BRDFExplorer

Projet de fin d'étude
proposé et encadré par M. Bénard et M. Pacanowksi

Émile Barjou, Christophe Caubet, Rémy Maugey, Maxime Valleron

Février 2019

1 Introduction

BRDFExplorer est un logiciel gratuit et open-source proposé par Walt Disney Animation Studio. Il permet de visualiser et analyser des fonctions de BRDF à l'aide de différents rendus, de graphes et de schémas.

La partie qui nous intéresse dans ce projet est le rendu 3D de la BRDF dans une scène comportant un objet et une carte d'environnement, qui permet d'avoir un visuel du matériau que produit la fonction de BRDF chargée.

Le rendu est fait en lancer de rayon, et plusieurs modes de rendus sont déjà proposés : sans carte d'environnement, avec échantillonage uniforme de la sphère et avec échantillonage préférentiel de la carte d'environnement.

Dans le cadre de notre projet de fin d'études, M. Bénard et M. Pacanowski nous ont proposé d'apporter des modifications à ce logiciel. Les modifications attendues sont principalement l'ajout de trois modes de rendu :

- Le rendu par convolution : Il consiste, au lieu d'échantillonner la carte d'environnement, à prendre en compte toutes les contributions possibles pour obtenir un rendu sans bruit.
- Le rendu par échantillonage préférentiel de la BRDF permet, en donnant avec sa BRDF une fonction d'échantillonage, de choisir des échantillons plus significatifs pour la BRDF, ce qui accélère la convergence du résultat.
- Le rendu par échantillonage préférentiel multiple : Cela consiste à utiliser l'échantillonnage préférentiel de la carte d'environnement et de la BRDF, qui permet de mieux converger lorsque la BRDF ou la carte d'environnement ont des échantillonnages très différents.

D'autres améliorations du logiciel de base sont prévues, comme le chargement de carte d'environnement au format longitude/latitude.

2 Existant

BRDF Explorer est une application qui permet de développer et d'analyser des fonctions de distribution de réflectance bidirectionnelles (*Bidirectional Reflectance Distribution Functions* ou BRDF), initialement développé par Walt Disney Animation Studios (voir 1).

Il peut charger et effectuer un rendu sur un objet 3D des fonctions BRDF analytiques écrites dans le langage de *shaders* GLSL d'OpenGL, des données de matériau mesurées de la base de données MERL et des données de matériau mesurées anisotropes de MIT CSAIL.

Les graphiques et les visualisations se mettent à jour en temps réel à mesure que les paramètres changent, ce qui en fait un outil utile pour évaluer et comprendre les différentes BRDF.

Il propose notamment un mode d'affichage *Image-Based Ligthing* (IBL) qui consiste à éclairer un objet 3D, choisi par l'utilisateur, à l'aide d'une carte d'environnement.

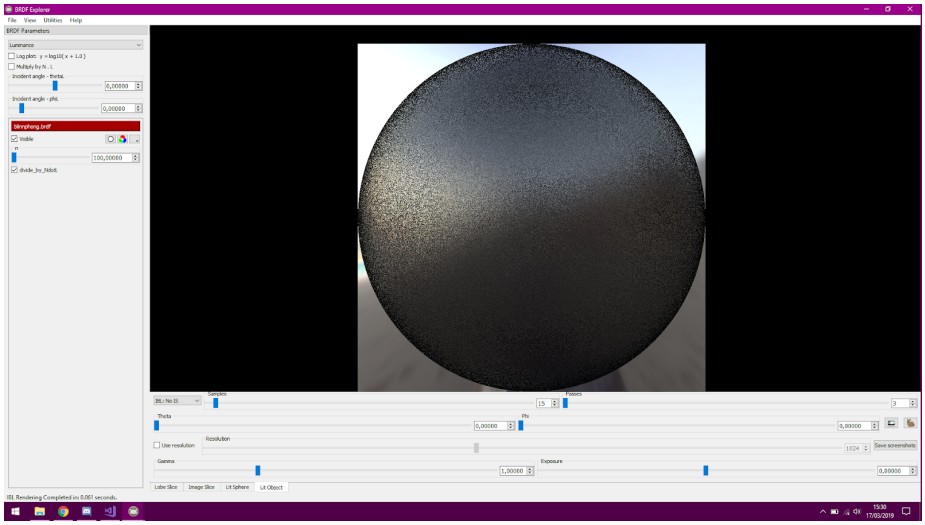


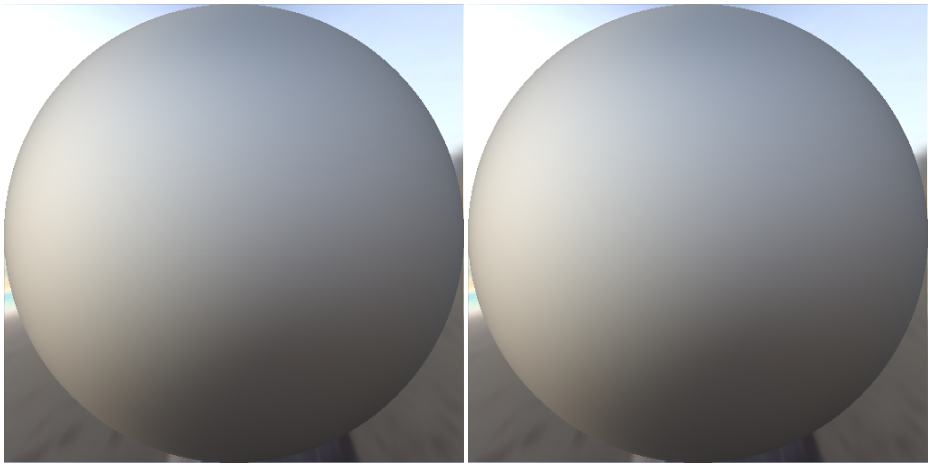
Figure 1: Interface de BRDFExplorer

Il propose notamment un mode d'affichage *Image-Based Ligthing* (IBL) qui consiste à éclairer un objet 3D, choisi par l'utilisateur, à l'aide d'une carte d'environnement.

Ce calcul d'éclairage est effectué par intégration, avec l'estimateur de Monte-Carlo(2c), de la radiance émise par la carte d'environnement et reçue en chaque point de l'objet pondérée par la BRDF. La vitesse de convergence de ce calcul peut être accélérée par l'échantillonnage préférentiel (*Importance Sampling* ou IS) de la carte d'environnement permettant de réduire le temps de rendu

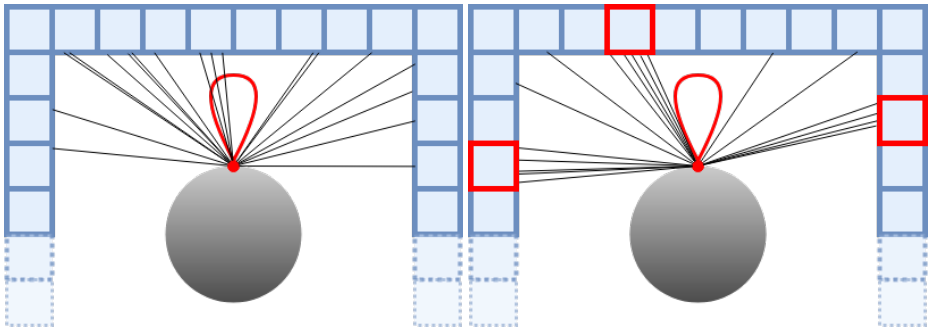
nécessaire pour obtenir une qualité visuelle équivalente. Cette méthode d'échantillonage préférentiel consiste à échantillonner principalement les points les plus lumineux de la carte d'environnement(2d , représentés en rouge sur le schéma). En effet, ce sont ces points qui ont la plus grande contribution à la radiance du pixel à calculer.

Cette méthode de visualisation peut être améliorée en proposant de nouvelles façons d'échantillonner la BRDF afin de réduire le nombre de passes et d'échantillons nécessaires pour avoir un rendu convergé. Cette amélioration se synthétise donc par l'ajout de nouvelles fonctionnalités.



(a) Rendu de modifiedphong.brdf avec échantillonage uniforme, en 300 passes de 100 échantillons

(b) Avec échantillonage préférentiel de la carte d'environnement, en 300 passes de 100 échantillons



(c) Échantillonage uniforme aléatoire

(d) Éch. de la carte d'environnement

Figure 2: Illustration de l'échantillonnage uniforme aléatoire(a) et préférentiel de la carte d'environnement(b). Sur ces schémas, les carrés représentent les pixels de la carte d'environnement, le lobe est celui de la BRDF chargée, et les traits représentent les échantillons tirés pour pondérer la carte d'environnement.

3 Cahier des charges

3.1 Cas d'utilisation

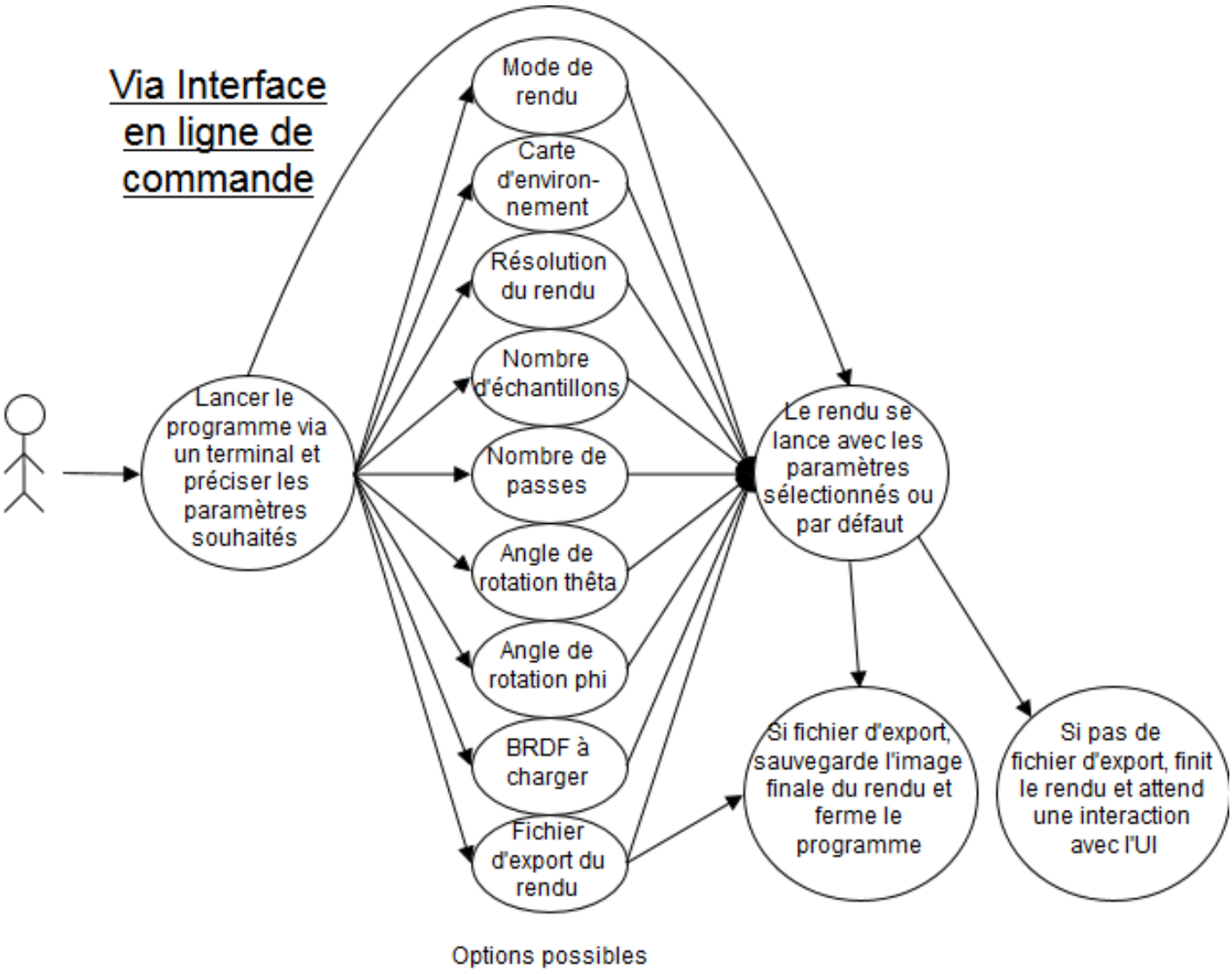


Figure 3: Cas d'utilisation du programme et sélection des paramètres en ligne de commande

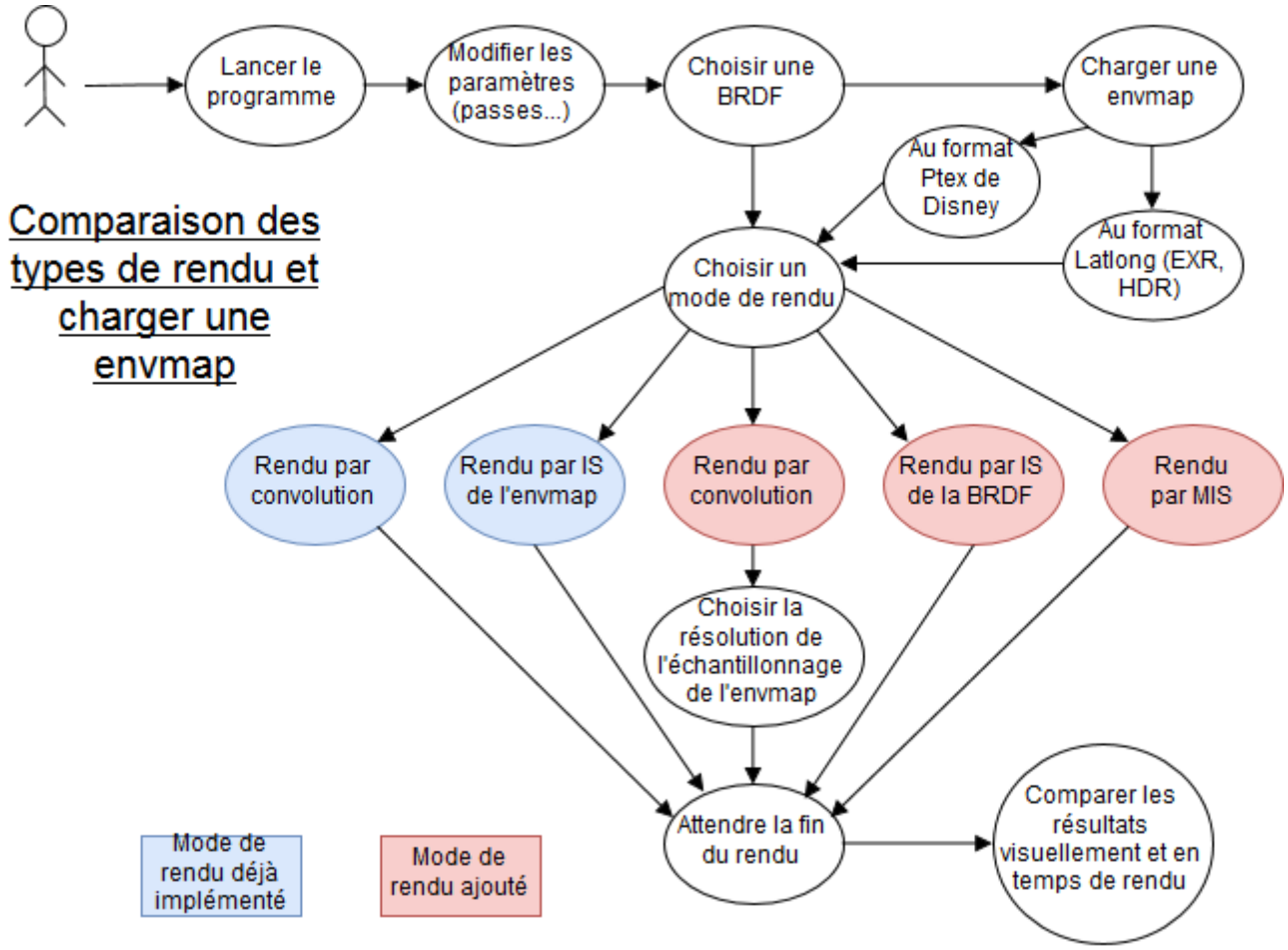


Figure 4: Sélection des modes de rendu, d'une envmap (carte d'environnement) et comparaison des résultats

3.2 Besoins

3.2.1 Besoins fonctionnels:

Les besoins sont rangés par ordre de priorité décroissante. Les besoins décrits comme optionnels sont des améliorations à faire après le reste des besoins.

- Ne pas changer changer l'interface existante, uniquement rajouter des fonctionnalités
- Ajouter l'option de rendu "Convolution" dans l'onglet "Lit Object" de l'interface graphique

- Faire la convolution de la partie visible de la carte d'environnement par la BRDF en plusieurs passes:
  - Effectuer le rendu sur GPU en le découpant sur un nombre de passes élevé pour éviter les *timeouts* du driver
  - Afficher le résultat de chaque passe progressivement dans le *widget*
  - Échantillonner les texels de manière linéaire pour chaque passe
  - Optionnel : parcourir les texels triés par luminance pour d'abord échantillonner les texels qui contribuent le plus à l'image (converge plus vite vers l'image finale)
  - Optionnel : calculer le nombre de passes minimum requis en faisant une passe sur un nombre réduit de texels pour la BRDF donnée et en mesurant le temps d'exécution
- Ajouter le support des formats EXR et HDR pour charger la carte d'environnement
- Charger la carte d'environnement en texture 2D en plus de la texture cubique déjà existante
- Convertir les cartes d'environnement 2D panoramiques vers *cubemap*, en déterminant automatiquement la résolution cible à partir de la résolution d'entrée
- Ajouter une option d'échantillonage préférentiel (*importance sampling* ou IS) de la brdf "IS BRDF" aux méthodes de rendu
- Charger la fonction d'échantillonage préférentiel depuis les fichiers de BRDF
- Ajouter une fonction d'échantillonage préférentiel par défaut à utiliser lorsque l'interface des fichiers de brdf chargés n'en contient pas
- Les fonctions d'échantillonage préférentiel doivent fournir la direction d'échantillonage ainsi que la PDF (*Probability density function*)
- Ajouter une option de multi échantillonage préférentiel "MIS" aux méthodes de rendu
- Implémenter le multi échantillonage préférentiel en combinant les deux modes d'échantillonage préférentiel : L'IS de la BRDF + l'IS de la carte d'environnement ("IBL IS") :
  - Dans un premier temps : Utiliser le même poids pour les deux IS (équivalent)
  - Optionnel : Déterminer dynamiquement les heuristiques à utiliser pour pondérer les deux IS

3.2.2 Besoins non fonctionnels

- Précision du résultat : Résultat obtenu avec moins de X% de différence avec une image convergée en mode sans IS, avec 512 échantillons, 1000 passes, comparé à l'aide d'ImageMagic
- Vitesse de calcul : Calcul du MIS moins de 2 fois plus long que celui de l'IS
- Vitesse de chargement : chargement de la carte d'environnement en moins de 5s pour une carte d'environnement de 2k*1k pixels (res 2k) en lat long sur un PC du CREMI
- Si la conversion de 2D vers *cubemap* est trop longue sur CPU (*parallélisé*), l'implémenter sur GPU

3.3 Contraintes

Les plateformes cibles du projet sont et doivent rester Windows, Mac et Linux. Afin de garantir cela le projet utilise OpenGL 4.1, qui est la dernière version d'OpenGL disponible sous Mac.

Aussi, le projet met à disposition deux systèmes de compilation :

- QMake mis à disposition par Disney à l'origine
- CMake ajouté par l'équipe de l'INRIA

Nous devons nous assurer de garder la compatibilité avec ces deux systèmes de compilation.

3.3.1 Tests

Deux types de tests seront à mettre en place :

- Des tests de non régression: Les modes de rendus initiaux seront testés contre une version du logiciel sans nos modifications
- Des tests de validation : Le mode de rendu par échantillonage uniforme sera utilisé comme référence pour générer des images de tests avec différentes cartes d'environnement et sous différents angles d'orientation

Les comparaisons d'images de référence se feront en prenant en compte la moyenne des erreurs au carré de chaque pixel. Nous avons choisi cette méthode pour avoir une valeur unique permettant de connaître l'écart entre deux images, ce qui semble représentatif pour nous et nos clients. Les images seront comparées au format EXR, utilisant des nombres flottants pour les couleurs, afin de conserver l'amplitude des valeurs de sortie.

Pour permettre l'automatisation de ces tests, des options de lancement seront rajoutées au programme afin de paramétrer le rendu et d'exporter les images finales directement.

BRDFExplorer permet déjà de donner une liste de fichiers de BRDF à charger en paramètre. A cela, les options suivantes seront à rajouter au projet :

`-rendering` <`mode`> L'identifiant du mode de rendu à utiliser

`-ibl` <`chemin`> Le chemin vers la carte d'environnement à charger

`-resolution` <`résolution`> la résolution à utiliser pour le rendu

`-export` <`chemin`> Le chemin et le nom à utiliser pour la capture du rendu, sans l'extension

`-samples` <`échantillons`> Le nombre d'échantillons par pixel

`-passes` <`passes`> Le nombre de passes de rendu

`-theta` <`angle`> L'angle thêta d'orientation de la carte d'environnement

`-phi` <`angle`> L'angle phi d'orientation de la carte d'environnement

3.4 Diagramme de GANTT

On peut constater sur les diagrammes de GANTT prévisionnel(fig. 5) et effectif(fig. 6) de nombreuses différences, en commençant par l'installation d'un environnement de travail qui fut beaucoup plus longue et laborieuse que prévue. L'écriture et l'exécution des tests s'est faite de manière prolongée et a demandé beaucoup plus d'efforts que prévus afin d'être efficace, avançant en même temps que les modifications apportées au programme, et l'implémentation des paramètres via interface en ligne de commande s'est faite assez tôt pour permettre l'exécution de ces tests.

La convolution, premier mode de rendu à implémenter, a commencé plus tôt afin d'avoir de premiers résultats de rendus. Ce fut au détriment des chargement et conversion des cartes d'environnement EXR/HDR, jugés moins prioritaires, qui furent cependant également implémentés plus rapidement que prévu grâce à l'utilisation de la bibliothèque FreeImage, déjà intégrée au programme.

L'intégration de l'échantillonnage préférentiel de la BRDF qui a demandé beaucoup moins d'efforts qu'on pensait, car une partie de l'intégration avait déjà été faite et qu'il nous a suffi de la compléter. L'écriture des fonctions cependant prit un peu plus de temps que prévu et doit encore être complétée dû à la présence de bugs. La convolution "intelligente", moins prioritaire, fut repoussée pour privilégier l'écriture de ces fonctions.

Le calcul du nombre de passe minimal fut simplifié et grandement accéléré. Quant à l'échantillonnage préférentiel combiné, nous y avons passé un peu de temps avant que l'un des clients l'implémente lui-même. A partir de la semaine 6 et particulièrement la semaine 7, nous sommes passés intensivement sur l'écriture du rapport après avoir complété la majeure partie de l'implémentation. La semaine 8 reste une prévision que nous faisons au cours de la semaine 7 et pourrait ne pas s'avérer exacte.

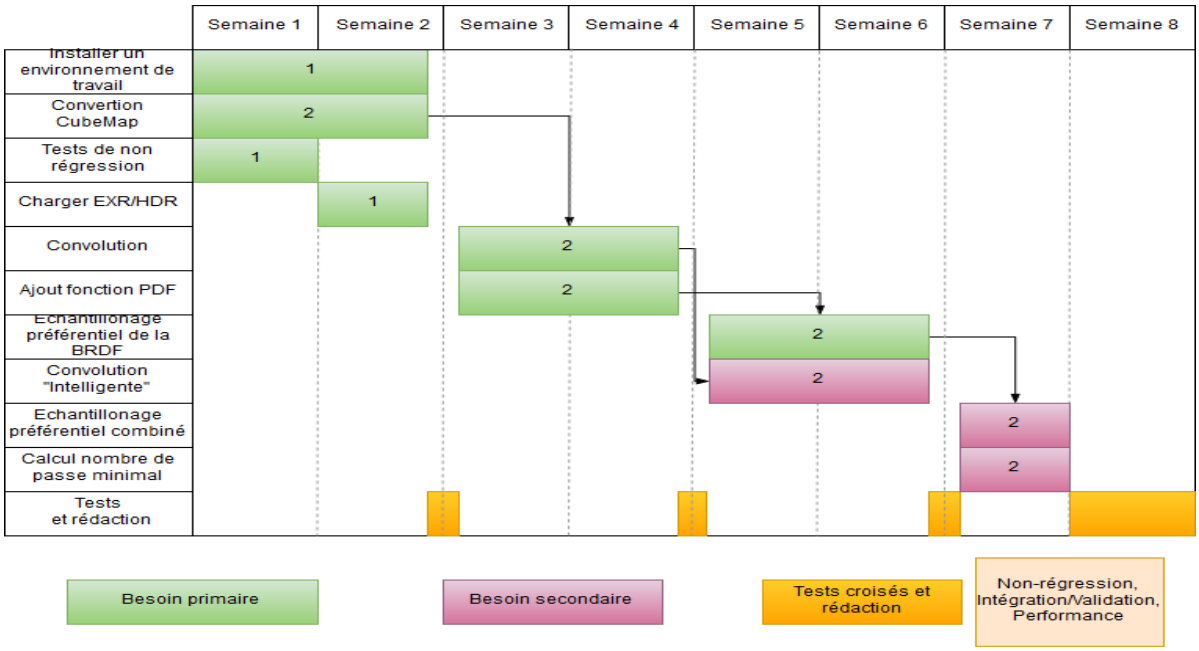


Figure 5: Diagramme de GANTT prévisionnel

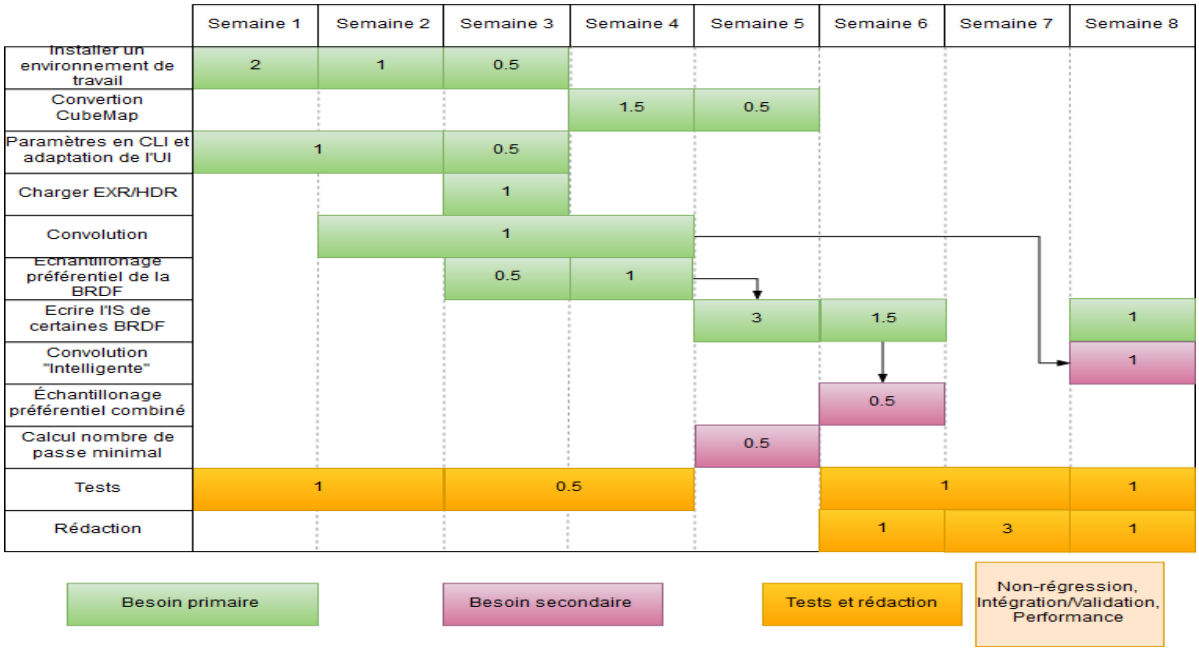


Figure 6: Diagramme de GANTT effectif

4 Architecture

BRDFExplorer utilise une architecture construite autour de celle de l'interface graphique Qt. En effet, la plupart de la logique du logiciel se trouve dans les différentes classes servant à créer les différentes parties de l'interface (voir figure 7).

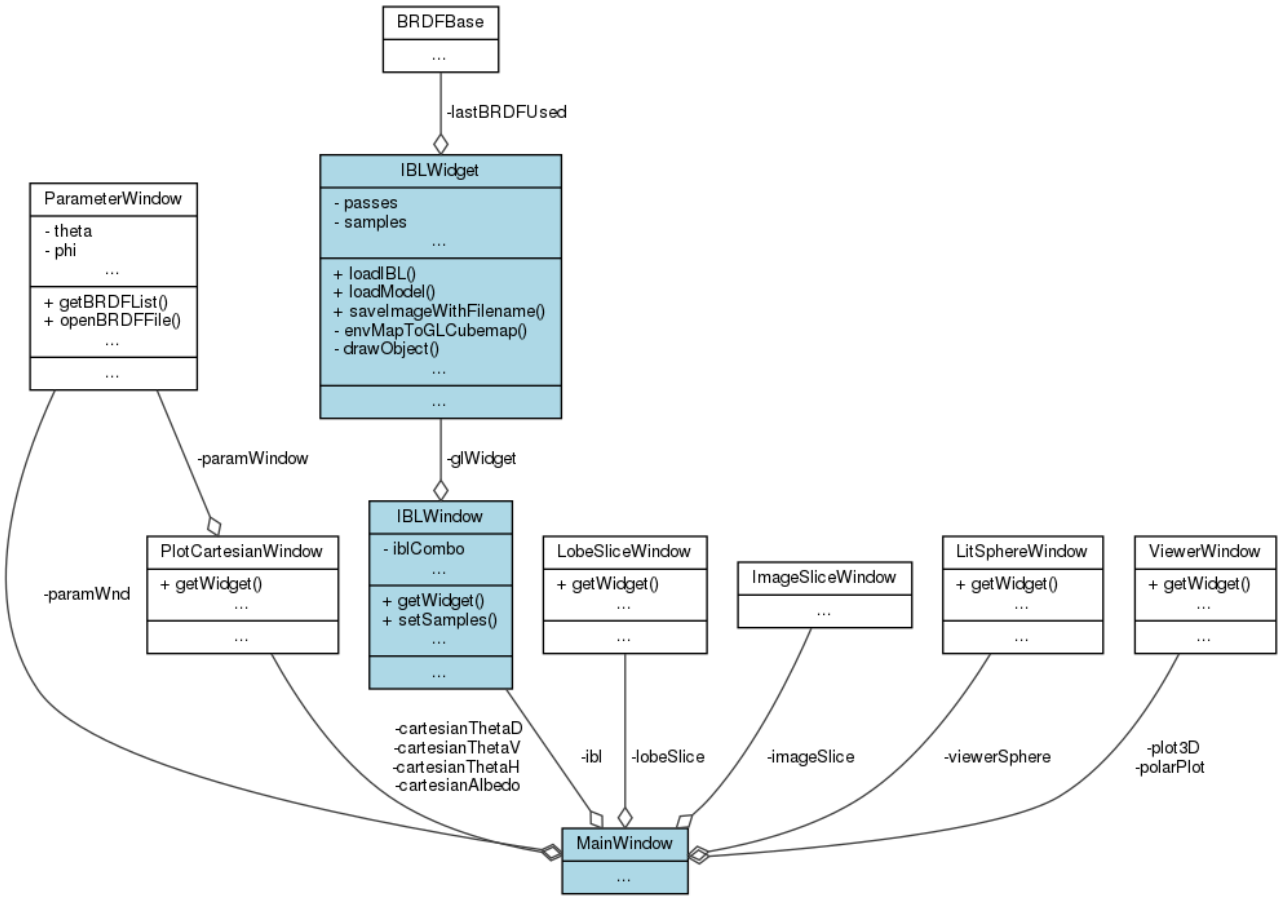


Figure 7: Architecture principale des classes de l'interface graphique (graphe de collaboration de la classe `MainWindow`.

La classe `BRDFBase` sert de parent aux différentes classes permettant de charger les fichiers BRDF (voir figure 8).

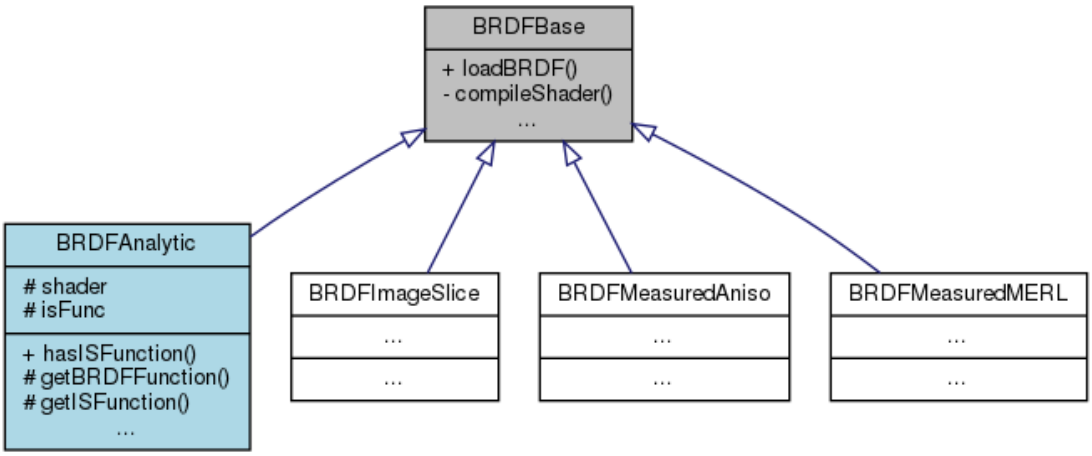


Figure 8: Graphe d'héritage de `BRDFBase`.

Dans les graphes des figures 7 et 8, les classes en bleu sont des classes que nous avons modifié.

`IBLWindow` représente l'onglet "Lit Object" dans l'interface graphique, et contient les *widgets* des réglages des modes de rendu IBL, ainsi que le widget de rendu `IBLWidget`. La classe `IBLWidget` permet de charger le modèle à afficher ainsi que la carte d'environnement.

La classe `BRDFAnalytic` représente un des types de BRDF supporté par BRDFExplorer. Cette classe permet de lire les fichiers de BRDF et d'en extraire ses paramètres, sa fonction de BRDF et sa fonction d'échantillonnage préférentiel si elle existe, ou de donner celle par défaut. Tous les rendus sont effectués côté GPU, à l'aide du *fragment shader* `brdfIBL.frag`.

5 Implémentation

5.1 Lecture des cartes d'environnement au format latitude/longitude

L'objectif de cette partie était de pouvoir utiliser des fichiers aux formats EXR et HDR contenant une image panoramique, et s'en servir comme carte d'environnement. Le programme n'étant prévu que pour charger des cartes d'environnement *cube-map* via des fichiers Ptex de Disney d'extensions ptex, penv ou ptx, il a d'abord fallu séparer les parties nécessaires dans le code en mettant à part la lecture et le traitement des fichiers Ptex.

Nous avons ensuite lu le contenu de l'image avec la bibliothèque FreeImage déjà utilisée dans le projet. La conversion nécessitait plus de travail cependant. Le traitement du Ptex se contentait d'envoyer les données de l'image *cubemap* à un container similaire, dans lequel les six faces sont collées côte-à-côte. Pour

traiter un panorama, il fallait le transformer en *cubemap* pour le donner au container.

Pour ce faire, l'idée est de projeter le panorama comme une sphère et de convertir cette sphère en cube pour obtenir la *cubemap*, et la découper pour la donner au container. Cependant, pour améliorer la qualité du résultat, nous utilisons une transformée inverse. Pour obtenir chaque pixel de la *cubemap*, on prend sa position dans le container, que l'on transforme en position sur le cube, elle-même transformée en position sur le panorama en se servant de ce cube comme d'une sphère d'échantillonage uniforme. On procède finalement à une interpolation bilinéaire sur les pixels les plus proches du point obtenu dans le panorama pour avoir un résultat visuellement plus lisse.

Sur les figures 9 et 10 on peut voir la répartition des faces utilisée pour la conversion. Sur la seconde, l'ordre des faces est celui dans lequel est traitée la *cubemap* lors de l'affichage. Sur ces images il est fait abstraction des déformations géométriques appliquées entre les deux images, mais elles existent bel et bien et sont d'autant plus grandes pour les faces `Haut` et `Bas`. La taille d'une face du cube est le quart de la longueur du panorama.

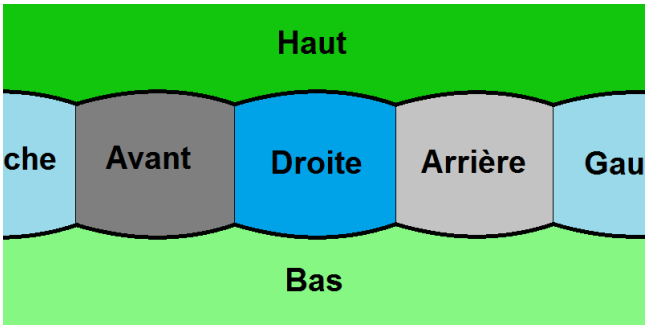


Figure 9: Répartition des faces sur le panorama avant la conversion en *cubemap*.

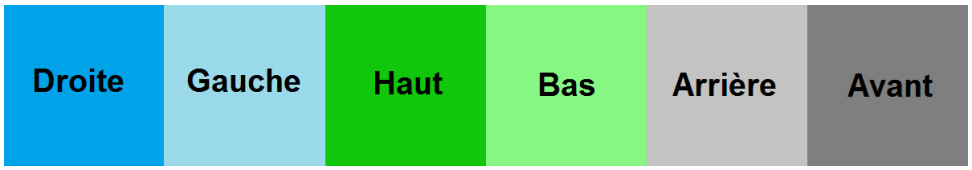


Figure 10: Répartition des faces sur la *cubemap* après la conversion, telles que traitées par le *shader*.

On peut voir cette transformation sur les figures 11 et 12. Sur la première on voit un panorama d'un studio, sur la deuxième la répartition des faces sur la *cubemap* pour l'affichage.

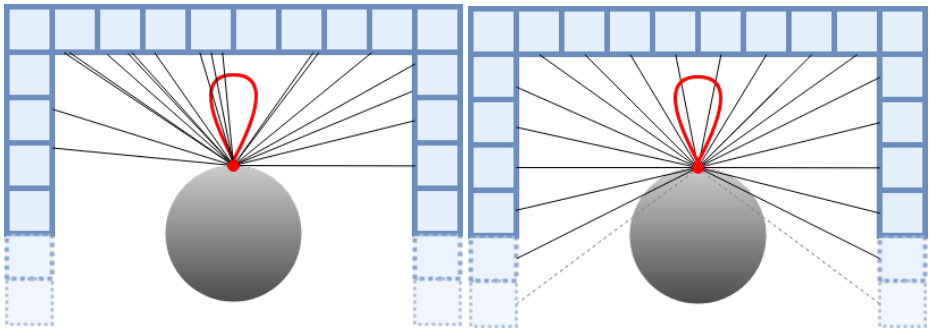
Figure 11: Répartition des faces du studio sur le panorama avant la conversion en *cubemap*. Image de `https://hdrihaven.com/hdri/?h=studio_small_03`

Figure 12: Répartition des faces du studio sur la *cubemap* après la conversion.

5.2 Intégration du rendu par convolution

Le rendu par convolution consiste ici à prendre en compte la participation de chaque pixel de la carte d'environnement (figure 13b), pour chaque pixel du rendu de l'objet, remplaçant l'échantillonnage aléatoire en place jusqu'alors (figure 13a).

Ajouter un nouveau mode de rendu n'a pas posé de problème, le choix pour les modes déjà présents se faisant dans le *shader* à l'aide d'une variable *uniform* déterminant quel mode est utilisé. Nous n'avons donc eu qu'à ajouter une ligne au menu déroulant des modes, et une condition dans le *shader*.



(a) Échantillonage uniforme aléatoire (b) Convolution

Figure 13: Illustration de l'échantillonnage uniforme aléatoire(a) et par convolution de la carte d'environnement(b)

La méthode à laquelle nous avions pensé au début était de partir de la carte d'environnement sous forme 2D et de déduire une direction pour chacun de ses pixels selon les coordonnées, ce qui permet ensuite de faire la somme de chaque pixel multiplié par la BRDF, donnant alors la valeur de la convolution pour un point de vue et une normale donnés.

Comme cela aurait nécessité de stocker la carte en 2D et un traitement supplémentaire pour chaque pixel, nous avons finalement utilisé directement la *cubemap* qui était déjà utilisée pour le rendu.

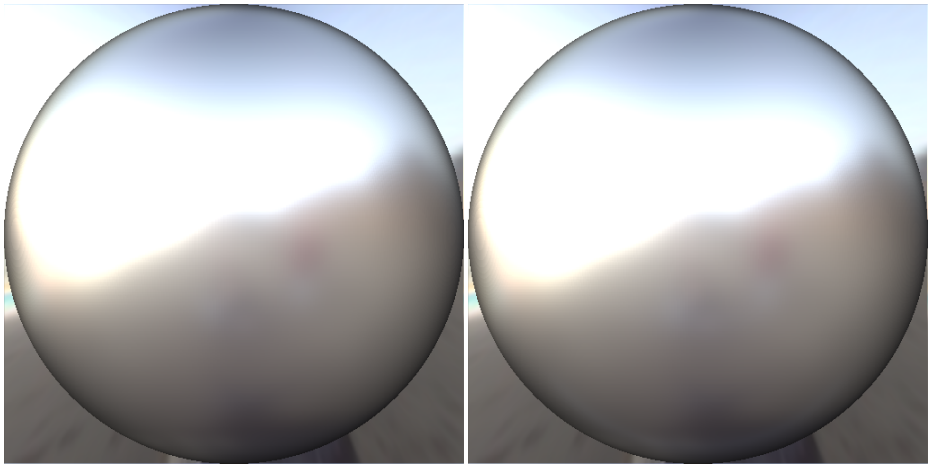
Pour cela, nous déterminons d'abord le nombre de pixels que comporte la carte d'environnement, puis le passons au *shader*. Dans le *shader*, une boucle utilise ce nombre pour reproduire les coordonnées 2D de la carte d'environnement, puis nous les transformons en coordonnées sur la sphère pour aller chercher un pixel de la carte d'environnement dans cette direction.

Une fonction déjà présente assure de passer des coordonnées 2D à celle de la *cubemap* de manière exacte, pour être sûr qu'une direction 3D corresponde bien à un pixel de la carte d'environnement. La définition de convolution par défaut, qui est aussi le minimum, est telle que la boucle tourne autant de fois qu'il y a de pixels dans la *cubemap*.

Afin de mieux gérer les BRDF s'approchant du miroir, nous donnons également la possibilité d'augmenter la définition de la convolution, afin de limiter les cas ou la direction de réflection de la BRDF est entre deux pixels de la carte.

Ce type de rendu pouvant être très long selon la taille de la carte d'environnement (plusieurs minutes), la boucle itérant sur chaque pixel de la carte est divisée en un certain nombre de passes, afin d'éviter un *timeout* du pilote de la carte graphique. Cela permet aussi de ne pas geler le processus trop longtemps, pour ne pas gêner l'utilisateur en rendant la fenêtre inutilisable.

Le nombre de passes par défaut est choisi selon la taille de la *cubemap* crée à partir de la carte d'environnement, de manière à ne pas *timeout* si on charge une carte de très haute résolution, mais sans faire un nombre de passes trop grand dans le cas contraire, ce qui augmenterait inutilement le temps de rendu. Sur la figure 14 on peut voir la comparaison entre la méthode originale d'échantillonnage uniforme et le rendu par convolution. Le rendu est plus convergé sur la convolution mais prend un peu plus de temps (voir partie 6 Tests).



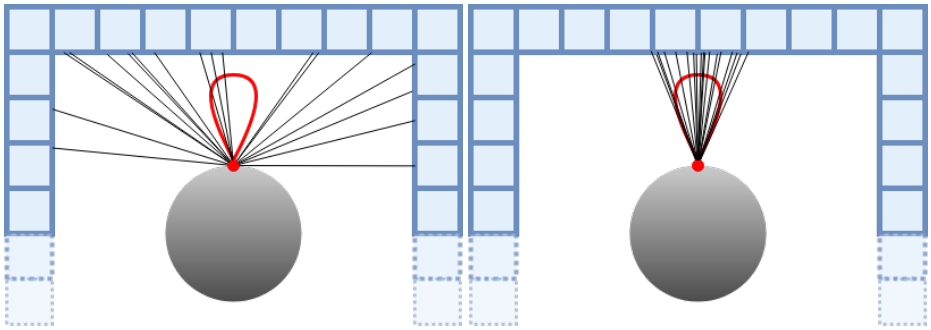
(a) (b)

Figure 14: Comparaison du rendu par échantillonage uniforme (a) (100 échantillons, 300 passes) et par convolution (b)

5.3 Intégration du rendu par échantillonage préférentiel de la BRDF

Contrairement au mode de convolution, le mode de rendu par échantillonage de la BRDF était partiellement implémenté dans le logiciel, ce qui nous a facilité la tâche. La partie sur le code C++ devant lire la fonction d'échantillonage dans le fichier BRDF et l'option d'activer ce mode de rendu était déjà présente, il nous restait donc à implémenter le rendu lui-même.

Cette méthode consiste à tirer des échantillons préférentiellement dans des direction où la BRDF est importante(15b), afin de minimiser les chances qu'un échantillon ne soit pas significatif dans le rendu final(15a).



(a) Échantillonage uniforme aléatoire (b) Échantillonage préférentiel de la BRDF

Figure 15: Illustration de l'échantillonnage uniforme aléatoire(a) et préférentiel de la BRDF(b)

Pour traiter cette nouvelle fonctionnalité dans le *shader*, nous avons modifié la fonction utilisée pour l'échantillonnage déjà présente, afin de prendre directement une direction 3D et non pas un couple uv représentant une coordonnée 2D, ainsi que prendre en argument une probabilité associée à cette direction.

La fonction *sampleBRDF(u, v, normal, tangent, bitangent, wo, out pdf)*, devant être implémenté dans le *shader* de la BRDF, permet de choisir une direction d'échantillonage selon deux nombres aléatoires et nous en donne la PDF (Probability Density Function). Tous les vecteurs que nous manipulons sont en espace caméra.

Un certaine partie de cette implémentation était déjà présente dans le code, les classes BRDFBase et BRDFAnalytics, (qui gèrent l'assemblage des fichiers de BRDF et des *shaders* de rendu) permettant déjà d'ajouter une fonction d'échantillonage préférentiel, mais qui n'étaient utilisées nulle part. BRDFBase sera modifié pour y ajouter l'implémentation de la fonction d'échantillonage par défaut.

Afin de tester notre implémentation de l'échantillonnage préférentiel de la BRDF, nous avons décidé de l'ajouter à trois BRDF types de BRDF : une isotrope, une anisotrope et une à micro-facette.

5.3.1 Le modèle isotropique de Blinn-Phong

Le modèle de Blinn-Phong est une BRDF isotrope calculée à partir d'une composante spéculaire et d'une composante diffuse. Pour implémenter un exemple d'échantillonnage préférentiel de ce type de BRDF, nous nous sommes servi de l'implémentation déjà présente dans le projet de la partie spéculaire d'une

version modifiée de ce modèle (fichier de BRDF `modifiedphong.brdf`). Nous nous sommes basé sur le papier de Lafortune [1] pour implémenter la fonction d'échantillonnage. Pour ξ_1 et ξ_2 deux variables aléatoire uniforme entre 0 et 1, la direction d'échantillonage est donnée telle que :

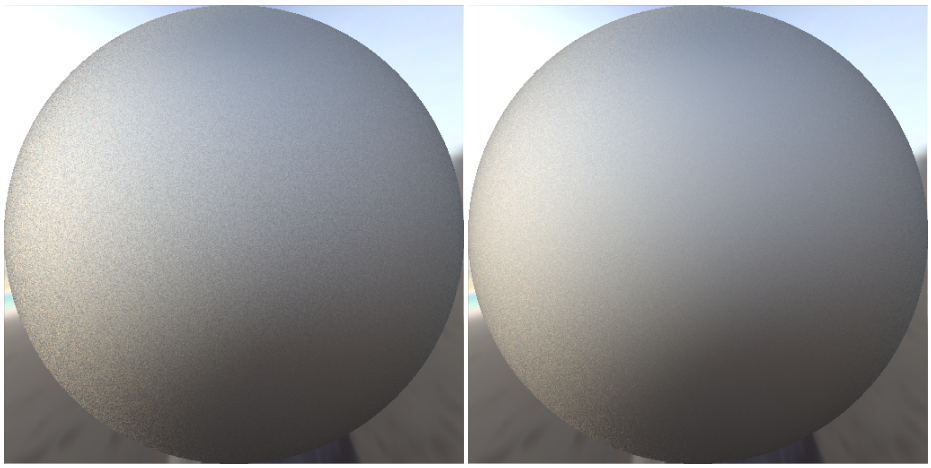
$$(\alpha, \phi) = (\arccos \sqrt{\xi_1}, 2\pi\xi_2)$$

La probabilité associée à cette direction est :

$$pdf = \frac{n+1}{2\pi} \cos^n \alpha$$

Avec n le coefficient spéculaire du matériau.

La figure 16 montre que la méthode avec échantillonnage préférentiel converge plus tôt que la version en échantillonnage uniforme.



(a) (b)

Figure 16: Comparaison du rendu sur la BRDF Blinnphong avec 15 échantillons et 5 passes par échantillonage uniforme (a) et par IS de la BRDF (b)

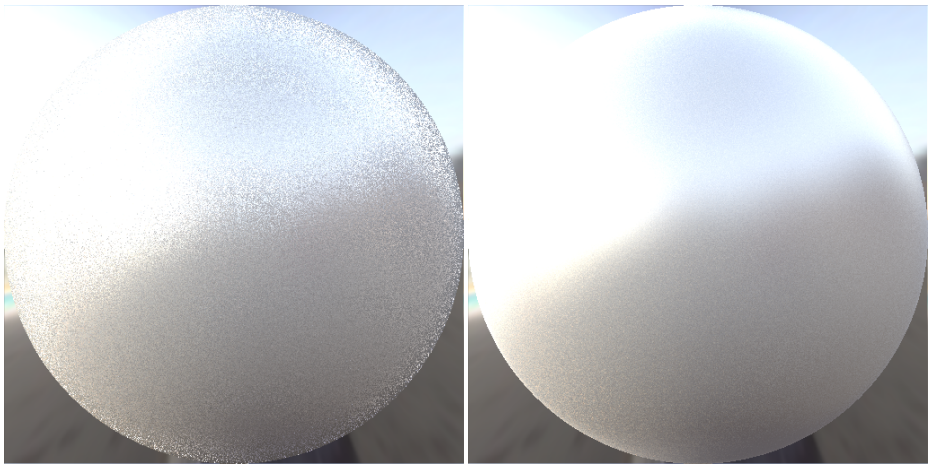
5.3.2 Le modèle anisotropique de Ward

Nous avons également écrit une fonction d'échantillonnage préférentiel BRDF de Ward basée sur son papier de 1992[4] et les notes de Walter[2] sur ce même papier. Pour cela il a fallu traiter les deux lobes de cette BRDF, un diffus et un spéculaire. Il faut également prendre en compte les coefficients αx et αy déterminant l'ouverture de ces lobes.

Lorsque l'on tire un échantillon, on décide avec un aléatoire pondéré par les coefficients diffus et spéculaire quel lobe on échantillonne comme décrit dans Lafortune et Willems 1994[1]. Si c'est le diffus qui est choisi, on échantillonne selon le cosinus comme dans la fonction par défaut. Si c'est le spéculaire, on calcule la direction comme décrit dans les notes de Walter. Dans les deux cas,

on calcule ensuite la pdf en pondérant la pdf diffuse et la pdf spéculaire par le probabilité de les avoir choisis et en combinant les deux résultats.

Sur la figure 17, avec 15 échantillons et 5 passes on peut voir que l'échantillonnage préférentiel de la BRDF sur Ward converge plus rapidement que l'échantillonnage uniforme qui est très bruité.



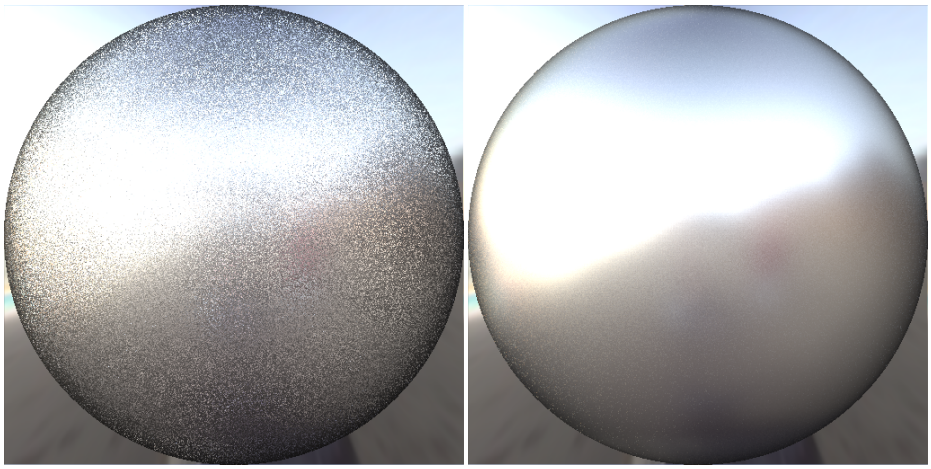
(a) (b)

Figure 17: Comparaison du rendu sur la BRDF Ward avec 15 échantillons et 5 passes par échantillonage uniforme (a) et par IS de la BRDF (b)

5.3.3 Le modèle à micro-facettes GGX de Walter

Pour la BRDF à micro-facette, nous avons décidé de choisir le GGX, car cette méthode est très bien documenté. Nous avons utilisé l'article de Walter[3] qui détaille un méthode d'échantillonage de GGX relativement simple à mettre en place.

Sur la figure 18, avec seulement 15 échantillons et 5 passes on peut voir que l'échantillonnage préférentiel de la BRDF sur Ward converge extrêmement rapidement comparé à l'échantillonnage uniforme qui est encore très bruité.



(a) (b)

Figure 18: Comparaison du rendu sur la BRDF GGX de Walter avec 15 échantillons et 5 passes par échantillonage uniforme (a) et par IS de la BRDF (b)

5.4 Intégration du rendu par échantillonage préférentiel multiple

Le rendu par échantillonage multiple consiste à choisir des échantillons préférentiellement dans deux distributions, ici celle de la BRDF et de la carte d'environnement (20b). Cette méthode à l'avantage de mieux gérer les cas limites des méthodes précédentes, par exemple une BRDF miroir pour l'échantillonnage de la carte d'environnement, ou une carte d'environnement proche du dirac pour l'échantillonnage de la BRDF.

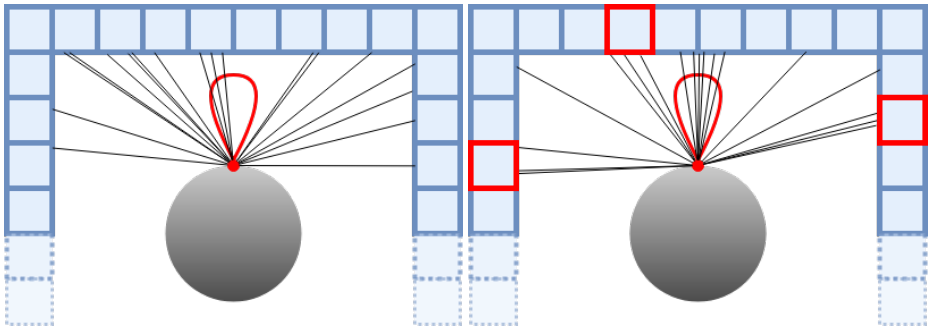
Dans notre première implémentation, nous nous somme contenté de tirer aléatoirement, à une chance sur deux, soit dans la BRDF soit dans la carte d'environnement. Cette méthode fonctionne fonctionne mais l'amélioration n'est pas flagrante par rapport aux précédentes méthodes d'échantillonage préférentiel.

M. Bénard à par la suite implémenté un version plus évolué de type balance heuristique, qui à nécessité un changement plus important du programme, ainsi que l'ajout d'une fonctions au fichiers de BRDF donnant la PDF associé à une direction donnée.

On peut voir sur la figure 19 que sur une carte d'environnement où une grande partie est sombre mais dont certaines petites zones sont très lumineuses, l'échantillonnage multiple donne des résultats très largement supérieurs à ceux de l'échantillonnage uniforme. C'est dû à l'échantillonnage qui vise principalement les zones importantes de la carte d'environnement (fig. 20).

(a) (b)

Figure 19: Comparaison du rendu sur la BRDF GGX de Walter avec 15 échantillons et 5 passes par échantillonage uniforme (a) et par MIS (b)



(a) Échantillonage uniforme aléatoire (b) Échantillonage préférentiel multiple

Figure 20: Illustration de l'échantillonnage uniforme aléatoire(a) et préférentiel multiple(b)

6 Tests

L'objectif de nos tests est de vérifier que les nouvelles méthodes de rendu intégrées au projet donnent un rendu identique (avec une marge d'erreur faible) à une méthode de rendu déjà présente que nous avons choisi comme référence.

Afin que nos tests soient facilement utilisables sur différentes plate-formes avec le minimum de dépendances, nous avons choisi de les écrire en `Python`, et d'utiliser la suite de logiciel `ImageMagick` pour effectuer les comparaisons d'image. Bien qu'il existe des modules `Python` pour utiliser l'API d'`ImageMagick` directement depuis le script, nous avons décidé de ne pas en utiliser. En effet, nous n'avons besoin que d'une seule fonction d'`ImageMagick` (la fonction de

comparaison d'image), et les fichiers à tester étant déjà stockés sur le disque, il est inutile de pouvoir les manipuler depuis le script `Python`. Cela nous permet d'éviter une dépendance, et donc de simplifier le lancement des tests sur n'importe quelle plate-forme, `Python` et `ImageMagick` étant disponibles facilement sur Linux, Mac et Windows.

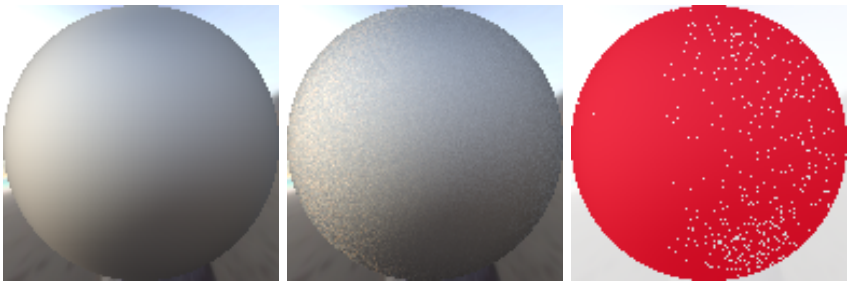
Nous avons aussi rajouté des options de lancement à BRDFExplorer pour paramétrer le rendu depuis la ligne de commande. Ces options permettent de changer le mode de rendu, la résolution, le nombre de passe *etc*, ainsi que d'exporter le rendu terminé aux formats EXR et PNG. L'affichage de la fenêtre est cependant nécessaire pour effectuer le rendu, afin d'obtenir un contexte OpenGL.

Nous avons donc créé un petit module Python proposant des fonctions pour lancer BRDFExplorer et obtenir les images de rendu, ainsi qu'effectuer les comparaisons d'images à l'aide de l'interface en ligne de commande d'`ImageMagick`. Les scripts de test utilisent ce module pour effectuer des comparaisons dans plusieurs configurations différentes. Si toutes les images de test correspondent aux images de référence, le test est validé.

La figure 21 montre un exemple de résultat de test réussi. La figure 22 montre un exemple de test raté. Les sorties de comparaison montrent les pixels qui diffèrent d'une image à l'autre.

(a) Image de référence (b) Image de test (c) Sortie de comparaison

Figure 21: Test réussi: MSE = 0.0139842



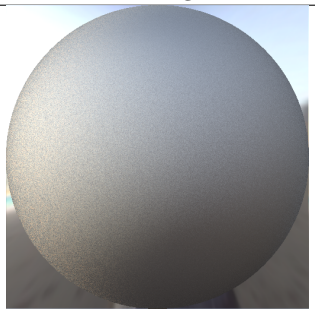
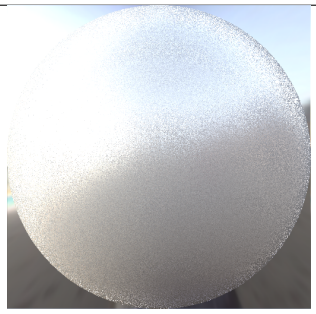
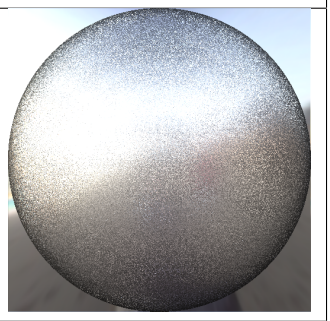
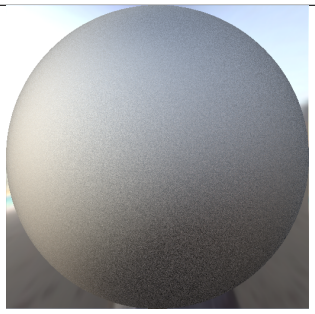
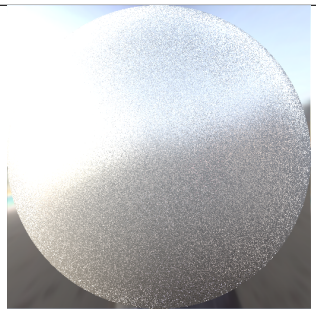
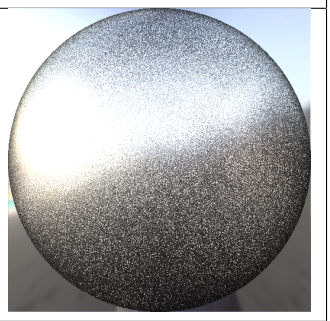
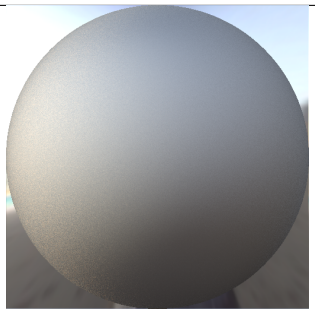
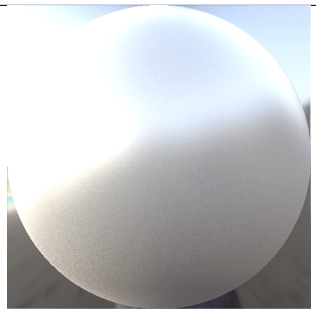
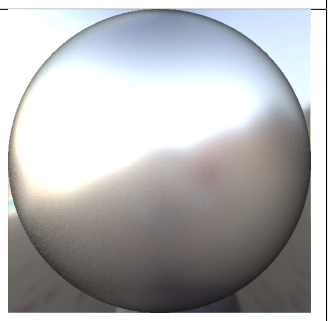
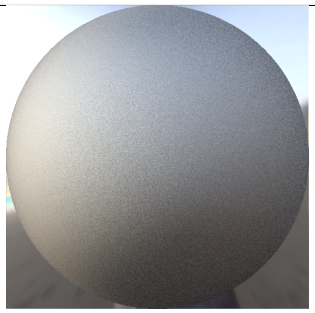
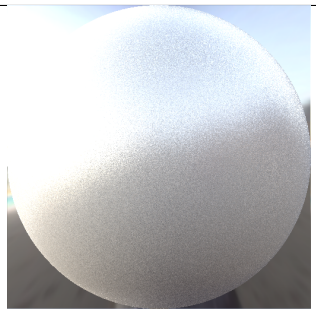
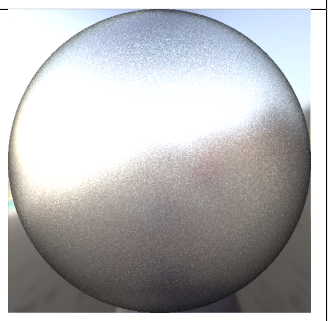
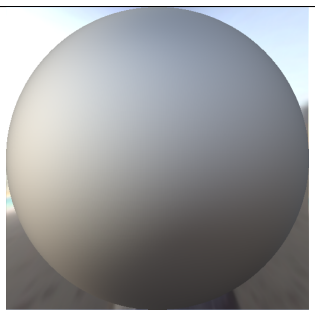
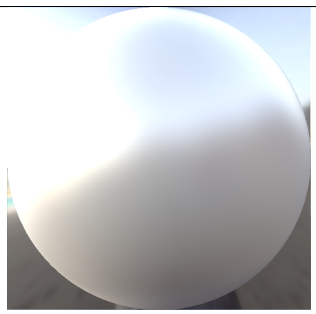
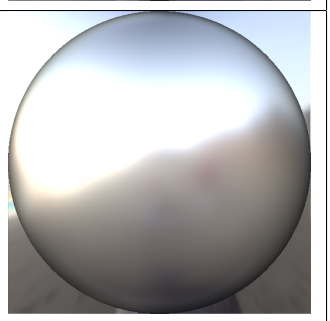
(a) Image de référence (b) Image de test (c) Sortie de comparaison

Figure 22: Test raté: MSE = 23.3012

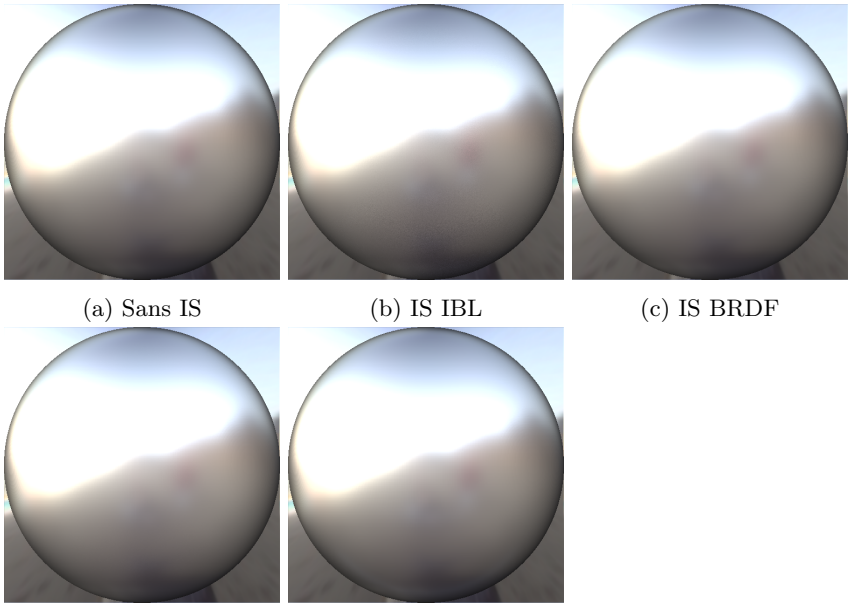
Des tests comme ceux-ci ont été effectués pour tous les modes de rendu que nous avons rajouté. Pour le mode d'échantillonnage préférentiel de la BRDF, nous avons effectué les tests pour la fonction par défaut ainsi que pour chaque BRDF que nous avons modifié.

7 Résultats

Nous allons maintenant vous présenter les résultats que nous avons obtenu à l'aide des mode de rendu du logiciel. Pour commencer, voici un tableau comparatif montrant les résultats pour des rendus de 5 passes et de 15 échantillons (sauf pour la Convolution), sur les BRDF de Phong, Ward et GGX.

	Phong	Ward	GGX
Sans IS			
IS de l'IBL			
IS de la BRDF			
MIS			
Convolution			

Nous pouvons voir sur la figure 23 que les différents modes de rendu, même si leurs vitesses de convergence varient selon les cas, convergent bien vers un même résultat.

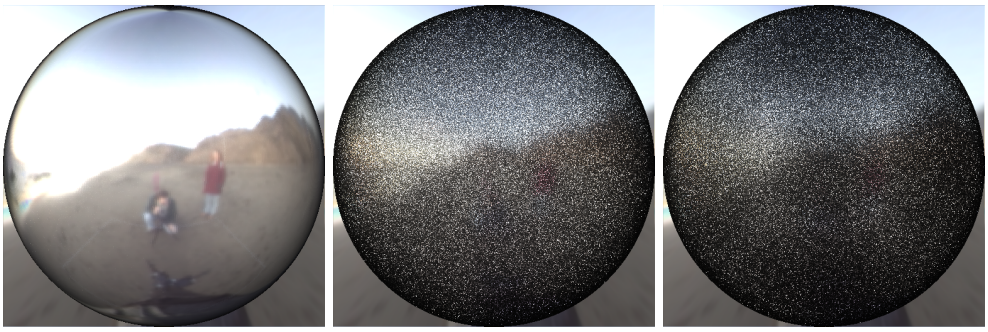


(a) Sans IS (b) IS IBL (c) IS BRDF

(d) MIS (e) Convolution

Figure 23: Convergence des résultats avec 300 passes de 100 échantillons

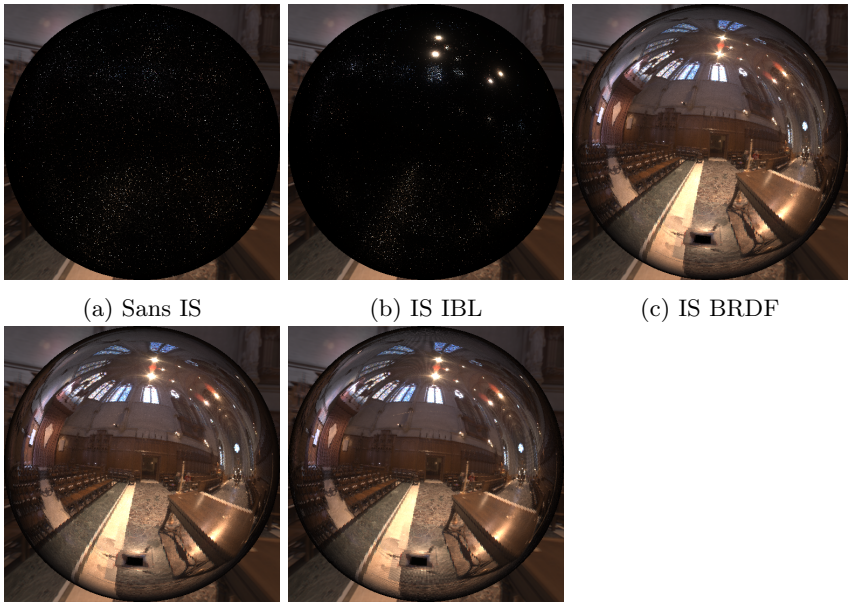
On peut voir en figure 24 que l'échantillonnage préférentiel par défaut, celui pondéré par le cosinus, donne de meilleurs résultats que l'échantillonnage uniforme, mais reste loin du résultat obtenu par une fonction d'échantillonage adaptée à la BRDF.



(a) BRDF IS (b) Pondéré par le Cosinus (c) Sans IS

Figure 24: Comparaison des fonction d'échantillonage de la BRDF par défaut et spécifique, sur GGX avec 15 échantillons et 5 passes.

Dans la figure 25, on peut voir l'intérêt de l'échantillonnage multiple, qui permet d'avoir un bon résultat sur une BRDF miroir, mais aussi de mieux aller chercher les points lumineux de la carte d'environnement que l'échantillonnage de la BRDF uniquement.



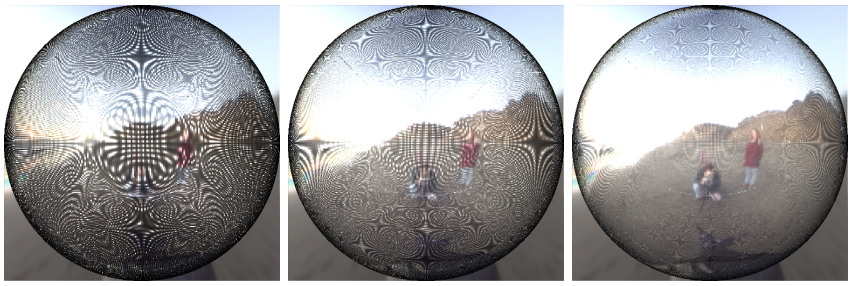
(a) Sans IS (b) IS IBL (c) IS BRDF

(d) MIS (e) Convolution

Figure 25: Rendu dans une carte d'environnement à fort contraste avec une BRDF proche du miroir, sur GGX avec 15 échantillons et 5 passes.

Nous pouvons cependant noter une limite à l'utilisation de la convolution. En

effet, si la BRDF se rapproche du miroir parfait et que la carte d'environnement a une définition trop basse, les pas d'angles de l'échantillonnage seront trop grands par rapport à la BRDF qui n'est supérieure à zéro que sur un très petit angle solide, ce qui cause un effet d'*aliasing* (voir figure 26) empêchant le rendu de converger.



(a) 128*128*6 pixels (b) 256*256*6 pixels (c) 512*512*6 pixels

Figure 26: Problème d'*aliasing* de la convolution

Pour finir, voici les temps de rendu que nous avons relevé pour le rendu d'images de 512*512 pixels :

	Sans IS	IS IBL	IS BRDF	MIS
Phong 15x5	0.029s	0.035s	0.030s	0.043s
Ward 15x5	0.030s	0.035s	0.041s	0.155s
GGX 15x5	0.028s	0.033s	0.031s	0.047s
Phong 100x300	5.04s	6.72s	5.77s	12.9s
Ward 100x300	6.35s	7.95s	12.2s	20.60s
GGX 100x300	5.07s	6.80s	6.63s	13.07s
Convolution (*cubemap* de 128*128*6 pixels)				
Phong	12.7s			
Ward	15.6s			
GGX	12.90s			

Et les temps de rendu pour une convolution avec la BRDF GGX en augmentant la définition d'échantillonage de la carte d'environnement :

Temps de rendu de convolutions	
128*128	28.6s
256*256	61.8s
512*512	218s

Les temps de rendu ont été relevés sur une machine avec un processeur i5-4570 4x3.60Ghz, 8Go de RAM et une carte graphique Radeon RX580 à 1.4Ghz.

8 Conclusion

À la fin de ce projet, nous pensons avoir atteint la majorité des objectifs fixés avec les clients, mais l'implémentation du dernier point demandé,

l'échantillonnage préférentiel multiple, a été finalement donnée par M. Bénard.

Nous avons pu mettre en oeuvre de nombreuses notions vues cette année en cours de synthèse d'image au cours de ce projet, particulièrement sur le lancer de rayon et l'échantillonnage préférentiel.

Au niveau du logiciel, certaines améliorations aurait encore pu être faites, notamment la pré-visualisation de la convolution qui pourrait être améliorée afin de ne pas rendre chaque passe quand celle-ci sont très rapides, afin d'accélérer le temps de rendu.

Le nombre de passes est déterminé de façon arbitraire, et pourrait causer des problèmes sur des cartes d'environnement de grande définition ou sur des cartes graphiques moins performantes, il aurait fallu pouvoir l'adapter selon le temps de rendu d'une passe et le temps de *timeout* du pilote graphique. Passer par des *compute shaders* au lieu des *fragment shaders* pourrait simplifier cette partie et accélérer le temps de calcul, mais la compatibilité avec Mac nous oblige à utiliser OpenGL 4.1 au maximum.

Pour conclure, nous souhaiterions remercier messieurs Bénard et Pacanowski qui nous ont accordé beaucoup de leurs temps durant nos entrevues hebdomadaires, et ont été d'une grande aide pour à mener à bien ce projet.

9 Bibliographie

References

[1] Eric P. Lafortune and Yves D. Willems. Using the modified phong reflectance model for physically based rendering. 1994.

[2] Bruce Walter. Notes on the ward brdf. *Tech.Rep. PCG-05-06, Cornell Program of Computer Graphics*, 2005.

[3] Bruce Walter, Stephen R. Marschner, Hongsong Li, and Kenneth E. Torrance. Microfacet models for refraction through rough surfaces. 2007.

[4] Gregory J. Ward. Measuring and modeling anisotropic reflection. *Computer Graphics (Proceedings of SIGGRAPH 92)*, 26:265–272, 1992.