RAY OF GOD
Rapport Recette

Yuko Baudet - Valentin Bonnesoeur - Bastien Schatt

20 février 2015

Clients : Mathias Paulin - David Vanderhaeghe

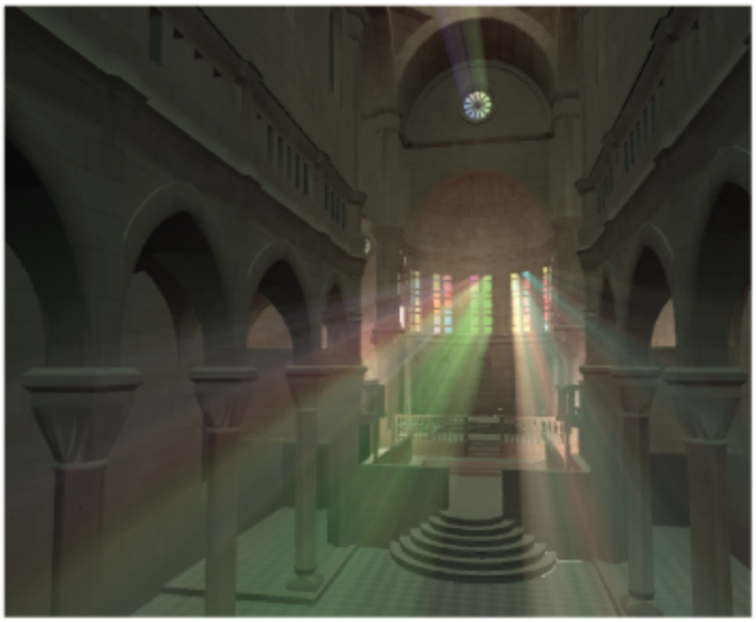


Table des matières

1 Introduction

Ce rapport entre dans le cadre du Chef d'Oeuvre du Master Image et Multimédia. Notre sujet concerne le développement d'une approche physique réaliste d'un modèle d'ombrage volumique en temps réel destiné au laboratoire de l'IRIT.

Ce dernier rapport servira à conclure notre chef d'oeuvre. Il s'agit de présenter le logiciel que nous avons conçu, expliquer le fonctionnement de ses principaux modules. Ce document permet également de mettre en avant les différences majeures entre ce qui avait été initialement prévu et ce qui aura été finalement effectif.

2 Fonctionnement du logiciel

Cette première partie de ce rapport permet de décrire les tâches principales de notre chef d'oeuvre. Mais dans un premier temps, rappelons l'algorithme du projet :

1. Création de la *shadow map* et de la *depth map*
2. Rectification épipolaire de la *shadow map*
3. Création des arbres min-max pour chaque lignes de la *shadow map* rectifiée
4. Approximation de la *scattering intégrale* par décomposition en éléments simples
5. Calcule de la luminance finale en réunissant les termes de la SVD (étape 4) ainsi que la visibilité (étape 3)

2.1 Réctification épipolaire de la shadow map

La rectification épipolaire attribue aux points de la scène, en coordonnées (x,y,z), de nouvelles coordonnées (α, β, γ) (cf figure 1), où β et γ sont respectivement les angles entre le rayon de vue lancés depuis la caméra avec l'axe camera $\rightarrow$ source, et les angles entre le rayon lumineux émis depuis une source et l'axe camera $\rightarrow$ source. α défini l'angle de rotation du plan épipolaire autour de l'axe camera $\rightarrow$ source (cf figure 1). Cela permet de caractériser les rayons de manières indépendantes. Les rayons de vue d'un plan épipolaire intersectent chaque rayons lumineux de ce plan. Par contre, les rayons d'un plan ne peuvent pas couper les rayons d'un autre plan.

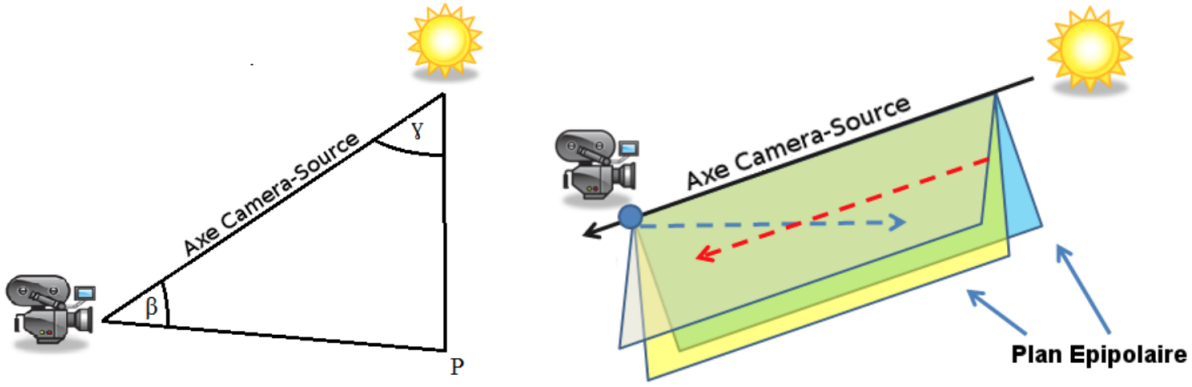


FIGURE 1 – (Gauche) Représentation des coordonnées β et γ. (Droite) Représentation des plans épipolaire

Chaque pixel de la *shadow map* rectifiée correspond à un couple (γ, α), c'est-à-dire un rayon lumineux, et contient la valeur de l'angle β du point à l'intersection entre le rayon lumineux et la scène. Nous partons de la texture de la *shadow map* rectifiée. Par conséquent, le couple (γ, α) est dès le départ connu. Seul β, alors inconnu, est calculé à partir du point d'intersection p. Ce point est lui aussi une inconnue mais est calculé grâce à la *shadow map*. En effet, en effectuant la projection inverse d'un point de la *shadow map*, nous retrouvons la coordonnées de ce point dans l'espace monde. Cette coordonnées dans l'espace image de la *shadow map* est déterminée grâce aux coordonnées (γ, α) connues du rayon lumineux et aux coordonnées du point épipolaire, c'est-à-dire le point d'intersection dans le plan image de la *shadow map* avec l'axe camera $\rightarrow$ source.

L'objectif final de cette fonctionnalité est d'indexer efficacement les points de la scène en fonction de leur coordonnées épipolaires afin de savoir facilement si un espace est illuminé ou non. L'espace est découpé en plan 2D indépendant (cf figure 2 et 3), l'intégration de la contribution lumineuse du milieu participant est efficacement calculée.

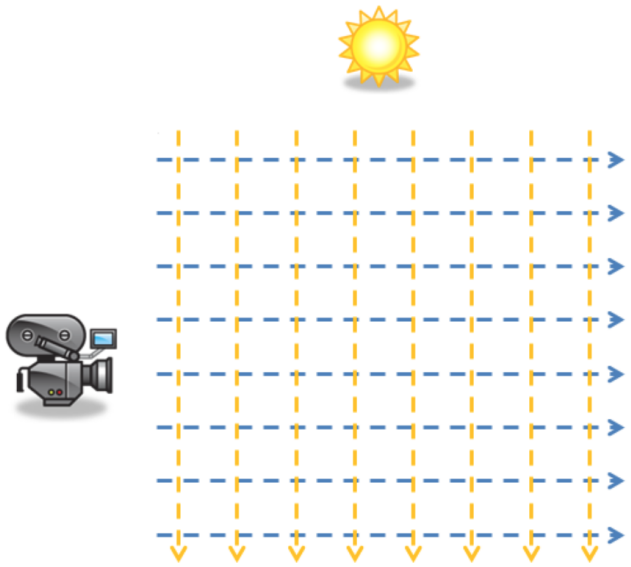


FIGURE 2 – Représentation des rayons de vue et de lumières après rectification

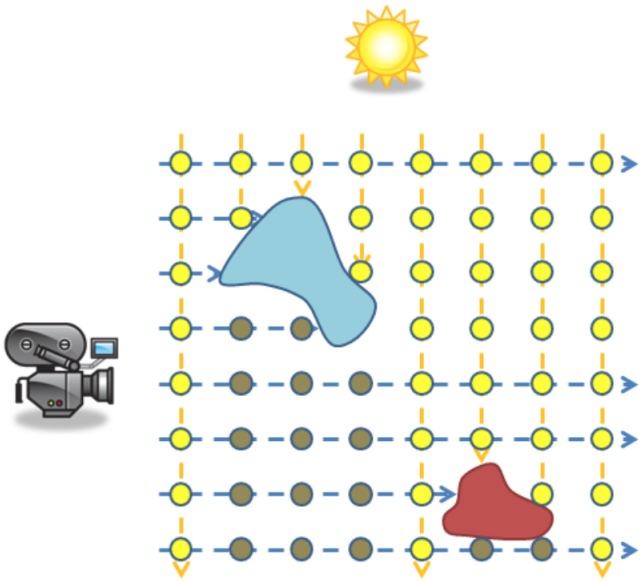


FIGURE 3 – Représentation des rayons de vue et de lumières après rectification

2.2 Construction des arbres min max

Le but de cette partie consiste à construire un arbre min-max pour chaque ligne de la *shadow map* rectifiée. Afin de calculer les valeurs minimum et maximum de chaque ligne, la technique du "ping pong" entre deux textures sera utilisé. Le principe du "ping pong" utilisé dans ce chef d'oeuvre consiste à trouver les valeurs *min* et *max* entre la coordonées de texture courante et sa voisine de droite. Une fois ces valeurs trouvées, elles sont rangées dans les composantes rouge et verte de la coordonnée courante.
Autrement dit, il s'agit d'effectuer une réduction parallèle jusqu'a ce que dans la première coordonnée de texture se trouvent les valeurs *min* et *max* de toute la ligne. Cette recherche de valeurs est donc implémentée sur GPU afin d'assurer la parallélisation du programme.

Au fur et à mesure qu'est effectué le "ping pong", chaque texture est copiée dans une troisième texture, qui permettra d'obtenir la structure en arbre min-max. La figure suivante permet de se faire une idée de la façon dont cette texture est utilisée.

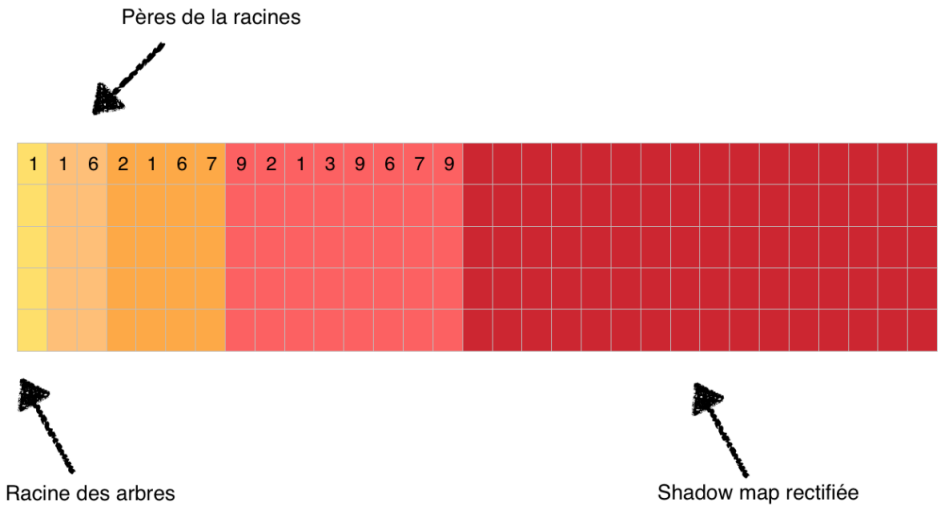


FIGURE 4 – Façon dont sont rangées les valeurs dans la texture afin de former un arbre

Une fois la texture remplie, elle servira à être utilisée en tant que arbre

min max. Afin de pouvoir mieux imaginer le résultat, les valeurs de la texture de la figure précédente peuvent être représenté sous forme d'un arbre sur la figure suivante.

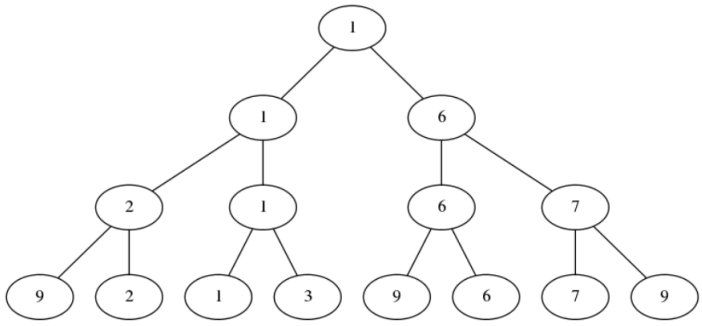


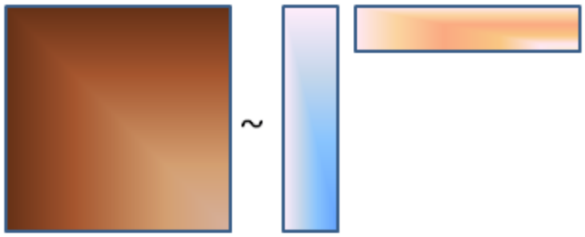
FIGURE 5 – Visualisation de l'arbre issue d'une ligne de la texture

Le type de remplissage de la texture est appelé l'indexage de Ahnentafel.

2.3 Approximation en valeurs singulière de la scattering intégrale

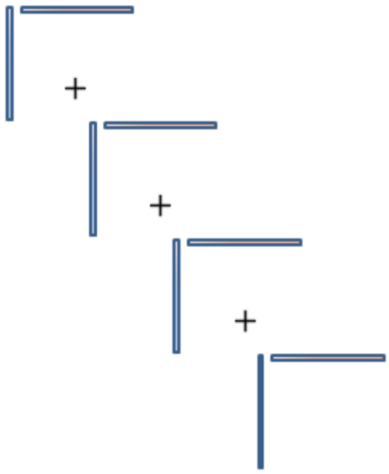
Nous souhaitons précalculer un maximum de termes sur CPU afin rendre les *shaders* plus "léger". Pour ce faire, tous les termes qui ne dépendent pas de la visibilité (β) sont *sampler* dans une matrice (de taille 64x64). Ces termes dépendent donc de α et γ.

Nous utilisons la décomposition en valeurs singulières (SVD) pour décomposer cette matrice en valeurs singulières et en vecteurs propres. La matrice peut alors être approximée en multipliant les vecteurs propres entre eux ($M = USV^*$). La figure ci-dessous illustre cette approximation.



Afin d'approximer la matrice de façon plus précise, nous utilisons plusieurs vecteurs propres. Les auteurs, de l'article que nous implémentons, remarquent que les 4 premiers vecteurs propres suffisent pour approximer la matrice avec suffisamment de précision. On utilise donc 4 vecteurs propres (on multiplie les valeurs singulière avec l'un des deux) pour approximer la matrice.

L'approximation est alors la somme des produits de vecteurs, comme le montre la figure ci-dessous. On remarque notamment que l'approximation d'une case de la matrice peut se faire par un simple produit scalaire.



Dans notre implémentation, nous utilisons des textures 1D (1x64) sur 4 composantes (RGBA) pour stocker les vecteurs propres. Cela nous permet

notamment de mettre à jour les vecteurs propres très simplement mais aussi de pouvoir bénéficier de l'interpolation matérielle des cartes graphiques.

2.4 Algorithme final

Une fois que ces trois premières parties sont terminées, il ne reste plus qu'à intégrer la contribution du milieu participant dans l'éclairage final. Pour chaque rayon de vue, il faut intégrer la participation du milieu à partir des coordonnées épipolaires, c'est-à-dire en calculant les coordonnées (α, β, γ) du point, à partir de ses coordonnées cartésiennes.
La contribution du milieu utilise l'arbre *minmax* à la ligne α et intègre l'approximation de la *scattering intégrale* (calcul permettant l'éclairage final) en fonction de la visibilité du rayon.
Afin de connaître la partie du rayon éclairée, on se base sur les valeurs min (partie supérieure du noeud) et max (partie inférieure du noeud) de chaque noeud de l'arbre. Lors du parcours, si β est compris entre le min et le max d'un noeud, cela signifie que le rayon est partiellement éclairé et qu'il faut poursuivre la recherche dans les fils. Si la valeur max est inférieure à β, alors cela correspond à une zone non éclairée et si la valeur min du noeud est supérieure à β alors la partie du rayon correspondante est entière illuminée.
Enfin, il ne reste plus qu'à utiliser les vecteurs propres pour approximer les valeurs de l'intégrale dont nous avons besoin.
Pour finir, la luminance du pixel saturé (pour simuler l'extinction du milieu) est additionné à la valeur calculée ci-dessus (estimation du "scattering").

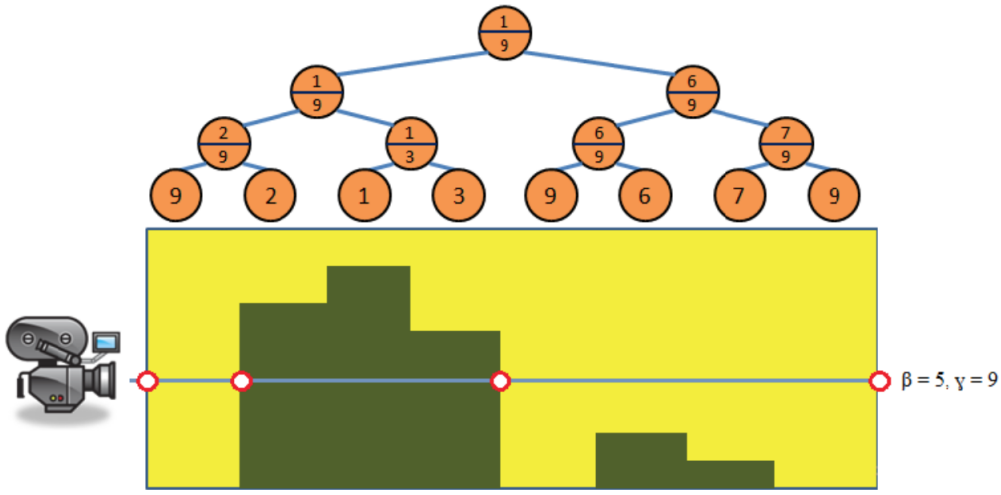


FIGURE 6 – Intégration de la luminosité des parties visibles. Dans cet exemple, il s'agit des parties du rayon en zone vert clair

3 Déroulement du projet

Au fur et à mesure de notre avancée dans le projet, nous avons été ammenés à effectuer des modifications de notre planning initiale. Afin de faire une retrospéction de notre organisation lors de ce chef d'oeuvre, nous avons souhaité comparer le planning initialement prévu ainsi que le planning effectif final.

3.1 Planning initial

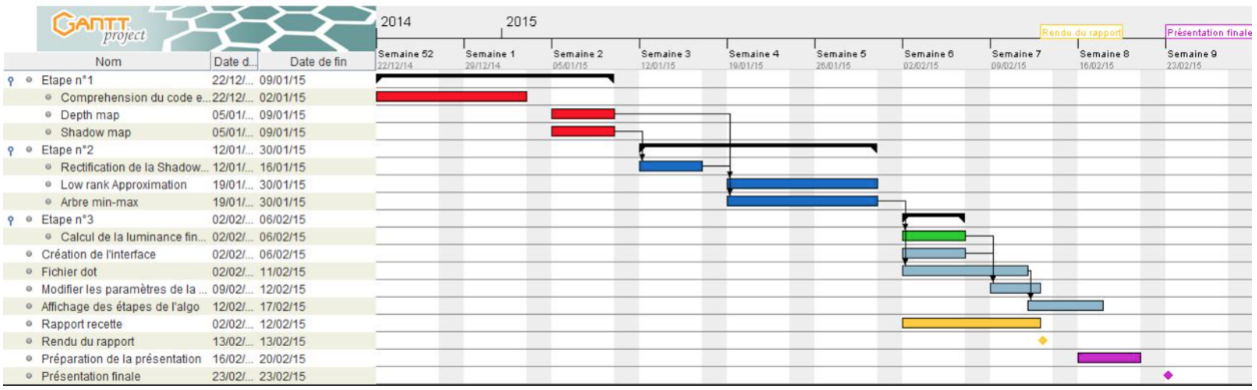


FIGURE 7 – Planning initial

3.2 Planning effectif

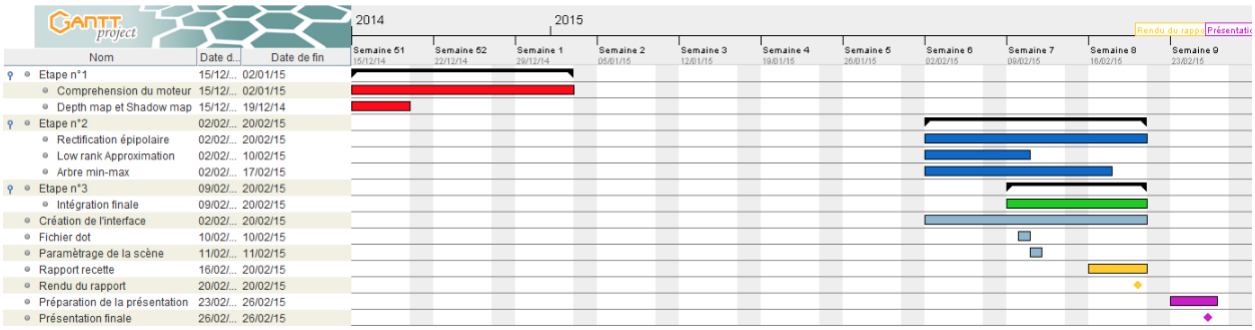


FIGURE 8 – Planning effectif

Lors de l'élaboration de notre chef d'oeuvre, plusieurs rectifications du planning se sont imposées. Un des changement majeur du planning initial a été que les membres du groupe ont travaillés en parallèle sur des tâches distinctes. En effet, lors de l'étude de l'algorithme de notre chef d'oeuvre, nous en avions conclu que chaque tâches principales étaient dépendantes de la précédente et que nous serions dans l'incapacité de travailler en parallèle. Or, la majorité des tâches ont pu être développées en parallèles puis finalement assemblées ensemble afin d'effectuer les liens entre les modules et de tester la bonne fonctionnalité du code final.

Par ailleurs, certaines tâches se sont révélées plus longues à développer que nous l'avions initialement pensé, particulièrement la rectification épipolaire de la *shadow map*.
De plus, le temps de développement n'a débuté qu'en février puisque nous nous sommes consacré aux derniers cours et examens de janvier. Ainsi, nous avons pris du retard par rapport au date du planning initial, mais nous avons pu gagner du temps sur les tâches secondaires, comme la création des fichiers ".dot", l'interface ou encore la visualisation des étapes de l'algorithme qui ont été bien plus rapides à coder que prévu.

Certaines fonctionnalités ont aussi été développés plus rapidement que

prévu, notamment la *low-rank approximation*. Dans les précédents rapports nous avions dit que nous utiliserions la libraire *Eigen* uniquement en cas de difficulté. Cependant nous avons pris en compte les remarques qui nous ont été faite lors des soutenances et nous avons fait le choix d'utiliser directement la librarie pour le calcul de la *SVD*. Nous avions entendu dire qu'*Eigen* était très lent pour l'évaluation de la *SVD*. Cela s'est confirmé dans le cas d'une compilation en mode "Debug", l'evaluation de la SVD mettant plus de 100 ms. Cependant, en mode "Release" la *SVD* ne met que quelques millisecondes (assez lent mais suffisant pour respecter notre cahier des charges). Nous avons donc décider de l'intégrer à notre projet.

Afin de faire le bilan de notre organisation temporel, nous avons mésestimés l'intervalle de temps de développement total, cependant la parallélisation des tâches nous permet tout de même de respecter la deadline finale. Notre organisation était sur le long terme érronée mais nos estimations concernant le développement des tâches était assez proche de la réalité.

3.3 Problèmes rencontrés

Le problème majeur qui s'est présenté lors du développement du projet a été la rectification épipolaire de la *shadow map*. Comme nous l'avions prévu dans les précédents rapports, ce module a été le plus difficile à effectuer. La difficulté de cette étape réside notamment dans le fait qu'il n'est pas possible de tester sa validité, aussi bien de façon visuelle qu'avec des tests unitaires. Si les autre modules permettant l'évaluation de l'éclairage finale fonctionne correctement mais que le résultat ne correspond pas à celui attendu, nous pouvons en déduire que le problème réside dans la réctification de la *shadow map*. Cependant, il est difficile de déterminer simplement d'où peut venir le dysfonctionnement.

Bien que représentant un gain de temps au niveau du développement (pas besoin de développer de nouveau une fonction de *SVD* existante), l'utilisation de la librairie *Eigen*, nous a également fait perdre du temps. Tout d'abord, comme mentionné précédemment, en "Debug" l'évaluation de la *SVD* était très lente, ce qui rendait le *debuggage* très difficile (avec des frames de plus de 120 ms). De plus la librairie *Eigen* est une librairie implémentée uniquement avec des *headers*. Chaque classe étant 100 % modulables (template C++) les temps de compilation du projet en ont été considérablement grandis (plu-

sieurs secondes avec *Eigen* à la place d'une petite seconde avant l'utilisation de la librairie).

Un problème que nous n'avions jamais eu l'occasion de rencontrer auparavant est également survenu. L'évaluation finale de la luminance utilise une boucle *while* dans un *shader*. Dans certains cas, quand les données étaient fausses (rectification épipolaires erronées par exemple) on pouvait faire face à une boucle infini dans le *shader*. Cette boucle infinie se traduisait alors en *freeze* complet du système, il fallait alors redémarrer la machine !

La grande difficulté de ce projet était finalement de ne pas pouvoir toujours tester le bon fonctionnement de nos modules de façon totalement fiable. Cela a eu pour conséquence de nous retarder par rapport au planning prévu.

4 Conception finale

Dans le rapport de conception, un diagramme de classe avait été élaboré (cf *Annexe A*). Dans la version finale de notre projet, ce diagramme de classe est globalement le même. On note simplement certains changements de noms de variables ou de fonction qui ont été renommés. Certaines modifications un peu plus importantes (ajout de fonctions, suppression de classes, ...) ont été apportées pour des raisons technique (réutilisation du code par exemple). Le diagramme de classe suivant permet de visualiser l'architecture finale de notre chef d'oeuvre.

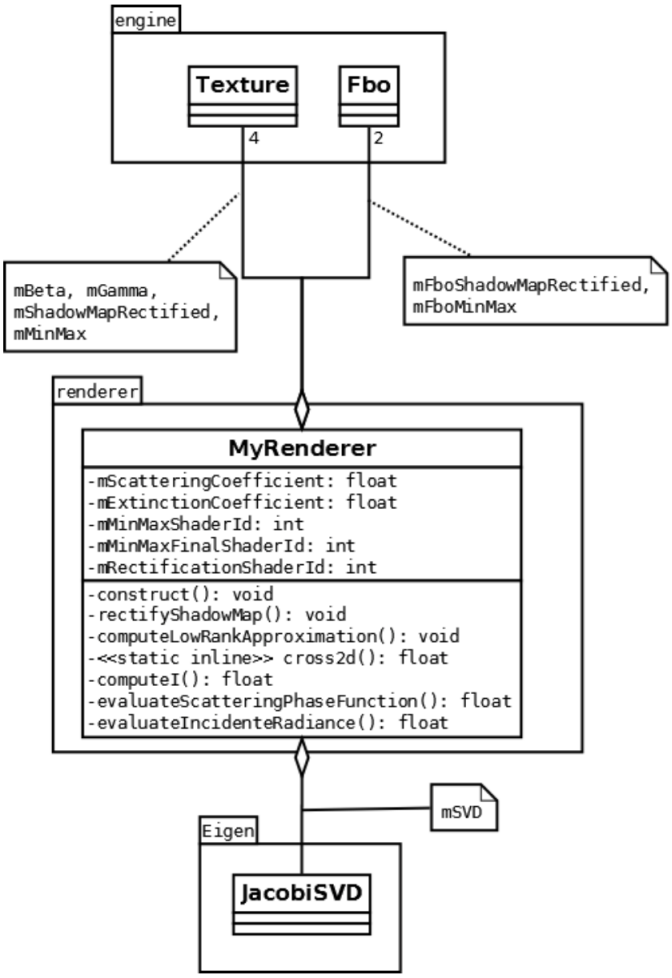


FIGURE 9 – Diagramme de classe effectif

Comme mentionné dans le précédent rapport, nous ajoutons nos contributions à la classe existante *MyRenderer* afin que le calcul luminance en présence de milieu participant soit pris en compte. Les méthodes *construct* et *rectifyShadowMap* ont pour but de paramétrer les *shaders* afin d'appliquer la rectification épipolaire et la construction de l'arbre *minmax*.
L'approximation de la luminance est faites sur CPU. Elle est principalement calculée dans *computeLowRankApproximation*. Les dernières fonctions contribue aussi à cette approximation.
Le package *utils*, dans lequel nous avions prévu de mettre les classes que nous

aurions pu réutiliser a été supprimé. En effet, comme nous l'avons dit, pour la *SVD* nous avons utilisé la librarie *Eigen* au lieu de le réimplementer. La construction de l'arbre, a finalement été développée sur GPU.

Comme prévu dans les précédents rapports, aucune modification majeures n'a été effectué en dehors de la classe *Renderer*. Nous avons simplement du modifier légèrement quelques classes du package *Engine*. Ces modifications sont principalement des corrections de bugs (position de la caméra mal initialisée, non-gestion des textures 1D, ...).

5 Résultats

5.1 Objectifs atteints

Lors de l'écriture du rapport, la plupart des fonctionnalités du projet fonctionne correctement. Cependant, certains détails doivent encore être réglés.

A l'heure actuelle, les objectifs atteints sont les suivants :

- Construction de la *depth map* ✓
- Construction de la *shadow map* ✓
- Décomposition en éléments simple de l'intégrale principale ✓
- Construction des arbres *minmax* ✓
- Génération des fichiers *dot* ✓
- Visualisation des étapes de l'algorithme ✓
- Réglage interactif des paramètres ✓

Il reste à valider les modules suivants :

- Rectification épipolaire de la *shadow map*
- Rassemblement de tous les termes afin d'évaluer la luminance finale

5.2 Tests

Le sujet de notre chef d'oeuvre étant un projet de "rendu visuel", nous avons été dans l'impossibilité de valider toutes nos fonctionnalités avec des tests unitaires.
Lorsque cela a été possible, nous avons développés des fonctions de test permettant de vérifier au maximum la validité des fonctions.

- Décomposition en éléments simples :
  Il est possible de vérifier le bon fonctionnement de la décomposition en éléments simples en lançant la méthode sur des formules dont le résultat est préalablement connu. En effectuant cette opération plusieurs fois, et en obtenant alors les résultats désirés, il est possible d'en conclure que la fonctionnalité fonctionne correctement.

- Arbres min-max :
  La validation des arbres min-max peut s'effectuer grâce à leur visualisation par les fichiers ".png" généré par les fichiers ".dot". Il est alors aisé de vérifier ou non le bon fonctionnement des arbres.

— Calcul de l'éclairage finale de la scène :
Pour l'évaluation du calcul final de la luminance, il n'est pas possible d'effectuer de test unitaires. La seule façon de conclure le bon fonctionnement du module est d'être absolument sûr que les modules précédents fonctionne correctement. En effet, si la décomposition en éléments simples ainsi que la structure permettant de connaître la visibilité (arbre min-max) sont validés, alors il ne suffit que de les assembler afin de connaître la luminance finale. Par ailleurs, il est également possible d'effectuer une validation visuelle (qui sera validée à la fin par les clients) du résultat final.

— Fichier dot :
Afin de valider le bon fonctionnement des fichiers ".dot", une fonction a été développée sur un petit nombre de données fournies au programme sous forme d'un *vector*. Connaissant alors exactement le résultat, il s'agit simplement de visualiser l'arbre généré par le fichier ".dot" et de vérifier le bon fonctionnement de l'arbre *minmax*.

5.3 Livrables

Une dernière réunion du groupe avec les clients est prévu, durant laquelle les documents suivants seront fournis au client :

— code source complet
— manuel d'utilisateur
— site web vitrine
— rapports effectués au long du projet

Une démonstration sera effectuée durant cette recette afin que le client valide le résultat. La recette permettra aussi d'expliquer comment utiliser le logiciel (de manière plus interactive que le manuel d'utilisateur). Des explications détaillées sur le fonctionnement interne de notre logiciel pourront aussi être données.

A ANNEXE - Diagramme de classe initial

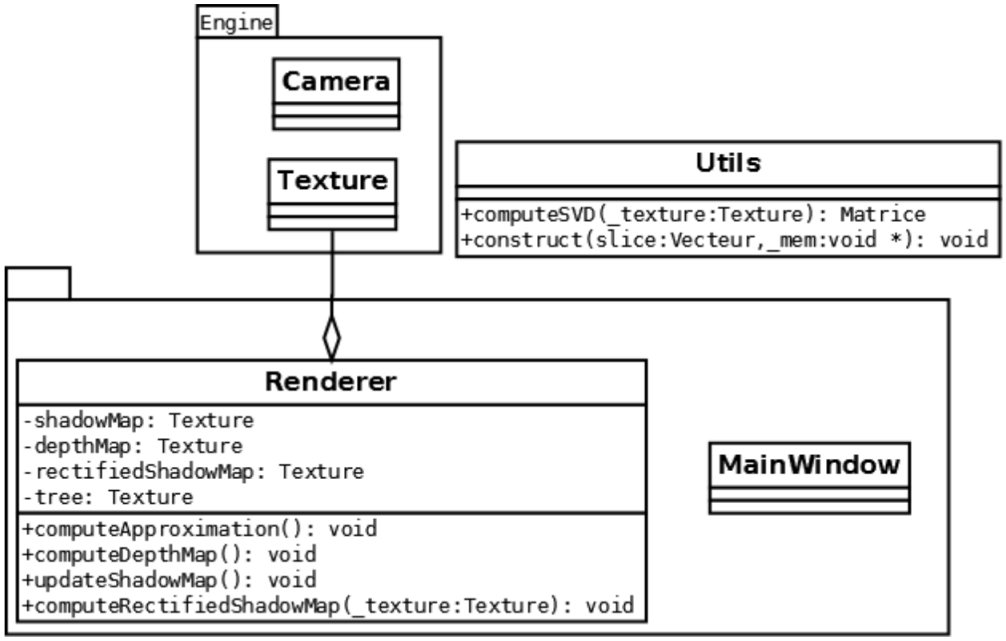


FIGURE 10 – Diagramme de classe initial