Université de Montpellier
Master 2 - IMAGINE

Marching Cloud

Application Qt permettant le rendu de nuages de points par Ray-marching

Rapport de projet
Projet informatique — HAI928I

Étudiants :
M. Khélian LARVET
M. Sylvain LECLERC

Encadrant :
Mme. Noura FARAJ

Année : 2022

Installation

Notre application fonctionne sur les versions suivantes :
— C++ : v17.0
— CUDA : v10.1
— Qt : v5.12.8
— libQGLViewer : v2.8

Introduction

Notre principal objectif pour ce projet était de créer une application interactive capable d'effectuer des rendus réalistes sur des nuages de points en utilisant la méthode "Ray-Marching" sur GPU. Nous avons donc mis en place différentes méthodes permettant à un utilisateur de créer des nuages de points et de les modifier. Ces nuages de points sont par la suite concaténés dans une structure "Kd-tree" permettant un rendu.

Nous avons utilisé les technologies suivantes pour notre projet :
— **Qt** : Framework orienté objet et développé en C++ offrant des composants d'interface graphique (widgets) et des signaux pour rendre une application interactive.
— **libQGLViewer** : Librairie utilisant Qt et OpenGL pour effectuer des calculs 3D.
— **Cuda** : Langage permettant d'exécuter des calculs sur le GPU.

Structures de données

Dans ce projet, nous avons utilisé une classe appelée `PointCloud` pour représenter les nuages de points dans la scène. Cette classe contient des vecteurs de positions et des normales pour chaque nuage de points, ainsi qu'un matériau pour définir l'apparence de celui-ci.

La classe `PointCloud` possède également des transformations pour chaque nuage de points, telles que la position, la rotation et l'échelle. Ces transformations permettent de déplacer, faire pivoter et redimensionner chaque nuage de points dans la scène pour obtenir l'effet désiré.

Afin de faciliter les calculs effectués sur le GPU en utilisant CUDA, nous regroupons les nuages de points en un seul objet : un grand nuage de points. Pour conserver l'information des matériaux, nous ajoutons un tableau d'entiers qui stocke pour chaque point l'index de son matériau.

Une fois ce grand nuage de points créé, nous l'utilisons pour construire un kd-tree. Cette opération est effectuée côté CPU. Le kd-tree est stocké sous la forme d'un tableau de nœuds. Ce tableau est ensuite envoyé au GPU, et les requêtes d'accès au kd-tree sont effectuées sur le GPU.

Surface implicite

Pour faire le rendu du nuage de points, nous avons besoin de connaître la distance signée entre un point de l'espace et le nuage de points, pour ce faire, nous utilisons une projection HPSS. Nous utiliserons également la dérivée de la fonction distance pour calculer la normale, indispensable au calcul de l'éclairage par Blinn-Phong.

Ray-marching

On utilise ensuite cette fonction de distance pour faire un rendu de la scène par "Ray-Marching", et plus spécifiquement par "Sphere Tracing", qui est une sous-catégorie de "Ray-Marching". Le principe est similaire à celui du raytracing, sauf qu'au lieu de calculer l'intersection entre le rayon et la surface, on avance le long

du rayon par petits pas, dont la longueur est donnée par la fonction de distance. Lorsque la distance atteint un seuil inférieur, on considère que la surface est atteinte. Lorsque la distance dépasse un seuil haut ou que le nombre d'itérations maximal est atteint, on considère que la surface n'est pas atteinte.

Interface Qt et LibQGLViewer

Notre application est scindée en deux parties :
- Une partie `Viewer` (à gauche) liée à la librairie LibQGLViewer permettant de visualiser en 3D les nuages de points et de les éditer avec une fonction "RayCast".
- Une partie `Édition` (à droite) composée de plusieurs "widgets" et signaux. Ces "widgets" permettent de modifier les informations des nuages de points (Transformations et Matériaux) qui ont été ciblés par la fonction "Raycast".

Nous avons mis en place des fonctions permettant d'importer des nuages de point, mais aussi de les créer à l'aide de maillage primitif. Un utilisateur peut donc créer les objets suivant en indiquant un indice de résolution voulue : Plan, Sphère, Cube, Torus.

Une fois le maillage créé, celui-ci peut être sélectionné et modifié dans la partie `Viewer` en utilisant les divers "widgets". Ceux dans la catégorie "Transform" modifient les vecteurs de position, rotation, échelle relatives au nuage de point qui seront ensuite utilisés pour mettre à jour la matrice du modèle. Ceux dans la catégorie "Material" modifient les vecteurs liés au Blinn-Phong (Ambiant, Diffuse, Specular).

Notre application propose également un système de chargement et de sauvegarde de scène à l'aide d'un parser JSON. Nous enregistrons l'ensemble des éléments permettant de reconstituer la scène, à savoir, les maillages primitifs utilisés, leurs transformations relatives et leurs matériaux. Dans le cas d'un nuage de point importé, nous sommes contraints de sauvegarder chaque position et chaque normale qui composent le nuage de point. Nous sauvegardons également la position et l'orientation de la caméra.

Une fois l'ensemble des éléments mis en place dans la scène, un rendu peut être effectué à l'appui de la touche "R" ou du bouton "MarchingCloud" avec les dimensions hauteur largeur passées. Le rendu sera sauvé à la racine du projet sous le nom "rendu.ppm".

Conclusion

À l'heure actuelle notre application permet pour un utilisateur de créer une scène complexe, composée de nuage de point et de la modifier avec aisance à l'aide des divers fonctions à disposition.

Le rendu n'est pas parfait, et nous observons déjà plusieurs artéfacts liés aux différentes étapes du rendu. Elles peuvent être dues aux imprécisions liées à la projection HPSS ou à la limitation du nombre d'itérations du Ray-marching. Notre méthode de rendu reste également très lente malgré son exécution sur le GPU.

Cependant, nous avions conscience de ces limitations dès le début du projet, et le but n'était pas de proposer la meilleure méthode de rendu de nuage de points, mais d'explorer de nouvelles possibilités de rendu.

Plusieurs améliorations restent possibles sur ce projet afin de le rendre plus complet et attrayant. Avec par exemple :
- Ajouter des structures d'accélérations dans l'interface pour pouvoir afficher efficacement des nuages de points massifs (avec des millions ou milliards de points).
- Ajouter des composantes supplémentaires pour pousser le réalisme avec par exemple les ombres, la réflexion, la réfraction.