

# Rendu réaliste et temps réel pour la réalité augmentée

Hadrien Croubois<sup>1,2</sup>

<sup>1</sup> ENS de Lyon

<sup>2</sup> Université Claude Bernard Lyon I

**Résumé** <Text of the summary of your article>

## **Remerciements**

## **Table des matières**

Rendu réaliste et temps réel pour la réalité augmentée .....  
*Hadrien Croubois*

## 1 Introduction

Le travail détaillé dans ce document a été réalisé de février à juillet 2014 au sein de l'équipe R3AM<sup>3</sup> en vue de l'obtention du Master IGI de l'UCBL<sup>4</sup>. Il a été encadré par Jean-Philippe Farrugia et Jean-Claude Iehl.

Ce travail a été financé par l'ENS de Lyon<sup>5</sup>, au même titre que le reste du M2 IGI, dans le cadre de la troisième année correspondant au statut de normalien.

Ce stage est issu du constat selon lequel les applications orientées temps réel actuellement disponible sur mobile ne correspondent pas à ce que propose l'état de l'art en terme de rendu réaliste. Une autre constatation est que les dispositifs mobiles actuels sont de plus en plus à même d'acquérir leur environnement (notamment au moyen de caméras frontales).

L'optique de ce stage est donc d'évaluer les possibilités d'acquisition et de rendu en temps réelle sur une plate-forme mobile et de proposer une solution logicielle complète mettant en œuvre les mécanismes considérés, du repérage à l'étape de rendu en passant par la reconstruction de l'environnement lumineux.

## 2 Repérage dans l'espace

### 2.1 Hiérarchie de référentiels

Afin de mettre en place les différents mécanismes de repérage, il a été nécessaire de hiérarchiser les repères relatifs aux différents référentiels considérés.

3. **R3AM** : <http://liris.cnrs.fr/r3am/>

4. **UCBL** : <http://www.univ-lyon1.fr/>

5. **ENS de Lyon** : <http://www.ens-lyon.eu/>

Le référentiel principal est le référentiel du monde. Il est par définition fixe au cours du temps et est défini par rapport à la mire utilisée pour l'acquisition. Ce repère est placé au centre géométrique de la mire et servira aussi bien à positionner l'objet à afficher qu'à définir le référentiel de base pour l'environnement.

Les différentes faces de la mire sont elles-mêmes fixes dans le repère du monde car liées à l'objet physique (la mire) qui le définit. Les transformations entre le repère de chacune des faces de la mire et le repère du monde sont codées dans les marqueurs présents sur les faces (matrice *model*).

Ainsi, ayant identifié une des faces à l'aide du marqueur présent sur cette dernière, il est possible, en appliquant la transformation codée par ce marqueur, de reconstruire le repère du monde.

L'interface d'acquisition est elle aussi positionnée vis-à-vis du repère principal. La transformation entre ces deux repères est codée par la matrice *view* et changera au cours du temps, l'utilisateur se déplaçant par rapport à la mire.

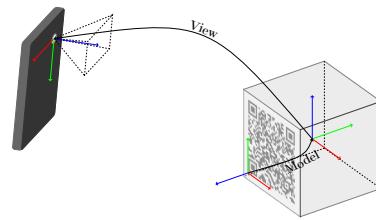


FIGURE 1: Les différents référentiels

Les différentes caméras présentes sur l'interface d'acquisition seront à leur tour positionnées relativement à l'interface d'acquisition. Cette dernière transfor-

mation est disponible dans les fichiers de configuration de caméra chargés au démarrage.

<b>QRcode</b>		lesquelles au moins une devra, pour toute orientation, être lisible. Les marqueurs constituants cette mire doivent par ailleurs être capables de porter les informations caractéristiques de la matrice <i>model</i> associée.
	↓	
<b>Face</b>		Il est donc nécessaire de choisir un modèle de marqueurs permettant à la fois une localisation précise d'au moins trois points (nécessaires au repérage) et portant des informations propres.
$model^{-1}$		Parmi les nombreux modèles de marqueurs répondant au pré-requis, le choix d'une implémentation s'est rapidement porté sur un modèle utilisant des QR codes <sup>6</sup> . L'adaptation à un autre modèle est particulièrement aisée puisqu'elle ne demande que le développement d'un module de scanner reconnaissant les marqueurs (voir section A.1, page III).
<b>Monde</b>		Sont disponibles en annexe les <i>projections</i> des mires cubiques (figure 17, page VI) et hémisphériques octogonales (figure 18, page VIII) construites à base de QR codes.
$view^{-1}$		
<b>Smartphone</b>		
$orientation^{-1}$		
<b>Caméra</b>		
$cvToGl^{-1}$		
<b>Vue OpenGL</b>		
$projection^{-1}$		
<b>Image rendue</b>		

TABLE 1: Hiérarchie des matrices de transformations

## 2.2 Utilisation de marqueurs

L'évaluation des transformations, qui est nécessaire au positionnement dans la scène et donc à la reconstruction de l'environnement, nécessite la reconnaissance de marqueurs. L'identification de ces marqueurs, fixes dans l'espace du monde, associée à la connaissance de la matrice *model* correspondante permettent de calculer l'orientation au regard de la scène.

Afin de permettre la reconnaissance de la scène dans sa globalité, il est nécessaire de construire une mire complète, disposant de plusieurs faces parmi

lesquelles au moins une devra, pour toute orientation, être lisible. Les marqueurs constituants cette mire doivent par ailleurs être capables de porter les informations caractéristiques de la matrice *model* associée.

Il est donc nécessaire de choisir un modèle de marqueurs permettant à la fois une localisation précise d'au moins trois points (nécessaires au repérage) et portant des informations propres.

Parmi les nombreux modèles de marqueurs répondant au pré-requis, le choix d'une implémentation s'est rapidement porté sur un modèle utilisant des QR codes<sup>6</sup>. L'adaptation à un autre modèle est particulièrement aisée puisqu'elle ne demande que le développement d'un module de scanner reconnaissant les marqueurs (voir section A.1, page III).

Sont disponibles en annexe les *projections* des mires cubiques (figure 17, page VI) et hémisphériques octogonales (figure 18, page VIII) construites à base de QR codes.

## 2.3 Évaluation des transformations

Compte tenu de la hiérarchie de référentiel décrite dans la section précédente, il est nécessaire, pour décrire complètement le système, de décrire chacune des matrices de transformation.

**Calibration de la caméra** L'étape de calibration de caméra permet de calculer les différents termes de la matrice *projection* propre à l'interface d'acquisition. Cela revient à ajuster le modèle de sténopé considéré dans la section ?? avec les paramètres physiques.

La matrice *projection* est de la forme suivante :

---

6. **QRCode** : <http://www.qrcode.com/>

$$\begin{pmatrix} \alpha_x & \gamma & u_0 \\ 0 & \alpha_y & v_0 \\ 0 & 0 & 1 \end{pmatrix} \quad (1)$$

avec

$$\begin{aligned} \alpha_x &= f * m_x \\ \alpha_y &= f * m_y \end{aligned}$$

où  $f$  est la longueur focale,  $m$  est la résolution du capteur selon les axes  $\vec{x}$  et  $\vec{y}$ ,  $u_0$  et  $v_0$  sont les coordonnées du centre optique sur le capteur et  $\gamma$  est un coefficient qui décrit la non orthogonalité des axes principaux.

Le calcul de cette matrice est résolu par la minimisation d'un système linéaire sur de nombreuses prises de vue du marqueur.

Cette étape de calibration est implémentée par la bibliothèque OpenCV<sup>7</sup>.

Une fois déterminés, les paramètres relatifs à la caméra sont sauvegardés dans un fichier xml.

**Positionnement à partir de marqueurs** Une fois la caméra calibrée, c'est-à-dire une fois les paramètres intrinsèques déterminés il est possible d'utiliser la connaissance de la matrice *projection* et des dimensions du marqueur pour reconstruire, à partir des coordonnées du marqueur dans l'espace image, la matrice de transformation *extrinsic* correspondant au passage de l'espace du QR code (face de la mire) à l'espace de la caméra.

Les matrices *model* et *orientation* étant connues, la première grâce aux informations écrites sur le QR code et la seconde étant pré-enregistrée (fixe, liée à la géométrie du téléphone), il est possible d'en déduire la matrice *view*

qui caractérise le positionnement dans la scène :

$$\begin{aligned} view &= orientation^{-1} \\ &\times extrinsic \\ &\times model^{-1} \end{aligned} \quad (2)$$

**Models OpenCV / OpenGL** Les différentes bibliothèques ont chacune leurs approches qui impliquent des différences de conventions.

OpenCV considère les images comme des matrices. À ce titre l'origine est placée en haut à gauche, avec le vecteur  $\vec{x}$  pointant vers la gauche et le vecteur  $\vec{y}$  pointant vers le bas. Ainsi regarder une image revient à regarder selon l'axe des  $\vec{z}$  croissants.

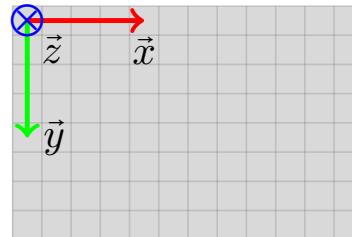


FIGURE 2: Convention d'orientation OpenCV

À l'inverse OpenGL s'intéresse à des espaces 3D complets et oriente donc naturellement le vecteur  $\vec{y}$  vers le haut. L'utilisateur regarde donc selon l'axe des  $\vec{z}$  décroissants.

Afin d'utiliser conjointement les deux bibliothèques, il convient de faire les transformations adéquates.

$$cvToGl = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \quad (3)$$

---

7. OpenCV : <http://opencv.org/>

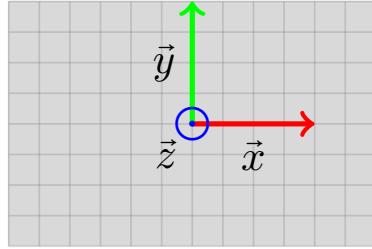


FIGURE 3: Convention d'orientation OpenGL

### 3 Les cartes d'environnement

#### 3.1 Principe

#### 3.2 Reconstruction dynamique

#### 3.3 Détails d'implémentation

### 4 Rendu

#### 4.1 État de l'art

#### 4.2 Éclairage ambiant

#### De la physique aux mathématiques

Le calcul de l'énergie reçue en un point de l'objet revient à intégrer le flux lumineux sur l'ensemble des directions visibles.

$$\mathcal{E}(p, \vec{n}) = \frac{1}{\pi} \int_{\mathcal{H}^2(\vec{n})} \mathcal{L}(p, \vec{\omega}) \times \vec{\omega} \cdot \vec{n} d\vec{\omega} \quad (4)$$

L'intégrale selon  $\int_{\mathcal{H}^2(\vec{n})} d\vec{\omega}$  correspond à une intégrale suivant l'hémisphère visible et pondérée par l'angle solide en supposant qu'il n'y a pas d'occultation. Afin de modéliser les phénomènes d'auto-occultation on devrait utiliser la formule suivante

$$\mathcal{E}(p, \vec{n}) = \frac{1}{\pi} \int_{\mathcal{V}(p, \vec{n})} \mathcal{L}(p, \vec{\omega}) \times \vec{\omega} \cdot \vec{n} d\vec{\omega} \quad (5)$$

où  $\mathcal{V}(p, \vec{n})$  est la restriction de l'hémisphère suivant le vecteur  $\vec{n}$  à l'espace effectivement visible depuis le point  $p$  (ce qui revient à considérer  $\mathcal{H}^2(\vec{n})$  privé des directions correspondants à de l'auto-occultation).

Ici on fait d'abord la supposition que l'éclairage selon une direction  $\vec{\omega}$  est indépendant du point considéré. Cette approximation est nécessaire pour utiliser, sans reconstruction 3D complexe, l'enmap reconstruite dynamiquement.

On obtient donc l'équation

$$\mathcal{E}(p, \vec{n}) = \frac{1}{\pi} \int_{\mathcal{V}(p, \vec{n})} \mathcal{L}_{glob}(\vec{\omega}) \times \vec{\omega} \cdot \vec{n} d\vec{\omega} \quad (6)$$

On fait alors la supposition que les phénomènes d'auto-occultation influencent de manière moyenne et globale l'énergie reçue en un point. Cela nous permet de revenir à une intégration sur tout le demi-espace  $\mathcal{H}^2(\vec{n})$  en ajoutant simplement un coefficient de  $\mathcal{P}_{\mathcal{V}}(p)$  valant 1 en l'absence d'auto-occultation et 0 pour une auto-occultation totale.

Ce coefficient définit sur la surface de l'objet pourra être pré-calculé et fournit sous forme de texture (voir section 5.1, page 11).

$$\begin{aligned} \mathcal{P}_{\mathcal{V}}(p) &= \frac{\int_{\mathcal{V}(p, \vec{n}(p))} \vec{\omega} \cdot \vec{n} d\vec{\omega}}{\int_{\mathcal{H}^2(\vec{n}(p))} \vec{\omega} \cdot \vec{n} d\vec{\omega}} \\ &= \frac{1}{\pi} \int_{\mathcal{V}(p, \vec{n}(p))} \vec{\omega} \cdot \vec{n} d\vec{\omega} \end{aligned} \quad (7)$$

d'où

$$\mathcal{E}(p, \vec{n}) = \frac{\mathcal{P}_{\mathcal{V}}(p)}{\pi} \int_{\mathcal{H}^2(p, \vec{n})} \mathcal{L}_{glob}(\vec{\omega}) \times \vec{\omega} \cdot \vec{n} d\vec{\omega} \quad (8)$$

Enfin, on fera une dernière approximation suivant la méthode proposée dans [Mcguire et al., 2013] : on considère la fonction  $\mathcal{L}_{glob}$  (décrise par l'enmap)

comme constante sur chaque face de l'envmap. Le niveau de mipmap le moins détaillé nous donne en effet une valeur moyenne pour la face considérée.

Si l'on oublie un instant les problèmes d'auto-occultation et que l'on s'intéresse à l'angle solide décrit par une face du cube centrée au point de vue, un rapide calcul permet d'évaluer l'angle solide décrit par une face complètement visible comme étant

$$\Omega_F = \iint_F \frac{1}{\|\vec{\omega}\|^3} d\omega = \frac{2\pi}{3} \quad (9)$$

L'angle solide sur une face partiellement visible (selon une demi-sphère caractérisée par l'hyperplan de vecteur  $\vec{n}$ ) est alors défini par

$$\Omega_F(\vec{n}) = \iint_F \frac{\text{HS}(\vec{\omega} \cdot \vec{n})}{\|\vec{\omega}\|^3} d\omega \quad (10)$$

Avec HS la fonction de HeavySide définit par :

$$\text{HS}(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{si } x \geq 0 \end{cases}$$

La figure 4 retrace l'évolution de  $\Omega_F(\vec{n})$  selon l'angle  $\theta$  pour des orientations de  $\phi = 0$  et  $\phi = \frac{\pi}{2}$

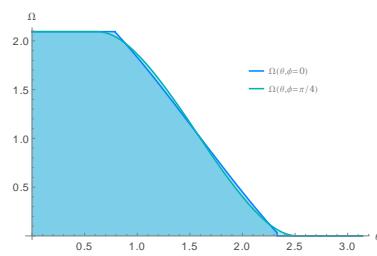


FIGURE 4: Évolution de  $\Omega_F(\vec{n})$  selon l'orientation de  $\vec{n}$

Les poids affectés aux valeurs de chaque face de l'envmap sont donc l'intégration du produit scalaire normalisé  $\frac{\vec{\omega} \cdot \vec{n}}{\|\vec{\omega}\|}$

par rapport à l'angle solide sur la face considérée

$$\begin{aligned} W_F(\vec{n}) &= \iint_F \frac{\vec{\omega} \cdot \vec{n}}{\|\vec{\omega}\|} \times \frac{\text{HS}(\vec{\omega} \cdot \vec{n})}{\|\vec{\omega}\|^3} d\omega \\ &= \iint_F \frac{\vec{\omega} \cdot \vec{n} \times \text{HS}(\vec{\omega} \cdot \vec{n})}{\|\vec{\omega}\|^4} d\omega \quad (11) \end{aligned}$$

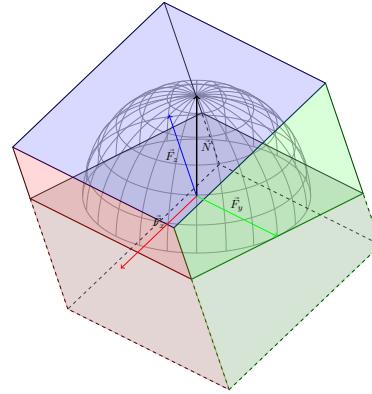


FIGURE 5: Intégration de l'envmap par rapport à une facette

On notera qu'on a bien :

$$\sum_{F \in \text{Faces}} \Omega(F) = 4\pi \quad (12a)$$

$$\forall \vec{n} \sum_{F \in \text{Faces}} \Omega(F, \vec{n}) = 2\pi \quad (12b)$$

$$\forall \vec{n} \sum_{F \in \text{Faces}} W_F(\vec{n}) = \pi \quad (12c)$$

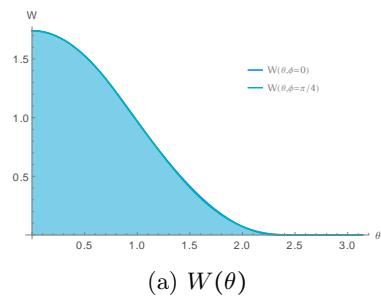
Dès lors, l'approximation selon laquelle  $\mathcal{L}$  est constante sur chaque face nous donne :

$$\begin{aligned} \mathcal{E}(p, \vec{n}) &= \frac{\mathcal{P}_V(p)}{\pi} \sum_{F \in \text{Faces}} \mathcal{L}_{\text{glob}}(\vec{F}) W_F(\vec{n}) \\ &= \frac{\mathcal{P}_V(p)}{\pi} \sum_{F \in \text{Faces}} \mathcal{L}_{\text{glob}}(\vec{F}) \iint_F \frac{\vec{\omega} \cdot \vec{n} \times \text{HS}(\vec{\omega} \cdot \vec{n})}{\|\vec{\omega}\|^4} d\omega \quad (13) \end{aligned}$$

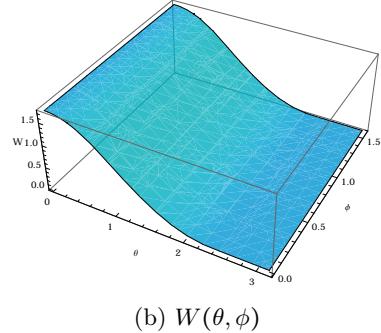
**Approximation numérique** En considérant la face  $F_{Z^+}$  on obtient (sans perte de généralité)

$$W_{F_{Z^+}}(\vec{n}) = \iint_{[-1,1]^2} \left[ \frac{(x.\vec{n}_x + y.\vec{n}_y + \vec{n}_z)}{(x^2 + y^2 + 1)^2} \right. \\ \left. \text{HS}(x.\vec{n}_x + y.\vec{n}_y + \vec{n}_z) dx dy \right] \quad (14)$$

Le problème est alors de trouver un moyen de calculer, ou au moins d'approcher, la fonction  $W_F(\vec{n})$ . Ce calcul étant par ailleurs fait en chaque noeud du maillage, il est primordial de le faire en utilisant un minimum de ressource, quitte à évaluer une valeur approchée qui affectera le résultat de manière faible comparativement avec l'approximation faite précédemment et selon laquelle  $\mathcal{L}$  est constante sur chaque face.



(a)  $W(\theta)$



(b)  $W(\theta, \phi)$

FIGURE 6: Évolution de  $W_F(\vec{n})$  selon l'orientation de  $\vec{n}$

Comme le montre la figure 6, la fonction  $W_F$  dépendant principalement de  $\theta$ , on tentera de l'approximer par une fonction de  $\vec{n} \cdot \vec{F} = \cos(\theta)$

Une approximation simple est la fonction

$$\text{approx} : \cos(\theta) \mapsto \frac{[\max(0.75 + \cos(\theta), 0)]^2}{1.75} \quad (15)$$

Comme le montre la figure 7, cette fonction est, malgré sa grande simplicité, proche de la fonction  $W_F(\vec{n})$ .

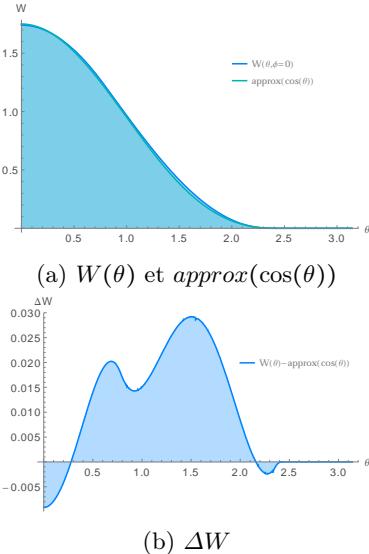


FIGURE 7: Comparaison entre  $W_F(\vec{n})$  et  $\text{approx}(\cos(\theta))$

**Discussion** La méthode développée ici repose sur l'approximation de la formule (13) et plus particulièrement de l'intégrale présentée dans la formule (11). La complexité de cette intégrale réside dans le domaine effectif d'intégration, conditionné à la fois par la forme carree des faces de l'envmap et la présence

d'un terme caractéristique de l'hémisphère visible.

Plusieurs formules ont été étudiée pour des domaines d'intégration hémisphériques ou partiellement hémisphériques ainsi que pour des polygones entièrement visibles [Snyder and Report, 1996] (ce qui fait disparaître le terme de HS).

Il aurait été possible d'adapter un modèle polygonale évoqué dans [Snyder and Report, 1996] mais les polygones sur lesquels il aurait fallu intégrer variant selon les conditions de visibilités, il aurait été nécessaire d'effectuer de coûteux calculs de géométrie.

L'objectif principal étant ici une grande vitesse d'exécution – l'inexactitude des résultats étant de toute façon largement oubliée au regard des approximations faites précédemment – on préféra utiliser une fonction grossièrement approché mais simple à calculer.

### 4.3 Ombrage

**Modèle** Un indice visuel primordial à la vraisemblance visuelle des images produites est la présence d'ombres portées provoquées par l'ajout de l'objet.

Modéliser l'impact d'un tel ajout nécessite théoriquement de connaître la géométrie de la scène et la nature des matériaux qui la compose.

On fera ici plusieurs hypothèses dans le but d'obtenir un modèle qui soit calculable en temps réel tout en donnant des résultats vraisemblables.

Le calcul d'ombres douces est alors fait en décomposant l'objet en une hiérarchie de sphères comme présenté dans [Iwanicki, 2013] et en sommant les contributions des différentes sphères.

Pour évaluer l'ombre douce projetée par une sphère il suffit alors d'évaluer

le manque d'illumination. On rappelle les formules suivantes :

$$\begin{aligned}\mathcal{E}_{ambiant}(p, \vec{n}) &= \frac{1}{\pi} \int_{\mathcal{H}^2(\vec{n})} \mathcal{L}_{glob}(\vec{\omega}) \times \vec{\omega} \cdot \vec{n} d\vec{\omega} \\ \mathcal{E}_{ombre}(p, \vec{n}) &= \mathcal{E}_{ambiant}(p, \vec{n}) - \frac{1}{\pi} \int_{\mathcal{S}(p)} \mathcal{L}_{glob}(\vec{\omega}) \times \vec{\omega} \cdot \vec{n} d\vec{\omega}\end{aligned}$$

avec  $\mathcal{S}(p)$  la portion de sphère visible depuis la point considéré.

L'ombre est rendu en assombrissant le pixel associé ce point d'un facteur :

$$\begin{aligned}\mathcal{F}(p) &= \frac{\mathcal{E}_{ombre}(p, \vec{n})}{\mathcal{E}_{ambiant}(p, \vec{n})} \\ &= 1 - \frac{\int_{\mathcal{S}(p)} \mathcal{L}_{glob}(\vec{\omega}) \times \vec{\omega} \cdot \vec{n} d\vec{\omega}}{\int_{\mathcal{H}^2(\vec{n})} \mathcal{L}_{glob}(\vec{\omega}) \times \vec{\omega} \cdot \vec{n} d\vec{\omega}}\end{aligned}\quad (16)$$

Les différents niveaux de détails de l'enmap nous permettant d'obtenir des approximations de  $L(p, \vec{\omega})$  sur  $\mathcal{S}(p)$  et sur  $\mathcal{H}^2(\vec{n})$ , on peut simplifier la formule en :

$$\mathcal{F}(p) = 1 - \frac{\mathcal{L}_{glob}(\mathcal{S}(p))}{\mathcal{L}_{glob}(\mathcal{H}^2(\vec{n}))} \left( \frac{1}{\pi} \int_{\mathcal{S}(p)} \vec{\omega} \cdot \vec{n} d\vec{\omega} \right)\quad (17)$$

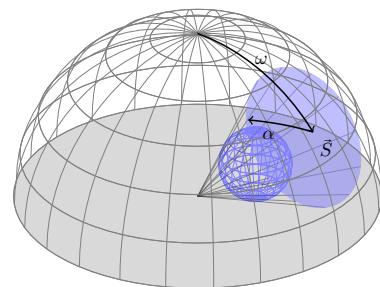


FIGURE 8: Obstruction par une sphère

Le calcul de l'angle solide formé par la sphère, et pondéré par un cosinus,

est détaillé dans [Snyder and Report, 1996]. On retiendra que dans notre cas où la sphère est supposée au dessus du plan sur lequel se projettent les ombres :

$$\int_{\mathcal{S}(p)} \vec{\omega} \cdot \vec{n} d\vec{\omega} = \cos(\omega) \sin^2(\alpha) \quad (18)$$

avec  $\alpha$  le demi angle sous lequel est vu la sphère et  $\omega$  l'angle entre la vertical (normale à la surface sur laquelle se projettent les ombres) et la direction de la sphère.

On obtient ainsi :

$$\mathcal{F}(p) = 1 - \cos(\omega) \sin^2(\alpha) \frac{\mathcal{L}_{glob}(\mathcal{S}(p))}{\mathcal{L}_{glob}(\mathcal{H}^2(\vec{n}))} \quad (19)$$

**Discussion** Là où il est habituel de segmenter l'environnement pour en extraire une hiérarchie de sources lumineuses, la méthode proposée ici permet d'évaluer des ombres douces à partir de données d'environnement sans étape de segmentation ni calcul de visibilité.

Un développement intéressant serait de construire une décomposition hiérarchique, potentiellement intégrée dans un octree, et de l'évaluer plus ou moins profondément selon la distance considérée.

#### 4.4 Reflets spéculaires

Le calcul de l'éclairage ambiant revient à considérer une BRDF purement lambertienne. Afin d'améliorer le réalisme du rendu, il convient d'adopter un modèle de Phong en ajoutant une part d'éclairage spéculaire.

Cet éclairage spéculaire permet de rendre de manière intéressante des surfaces métalliques, dans la limite d'un seul reflet. Le modèle adopté ici est par ailleurs isotrope, ce qui ne permet par

le rendu de matières comme du métal brossé pour lesquels on retrouve des directions privilégiées.

Dans notre cas, l'évaluation du reflet se fait naturellement en calculant la réflexion du rayon incident, donnée par la position du point considéré dans l'espace de la caméra, relativement à la normale de l'objet dans ce même espace. On accédera ensuite à la valeur d'éclairage directement dans l'envmap.

Une fois de plus on pourra utiliser les différents niveaux de mipmap à notre avantage, la considération du niveau de mipmap revenant à considérer l'angle d'un cône autour de l'axe du reflet. Un tel cône permet ainsi de décrire le caractère spéculaire de l'objet, ce dernier pouvant varier d'un reflet parfait – miroir – à un reflet plus diffus – plastique –. On utilisera également l'ombre projetée calculée précédemment afin de moduler les reflets.

#### 4.5 Vers un modèles à micro-facettes

Les modèles employés ici permettent un rendu rapide mais présentent des limitations en terme de qualité. Il serait intéressant, dans l'optique d'améliorer encore la qualité du rendu, de considérer un modèle à micro-facettes. Le principe des modèles à micro-facettes est de considérer la surface de l'objet comme un ensemble de petites faces orientées selon des normales propres à chacune (micro-normales). La distribution statistique des micro-normales autour de la normale géométrique du maillage (macro-normale) permet de déterminer les mécanismes de réflexion [Bruneton et al., 2010] [Dupuy et al., 2012] [Heitz et al., 2013b].

Parmi les nombreux avantages d'une telle méthode, il y a la possibilité de ca-

ractériser des surfaces anisotropes par le biais de directions privilégiées dans la distribution de facettes.

L'auto-occultation entre facettes, variable selon le point de vue, permet par ailleurs de calculer une normal intermédiaire entre la normale géométrique et les micro-normales des facettes (mézo-normal) qui correspond à l'intégration des micro-normales sur l'ensemble des facettes visibles (voir figure 9, page 9).

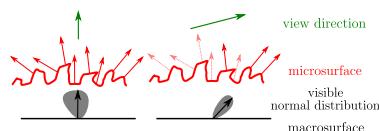


FIGURE 9: Mézo-normales et point de vue

SOURCE [Dupuy et al., 2012]

Cette mézo-normale, utilisée à la place de normale géométrique, permet de corriger les reflets sur des surfaces vue sous un angle important.

L'intégration complète d'un tel modèle a été envisagé de la manière suivante :

1. Choix d'une distribution de normale (gaussienne) et étude du modèle associé ;
2. Ajout aux objets 3D d'une texture (optionnelle) caractérisant localement la direction privilégiée et la force de l'anisotropie associée ;
3. Évaluer, dans le fragment shader, la distribution de direction reflétées en fonction de l'angle de vue et des caractéristiques stockées dans la texture ;
4. Évaluer la lumière incidente selon la distribution calculée précédemment.

Au delà des calculs complexes, mais heureusement déjà documentés, du premier point [Heitz et al., 2013a], l'évaluation de l'envmap selon des distributions anisotropes nécessite de lourds calculs. Il serait possible de les réduire fortement via un pré-calcule (convolution) mais dans notre cas le caractère dynamique de l'environnement ne permet pas une telle approche.

Ici il serait nécessaire d'évaluer, sans pré-calcule, l'intégration anisotrope de l'envmap. Les outils de filtrages anisotropes ne sont ici pas exploitables en l'état car un filtre anisotrope est défini entre autre par un facteur d'anisotropie. Le nombre d'unité de texture étant grandement limité, il n'est pas possible de charger plusieurs fois l'envmap avec des filtrages différents qui couvrirraient toutes nos attentes.

Les outils de lecture de texture par gradient<sup>8</sup> mis en place dans OpenGL ne conviennent pas non plus car même si les vecteurs fournis ( $dX$  et  $dY$ ) sont caractéristiques d'une forte anisotropie, le filtrage anisotropique est en l'état limité aux faces constituant l'envmap cubique. Il en résulte des défauts aux jonctions entre les faces.

Il conviendrait donc, pour mettre en place les mécanismes d'évaluation anisotrope voulus, d'approximer l'intégration via de multiples accès à la texture, le long de la direction privilégiée d'anisotropie pour procéder à une intégration implicite.

#### 4.6 Pipeline

Compte tenu des méthodes décrites précédemment, le pipeline de rendu se décompose en différentes parties (voir figure 10)

8. **textureGrad** : <https://www.opengl.org/sdk/docs/man/html/textureGrad.xhtml>

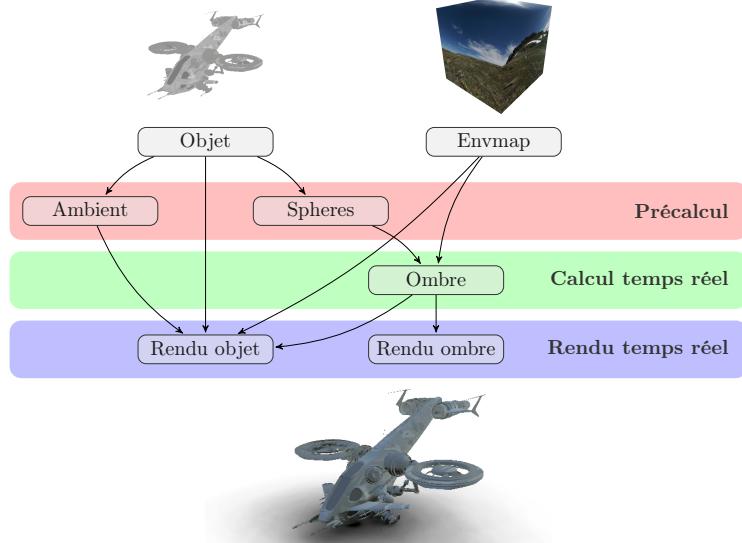


FIGURE 10: Pipeline développé

1. L'étape de pré-calculation permet l'évaluation de données propres au modèle. Ces données n'étant pas influencées par la localisation dans l'espace ni par les caractéristiques d'environnement lumineux, il n'est pas nécessaire de les recalculer en temps réel et on préférera donc stocker les résultats pré-calculés.

**Ambiant** : information d'auto-occultation ( $\mathcal{P}_V(p)$ ), stockée dans une `texture2D` ;

**Sphères** : décomposition de l'objet comme union de sphères, stockée sous forme de `vec4[]` .

2. L'étape de calcul temps réel, qui évalue des résultats temporaires nécessaires à la réalisation du rendu final. Ces résultats doivent être régulièrement évalués dynamiquement car ils dépendent de paramètres dynamiques

tels que les données d'environnement.

**Ombre** : ombre douce projetée par l'objet, elle dépend de l'environnement lumineux décrit par l'envmap.

3. L'étape de rendu qui produit l'image telle qu'elle est vue par l'utilisateur.

**Rendu objet** : affichage de l'objet, en tenant compte de l'éclairage ambiant, et des reflets spéculaires ;

**Rendu ombre** : affichage des ombres en surimpression afin d'intégrer l'objet de manière plus réaliste.

## 5 Pré-calculs

Nous avons détaillé dans le chapitre précédent les différentes étapes de rendu

pour l'intégration d'un objet 3D dans une scène dynamique acquise en temps réel. Certaines des données nécessaires à un tel rendu étant indépendantes de la scène, nous nous sommes proposés de les pré-calculer.

### 5.1 Éclairage ambiant sous forme de texture

Nous avons vu dans la section 4.2, page 4, le calcul de l'éclairage ambiant. Dans ce calcul intervenait un terme d'auto-occultation propre à l'objet 3D considéré.

$$\mathcal{P}_V(p) = \frac{1}{\pi} \int_{V(p, \vec{n}(p))} \vec{\omega} \cdot \vec{n} \, d\vec{\omega} \quad (20)$$

**Intégration statistique** Il s'agit donc d'évaluer, en tout point de l'objet, l'intégrale du produit scalaire entre le vecteur d'intégration et la normale à l'objet, pour un vecteur d'intégration parcourant l'espace visible.

On peut alors reformuler le calcul comme suit :

$$\begin{aligned} \mathcal{P}_V(p) &= \frac{1}{\pi} \int_{V(p, \vec{n}(p))} \vec{\omega} \cdot \vec{n} \, d\vec{\omega} \\ &= \frac{1}{\pi} \int_{H^2(\vec{n}(p))} \vec{\omega} \cdot \vec{n} \times \text{visible}(p, \vec{\omega}) \, d\vec{\omega} \\ &= \frac{1}{\pi} \int_S \vec{\omega} \cdot \vec{n} \times \text{HS}(\vec{\omega}, \vec{n}) \times \text{visible}(p, \vec{n}) \, d\vec{\omega} \end{aligned} \quad (21)$$

Pour évaluer une telle intégrale, on procède par une intégration statistique (Monte Carlo). Pour cela on simule le domaine d'intégration sur l'espace de définition en tirant  $n$  points uniformément sur une sphère. Ces points représentent les vecteurs  $\vec{\omega}$ .

Pour chacun de ces points il s'agit d'évaluer :

$$\vec{\omega} \cdot \vec{n} \times \text{HS}(\vec{\omega}, \vec{n}) \times \text{visible}(p, \vec{n})$$

Le terme  $\text{HS}(\vec{\omega}, \vec{n})$  est en fait inutile car tout point pour lequel  $\text{visible}(p, \vec{n})$  sera non nul sera assuré d'être dans l'hémisphère décrit par  $\text{HS}(\vec{\omega}, \vec{n})$ . La fonction à intégrer peut donc se résumer à :

$$\vec{\omega} \cdot \vec{n} \times \text{visible}(p, \vec{n})$$

**Évaluation de la visibilité** L'évaluation la visibilité est faite à l'aide d'une shadowmap. Étant donné un point de vue (vecteur  $\vec{\omega}$  aléatoire), il s'agit d'effectuer une première étape de rendu qui donnera, dans le Z-buffer, les informations de visibilité (voir figure 11).

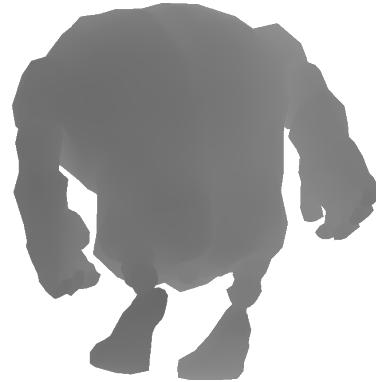


FIGURE 11: Z-buffer issu de la première étape de rendu (shadowmap)

Ces données de profondeur permettent ensuite d'évaluer si un texel est visible ou non, simplement en comparant la profondeur du texel considéré et celui retenu dans la texture et qui caractérise le texel le plus proche selon le rayon associé.

La seconde étape de rendu consiste alors à écrire, dans l'espace texture associé à l'objet, les informations de visibilités qui auront été calculées (voir figure 12). À cette étape, il est important de s'assurer que la texture est bien

écrite, y compris au niveau des jointures entre les différents éléments qui peuvent demander un ajustement. [Aila, 2005] [Manson and Schaefer, 2012]



FIGURE 12: Rendu en espace texture pour une source

**Post-traitement** Une fois évalués pour une source, les résultats sont ajoutés à une texture qui somme les contributions des différentes sources. Cette texture contient ainsi le résultat de l'intégration considérée.

Une fois toutes les sources évaluées, il reste quelques étapes de post-traitement :

- L'intégration étant normalisée, les valeurs calculées sont théoriquement entre 0.0 et 1.0. Cependant du fait de la distribution aléatoire des sources utilisées pour l'intégration, il peut arriver qu'en certains points ne pressentant pas d'auto-occultation la valeur dépasse 1.0. Les valeurs sont alors seuillées afin de ne pas poser de problèmes au moment du stockage au format .png. Les données étant quantifiées entre 0 et 255, un dépassement de la

valeur flottante à stocker peut provoquer une traduction en un entier supérieur à 255. Ces nombres étant stockés sous forme de caractères ASCII, un dépassement à 256 ou 257 peut alors provoquer une évaluation modulo 256 soit un stockage sous forme de 0 ou de 1.

- Les parties de l'image ne correspondant pas à des coordonnées de textures valides étant jusqu'à vides. L'évaluation des niveaux de mipmap pour la texture considérée risque de prendre en compte des éléments qui ne sont pas représentatifs de la réalité géométrique de l'objet. Afin de limiter ce biais dans l'évaluation des niveaux de mipmap, on remplit les parties vides et non représentative avec la valeur moyenne calculée sur les parties représentatives. Ainsi on limite l'incohérence des données considérées en bordure de patch pour les niveaux de mipmap les plus élevés.

A l'issue de ces différents post-traitement, il ne reste qu'à exporter la texture sous forme d'image (voir figure 13) qui sera chargée le moment voulu.

## 5.2 Décomposition en sphères

La section 4.3, page 7, traite du rendu d'ombres portées issues de l'objet 3D ajouté à la scène. Cette étape nécessite une représentation de l'objet en question comme union de sphères.

Le pré-calcul étudié ici consiste donc à construire, étant donné un maillage, un ensemble limites de sphères qui s'ap- pendent au mieux de la surface corres- pondant au maillage.

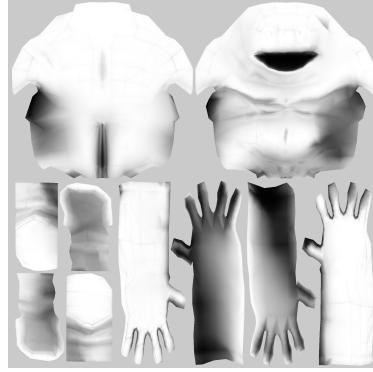


FIGURE 13: Données pré-calculées en espace texture après post-traitement (1000 sources)

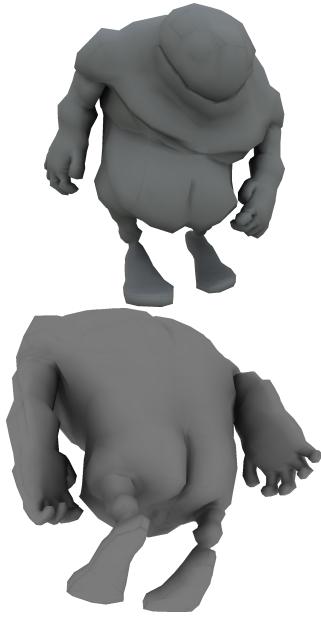


FIGURE 14: Utilisation des données d'éclairage ambiant lors du rendu

Ce problème d'optimisation est longuement développé dans la littérature [Marshall and Martin, 1997] [Gross, 2003], notamment pour la reconstruction de surfaces implicites à partir de points

(orientées ou non). La plupart des méthodes consistent à ajuster des surfaces contraintes à l'aide d'outils statistiques. Ces méthodes présentent cependant le défaut d'être mathématiquement complexes et numériquement complexes.

Le choix s'est donc porté sur une méthode simple et inexacte mais suffisante au regard de l'utilisation des résultats par l'algorithme de calcul des ombrages.

```

Entrées : Maillage  $m$ , int  $n$ , int  $s$ 
/* Échantillonnage du
   maillage */  

Point  $pts[s]$ ;  

pour  $i \leftarrow [1..s]$  faire  

|    $pts[i] \leftarrow$   

|   EchantillonageUniforme( $m$ );  

fin  

/* Initialisation des
   sphères */  

pour  $i \leftarrow [1..n]$  faire  

|    $sphs[i] \leftarrow pts[i];$   

fin  

tant que  $sphs$  n'est pas stable  

faire  

|   Point  $clsts[n][]$ ;  

|   pour  $i \leftarrow [1..n]$  faire  

|   |    $clsts[i] \leftarrow$   

|   |   Appareillement( $pts, sphs[i]$ );  

|   fin  

|   pour  $i \leftarrow [1..n]$  faire  

|   |    $sphs[i] \leftarrow$   

|   |   SphereOptimale( $clsts[i]$ );  

|   fin  

fin  

retourner  $sphs$ ;
```

**Algorithme 1 :** Ajustement de sphères à un maillage

**Échantillonnage du maillage** Afin d'obtenir une distribution uniforme de points à la surface de l'objet, on procède à un échantillonnage par importance<sup>9</sup>.

Pour construire un point uniformément à la surface du maillage, on sélectionne d'abord une face proportionnellement à sa surface avant de choisir un point uniformément sur la face considérée.

En répétant cette opération un grand nombre de fois on obtient un nuage de points qui représente correctement les grandes faces. Ce sont ces points qui seront utilisés par la suite pour la construction des sphères représentatives.

**Appareillement de points et calcul de sphères optimales** La phase d'appareillement de l'algorithme consiste à organiser les points en clusters relatifs aux sphères déjà construites. Il s'agit donc d'établir une métrique qui caractérise la proximité d'un point à une sphère. De la même manière, la phase de calcul de sphères optimales consiste à calculer la sphère qui minimisera la métrique considérée précédemment sur le cluster qui lui a été associé.

La principale difficulté ici est de construire un métrique représentative des critères de qualités voulus, pour laquelle le calcul d'un élément optimal n'est pas trop complexe et qui converge vers un état stable.

*Première approche* Une métrique envisagé est d'évaluer la distance d'un

---

9. Importance sampling

point à la surface de la sphère associé :

$$\begin{aligned} f(P, S) &= \|P - S\|^2 - S.r^2 \quad (22) \\ &= (P.x - S.x)^2 + (P.y - S.y)^2 + (P.z - S.z)^2 - S.r^2 \\ &= (P.x^2 + P.y^2 + P.z^2) + (S.x^2 + S.y^2 + S.z^2) \\ &\quad - 2(P.xS.x + P.yS.y + P.zS.z) - S.r^2 \\ &= \|P\|^2 + \|S\|^2 - 2(P.xS.x + P.yS.y + P.zS.z) - S.r^2 \end{aligned}$$

Ce calcul de la sphère optimal associé a un cluster peut ensuite se faire par l'optimisation d'un système matriciel  $AX - B$ , la matrice  $A$  représentant les données en entrée (points échantillonées), le vecteur  $X$  représentant la solution à optimiser et le vecteur  $B$  étant l'objectif à atteindre.

L'équation 22 nous indique comment construire  $A$ ,  $X$  et  $B$  :

$$A = \begin{pmatrix} -2P_1.x & -2P_1.y & -2P_1.z & 1 \\ -2P_2.x & -2P_2.y & -2P_2.z & 1 \\ \vdots & \vdots & \vdots & \vdots \\ -2P_n.x & -2P_n.y & -2P_n.z & 1 \end{pmatrix} \quad (23a)$$

$$X = \begin{pmatrix} S.x \\ S.y \\ S.z \\ \|S\|^2 - S.r^2 \end{pmatrix} \quad (23b)$$

$$B = \begin{pmatrix} \|P_1\|^2 \\ \|P_2\|^2 \\ \vdots \\ \|P_n\|^2 \end{pmatrix} \quad (23c)$$

Ainsi on a :

$$A.X + B = \begin{pmatrix} f(P_1, S) \\ f(P_2, S) \\ \vdots \\ f(P_n, S) \end{pmatrix} \quad (23d)$$

Cette première considération géométrique n'est cependant pas satisfaisante. La simple proximité entre un point et la surface d'une sphère ne prend pas en compte les différences d'orientation

de normale entre une face du maillage et la sphère qui le modélise, lesquels sont des éléments de qualité essentiels dans notre cas. Au delà de ce problème d'orientation, cette métrique fait aussi face au problème des outliers présents sur les modèles non convexes. Si quatre points extrémaux se retrouvent loin de tout, il est possible qu'une sphère viennent les adopter comme point d'attache, englobant alors tout le modèle et cachant la géométrie que l'on voulait représenter.

Par ailleurs la métrique proposé précédemment ne garanti pas la convergence du système.

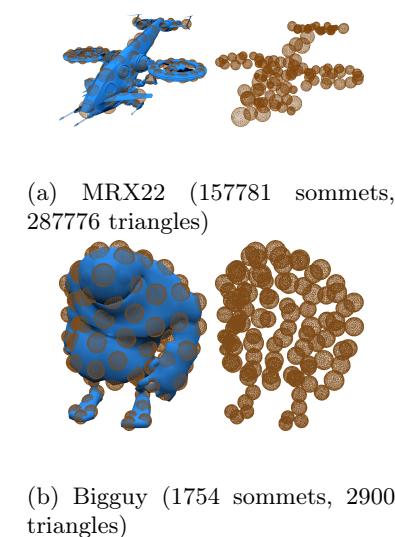


FIGURE 15: Décomposition en sphères d'objets 3D (100000 échantillons, 100 sphères)

*Méthode adoptée* Finalement la solution adoptée, bien que géométriquement pauvre, nous donne des résultats visuellement acceptables tout en garantissant une convergence rapide. L'idée est de

paver la surface du maillage à l'aide de sphères de rayon uniforme, ce qui correspond à la minimisation de la distance entre les points et le centre des sphères qui leur sont associés (voir figure 15).

La distance considérée

$$f(P, S) = \|P - S\| \quad (24)$$

s'optimisant trivialement sur un cluster  $C$  par

$$\overline{S} = \left\{ \left( \bar{p}, \overline{\|p - s\|} \right) \mid p \in C \right\} \quad (25)$$

## 6 Perspectives d'évolution

Au delà des travaux développés au cours de ce stage, certains aspect sont encore en travaux. Les problématiques discutées dans ce chapitre sont sujet à modifications et pourrait faire l'objet de nouveaux travaux dans ma continuité des résultats déjà obtenus.

### 6.1 Acquisition HDR

La méthode de rendu développée utilise les informations le luminosité de la scène. Ces informations, issues de camera standard, ne traduisent pas la réalité physique de la luminosité de la scène mais plutôt la perception que l'on peut avoir de cette scène. Les données obtenus par le capteur photosensible sont en effet traités pour obtenir une image classique.

L'imagerie HDR<sup>10</sup> (High dynamic range), par opposition aux images classique ou LDR (Low dynamic range), vise à considérer une dynamique de luminosité plus large qui permettent de

10. **HDR** : [http://en.wikipedia.org/wiki/High-dynamic-range\\_imaging](http://en.wikipedia.org/wiki/High-dynamic-range_imaging)

représenter les contrastes présent dans la nature.

Les calculs d'éclairement développée dans ce rapport étant fait du point de vue de l'intensité lumineuse reçue par unité de surface, il faudrait théoriquement les appliquer à une envmap HDR qui reflètent les fortes différences de luminosité présentes dans la scène. Cela implique d'acquérir, en temps réel, une vue HDR de la scène.

Les méthodes d'acquisition HDR nécessite généralement plusieurs prise de vues avec différentes expositions qui sont par la suite fusionnées en un unique image HDR. D'autres méthodes proposent, à défaut d'un grand nombres de prise de vue, d'utiliser différentes heuristiques pour reconstruire une image HDR à partir d'une unique image LDR [Rempel et al., 2006].

L'utilisation d'une envmap devrait améliorer les résultats de rendu, notamment pour des scènes présentant de forts contrastes. Le modèle d'ombre douce présenté plus tôt devrai par ailleurs, avec une telle envmap, permettre de reconstruire des ombres assez dures dans le cas d'une source principale comme le soleil.

La principale difficultés viens de la reconstruction de l'image HDR en temps réel, notamment avec des cameras ne permettant pas de régler ou même de fixer les paramètres d'exposition. L'étape de calibration des cameras d'être une étape difficile.

La version 3.0 d'OpenCV<sup>11</sup> devrait voir l'apparition d'outils pour la reconstruction et la manipulation d'images HDR. Il semble intéressant, une fois cette mise à jour disponible, d'étudier les possibilités offertes pour éventuellement les intégrer au module d'acquisition `videodevice_opencv`.

---

11. **OpenCV** : <http://opencv.org/>

## 6.2 BRDF anisotropes

## 6.3 Modèles dynamiques

## A Annexes

### A.1 L'application

#### Généralités

*Obtenir le code source* Le code source de l'application est disponible en ligne sur GitHub<sup>12</sup>. Ce code disponible est sous licence CeCILL-B<sup>13</sup>.

*Installation* Développé en C++11, la compilation de l'application nécessite gcc 4.7 ou supérieure. L'outil cmake est par ailleurs utilisé afin d'automatiser les différentes étapes de la compilation et de l'installation.

Pour plus d'informations sur la procédure d'installation, un fichier README.txt est disponible avec le code.

*Dépendances* Le code développée fait appel à différentes bibliothèques pour remplir des taches spécifiques. À ce titre, l'utilisation de l'application peut demander l'installation préalable des outils utilisés.

**OpenCV** : Regroupant de nombreuses opérations de traitement d'image, OpenCV<sup>14</sup> est utilisé pour la manipulation des images ainsi que pour les différentes étapes de calibration et les calculs matriciels associés.

**Eigen** : Utilisée pour le pré calcul des sphères (voir section 5.2, page 12), Eigen<sup>15</sup> est une bibliothèques de calcul matriciel ne nécessitant pas d'installation préalable.

Le framework gKit, développé au LIRIS<sup>16</sup>, utilisé dans cette application

induit également un certain nombre de dépendances.

**GLEW** : GLEW<sup>17</sup> est un bibliothèques multiplateforme utilisée pour la gestion des extentions OpenGL.

**SDL2** : SDL<sup>18</sup> est une bibliothèque multiplateforme permettant un accès bas niveau aux différentes interfaces utilisateurs ainsi qu'au matériel graphique via OpenGL et Direct3D.

**SDL2 Image** : SDL2\_image<sup>19</sup> est une extension de SDL permettant la gestion d'images dans différents formats.

**SDL2 TTF** : SDL2\_ttf<sup>20</sup> est une extension de SDL permettant la gestion des polices de caractères.

**OpenEXR** : OpenEXR<sup>21</sup> est un dépendance optionnel utilisé pour la gestion des formats d'image HDR.

Enfin la construction des différents modèles induit aussi certaines dépendances supplémentaires.

**OpenCV** : En plus des outils de traitement d'image, OpenCV<sup>22</sup> est utilisé par le module du même nom pour la gestion des interfaces d'acquisition vidéo (webcam).

**ZBar** : La bibliothèque ZBar<sup>23</sup> est utilisé par le module du même nom pour la détection et le décodage des QRCode présents dans les images issues des cameras.

17. **GLEW** : <http://glew.sourceforge.net/>

18. **SDL2** : <http://www.libsdl.org/>

19. **SDL2**

**Image** : [http://www.libsdl.org/projects/SDL\\_image/](http://www.libsdl.org/projects/SDL_image/)

**TTF** : [http://www.libsdl.org/projects/SDL\\_ttf/](http://www.libsdl.org/projects/SDL_ttf/)

21. **OpenEXR** : <http://www.openexr.com/>

22. **OpenCV** : <http://opencv.org/>

23. **ZBar** : <http://zbar.sourceforge.net/>

12. <https://github.com/Amxx/MobileReality>

13. **CeCILL** : <http://www.cecill.info/>

14. **OpenCV** : <http://opencv.org/>

15. **Eigen** : <http://eigen.tuxfamily.org/>

16. **LIRIS** : <http://liris.cnrs.fr/>

**Structure** Écrite en C++11, l’application développée profite de la programmation orientée objet pour organiser le code suivant les différentes fonctionnalités. Afin de rendre l’application modulable, certains composants sont par ailleurs encapsulé suivant des modèles de classes virtuelles et chargé dynamiquement.

**Hiérarchie** Les différentes composants logiciels sont encapsulé dans des objets spécifiques selon le paradigme de programmation orienté objet. La figure 16 (page III) présente cette hiérarchie.

**Core** : La classe **Core** décrit un instance de l’application. Héritant du modèle `gk::App` présent dans `gKit`, elle est créée au lancement du programme et gère l’application graphique ainsi que toute la hiérarchie de classes gérant les différents composants et fonctionnalités.

**Module** : La classe **Module** est une classe mettant en places les différentes méthodes nécessaire au chargement dynamique d’objet pré-compilés correspondant à différents types de structures. Cette classe permet ainsi une grande flexibilité dans le chargement de composants externes.

**VideoDevice** : La classe **VideoDevice** est une classe virtuelle décrivant l’interface acquisition vidéo. Cette classe est implémentée par les modules `videodevice` et `videodevice_uvc` qui constituent l’interface entre l’application et les périphériques d’acquisition vidéo.

**Calibration** : La classe **Calibration** gèrent les différentes valeur caractérisant un objet de type **VideoDevice**. Elle intègre aussi les méthodes de calcul et d’enregistrement / chargement de ces données de calibration.

**Camera** : La classe **Camera** regroupe un objet de type **VideoDevice** ainsi qu’un objet de type **Calibration**. Héritant de la structure virtuelle **VideoDevice**, elle permet de gérer un périphérique vidéo calibré de manière transparente.

**Scanner** : La classe **Scanner** est une classe virtuelle décrivant l’interface relatives aux modules de détection de symboles dans les images. Cette classe virtuelle est implémentée dans le module `scanner_zbar` qui utilise par bibliothèque ZBar<sup>24</sup> pour l’identification de QRCodes.

**Symbol** : La classe **Symbol** décrit les symboles identifiées par les modules de type **Scanner**. Contenant les informations de positionnement ainsi que les données encodées, ils permettent de calculer les données de positionnement (voir section 2.3, page 3).

**Envmap** : La classe **Envmap** encapsules les fonctions nécessaires à la reconstruction de la carte d’environnement, sur GPU, à partir des différentes images issues des cameras ainsi que des données de positionnement correspondantes.

**Occlusion** : La classe **Occlusion** gère les données relatives aux sphères d’occlusion pré-calculées ainsi que les méthodes de construction des ombres.

**Configuration** : La classe **Configuration** contient l’ensemble des paramètres lues dans le fichier de configuration.

*Modules “videodevice”* La classe virtuelle **VideoDevice** décrit les méthodes utilisées par l’application pour communiquer avec les périphériques d’acquisition. Avec de simplifier leur chargement, les implémentations de cette classes sont

24. **ZBar** : <http://zbar.sourceforge.net/>

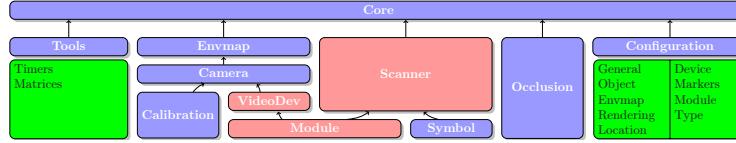


FIGURE 16: Structure de l’application

encapsulées dans un module qui facilite l’instanciation d’un objet à partir de sources pré-compilées.

L’implémentation des différents prototypes relatives à cette API donne donc le schéma suivant :

```

namespace videodevices
{
    class myVideoObject : public Module<VideoDevice>
    {
        public:
            myVideoObject();
            ~myVideoObject();

            bool      open(int idx = 0);
            void      close();
            bool      isopen();
            void      grabFrame();
            IplImage* getFrame();
            IplImage* frame();
            int       getParameter(control);
            void      setParameter(control, int);
            void      resetParameter(control);
            void      showParameters();
        private:
            [...]
    };
}

```

CODE 1: Prototype d’un module VideoDevice

Les méthodes `open`, `close` et `isopen` ont pour rôle de réserver, libérer et indiquer le statut des ressources matérielles. L’entier fournit à la fonction `open`

indique quelle caméra utiliser. Ces indices commencent à 1, l’indice 0 représentant la caméra par défaut.

L’acquisition des images, au format `IplImage`, est fait via les fonctions `grabFrame()`, `getFrame()` et `frame()` :

- La méthode `grabFrame()` récupère l’image courante ;
- La méthode `getFrame()` récupère l’image courante et la renvoie ;
- La méthode `frame()` renvoie la dernière image récupérée.

Enfin, les méthodes `getParameter()`, `setParameter()`, `resetParameter()` et `showParameters()` sont utilisées pour le contrôle des paramètres d’exposition et de contraste de la caméra.

*Modules “scanner”* Les objets correspondant à la classe virtuelle `Scanner` implémentent l’identification des marqueurs présents dans les images fournies par les caméras. L’implémentation des différents prototypes relatives à cette API donne donc le schéma suivant :

La méthode `scan()` prend une image en paramètre et renvoie la liste des marqueurs identifiés.

**Configuration** Le comportement de l’application est configurable par le biais d’un fichier de configuration au format XML (voir code 4, en annexe). Les options sont les suivantes :

**OBJECT** : Paramètres décrivant l’objet à intégrer à la scène ;

```

namespace scanners
{
    class myScannerObject : public Module<Scanner>
    {
        public:
            myScannerObject();
            ~myScannerObject();
            std::vector<Symbol> scan(IplImage*);
        private:
            [...]
    };
}

```

CODE 2: Prototype d'un module Scanner

**obj** : Chemin vers le fichier contenant le maillage (au format .obj);  
**spheres** : Chemin vers le fichier contenant les sphères englobantes (si cette entrée est vide, une seul sphère, construite à partir de la sphère englobante, sera utilisée);  
**scale** : Facteur de mise à l'échelle de l'objet ;  
**DEVICE** : Paramètres décrivant les périphériques à utiliser. **front** représente la camera principale et **back** représente la camera arrière. Pour chacun de ces camera on a les options suivantes ;  
**enable** : Activer / désactiver la camera (désactiver la camera désactive également certains composants pour lesquels elle est indispensable) ;  
**id** : Identifiant de la camera à ouvrir automatiquement ;  
**param** : Chemin vers le fichier de configuration contenant les paramètres de la camera. Si le fichier n'existe pas il sera créé et rempli avec les informations de calibration calculés automatiquement ;  
**MARKERS** : Paramètres des marqueurs ;  
**size** : Taille des marqueur (définit l'unité de distance) ;  
**scale** : Facteur d'échelle entre le marqueur et la mire qui le contient<sup>25</sup> ;  
**GENERALPARAMETER** : Paramètres généraux de l'application  
**verbose** : Niveau de verbosité de l'application ;  
**defaultValues** : Paramètre de luminosité et de contraste. L'option **persistency** indique le nombre d'images pendant lesquels sont conservé les informations de positionnement en cas de perte des marqueurs ;  
**envamp** : Paramètres de l'enmap telle que la définition, l'activation de la construction au démarrage ou le chargement d'une envmap déjà reconstruite ;  
**localisation** : Activer / désactiver les mécanisme de positionnement à partir de marqueurs. Taille de la scène (pour la correction de parallaxe) ;  
**rendering** : Options de rendu (**background** affiche l'arrière plan, **scene** rend l'objet et son ombre porté, **view** est un mode de debug) ;  
**modules** : Chemins vers les sources binaires des modules à utiliser.

## Portage

25. 1.0 pour un marqueur qui occuperai tout la surface, 0.5 pour un marqueur centré représentant 1/4 de la surface

```

<?xml version="1.0"?>
<opencv_storage>
<Camera_Matrix type_id="opencv-matrix">
    <rows>3</rows>
    <cols>3</cols>
    <dt>f</dt>
    <data>
        5.35921936e+02 0. 3.2000000e+02
        0. 5.59243896e+02 2.4000000e+02
        0. 0. 1.</data></Camera_Matrix>
<Distortion_Coefficients type_id="opencv-matrix">
    <rows>1</rows>
    <cols>5</cols>
    <dt>d</dt>
    <data>
        1.2795843004319202e-01 -1.2722157813678139e-01
        -5.3120929078121173e-02 -3.1157837204309542e-02
        4.5978051085070831e-01</data></Distortion_Coefficients>
<Image_Size>
    640 480</Image_Size>
<Root_Mean_Square>3.8408414491248599e-01</Root_Mean_Square>
</opencv_storage>

```

CODE 3: Fichier de calibration

## A.2 Codes et figures

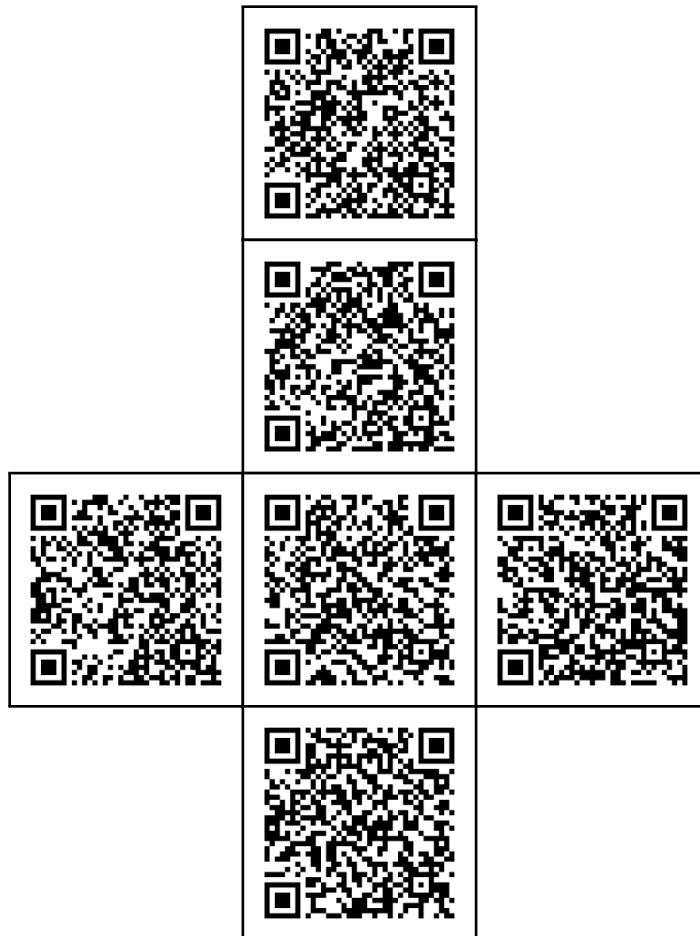


FIGURE 17: Patron de mire cubique

```

<?xml version="1.0"?>
<opencv_storage>

<!-- ##### Object #####
<object>
    <obj>data/objects/bigguy/bigguy.obj</obj>
    <spheres>data/objects/bigguy/bigguy.obj.spheres</spheres>
    <scale>1.0</scale>
</object>
<!-- ##### Devices options #####
<devices>
    <front>
        <enable>on</enable>                                <!-- on / off
        <id>0</id>                                         <!-- [int] / auto / manual
        <params>data/params/logitech.xml</params>
    </front>
    <back>
        <enable>on</enable>                                <!-- on / off
        <id>1</id>                                         <!-- [int] / auto / manual
        <params>data/params/c910-fisheye.xml</params>
    </back>
</devices>
<!-- ##### Markers description #####
<markers>
    <size>6.0</size>
    <scale>0.786667</scale>
</markers>
<!-- ##### General parameters of the application #####
<generalParameters>
    <verbose>2</verbose>                                <!-- [int] / off / low / high
    <defaultValues>
        <brightness>128</brightness>                      <!-- -1 for auto
        <gain>64</gain>                                     <!-- -1 for auto
        <persistence>10</persistence>
    </defaultValues>
    <envmap>
        <path>data/envmap/%03d.jpg</path>                  <!-- [path] / empty for clean
        <build>on</build>                                    <!-- on / off
        <size>512 512</size>
        <dual>off</dual>                                    <!-- on / off
    </envmap>
    <localisation>
        <type>dynamic</type>                                <!-- dynamic / debug
        <size>400.0</size>                                   <!-- [float] (0 for infinity)
    </localisation>
    <rendering>
        <background>on</background>                         <!-- on / off
        <scene>on</scene>                                    <!-- on / off
        <view>off</view>                                     <!-- on / off
    </rendering>
    <modules>
        <scanner>install/share/libscanner_zbar.so</scanner>
        <video>install/share/libvideodevice_uvc.so</video>
    </modules>
</generalParameters>
</opencv_storage>

```

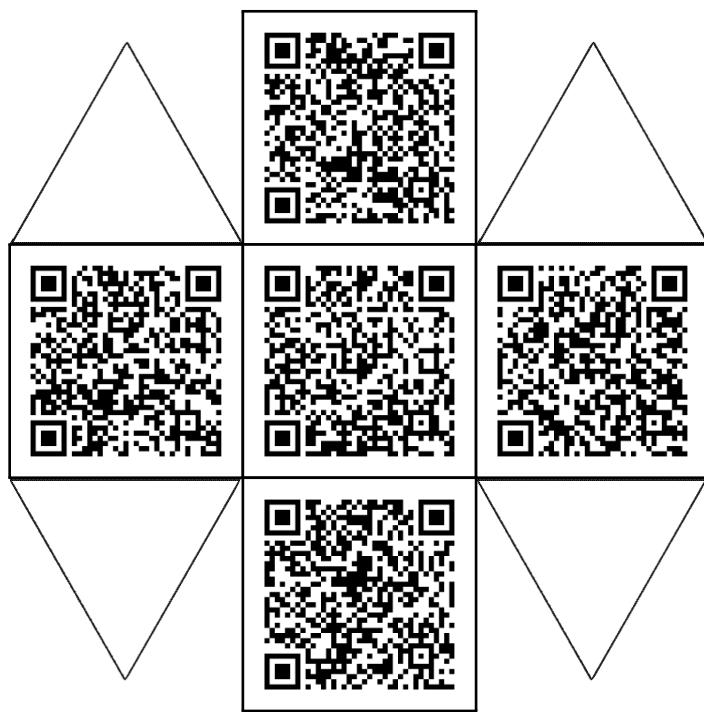


FIGURE 18: Patron de mire hémisphérique octogonale

## Références

- Aila, 2005. Aila, T. (2005). Conservative and Tiled Rasterization Using a Modified Triangle Setup. 10(3) :1–7.
- Akerlund et al., 2007. Akerlund, O., Unger, M., and Wang, R. (2007). Precomputed Visibility Cuts for Interactive Relighting with Dynamic BRDFs. *15th Pacific Conference on Computer Graphics and Applications (PG'07)*, pages 161–170.
- ATI, . ATI. ATI Technologies Inc. Diffuse Cube Mapping. pages 4–6.
- Blinn, 1977. Blinn, J. F. (1977). Models of light reflection for computer synthesized pictures. pages 192–198.
- Brennan, 2002. Brennan, C. (2002). Accurate Environment Mapped Reflections and Refractions by Adjusting for Object Distance. pages 1–6.
- Bruneton et al., 2010. Bruneton, E., Neyret, F., and Holzschuch, N. (2010). Realtime Realistic Ocean Lighting using Seamless Transitions from Geometry to BRDF. *Computer Graphics Forum*, 29(2) :487–496.
- Dupuy et al., 2012. Dupuy, J., Heitz, E., Iehl, J.-c., Poulin, P., Neyret, F., and Ostromoukhov, V. (2012). Linear Efficient Antialiased Displacement and Reflectance Mapping. 32(6) :1–11.
- Gibson et al., 2003. Gibson, S., Cook, J., Howard, T., and Hubbold, R. (2003). Rapid Shadow Generation in Real-World Lighting Environments.
- Gibson et al., 2001. Gibson, S., Howard, T., and Hubbold, R. (2001). Flexible Image-Based Photometric Reconstruction using Virtual Light Sources. *Computer Graphics Forum*, 20(3) :203–214.
- Gross, 2003. Gross, M. (2003). Algebraic Point Set Surfaces.
- Havran et al., 2005. Havran, V., Smyk, M., Krawczyk, G., and Myszkowski, K. (2005). Interactive System for Dynamic Scene Lighting using Captured Video Environment Maps.
- Havran et al., 2003. Havran, V., Smyk, M., Krawczyk, G., Myszkowsky, K., and Seidel, H.-P. (2003). Importance Sampling for Video Environment Maps. (Eurographics) :2003.
- Heitz et al., 2013a. Heitz, E., Nowrouzezahrai, D., Poulin, P., and Neyret, F. (2013a). Filtering color mapped textures and surfaces. *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games - I3D '13*, page 129.

- Heitz et al., 2013b. Heitz, E., Nowrouzezahrai, D., Poulin, P., and Neyret, F. (2013b). Filtering Non-Linear Transfer Functions on Surfaces. *IEEE transactions on visualization and computer graphics*, XX(X) :1–14.
- Heymann et al., 2005. Heymann, S., Smolic, A., and Froehlich, B. (2005). Illumination reconstruction from real-time video for interactive augmented reality. pages 1–4.
- Igehy, 1999. Igehy, H. (1999). Tracing ray differentials. *Proceedings of the 26th annual conference on Computer graphics and interactive techniques - SIGGRAPH '99*, pages 179–186.
- Isidoro, 2005. Isidoro, J. R. (2005). Filtering Cubemaps, Angular Extent Filtering and Edge Seam Fixup Methods.
- Iwanicki, 2013. Iwanicki, M. (2013). Lighting technology of The Last of Us.
- Khan et al., 2002. Khan, E. A., Reinhard, E., and Fleming, R. W. (2002). Image-Based Material Editing 1.
- Křivánek and Colbert, 2008. Křivánek, J. and Colbert, M. (2008). Real-time Shading with Filtered Importance Sampling. *Computer Graphics Forum*, 27(4) :1147–1154.
- Laffont et al., 2012. Laffont, P.-Y., Bousseau, A., and Drettakis, G. (2012). Rich Intrinsic Image Decomposition of Outdoor Scenes from Multiple Views. *IEEE transactions on visualization and computer graphics*, pages 1–16.
- Lopez-Moreno et al., 2010. Lopez-Moreno, J., Hadap, S., Reinhard, E., and Gutierrez, D. (2010). Compositing images through light source detection. *Computers & Graphics*, 34(6) :698–707.
- Madsen and Laursen, 2003. Madsen, C. B. and Laursen, R. (2003). A scalable gpu-based approach to shading and shadowing for photorealistic real-time augmented reality.
- Manson and Schaefer, 2012. Manson, J. and Schaefer, S. (2012). Parameterization-Aware MIP-Mapping. *Computer Graphics Forum*, 31(4) :1455–1463.
- Marshall and Martin, 1997. Marshall, A. D. and Martin, R. R. (1997). Geometric least-squares fitting of spheres, cylinders, cones and tori. pages 1–20.
- Mcguire et al., 2013. McGuire, M., Evangelakos, D., Wilcox, J., Donow, S., and Mara, M. (2013). Plausible Blinn-Phong Reflection of Standard Cube MIP-Maps. pages 1–8.
- Mercier et al., 2006. Mercier, B., Meneveaux, D., and Fournier, A. (2006). A Framework for Automatically Recovering Object Shape, Reflectance and Light Sources from Calibrated Images. *International Journal of Computer Vision*, 73(1) :77–93.
- Oat and Sander, 2006. Oat, C. and Sander, P. (2006). Ambient aperture lighting. *ACM SIGGRAPH 2006 Courses on - SIGGRAPH '06*, page 143.
- P. Debevec, 2005. P. Debevec (2005). A Median Cut Algorithm for Light Probe Sampling.
- Parker et al., 1999. Parker, S., Martin, W., Sloan, P.-P. J., Shirley, P., Smits, B., and Hansen, C. (1999). Interactive ray tracing. *Proceedings of the 1999 symposium on Interactive 3D graphics - SI3D '99*, pages 119–126.
- Rempel et al., 2006. Rempel, A. G., Trentacoste, M., Seetzen, H., Young, H. D., Heidrich, W., Whitehead, L., and Ward, G. (2006). Ldr2Hdr : On-the-fly Reverse Tone Mapping of Legacy Video and Photographs. pages 2–7.
- Sloan and Corporation, . Sloan, P.-p. and Corporation, M. Stupid Spherical Harmonics (SH) Tricks.
- Snyder and Report, 1996. Snyder, J. M. and Report, T. (1996). Area Light Sources for Real-Time Graphics.
- Unger et al., 2008. Unger, J., Gustavson, S., Larsson, P., and Ynnerman, a. (2008). Free Form Incident Light Fields.

*Computer Graphics Forum*, 27(4) :1293–1301.

Wang and Åkerlund, 2009. Wang, R. and Åkerlund, O. (2009). Bidirectional Importance Sampling for Unstructured Direct Illumination. *Computer Graphics Forum*, 28(2) :269–278.

**Table des figures**

[Liste des tableaux](#) [Liste des Algorithmes](#)

[Liste des codes](#)