

RAPPORT DE CONCEPTION DÉTAILLÉE

Global Illumination with Radiance Regression Functions

ETUDIANTS :

Yannick BERNARD

Pierre GUERINEAU

Kevin MENIEL

Romain MOUTRILLE

Matthias ROVES

ENCADRANT :

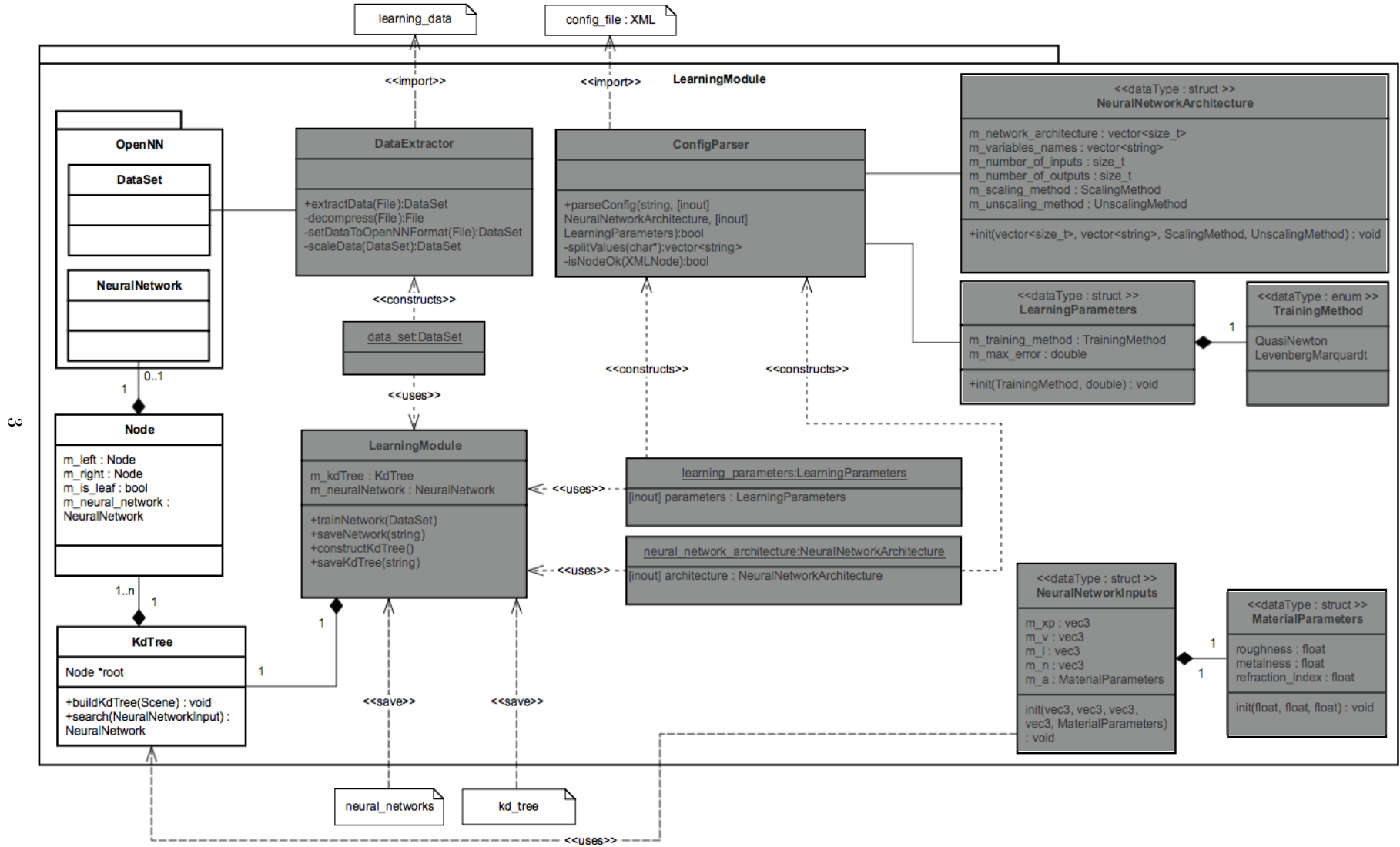
Mathias PAULIN

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	2
3.1	Module d'extraction	2
3.2	Module d'apprentissage	3
3.3	Module de rendu	4
4	Tests unitaires	5
4.1	Extraction Module	5
4.2	Learning Module	6
4.3	Render Module	6
5	Planning prévisionnel	7
5.1	Analyse des risques	8

3.2 Module d'apprentissage



4 Tests unitaires

4.1 Extraction Module

classe HypercubeSceneSampler
<code>sample</code> - Affichage des position des samples dans la scène à l'aide d'OpenGL et affichage de <code>GL_POINT</code> .
classe Compressor
<code>compress</code> - 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 <code>diff</code> que le contenu du fichier après compression/décompression est identique.
classe Camera
<code>GenerateRay</code> - 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.
classe FibonacciSphereCamera
<code>createFibonacciSphereCamera</code> - 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
<ul style="list-style-type: none">— 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 (<i>Cornell Box</i> avec différents objets plus ou moins complexes), de configurations différentes.

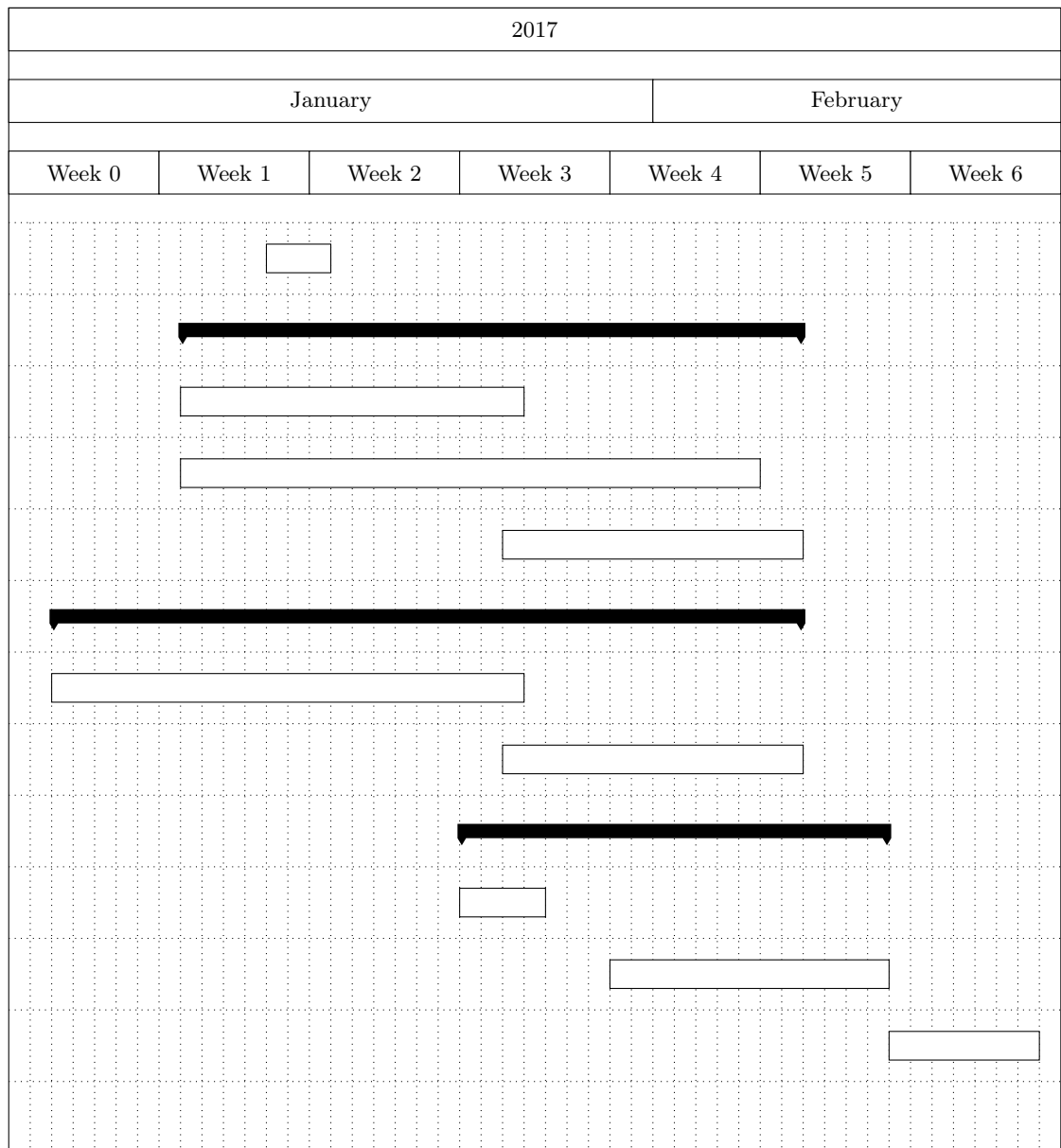
4.2 Learning Module

classe LearningModule
<code>constructKdTree</code> - On implémente une méthode <code>drawKdTree()</code> 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
<ul style="list-style-type: none">— <code>extractData</code> - vérification que le nombre d'inputs/outputs du <code>opennn::DataSet</code> généré est correct.— <code>decompress</code> - On compresse le fichier, puis on le décompresse en vérifiant à l'aide de la commande bash <code>diff</code> que le contenu du fichier après compression/décompression est identique.— <code>setDataToOpenNNFormat</code> - la construction du <code>opennn::DataSet</code> 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
<ul style="list-style-type: none">— 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

n°	Type	Description	Probabilité	Impact	Solutions	
					Évité	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'algorithme 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 Levenberg 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 fonctionnement avant le développement du module de rendu	Traiter les parties CUDA sur CPU

Probabilité & Impact :  faible  moyen  forte  très fort