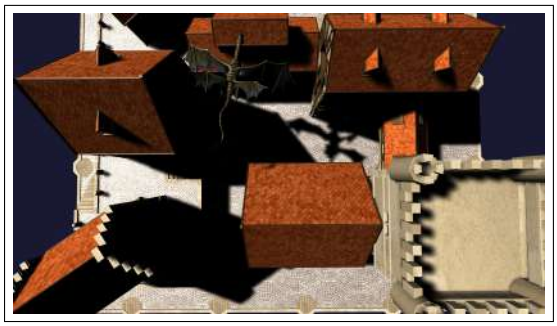
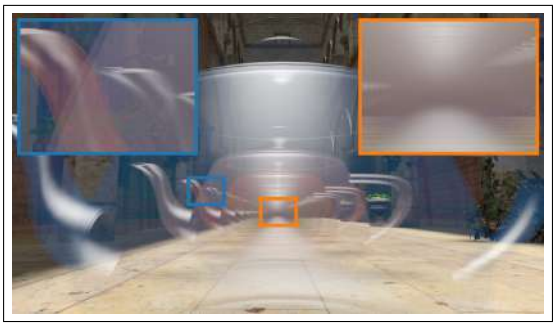
Chef-d'Œuvre

Moment Based Rendering

Méthodes et Algorithmes





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

Encadrant : Mathias Paulin

19 novembre 2018

Université Toulouse III Paul Sabatier
Université de Toulouse

IRIT

1 Introduction

Ce rapport présente, à travers deux grandes parties, les aspects algorithmiques et les méthodes d'implantation de travaux de recherche récents, orientés sur le calcul des ombres portées et la gestion de la transparence. La première section dresse la problématique générale du projet, les bases logicielles utilisées ainsi que les mesures de comparaisons des résultats envisagées. La seconde partie de ce rapport aborde en détail les aspects algorithmiques du projet. Nous détaillerons dans un premier temps les méthodes existantes prises en référence, en détaillant leurs avantages et limitations. Enfin, nous nous attarderons davantage sur la démarche d'implantation de ces travaux. Les deux articles qui servent de base à ce chef-d'oeuvre sont [MKKP18] et [PK15]. Dans ce rapport, nous n'aborderons pas en profondeur les algorithmes basiques d'ombrage et de transparence.

2 Méthodes

2.1 Attente et vision globale des articles

Le travail que nous sommes amenés à effectuer est l'implantation des principaux algorithmes de transparence et d'ombrage, en plus des méthodes basées sur les moments. Ce travail correspond typiquement à l'implantation de différents *renderer* dans un moteur 3D temps réel. Nous comparerons les différentes méthodes suivant deux critères :

- un critère visuel, d'abord par comparaison subjective de zones de l'image d'une part, et par l'utilisation d'opérateurs de comparaison d'autre part.
- un critère de performance algorithmique, mesuré par comparaison du temps de calcul des images entre les différentes méthodes.

2.2 Problème principal

Dans le cadre du rendu temps réel, l'objectif principal consiste à reconstruire des fonctions continues à l'aide de fonctions discrètes. En effet, on travaille sur une version discrétisée de la scène. Lors du processus de *rasterization*, les calculs sont réalisés sur des fragments, ensembles de données nécessaires pour générer un pixel. Par exemple, pour le calcul d'ombrage, on peut vouloir reconstruire la fonction de visibilité pour l'ombre, ou la fonction d'absorbance pour la transparence. La réalisation de ces reconstructions en temps réel nécessite de mettre en place des filtres de reconstruction efficaces pour minimiser leur temps de calcul. Toutefois, le problème qui vient se poser est celui du filtrage linéaire. Les cartes graphiques traitent directement ce filtrage de manière matérielle, mais celui-ci n'est par définition pas applicable pour la reconstruction de fonctions non linéaires. Pour venir calculer l'ombrage ou la transparence dans une scène, on travaille sur la profondeur des fragments. La profondeur étant rarement une donnée linéaire, comme le montre la figure 1, on se retrouve rapidement sur des cassures dans la profondeur dès lors que les objets sont espacés. La fonction de visibilité et la fonction de transmission ne sont donc pas des fonctions linéaires. L'une des solutions explorées dans ce rapport pour pouvoir appliquer du filtrage linéaire est finalement de représenter la profondeur par des données linéaires.

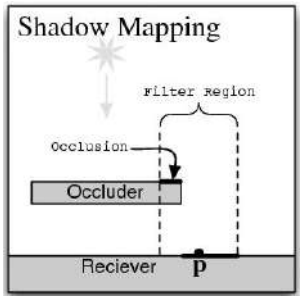


FIGURE 1 – Profondeur non linéaire

2.3 Point de départ

Nous avons choisi de prendre à la base de ce projet un moteur 3D développé par un membre de l'équipe, une première raison qui justifie ce choix logiciel. Le moteur possède une base qui évitera de devoir développer un nombre excessif de fonctionnalités manquantes avant de pouvoir véritablement implanter les algorithmes qui représentent le coeur du chef d'oeuvre. Une des toutes premières tâches à réaliser sera de s'assurer que le moteur est compilable sous MacOS. Nous pourrons ensuite implanter des *renderer* qui s'occuperont de faire de l'ombrage ou de la transparence de manière plus ou moins basique. Enfin, nous pourrons implanter les algorithmes présentés dans les papiers.

FIGURE 2 – Moteur Rogue

Présentation du moteur (03/11/18) : A l'heure actuelle, le moteur possède les caractéristiques suivantes :
- Au moins deux types de caméras (*EulerCamera, Trackball*).
- Trois types de lumières (*point, spot et directionnelle).*
- Chargement de modèle avec la bibliothèque *Assimp*.
- Son propre type de fichier pour rapidement rajouter des éléments.
- Matériaux Blinn-Phong.
- Structure pour les objets (Un objet peut posséder plusieurs éléments, chaque élément possède son propre maillage).
- Maillage géré avec les sommets, une normale et une coordonnée de texture pour chaque sommets. Un vecteur d'indice paramétrable pour afficher lignes, triangles, quadrilatères ...
- *Renderer forward*.
- Shader géré avec *vertex shader* et *fragment shader*.
- Gestion du *ray casting*.

La mise à disposition de *shaders* commentés et écrits en *HLSL* par les auteurs des deux méthodes, pourront en outre contribuer aux choix d'implantation des shaders en *GLSL* à intégrer dans le moteur Rogue.

2.4 Comparaisons

Ce projet étant orienté sur une étude comparative des méthodes [MKKP18] et [PK15] par rapport aux algorithmes basiques d'ombrages et de transparence, différentes approches sont envisageables. La mise en place délicate de métriques robustes pour la comparaison de ces méthodes impose l'utilisation de métriques perceptuelles classiques (différences d'images résultats, distances euclidiennes dans l'espace L*a*b*, RGB, etc.).

Concernant le *Shadow Mapping*, la technique qui servira de référence sera le *Percentage-Closer Filtering* (nvidia-GPU gems). Pour la transparence, l'algorithme de référence sera la transparence avec le *Depth peeling* (nvidia-OIT). Ces approches nous laissent ainsi le choix d'exploiter des outils existants tels que les opérateurs de comparaisons de MATLAB, ou d'implanter ces outils dans le moteur directement.

3 Algorithmes

3.1 Moment Shadow Mapping

3.1.1 Rappel sur le shadow mapping

Une shadow map stocke la profondeur des fragments depuis un point de vue particulier. En général, on utilise une shadow map avec $m \in \mathbb{N}$ canaux et une map $b : [0,1] \to \mathbb{R}^m$ qui fait correspondre la profondeur $z \in [0,1]$ d'un texel de shadow map à un vecteur $b(z)$ stocké dans la shadow map. Ici, le nombre de canaux correspond au nombre d'informations que l'on vient stocker par pixel. Le shadow mapping représente le cas basique où $\mathbf{b}(z) := \mathbf{z}(z) := z$. En d'autres termes, on n'utilise qu'un seul canal qui permet de stocker seulement la profondeur.

3.1.2 Moment Shadow Mapping

Dans cette section, nous aborderons la publication concernant la méthode de *Moment Shadow Mapping* [PK15]. Cette publication présente une nouvelle méthode de résolution d'ombrage pour le rendu temps réel. L'idée est, ici, de ne plus se baser uniquement sur la profondeur des fragments, mais sur les moments statistiques des profondeurs. L'espérance mathématique étant linéaire, l'accélération matérielle est désormais possible. La *shadow map* est devenue filtrable. Le problème qui se pose maintenant est de pouvoir reconstruire la fonction de visibilité depuis la valeur des moments. Nous implanterons ici quatre techniques : *Hamburger four moments shadow mapping* (H4MSM ou MSM), *Hausdorff four moments shadow mapping* (Hausdorff 4MSM), *Trigonometric moment shadow mapping* (TMSM) et la technique *Variance shadow mapping* (VSM). Ces quatre techniques se basent sur des moments différents. Selon les moments utilisés, on peut récupérer certaine propriétés plus ou moins intéressantes. Par exemple, la méthode *Hamburger* minore fidèlement la luminosité réelle, ce qui est plutôt intéressant pour le calcul de visibilité.

Positionnement du problème Le principe consiste à échantillonner une shadow map via une variable aléatoire sur $\mathbb{R}^2$. Soit P une distribution de probabilité sur $\mathbb{R}^2$, $t : \mathbb{R}^2 \to \mathbb{R}^2$ une variable aléatoire et $s : \mathbb{R}^2 \to [0,1]$ une shadow map. On peut considérer qu'une image est un processus aléatoire, P représentant ainsi cette distribution aléatoire. On a alors $\mathbf{z} := s \circ t$ une variable aléatoire dénotant la profondeur sur $[0,1]$ et $Z := P_Z : \mathcal{B}([0,1]) \to [0,1]$ est la distribution de profondeur par le noyau t sur la shadow map s.

Soit $\mathfrak{P}(\Omega)$ l'ensemble des distributions de probabilité sur Ω. Par définition $P \in \mathfrak{P}(\Omega)$ est une fonction sur $\mathbb{R}$. Soit $\omega_1, \omega_2 \in \mathbb{R}, P_1, P_2 \in \mathfrak{P}(\Omega)$ et $\omega \in \mathcal{B}(\Omega)$. $\mathcal{B}$ étant une σ-algèbre sa structure d'espace vectorielle nous donne

$$(\omega_1 P_1 + \omega_2 P_2)(\omega) = \omega_1 P_1(\omega) + \omega_2 P_2(\omega)$$

Cette égalité implique qu'une combinaison linéaire de profondeur se traduit par une combinaison linéaire des ombres. On observe alors qu'un texel de la shadow map s peut être considéré comme une distribution de Dirac $P_t = \delta_t$. Par extension, on a $P_{\mathbf{z}} = \delta_{s(t)}$.

A partir de ce résultat, nous pouvons construire $\phi : \mathfrak{P}([0,1]) \to \mathbb{R}^m$ une application linéaire tel que :

$$\begin{aligned} \mathbf{b} : [0,1] &\to \mathbb{R}^m \\ z &\mapsto \phi(\delta_z) \end{aligned}$$

On en déduit que $\forall Z \in \mathfrak{P}([0,1])$ à support fini, on a $\phi(Z) = \mathbb{E}_Z(\mathbf{b})$ et inversement si $\mathbf{b} : [0,1] \to \mathbb{R}^m$ est une variable aléatoire bornée :

$$\begin{aligned} \phi : \mathfrak{P}([0,1]) &\to \mathbb{R}^m \\ Z &\mapsto \mathbb{E}_Z(\mathbf{b}) \end{aligned}$$

ϕ est linéaire.

On va alors stocker un vecteur $\mathbf{b}(z)$ dans une shadow map filtrable. Puisque ces coefficients sont linéairement filtrables on va pouvoir réaliser un anti-aliasing sur la shadow map.

Nous possédons maintenant l'ensemble des données nécessaires à la construction d'une bonne approximation de la fonction de visibilité. La première idée serait de faire une reconstruction linéaire, mais la distribution effective

des profondeurs sur les bords des objets dans une scène est très différente d'une simple distribution de Dirac. On va alors se tourner vers une reconstruction non-linéaire. C'est le principe introduit par le Variance shadow mapping qui utilise $\mathbb{E}(\mathbf{z})$ et $\mathbb{E}(\mathbf{z}^2)$ pour trouver un minorant de la fonction de visibilité. Cette approche fournit un minorant optimal sans plus d'information.

On choisit de trouver un minorant de la fonction de visibilité pour éviter le problème d'acné dû à l'auto-occlusion engendré par une surestimation de la fonction de visibilité. Le problème de modélisation de la fonction de visibilité par une fonction minorante est l'introduction de *light leaking*. Le *light leaking* est une surestimation de la luminosité de l'ombre portée. Le but du *four moment shadow mapping* est donc de minimiser ce *light leaking* en trouvant un meilleur minorant de la fonction de visibilité.

Ainsi, on va définir un espace de recherche $\mathfrak{S}$: Soit $I \subset \mathbb{R}$, soit $\mathbf{b} : I \to \mathbb{R}^m$, et soit $Z \in \mathfrak{P}(I)$. On pose $b := \mathbb{E}_Z(\mathbf{b})$

$$\mathfrak{S}_b := \{S \in \mathfrak{P}(I) | \mathbb{E}_S(\mathbf{b}) = b\}$$

On définit le minorant optimal pour $F_S(z_r) = P([-\infty, z_r])$ par

$$\rho_I(b, z_r) := \inf_{S \in \mathfrak{S}_b(I)} F_S(z_r)$$

Notre but est donc de trouver $\rho_I(b, z_r) := \inf_{S \in \mathfrak{S}_b(I)} F_S(z_r)$ et d'indiquer un échec si $\mathfrak{S}_b = \varnothing$ (**problème 1**).

Solution au problème Le fait que $\mathfrak{S}_b$ soit un espace de dimension infinie pose un problème important concernant la résolution de notre problème par un algorithme. Nous allons discrétiser I, en gardant les mêmes notations que précédemment.
Soit $T \in [0, 1]$ de cardinal fini. On a $\mathfrak{P}(T) \subset \mathfrak{P}([0, 1])$.
Notre but devient donc de trouver $\rho_T(b, z_r) := \inf_{S \in \mathfrak{S}_b \cap \mathfrak{P}(T)} F_S(z_r)$ et d'indiquer un échec si $\mathfrak{S}_b \cap \mathfrak{P}(T) = \varnothing$.

On se trouve donc dans un espace de recherche linéaire fini de dimension $Card(T)$, linéairement contraint par $S([0, 1]) = 1$ et $\mathbb{E}_S(\mathbf{b}) = b$. L'algorithme 1 est une solution à ce problème, puisque nous devons trouver $Z \in \mathfrak{S}_b \cap \mathfrak{P}(T)$ et $F_Z(z_r) = \rho_T(b, z_r)$. Or, soit $\bar{A} := \begin{pmatrix} 1 \cdots 1 \\ A \end{pmatrix}$ et $\bar{b} := \begin{pmatrix} 1 \\ b \end{pmatrix}$. On remarque que

$$\bar{A}.\omega = \bar{b} \iff (1, \cdots, 1).\omega = 1 \text{ et } A.\omega = b\text{d}$$

En particulier on remarque que $\sum_{i=1}^{n} \omega_i = 1$. Donc Z est une combinaison linéaire de distribution de $\mathfrak{P}(T)$ de ce fait $Z \in \mathfrak{P}(T)$.

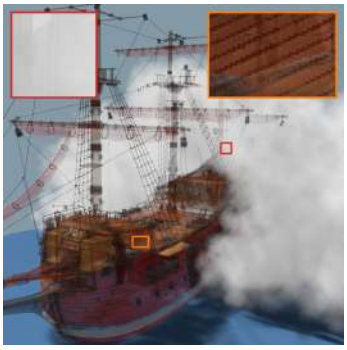
Hamburger Four Moments Shadow Mapping Cet algorithme est l'algorithme 2 présenté dans l'article *Moment Shadow Mapping* [PK15]. Il propose une solution pour $b := \mathbf{b}_{MSM} := (z, z^2, z^3, z^4)^T$ et $I = \mathbb{R}$. Nous laissons le soin au lecteur d'approfondir l'algorithme sur [PK15] pour une meilleure compréhension des explications. Dans notre cas, nous travaillons donc sur les quatre premiers moments.

Un autre idée est d'utiliser seulement 16 bits pour définir chacun des moments, puisque selon les calculs, seuls les 16 premiers bits apportent une précision quantifiable. De cette manière, on va pouvoir diviser par deux le besoin en mémoire pour stocker les textures. Il est prévu dans un premier temps de coder la technique sur les 32 bits classiques. Dans un second temps, on mettra en place la technique sur 16 bits. On en profitera pour voir s'il existe une réelle différence. L'un des autres avantages de cet algorithme est le fait qu'il n'est pas sensible à la distance Z_{near}, Z_{far}. Cela est dû au fait qu'il est défini sur $\mathbb{R}$ et non sur $[0, 1]$ comme Hausdorff.

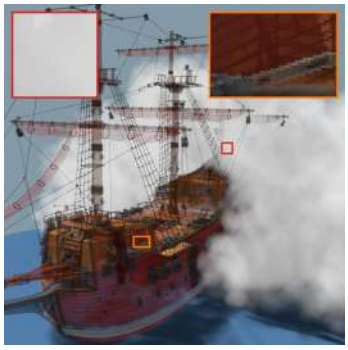
Hausdorff Four Moment Shadow Mapping *Hausdorff 4MSM* est très similaire à *MSM*, mais se différencie comme dit plus tôt par son intervalle de définition $I = [0, 1]$. La conséquence de ce choix est que les ombres proches apparaissent plus sombres. Ce phénomène est d'autant plus amplifié lorsqu'on utilise une quantification sur 16bits.

Trigonometric Moment Shadow Mapping *TMSM* utilise un nouvel algorithme de résolution passant par la minimisation d'une quartique. La minimisation souffre d'instabilités, notamment lorsque le minimum est proche de zéro, les calculs d'inverses deviennent alors périlleux. Son ensemble de définition est $I = [0, 1]$. Il se différencie aussi par la conception de son vecteur $b := b_{TMSM} := (sin(2\pi z), cos(2\pi z), sin(4\pi z), cos(4\pi z))$.

3.2 Moment-Based Order Independent Transparency



MBOIT avec 6 moments



Ground truth : Depth Peeling

3.2.1 Rappel sur l'Order Independent Transparency (OIT)

L'objectif premier considéré dans la gestion de la transparence consiste à faire la composition de n $\in \mathbb{N}$ fragments de couleurs L_0,..., L_{n-1}, d'opacités α_0,...n α_{n-1} et de profondeurs $z_0, ..., z_{n-1}$. La couleur d'un fragment est donnée par la formule de composition suivante :

$$\sum_{l=0}^{n-1} L_l \cdot \alpha_l \cdot T(z_l) \tag{1}$$

Le terme $T(z_l)$ correspond à la transmittance du fragment à la profondeur z_l. La méthode d'*alpha blending* est une manière courante de déterminer cet indice de transmission :

$$T(z_l) := \prod_{\substack{k=0 \\ z_k < z_l}}^{n-1} (1 - \alpha_k) \tag{2}$$

Ce calcul nécessite que les fragments alignés sous un pixel soient triés : un tri de la géométrie est nécessaire, de même qu'un tri des fragments en cas d'intersection d'objets de la scène. Le coup en terme de ressources de ces ordonnancements complique son implantation pour des applications temps réel.

Dans ces problématiques, les méthodes OIT tentent d'annuler cette dépendance à l'ordre des fragments en déterminant une approximation de la transmittance $T(z_l)$ sans tri préalable.

3.2.2 Moment-Based Order Independent Transparency (MBOIT)

La méthode MBOIT [MKKP18] s'inscrit dans cet objectif en exploitant, sur le modèle du *Moment Shadow Mapping* [PK15] décrit en *3.1.3*, les moments d'une profondeur $z \in [0, 1]$ donnée. Le taux d'absorption $A(z_f)$ du fragment à la profondeur z_f est ainsi déduite de sa transmittance $T(z_f)$ définie par

$$A(z_f) := -\ln(T(z_f)) = \sum_{\substack{z_l < z_f \\ l=0}}^{n-1} -\ln(1 - \alpha_l)$$

Me passage en log permet ici de traiter la composition de manière additive. Cet indice d'absorption est considéré comme la fonction de distribution cumulative de la mesure Z telle que

$$Z := \sum_{l=0}^{n-1} -\ln(1 - \alpha_l) \cdot \delta(z_l)$$

qui annule la dépendance à l'ordre des fragments. L'exploitation des moments de z permet de définir le vecteur de moments $\mathbf{b}(z)$. Dans le cas de l'algorithme HFMSM, $\mathbf{b}(z) = (1, z, z^2, z^3, z^4)^T$. L'espérance $\mathbb{E}_z(\mathbf{b})$ définit alors la

transmittance du fragment à la profondeur z, indépendante de son ordre :

$$b := \mathbb{E}_z(\mathbf{b}) := \sum_{l=0}^{n-1} -\ln(1-\alpha_l) \cdot \mathbf{b}(z_l)$$

L'algorithme 1 de l'article correspondant évalue alors la couleur d'un pixel au travers de quatre principales étapes : le calcul du taux d'absorption de la surface opaque sous le pixel, l'estimation de la transmittance de chaque fragment transparent qui réside entre la surface opaque et le pixel, le calcul de l'indice d'absorption de ces fragments, et la composition finale des deux dernières mesures pour l'estimation de la radiance résultante à ce pixel.

1) Calcul de la transmittance totale Le premier coefficient b_0 fournit l'indice d'absorption totale $\sum_{l=0}^{n-1} -\ln(1-\alpha_l)$, et e^{-b_0} représente alors l'indice de transmittance totale qui pondère la radiance de la surface opaque. Contrairement à la méthode *MSM* dont le moment b_0 est 1, il devient donc nécessaire de stocker cette valeur dans la matrice B.

2) Approximation de la transmittance L'algorithme 2 décrit dans l'article [PK15] permet de déterminer ensuite une approximation $A(z_f, b, \beta)$ de l'indice d'absorption du fragment de profondeur z_f, de laquelle est déduite la transmittance $T(z_f, b, \beta) = \exp(-A(z_f, b, \beta))$ en chaque pixel.

3) Composition des fragments transparents La couleur produite par la combinaison de toutes les surfaces transparentes est déterminée en intégrant à l'équation (1) l'approximation de la transmittance T :

$$\sum_{l=0}^{n-1} L_l \cdot \alpha_l \cdot T(z_f, b, \beta)$$

avec L_l la couleur du fragment transparent d'indice l, stockée de manière temporaire dans un Render Buffer.

4) Détermination de la couleur finale Le calcul de la couleur du pixel consiste intuitivement à ajouter au résultat de la composition des radiances des surfaces transparentes, la radiance du fragment opaque de fond L_n pondérée par l'indice de transmittance totale $\exp(-b_0)$. La formule suivante donne ainsi la couleur finale du pixel considéré :

$$\exp(-b_0) \cdot L_n + \Gamma \cdot \sum_{l=0}^{n-1} L_l \cdot \alpha_l \cdot T(z_f, b, \beta)$$

avec

$$\Gamma = \frac{1 - \exp(-b_0)}{\sum_{l=0}^{n-1} \alpha_l \cdot T(z_f, b, \beta)}$$

le terme de normalisation repris de la méthode *Weighted Blended OIT* [MB13].

Minimisation des erreurs numériques

La reconstruction de la transmittance basée sur les moments reste sensible aux erreurs numériques, d'autant plus que celles-ci augmentent avec l'intervalle de profondeur considéré. Une approche pour pallier l'effet de ces imprécisions et améliorer la qualité visuelle globale propose de borner au sens logarithmique les profondeurs z avant leur manipulation. L'approche tient compte d'un intervalle $[z_{min}, z_{max}]$ dont les bornes représentent les profondeurs minimale et maximale de la sphère englobante construite sur l'ensemble des géométries transparentes de la scène. La profondeur z devient :

$$z := \frac{lnz_v - \ln z_{min}}{\ln z_{max} - \ln z_{min}} \cdot 2 - 1$$

où $z_v \in \mathbb{R}$ représente les profondeurs dans l'espace de vue. Cette approche de *depth warping* permet de conserver une précision relative indépendante de l'échelle de la scène.

4 Annexes

Algorithme 1 de l'article *Moment Shadow Mapping* [PK15]

Algorithme 1 Solution au problème 1
Entrées : $T := \{z_1, ..., z_n\} \subset [0,1], \mathbf{b} : [0,1] \to \mathbb{R}^m, z_r \in [0,1]$
Sortie : $Z \in \mathfrak{S}_b \cap \mathfrak{P}(T)$ avec $F_Z(z_r) = \rho_T(b, z_r)$

$A := (\mathbf{b}_j(z_i)) \in \mathbb{R}^{m \times n}$

$\bar{A} := \begin{pmatrix} 1 \cdots 1 \\ A \end{pmatrix}$

$\bar{b} := \begin{pmatrix} 1 \cdots 1 \\ b \end{pmatrix}$

$p \in \mathbb{R}^n$ avec $p_i := \chi_{-\infty, z_r}(z_i)$

Trouver ω qui minimise $p^T.\omega$ sujet à la contrainte $\bar{A}.\omega = b$ et $\omega_i \geq 0 \forall i \in \{1, ..., n\}$

Retourner $Z = \sum_{i=1}^{n} \omega_i \delta_{z_i}$ en cas de succès

Indiquer $\mathfrak{S}_b \cap \mathfrak{P}(T) = \varnothing$ sinon

---

Références

[MB13] Morgan McGuire and Louis Bavoil. Weighted blended order-independent transparency. *Journal of Computer Graphics Techniques (JCGT)*, 2(2) :122–141, December 2013.

[MKKP18] Cedrick Münstermann, Stefan Krumpen, Reinhard Klein, and Christoph Peters. Moment-based order-independent transparency. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 1(1) :7 :1–7 :20, May 2018.

[PK15] Christoph Peters and Reinhard Klein. Moment shadow mapping. In *Proceedings of the 19th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, i3D '15, pages 7–14, New York, NY, USA, 2015. ACM.