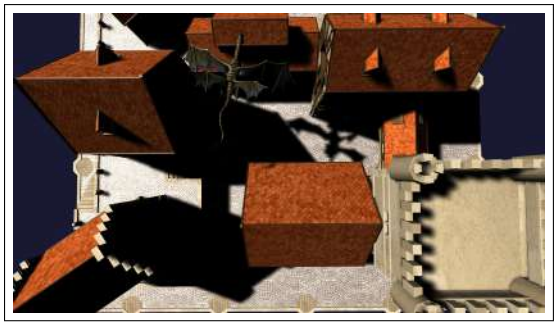
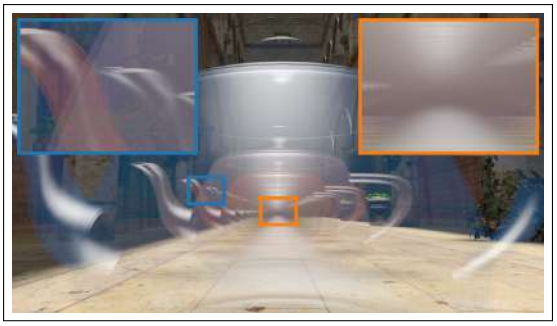
Chef-d'Œuvre

Moment Based Rendering

Rapport de recette





Baptiste Delos, Mehdi Djemai, Alban Odot,
Pierre Mézières et Jean-Baptiste Sarazin

Encadrant : Mathias Paulin

19 février 2019

Université Toulouse III Paul Sabatier
Université de Toulouse

IRIT

Introduction

La synthèse réaliste et interactive de phénomènes lumineux en informatique graphique constitue un vaste champ de recherche, en tant qu'elle est soumise aux exigences de temps d'exécution et de ressources mémoires inhérentes au rendu temps réel. En particulier, la qualité visuelle des effets d'ombres portées et de transparence représentés dans ce modèle de calcul formulent des problématiques encore largement étudiées. Les représentations fonctionnelles de ces phénomènes ne suivant pas un modèle linéaire, elles ne peuvent être naturellement intégrées/filtrées de manière efficace sur processeur graphique pour répondre à ces contraintes d'efficacité. L'intégration classique de la fonction de visibilité requiert un échantillonnage d'ordre exponentiel de cartes d'ombres pour assurer la qualité de l'ombrage [RSC87]. Cette contrainte de linéarité concerne aussi bien la reconstruction exacte de la fonction de transmittance, puisque la combinaison des couleurs de fragments transparents, non commutative, nécessite de trier ces derniers sur le pipeline graphique [NVI01].

Diverses approches ont tenté de résoudre le problème de non-linéarité de ces fonctions, en proposant de nouvelles représentations de ces dernières. [DL06] intègrent la fonction de visibilité pour chaque fragment ombragé, en représentant et en approximant une distribution de profondeurs à partir de sa moyenne et de sa variance. [MB13] évalue en outre la fonction de transmittance indépendamment de l'ordre des fragments transparents, en définissant des fonctions de coût paramétrées par leur profondeur qui supposent une distribution uniforme des surfaces transparentes entre le fragment traité et la caméra. En réponse à l'objectif de qualité des effets de transparence et d'ombrage, les études récentes de [PK15] et [Sha18] se basent sur l'exploitation de moments statistiques des paramètres de calcul (indice de transmittance et de visibilité) pour intégrer les deux phénomènes.

Dans le cadre de ces recherches, notre projet porte sur l'implantation des méthodes *Moment Shadow Mapping* et *Moment-Based Order Independent Transparency*, et leur évaluation vis-à-vis des approches précédentes, à mettre en place également. Les méthodes de *Percentage Closer Filtering* [RSC87] et de *Depth Peeling* [NVI01] permettant respectivement de reconstruire les fonctions de visibilité et de transmittance de manière exacte, elles sont par conséquent prises en référence pour l'évaluation de la qualité des méthodes basées sur les moments. Les approches de [DL06] et [MB13] permettront d'estimer, dans un second temps, leur efficacité. Dans la nécessité d'évaluer ces techniques de rendu sur le plan visuel comme celui de la performance, une part essentielle du projet a également été consacrée au développement d'outils de comparaison et de mesure de temps d'exécution.

Ce document réalise la synthèse des travaux effectués pour atteindre l'ensemble des objectifs présentés. Une première partie décrit la démarche à suivre pour la mise en place du projet, en détaillant la récupération et l'installation de la base logicielle du projet, ainsi que les commandes d'usage pour son utilisation. La seconde section de ce rapport aborde en détail l'ensemble des fonctionnalités implémentées dans le cadre du projet, accompagnées de résultats illustratifs [1]. Une première partie de ces réalisations concerne les améliorations apportées au moteur pour la simplification du développement ; la seconde présente les détails des implantations des techniques de rendu intéressées. Nous abordons en suivant les difficultés reconnues dans le développement, et les améliorations envisagées, pour conclure sur une validation des tests élaborés en phase de conception.

1. Une compilation d'images résultats est disponible sur la plateforme ArtStation

Table des matières

1 Manuel d'utilisation

Le projet est disponible sur ce dépôt **git**. Le fichier *README* du projet indique le processus de compilation à suivre. Mais ce manuel d'utilisation vient directement condenser les informations utiles pour faire fonctionner le projet dans le cadre du chef-d'oeuvre. La version finale du projet est disponible sur la branche **CO-master**.

1.1 Compilation

```
# Clonage du projet
git clone https://gitlab.com/PierreMezieres/Rogue.git
cd Rogue

# Utilisation des sous-modules
git subdmodule init
git submodule update

# Changement de branche
git checkout CO-master

# Compilation du projet
mkdir build
cd build
cmake ..
make

# Execution du projet
cd bin
./rogue
```

Pour éviter que le dépôt git ne devienne trop important, des assets sont fournis sur un dépôt **Google Drive**. Dans ce rapport, les assets du dépôt seront fournis accompagnés d'exemples d'utilisation. Il sera alors nécessaire de les télécharger et de mettre le dossier *Assets* dans le dossier principal du projet (*Rogue*) à côté des autres dossiers *Shaders/src* ....

1.2 Raccourcis

1.2.1 Fonctionnement de la souris

- **Clic droit maintenu** : Mouvement de la caméra
- **Clic gauche** : Sélection du modèle sous le curseur
- **Clic gauche maintenu** : Rotation du modèle sous le curseur

La caméra par défaut est une *TrackBall*. L'utilisateur peut choisir différentes caméras, mais leur fonctionnement est légèrement différent.

	Trackball	EulerCamera	Camera2D
Clic droit maintenu	Mouvement de la caméra sur la sphère	Mouvement de la caméra libre	Mouvement de la caméra sur un plan 2D
Molette	Gestion de la distance au centre de la sphère	Gestion du *fov*	Gestion du *fov*
Clic molette maintenu	Mouvement du centre de la sphère	Déplacement de la caméra en fonction du vecteur *front* de la caméra	Déplacement de la caméra en fonction du vecteur *front* de la caméra

1.2.2 Raccourcis clavier utile

Vous pouvez bouger les modèles dans la scène. Pour ce faire, il faut d'abord sélectionner le modèle avec un clic gauche. Vous pouvez ensuite appliquer des translations à l'aides des touches du clavier numérique (4, 5, 6, 7, 8 et 9). Une mise à l'échelle peut être effectuée au moyen des touches '+' et '-'.

1.3 Informations interface

Vous trouverez en annexe (section 5.1) des captures d'écrans de l'interface annotées d'informations jugées utiles pour pouvoir utiliser le *plugin* du chef-d'oeuvre.

2 Réalisations

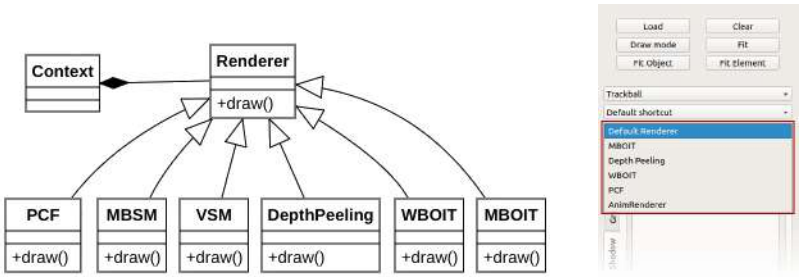
2.1 Améliorations du moteur

Dans le cadre du chef d'oeuvre, nous avons travaillé à partir du moteur 3D Rogue. L'idée de départ était de ne réaliser qu'un simple *plugin* pour le moteur. Comme expliqué dans le rapport de conception cependant, ce moteur ne présentait pas toutes les fonctionnalités nécessaires au bon développement du projet. Nous avions donc essayé d'identifier au plus tôt les fonctionnalités à rajouter au sein même du moteur. Dans un premier temps, nous allons revoir celles qui ont été identifiées. Dans un second temps, nous allons essayer de voir de manière exhaustive ce qui à été réalisé au sein du moteur.

2.1.1 Identifiées en phase de conception

En vue de faciliter le développement des différentes méthodes de rendu, diverses améliorations de la structure de notre base logicielle ont été envisagées dès la phase de conception du projet.

Hiérarchie de *renderers* La nécessité de pouvoir produire de manière dynamique une scène selon une méthode d'ombrage ou de transparence, au nombre de six dans le cadre de ce projet, a motivé le choix de spécialiser la gestion du rendu. Le patron de conception Stratégie permet de mettre en place une hiérarchie sur le principe de l'héritage, qui définit ainsi une entité (*renderer*) pour chaque technique de rendu (a). Chaque technique est sélectionnable par l'utilisateur via l'interface du plugin (b).



(a) (b)

FIGURE 1

Gestion avancée de la caméra La comparaison des différentes méthodes implantées implique naturellement de générer des images selon le même point de vue. Le système de chargement d'une scène (présenté en 2.1.2 au travers des améliorations supplémentaires) permet dans ce but la configuration de plusieurs caméras au sein d'une même scène. Chaque caméra enregistrée, définie par sa position, sa direction de vue et un vecteur d'orientation (*up vector*), peut être sélectionnée dans l'interface de base du moteur.

2.1.2 Fonctionnalités additionnelles

Plusieurs fonctionnalités ont pu être ajoutées, ou du moins améliorées. Voici donc, de la manière la plus exhaustive possible, les fonctionnalités qui ont été rajoutées.

Gestion des objets transparents

La plupart des méthodes de rendu des objets transparents se basent sur une passe sur les objets opaques, puis une ou plusieurs passes sur les objets transparents. Il nous était donc indispensable de gérer de manière suffisamment indépendante les deux types d'objets pour facilement itérer sur les différentes entités de la scène. Dans le moteur, la gestion des objets se résumant à une simple liste d'objets, nous avons amélioré le gestionnaire d'objets pour prendre en compte cette contrainte.

Amélioration des *Shaders*

Il s'agit d'une amélioration mineure. Initialement, le moteur ne permettait de gérer que les *vertex* et *fragment shaders*. Nous avons rajouté la possibilité de gérer les *geometry shaders*, qui se sont avérés très utiles pour la gestion de l'ombre des lumières points.

Matériaux

Cette amélioration concerne principalement la transparence. Les matériaux ne permettaient pas de gérer la transparence correspondant au canal *alpha*. La couleur des matériaux est gérée soit par une couleur appliquée sur tout l'objet, soit par une texture. Nous avons donc rajouté la possibilité de gérer la transparence de l'objet par la couleur diffuse de l'objet ou par la texture.

RogueLoader

Il s'agit ici du système permettant de charger des fichiers directement dans le moteur. Nous avons ajouté ou modifié plusieurs *tokens*. Il s'agissait de faciliter le plus possible la mise en place des scènes de test, et de pouvoir ainsi multiplier rapidement le nombre de scènes de tests.

— Création de shaders.
— Création de matériaux.
— Création de forme.

— Ajout des objets transparents.

Espaces de couleurs

Le module de comparaison comporte plusieurs opérateurs de comparaison définis dans différents espaces de couleurs (les opérateurs sont détaillés dans la sous-section 2.4). Pour ce faire, une partie de ce module permet de plonger les couleurs manipulées dans différents espaces. La gestion de ces derniers repose en grande partie sur la classe *QColor* de la librairie *Qt* utilisée dans ce projet. L'espace L*a*b* n'étant nativement pas pris en charge, nous avons donc rajouté des fonctions de conversion supplémentaires.

2.2 Module d'ombrage

Le module d'ombrage comprend deux composantes principales. La première porte sur le calcul des cartes de profondeurs vue depuis les lumières. La seconde comprend, l'utilisation de ces cartes de profondeurs pour déterminer si oui ou non la zone traitée (le fragment) est dans l'ombre, et l'application de l'une des trois méthodes que nous décrirons ci-dessous à savoir : *Percentage Closer Filtering*[RSC87], *Moment Shadow Mapping*[PK15] et *Variance Shadow Mapping*[DL06]. Pour l'ensemble des méthodes et de nos résultats, les cartes d'ombres sont de définition 1024*1024.

2.2.1 *Shadow maps*

Les *shadow maps* (cartes d'ombrages/profondeurs) comme dit plus haut représentent la distance aux objets de la scène vue depuis chaque lumière. Il y en a donc une par lumière. Le moteur Rogue en supporte trois types, à savoir : spot, directionnelle et ponctuelle. Les lumières spot et directionnelles ont un traitement très similaire de par le fait qu'elles émettent dans une seule direction. La lumière ponctuelle émet de la lumière dans toutes les directions, elle nécessite de produire six *shadow maps* dans six directions. Cela implique donc que les cartes de profondeurs pour les lumières orientées soient des texture 2D, alors que celles des lumières non-orientées seront des cubes de texture 2D. Les cubes seront centrés sur la lumière afin de couvrir le plus uniformément possible l'ensemble des directions possibles.

2.2.2 *Percentage Closer Filtering* [RSC87]

Percentage Closer Filtering est l'une des trois méthodes que nous avons implémentées. Nous l'avons choisie comme méthode de référence pour la qualité des ombres générées. De ce fait, nous lui avons accordé une forte priorité dans l'ordonnancement des tâches. Bien que le rendu soit très acceptable, cette méthode est coûteuse en ressources et demande donc beaucoup de temps de calcul pour du temps réel. Le filtrage des ombres se fait pendant le calcul de l'ombrage impliquant de nombreux accès textures. Nous avons implémenté un flou gaussien bilinéaire avec comme paramètres :

— La taille du noyau de convolution.
— L'écart-type
— Le biais de profondeur

La taille du noyau permet d'augmenter la taille du filtre de convolution, ainsi le paramétrer à 3 produira un noyau de taille 3x3. L'écart-type permet de définir la diffusion de l'échantillonnage, un écart-type élevé permettra d'échantillonner des texels éloignés du texel central. Enfin, le biais de profondeur permet d'éviter l'artefact d'acné d'ombre dû aux problèmes de discrétisation de la scène. Pour les images de nos résultats, nous utilisons par défaut un noyau de taille 9, un écart-type de 2.4 afin d'obtenir des résultats proche de ceux de [PK15].
Nous avons défini un jeu de trois tests qui feront office de tests unitaires. Afin d'obtenir des comparaisons entre les trois méthodes d'ombrage nous utiliserons les mêmes scènes pour Percentage Closer Filtering[RSC87], Moment Shadow Mapping[PK15] et Variance Shadow Mapping[DL06].

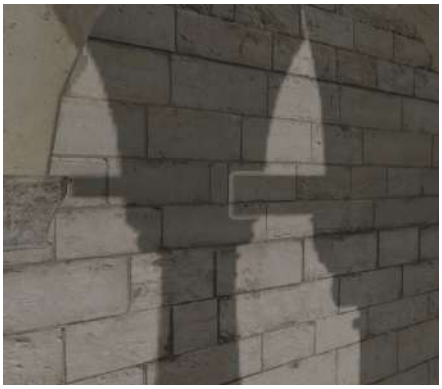
(a) Taille de noyau 0 et écart-type 2.4

(b) Taille de noyau 9 et écart-type 2.4

FIGURE 2 – PCF, biais de profondeur 0.005

2.2.3 *Variance Shadow Mapping* [DL06]

Variance Shadow Mapping propose de réduire la complexité algorithmique du calcul d'ombrage. La PCF est une méthode pouvant être relativement peu efficace du fait de ses nombreux accès textures, la VSM propose de rendre filtrable la *shadow map* avant le calcul de l'éclairage, en contrepartie d'un calcul de reconstruction de la fonction de profondeur. Cette méthode a pour but d'accélérer grandement le temps de calcul de l'ombrage. Sur la frontière des objets, la variation de profondeur est brutale (non-linéaire) et des artefacts apparaissent alors si on utilise seulement le moment d'ordre un avec une interpolation linéaire. Bien que proposant de très bons résultats, lorsque deux ombres se superposent cette méthode a tendance à surestimer la visibilité, et fournis donc des ombres trop claires. De plus, afin de lisser les ombres, nous appliquons un filtrage Gaussien sur les cartes de profondeurs. Ce filtre étant séparable, l'implantation en $O(nlog(n))$ permet d'obtenir de meilleures performances que la méthode de référence dont le filtrage est en $O(n^2)$. Nous paramétrons notre implantation de la VSM avec les paramètres du filtrage Gaussien dans un premier temps, comme pour la méthode PCF ainsi qu'avec une variance minimale.



(a) Taille de noyau 0 et écart-type 2.4

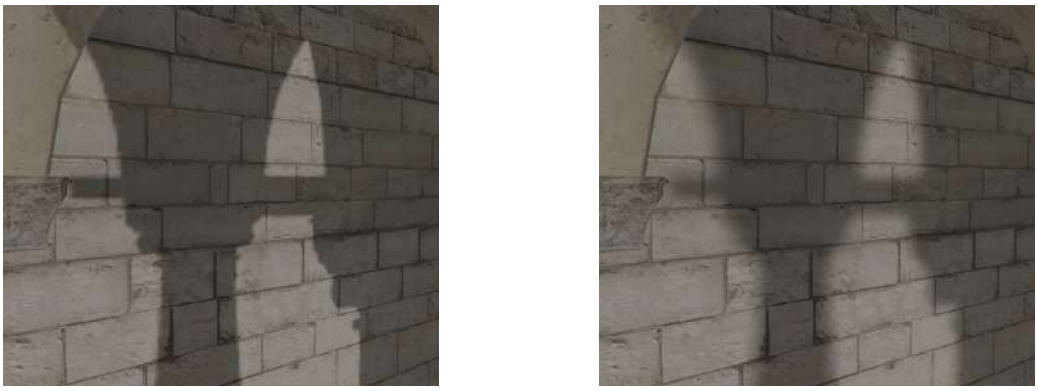
(b) Taille de noyau 9 et écart-type 2.4

FIGURE 3 – VSM, variance minimale 0.000002

2.2.4 *Moment Shadow Mapping* [PK15]

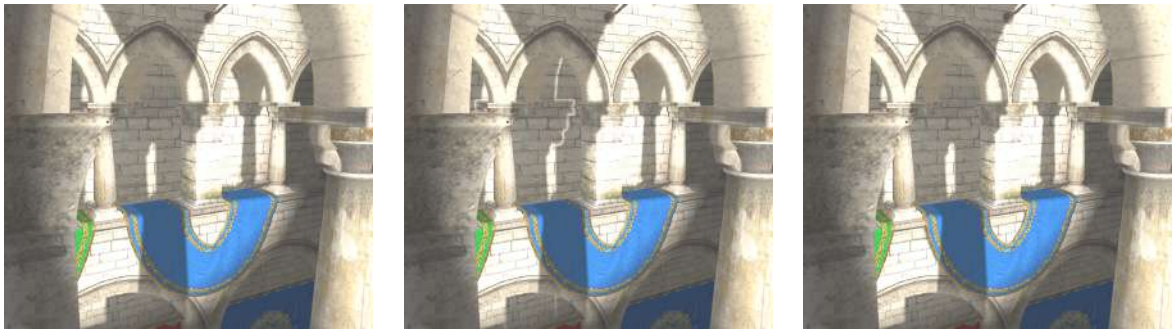
La méthode *moment based shadow mapping* permet, à l'instar de la méthode de *variance shadow mapping*, de pré-filtrer nos cartes d'ombres, mais l'utilisation de quatre moments au lieu de deux permet une meilleur reconstruction de la fonction de profondeur. De la même manière que le *Variance Shadow Mapping* nous faisons une passe de filtrage gaussien paramétrée avec une taille de noyau et un écart-type, la passe de calcul de l'intensité de l'ombre est paramétrée par un biais de profondeur et un biais de moment. Le biais de moment sert à compenser les imprécisions

numériques, cette valeur est petite, de l'ordre de 10^{-5}. Comme précédemment, nous utilisons une taille de noyau de 9 et un écart-type de 2.4.



(a) Taille de noyau 0 et écart-type 2.4 (b) Taille de noyau 9 et écart-type 2.4

FIGURE 4 – MSM, Biais de moment $3 \times 10^{-}5$, biais de profondeur 0.002



(a) PCF (b) VSM (c) MSM

FIGURE 5 – Les trois méthodes d'ombrages, la lumière est derrière la colonne de droite du premier plan.

2.3 Module de transparence

Le module rassemble l'ensemble des méthodes intéressées dans ce projet pour la synthèse de la transparence, traitées dans l'ordre de priorité suivant : la méthode de référence par *Depth Peeling*, les méthodes *Moment-Based Order Independent Transparency* et *Weighted Blended Order Independent Transparency* pour la comparaison de la seconde. Dans le même esprit que la conception du module d'ombrage, la synthèse d'une scène par chacune des techniques repose sur un *renderer* en interne, et l'utilisation de tampons spécifiques. À chacune correspond, en outre, un panneau de configuration des différents hyperparamètres associés dans l'interface.

2.3.1 *Depth peeling* [NVI01]

Le *Depth peeling* implémenté suit un modèle de composition *front-to-back*. C'est-à-dire que l'on rend en premier les couches les plus proches de la caméra. Comme on permet à l'utilisateur de choisir le nombre de couches à rendre, cela nous permet de garder un résultat visuellement acceptable. Si l'utilisateur choisit d'afficher très peu de couches, nous avons les premières couches affichées alors qu'avec l'ordre inverse (*back-to-front*) nous n'aurions que les dernières couches qui s'afficheraient.

(a) 3 passes (b) 10 passes

FIGURE 6 – Changement du nombre de passes sur les objets transparents avec *Depth peeling*.

2.3.2 *Weighted Blended Order-Independent Transparency* [MB13]

Cet algorithme propose de calculer la contribution de chaque surface transparente à l'aide de fonctions de poids. Ces dernières supposent, entre chaque surface et la caméra, une distribution uniforme des autres surfaces transparentes. Nous mettons à disposition de l'utilisateur plusieurs fonctions de poids, dont plusieurs ont été reprises de l'article original.

$$\text{Generic 1} = \alpha \cdot \max\left[10^{-2}, 3 \times 10^3 \cdot (1 - |z|)^3)\right] \tag{1}$$

$$\text{Generic 2} = \alpha \cdot \max\left[10^{-2}, \min\left[3 \times 10^3, \frac{10}{10^{-5} + (|z|/5)^3 + (|z|/200)^6}\right]\right] \tag{2}$$

$$\text{Generic 3} = \alpha \cdot \max\left[10^{-2}, \min\left[3 \times 10^3, \frac{10}{10^{-5} + (|z|/10)^3 + (|z|/200)^6}\right]\right] \tag{3}$$

$$\text{Scene adaptive} = \alpha \cdot \max\left[10^{-2}, \min\left[3 \times 10^3, \frac{0.03}{10^{-5} + (|z|/200)^4}\right]\right] \tag{4}$$

$$\text{Because why not} = \max\left[\min\left[1.0, \max\left[\max\left[R, G, B\right] * \alpha\right], \alpha\right] \times \text{clamp}(\frac{0.03}{(10^{-5} + (|z|/200)^4}, 10^{-2}, 3 \times 10^3)\right] \tag{5}$$

Une dernière fonction permet à l'utilisateur de fixer lui-même des paramètres.

$$\alpha^r \cdot \text{clamp}\Big[\frac{0.03}{10^{-5} + (|z|/d)^S}, 10^{-2}, 3 \times 10^3\Big] \tag{6}$$

où

— r définit l'indice de résistance de la couleur (*color resistance*), qui permet de modérer le poids de l'opacité α du fragment. Ce paramètre est à incrémenter dans le cas où, par exemple, certaines surfaces transparentes de premier plan, à faible opacité, affectent la contribution des surfaces transparentes de fond.

— d représente l'intervalle de profondeur de la scène (*depth range*) au-delà duquel la contribution des surfaces transparentes doit nécessairement tenir compte de leur ordre. Il est d'usage d'augmenter ce paramètre si des surfaces quasi-opaques semblent excessivement transparentes ; de le diminuer dans le cas où les contributions de fragments transparents relativement distants sont indifférenciées.

— S est un poids supplémentaire permettant de réguler l'importance de la couleur des surfaces d'arrière-plan vis-à-vis de celles des premiers plans (*ordering strength*).

(a) équation 1 (b) équation 2 (c) équation 3

FIGURE 7 – *WBOIT* avec différentes fonction de poids.

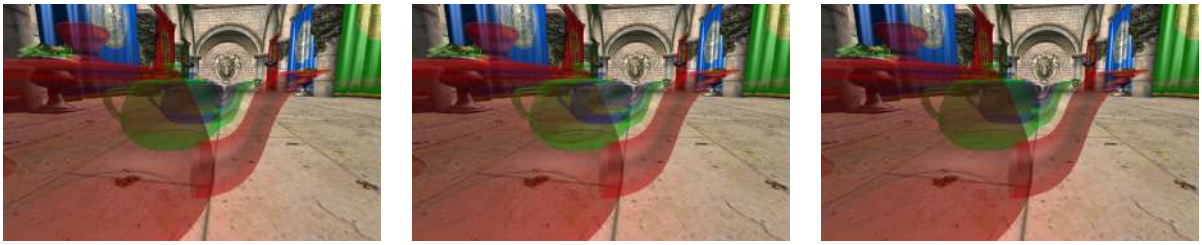
2.3.3 *Moment-Based Order-Independent Transparency* [Sha18]

La méthode *MBOIT* estime la transmittance de chaque fragment transparent indépendamment de son ordre, en fournissant une approximation de la fonction de transmittance basée sur un nombre fixé de moments calculés sur le paramètre de profondeur. La version de l'algorithme intégré dans le moteur réalise un traitement sur quatre passes.

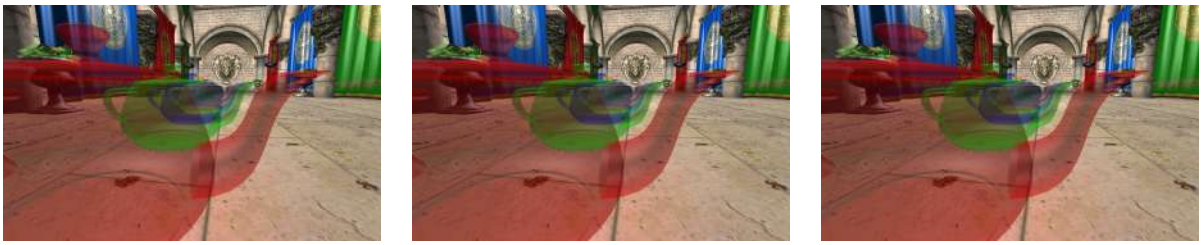
— **Synthèse des objets opaques**
— **Génération des moments**
— **Calcul de la transmittance** de chaque fragment en fonction de sa profondeur
— **Combinaison finale** des fragments opaques et transparents

Les deuxième et troisième passes font aussi intervenir un ensemble de shaders rédigés en GLSL directement sur le modèle de ceux fournis par [MKKP18], qui se chargent d'appeler les méthodes de génération (*MomentIOTgeneration.glsl*) et de déterminer l'indice de transmittance (*MomentIOTresolution.glsl*) sélectionnées en fonction des choix de l'utilisateur dans l'interface.

Cette méthode propose l'utilisation de moments exponentiels (au nombre de 4, 6, ou 8) et trigonométriques (au nombre de 2, 3 ou 4), autant de combinaisons de moments possibles que l'utilisateur est en mesure d'en spécifier via l'interface du plugin. Les hyperparamètres α et β qui définissent respectivement le biais du moment et le facteur de surestimation, identifiés en phases de spécifications, sont également éditables.



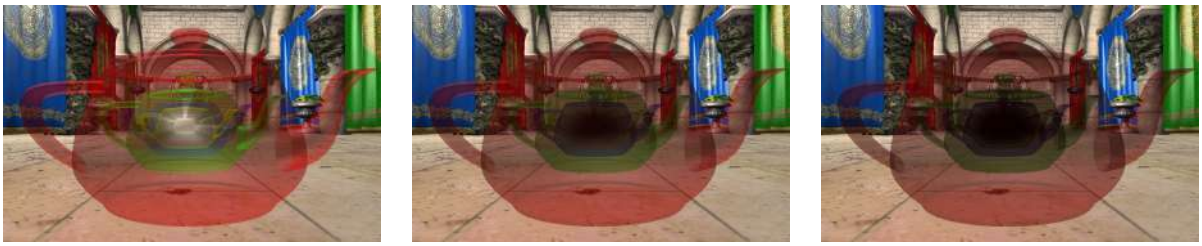
(a) 4 moments exponentiels (b) 6 moments exponentiels (c) 8 moments exponentiels



(d) 2 moments trigonométriques (e) 3 moments trigonométriques (f) 4 moments trigonométriques

FIGURE 8 – Résultats obtenus sur des cas de transparence par l'utilisation de moments exponentiels et trigonométriques, avec $\alpha = 5 \cdot 10^{-4}$ et $\beta = 0.25$.

La figure suivante présente des cas de transparence résolus à partir de 4 moments exponentiels et des valeurs du facteur de surestimation β prises sur l'exemple de celles présentées dans l'article original.



(a) $\beta = 0.0$ (b) $\beta = 0.25$ (c) $\beta = 1.0$

FIGURE 9 – Conséquence de la variation du paramètre de surestimation β.

2.4 Module de comparaison

Le module de comparaison contient principalement deux parties. Dans un premier temps, la comparaison d'images, pour voir ce que les différents algorithmes ajoutent ou enlèvent au résultat final de l'image. Et dans un seconds temps, nous tentons de venir mesurer le temps de génération d'une image pour mesurer l'efficacité des algorithmes en terme de temps.

2.4.1 Comparaison d'images

Pour la comparaison d'images, on a implémenté plusieurs opérateurs à l'aide de différents espaces de couleurs.

Niveaux de gris

Trois méthodes ont été implémentées pour la comparaison en niveaux de gris.

— **Clarté :** Niveaux de gris = $\frac{(max(R,G,B)+min(R,G,B))}{2}$
— **Moyenne :** Niveaux de gris = $\frac{(R+G+B)}{3}$
— **Luminosité :** Niveaux de gris = $0.21R + 0.72G + 0.07B$

Couleurs - espace HSV

L'espace HSV nous permet d'exhiber la teinte, la saturation et la valeur. Nous avons mis en place quatre comparaisons pour cet espace. Trois pour comparer chacune des composantes une à une et la quatrième pour comparer directement la longueur des vecteurs dans ces 3 dimensions.

Couleurs - espace L*a*b*

Dans l'espace L*a*b*, nous avons mis en place la formule du *Delta E76*, qui correspond à une distance euclidienne dans cet espace. L'avantage de cet espace est que la répartition des couleurs est très proche de la perception des écarts de couleur par le système visuel humain. Donc, une simple distance euclidienne permet déjà de quantifier notre perception.

$$\Delta E^*_{ab} = \sqrt{(L^*_2 - L^*_1)^2 + (a^*_2 - a^*_1)^2 + (b^*_2 - b^*_1)^2}$$

2.4.2 Temps de calcul

Pour la mesure du temps de calcul, nous voulions essayer de mettre en place une mesure assez précise avec des requêtes OpenGL. Mais nous n'avons pas eu le temps d'aller aussi loin que nous l'aurions voulu. Alors, on mesure simplement le temps entre le lancement du rendu de l'image et une fois que l'image a été généré. C'est-à-dire, une fois que toutes les requêtes OpenGL lancées lors du rendu sont terminées.

```
// Mesure du temps
int start = timesMSecs();
scene->draw();
glFinish();
int end = timesMSecs();
times = end - start;
```

3 Difficultés et améliorations

3.1 Difficultés rencontrées

Bien que les différentes méthodes d'ombrages et de transparences ont été implantées, des difficultés ont été rencontrées concernant la compatibilité mac ou encore suivant les différentes configurations, nous poussant à revoir nos objectifs. Les lumières points demandent un traitement particulier, d'autant plus si elles doivent être filtrable, nous avons créé nos propres cube maps et fonction d'échantillonnage de ces cubes maps, malheureusement nous n'avons pas eu de résultats à temps, nous poussant ainsi à ne pas tenir compte des lumières points pour le calcul de l'ombrage. De plus, la variante des moments trigonométriques pour le *moment based shadow mapping* n'est pas fonctionnelle.

3.2 Pistes d'améliorations

Concernant les méthodes basées sur les moments, une amélioration possible est d'optimiser l'espace en mémoire prises par les textures stockant les moments, en passant d'une précision de réels sur 32bits à 16bits, cette perte de précision se paie par un surcoût algorithmique et n'a pas été implanté dans le cadre de ce projet. La prise en charge des lumières points pour le calcul d'ombrage est une amélioration intéressante pour un moteur graphique. Il aurait été intéressant également d'implémenter le calcul de l'ombrage avec les moments trigonométriques d'ordre deux. Concernant le module de comparaison, celui-ci aurait pu fournir une mesure du temps plus précise et intégrer une fonction de comparaison plus proche de la perception visuelle.

4 Résultats des tests

4.1 Ombrage

N°	Principe du test	Note
0	Une scène sans objet	Nous devrions obtenir une texture blanche.
1	Une scène avec une sphère centrée sur la lumière	Nous devrions obtenir une texture avec une couleur unie, dont la valeur représentera le rayon de la sphère.
2	Une scène avec deux objets l'un devant l'autre	Nous devrions obtenir texture avec une rupture de profondeur, les valeurs devant être conformes aux distances mesurées sur la scène.

Afin de ne présenter que les résultats ayant un intérêt visuel nous ne présenterons pas les images des tests 0 et 1. En effet l'image générée est comme prévue blanche pour le test 0 et monochrome gris pour le test 1.

Test 2 : Deux objets l'un devant l'autre

(a) PCF (b) VSM (c) MSM

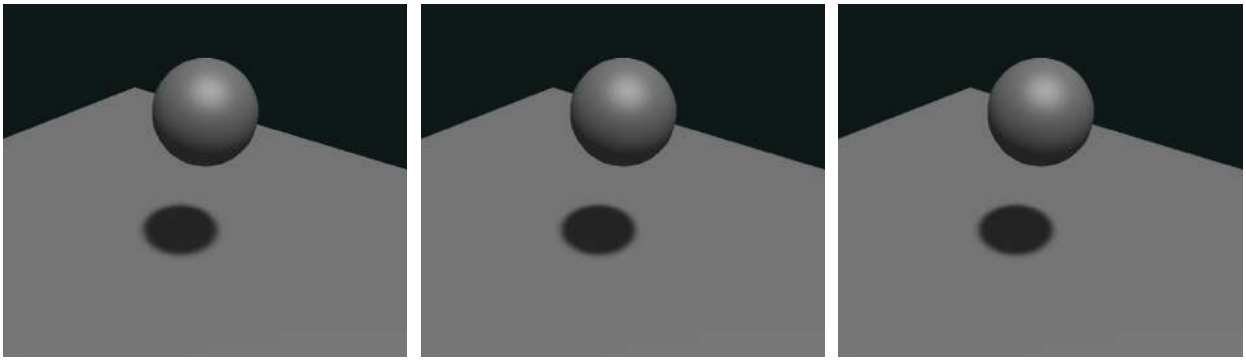
FIGURE 10 – Carte de profondeur générée par nos 3 méthodes pour une lumière directionnelle.

(a) PCF (b) VSM (c) MSM

FIGURE 11 – Carte de profondeur générée par nos 3 méthodes pour une lumière spot.

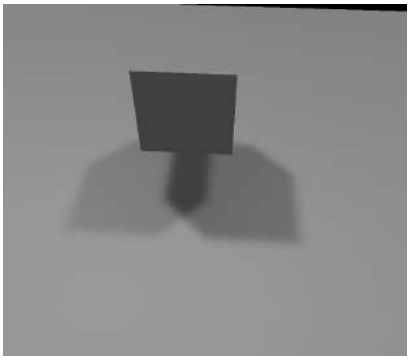
N°	Principe du test	Note
3	Un objet et un plan sur lequel l'ombre se projette et une lumière.	Nous allons pouvoir observer la qualité de l'ombrage (aliasing)
4	Un objet et un plan sur lequel l'ombre se projette et deux lumières placées pour que les ombres s'intersectent.	Observer le niveau d'ombre entre une zone non occultée, une zone occultée par un objet et un zone occultée par les deux objets.
5	Plusieurs objets et plusieurs lumières	Observer le recouvrement d'ombre qui devrait occasionner une fuite de lumière dans le cas de la méthode VSM mais pas dans les autres méthodes.

Test 3 : Un objet et une lumière

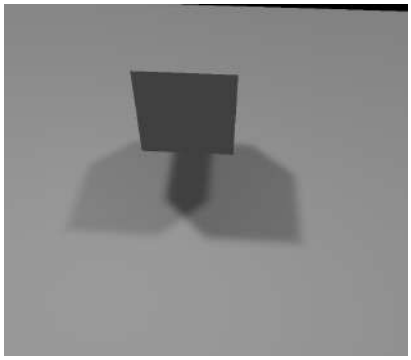


(a) *PCF* (b) *MSM* (c) *VSM*

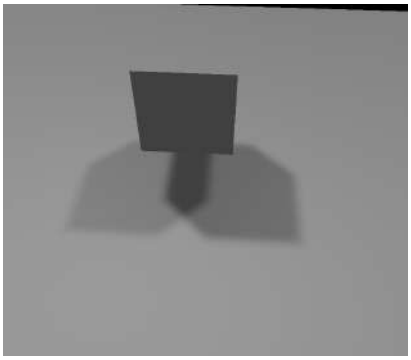
Test 4 : Un objet et deux lumières



(d) *PCF*

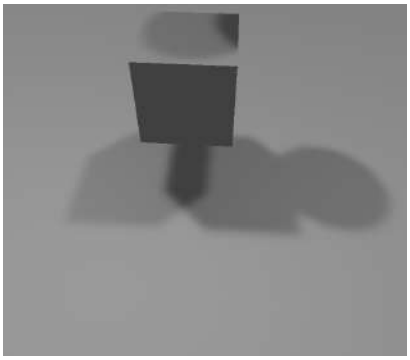


(e) *MSM*

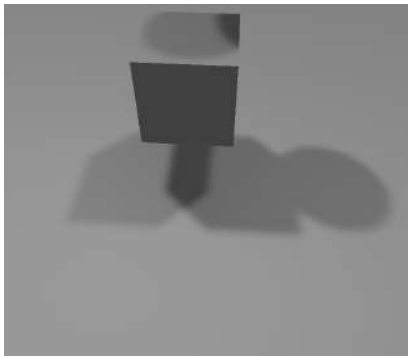


(f) *VSM*

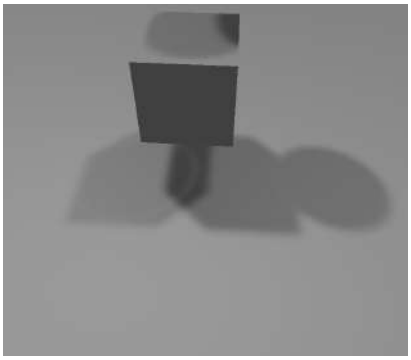
Test 5 : Deux objets et deux lumières



(g) *PCF*



(h) *MSM*



(i) *VSM*

4.2 Transparence

À la manière de la validation des tests des méthodes d'ombrage, il s'agit de présenter les résultats obtenus à l'issue des tests de transparence, élaborés dans le document de conception du projet. Il nous paraît pertinent de rappeler que la validation de ces tests est majoritairement basée sur la perception visuelle, étant donnée la difficulté à évaluer numériquement le traitement réalisé par les *shaders* sur le GPU.

N°	Principe	Résultat attendu
6	Scène sans objet transparent	Vérifier que le rendu des objets opaques est bien le même avec ou sans objet transparent
7	Un objet transparent devant un objet opaque	Vérifier que l'objet transparent est bel et bien transparent
8	Intersection entre un objet transparent et un objet opaque	Vérifier que l'objet transparent ne ressort pas de l'objet opaque
9	Intersection entre un objet transparent et un autre objet transparent	Vérifier s'il n'y a pas d'artefacts apparent et que le résultat est cohérent
10	Succession d'objets transparents	Vérifier que la transmittance baisse au fur et à mesure de la rencontre avec les objets transparent.

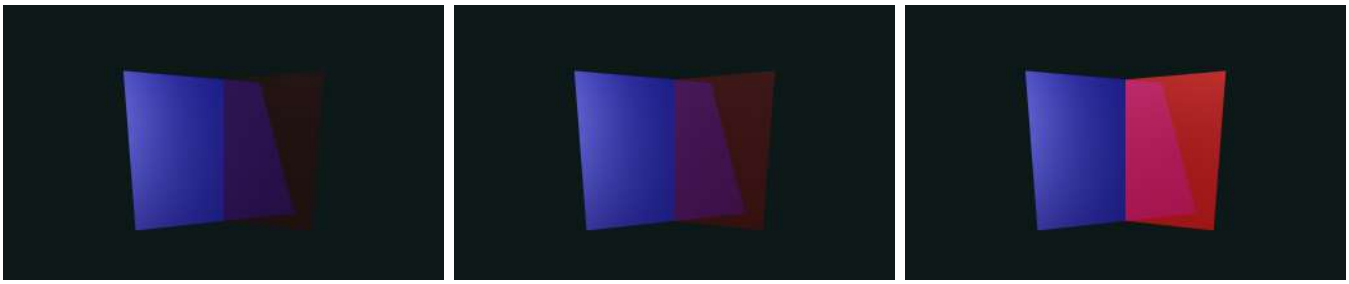
Test 6 : Préservation de l'environnement opaque

(j) Rendu par défaut (k) *Depth Peeling* (l) *MBOIT* (m) *WBOIT*

Test 7 : Contribution amoindrie des surfaces transparentes

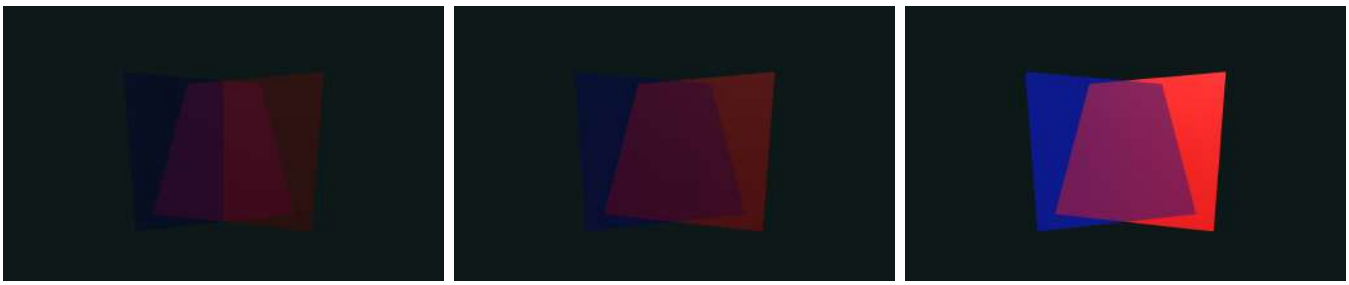
(n) *Depth Peeling* (o) *MBOIT* (p) *WBOIT*

Test 8 : Intersection de surfaces opaques et transparentes



(q) *Depth Peeling* (r) *MBOIT* (s) *WBOIT*

Test 9 : Intersection de surfaces transparentes



(t) *Depth Peeling* (u) *MBOIT* (v) *WBOIT*

Test 10 : Diminution de la transmittance totale avec le nombre de surfaces transparentes

(w) *Depth Peeling* (x) *MBOIT* (y) *WBOIT*

4.3 Comparaison

Pour le module de comparaison, nous n'avions pas forcément défini tout les opérateurs de comparaison qui seraient implémentés au final. Nous avions seulement défini des tests sur un opérateur de moindre carrée. Mais, dans un soucis de pertinence, nous avons abandonné cet opérateur au profit d'autres opérateurs définis dans la section 2.4.

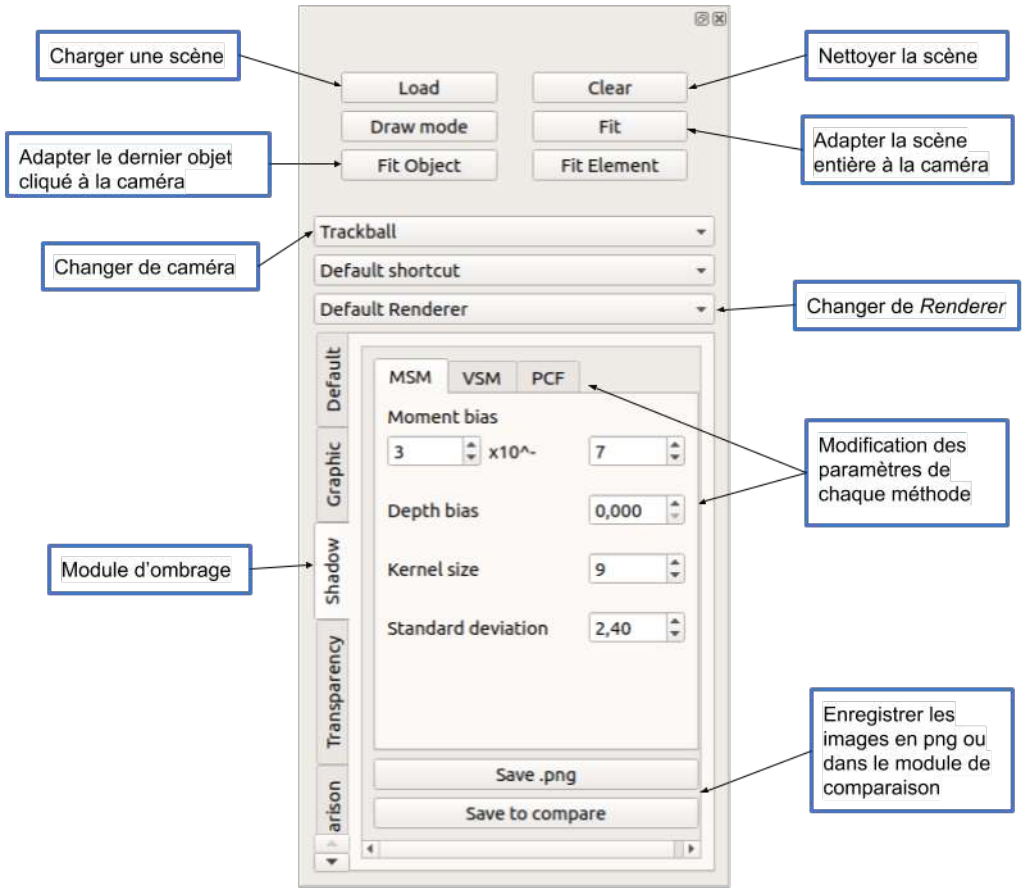
Comme défini, pour deux images identiques, chaque opérateurs est censé nous donner une image noire. Cependant, pour 2 images totalement différentes, les opérateurs ne donneront pas le même résultat. Puisque la notion de différence est différente en fonction de l'espace de couleur. Nous avons donc rapidement mis en place quelques tests pour savoir si les résultats donnés par les opérateurs étaient cohérents. Nous avons essayé de générer des images intégralement noires *(rgb=0,0,0)* ou blanches *(rgb=1,1,1)* pour voir les résultats des opérateurs. Au final, nous obtenons les résultats suivants :

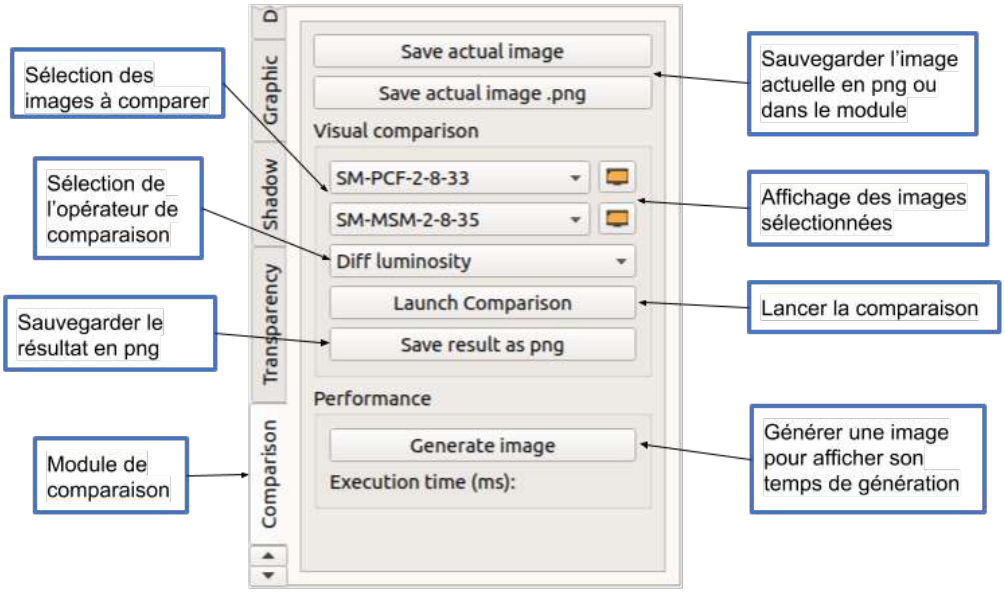
— Niveaux de gris (clarté) : image blanche
— Niveaux de gris (Moyenne) : image blanche
— Niveaux de gris (Luminosité) : image blanche
— Couleurs - espace HSV : image grise
— Couleurs - espace HSV (teinte) : image noire
— Couleurs - espace HSV (saturation) : image noire
— Couleurs - espace HSV (valeur) : image blanche

Ces résultats semblent bien cohérents à ce que nous étions en droit d'attendre des opérateurs.

5 Annexe

5.1 Informations pour l'interface





Références

[DL06] William Donnelly and Andrew Lauritzen. Variance shadow maps. 2006.

[MB13] Morgan McGuire and Louis Bavoil. Weighted blended order-independent transparency. 2013.

[MKKP18] Cedrick Münstermann, Stefan Krumpen, Reinhard Klein, and Christoph Peters. Moment-based order-independent transparency. 2018.

[NVI01] Cass Everitt NVIDIA. Interactive order-independent transparency. 2001.

[PK15] Christoph Peters and Reinhard Klein. Moment shadow mapping. 2015.

[RSC87] William T. Reeves, David H. Salesin, and Robert L. Cook. Rendering antialiased shadows with depth maps. 1987.

[Sha18] Brian Sharpe. Moment transparency. 2018.