Chef d'oeuvre - Recette
Approximation d'éclairage indirect en temps réel

Blaise Cardonne Gauthier Bouyjou Valentin Camus
Rihab Elrifai Sylvain Durand

Encadrant/Client : François Desrichard

Février
2020

Contents

1 Contexte / Introduction

Ce document est rédigé dans le cadre de la livraison du logiciel, il va notamment permettre de définir quelles fonctionnalités ont été implémentées ou non. Il vient clore l'UE Chef-d'oeuvre du Master 2 IGAI 2019/2020 mettant en avant nos capacités de développement en équipe autour d'un projet universitaire.

Ce projet a été proposé et encadré par M. François Desrichard actuellement doctorant à l'IRIT (Institut de Recherche en Informatique de Toulouse).
L'équipe GIR chargée du développement est composée de cinq étudiants du Master 2 IGAI : Blaise Cardonne, Gauthier Bouyjou, Valentin Camus, Rihab Elrifai et Sylvain Durand.

L'éclairage indirect d'une scène 3D en temps réel reste un défi majeur, notamment pour les industries du jeu vidéo où la qualité du rendu est primordiale pour rivaliser avec de nombreux concurrents.

Le but de ce projet est d'implémenter l'article de recherche sur les *Reflective Shadow Maps*[1] permettant un éclairage indirect de la scène 3D ainsi que l'utilisation des gaussiennes sphériques citées dans l'article de *Square Enix*[2] permettant l'élimination d'artefacts indésirables (splotch) sur l'image lors du calcul d'un éclairage global. Un troisième article nous a été proposé pour compléter les deux approches précédentes, les *Imperfect Shadow Maps*[3].

Le produit final devra permettre un rendu physiquement réaliste d'une scène en temps réel.

2 Description du projet

2.1 Object global

Notre projet s'est déroulé en deux temps. Dans un premier temps, nous avons créer une base moteur permettant un rendu temps-réel d'une scène 3D précédemment importée selon un format standard, ainsi que d'avoir un éclairage direct des sources de lumières dispersées dans la scène.

Puis dans un deuxième temps nous avons répondu aux problèmes d'éclairage indirect proposé par notre client en implémentant une solution présentée dans un article de recherche, les *Reflective shadow maps*[1].

Il est important pour le client d'avoir une démo utilisable, simple et minimaliste, mettant en évidence au moins une solution d'éclairage indirect.

3 Rappel du cahier des charges

3.1 Exigences fonctionnelles initiales

Le produit fini devra offrir l'ensemble des fonctionnalités suivantes (fonctionnalités optionnelles incluses):

- Le chargement d'une scène 3D
- L'affichage d'une scène 3D via un modèle de BRDF physiquement réaliste
- La gestion de caméra dynamique
- La gestion de sources lumineuses (éventuellement dynamiques)
- Le changement de mode de rendu (notamment pour comparer les différentes méthodes)
- La gestion de l'éclairage indirect selon les méthodes suivantes:
  - *Reflective shadow maps*[1]
  - *Reflective shadow maps*[1] et les *gaussiennes sphériques*[2]
  - *Reflective shadow maps*[1], les *gaussiennes sphériques*[2] et les *Imperfect Shadow Maps*[3]
- La sauvegarde d'images pour comparaison avec des références

3.2 Rappel des scénarios de tests

3.2.1 Validation des fonctionnalités

FP = Fonction Principale
FO = Fonction Optionnelle

Fonction	Objectif à remplir pour validation
FP1	L'utilisateur pourra importer des scènes 3D à partir du logiciel.
FP2	L'utilisateur après avoir importé une scène pourra la visualiser en utilisant l'éclairage direct avec calcul de visibilité.
FP3	L'utilisateur pourra déplacer la caméra au clavier.
FO1	L'utilisateur peut sélectionner une source de lumière (et optionnellement la déplacer).
FP4	L'utilisateur peut choisir quel mode de rendu utiliser parmis les méthodes proposées.
FP5	L'utilisateur peut visualiser le résultat avec les *RSMs* seules.
FP6	L'utilisateur peut visualiser le résultat avec les *RSMs* et les gaussiennes sphériques.
FO2	L'utilisateur peut visualiser le résultat avec les *RSMs*, les gaussiennes sphériques et les *ISMs*.
FO3	L'utilisateur peut sauvegarder une image.

Figure 1: Scénario de tests des fonctions principales et optionnelles du système

3.2.2 Validation de l'éclairage indirect

La validation du résultat se fera au final uniquement de manière visuelle et pas par un calcul de distance par rapport à une image de référence générée par exemple sous PBRT, comme nous l'avions précédemment évoqué.

3.2.3 Rappel des exigences non fonctionnelles

Nous avons une exigence non fonctionnelle fondamentale dans le cadre de ce projet, qui est en réalité une contrainte: le fonctionnement en temps réel. En effet, au moins pour la méthode *Reflective shadow maps*[1] et les *gaussiennes sphériques*[2], nous aurons besoin d'obtenir un résultat fonctionnant à un taux d'images par seconde satisfaisant (avoisinant les 60 images par seconde). Cette contrainte n'est pas simple à considérer étant donné que le nombre d'image par seconde est lié à la complexité de la scène (et cela même en rendu différé).

4 Réalisations / Présentation du travail effectué

4.1 Différences entre prévisionnel et réalisé

Cette partie va reprendre les exigences principales et optionnelles définies dans le dossier de spécification qui ont été validées ou non.

4.1.1 Exigences principales

FP = Fonction Principale

Fonction	Description	Priorité	Avancement
FP1	Charger une scène 3D.	Forte	Fait
FP2	Dessiner la scène avec éclairage direct.	Forte	Fait
FP3	Déplacer la caméra.	Forte	Fait
FP4	Choisir quelle méthode d'éclairage indirect utiliser.	Forte	Fait
FP5	Générer de l'éclairage indirect par avec la méthode *Reflective Shadow Maps*.	Forte	Fait
FP6	Générer de l'éclairage indirect avec les méthodes *RSM* et les gaussiennes sphériques.	Moyenne	Non fait

Figure 2: Développement des fonctions principales

Comme vous pouvez le voir, la plupart des exigences principales ont été respectées. Il manque à ce jour qu'une seule exigence principale qui aurait permis un meilleur résultat mais qui n'est pas nécessaire pour avoir une estimation de l'éclairage global de la scène.

4.1.2 Exigences optionnelles

FO = Fonction Optionnelle

Fonction	Description	Priorité	Avancement
FO1	Déplacer les lumières.	Faible	Fait
FO2	Générer de l'éclairage indirect avec les méthodes *RSM*, les gaussiennes sphériques et *Imperfect Shadow Maps*.	Faible	Non fait
FO3	Sauvegarder une image rendue sur disque.	Faible	Fait

Figure 3: Développement des fonctions optionnelles

Deux exigences optionnelles ont été implémentées malgré le fait que nous n'avons pas pu implémenter toutes les exigences principales et la faible priorité de ces exigences optionnelles, car elles permettent de compléter ce qui a été fait sans gros surcoût de développement et qu'elles présentent un intérêt lors des test. On peut par exemple remarquer des choses intéressantes grâce au déplacement des sources lumineuses.

4.2 Base de départ

Nous sommes partit sur un nouveau dépot github vierge pour développer la démo à partir de zéro.

4.3 Développement des modules

4.3.1 Module Core

Tâche	Description	Avancement	Difficulté
Boucle de rendu	Ecriture de notre boucle de rendu de base.	Fait	Faible
Algorithme de rendu	Implémentation de l'algorithme du *deferred shading*.	Fait	Forte
Shader program	Création de shader program : vertex shader prenant en compte les diverses transformations liés aux objets de la scène, fragment shader simpliste utilisant l'albédo du matériau.	Fait	Faible
Gestion des matériaux	Matériau simpliste : modèle de BRDF dans le fragment shader (rugosité, albedo, diffus, spéculaire).	Fait	Faible
Gestion de textures	Gestion et importation de textures avec *stb_image*.	Fait	Faible
Gestion des primitives d'éclairage	Structure de données pour les lumières : ponctuelles, directionnelles, spot, ...	Fait	Faible
Gestion d'un nuage de points	Structure de donnée "nuage de point" pour l'échantillonnage des VPLs pour l'ISM.	Non fait	Moyenne
Gestion caméra	Gestion de caméras.	Fait	Faible

Figure 4: Avancement du module Core

4.3.2 Module GUI

Tâche	Description	Avancement	Difficulté
Création fenêtre de rendu	Création de la fenêtre de rendu avec GLFW.	Fait	Faible
Gestion entrées clavier	Création d'"event handler" lors d'entrées clavier typiquement pour les déplacements de caméras.	Fait	Faible
Gestion entrées souris	Sur la fenêtre de rendu (changement de direction caméra, ...).	Fait	Faible
Selection mode d'éclairage	Création d'une interface utilisateur pour la sélection des différents mode de rendu.	Fait	Faible
Transformation objet	Permettre le déplacement ou la rotation d'objets (lumière ou mesh).	Fait	Forte
Selection d'objet	Permet la sélection d'un objet pour modification des matrices de transformation.	Fait	Moyenne
Sauvegarde sur disque	Permettre à l'utilisateur de sauvegarder l'image courante.	Fait	Faible
Import de fichier scène	Importer un fichier de scène .	Fait	Faible
Éditeur de lumière	Permettre d'éditer la couleur des lumières et la puissance pour mieux visualiser le résultat obtenu lors de comparaison de techniques d'éclairage indirect.	Fait	Faible

Figure 5: Avancement du module GUI

4.3.3 Module IO

Tâche	Description	Avancement	Difficulté
Importer scène	Utilisation de la librairie externe Assimp pour importer tous types de fichier compatibles vers notre structure de scène.	Fait	Moyenne
Sauvegarde image sur disque	Sauvegarde de l'image courante sur disque. Utilisation de write de *stb_image*.	Fait	Faible

Figure 6: Avancement du module IO

4.3.4 Module Lighting

Tâche	Description	Avancement	Difficulté
“G-Buffer”	Utilisation d’un “G-Buffer” pour les cartes de profondeurs de chaque source lumineuse.	Fait	Moyenne
Contribution	Évaluation de l’éclairage pour un fragment de l’image.	Fait	Forte
Interpolation	Interpolation en espace écran.	Fait	Forte
Opérations arithmétiques	Opérateurs arithmétiques pour les gaussiennes.	Non fait	Moyenne
Échantillonnage spatial	Phase de pré-calcul, création d’un nuage de points de la scène.	Non fait	Moyenne
Atlas de VPLs	Création d’une texture regroupant toutes les cartes de profondeurs des VPLs.	Non fait	Forte
Contribution	Calcul de la contribution des VPLs avec les gaussiennes sphériques.	Non fait	Forte

Figure 7: Avancement du module Lighting

5 Manuel d'utilisation

Le projet utilisant le gestionnaire de version git, le code source est disponible sur ce lien github `https://github.com/ValentinCamus/GIR-Engine`. La version finale du projet se trouve sur la branche master. Un fichier Readme vient appuyer la démarche de compilation. Ce manuel d'utilisation vient aider les personnes voulant en savoir plus sur ce logiciel.

5.1 Compilation

```
1 git clone https://github.com/ValentinCamus/GIR-Engine.git
2 cd GIR-Engine
3 mkdir Build
4 cd Build
5 cmake ..
6 make -j4
```

Pas besoin de faire un git submodule, les sous modules (assimp, glad, glfw, glm, imgui, stb, ...) sont automatiquement récupérés via le Cmake. Les assets (Sponza de Crytek et les textures "PBR", environ 200Mo) sont déja présents dans le dépot github.

5.2 Execution

Sous linux, vous pouvez lancer la démo simplement:

```
1 ./Sources/GIR_Engine      # depuis le dossier Build
```

5.3 Utilisation

5.3.1 Fonctionnement de la souris

Vous pouvez agir sur l'orientation de la caméra en maintenant le clic molette enfoncé et en déplaçant la souris.

5.3.2 Raccourcis clavier

Déplacement caméra d'Euler	Raccourci clavier (sur un clavier qwerty)
En avant	W
En arrière	S
A droite	D
A gauche	A
En haut	Q
En bas	E

Figure 8: Touche clavier pour déplacer la caméra d'Euler

La touche F5 permet quant à elle de recharger les shaders (ce qui est pratique quand on expérimente) et la touche F6 permet de prendre une capture de l'écran de rendu.

5.4 Informations sur l'interface

L'interface est constituée d'une grande fenêtre de rendu, ainsi que d'un panneau latéral sur la droite permettant plusieurs actions utilisateurs.

Figure 9: Vue d'ensemble

Le panneau latéral permet des interactions avec les objets de la scène permettant de voir comment se comporte l'éclairage indirect en orientant différemment les différentes sources de lumières. La sélection d'un objet dans la hiérarchie de scène fait apparaître un gizmo dans la fenêtre de rendu sur cet objet. Un système de docking inclus dans la librairie imGui permet d'organiser l'interface à la guise de l'utilisateur.

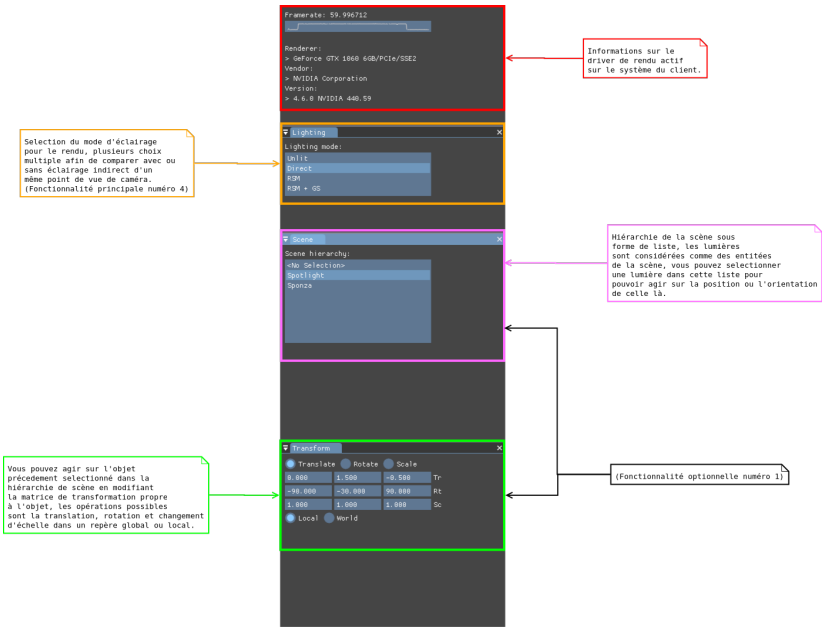

Figure 10: Panneau latéral servant d'interface utilisateur

6 Résultats des tests

6.1 Tests de recette et validation des fonctionnalités

Fonction	Objectif à remplir pour validation	Validé
FP1	L'utilisateur pourra importer des scènes 3D à partir du logiciel.	oui
FP2	L'utilisateur après avoir importé une scène pourra la visualiser en utilisant l'éclairage direct avec calcul de visibilité.	oui
FP3	L'utilisateur pourra déplacer la caméra au clavier.	oui
FO1	L'utilisateur peut sélectionner une source de lumière (et optionnellement la déplacer).	oui
FP4	L'utilisateur peut choisir quel mode de rendu utiliser parmis les méthodes proposées.	oui
FP5	L'utilisateur peut visualiser le résultat avec les *RSMs* seules.	oui
FP6	L'utilisateur peut visualiser le résultat avec les *RSMs* et les gaussiennes sphériques.	non
FO2	L'utilisateur peut visualiser le résultat avec les *RSMs*, les gaussiennes sphériques et les *ISMs*.	non
FO3	L'utilisateur peut sauvegarder une image.	oui

Figure 11: Tests de recette

6.2 Tests de validation de l'éclairage indirect

La fonctionnalité de sauvegarde du FBO courant sur le disque permettant une comparaison d'image entre éclairage direct et ayant été implantée, on pourrait mesurer la dissemblance des images pour mettre en valeur la contribution de la méthode d'éclairage indirect.

L'importation d'un fichier scène type PBRT avec un logiciel utilisant du lancer de rayons n'ayant pas été fait, nous ne pouvons pas comparer le réalisme de nos rendu avec une image de référence. On ne peut que remarquer certains défauts de la méthode qui sont flagrants pour l'oeil humain.

Nous allons donc proposer plusieurs échantillons d'images comparant la scène 3D d'un même point de vue avec ou sans éclairage indirect avec la méthode des RSMs[1], comparé avec un simple éclairage direct avec calcul d'ombrage (SM).

6.3 Résultats

En considérant un nombre suffisant de VPLs, on obtient des résultats plutôt lisses et propre dans la majorité des cas observés.

Figure 12: Eclairage direct uniquement

La différence entre les rendus avec et sans éclairage indirect est évidemment flagrante. Certaines variations restent cependant subtiles à percevoir. Un écran avec des noirs suffisamments profond peut être utile pour observer certaines des subtilités de la méthodes.

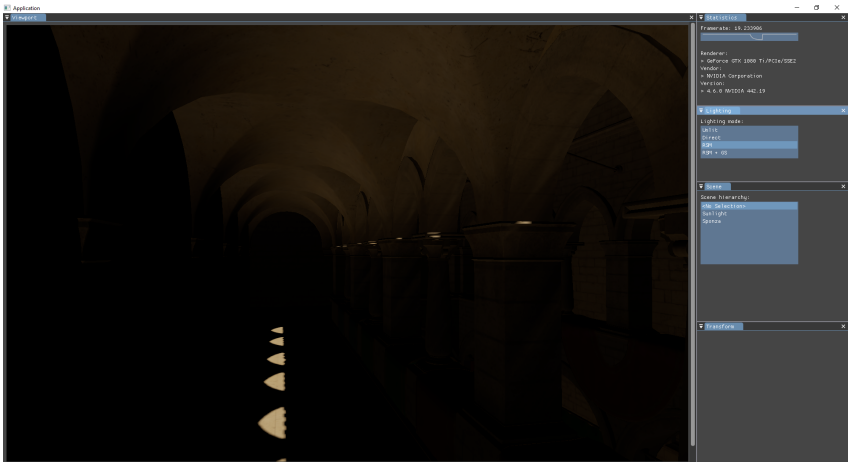

Figure 13: Eclairage direct et indirect (600 VPLs)

Vous noterez qu'on observe quand même quelques artefacts liés au nombre de VPLs considérées en haut des poteaux (à la base des arches), et cela même avec 600 VPLs par pixel.

Vous trouverez un certain nombre d'images supplémentaires dans l'archive du rapport.

6.4 Remarques

Même si l'exemple choisi précédemment est assez flatteur pour les RSMs[1], ces dernières sont loins d'être exempts de défauts.

Comme nous en avions parlé lors de la présentation et du rapport sur les méthodes et algorithmes, les RSMs[1] possèdent notamment le gros inconvénient de ne pas prendre en compte la visibilité entre les points de la scène et les VPLs. Cela donne par endroit des "applats" d'éclairage qui parraissent assez faux.

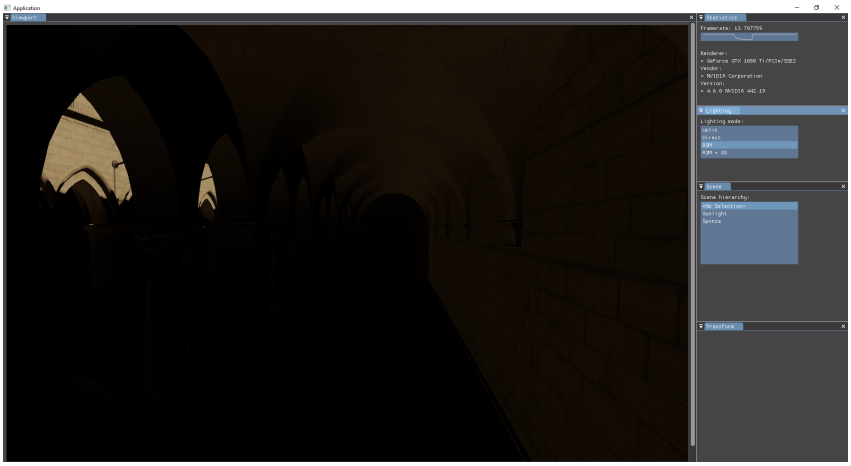

Figure 14: L'ombre des piliers n'est pas visible

Un autre défaut apparait quand on ne considère pas suffisamment de VPLs par pixels. C'est en réalité un artefact dont on vient de parler car il apparaissait à la base des arches sur la figure 13, le "banding".

Figure 15: Effet de banding dû au manque d'échantillons

C'est un artefact qui s'atténue bien avec le nombre d'échantillons, mais malheureusement l'impact sur le taux d'images par seconde devient vite important.

Globalement, les RSMs[1] tiennent leurs promesses. On obtient un résultat qui est plausible (pour reprendre les mots des auteurs) et qui apporte clairement un plus à l'image finale.

6.5 Analyse de performance

Les performances avec le calcul des RSMs sans interpolation en espace écran sont décevantes. En effet, un des gros points noirs de la méthode "naïve" (comprendre, sans interpolation en espace écran) est qu'on a besoin de réaliser un très grand nombre d'accès de textures par pixel de l'image finale (étant donné qu'on travaille avec un pipeline de rendu différé). Ces accès sont d'autant plus coûteux à réaliser qu'ils sont totalement aléatoire, ils ne profitent donc pas bien des caractéristiques du matériel.

Au final, en considérant autour de 70 VPLs par pixel (ce qui permet un résultat convenable, mais plus bruités que les images ci-dessus), on parvient à dessiner la scène entre 30 et 40 fois par secondes sur une Nvidia GTX 1050. Nous comptons assez fortement sur le temps qu'il nous reste avant la présentation pour finaliser l'implantation de l'interpolation en espace écran des RSMs et ainsi obtenir un taux de rafraichissement plus important avec un nombre suffisant de VPLs (200 par pixel).

7 Difficultés et améliorations

7.1 Difficultés

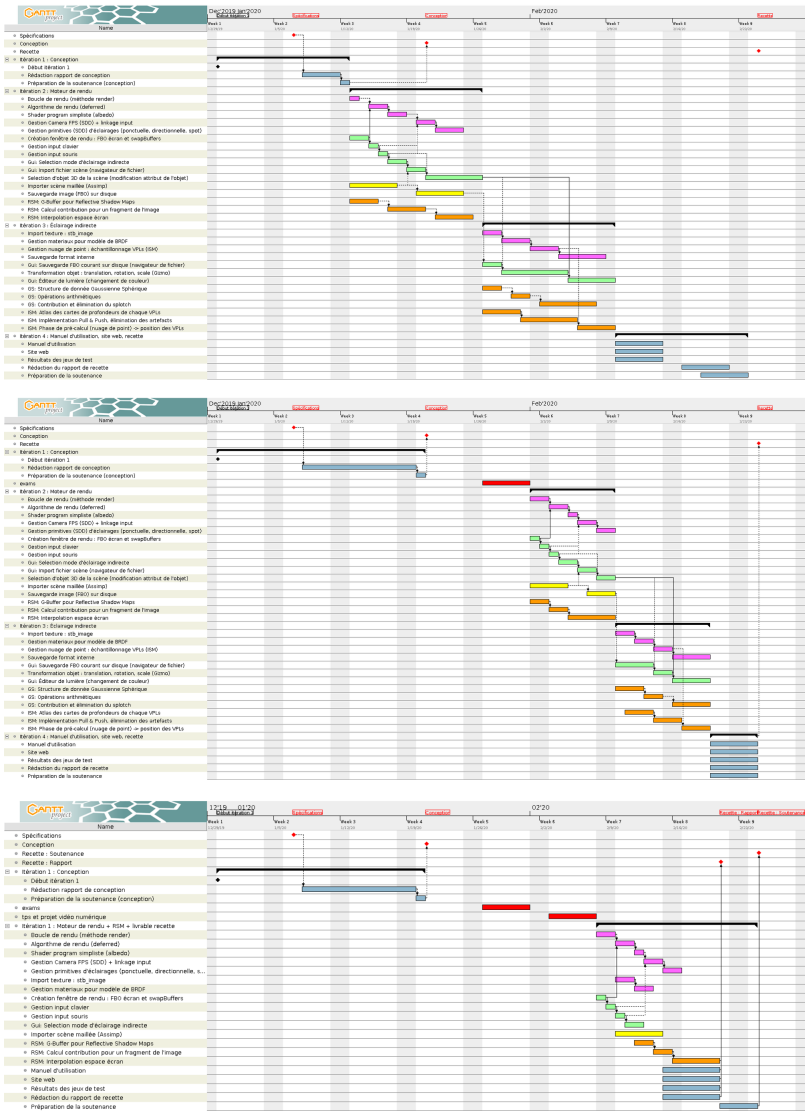

Figure 16: Planning prévisionnels 1, 2 et 3

De haut en bas dans la figure 16 ci haut: le diagramme de Gantt initial livré lors de la spécification, le deuxième diagramme de Gantt modifié pour le dossier de conception (modification lors de l'ajout d'une semaine d'examens

imprévue initialement) et le troisième diagramme de Gantt remonté après avoir pris connaissance d'un rapport de TP ainsi que d'un projet de vidéo numérique à rendre (la date limite de dépot du projet de vidéo numérique était le 16/02/2020).

Le projet idéal ce serait déroulé comme prévu dans le Gantt initial. Mais à cause de plusieurs imprévus, vous pouvez remarquer la suppression de tâches secondaires et la diminution du temps de travail sur ce projet en conséquence.

7.2 Risques déclenchés

Des risques initialement anticipés se sont déclenchés, avec notamment la perte de temps sur la construction du moteur en lui même et du coup moins de temps consacré au réel sujet de ce chef d'oeuvre qui reste l'éclairage global d'une scène 3D.

7.3 Risques imprévus

Un risque imprévu qui s'est présenté a été l'ajout d'un travail supplémentaire externe au projet. Comme vous venez de le voir, cela à bien sûr eu des conséquences sur le temps de travail consacrés au chef d'oeuvre et sur la planification prévue.

7.4 Évolutions possibles / Améliorations

Bien que certaines fonctionnalités initialement prévues n'aient pu être implantées, la base moteur reste solide avec une architecture simple et générique et une compilation multi plateformes, prête à recevoir d'éventuels ajouts ou perfectionnements des méthodes d'éclairages. Les évolutions possibles sont principalement liées à l'ajout des gaussiennes sphériques pour une meilleure estimation de la contribution des VPLs, ainsi que l'ajout des ISMs[3] pour répondre à une partie des défauts des RSMs[1].

Il serait également intéressant d'ajouter un éditeur de matériaux (rugosité, albedo, ...) pour voir comment la réflexion se comporte selon les caractéristiques du materiau.

La répartitions des tâches a été difficile à mettre en place car on ne pouvait travailler sur les modules d'éclairages sans avoir un moteur de rendu fini, et encore une fois, des travaux extérieurs sont venus perturbés les progressions de chacuns sur le projet. Le développement s'est fait de manière très "itérative", ce qui a empêché la parallélisation de certaines tâches et cela malgré le découpage en modules.

8 Conclusion

Nous avons réalisé (et donc testé) la plupart des fonctionnalités principales demandées par le client. Etant donné les fonctionnalités non accomplies, nous n'avons malheureusement pas pu présenter un résultat final équivalent à celui du papier de Square Enix[2], qui aurait permis un éclairage global de la scène encore plus réaliste. Tous les modules sont présents, il n'y a pas eu de modifications majeures par rapport aux spécifications et conceptions détaillées faites au préalable. Les objectifs de performances ne sont pas totalement atteints sans impact notable sur la qualité de l'éclairage indirect (en tout cas, à l'heure où ces lignes sont écrites). Mais l'objectif principal de simuler un éclairage en temps réel sur une scène 3D a été atteint.

9 Annexe

9.1 Planning des tâches

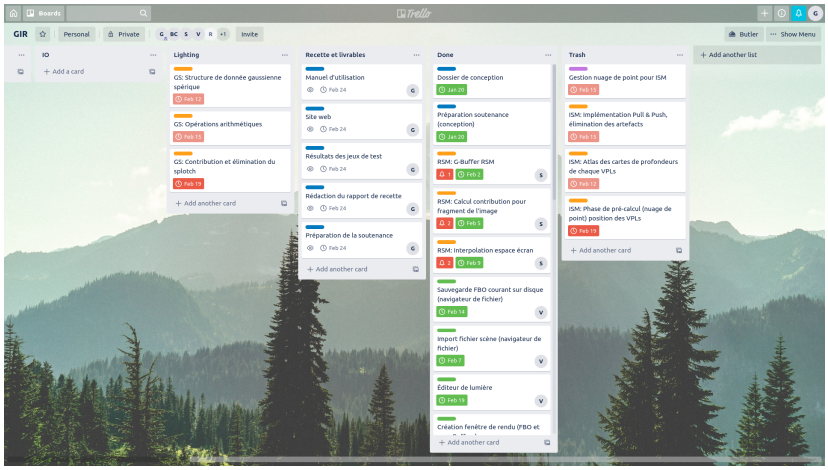

Figure 17: Planning des tâches finales

Image du Trello montrant l'ajout d'une colonne "Trash" ou certaines tâches ont été déplacées pour soulager la charge de travail sur deux semaines.

References

[1] Carsten Dachsbacher and Marc Stamminger. Reflective shadow maps. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, I3D '05, pages 203–231, New York, NY, USA, 2005. ACM.

[2] Yusuke Tokuyoshi. Virtual spherical gaussian lights for real-time glossy indirect illumination. *Computer Graphics Forum*, 34(7):89–98, 2015.

[3] T. Ritschel, T. Grosch, M. H. Kim, H.-P. Seidel, C. Dachsbacher, and J. Kautz. Imperfect shadow maps for efficient computation of indirect illumination. *ACM Trans. Graph.*, 27(5):129:1–129:8, December 2008.