RAPPORT DE CONCEPTION DÉTAILLÉE Global Illumination with Radiance Regression **Functions** ETUDIANTS: Yannick BERNARD Pierre GUERINEAU Kevin MENIEL Romain MOUTRILLE Matthias ROVES ENCADRANT: Mathias PAULIN 18 janvier 2017

Table des matières

1	Introduction	1	
2	Vue d'ensemble détaillée du système	1	
3	Structures de données et décomposition en classes 3.1 Module d'extraction	3	
4	Tests unitaires 4.1 Extraction Module	6	
5	Planning prévisionnel 5.1 Analyse des risques	7	

1 Introduction

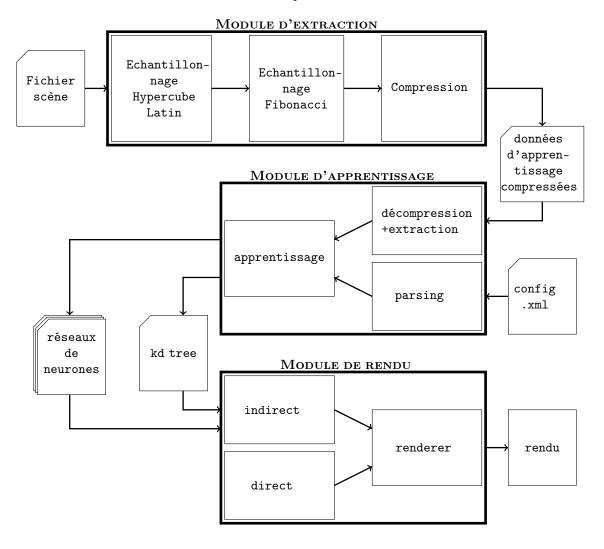
L'éclairage indirect est un des problèmes à résoudre lors du rendu physiquement réaliste d'une scène en temps réel. Un effet d'éclairage global détaillé s'obtient grâce à de nombreux rebonds effectués par la lumière, ceci engendrant un temps de calcul coûteux. Pour contourner ce problème, nous utiliserons la « Fonction de Régression de la Radiance » (RRF : Radiance Regression Function) nous permettant ainsi de calculer l'éclairage indirect pour chaque point des surfaces en connaissant la direction de la caméra et les conditions lumineuses.

Le projet a pour objectif de pouvoir effectuer le rendu physiquement réaliste d'une scène en temps réel avec calcul de l'éclairage global par une fonction de régression de radiance. Cette fonction sera modélisée par un réseau de neurones qui permettra donc un calcul de l'éclairage indirect en temps réel lors du rendu.

Dans ce rapport, nous présentons de façon détaillée la conception de notre projet. Nous avons 3 modules indépendants qui sont connectés de par leurs entrées/sorties. Il s'agit de fichiers de différents formats (XML, ASCII,...). Les modules sont un module d'extraction des données, un module d'apprentissage et un module de rendu. Eux-mêmes découpés en différentes classes. Nous avons présenté cette conception sous forme de diagrammes de classes du langage UML 2.0.

Nous détaillerons également dans ce document les tests unitaires à réaliser pour les différents modules et nous présenterons des versions raffinées et mises à jour de l'analyse des risques et du planning.

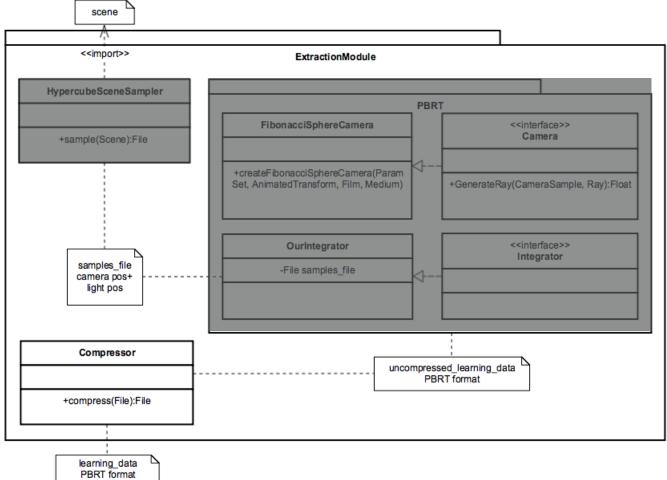
2 Vue d'ensemble détaillée du système



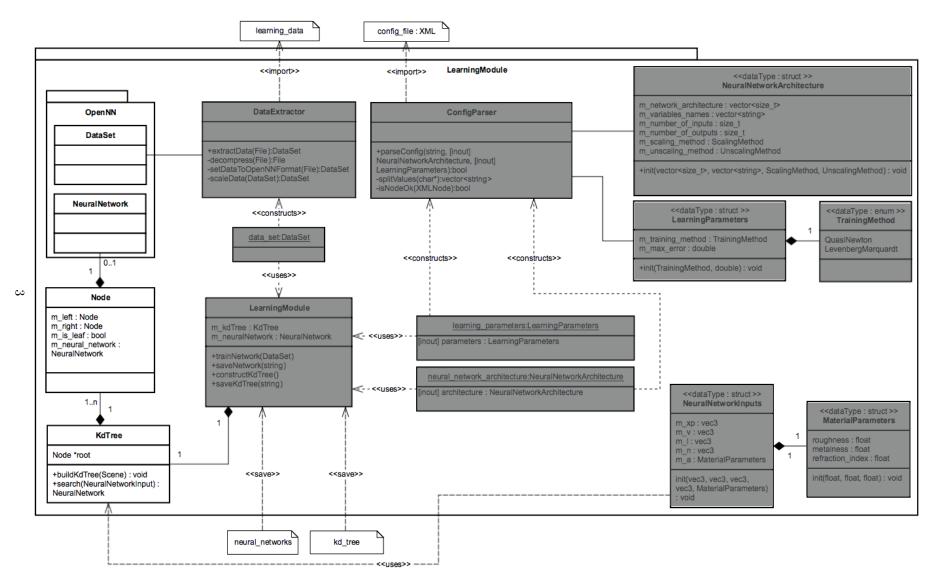
Structures de données et décomposition en classes

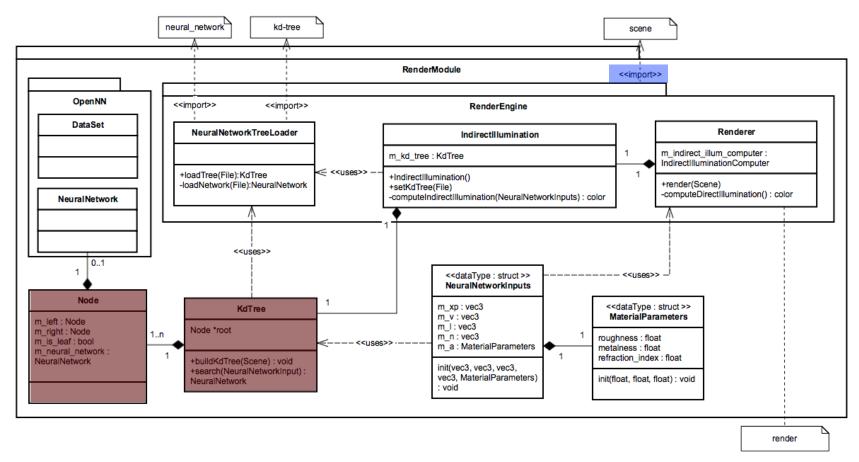
ಬ

3.1 Module d'extraction



ယ





4 Tests unitaires

4.1 Extraction Module

classe HypercubeSceneSampler

sample - Affichage des position des samples dans la scène à l'aide d'OpenGL et affichage de GL_POINT.

classe Compressor

compress - Vérification de l'intégrité des fichiers compressés dans le module de compression : On compresse le fichier, puis on le décompresse en vérifiant à l'aide de la commande bash diff que le contenu du fichier après compression/décompression est identique.

classe Camera

GenerateRay - On vérifie que le rayon généré a de bonnes propriétés et que celui-ci permet de calculer des couleurs pour des rayons connus par avance de manière cohérente au vu des paramètres passés en entrée.

${\it classe} \ {\bf Fibonacci Sphere Camera}$

createFibonacciSphereCamera - Verification que la caméra a été correctement créée et que les rayons peuvent être envoyés de manière uniforme depuis cette camera. Vérification visuelle en affichant les échantillons générés par des points en OpenGL.

Vérification de l'exactitude des données

- Execution du module sur une scène très simple et vérification des résultats. scène simple : trois plans perpendiculaires, monochromes et de couleurs différentes, avec une source placée de manière a éclairer les 3 plans.
- Modification d'un paramètre (ex : couleur, position lumière...) et vérification des résultats.

Capacité de traiter de n'importe quelle scène

Execution du module sur différentes scènes (Cornell Box avec différents objets plus ou moins complexes), de configurations différentes.

4.2 Learning Module

classe LearningModule

constructKdTree - On implémente une méthode drawKdTree() permettant l'affichage d'un KdTree à l'aide d'OpenGL, en affichant ses limites avec des lignes. On vérifiera que la scène est bien divisée, si besoin est.

classe DataExtractor

- extractData vérification que le nombre d'inputs/outputs du opennn::DataSet généré est correct
- decompress On compresse le fichier, puis on le décompresse en vérifiant à l'aide de la commande bash diff que le contenu du fichier après compression/décompression est identique.
- setDataToOpenNNFormat la construction du opennn::DataSet ne doit pas retourner d'erreur.

Vérification de la déduction

Génération de 1000 inputs aléatoires et 1000 outputs étant une combinaison linéaire des inputs, application de la cross-validation, apprentissage sur 700 données et vérification sur 430.

Vérification de l'exactitude des données évaluées

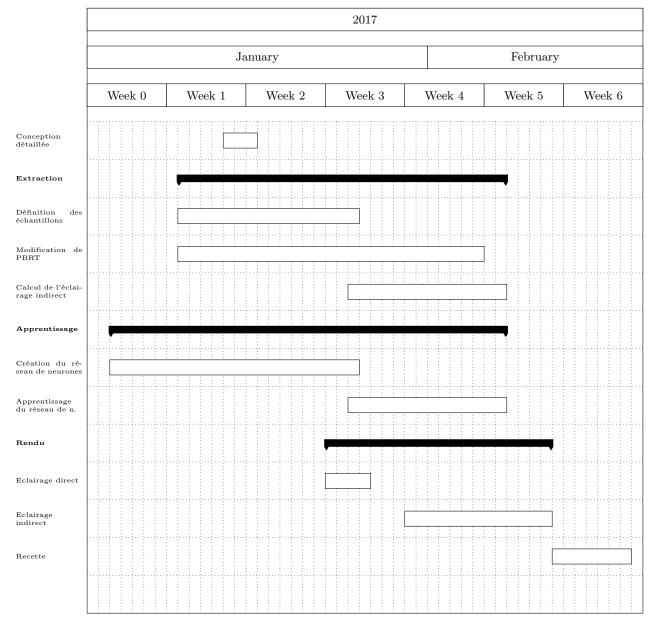
Application de la cross-validation sur les données d'apprentissage : Apprentissage avec 70% des données et test sur les 30% restants, comparaison des résultats.

4.3 Render Module

Vérification de la validité du rendu

- Comparaison avec/sans ajout de la composante indirecte.
- Vérification visuelle.
- Comparaison d'une frame du rendu avec un rendu équivalent effectué avec PBRT.

5 Planning prévisionnel



Nom	Date début	Date fin
Conception détaillée	14/01/17	16/01/17
Extraction	10/01/17	07/02/17
Définition des échantillons	10/01/17	25/01/17
Modification de PBRT	10/01/17	05/02/17
Calcul de l'éclairage indirect	25/01/17	07/02/17
Apprentissage	04/01/17	07/02/17
Création du réseau de n.	04/01/17	25/01/17
Apprentissage du réseau de n.	25/01/17	07/02/17
Rendu	23/01/17	11/02/17
Eclairage direct	23/01/17	26/01/17
Eclairage indirect	30/01/17	11/02/17
Recette	12/02/17	18/02/17

5.1 Analyse des risques

nº	Type	Description	Probabilité	Impact	Solutions	
	, -	_		_	Evité	Accepté
1	_	Impossibilité de compresser les données d'apprentissage	•	•	Implémenter un compresseur avant l'apprentissage	Ne pas compresser les données
3	_	Le KdTree se construit mal	•	•	Implémenter l'algo- rithme de construction au plus tôt	Plans de coupe aléatoires
3	_	Durée trop grande d'apprentissage	•	•	Commencer l'apprentissage le plus tôt possible et en connaître le temps estimé	On n'exploite que les données valides
4	_	Impossibilité d'utiliser l'algorithme de Leven- berg Marquardt	•	•	Faire fonctionner l'algorithme au plus tôt	Utiliser l'algorithme QuasiNewton
5	_	Difficultés d'utilisation de PBRT	•	•	Se documenter et se former au plus tôt sur PBRT	Utiliser Mitsuba
6	_	Ne pas avoir une base fonctionnelle de notre moteur	•	•	Réaliser le module d'évaluation au plus tôt	Utiliser le Radium Engine
7	_	Impossible de paramétrer l'erreur d'apprentissage	•	•	Se documenter au plus tôt sur OpenNN	Modifier les fichiers OpenNN
8	_	Impossible d'utiliser OpenNN pour chaque fragment lors du rendu	•	•	Se documenter au plus tôt sur OpenNN	Modifier les fichiers OpenNN
8	_	Difficultés lors du développement CUDA	•	•	Linker CUDA et vérifier son fonc- tionnement avant le développement du module de rendu	Traiter les parties CUDA sur CPU

Probabilité & Impact : • faible • moyen • forte • très fort