fera soit en ligne de commande, à travers git ou via l'interface graphique de Ungit.

L'utilisation de la bibliothèque Patate est une des recommandations faites par notre client. En effet, elle intègre déjà plusieurs fonctionnalités qui sont utiles pour l'aboutissement de notre projet.

5 Scénarios de tests

Les tests d'intégration se feront sur un ordinateur équipé d'un système d'exploitation Linux et d'une carte graphique NVIDIA compatible CUDA afin de valider l'exigence P2.

Nom du scénario	Test plugin
Fichiers nécessaires	*Radium-Engine* avec notre plugin intégré
Actions à effectuer	Suppression de notre plugin de *Radium-Engine*
Résultats attendus	Le moteur doit fonctionner comme à l'initial
Exigences couvertes	P1

TABLE 1 – Scénario : Test plugin

Nom du scénario	Test lecture d'un *fichier .PLY*
Fichiers nécessaires	*Radium-Engine* avec notre plugin intégré, un *fichier .PLY*
Actions à effectuer	Exécuter *Radium-Engine* , charger le *fichier .PLY* , enregistrer le nuage de points chargé dans un *fichier .PLY*
Résultats attendus	Le contenu du *fichier .PLY* issu de la sauvegarde doit être le même que le contenu du *fichier .PLY* qui a servit au chargement
Exigences couvertes	P4

TABLE 2 – Scénario : Test lecture d'un *fichier .PLY*

Nom du scénario	Test affichage d'un nuage de points
Fichiers nécessaires	*Radium-Engine* avec notre plugin intégré, un *fichier .PLY*
Actions à effectuer	Exécuter *Radium-Engine* , charger le *fichier .PLY*
Résultats attendus	L'affichage du nuage de points doit se faire correctement par rapport aux données du *fichier .PLY*
Exigences couvertes	P3

TABLE 3 – Scénario : Test affichage d'un nuage de points

Nom du scénario	Test taux de rafraîchissement
Fichiers nécessaires	*Radium-Engine* avec notre plugin intégré, un *fichier .PLY* contenant une scène de 1 million de points
Actions à effectuer	Exécuter *Radium-Engine* , charger le *fichier .PLY*
Résultats attendus	Le nombre d'image par seconde indiqué doit être au minimum de 50.
Exigences couvertes	P5

TABLE 4 – Scénario : Test taux de rafraîchissement

Nom du scénario	Test taux de rafraîchissement 2
Fichiers nécessaires	*Radium-Engine* avec notre plugin intégré, un *fichier .PLY* contenant une scène de 2 millions de points
Actions à effectuer	Exécuter *Radium-Engine* , charger le *fichier .PLY*
Résultats attendus	Le nombre d'image par seconde indiqué doit être au minimum de 50.
Exigences couvertes	S1

TABLE 5 – Scénario : Test taux de rafraîchissement 2

Nom du scénario	Test non création de trous dans les surfaces
Fichiers nécessaires	*Radium-Engine* avec notre plugin intégré, 4 *fichiers .PLY*
Actions à effectuer	Exécuter *Radium-Engine* , charger les *fichiers .PLY* un à un
Résultats attendus	Les surfaces des objets doivent présenter aucune ouverture de plus que celle dues à la description des nuages de points
Exigences couvertes	S2

TABLE 6 – Scénario : Test non création de trous dans les surfaces

Nom du scénario	Test continuité des surfaces
Fichiers nécessaires	*Radium-Engine* avec notre plugin intégré, un *fichier .PLY*
Actions à effectuer	Exécuter *Radium-Engine* , charger le *fichier .PLY*
Résultats attendus	La surface des objets affichés doit être visuellement lisse
Exigences couvertes	S3

Table 7 – Scénario : Test continuité des surfaces