ÉCOLE **POLYTECHNIQUE**

THESE
pour obtenir le titre de
Docteur en Sciences
Spécialité : Informatique

présentée par :

Samuel Boivin

SIMULATION PHOTORÉALISTE DE SCÈNES D'INTÉRIEUR À PARTIR D'IMAGES RÉELLES

Soutenue le 26 janvier 2001 devant le jury composé de :

Georges Stamon	Président
Eugene Fiume	Rapporteur
Bernard Péroche	Rapporteur
Jos Stam	Examinateur
André Gagalowicz	Examinateur

Table des matières

Table des figures

Chapitre 1

Introduction Générale

1.1 Objectifs de cette thèse

Les objectifs de cette thèse sont multiples. Le premier consiste à développer un logiciel de rendu réaliste rapide et capable de produire des images photoréalistes. Ces deux objectifs antinomiques (rapidité et photo-réalisme) posent plusieurs problèmes, car il est effectivement fréquent qu'une accélération excessive d'un algorithme conduise à des approximations mathématiques et physiques fortes, produisant ainsi des lacunes dans le réalisme de l'image créée. Ayant rencontré ce problème, et bien que notre algorithme soit rapide, nous avons souvent -mais pas toujours- privilégié une précision dans les calculs pour obtenir des images de qualité photoréaliste.
Le second objectif de cette thèse a été de développer une technique complètement novatrice, permettant de regénérer une image de synthèse depuis une image réelle, avec un minimum de données. Nous employons en effet quatre types de données : le modèle 3D de la scène, la position et la géométrie des sources de lumière, la position de la caméra ainsi que ses paramètres intrinsèques, et une **seule image de la scène**. C'est sur ce dernier point que nous insisterons d'ailleurs, car peu de techniques savent retrouver des paramètres de réflectance depuis une seule image. Nous présentons donc ici l'état d'avancement de nos travaux dans ce domaine, et nos différents algorithmes capables en particulier de retrouver des BRDF anisotropes. Nous verrons que notre logiciel entièrement automatique (mais qui peut-être basculé en mode "manuel" si l'utilisateur le souhaite) est en mesure de regénérer des images très proches visuellement de l'image réelle, tout en conservant au mieux les propriétés de réflectance des surfaces : les miroirs sont simulés comme de vrais miroirs et non pas comme de simples textures, et les surfaces anisotropes sont simulées comme telles quand les données le permettent (il faut un minimum de pixels dans l'image pour pouvoir analyser la photométrie des surfaces).
Enfin, un troisième objectif consistait à trouver des applications à notre méthode, et nous en proposons plusieurs, comme par exemple la réalité augmentée .

1.2 Contributions scientifiques et pratiques

Les contributions scientifiques de ce mémoire sont nombreuses. Tout d'abord, notre méthode de recouvrement de paramètres de réflectance est unique en son genre. La méthode employée, hiérarchique et itérative, permet de calculer les réflectances de surfaces particulièrement complexes (par exemple une plaque d'aluminium sur laquelle se réfléchissent des objets texturés), et ce avec une seule image de la scène 3D. Nous avons donc mis au point une technique capable de retouver les réflectances diffuses pures, spéculaires parfaites, à la fois diffuses et spéculaires, isotropes, anisotropes ou texturées.
Dans un autre ordre d'idée, une autre de nos contributions que nous qualifierons de "pratique" est de fournir pour la première fois l'explication complète et détaillée du A-Buffer, mais surtout une démonstration prouvant l'inexactitude d'un fait pourtant souvent considéré comme acquis

dans la littérature infographiste, et qui énonce que l'utilisation du *hardware* dédié de certaines stations permet l'accélération des calculs des facteurs de forme pour la méthode de la radiosité. Nous prouverons à l'aide d'expérimentations que cette affirmation est plus qu'inexacte.

1.3 Organisation de ce mémoire

Ce mémoire de thèse est organisé en deux grandes parties distinctes.

La première partie est consacrée à la physique du rendu et la synthèse d'images. Ainsi, nous expliquerons d'abord dans le chapitre 2 les grandeurs photométriques et radiométriques qui sont généralement employées pour le rendu réaliste, ainsi que les différentes fonctions de réflectance bidirectionnelle (BRDF) que l'on peut trouver dans la littérature scientifique. Nous expliquerons notamment pourquoi nous avons retenu le modèle de BRDF de Ward[236]. Dans le chapitre 3, nous passons en revue plusieurs méthodes pour résoudre l'équation de radiosité et calculer les facteurs de forme. Cette technique de la radiosité est le cœur même de notre logiciel de rendu réaliste, et il est indispensable d'expliquer les bases de l'illumination globale pour comprendre d'où notre logiciel tire son photo-réalisme. Enfin, dans le chapitre 4, nous proposons d'expliquer en détail notre logiciel de rendu réaliste *Phoenix* et quelles sont les raisons qui ont guidé nos choix des différents algorithmes le constituant. Le lecteur pourra aussi trouver les détails complets d'implémentation d'un A-Buffer optimal, ainsi que des comparaisons entre différentes techniques d'accélération (emploi de *hardware* dédié). Quelques photographies d'écran sont également fournies, pour expliquer l'intérêt de l'interface graphique de *Phoenix* et de ses outils d'analyse photométriques.

La seconde partie de ce mémoire développe le rendu réaliste à base d'images réelles. Dans le chapitre 5, nous présentons toutes les techniques existantes pour regénérer une image réelle depuis une ou plusieurs images. Nous verrons ainsi que certaines techniques puissantes emploient plus d'une centaine d'images pour retrouver les BRDF des surfaces, tandis que d'autres avec une seule, ne savent traiter que des surfaces diffuses. Nous verrons aussi que toutes ces méthodes sont classables en deux catégories : celles qui savent prendre en compte l'influence des objets les uns sur les autres (comme la nôtre), et celles qui en font abstraction, mais qui généralement produisent des images moins réalistes.
Le chapitre 6 est notre contribution scientifique majeure, et expose en détails tous nos algorithmes récents et novateurs dans le domaine de la regénération d'images à partir d'images réelles. Nous démontrons en effet que, bien que dans le cas d'une seule image réelle, nous obtenons des résultats supérieurs à ceux obtenus par des techniques en employant plusieurs. Nous verrons aussi dans ce chapitre que nous sommes en mesure de retrouver des BRDF complexes anisotropes sur des surfaces subissant de multiples interactions diffuses ou spéculaires (ou les deux à la fois) avec leurs voisines. En particulier, nous ne nous contenterons pas de montrer des images de synthèse regénérées à côté des images réelles, car nous croyons qu'il est encore plus intéressant d'analyser les cartes d'erreurs issues des différences entre nos images regénérées et les images réelles.
Enfin, dans le chapitre 7, nous présentons tout un ensemble d'images résultats, ainsi que plusieurs applications possibles de notre méthode pour la génération d'effets spéciaux ou la compression de données par exemple.

Première partie

Physique du rendu et synthèse d'images

Chapitre 2

Grandeurs physiques et phénomènes optiques

Les formes sont plus des solides
de lumière que de matière.
René Berger

2.1 Introduction

La synthèse d'images est un domaine où des méthodes puissantes permettent de simuler un environnement, à priori réel, par un autre, virtuel. Mais pour pouvoir réaliser des scènes de qualité photoréaliste, on fait appel en phase finale, après une difficile étape de modélisation géométrique des objets, au **rendu réaliste**. Le rendu réaliste est, pour la simulation de scènes réelles, une étape fondamentale et indispensable du réalisme : elle permet d'approximer des objets réels par des objets simulés en représentant le mieux possible leurs comportements physiques (réflexion, diffusion, réfraction...) face à la lumière et à d'éventuels échanges énergétiques avec d'autres acteurs de l'environnement (un autre objet par exemple).

Pour pouvoir atteindre un tel réalisme, les lois de la physique et en particulier la thermodynamique et l'optique, doivent être les piliers des logiciels de rendu photoréaliste [1]. Nous proposons dans ce chapitre un récapitulatif de toutes les grandeurs physiques et des phénomènes optiques qui nous sont utiles pour la conception d'un tel logiciel. En revanche, nous ne détaillerons pas chacun de ces phénomènes, ni comment ils ont été obtenus à l'origine, ceci pouvant être trouvé dans [190, 27], auxquels sont d'ailleurs largement empruntés les principes physiques décrits plus loin.

2.2 Notions d'angle solide

Durant ce chapitre, nous considérons l'énergie répartie entre deux éléments de surface sphériques le plus souvent. La demi-sphère utilisée plus loin, permettra en fait de représenter les directions et les différents vecteurs dans un système de coordonnées sphériques (voir figure 2.1).

En plus de ce système de coordonnées, nous utilisons un système de coordonnées cartésiennes. L'angle $\theta \in [0, \pi]$ est l'angle entre la direction $\vec{W}$ et l'axe z tandis que $\alpha \in [0, 2\pi[$ décrit l'angle entre la projection de $\vec{\omega}$ sur le plan (xy) et l'axe x. Ainsi un point 3D M sera donné par un ensemble

[1] Nous distinguons dans ce mémoire, les techniques de rendu photoréaliste qui permettent de générer une image de synthèse proche d'une photographie (de référence éventuellement), tandis que les techniques de rendu réaliste tendent à simuler un effet (caustique, flamme, ...) sans pour autant avoir besoin d'une qualité photographique. Cependant, par abus de langage, nous utiliserons ici indifféremment rendu, rendu réaliste ou rendu photoréaliste ; ces trois termes servant à qualifier en fait, le processus final visant à générer une image proche d'une photographie.

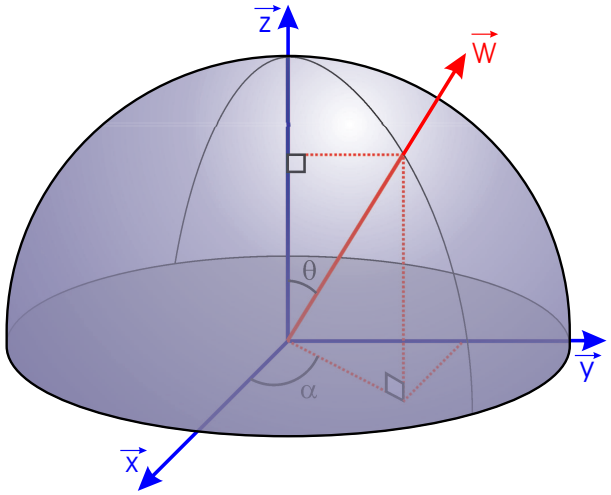

FIG. 2.1 – Système de coordonnées sphériques

de trois composantes scalaires $M = (m_1, m_2, m_3)$ et le vecteur direction $\vec{\omega}$ par une combinaison de deux scalaires (θ, α).

Nous projetons suivant des angles solides (= angles $3D$) et des surfaces sphériques sur la base d'une hémisphère. Si nous considérons $\vec{N}$ comme le vecteur normal à cette base, alors la projection de la surface dA sur celle-ci se notera $dA^{\vec{N}}$, et un angle solide projeté $d\vec{\omega}^{\vec{N}}$ (figure 2.2).

Nous pouvons donc en déduire l'expression de l'aire d'une surface : soient la surface rectangulaire dA de taille $(d\alpha, d\theta)$, r le rayon de l'hémisphère, et $r \cdot d\theta$ la longueur de l'arc sous-tendu par "le coté vertical" de dA (donc la taille de ce côté est $rd\theta$). De même, soient $d\alpha$ l'angle de l'arc sous-tendu par "le coté horizontal" de dA et $r \cdot \sin\theta$ le rayon du "petit" cercle (voir figure 2.2), nous obtenons alors pour la taille du côté horizontal de dA : $r \cdot \sin\theta d\alpha$.
D'où l'aire de dA :

$$dA = (r \cdot \sin\theta d\alpha) \cdot (r \cdot d\theta) = r^2 \cdot \sin\theta d\theta d\alpha$$

L'angle solide élémentaire associé est donc :

$$d\vec{\omega} = \frac{dA}{r^2} = \sin\theta d\theta d\alpha$$

D'où l'aire élémentaire projetée :

$$dA^{\vec{N}} = r^2 \sin\theta\cos\theta d\theta d\alpha$$

et
l'angle solide projeté élémentaire :

$$d\vec{\omega}^{\vec{N}} = \sin\theta\cos\theta d\theta d\alpha$$

De même, si la normale à la surface dA fait un angle θ_A avec la direction $\vec{v}$ de l'angle solide, s'appuyant sur dA (voir figure 2.3) et dont la longueur est r, alors l'angle solide élémentaire $d\omega$ sous lequel le point P voit la surface élémentaire dA, est défini comme :

$$d\omega = \frac{dA\cos\theta_A}{r^2} = \frac{\vec{dA}\cdot\vec{v}}{r^2}$$

2.3 Radiométrie et Photométrie

Dans cette partie, nous allons définir la terminologie de base pour la radiométrie et la photométrie. Il est en effet fondamental de bien comprendre celles-ci pour pouvoir assimiler par la suite le concept de radiosité.

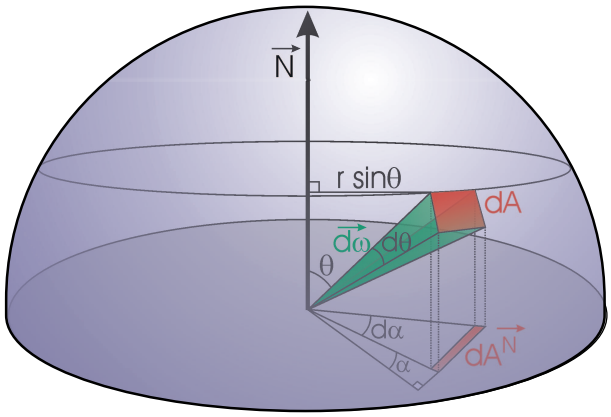

FIG. 2.2 – Surface élémentaire avec son angle solide $d\vec{\omega}$ associé

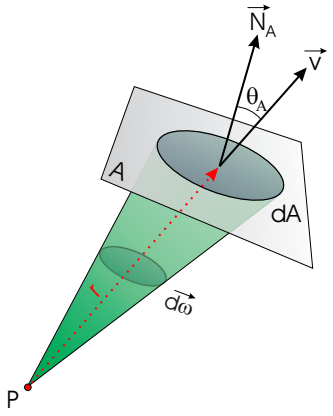

FIG. 2.3 – Angle solide élémentaire $d\omega$ à travers lequel le point P voit la surface élémentaire dA

2.3.1 Différence entre radiométrie et photométrie

La radiométrie est la science de la mesure physique de l'énergie rayonnante. Une mesure radiométrique peut être exprimée pour l'énergie ou la puissance par des unités de mesure du SI (Système International), en Joules ou en Watts respectivement. La valeur de la lumière peut donc être mesurée et le résultat des mesures obtenues s'appelle le **spectre**.

Quant à la photométrie, elle est la mesure physique se rapportant au rayonnement perçu par l'œil ou d'autres capteurs. Nos yeux sont en fait seulement sensibles à un spectre électromagnétique situé entre l'ultraviolet (380nm) et l'infrarouge (770nm). Pour pouvoir différencier la perception visuelle de deux individus, un œil standard moyen normalisé a été établi : il est representé par un ensemble de fonctions d'efficacité lumineuse relative spectrale (encore appelée coefficient de visibilité -à ne pas confondre avec facteur de forme qui n'a aucun rapport- et exprimée en Lumens/Watt), caractérisable pour la vision de jour (ou vision photopique) et pour la vision de nuit (vision scotopique).

Bien qu'ayant des unités différentes, comme nous le verrons plus loin, les quantités photométriques peuvent être calculées à partir des mesures radiométriques spectrales. Pour cette raison, il vaut mieux utiliser des quantités radiométriques pour l'image de synthèse, mais nous donnerons néanmoins leurs homologues photométriques au paragraphe 2.3.4.

2.3.2 Définitions de quantités radiométriques

La quantité basique en radiométrie est l'**énergie électromagnétique** (notée Q et mesurée en Joules).

La quantité d'énergie électromagnétique reçue ou émise par une surface et par unité de temps est le **flux énergétique** et s'écrit :

$$\Phi = \frac{dQ}{dt} \quad (J/s = Watts)$$

Si on se trouve dans le cas où le flux d'énergie reçue Φ_r est incident à une surface élémentaire dA, alors on définit E comme l'**éclairement énergétique** :

$$E = \frac{d\Phi_r}{dA} \quad (W/m^2) \tag{2.1}$$

La quantité d'énergie quittant un point dans la direction Φ, par unité d'angle solide, est appelée l'**intensité énergétique** I :

$$I = \frac{d\Phi}{d\vec{\omega}} \quad (W/sr)$$

On écrit plus généralement l'intensité énergétique comme une fonction de la direction d'émission θ, pour caractériser les sources de lumière non isotropes (comme un laser par exemple). On définit alors un angle solide $d\omega$ ayant pour axe de symétrie la direction θ choisie, comme le montre la figure 2.4. L'intensité énergétique s'écrit alors simplement :

$$I(\theta) = \frac{d\Phi}{d\vec{\omega}} \quad (W/sr) \tag{2.2}$$

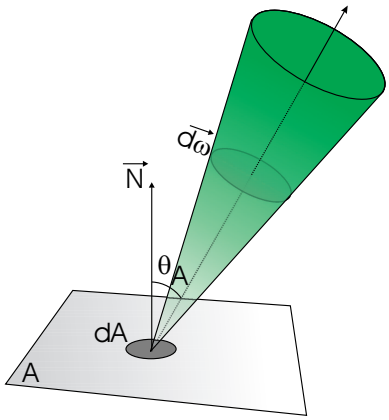

FIG. 2.4 – Intensité énergétique pour une direction d'émission θ

L'intensité énergétique dépend donc de la direction d'émission. La caractérisation de ces sources s'effectue par un diagramme de représentation des émissions autour de π, appelé **indicatrice énergétique de la source**. Quand cette indicatrice est uniforme (l'intensité est identique dans toutes les directions), la source est dite **isotrope** ($I(\theta) = constante = I$). C'est le cas des sources de lumière employées dans des modèles d'illumination simples (comme celui de Phong [159] par exemple).

L'énergie arrivant ou quittant une surface A (découpée en plus petits éléments dA) par unité d'angle solide et par unité d'aire projetée, est appelée la **luminance énergétique** L. Cette quantité est la "plus importante" parmi celles énoncées jusqu'à présent, car la plupart des capteurs optiques et notamment l'œil, y sont sensibles. Elle permet également de comparer des sources étendues de lumière de même intensité, mais de surface émissive différente. Elle se définit par :

$$L = \frac{d^2\Phi}{dA^{\vec{W}}\, d\vec{\omega}} = \frac{d^2\Phi}{dA\, d\vec{\omega}^{\vec{W}}} \quad (W/sr/m^2) \tag{2.3}$$

ou : $L = \frac{dI}{dA^{\vec{W}}}$ ou $L = \frac{dE}{d\vec{\omega}^{\vec{W}}}$

ou encore, pour des sources non uniformes, où la luminance L dépend du point source :

$$L(\theta) = \frac{dI(\theta)}{dA\,\cos\theta} = \frac{d^2\Phi}{d\vec{\omega} dA\cos\theta} \quad (W/sr/m^2) \tag{2.4}$$

Des équations 2.2 et 2.4, il vient :

$$d^2\Phi = L\, d\vec{\omega}\, dA\,\cos\theta = L \underbrace{\frac{dA\,\cos\theta\, dA'\,\cos\theta'}{r^2}}_{\text{étendue géométrique}}$$

Les sources utilisées ici (spot halogène) pour cette thèse, lors de la création des scènes réelles (voir chapitre 6), sont lambertiennes [2].

Le rapport du flux total (flux d'émission propre + flux réémis) rayonné par élément de surface dA est la **radiosité ou émittance**[3]. L'exitance énergétique permet également, comme pour l'intensité, de caractériser des sources pouvant émettre le même flux énergétique mais étant constituées de surfaces émissives différentes. La radiosité s'écrit donc :

$$B = \frac{d\Phi_E}{dA} \quad (W/m^2)$$

$$d\Phi_E = d\Phi_{ep} + \rho \cdot d\Phi_r$$

avec :
ρ fonction de réflectance de la surface[4]
$d\Phi_{ep}$ le flux d'émission propre et $d\Phi_r$ le flux reçu par dA [5].

On peut donc également écrire l'équation 2.3 sous la forme :

$$L = \frac{dB}{d\vec{\omega}^{\vec{W}}} \tag{2.5}$$

L'énergie reçue ou transportée par un faisceau pendant un intervalle de temps t, est la **quantité de lumière**. On peut donc l'écrire comme l'intégrale du flux sur la durée t, et elle s'exprime en *Joules(J)*.
L'énergie reçue par unité de surface pendant une durée t est l'**exposition**. Cette exposition est l'intégrale de l'éclairement de cette surface pendant la durée t, et elle s'exprime en $J \cdot m^2$.

2.3.3 Relations radiométriques

Dans le cas où une surface est émettrice et l'autre réceptrice, plusieurs relations peuvent être établies. Nous proposons un rapide résumé des plus importantes.

$\vec{N}_S, \vec{N}_A$ sont les normales aux surfaces dS , dA (voir figure 2.5). Comme nous considérons $\vec{N}_S$ et $\vec{N}_A$ petits, alors $\|\vec{W}(S,A)\|$, $\vec{W}(S,A)\cdot\vec{N}_S$ et $\vec{W}(S,A)\cdot\vec{N}_A$ sont considérés comme constants en

[2] Les sources dont la luminance est indépendante de la direction d'émission sont dites lambertiennes.

[3] Ces termes de *radiosité* et d'*émittance* sont inconnus des ouvrages français de physique que nous avons consultés [27, 157, 120, 52]. En fait, le rapport du flux total rayonné par élément de surface est connu dans la littérature scientifique française sous le terme d'**exitance énergétique**. Le terme d'émittance est désuet, et ne fut utilisé que de 1954 à 1970 [52].

[4] Les propriétés de réflexion des surfaces sont traitées plus en détail au paragraphe 2.4.

[5] Si la surface n'est pas une source de lumière, $d\Phi_{ep} = 0$.

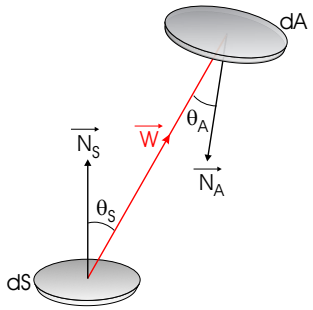

FIG. 2.5 – Géométrie des surfaces

tout point de A et S. Si on appelle $d\vec{\omega}_A$ l'angle solide avec lequel le centre de dS voit dA et $d\vec{\omega}_S$ celui avec lequel le centre de dA voit dS (voir figure 2.6), alors il vient :

$$dS^{\vec{W}}\, d\vec{\omega}_A = [dS\cos\theta_S]\left[\frac{dA\cos\theta_A}{r^2}\right] \quad (2.6)$$

$$= \left[\frac{dS\cos\theta_S}{r^2}\right][dA\cos\theta_A] \quad (2.7)$$

$$= dA^{\vec{W}}\, d\vec{\omega}_S \quad (2.8)$$

($\vec{W}(S,A)$ étant le vecteur joignant S à A).

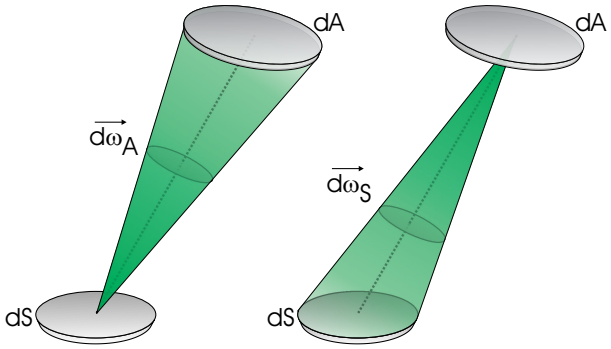

FIG. 2.6 – Réciprocité des angles solides

L'équation 2.8 décrit un principe de réciprocité géométrique entre la source et le récepteur, car les deux termes correspondent en fait à l'angle solide avec lequel dS voit dA et réciproquement. Nous pouvons écrire ici que la luminance L_A arrivant sur dA, fonction de $d\vec{W}_A$, est :

$$dL_A = \frac{d\Phi_A}{d\vec{\omega}_A\, dS^{\vec{W}}}$$

et grâce au principe décrit précédemment, nous obtenons les relations flux-luminance suivantes :

$$d\Phi_A = dL_A\, d\vec{\omega}_A\, dS^{\vec{W}} = dL_A\, dA^{\vec{W}}\, d\vec{\omega}_S$$

D'après les équations 2.1 et 2.8, nous pouvons donc écrire la relation entre la luminance et la densité d'énergie radiante :

$$E_A = \frac{d\Phi_A}{dA^{\vec{W}}} = \frac{dL_A\, d\vec{\omega}_A\, dS^{\vec{W}}}{dA^{\vec{W}}} = \frac{dL_A\, dA^{\vec{W}}\, d\vec{\omega}_S}{dA^{\vec{W}}} = dL_A\, d\vec{\omega}_S \quad (2.9)$$

2.3.4 Définitions de quantités photométriques

Les grandeurs photométriques sont directement dérivables des grandeurs radiométriques, grâce à la définition de la candela : "la candela est l'intensité lumineuse, dans une direction donnée, d'une source monochromatique de fréquence $540 \cdot 10^{12}$Hz, et dont l'intensité énergétique dans cette direction est de 1/683 watts par stéradian" [27]. Ce lien entre l'unité d'intensité énergétique (grandeur radiométrique) et l'unité d'intensité lumineuse (grandeur photométrique) est exprimable sous la forme :

$$I_v \;=\; k_m \cdot \int_0^\infty I_{e,\lambda}(\lambda) \cdot V(\lambda) \cdot d\lambda \tag{2.10}$$

avec :

I_v l'intensité lumineuse[6],

k_m l'efficacité lumineuse maximale en vision photopique[7],

$I_{e,\lambda}(\lambda)$ l'intensité énergétique pour une longueur d'onde λ,

$V(\lambda)$ [8] l'efficacité lumineuse relative spectrale.

On peut donc écrire la relation entre le flux énergétique spectrique $d\Phi_{e,\lambda}$ et le flux lumineux $d\Phi_{v,\lambda}$:
$d\Phi_{v,\lambda} \;=\; k_m \cdot d\Phi e_{,\lambda} \cdot V(\lambda) \;=\; 683 \cdot d\Phi_{e,\lambda} \cdot V(\lambda)$
Si la longueur d'onde vaut 550nm, alors $d\Phi_{v,\lambda}$ vaut simplement : $d\Phi_{v,\lambda} \;=\; 683 \cdot d\Phi_{e,\lambda}$

De même, pour les autres grandeurs radiométriques, nous pouvons établir une table de conversion pour leurs homologues photométriques[9] :

Radiométrie		**Photométrie**	
Grandeur	**Unités**	**Grandeur**	**Unités**
Flux Energétique	W	Flux Lumineux	lm
Intensité Energétique	W/sr	Intensité Lumineuse	$cd\,(lm/sr)$
Luminance Energétique	$W/sr/m^2$	Luminance Lumineuse	$cd/m^2\,(lm/sr/m^2)$
Exitance Energétique	W/m^2	Exitance Lumineuse	lm/m^2
Eclairement Energétique	W/m^2	Eclairement Lumineux	lux

2.3.5 Rapport avec l'image

Lors du rendu réaliste d'une scène synthétique, l'ensemble de ses objets se projette sur l'écran virtuel pour former une image. La quantité radiométrique qui caractérise ce qui se projette en un pixel de cette image est la **luminance**. L'énergie émise par le pixel s'écrit donc : $E_S \;=\; \dfrac{d^2\Phi}{dS}$

En effet, lorsque l'on regarde la surface S, c'est toute l'énergie du rayonnement reçu par l'élément de surface dS qui nous intéresse.
Le flux reçu par le pixel P_k dans la direction de P par angle solide élémentaire $d\vec{\omega_{P_k}}$, et par élément de surface dS_{P_k} (voir figure 2.7) est :

$$I_k(X) \;=\; \frac{d^2\Phi_P}{dS_{P_k}\,\vec{\omega_{P_k}}\,\cos\theta_{P_k}}$$

[6] On utilise généralement comme convention de notation l'ajout d'un "e" ou d'un "v" en indice, suivant qu'il s'agit respectivement, de grandeurs radiométriques ou photométriques.
[7] Par définition, k_m vaut $683\,lm/W$(lumens/watt) à la fréquence de $540 \cdot 10^{12}\,Hz$.
[8] Cette fonction gaussienne possède des valeurs bornées entérinées par le Comité Internationale des Poids et Mesures (CPIM), telles que $V(380\text{nm}) \simeq 0$, $V(550\text{nm}) = 1$ et $V(780\text{nm}) \simeq 0$ en vision photopique.
[9] Le lecteur pourra trouver la traduction anglaise de ces termes en annexe A.

Nous pouvons donc écrire :

$$\underbrace{I^{(i,j)}_{\text{pixel}}}_{\text{intensité lumineuse du pixel (i,j)}} = \int \left[\underbrace{I_k(X)}_{\text{luminance du sous-pixel k}} dS_{P_k} \right]$$

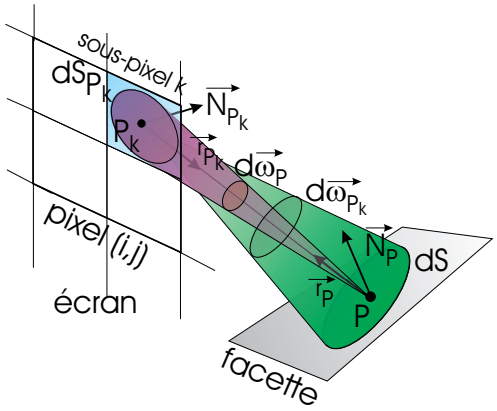

FIG. 2.7 – Relation entre les surfaces et le pixel

$\theta_{P_k} = (\widehat{\vec{N}_{P_k}, \vec{r}_{P_k}})$
$\theta_P = (\widehat{\vec{N}_P, \vec{r}_P})$
$P_k^{i,j}$ = sous-pixel k du pixel i,j
r = distance entre P_k *et* P
$\vec{N}_{P_k}$, $\vec{N}_P$ normales aux surfaces dS_{P_k}, dS
$\vec{r}_{P_k}$, $\vec{r}_P$ directions d'émission des surfaces dS_{P_k}, dS
dS élément de surface de la facette
dS_{P_k} élément de surface du sous-pixel k du pixel (i,j)

2.4 Propriétés de réflexion des surfaces

Afin de permettre une analogie simple avec la littérature anglophone, nous avons adopté le système de notation de [40] pour les fonctions de réflexion.

2.4.1 Définition générale

La réflexion est le processus par lequel le flux électromagnétique incident à une surface, est renvoyé dans l'espace par cette surface, en fonction des propriétés intrinsèques la caractérisant. Ainsi, une porte granuleuse ne réfléchit pas la lumière de la même façon qu'un miroir parfait, et chacune des façons de réfléchir cette lumière est représentable par une fonction dont les paramètres modéliseront le comportement de l'objet face à un faisceau incident.

2.4.2 Fonction de distribution de réflectance bidirectionnelle

La fonction de réflexion la plus connue est la BRDF (Fonction de Distribution de Réflectance Bidirectionnelle[10]) et fut introduite par Nicodemus [147, 148, 149]. Celle-ci se définit comme le

[10] *Bidirectional Reflectance Distribution Function* en anglais. Nous utiliserons indifféremment les termes de *réflectance* et de *réflexion* pour caractériser la propension d'une surface à réfléchir la lumière incidente.

rapport entre le flux réfléchi dans la direction $\vec{\omega}_r$ et le flux incident depuis la direction incidente $\vec{\omega}_i$. On peut donc l'exprimer comme le rapport entre la luminance et l'éclairement (voir figure 2.8). La BRDF [190] est bidirectionnelle car elle dépend à la fois des directions incidentes et des directions réfléchies, et c'est une fonction strictement positive. Par ailleurs, la luminance s'exprimant en $W/sr/m^2$ et l'éclairement en W/m^2, l'unité de mesure de la BRDF devient donc le sr^{-1}. Ses valeurs varient entre 0 et l'infini.
La BRDF s'écrit donc :

$$f_r(\vec{\omega}_i \rightarrow \vec{\omega}_r) \;=\; \frac{L_r(\vec{\omega}_r)}{L_i(\vec{\omega}_i)\cos\theta_i\, d\omega_i} \;=\; \frac{Luminance}{Eclairement} \;=\; \frac{dL}{dE} \tag{2.11}$$

où le dénominateur $L_i(\vec{\omega}_i)\cos\theta_i\, d\vec{\omega}_i$ est en fait l'**éclairement partiel** reçu depuis la direction incidente $\vec{\omega}_i$.

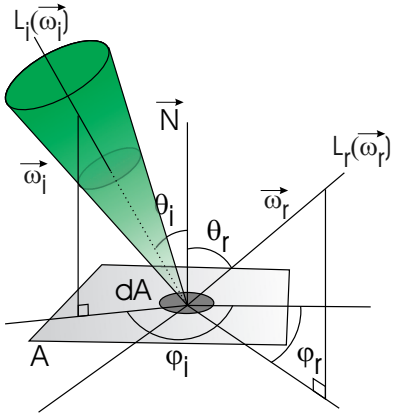

FIG. 2.8 – Fonction de distribution de réflectance bidirectionnelle

Propriétés physiques de la BRDF :

- La BRDF est **symétrique** par rapport aux angles de réflexion et d'incidence. Ce qui signifie que la réflectance pour une direction d'incidence $\vec{\omega}_i$ et une direction de réflexion $\vec{\omega}_r$ est la même que celle définie pour la direction d'incidence $\vec{\omega}_r$ et la direction de réflexion $\vec{\omega}_i$ (principe de réciprocité de Helmoltz) : $f_r(\vec{\omega_r} \rightarrow \vec{\omega_i}) = f_r(\vec{\omega_i} \rightarrow \vec{\omega_r})$.
- La BRDF est, en général, **anisotrope**. Cette propriété caractérise notamment les surfaces rugueuses[11]. Si les directions incidentes et réfléchies à une surface sont fixées, et que cette dernière subit une rotation autour de sa normale, la quantité de lumière qui sera réfléchie peut changer suivant cette direction de rotation (dépendance de $\vec{\omega}_r$ et $\vec{\omega}_i$).
- La BRDF respecte le principe de **conservation d'énergie**[12]. Quelles que soient la quantité d'énergie et la direction incidente à une surface caractérisée par une BRDF, celle-ci ne peut réémettre plus d'énergie qu'elle n'en a reçue.
  On a :

$$\forall\vec{\omega}_i \int_{\Omega_i} f_r(\vec{\omega_i} \rightarrow \vec{\omega_r})\cos\theta_r d\vec{\omega_r} \;\leq\; 1$$

  Cependant, beaucoup de matériaux sont lisses et leur réflectivité ne dépend pas de l'orientation de la surface (on parle alors de surfaces **isotropes**). C'est pour cette raison que les fonctions de réflexion ne changent pas si la surface est tournée, et on a alors :

$$f_r((\theta_i, \varphi_i + \varphi) \rightarrow (\theta_r, \varphi_r + \varphi)) \;=\; f_r((\theta_i, \varphi_i) \rightarrow (\theta_r, \varphi_r))$$

[11] Plus connues sous la dénomination de *glossy surfaces* en anglais.
[12] On pourra trouver la démonstration de cette propriété dans [152] page 26.

2.4.3 Equation de réflectance

La réflectance se comporte de manière linéaire, et la valeur totale de lumière réfléchie par une surface dans une direction spécifique est donnée par une intégrale hémisphérique sur toutes les directions incidentes possibles. On en déduit donc l'**équation de réflectance** suivante :

$$L_r(\vec{\omega}_r) \;=\; \int_{\Omega_i} f_r(\vec{\omega}_i \rightarrow \vec{\omega}_r)\, L_i(\vec{\omega}_i)\, \cos\theta_i \, d\vec{\omega}_i \tag{2.12}$$

2.4.4 Réflectance ρ (ou facteur de réflexion)

La BRDF est très souvent utilisée pour modéliser la réflectance d'un objet, mais ses valeurs résultats variant entre 0 et l'infini, une façon plus simple et plus naturelle de caractériser cette réflexion de la lumière sur une surface, est de définir un *facteur de réflexion*. Ce facteur représente le rapport entre le flux d'énergie réfléchie Φ_r dans une direction $\vec{\omega}_r$ et le flux incident Φ_i depuis une direction $\vec{\omega}_i$, soit le rapport entre l'exitance (ou radiosité[13]) et l'éclairement.
On peut donc écrire la réflectance (ou réflectivité) [190] sous la forme :

$$\rho(\vec{\omega}_i \rightarrow \vec{\omega}_r) \;=\; \frac{\Phi_r(\vec{\omega}_r)}{\Phi_i(\vec{\omega}_i)} \;=\; \frac{\int_{\Omega_r} L_r(\vec{\omega}_r)\cos\theta_r \, d\vec{\omega_r}}{\int_{\Omega_i} L_i(\vec{\omega}_i)\cos\theta_i \, d\vec{\omega}_i} \;=\; \frac{Exitance}{Eclairement} \;=\; \frac{dB}{dE} \tag{2.13}$$

où Ω_r est l'hémisphère des directions réfléchies centré sur la surface.

Relation entre la réflectance ρ et la BRDF :
En remplaçant $L_r(\vec{\omega_r})$ par son expression issue de l'équation 2.12 dans l'équation 2.13, et en supposant que la distribution de lumière incidente est uniforme et isotrope, on peut alors en déduire la définition même de la réflectance :

$$\rho(\vec{\omega}_i \rightarrow \vec{\omega}_r) \;=\; \frac{\int_{\Omega_i}\int_{\Omega_r} f_r(\vec{\omega}_i \rightarrow \vec{\omega}_r)\, d\vec{\omega}_i^{\vec{N}}\, d\vec{\omega}_r^{\vec{N}}}{\int_{\Omega_i} d\vec{\omega}_i^{\vec{N}}}$$

avec :

$$d\vec{\omega}_i^{\vec{N}} = d\vec{\omega}_i \cos\theta_i$$

$$d\vec{\omega}_r^{\vec{N}} = d\vec{\omega}_r \cos\theta_r$$

2.4.5 Réflectance lambertienne diffuse

La réflectance lambertienne diffuse est un cas particulier de BRDF. Dans cette situation précise, l'ensemble des rayons incidents à la surface diffuse est réémis de manière égale suivant toutes les directions appartenant à l'hémisphère centré sur cette surface. Ceci signifie donc que la BRDF est constante, et nous pouvons ainsi illustrer la relation précédemment énoncée à l'aide de l'équation de réflectance (2.12) :

$$L_{r,d}(\vec{\omega}_r) \;=\; \int_{\Omega_i} f_{r,d}\, L_i(\omega_i)\cos\theta_i \, d\vec{\omega}_i \tag{2.14}$$

La BRDF étant constante, on en déduit simplement :

$$L_{r,d}(\vec{\omega}_r) \;=\; f_{r,d} \int_{\Omega_i} L_i(\vec{\omega}_i)\cos\theta_i \, d\vec{\omega}_i \tag{2.15}$$

[13] Bien que le terme de radiosité n'existe pas en physique, comme précisé précédemment, nous l'utiliserons tout au long de ce manuscrit afin de rester cohérent avec la littérature infographiste.

En remplaçant $L_i(\vec{\omega}_i)$ par son expression issue de l'équation 2.4, on obtient :

$$\begin{aligned} L_{r,d}(\vec{\omega}_r) &= f_{r,d} \int_{\Omega_i} \frac{d^2\Phi \cos\theta_i \, d\vec{\omega}_i}{dA \cos\theta_i \, d\vec{\omega}_i} \\ &= f_{r,d} \cdot \frac{d\Phi}{dA} \\ &= f_{r,d} \cdot E \text{ (d'après l'équation 2.1)} \end{aligned}$$

Ainsi, dans le cas d'une surface purement diffuse, la valeur de la luminance réfléchie est proportionnelle à l'éclairement incident et est la même dans toutes les directions.
Par ailleurs, d'après la propriété de conservation d'énergie des BRDF, on sait que les valeurs résultats de la réflectance sont comprises entre 0 et 1.
On a :

$$\rho_d(\vec{\omega}_i \rightarrow \Omega_r) = \frac{\Phi_r}{\Phi_i} \tag{2.16}$$

Or, d'après l'équation 2.1, on a $d\Phi_i = E\,dA$, d'où :

$$\rho_d(\vec{\omega}_i \rightarrow \Omega_r) = \frac{\int_{\Omega_r} L_{r,d}\, dA\, d\vec{\omega}_r^{\vec{N}}}{E\,dA} \tag{2.17}$$

$L_{r,d}$ et dA sont indépendants de Ω_i ($L_{r,d}$ est une constante ici), on obtient alors :

$$\rho_d(\vec{\omega}_i \rightarrow \Omega_r) = \frac{L_{r,d}\, dA \int_{\Omega_r} d\vec{\omega}_r^{\vec{N}}}{E\,dA} \tag{2.18}$$

Or, d'après l'équation 2.11, on a $f_{r,d} = \frac{L_{r,d}}{E}$, donc :

$$\rho_d(\vec{\omega}_i \rightarrow \Omega_r) = \frac{f_{r,d}\, E\, dA \int_{\Omega_r} d\vec{\omega}_r^{\vec{N}}}{E\,dA} \tag{2.19}$$

La somme des angles solides réfléchis valant π, car il s'agit d'une réflexion purement diffuse, l'équation devient :

$$\rho_d(\vec{\omega}_i \rightarrow \Omega_r) = \pi\, f_{r,d} \tag{2.20}$$

Comme la BRDF est une constante, on peut en déduire que la réflectance est aussi une constante dans le cas des surfaces lambertiennes diffuses. Cette relation peut être utilisée pour paramétrer la BRDF en terme de réflectance :

$$f_{r,d} = \frac{\rho_d}{\pi}$$

Par ailleurs, la luminance réfléchie étant constante, et sachant que les surfaces sont ici lambertiennes, on peut déduire de l'équation 2.5 la relation $B = \pi\, L_{r,d}$ en fonction de la luminance. On retrouve donc l'équation 2.13 : $\rho_d = \frac{B}{E}$.

2.4.6 Réflexion spéculaire

La réflexion spéculaire [190, 91, 40] est typiquement ce qui caractérise les miroirs. Ici, nous ne nous intéressons pas au cas des objets miroirs ayant des propriétés de rugosité, comme l'aluminium ou le plastique par exemple. Ces réflexions sont traitées dans le paragraphe suivant.

Dans le cas d'un miroir parfait, l'angle de réflexion est égal à l'angle d'incidence (Loi de Descartes) et le vecteur réfléchi est dans le plan déterminé par le vecteur incident et la normale à la surface.

On en déduit donc :

$$\theta_r = \theta_i$$

$$\varphi_r = \varphi_i \pm \pi$$

De plus, la luminance réfléchie pour un miroir parfait est exactement égale à la luminance incidente :

$$L_r(\theta_r, \varphi_r) = L_i(\theta_i, \varphi_i)$$

On peut donc y associer une BRDF $f_{r,s}$ telle que :

$$f_{r,s} = \frac{\delta(\cos\theta_i - \cos\theta_r)}{\cos\theta_i}\,\delta(\varphi_i - (\varphi_r \pm \pi))$$

avec δ, dirac défini par :

$$\begin{cases} \delta(x) = 0 \text{ si } x \neq 0 \\ \int_{-\infty}^{+\infty} \delta(x)\,dx = 1 \\ \int_{-\infty}^{+\infty} \delta(x-y)\,f(x)\,dx = f(y) \end{cases}$$

2.4.7 Réflexion complexe pour les surfaces rugueuses

La plupart des objets de notre monde réel présentent des propriétés de réflexion de la lumière complexes. Bien que certaines puissent être approximées de façon simple (voir chapitre 6), d'autres nécéssitent une modélisation beaucoup plus fine. Ainsi, une feuille d'aluminium, par exemple, dispose d'une fonction de réflexion complexe, car elle n'est ni parfaitement spéculaire ni parfaitement diffuse mais un peu des deux à la fois, avec en plus une granulosité (ou rugosité) locale.
En fait, ce type de matériau peut être vu comme un ensemble de petits éléments[14] répartis de façon plus ou moins régulière sur sa surface. Du point de vue de la représentation mathématique, plusieurs modèles existent pour caractériser, d'une part la luminance réfléchie d'une surface, et d'autre part, l'aspect géométrique des petits éléments la constituant.

Dans tous les cas, on retrouve souvent la même approximation pour simuler une BRDF. Cette dernière peut être considérée comme une somme de trois termes séparés : la partie diffuse pure, la partie spéculaire pure et la partie rugueuse[15]. Cette approximation est illustrée sur la figure 2.9.

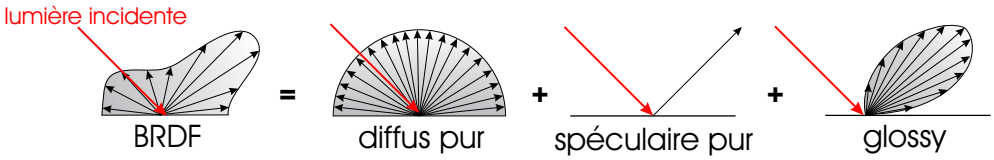

FIG. 2.9 – Approximation de la BRDF par la somme de 3 composantes

Nous avons étudié cinq modèles de BRDF pour la synthèse d'images, en sachant qu'il en existe beaucoup d'autres comme les BRDF de Schlick [184, 185] ou de Lafortune [121] par exemple. Plus récemment encore, d'autres représentations de BRDF ont également été proposées par Ashikhmin [8] ou encore Pellacini [156]. Une étude de différentes BRDF est aussi présente dans [152, 125, 109].

[14] On parle souvent de micro-facettes pour caractériser l'ensemble des petits éléments d'une surface rugueuse.
[15] Nous avons adopté comme convention de notation *rugueuse* pour qualifier *glossy*. Néanmoins, nous espérons que le lecteur ne s'offusquera pas de trouver parfois le terme anglais en lieu et place du terme français.

2.4.8 Modèles de réflexion pour la synthèse d'images

Un modèle empirique de BRDF pour la synthèse d'images : Lambert

Le modèle de réflexion de Lambert [65] est intrinsèquement le plus simple qu'on puisse trouver. Il ne s'applique qu'aux matériaux parfaitement diffus. Bien que ce dernier cas soit extrêmement difficile à atteindre sur des objets réels, nous verrons au chapitre 6, que beaucoup de réflectances de surfaces réelles peuvent être approximées comme telles, pour obtenir un résultat visuel satisfaisant. Le modèle de Lambert s'écrit (voir figure 2.10) :

$$\begin{aligned} f_{lambert,diffus} &= k_d \cdot I_s \, cos\theta \\ &= k_d \cdot I_s \, (\vec{N} \cdot \vec{L}) \end{aligned}$$

avec :

k_d le coefficient de réflexion diffus de la surface,

I_s l'intensité de la source de lumière.

Un autre modèle empirique : Phong

Le modèle de réflectance de Phong [159] date de 1975 et fut le premier à tenter de produire des images réalistes. Encore utilisé aujourd'hui, notamment pour les jeux vidéo (encore qu'il s'agisse d'une version simplifiée[16]), le modèle de Phong présente plusieurs avantages dont les principaux sont sa simplicité (un seul paramètre pour la simulation de la rugosité : n le coefficient de rugosité) et sa facilité à être programmé pour un résultat relativement réaliste.

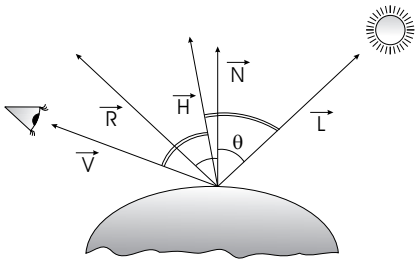

FIG. 2.10 – Géométrie pour le modèle de Phong/Blinn et le modèle de Lambert

La fonction de réflectance pour la partie spéculaire dans le modèle de Phong (voir figure 2.10) s'écrit :

$$\begin{aligned} f_{\text{phong,spec}} &= \Upsilon(\theta) \cdot \cos^n(\alpha) \\ &= k_s \cdot (\vec{R} \cdot \vec{V})^n \end{aligned}$$

avec :

$\Upsilon(\Theta) = k_s$ la fraction de lumière réfléchie par spécularité,

n le coefficient de rugosité simulant un éclat spéculaire.

[16] Dans les jeux vidéo utilisant un modèle d'éclairement avec traitement du spéculaire, on devrait théoriquement procéder à une interpolation des normales, ce qui n'est jamais effectué.

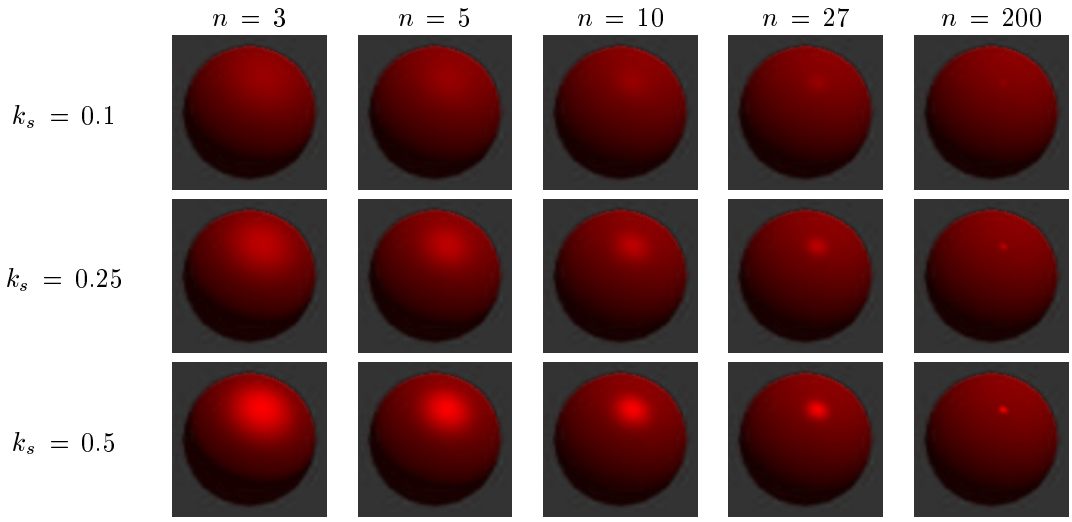

FIG. 2.11 – Influence du coefficent de rugosité n et de spécularité k_s dans le modèle de Phong (nous avons choisi les mêmes valeurs pertinentes que [65]).

Plus le coefficient de rugosité est grand, plus la largeur du lobe de Phong est petite : les éclats spéculaires sont donc plus ou moins étendus et leur intensité est atténuée par le coefficient de spécularité k_s (voir figure 2.11) .

Blinn [22] proposa par la suite une amélioration de ce modèle en remplaçant le terme de spécularité $\vec{R} \cdot \vec{V}$ par $\vec{H} \cdot \vec{N}$ avec $\vec{H}$ le vecteur moyen défini par $(\vec{L} + \vec{V})/2$, soit :

$$f_{\text{phong/blinn,spec}} = k_s \cdot (\vec{H} \cdot \vec{N})^n \tag{2.21}$$

Outre le fait qu'on ne peut traiter les surfaces anisotropes avec ce modèle, celui-ci repose sur des bases physiques très faibles, car il crée de l'énergie lorsque le cosinus devient négatif (Noe [152] présente la méthode du *lobe de cosinus* qui tend à résoudre ce problème en tronquant à zéro les valeurs de cosinus négatives).

Torrance-Sparrow : un modèle plus physique

Le modèle de Torrance-Sparrow [220, 219] fut le premier modèle vraiment physique (fondé sur l'optique géométrique) de réflexion de la lumière pour les surface rugueuses. Blinn l'appliqua -sous une forme simplifiée- à la synthèse d'images [22] et le compara au modèle de Phong précédemment décrit. Cook et Torrance [41, 42] proposèrent une approximation spectrale de la lumière réfléchie en utilisant ce modèle.

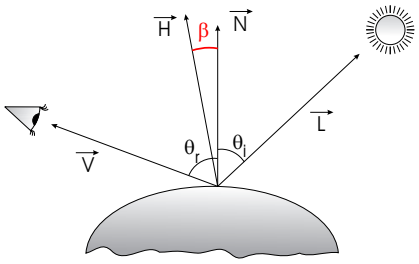

FIG. 2.12 – Géométrie pour le modèle de Cook et Torrance

La forme de la composante spéculaire s'écrit (voir figure 2.12 pour la signification des différents vecteurs et angles) :

$$\begin{aligned} f_{torrance,spec} &= \frac{D \cdot G \cdot F_\lambda}{\cos\theta_i \ \cos\theta_r} \\ &= \frac{D \cdot G \cdot F_\lambda}{(\vec{N} \cdot \vec{L})(\vec{N} \cdot \vec{V})} \end{aligned}$$

D représente une fonction statistique chargée de simuler la distribution des micro-facettes sur la surface. En effet, il est quasiment impossible, notamment pour des raisons de coût mémoire, de décrire une surface rugueuse avec un véritable modèle géométrique 3D (plusieurs millions de facettes seraient nécessaires). On choisit donc d'utiliser plutôt des fonctions de distribution statistiques. Dans l'article d'origine [220], Torrance et Sparrow utilisaient une simple fonction gaussienne comme modèle statistique. Cook et Torrance [41, 42] montrèrent que celle de Beckmann [17][17] était plus adéquate :

$$D = \frac{1}{4\,\pi\,m^2\,\cos^4\,\beta}\, e^{-(\frac{\tan\,\beta}{m})^2}$$

avec :

β l'angle entre $\vec{N}$ et $\vec{H}$,

m le coefficient de rugosité (on pourra trouver son influence dans [41, 42]).

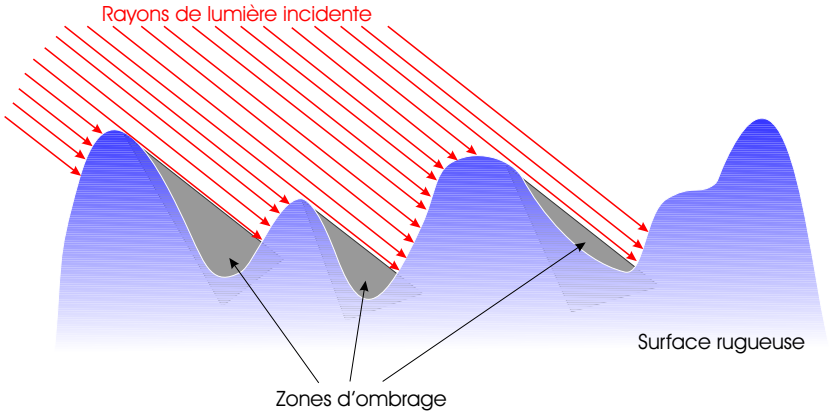

FIG. 2.13 – Phénomène de l'ombrage pour les surfaces rugueuses

On remarquera, par ailleurs, que l'expression originale décrite par Cook et Torrance ne comportait pas le facteur 4 au dénominateur. Il s'agit d'une omission de la part de Cook et Torrance dans les deux articles précédemment cités et qui est signalée par Hall[18] dans son ouvrage [92], et déjà corrigée par Koestner [116].

G est le facteur d'atténuation géométrique. Il s'agit en fait de simuler la propension des microfacettes à s'ombrer et à se masquer les unes des autres[19] (voir figures 2.13 et 2.14.). Ceci est réalisable par plusieurs méthodes comme celles décrites dans [15, 199], par exemple .

[17] Il s'agit en fait d'une réédition du livre [16] de 1963 qui fait référence en la matière.
[18] Hall découvrit cette erreur lors de l'intégration numérique de la fonction de réflexion sur l'hémisphère d'illumination.
[19] Le terme employé par la littérature anglaise est *shadowing/masking*.

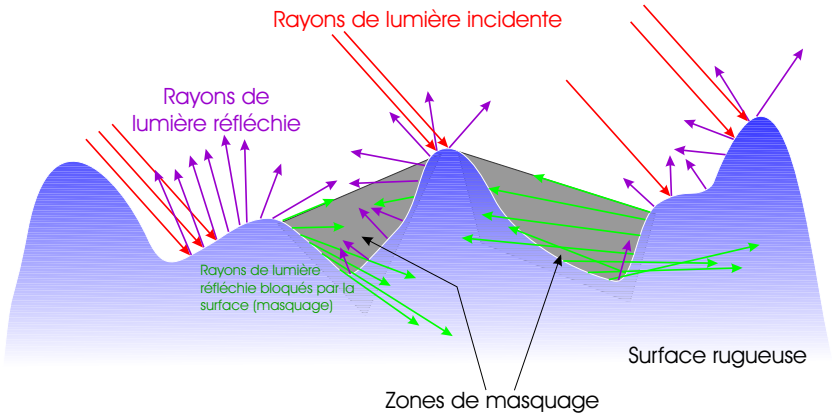

FIG. 2.14 – Phénomène du masquage pour les surfaces rugueuses

Même s'il ne traite, dans cette forme, que les surfaces isotropes, le modèle de Torrance-Sparrow présente l'avantage d'avoir été validé physiquement par des expérimentations : des mesures expérimentales montrent une similitude intéressante entre la réflexion réelle et celle estimée par le modèle [219].

He-Torrance : un modèle physique, mais complexe

Le modèle de réflexion de la lumière décrit par He, Torrance, Sillion et Greenberg [97][20] est, sans doute, le plus rigoureux physiquement à ce jour. Néanmoins, sa complexité le rend moins attractif que ses homologues.

Le modèle HTSG repose d'une part, sur la théorie des ondes électromagnétiques de Kirchhoff[21], et d'autre part, sur les travaux de Beckmann [17]. Plusieurs hypothèses sont sous-jacentes à son utilisation :

- Il ne peut traiter, dans la forme présentée dans l'article, que des surfaces isotropes.
- Les ondes incidentes sont planes et monochromatiques.
- Les surfaces sont caractérisables avec des champs de hauteurs (il s'agit en fait d'une fonction de distribution gaussienne).
- La géométrie de la rugosité de la surface est exprimée comme le rapport de deux grandeurs, σ_0 et τ, correspondant respectivement à l'amplitude des pics et à leur éloignement (distance moyenne entre deux sommets de la surface).
- La longueur d'onde est très inférieure aux dimensions de la surface.

La BRDF décrite par HTSG est approximée par une somme de trois termes différents (figure 2.9) qui sont :

- ρ_{ud} le terme uniforme diffus (diffus pur),
- ρ_{sp} le terme spéculaire pur,

[20] Nous appellerons par la suite ce modèle *HTSG* par souci de compacité.

[21] Nous ne décrirons pas ici en détail la théorie de Kirchhoff, qui est expliquée dans le chapitre 3 du livre de Beckmann et Spizzichino [16, 17].

– ρ_{dd} le terme directionnel diffus[22].

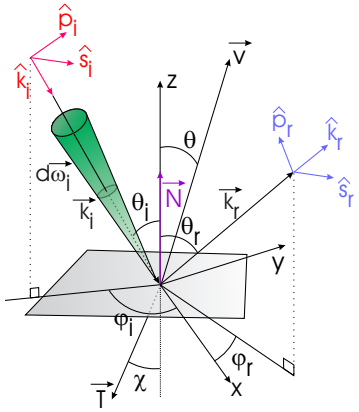

FIG. 2.15 – Géométrie du modèle de He, Torrance, Sillion et Greenberg

On écrit donc le modèle de He, Torrance, Sillion et Greenberg sous la forme :

$$\rho_{htsg} = \rho_{ud} + \rho_{sp} + \rho_{dd} \tag{2.22}$$

avec :

$$\begin{aligned}
\rho_{ud} &= a(\lambda) \text{ un terme ambiant fixé par l'utilisateur} \\
\rho_{sp} &= \frac{\rho_s}{\cos\theta_i \, d\omega_I} \cdot \Delta \\
\rho_s &= F \cdot e^{-g} \cdot S \text{ appelé le coefficient de spécularité de la surface} \\
\Delta &= \begin{cases} 1 \text{ dans le cône spéculaire} \\ 0 \text{ sinon} \end{cases} \\
\rho_{dd} &= \frac{F}{\pi} \cdot \frac{G \cdot S \cdot D}{\cos\theta_i \, \cos\theta_r} \\
F &= \frac{1}{2} \cdot \frac{\sin^2(\theta-\chi)}{\sin^2(\theta+\chi)} \left[1 + \frac{\cos^2(\theta-\chi)}{\cos^2(\theta+\chi)}\right] \text{ le terme de Fresnel pour le vecteur bissectal} \\
&\quad \text{entre } \vec{k_r} \text{ et } \vec{k_i} \text{ (voir figure 2.15) et sachant que } \theta \text{ est lié à } \chi \text{ par la relation } \frac{\sin\chi}{\sin\theta} = \frac{n1}{n2} \\
&\quad (n1 \text{ étant l'indice de l'extérieur de la surface et } n2 \text{ celui de la surface}).
\end{aligned}$$

[22] Généralement connu sous *glossy* dans la littérature anglaise.

$$
\begin{aligned}
G &= \left(\frac{\vec{v}^2}{\vec{v}\cdot\vec{z}}\right)^2 \cdot \frac{\left[(\hat{s_r}\cdot\hat{k_i})^2+(\hat{p_r}\cdot\hat{k_i})^2\right]\cdot\left[(\hat{s_i}\cdot\hat{k_r})^2+(\hat{p_i}\cdot\hat{k_r})^2\right]}{|\hat{k_r}\times\hat{k_i}|^4} \\
\vec{v} &= \hat{k_r}-\hat{k_i} \qquad \hat{s_i} = \frac{\hat{k_i}\times\hat{N}}{|\hat{k_i}\times\hat{N}|} \qquad \hat{s_r} = \frac{\hat{k_r}\times\hat{N}}{|\hat{k_r}\times\hat{N}|} \\
\hat{p_i} &= \hat{s_i}\times\hat{k_i} \qquad \hat{p_r} = \hat{s_r}\times\hat{k_r} \\
S &= S(\theta_i)+S(\theta_r) \text{ la fonction de masquage/ombrage} \\
S(\theta) &= \frac{1-\frac{1}{2}\mathrm{erfc}\left(\frac{\tau\, cot\theta}{2\,\sigma_0}\right)}{\Lambda(cot\theta)+1} \\
\Lambda(\theta) &= \frac{1}{2}\left(\frac{2}{\sqrt{\pi}}\cdot\frac{\sigma_0}{\tau\, cot\theta}-\mathrm{erfc}(\frac{\tau\, cot\theta}{2\,\sigma_0})\right) \\
\mathrm{erfc}(x) &= 1-\frac{2}{\sqrt{\pi}}\sum_{n=0}^{\infty}\frac{(-1)^n\, x^{2n+1}}{n!\,(2n+1)}, \quad \text{la } \textit{fonction d'erreur complémentaire}^{23} \text{ où } \mathrm{erfc}(0)=1. \\
&\quad \text{et } \lim_{x\to\infty}\mathrm{erfc}(x)=0 \\
D &= \frac{\pi^2\,\tau^2}{4\lambda^2}\cdot\sum_{m=1}^{\infty}\left(\frac{g^m\, e^{-g}}{m!\cdot m}\cdot e^{\frac{\tau^2\cdot(u_x^2+u_y^2)}{4m}}\right) \\
\vec{u} &= \frac{2\pi}{\lambda}\cdot(\hat{k_r}-\hat{k_i}) \\
g &= \left[\frac{2\pi\sigma}{\lambda}\cdot(\cos\theta_i+\cos\theta_r)\right]^2 \\
\sigma &= \sigma_0\cdot\sqrt{1+\left(\frac{z_0}{\sigma_0}\right)^2} \\
\sqrt{\frac{\pi}{2}}\, z_0 &= \frac{\sigma_0}{4}(K_i+K_r)\cdot e^{-\frac{z_0^2}{2\sigma_0^2}} \\
K_i &= \tan\theta_i\cdot\mathrm{erfc}(\frac{\tau}{2\sigma_0}cot\theta_i) \\
K_r &= \tan\theta_r\cdot\mathrm{erfc}(\frac{\tau}{2\sigma_0}cot\theta_r)
\end{aligned}
$$

Ce modèle de fonction de réflexion, bien que très proche de la réalité de par sa rigueur physique[24], reste donc très complexe. L'article le décrivant mentionne que cette BRDF ne peut traiter que des surfaces isotropes, néanmoins, nous considérons que cela ne constitue pas une véritable limitation puisque ce modèle est extensible à l'anisotropie sans trop de difficultés [51].

Pour nous, ce modèle présente plusieurs inconvénients, dont le plus gênant est celui du temps de calcul qu'il nécessite pour une surface. En effet, il y a au moins, dans cette BRDF, une série lourde en calcul et qui, de plus, est connue pour ne pas converger très vite[25]. Indépendamment de cela, le nombre de termes à évaluer est important et plusieurs d'entre eux font appel à des fonctions transcendantes que l'on sait aussi très coûteuses en temps machine.

Ceci est complètement antinomique avec le fait que notre logiciel de rendu doit être très rapide (voir chapitre 4) pour pouvoir être employé par l'algorithme de regénération d'images (voir

[23] Cette fonction est directement accessible sous le nom *erfc* dans la librairie mathématique C.

[24] Nous pourrions cependant discuter du fait que toutes les surfaces rugueuses sont supposées avoir une distribution gaussienne de hauteurs, ce qui n'est pas forcément vrai.

[25] Une bonne description détaillée de cet article, de son implémentation et de ses inconvénients, est disponible dans [51].

chapitre 6).

Enfin, Ward [236] signale également que ce modèle n'a pas été comparé avec de véritables mesures de BRDF.

Ward : un modèle simple et puissant

La fonction de réflectance que propose Ward [236] est une BRDF capable de modéliser indifféremment les surfaces isotropes et anisotropes. Les avantages que nous voyons à ce modèle sont multiples. Tout d'abord, la BRDF de Ward est simple et facile à implémenter et ne comporte que peu de paramètres (trois dans le cas isotrope, cinq dans le cas anisotrope) : Yu [251] propose d'ailleurs de l'utiliser pour son algorithme de regénération d'images[26], à travers Radiance [237].

Le second avantage que nous avons trouvé à ce modèle est le fait qu'il traite les surfaces anisotropes. En effet, alors que nous pensions, dans notre analyse des réflectances des surfaces (voir chapitre 6), nous limiter aux cas isotropes, certaines surfaces (en l'occurence une plaque d'aluminium) se sont avérées impossibles à simuler avec un paramètre indépendant de l'orientation de la surface (rotation autour de sa normale). L'utilisation de ce modèle a permis d'y remédier.

Le troisième avantage, et c'est sans doute le plus intéressant pour nous, est le fait que Ward a validé son modèle avec de vraies mesures effectuées avec un appareil spécifique. Le fait que les valeurs α_x et α_y (coefficients de rugosité des surfaces) aient une véritable réalité mesurable, nous a poussé à choisir ce modèle pour notre logiciel de rendu.

Par ailleurs, le principal défaut que nous trouvions à [97] se trouve balayé ici : le modèle de Ward est très rapide à évaluer et il est en plus facilement accélérable (l'ensemble des techniques d'accélération du lancer de rayons est applicable). La qualité des images est également réglable avec le simple paramètre du nombre de rayons à relancer depuis une surface rugueuse.

Enfin Ward propose dans son article une méthode pour échantillonner le cône de réflexion de la BRDF, suivant les deux paramètres principaux d'anisotropie (α_x, α_y).

Contrairement à He,Torrance, Sillion et Greenberg, la BRDF de Ward est approximée comme la somme de deux termes, l'un diffus et l'autre *glossy*. En fait, Ward propose plus qu'une simple BRDF : il décrit en effet une équation de luminance, à laquelle il ajoute d'une part, la BRDF (qui lui sert en réalité à simuler les éclats spéculaires) et d'autre part, un terme de spécularité pure et un terme issu de la résolution de l'équation de radiosité.

L'équation d'illumination s'écrit donc :

$$L(\theta_i,\varphi_i,\theta_r,\varphi_r) = \underbrace{I\cdot\frac{\rho_d}{\pi}}_{\text{Terme diffus}} + \underbrace{L_s\rho_s}_{\text{Terme spéculaire}} + \underbrace{\sum_{i=1}^{N} L_i\omega_i\,\cos\theta_i\, f_{bd,ward}(\theta_i,\varphi_i,\theta_r,\varphi_r)}_{\text{Terme } \textit{glossy} \text{ pour les éclats spéculaires}} \tag{2.23}$$

avec :
I l'éclairement au point i résultat du calcul de radiosité,
ρ_d la réflectance diffuse de la surface,
ρ_s la réflectance spéculaire de la surface,
L_s la luminance dans la direction d'échantillonnage stochastique donnée en fonction de deux angles θ et Φ évalués par les équations 2.35 et 2.36,

[26] Le terme de *rerendering* ou *image-based rendering* n'a pas d'équivalent direct en français. Nous proposons comme traduction *regénération d'images*, bien que nous employons le terme anglais tout au long de cet ouvrage pour rester cohérent avec la littérature scientifique de ce domaine.

L_i la luminance de la source de lumière i,
N le nombre de sources de lumière,
ω_i l'angle solide entre la surface et la source de lumière i,
$f_{bd,ward}$ la fonction de réflectance bidirectionnelle, elliptique gaussienne, équation 2.24 pour le cas isotrope et 2.25 pour le cas anisotrope,
$\theta_i, \varphi_i, \theta_r, \varphi_r$ des angles définis d'après la figure 2.16.

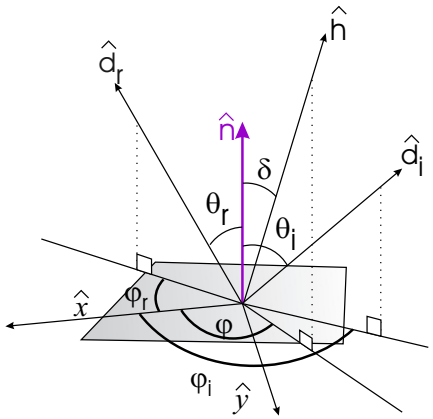

FIG. 2.16 – Géométrie du modèle de Ward. On remarquera que $\hat{x}$ représente en fait la direction des stries pouvant apparaître sur une surface rugueuse : ce paramètre n'est utile que pour les surfaces anisotropes.

$$f_{bd,iso,ward}(\theta_i, \varphi_i, \theta_r, \varphi_r) = \frac{\rho_d}{\pi} + \rho_s \cdot \frac{1}{\sqrt{\cos\theta_i \cos\theta_r}} \cdot \frac{e^{\frac{-\tan^2\delta}{\alpha^2}}}{4\pi\alpha^2} \tag{2.24}$$

avec (voir figure 2.16) :
ρ_d la réflectance diffuse de la surface,
ρ_s la réflectance spéculaire de la surface,
δ l'angle entre $\hat{n}$ et $\hat{h}$,
α le coefficient de rugosité de la surface isotrope (sorte d'inclinaison moyenne des micro-facettes),
$\vec{h} = \hat{d}_r + \hat{d}_i$,
$\hat{h} = \frac{\vec{h}}{\|\vec{h}\|}$.
$\hat{d}_i$ le vecteur de direction incidente.
$\hat{d}_r$ le vecteur de direction réfléchie.

$$f_{bd,aniso,ward}(\theta_i, \varphi_i, \theta_r, \varphi_r) = \frac{\rho_d}{\pi} + \rho_s \cdot \frac{1}{\sqrt{\cos\theta_i \cos\theta_r}} \cdot \frac{e^{-\tan^2\delta\left(\frac{\cos^2\Phi}{\alpha_x^2} + \frac{\sin^2\Phi}{\alpha_y^2}\right)}}{4\pi\alpha_x\alpha_y} \tag{2.25}$$

avec :
ρ_d la réflectance diffuse de la surface,
ρ_s la réflectance spéculaire de la surface,
δ l'angle entre $\hat{n}$ et $\hat{h}$,
α_x le coefficient de rugosité dans la direction $\hat{x}$ de la surface anisotrope (sorte d'inclinaison moyenne des micro-facettes dans le sens des $\hat{x}$),
α_y le coefficient de rugosité dans la direction $\hat{y}$ de la surface anisotrope (sorte d'inclinaison moyenne des micro-facettes dans le sens des $\hat{y}$),
δ l'angle entre $\hat{n}$ et $\hat{h}$,
φ l'angle azimutal entre la projection de $\hat{h}$ sur la surface et $\hat{x}$.
φ_i l'angle azimutal entre la projection de $\hat{d}_i$ sur la surface et $\hat{x}$.

φ_r l'angle azimutal entre la projection de $\hat{d_r}$ sur la surface et $\hat{x}$.

La fonction 2.25 semble lourde à calculer. Dans son article, Ward propose une simplification de son équation sous la forme d'une approximation. Malheureusement, cette approximation est clairement fausse (équation 5b de [236]). Nous nous sommes rendu compte de cette erreur en cherchant à comprendre comment il l'avait obtenue. Par ailleurs, dans Radiance, cette erreur a été corrigée. La bonne équation est donc :

$$f_{bd,aniso,ward}(\theta_i,\varphi_i,\theta_r,\varphi_r) = \frac{\rho_d}{\pi} + \rho_s \cdot \frac{1}{\sqrt{(\hat{d_i}\cdot\hat{n})\,(\hat{d_r}\cdot\hat{n})}} \cdot \frac{1}{4\pi\alpha_x\alpha_y} \cdot e^{-\frac{\left(\frac{\hat{h}\cdot\hat{x}}{\alpha_x}\right)^2+\left(\frac{\hat{h}\cdot\hat{y}}{\alpha_y}\right)^2}{(\hat{h}\cdot\hat{n})^2}} \tag{2.26}$$

Démonstration
Nous nous intéressons en fait au terme exponentiel (le reste de l'équation étant similaire) et nous cherchons à prouver que :

$$\tan^2\delta\left(\frac{\cos^2\Phi}{\alpha_x^2} + \frac{\sin^2\Phi}{\alpha_y^2}\right) = \frac{\frac{(\hat{h}\cdot\hat{x})^2}{\alpha_x^2} + \frac{(\hat{h}\cdot\hat{y})^2}{\alpha_y^2}}{(\hat{h}\cdot\hat{n})^2} \tag{2.27}$$

On a :

$$\tan^2\delta = \frac{\sin^2\delta}{\cos^2\delta} \tag{2.28}$$

$$(\hat{h}\cdot\hat{x})^2 = \|\vec{h}\|^2\cdot\cos^2\Phi\cdot\sin^2\delta \tag{2.29}$$

$$\Leftrightarrow \cos^2\Phi = \frac{(\hat{h}\cdot\hat{x})^2}{\|\vec{h}\|^2\cdot\sin^2\delta} \tag{2.30}$$

$$(\hat{h}\cdot\hat{y})^2 = \|\vec{h}\|^2\cdot\sin^2\Phi\cdot\sin^2\delta \tag{2.31}$$

$$\Leftrightarrow \sin^2\Phi = \frac{(\hat{h}\cdot\hat{y})^2}{\|\vec{h}\|^2\cdot\sin^2\delta} \tag{2.32}$$

En remplaçant $\tan^2\delta$, $\cos^2\Phi$ et $\sin^2\Phi$ par leur expressions respectives 2.28, 2.30 et 2.32 dans le terme de gauche de l'équation 2.27, on obtient alors :

$$\begin{aligned}\tan^2\delta\left(\frac{\cos^2\Phi}{\alpha_x^2} + \frac{\sin^2\Phi}{\alpha_y^2}\right) &= \frac{\sin^2\delta}{\cos^2\delta}\cdot\left[\frac{\frac{(\hat{h}\cdot\hat{x})^2}{\alpha_x^2}}{\|\vec{h}\|^2\cdot\sin^2\delta} + \frac{\frac{(\hat{h}\cdot\hat{y})^2}{\alpha_y^2}}{\|\vec{h}\|^2\cdot\sin^2\delta}\right]\\ &= \frac{1}{\cos^2\delta}\cdot\left[\frac{\frac{(\hat{h}\cdot\hat{x})^2}{\alpha_x^2}}{\|\vec{h}\|^2} + \frac{\frac{(\hat{h}\cdot\hat{y})^2}{\alpha_y^2}}{\|\vec{h}\|^2}\right]\end{aligned}$$

Sachant que :

$$\|\vec{h}\|^2\cdot\cos^2\delta = (\hat{h}\cdot\hat{n})^2$$

On peut en déduire :

$$\tan^2\delta\left(\frac{\cos^2\Phi}{\alpha_x^2} + \frac{\sin^2\Phi}{\alpha_y^2}\right) = \left[\frac{\frac{(\hat{h}\cdot\hat{x})^2}{\alpha_x^2} + \frac{(\hat{h}\cdot\hat{y})^2}{\alpha_y^2}}{(\hat{h}\cdot\hat{n})^2}\right]$$ On retrouve bien l'équation 2.27

L'équation (5b) dans l'article de Ward diffère de 2.26 sur le terme situé à l'intérieur de l'exponentielle 2.27, Ward écrit en effet ce terme comme :

$$\tan^2\delta\left(\frac{\cos^2\Phi}{\alpha_x^2} + \frac{\sin^2\Phi}{\alpha_y^2}\right) = \left[2\cdot\frac{\frac{(\hat{h}\cdot\hat{x})^2}{\alpha_x^2} + \frac{1+(\hat{h}\cdot\hat{y})}{\alpha_y^2}}{1+(\hat{h}\cdot\hat{n})}\right] \tag{2.33}$$

Cette équation est fausse car il faudrait que :

$$\begin{aligned} \frac{1}{(\hat{h}\cdot\hat{n})^2} &= \frac{2}{1+(\hat{h}\cdot\hat{n})} \\ \Leftrightarrow 1+(\hat{h}\cdot\hat{n}) &= 2\cdot(\hat{h}\cdot\hat{n})^2 \\ \Leftrightarrow \hat{h}\cdot\hat{n} &= 1 \\ \text{ou } \hat{h}\cdot\hat{n} &= -0.5 \end{aligned}$$

$\hat{h}\cdot\hat{n} < 0$ est impossible car cela signifierait que le rayon réfléchi passerait à travers la surface et on doit avoir $\hat{d_i}\cdot\hat{n} > 0$ ainsi que $\hat{d_r}\cdot\hat{n} > 0$.
$\hat{h}\cdot\hat{n} = 1$ signifie que $\hat{h}$ est dans la direction de $\hat{n}$. Ceci implique que l'angle d'incidence est égal à l'angle de réflexion, et donc que la surface est spéculaire parfaite, ce qui est également faux.

De même pour le cas isotrope, on peut écrire :

$$f_{bd,iso,ward}(\theta_i,\varphi_i,\theta_r,\varphi_r) = \frac{\rho_d}{\pi} + \rho_s\cdot\frac{1}{\sqrt{(\hat{d_i}\cdot\hat{n})\,(\hat{d_r}\cdot\hat{n})}}\cdot\frac{1}{4\pi\alpha^2}\cdot e^{-\frac{(\hat{h}\cdot\hat{x})^2+(\hat{h}\cdot\hat{y})^2}{\alpha^2\cdot(\hat{h}\cdot\hat{n})^2}} \tag{2.34}$$

Démonstration
En remplaçant directement α_x et α_y par α dans l'équation 2.27, nous obtenons :

$$\begin{aligned} \frac{1}{\alpha^2}\cdot\tan^2\delta &= \frac{1}{\alpha^2}\cdot\frac{(\hat{h}\cdot\hat{x})^2+(\hat{h}\cdot\hat{y})^2}{(\hat{h}\cdot\hat{n})^2} \\ \Leftrightarrow \tan^2\delta &= \frac{(\hat{h}\cdot\hat{x})^2+(\hat{h}\cdot\hat{y})^2}{(\hat{h}\cdot\hat{n})^2} \end{aligned}$$

Enfin, l'échantillonnage stochastique de rayons spéculaires, dans un angle solide donné, en fonction de deux angles θ et Φ s'effectue grâce aux équations suivantes :

$$\theta = \tan^{-1}\left[\sqrt{\frac{-log(u_1)}{\frac{\cos^2\Phi}{\alpha_x^2}+\frac{\sin^2\Phi}{\alpha_y^2}}}\right] \tag{2.35}$$

$$\Phi = \tan^{-1}\left[\frac{\alpha_x}{\alpha_y}\cdot\tan(2\cdot\pi\,u_2)\right] \tag{2.36}$$

avec :
u_1 et u_2 des variables aléatoires uniformes comprises dans l'intervalle]0, 1].

2.5 Conclusion

Nous avons présenté l'ensemble des outils physiques nécessaires à la compréhension du rendu réaliste par radiosité, ainsi que les différentes BRDF que nous avons étudiées durant cette thèse. Finalement, nous avons retenu l'équation d'illumination de Ward [236] ainsi que son modèle de BRDF simple, mais puissant puisqu'il permet de traiter des surfaces anisotropes complexes. L'implémentation de ce modèle est également relativement simple en comparaison avec d'autres, comme celui de He et al. [97], plus rapide à exécuter (nous n'avons pas le coûteux terme de Fresnel à évaluer) et, dans sa forme actuelle, plus puissant également puisque le modèle de He et al. est limité aux surfaces isotropes (mais ce dernier reste le modèle le plus physique et pourrait être éventuellement étendu à l'anisotropie). Bien que nous ayons travaillé sur la BRDF de Ward pour retrouver les paramètres de réflectance des surfaces (voir chapitre 6), nous envisageons comme travaux futurs, l'implémentation d'autres modèles afin de pouvoir les comparer lors du recouvrement de réflectances, et de juger ainsi du plus efficace.

Chapitre 3

Méthode de la radiosité

À travers la poussière des siècles résonne le chant de l'homme,
comment ne rendrait-on pas grâce aux techniques de la maintenir.
Paul Valery

3.1 Introduction

Lorsque nous souhaitons calculer une image de synthèse photoréaliste[1], il est indispensable d'utiliser un modèle d'illumination sophistiqué intégrant notamment les inter-réflexions diffuses, et de façon générale les échanges d'énergie entre les objets de la scène. Une technique se révèle particulièrement adaptée à cette tâche : la **radiosité**. Ce chapitre a pour but de rappeler les principaux concepts de la radiosité. Nous présentons donc les différentes façons de résoudre l'équation de radiosité, ainsi que les méthodes de calcul des facteurs de forme gérant la façon dont se voient deux surfaces de l'espace 3D. Ce domaine particulier a fait l'objet de nombreux travaux, aussi nous ne prétendons pas à l'exhaustivité.

3.2 L'équation de rendu

Nous avons, au chapitre précédent, décrit plusieurs modèles de BRDF qui ont pour but de décrire la manière dont une surface *réfléchit* la lumière. De façon duale, un **modèle d'illumination** permet d'estimer la distribution de lumière *incidente* à une surface. Nous ne nous étendrons pas ici sur les modèles d'illumination *locale*, car ils ne prennent pas en compte les inter-réflexions entre surfaces, ce qui conduit généralement à des images non photoréalistes. Nous nous intéressons dans ce chapitre aux modèles d'illumination *globale*, qui ont un éventail beaucoup plus large et plus physique de la simulation de transfert d'énergie entre des surfaces.

Le premier modèle véritablement global fut introduit par Kajyia [108] sous le nom d'**équation de rendu**. Cette équation s'écrit sous la forme :

$$I(x,x') \;=\; g(x,x') \left[\epsilon(x,x') \;+\; \int_S \rho(x,x',x'')\, I(x',x'')\, dx''\right] \tag{3.1}$$

[1] Nous verrons en effet qu'il ne s'agit nullement ici d'un réalisme visuel subjectif, puisque nous comparons nos images de synthèse générées avec de vraies images du monde réel (voir chapitre 6).

avec :

$g(x,x')$ un terme de géométrie tel que $g(x,x') = \begin{cases} \frac{1}{d(x,x')^2} \text{ si } x \text{ et } x' \text{ se voient mutuellement;} \\ 0 \text{ sinon} \end{cases}$,

$\rho(x,x',x'')$ une fonction de réflexion de la lumière depuis x'' vers x en passant par x',

$I(x,x')$ la quantité de lumière passant depuis le point x' vers le point x,

$\epsilon(x,x')$ l'intensité de la lumière émise depuis le point x' vers le point x,

S l'ensemble des surfaces de la scène.

Une formulation plus employée de ce modèle consiste à écrire cette équation sous la forme d'une équation de luminance :

$$L(x,\vec{\omega}) = L_e(x,\vec{\omega}) + \int_\Omega \rho_{bd}(x,\vec{\omega'} \to \vec{\omega})\, L(x,\vec{\omega'}) \cos\theta'\, d\omega' \tag{3.2}$$

avec :

$L(x,\vec{\omega})$ la luminance énergétique quittant le point x dans la direction $\vec{\omega}$,

$L_e(x,\vec{\omega})$ la luminance énergétique émise par le point x dans la direction $\vec{\omega}$ (propriété de la surface) ,

$L(x,\vec{\omega'})$ la luminance incidente au point x de la surface, depuis la direction $\vec{\omega}'$,

$\rho_{bd}(x,\vec{\omega'} \to \vec{\omega})$ la fonction de réflectance bidirectionnelle de la surface au point x,

Ω l'ensemble des directions $\vec{\omega}'$ autour du point x de la surface,

θ' l'angle de $\vec{\omega'}$ avec la normale au point x.

3.3 L'équation de radiosité

La première contribution scientifique à utiliser le modèle de radiosité est celle de Goral et al. [84], qui se limitait à des environnements sans occultations entre les surfaces. L'équation de rendu précédente est plus générale, et peut être simplifiée pour obtenir l'équation de **radiosité**, tout en tenant compte des occultations.

En effet, en radiosité toutes les surfaces de l'environnement sont considérées comme des réflecteurs diffus lambertiens. Ainsi, la BRDF est indépendante des directions incidentes ou réfléchies et peut être sortie de l'intégrale :

$$\begin{aligned} L(x,\vec{\omega}) &= L_e(x,\vec{\omega}) + \int_\Omega \rho_{bd}(x,\vec{\omega'} \to \vec{\omega})\, L(x,\vec{\omega'}) \cos\theta'\, d\omega' \\ &= L_e(x,\vec{\omega}) + \rho_{bd}(x,\vec{\omega'} \to \vec{\omega}) \int_\Omega L(x,\vec{\omega'}) \cos\theta'\, d\omega' \\ &= L_e(x,\vec{\omega}) + \frac{\rho_d(x)}{\pi} \int_\Omega L(x,\vec{\omega'}) \cos\theta'\, d\omega' \end{aligned}$$

De plus, la luminance réfléchie depuis une surface lambertienne est la même dans toutes les directions et est en fait égale à la radiosité B divisée par π (voir paragraphe 2.4.5). On en déduit donc (en multipliant l'équation à gauche et à droite par π) :

$$B(x) = E(x) + \rho_d(x) \int_\Omega \frac{B(x') \cos\theta'}{\pi}\, d\omega' \tag{3.3}$$

avec :

$B(x)$ fonction de radiosité,

$E(x)$ fonction d'émission propre soit l'énergie par unité d'aire émise par la surface et $E(x) = \dfrac{L_e(x,\vec{\omega})}{\pi}$

x, x' deux points $3D$ dans le domaine S

L'équation la plus connue de la radiosité est en fait de la forme :

$$B_i \;=\; E_i \;+\; \rho_i \sum_{j=1}^{n} B_j \, F_{ij} \tag{3.4}$$

ou encore :

$$B_i \, A_i \;=\; E_i \, A_i \;+\; \rho_i \sum_{j=1}^{n} B_j \, F_{ij} \, A_i \tag{3.5}$$

avec :

B_i, B_j	la radiosité de la surface i et j respectivement,
A_i	l'aire de la surface i,
E_i	l'émission propre de la surface i,
ρ_i	la réflectivité diffuse de la facette i,
n	le nombre de surfaces dans la scène,
F_{ij}	le facteur de forme donné par $F_{ij} \;=\; \dfrac{1}{A_i} \displaystyle\int_{A_i} \int_{A_j} \dfrac{\cos\theta \cos\theta'}{\pi r^2} \cdot V_{ij} \, dA_j \, dA_i$ (ce facteur de forme représente la fraction d'énergie quittant l'élément i et arrivant directement sur l'élément j),
V_{ij}	un facteur de visibilité qui vaut 1 si i et j se voient mutuellement, 0 sinon,
θ	l'angle entre la normale à l'élément i et le vecteur reliant i à j,
θ'	l'angle entre la normale à l'élément j et le vecteur reliant i à j.

Dans l'équation 3.4, la radiosité B_i représente l'énergie totale par unité de surface, réfléchie par une surface i, et est exprimable comme la somme de deux termes. C'est-à-dire la somme de son énergie d'émission propre E_i (nulle si la surface i n'est pas une source de lumière), plus une fraction F_{ij} d'énergie B_j reçue depuis toutes les autres surfaces j de la scène et pondérée par la réflectivité locale ρ_i de la surface réceptrice.
En appliquant la relation suivante de réciprocité du facteur de forme[2] : $F_{ji} \, A_j \;=\; F_{ij} \, A_i$, nous pouvons réécrire l'équation 3.4 sous la forme de l'**équation finale de la radiosité** :

$$B_i \, A_i \;=\; E_i \, A_i \;+\; \rho_i \sum_{j=1}^{n} B_j \, F_{ji} \, A_j \tag{3.6}$$

C'est-à-dire sous forme matricielle (avec ici F_{ji} intégrant le terme $\dfrac{A_j}{A_i}$:

$$\begin{bmatrix} 1-\rho_1 F_{1,1} & \cdots & -\rho_1 F_{1,n} \\ -\rho_2 F_{2,1} & \cdots & -\rho_2 F_{2,n} \\ \vdots & \ddots & \vdots \\ -\rho_{n-1} F_{n-1,1} & \cdots & -\rho_{n-1} F_{n-1,n} \\ -\rho_n F_{n,1} & \cdots & 1-\rho_n F_{n,n} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_{n-1} \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_{n-1} \\ E_n \end{bmatrix} \tag{3.7}$$

ou sous sa forme simplifiée [108] :

$$M \cdot B \;=\; E \tag{3.8}$$

3.4 Résolution de la matrice de radiosité

On remarque que dans l'équation 3.8, lorsque n est grand, l'inversion directe de M est impossible. Il existe donc plusieurs autres techniques pour résoudre cette équation, nous proposons ici les plus connues.

[2]Le terme physique exact généralement employé est *facteur de configuration*.

3.4.1 Itération de Jacobi

Cette méthode itérative procède par la mise à jour de chaque $B_i(k)$ du vecteur solution, en calculant cette variable à l'aide de l'estimée courant de $B(k)$. Avec le vecteur estimé initial $B(0)$, nous pouvons chercher à résoudre pour un B_i, l'équation :

$$\sum_j M_{ij}\, B_j \;=\; E_i \tag{3.9}$$

$$\Leftrightarrow \quad M_{ii}\, B_i \;=\; Ei \;-\; \sum_{j \neq i} M_{ij}\, B_j \tag{3.10}$$

$$\Leftrightarrow \quad B_i \;=\; \frac{Ei}{M_{ii}} \;-\; \sum_{j \neq i} \frac{M_{ij}}{M_{ii}}\, B_j \tag{3.11}$$

Grâce à l'équation 3.9, à chaque nouvelle étape $k+1$ il est possible d'obtenir une nouvelle approximation du vecteur solution $B_i(k+1)$ en calculant pour chaque valeur de i :

$$B_i(k+1) \;=\; \frac{E_i}{M_{ii}} \;-\; \sum \frac{M_{ij}}{M_{ii}\, B_j(k)} \tag{3.12}$$

3.4.2 Méthode itérative de Gauss-Seidel

Cette technique de relaxation est une variante de l'algorithme de Jacobi, et est bien plus utilisée. Sa convergence est assurée par le fait que la matrice des facteurs de forme M est diagonale dominante [83].

L'avantage de cette technique sur celle de Jacobi est qu'un vecteur de radiosités estimées peut être employé et mis à jour directement pendant le calcul. À chaque étape durant une itération, la méthode de Gauss-Seidel utilise donc le vecteur solution le plus récent, plutôt que le vecteur calculé à l'itération précédente. Ainsi, à chacune des n étapes d'une itération, une nouvelle valeur de B est calculée en utilisant les valeurs $B(k+1)$ estimées aux étapes précédentes de l'itération courante. Ceci revient donc à employer l'équation suivante (la résolution totale du système d'équations est en $O(n^2)$:

$$B_i(k+1) \;=\; E_i \;-\; \sum_{j=1}^{i-1} M_{ij} \frac{B_j(k+1)}{M_{ii}} \;-\; \sum_{j=i+1}^{n} M_{ij} \frac{B_j(k)}{M_{ii}} \tag{3.13}$$

Algorithme 1 Pseudo-Algorithme de Gauss-Seidel

//Initialisation des radiosités
Pour Tous les éléments i **Faire**
  $B_i \;=\; E_i$
Fin
//Calcul itératif des radiosités
Tant Que pas de convergence **Faire**
  Pour chaque surface i **Faire**
    $B_i \;=\; E_i \;-\; \sum_{j=1, j \neq i}^{n} \frac{B_j\, M_{ij}}{M_{ii}}$
  Fin
Fin

Cohen et al. [39] ont montré qu'une variante de l'algorithme de Gauss-Seidel (voir algorithme 1) génère des résultats tout à fait satisfaisants pour la résolution de la matrice de radiosité. Par ailleurs, l'interprétation physique pour cette méthode consiste à observer qu'à chaque itération, l'algorithme calcule pour un élément de la scène la contribution énergétique de toutes les autres surfaces. Cette façon de procéder est appelée dans la littérature infographiste anglaise : *gathering*.

3.4.3 Méthode itérative de Southwell

Cette technique est une variante de la méthode de Gauss-Seidel et apporte un autre algorithme dont l'interprétation physique est différente. Dans cette méthode de Southwell [85], la rangée i avec le plus grand résidu, Max(r), sera toujours séléctionnée, alors que précédemment on relaxait chaque résidu à son tour. Ainsi, une étape simple de l'algorithme de Southwell[3] est définie par :

`Pour i, tel que` $r_i = \text{Max}(r)$ `Faire` $B_i = E_i - \sum_{j \neq i} B_j \frac{M_{ij}}{M_{ii}}$

Il semble que la complexité de cette étape soit en $O(n^2)$ pour pouvoir calculer tous les r_i avant de trouver le meilleur. Heureusement, à certains moments, $r(k)$ est connu pour un $B(k)$ donné, on peut donc trouver la prochaine approximation si on connait $\Delta B(k)$. Ainsi, les nouveaux B sont estimables comme :

$$B(k+1) = B(k) + \Delta B(k) \tag{3.14}$$

Le résidu r peut être mis à jour suivant :

$$\begin{aligned} r(k+1) &= E - M\,(B(k) + \Delta B(k)) \\ \text{or} \quad r(k) &= E - M\,B(k) \\ \text{donc} \quad r(k+1) &= r(k) - M\,\Delta B(k) \end{aligned}$$

Cependant, comme ici nous ne mettons à jour qu'un seul élément à la fois, tous les $\Delta B(k)$ sont nuls excepté celui de l'élément i, ΔB_i, qui vaut $\frac{r_i(k)}{M_{ii}}$. Nous en déduisons donc :

$$r_j(k+1) = r_j(k) - \frac{M_{ji}}{M_{ii}} \times r_i(k), \forall j$$

Si nous considèrons i et j comme des éléments constants ($F_{ii} = 0$ dans ce cas) et que M_{ji} est remplacée par son expression en fonction de F_{ji}, alors nous obtenons :

$$r_j(k+1) = r_j(k) + \rho_j\, F_{ji} \times r_i(k), \forall j$$

Enfin, le dernier outil indispensable est de calculer $r(0)$ dès le début de l'algorithme. En choisissant $B(0)$ à zéro (vecteur nul), on obtient : $r(0) = E - M\,B(0) = E$. On obtient ainsi l'algorithme 2.

L'interprétation physique de cette technique est, en quelque sorte, l'inverse de celle donnée pour la méthode de Gauss-Seidel. En effet, la méthode itérative de Southwell procède en fait en émettant l'énergie depuis une surface sélectionnée vers toutes les autres de la scène. Ainsi, nous réalisons une étape connue sous le nom de *shooting*, dans la littérature scientifique. Quant au vecteur résidu r, il représente la quantité d'énergie qu'il reste à réémettre pour chaque élément.

3.4.4 Radiosité progressive

Cohen, Chen, Wallace et Greenberg décrivent une forme différente de l'algorithme de Southwell, appelée **radiosité progressive** [38]. L'avantage principal de la radiosité progressive est qu'elle permet d'obtenir rapidement une image affichable des différentes étapes. En effet, comme elle procède en émettant d'abord l'énergie des surfaces pour lesquelles elles sont les plus fortes, il est évident que l'algorithme commence par les sources de lumière. Ainsi, l'objectif n'est pas seulement de fournir un algorithme qui converge rapidement, mais aussi qui approxime la solution exacte autant que possible dès les premières itérations. En radiosité progressive, le coeur de la radiosité traditionnelle devient une répétition itérative des étapes de calculs des facteurs de forme ainsi que des étapes de calcul de la solution de la matrice et d'affichage des résultats intermédiaires. Pour des éléments constants, le pseudo-code de l'algorithme de radiosité progressive est le suivant :

[3] Une preuve de la convergence de cette technique vers une solution est fournie dans [194], pages 58-59.

Algorithme 2 Pseudo-Algorithme de Southwell

//Phase d'initialisation
Pour Tous les éléments i **Faire**
$B_i = 0$
$r_i = E_i$
Fin
//Début du calcul itératif des radiosités
Tant Que pas de convergence **Faire**
Choisir i tel que $|r_i|$ soit le plus grand
$B_i = B_i + \frac{r_i}{M_{ii}}$
$r_{tmp} = r_i$
Pour Tous les éléments j **Faire**
$r_j = r_j - \frac{M_{ji}}{M_{ii}} \times r_{tmp}$
Fin
Fin

Algorithme 3 Pseudo-Algorithme de Radiosité Progressive

//Phase d'initialisation
Pour Tous les éléments i **Faire**
$B_i = E_i$
$\Delta B_i = E_i$
Fin
//Début du calcul itératif des radiosités
Tant Que pas de convergence **Faire**
Choisir i tel que $\Delta B_i \times A_i$ soit le plus grand possible
Pour Tous les éléments j **Faire**
$\Delta B = \Delta B_i \times \rho_j F_{ji}$
$\Delta B_j = \Delta B_j + \Delta B$
$B_j = B_j + \Delta B$
Fin
$\Delta B_j = 0$
Afficher l'image en utilisant B_i comme intensité de l'élément i
Fin

Dans cet algorithme, la boucle sur les éléments j met à jour la radiosité de chacun de ces éléments : l'énergie est lancée depuis l'élément i qui a, à ce moment là, l'énergie la plus importante à envoyer. Une telle étape de répartition prend $O(n)$ opérations et peut être perçue comme la multiplication d'une radiosité B_i par une colonne de la matrice de facteurs de forme. Cohen et al. [38] ont montré que dans plusieurs cas, seulement une petite fraction des n étapes de répartition est nécessaire pour atteindre la solution correcte.

Par ailleurs, il est évident qu'un tel algorithme converge nettement plus vite[4] vers la solution qu'une méthode de Gauss-Seidel, et tout particulièrement dans les premières étapes d'émissions d'énergie des sources de lumière. Cette vitesse de convergence a d'ailleurs été encore améliorée en utilisant l'ajout d'un terme *ambiant* [38] qui permet de tenir compte de la quantité d'énergie restant à émettre dans toute la scène : à la fin de chaque itération, la quantité d'énergie restant à réémettre est ajoutée aux radiosités calculées de tous les éléments de la scène, pour fournir de nouvelles radiosités qui serviront uniquement à l'affichage de la solution courante. Une autre optimisation possible est proposée par Feda et al. [62], et est connue dans la littérature scientifique anglophone sous le nom de *overshooting*[5]. Cette technique consiste à envoyer depuis une surface émettrice plus d'énergie qu'elle n'en possède réellement. Cette énergie, calculée depuis le terme ambiant, est ensuite réémise depuis les surfaces réceptrices, qui auront ainsi tendance à annuler progressivement son effet (de l'énergie négative sera donc émise plus tard depuis les surfaces). Shao et al. [254] proposent une modification de cette technique, appelée *positive overshooting*, en remplaçant le calcul du terme de sur-émission (initialement estimé par le facteur ambiant) d'un patch, par une somme de l'énergie restant à émettre depuis les autres surfaces vers celui-ci. Ce calcul est plus efficace car il est basé sur l'interaction que possède ce patch avec l'environnement. De plus, aucune étape d'émission d'énergie négative n'est nécessaire puisque cette énergie est déjà pré-émise par le reste de l'environnement, ce qui garantie son annulation plus tard dans les itérations de radiosité : tous les patchs possédant une énergie négative sont donc négligés. Shao et al. ont montré que cette méthode permet au minimum de doubler la vitesse de convergence de l'algorithme de radiosité progressive.

3.4.5 Méthodes hiérarchiques

La résolution de la matrice de radiosité reste une opération coûteuse en raison du grand nombre de facteurs de forme (et plus particulièrement de facteurs de visibilité) à calculer. En radiosité traditionnelle, nous calculons systématiquement les facteurs de forme entre un élément de la scène et tous les autres. Cohen et al. [37] proposent de créer une hiérarchie dans les surfaces de la scène : ainsi, un *patch* est subdivisé en quatre sous-éléments, et la structure de données qui sert à stocker cette représentation est appelée un *quadtree*[6]. Lors d'une étape d'émission de l'énergie depuis une surface, cette radiosité n'est émise que depuis les *patchs* (qui sont bien moins nombreux que les éléments) vers tous les autres éléments de la scène (les *patchs* parents de ces éléments sont également mis à jour en vue d'une prochaine étape d'émission). Cette idée est également motivée par le fait que nous avons besoin d'un niveau de détail plus important pour emmagasiner l'énergie (à cause des zones d'ombres et de pénombres qui créent des discontinuités dans la fonction radiosité) que pour la réémettre. Cohen et al. [37] proposent d'ailleurs d'adapter automatiquement le maillage des éléments lorsque leur fonction de radiosité possède un gradient élevé (calculé depuis les sommets de l'élément). Cette technique appelée *subdivisions adaptatives*[7] permet de créer un maillage adapté au comportement de la fonction de radiosité sur les surfaces, et a tendance à fortement subdiviser les éléments dans les zones d'ombres et de pénombres. Cependant cette hiérarchie n'a

[4] Gortler et al. [85] proposent un tour d'horizon de ces méthodes de résolution matricielle pour la radiosité, ainsi que leurs preuves de convergence.

[5] Bien que nous puissions traduire ce terme par *surémission d'énergie*, nous pensons qu'il est raisonnable de garder le terme anglophone, car il est très connu dans le domaine de la radiosité.

[6] On pourrait proposer comme traduction *arbre quaternaire* ou encore *quatre-arbre*, mais nous pensons préférable de conserver le terme original extrêmement employé dans la littérature scientifique.

[7] Nous ne nous livrerons pas ici à un tour d'horizon de toutes les techniques permettant la subdivision automatique des éléments, ou l'obtention d'un maillage optimal. Néanmoins, il est possible d'obtenir une excellente description de toutes ces techniques dans [40], chapitres 6,8 et de les complèter avec [213, 98, 75, 128, 76, 127, 232].

que deux niveaux : le niveau supérieur *patch* et le niveau terminal *élément*. On n'échange donc de l'énergie qu'entre ces deux niveaux.

```
Type Structure _NoeudQuaternaire
{
flottant B_emg;  // Radiosité emmagasinée
flottant B_env;  // Radiosité envoyée
flottant E;      // Emission propre
flottant aire;   // Aire de la surface
flottant ρ;      // Réflectivité diffuse de la surface
Structure _NoeudQuaternaire *fils[4]; // Pointeur sur une liste de 4 fils
Structure _Lien *L; // Pointeur vers le premier lien de récolte de radiosités
} NoeudQuaternaire;

Type Structure _Lien
{
NoeudQuaternaire *q; // Noeud récepteur
NoeudQuaternaire *p; // Noeud émetteur
flottant F_qp; // Facteur de forme de q à p
Structure _Lien *suivant; // Pointeur vers le lien de récolte suivant
} Lien;
```

FIG. 3.1 – Structures de données *NœudQuaternaire* et *Lien* pour la radiosité hiérarchique.

Algorithme 4 Pseudo-algorithme *Oracle1* pour la radiosité hiérarchique

Procédure Booléen Oracle1(NoeudQuaternaire *p, NoeudQuaternaire *q, flottant F_ϵ)
//Cette fonction retourne un booléen qui détermine si le nœud p doit être lié au nœud q,
//en estimant l'erreur générée par ce lien par rapport à l'erreur que créeraient plusieurs liens
//sur les fils de p et q. Si p et q ont déjà atteint le niveau maximal de subdivision, alors la
//fonction renvoie FAUX.
Si $((p \rightarrow aire < A_\epsilon)$ && $(q \rightarrow aire < A_\epsilon))$ **Alors**
retourne FAUX ;
Fin si
Si (Facteur_de_Forme(p, q) $< F_\epsilon$) **Alors**
retourne FAUX ;
Sinon
retourne VRAI ;
Fin si
Fin Procédure

Hanrahan et al. [93] proposent une généralisation de cette méthode hiérarchique en introduisant une hiérarchie multi-niveaux permettant l'échange d'énergie entre des groupes de surfaces, situés à des niveaux différents des *quadtrees* et non plus seulement à leurs extrémités. Ainsi, lorsque deux groupes de surfaces s'échangent de l'énergie à des niveaux différents et supérieurs aux nœuds terminaux, cette énergie est alors propagée à ces nœuds[8] par la suite. Cette façon de procéder permet de réduire le nombre de facteurs de forme à calculer, qui passe de $O(n \times m)$(approche de Cohen et al. [37], avec m et n le nombre de *patchs* et d'*éléments* respectivement, et $m >> n$), à seulement $O(n)$ ici.

La technique de Hanrahan et al. procède en établissant des *liens* entre les nœuds des *quadtrees* susceptibles d'échanger de l'énergie. Ainsi, pour chacune des paires de nœuds échangeant effectivement de l'énergie, un facteur de forme est calculé. Nous avons donc deux grandes structures de données principales : les nœuds quaternaires (car chaque surface (=nœud) peut être subdivisé en

[8] On appelle *nœud*, l'entité située à un niveau quelconque d'un *quadtree* et qui représente indifféremment une surface ou un groupe de surfaces.

quatre fils) et les liens (voir figure 3.1). On constate que pour ces nœuds quaternaires, on stocke une liste de liens qui correspondent à la liste des interactions qu'a ce nœud avec d'autres surfaces ou groupes de surfaces dans la scène. La question qui se pose donc désormais est : "comment sont créés ces liens ?"

Le critère qui détermine cette création de liens est réalisé par une fonction connue sous le nom de **oracle**[9], et part du principe que deux nœuds s'échangent de l'énergie si leur facteur de forme est inférieur à un certain seuil donné (voir algorithme 4). Dans l'article de Hanrahan et al., cette fonction *oracle* (qui est présentée sous une autre forme) estime un facteur de forme majoré, qui est en réalité un facteur de forme de type "aire différentielle vers aire" et sans occultations. Il est cependant possible de calculer une meilleure approximation de ce facteur de forme, par lancer de rayons par exemple.
À l'aide de cet *oracle*, la fonction *Raffiner* va rechercher et mémoriser pour les couples de nœuds (p, q) leur éventuel niveau d'interaction entre eux, en les subdivisant récursivement (voir algorithme 5) en fonction de leurs facteurs de forme réciproques. Cette fonction (appelée *Subdivise()* dans l'algorithme 5) retourne la surface qui a été subdivisée, c'est-à-dire soit le nœud p, soit le nœud q (avec leurs 4 fils subdivisés), soit un pointeur nul. Elle renvoie p si le facteur de forme de p à q est plus petit que le facteur de forme de q à p, ou renvoie q si c'est l'inverse. Si le niveau maximal de subdivision a été atteint ou si les facteurs de forme sont égaux, alors la fonction force l'interaction entre ces deux nœuds en créant un lien.

Algorithme 5 Pseudo-Algorithme Raffiner pour la radiosité hiérarchique

```
Procédure Raffiner(NoeudQuaternaire *p, NoeudQuaternaire *q, flottant F_ε)
  NoeudQuaternaire *p_ou_q, r ;
  //Oracle peut être soit Oracle1(), soit Oracle2() (voir pages suivantes)
  Si Oracle(p, q, F_ε) Alors
    Créer_Lien(p, q) ;
  Sinon
    p_ou_q=Subdivise(p, q) ; //retourne p, q, ou NULL (voir texte)
    Si (p_ou_q==q) Alors
      Pour Tous les noeuds fils r de q Faire
        Raffiner(p, r, F_ε)
      Fin
    Sinon
      Si p_ou_q==p Alors
        Pour Tous les noeuds fils r de p Faire
          Raffiner(r, q, F_ε)
        Fin
      Sinon
        Créer_Lien(p, q) ;
      Fin si
    Fin si
  Fin si
Fin Procédure
```

Lorsque chaque paire de nœuds a été traitée par l'algorithme 5, nous avons un ensemble de liens reliant deux nœuds de différents *quadtrees* de la scène. Il n'y a donc pas de matrice de radiosités qui soit construite, mais cet ensemble de liens constitue un système linéaire d'équations, qui peut être résolu pour en déduire les radiosités des éléments.
On procéde alors à la résolution du système hiérarchique (voir algorithme 6), pour estimer ces radiosités. Cette résolution passe par deux étapes : la première consiste à emmagasiner l'énergie

[9] Le système de notations, ainsi que les algorithmes suivant sont directement extraits de [40] et diffèrent légèrement de la version originale de l'article d'Hanrahan et al. [93].

transportée par chaque lien au nœud récepteur (voir algorithme 7). Il s'agit en fait ici d'une étape de *gathering*, similaire à celle de Jacobi décrite précédemment. Pour chaque lien partant d'un premier nœud vers un autre, la radiosité émise B_{env} par ce premier nœud est convertie en radiosité reçue (ou emmagasinée) B_{emg} vers le nœud avec lequel il interagit. Il se peut alors que les nœuds recepteurs ne soient pas des nœuds terminaux de la hiérarchie, et qu'ils possèdent des fils issus des subdivisions. La seconde étape propage donc la radiosité emmagasinée à chaque nœud récepteur vers leurs fils , et remonte ensuite la radiosité moyenne de quatre fils vers leur père (voir algorithme 8).
Cette étape de résolution est réalisée de manière itérative, jusqu'à ce que l'évolution des radiosités des surfaces ne changent pas plus d'une itération à l'autre, qu'un seuil initial fixé par l'utilisateur.

Algorithme 6 Pseudo-Algorithme de calcul de la solution du système hiérarchique

```
Procédure Résoudre_Système()
  //L'algorithme s'arrête lorsque la différence de radiosités d'une itération
  //à l'autre ne varie pas plus qu'un seuil fixé par l'utilisateur.
  Tant Que pas de convergence Faire
    Pour Toutes les surfaces p Faire
      Emmagasiner_Radiosité(p) ;
    Fin
    Pour Toutes les surfaces p Faire
      Propager_Radiosité(p, 0.0) ;
    Fin
  Fin
Fin Procédure
```

Algorithme 7 Pseudo-Algorithme de récolte des radiosités

```
Procédure Emmagasiner_Radiosité(NoeudQuaternaire *p)
  NoeudQuaternaire *q ;
  Lien *L ;
  p → B_emg = 0.0 ;
  Pour Tous les liens récepteurs L de p Faire
    p → B_emg += (p → ρ * L → F_pq * L → q → B_env) ;
  Fin
  Pour Tous les fils r de p Faire
    Emmagasiner_Radiosité(r) ;
  Fin
Fin Procédure
```

Les algorithmes que nous avons décrits jusqu'à présent permettent de générer une hiérarchie de surfaces, reliées entre elles par des liens situés à différents niveaux, et de résoudre ainsi le système obtenu. Il est néanmoins possible d'améliorer la qualité des subdivisions obtenues, et d'augmenter ainsi les perfomances de l'algorithme. En effet, jusqu'ici la fonction *Oracle1* ne prenait ses décisions que sur un critère purement géométrique (un facteur de forme), et donc complètement indépendant du flux d'énergie. C'est pourquoi l'algorithme de radiosité hiérarchique (voir algorithme 9) que nous donnons ici, utilise une autre fonction *Oracle* (appelée *Oracle2*, voir algorithme 11) qui va construire cette fois-ci les liens entre les nœuds, suivant la quantité d'énergie transférée d'un nœud à l'autre[10]. En première passe de cet algorithme, la fonction *Raffiner()* utilise donc maintenant *Oracle2()* et crée des liens entre des nœuds placés en haut de la hiérarchie (ces nœuds ne répondent à aucun des critères de subdivision puisque leur radiosité est nulle au début), sauf si la

[10] On constate par ailleurs que le seuil d'erreur fixé par l'utilisateur dépend désormais de la radiosité, de l'aire et du facteur de forme.

Algorithme 8 Pseudo-Algorithme de propagation des radiosités

Procédure Propager_Radiosité(NoeudQuaternaire *p, flottant B_{bas})
  flottant B_{haut}, B_{tmp} ;
  Si ($p \rightarrow fils$ ==NULL) **Alors**
    $B_{haut} = p \rightarrow E + p \rightarrow B_{emg} + B_{bas}$;
  Sinon
    $B_{haut} = 0.0$;
    Pour Tous les fils r de p **Faire**
      B_{tmp} = Propager_Radiosité(r,$p \rightarrow B_{emg} + B_{bas}$) ;
      $B_{haut} += B_{tmp} \times \frac{r \rightarrow aire}{p \rightarrow aire}$;
    Fin
  Fin si
Fin Procédure

surface est une source de lumière (la radiosité étant différente de zéro, il devient possible de créer des subdivisions). En seconde passe, la procédure *Raffiner_Lien()* (voir algorithme 10) va raffiner ces liens, au fur et à mesure qu'il existe de plus en plus de surfaces dans la scène, qui reçoivent de l'énergie, et qui doivent en réémettre : les liens de première passe sont donc modifiés pour créer des liens à des niveaux plus bas de la hiérarchie.

Algorithme 9 Pseudo-Algorithme de radiosité hiérarchique

Procédure Radiosité_Hiérarchique(flottant BF_ϵ)
  NoeudQuaternaire $*p, *q$;
  Lien $*L$;
  Booléen terminé=FAUX ;
  Pour Toutes les surfaces p **Faire**
    $p \rightarrow B_{env} = p \rightarrow E$;
  Fin
  Pour Toutes les paires de surfaces p, q **Faire**
    Raffiner(p, q, BF_ϵ) ;
  Fin
  Tant Que !terminé **Faire**
    terminé=VRAI ;
    Résoudre_Système() ;
    Pour Tous les liens L **Faire**
      Si (Raffiner_Lien(L, BF_ϵ)==FAUX) **Alors**
        terminé=FAUX ;
      Fin si
    Fin
  Fin
Fin Procédure

Les techniques d'illumination globale faisant appel à la radiosité hiérarchique sont très nombreuses aujourd'hui, comme en témoigne le nombre d'articles relatifs à ce sujet [193, 191, 18, 57, 93, 9, 60, 75, 128, 100, 200] et proposant des extensions de cette méthode. Par ailleurs, certains grands logiciels de calcul d'illumination dans des scènes architecturales complexes, utilisent ce concept de radiosité hiérarchique[11], pour calculer des images de synthèse photoréalistes.

[11] *Lightscape* (www.lightscape.com) est probablement le plus connu des logiciels de rendu photoréaliste, utilisant la radiosité hiérarchique.

Algorithme 10 Pseudo-Algorithme Raffiner_Lien

```
Procédure Booléen Raffiner_Lien(Lien *L, flottant BF_ε)
  NoeudQuaternaire *p = L → p, *q = L → q, *p_ou_q ;
  Booléen pas_de_subdivision=TRUE ;
  Lien *L ;
  Si Oracle2(L,BF_ε) Alors
    pas_de_subdivision=FAUX ;
    p_ou_q=Subdiv(p, q) ;
    Détruire_Lien(L) ;
    Si (p_ou_q == q) Alors
      Pour Tous les fils r de q Faire
        Créer_Lien(p, r) ;
      Fin
    Sinon
      Pour Tous les fils r de p Faire
        Créer_Lien(r, q) ;
      Fin
    Fin si
  Fin si
  retourne pas_de_subdivision ;
Fin Procédure
```

Algorithme 11 Pseudo-Algorithme de Oracle2() prenant en compte le flux d'énergie

```
Procédure Booléen Oracle2(NoeudQuaternaire *p, NoeudQuaternaire *q, flottant BF_ε)
  NoeudQuaternaire *p = L → p ; //émetteur
  NoeudQuaternaire *q = L → q ; //receveur
  Si ((p → aire < A_ε) && (q → aire < A_ε)) Alors
    retourne FAUX ;
  Fin si
  Si (p → B_env == 0.0) Alors
    retourne FAUX ;
  Fin si
  Si (p → B_env × p → aire × L → F_pq < BF_ε) Alors
    retourne FAUX ;
  Sinon
    retourne VRAI ;
  Fin si
Fin Procédure
```

Cependant, bien que la radiosité hiérarchique semble constituer, à l'heure actuelle, la quintessence des méthodes de résolution de l'équation de radiosité, et qu'elle réduise de manière drastique le nombre de facteurs de forme, elle doit comme toutes les autres méthodes passer par cette lourde étape de calcul. Nous proposons donc dans le paragraphe suivant, un tour d'horizon non exhaustif des méthodes les plus connues pour estimer ces facteurs de forme.

3.5 Calcul des facteurs de forme

3.5.1 Introduction

Le facteur de forme est un des éléments essentiels du rendu réaliste par radiosité, car il monopolise la quasi-totalité de temps de résolution de l'équation de radiosité. En fait, pour être plus précis, c'est le terme de visibilité qui permet de prendre en compte les occultations entre surfaces qui est le plus coûteux. C'est pour cette raison que bon nombre d'auteurs tentent, soit d'en réduire la quantité à calculer (comme pour la radiosité hiérarchique par exemple), soit de réduire le temps de calcul d'un seul d'entre eux (en préconisant l'emploi de *hardware* dédié ou de techniques d'optimisations connues). De nombreuses techniques existent donc pour calculer les facteurs de forme comme l'hémicube [39], l'hémisphère [202], le lancer de rayons [234], la méthode de Monte-Carlo [18, 225], l'intégrale de contour de Stokes [84], le calcul analytique [190], le plan unique de projection [197, 164], etc. . Nous ne présentons ici que les plus utilisées et les plus connues.

3.5.2 Hémicube

L'hémicube introduit par Cohen et al. [39] est probablement la plus connue des techniques de calcul de facteur de forme, car elle permet l'emploi de l'algorithme de rendu le plus commun qui soit : le *Z-buffer*. L'hémicube est constitué de cinq faces, qui sont discrétisées en *cases élémentaires* et à une résolution fixée par l'utilisateur. Chacune de ces cases contient un facteur de forme élémentaire[12] précalculé dans une table statique. La technique de l'hémicube consiste à projeter successivement toutes les facettes de la scène sur chacune de ces cinq faces, et de procéder à un tri de ces facettes en fonction de leur profondeur pour déterminer le coûteux terme de visibilité (ce qui n'est donc rien d'autre qu'un Z-Buffer). Lorsque "l'image" des parties cachées a été estimée (cette image contient des pointeurs sur les faces visibles), on procède au parcours de celle-ci en comptabilisant pour chaque surface visible, le nombre de *cases élémentaires* qu'elle recouvre. Calculer un facteur de forme par hémicube entre une surface (une aire différentielle en fait) et les autres surfaces de la scène, revient donc d'une part à effectuer cinq Z-Buffer (un pour chaque face) pour estimer cinq images d'index de facettes[13], et d'autre part, à cumuler les facteurs de forme élémentaires recouverts par chacune des surfaces sur ces images (voir algorithme 3.5.2). Un pseudo-code détaillé est également disponible dans [92], tandis qu'une implémentation complète en $C++$ est fournie par [7].

Les avantages de l'hémicube sont sa relative simplicité, le fait de pouvoir utiliser un *Z-Buffer* pour calculer les facteurs de forme et sa facilité à être implémenté. Néanmoins, il présente plusieurs défauts principalement inhérents au fait que l'on utilise un espace discrétisé en cases (*delta facteur de forme*), pour calculer le facteur de forme de chacune des surfaces se projetant dessus. Baum et al. [14] ont démontré que l'emploi de l'hémicube fournit une bonne approximation du facteur de forme si trois hypothèses précises sont vérifiées. La première hypothèse est que la distance entre deux surfaces i et j doit être grande comparativement à la taille de i sur lequel est placé l'hémicube (appelée *hypothèse de proximité*). La seconde hypothèse précise que la facette i est visible de la même manière de tout point de la facette j (connue sous le nom d'*hypothèse de visibilité*). Et enfin, la troisième hypothèse repose sur le fait que la projection de chaque surface sur l'hémicube peut être représentée par un nombre fini de cases élémentaires (*hypothèse d'aliassage*). Les inconvénients de l'hémicube sont donc clairs : si ces hypothèses ne sont pas vérifiées, le

[12] Nous ne redonnons pas ici la manière dont sont obtenus les *delta facteurs de forme* (ou facteurs de forme élémentaires), car le lecteur pourra les trouver aisément dans la littérature infographiste [40, 39, 7].

[13] Cette notion de *buffer* d'index est plus connue sous le nom de *item buffer* dans la littérature scientifique [240].

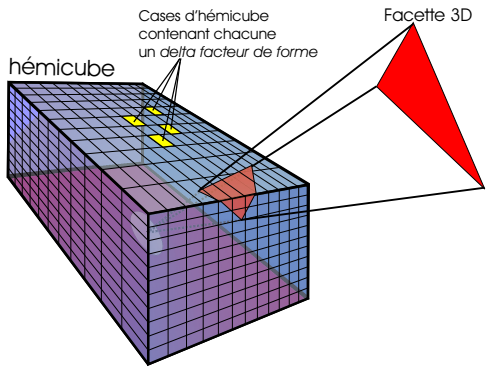

FIG. 3.2 – Facette se projetant sur un hémicube. Les cases recouvertes par la projection de la facette sur l'hémicube vont permettre d'estimer le facteur de forme entre l'aire différentielle (petite ellipse au centre de l'hémicube) et l'aire de la facette 3D. Si plusieurs facettes se projettent sur des cases de l'hémicube, il est procédé à un test de profondeur pour chacune des cases, afin de déterminer quelle est la facette visible (cette opération est généralement réalisée par un *Z-Buffer*).

Algorithme 12 Pseudo-Algorithme de calcul de facteur de forme par hémicube

Pour Tous tous les éléments i et j **Faire**
  $FF[i][j] = 0.0$; //facteur de forme de i à j
Fin
Pour Toutes les cellules l des 5 faces de l'hémicube **Faire**
  cellule$[l].eltidx = -1$;
  cellule$[l].Z = +\infty$;
Fin
Génerer un point A aléatoirement sur l'élément i ;
Centrer l'hémicube sur A ;
Pour les 5 faces k de l'hémicube **Faire**
  Pour Tous tous les éléments j $(j \neq i)$ **Faire**
    Clipper et Projeter élément j sur la face k ;
    Pour chaque cellule l recouverte par la projection de j **Faire**
      Si $face \rightarrow cellule[l].Z < face \rightarrow cellule[l].Z$ **Alors**
        $face \rightarrow cellule[l].eltidx = j$;
        $face \rightarrow cellule[l].Z = face \rightarrow cellule[l].Z$;
      Fin si
    Fin
  Fin
  Pour Toutes les cellules l de la face k **Faire**
    Si $cellule[l].eltidx \neq< -1$ **Alors**
      //Calcul du facteur de forme par cumul des facteurs de forme élémentaires
      $FF[i][cellule[l].eltidx] += cellule[l].\Delta FF$;
    Fin si
  Fin
Fin

calcul des facteurs de forme devient faux. Si on peut tenter de résoudre certains de ces problèmes (comme l'aliassage, en augmentant la résolution de l'hémicube, ou en positionnant et en orientant aléatoirement l'hémicube sur les facettes [140]), on ne peut les faire disparaître complètement. Il existe bien entendu d'autres formes géométriques 3D, plutôt qu'un hémicube, pour calculer les facteurs de forme, comme le tétraèdre cubique [19, 20][14], l'hémisphère discrétisé [202][15], ou encore le plan unique de projection [197, 195, 164].

3.5.3 Méthode du plan unique

L'utilisation d'un hémicube présente quelques inconvénients, en plus des hypothèses d'utilisation, puisqu'il faut procéder à cinq transformations perspectives pour pouvoir générer les cinq images correspondantes. La méthode introduite par Sillion [197, 195] consiste à projeter la scène une seule fois, sur un plan parallèle à la surface dont on cherche à calculer les facteurs de forme.

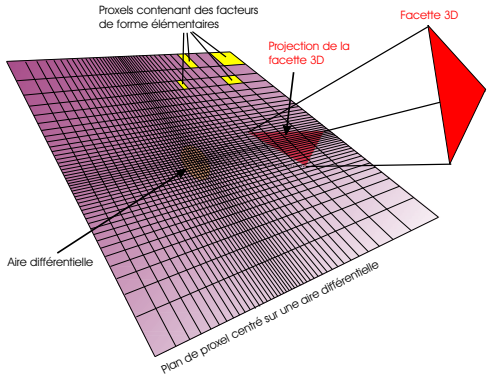

FIG. 3.3 – Plan de proxels centré sur une aire différentielle (symbolisée par une ellipse jaune), et situé à une hauteur ϵ de la surface. On constate que le plan de proxels est de plus en plus échantillonné au fur et à mesure que l'on se rapproche de son centre. La facette rouge se projette sur le plan de proxels, et la somme des facteurs de forme élémentaires des proxels recouverts par la projection détermine le facteur de forme entre l'aire différentielle et la facette 3D.

La projection du demi-espace supérieur sur un plan horizontal recouvre tout le plan, et on doit se limiter à une région finie de ce plan, que l'on choisit carrée par commodité, en négligeant délibérément la partie qui se projette en dehors du carré d'analyse. Ce plan de projection est échantillonné en des éléments rectangulaires tout comme l'hémicube. Pour chacune de ces cases élémentaires que l'on appelle en réalité des **proxels** (=*projection element* par analogie avec pixel), le facteur de forme[16] est précalculé et constant. Pour cette raison, le plan de proxels n'est pas uniforme et il est subdivisé de telle façon que, plus on s'éloigne du centre du plan, plus les proxels deviennent grands (voir figure 3.3).

Dans le même ordre d'idée, Recker et al. [164] proposent également l'emploi d'un plan unique de projection, mais n'utilisant que deux niveaux de discrétisation. Ainsi deux zones distinctes sont visibles : l'une est très échantillonnée au centre du plan, tandis que la seconde située en périphérie de la première, possède une discrétisation plus grossière (voir figure 3.4).

Si l'avantage majeur de la méthode de Sillion [197] ou de Recker [164] sur la technique de l'hémicube est de n'utiliser qu'une seule projection, elle engendre des erreurs plus importantes sur les facettes "rasantes", pour lesquelles leur projection risque de ne pas se trouver sur le plan. Recker et al. proposent alors de fermer le volume en ajoutant quatre faces latérales, ce qui nous ramène à une technique proche de celle de l'hémicube.

[14] Bien que nous ne décrivions pas cet algorithme ici, une implémentation complète et détaillée de cette méthode est disponible dans [7].

[15] Nous ne décrirons pas non plus ici cette technique car elle procède de façon similaire à l'hémicube, mais en employant un système de coordonnées sphériques également associé à un *Z-Buffer*.

[16] Le calcul des valeurs des facteurs de forme élémentaires peut être trouvé en détails dans [195] pages 52-59, ou plus sommairement dans [197].

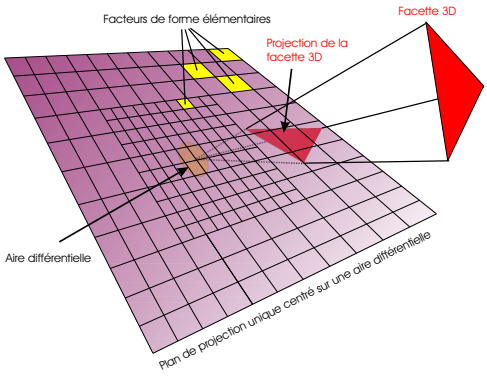

FIG. 3.4 – Plan de projection unique de Recker [164], centré sur une aire différentielle (symbolisée par une ellipse jaune), et situé à une hauteur ϵ de la surface. On remarque que la discrétisation du plan est importante au centre et plus faible en périphérie (deux niveaux seulement). La facette rouge se projette sur le plan unique, et la somme des facteurs de forme élémentaires des cases recouvertes par la projection détermine le facteur de forme entre l'aire différentielle et la facette 3D.

3.5.4 Lancer de rayons classique et Monte-Carlo

Les algorithmes précédents de calcul de facteurs de forme emploient des techniques dites *projectives* pour estimer notamment le facteur de visibilité. Naturellement, ce facteur de visibilité peut également être évalué par lancer de rayons, et c'est notamment ce que proposent Malley [135], Wallace et al. [234], ainsi que Tampieri [212].

La méthode proposée par Wallace et al. [234] permet de calculer le facteur de forme entre un élément de surface différentiel (le récepteur) dA_i et une autre surface A_j de la scène. Cette surface A_j est alors subdivisée en n éléments plus petits A_j^k d'aire ΔA_j^k, et des rayons sont lancés entre le centre de dA_i et le centre de chaque élément subdivisé A_j^k. On réalise alors la somme des facteurs de forme entre dA_i et A_j^k, en tenant compte de la visibilité ainsi calculée. Le facteur de forme s'écrit donc :

$$F_{dA_i \to A_j} = \sum_{k=1}^{n} \frac{\cos\theta_i^k \cos\theta_j^k}{\pi (r^k)^2} \cdot V(dA_i, \Delta A_j^k) \Delta A_j^k \tag{3.15}$$

Cette formulation présuppose que la distance entre les deux facettes A_i et A_j est grande en comparaison de l'aire de chaque élément A_j^k, et pose des problèmes lorsque les deux facettes sont très proches l'une de l'autre (r^k tend alors vers 0). Une amélioration de cette formule consiste donc à approximer chaque ΔA_j^k, comme un disque de même aire soit $\frac{A_j}{n}$. Par ailleurs, comme le facteur de forme entre un élément différentiel et un disque d'orientation quelconque s'écrit :

$$F_{dA_i \to \Delta A_j^k} = \frac{\cos\theta_i^k \cos\theta_j^k \Delta A_j^k}{\pi (r^k)^2 + \Delta A_j^k} \tag{3.16}$$

On obtient alors la nouvelle équation :

$$F_{dA_i \to A_j} = \frac{A_j}{n} \sum_{k=1}^{n} \frac{\cos\theta_i^k \cos\theta_j^k}{\pi (r^k)^2 + \frac{A_j}{n}} \cdot V(dA_i, \Delta A_j^k) \tag{3.17}$$

On remarque que cette équation ne pose plus de problème lorsque r^k tend vers 0, cependant des problèmes d'aliassage peuvent faire leur apparition si r_k est plus petit que $\frac{A_j}{n}$. Tampieri [212] propose une méthode pour résoudre ce problème, en subdivisant adaptativement l'aire A_j de telle sorte que les éléments subdivisés soient toujours plus petits que r_k.

Une autre technique pour calculer les facteurs de forme en utilisant le lancer de rayons, est celle de Malley [135], qui consiste à générer des points aléatoirement dans un disque, puis à projeter

ces points sur un hémisphère centré sur ce disque. Les points intersectés sont alors utilisés en combinaison avec le centre de l'hémisphère pour générer un rayon dans l'espace. Ce rayon est ensuite intersecté avec l'ensemble des surfaces de la scène, pour déterminer quelle surface est atteinte[17]. Le facteur de forme entre une surface et l'élément émetteur est déterminé comme le rapport du nombre de rayons intersectant cet élément, divisé par le nombre total de rayons émis.

Enfin, il est possible de calculer les facteurs de forme par le lancer de rayons de *Monte-Carlo*. Cette méthode consiste à générer des points aléatoirement (en utilisant une loi de Poisson par exemple) sur les deux éléments dont on cherche le facteur de forme, puis à lancer des rayons entre ces points sur les deux surfaces, deux à deux pour déterminer le facteur de visibilité. Une excellente description détaillée de cette méthode et du lancer de rayons par Monte-Carlo est disponible dans [18] et dans [225].

3.5.5 Méthodes analytiques

Dans certains cas particuliers, les facteurs de forme peuvent être calculés de manière analytique, et donc exacte. Ces cas correspondent par exemple, à des surfaces perpendiculaires ou parallèles. L'ouvrage [190] pages 981-1037, résume l'ensemble de tous les facteurs de forme disponibles de manière analytique, en fournissant soit une référence bibliographique, soit directement l'équation.

3.6 Conclusion

Nous avons présenté dans ce chapitre un rapide tour d'horizon des différentes méthodes existant pour réaliser une simulation d'illumination globale par radiosité. Nous avons également introduit les techniques les plus connues pour résoudre le problème des facteurs de forme. L'ensemble de cette partie permet de mieux saisir les choix qui ont été réalisés par la suite, pour la conception de notre logiciel de rendu réaliste *Phoenix*, dont nous allons maintenant donner une complète description dans le chapitre suivant.

[17] Bien sûr l'ensemble des techniques classiques d'optimisations en lancer de rayons est utilisable, pour éviter des calculs inutiles.

Chapitre 4

Notre logiciel de Rendu : Phoenix

Je ne puis préciser ma perception d'une
chose sans la dessiner virtuellement.
René Berger

4.1 Présentation générale de Phoenix

Le sujet principal de cette thèse tourne autour du rendu réaliste depuis des images réelles. Comme nous le verrons dans le chapitre 6, notre technique de regénération d'images est itérative et hiérarchique, ce qui implique notamment le calcul répété d'une image de synthèse, de qualité photoréaliste.
La difficulté de réaliser ce logiciel de rendu réside dans le fait que nous tentons d' obtenir la meilleure image (qualité visuelle) en un temps le plus court possible, puisque nous recalculons cette image jusqu'à ce qu'elle atteigne son réalisme maximal (par comparaison à une image réelle de référence). Nous avons donc développé durant cette thèse, un logiciel de calcul d'images de synthèse photoréalistes, que nous avons appelé *Phoenix*. *Phoenix* ne produit pas spécialement des images plus réalistes -ni moins- que ses concurrents indirects[1], mais il les réalise en un temps qui peut être inférieur (on trouvera un comparatif avec le logiciel *Lightscape* au paragraphe 4.4). *Phoenix* n'apporte pas, à proprement parler, d'innovations scientifiques mais une combinaison unique de plusieurs outils pour la synthèse d'image, dont certains sont spécifiques à l'architecture de la machine employée[2].
Nous avons souvent préféré allier la simplicité de certaines méthodes (radiosité progressive, hémicube, etc.), à des techniques de rendu réaliste rapides (A-Buffer, voxellisation..), plutôt que de rechercher une exactitude dans les calculs : on sait par exemple que l'hémicube est fortement lié au problème de l'aliassage (nous avons réduit fortement ce phénomène en utilisant un A-Buffer dans le calcul des facteurs de forme). *Phoenix* calcule donc une image de synthèse, en utilisant l'**illumination globale** et la simulation des transferts d'énergie entre surfaces.
Phoenix utilise également le hardware graphique de certaines stations *Silicon Graphics* pour optimiser certains calculs[3]. *Phoenix* dispose d'une partie parallélisée (sur les facteurs de forme)[32, 61, 86, 87, 104, 105, 58, 224, 229, 29, 123, 160, 33, 198, 207], et permet l' utilisation de supercalculateurs Origin 2000 ou d'autres machines SGI multiprocesseurs[4]. Notre logiciel est également multi-threads, puisqu'il fait fonctionner sa lourde interface graphique, en même temps qu'il calcule

[1] *Lightscape* de *Autodesk* et *Radiance* [238] de Greg Ward sont des exemples de logiciel de calcul d'images de synthèse photoréalistes.
[2] Nous avons réalisé cette thèse sur une SGI Octane 2xR12000 300Mhz.
[3] Nous sommes notamment les auteurs d'une page Web, qui fait référence en matière d'*offscreen rendering* (rendu en hardware sans ouverture de fenêtre). Plus de détails sont donnés au paragraphe 4.2.2.1.2.
[4] Les comparaisons effectuées plus loin dans ce manuscrit entre différents logiciels de rendu réaliste ont été réalisées, bien entendu, dans les mêmes conditions expérimentales.

les images[5]. Ceci permet notamment, d'examiner en temps réel la luminance des surfaces, durant les itérations du calcul de radiosité.
Nous présentons rapidement, au paragraphe 4.3.2 l'interface graphique de *Phoenix*, qui s'est révélée indispensable au fur et à mesure de l'évolution de notre logiciel de synthèse d'images : nous avons en effet besoin de suivre en temps réel le comportement des réflectances des surfaces et leur vitesse de convergence, tout en comparant les images regénérées avec l'image réelle, sans passer par un ensemble d'outils de conversion extérieurs et fastidieux.

4.2 Méthode de rendu réaliste en deux passes

4.2.1 La structure globale

Le rendu réaliste à proprement parler de *Phoenix*, est construit comme beaucoup d' autres logiciels sur un modèle en deux passes [233, 231, 117, 197, 192, 119, 166].
La première passe est consacrée à la résolution de l'équation de radiosité, par la méthode de la radiosité progressive[38, 34, 170, 234, 165, 163, 172, 171, 235, 255, 118, 254, 189] et des subdivisions adaptatives[2, 3, 99, 227, 9, 122, 158, 232, 75, 76, 155, 206, 228, 37], tout en prenant en compte des fonctions de réflectance non diffuses, comme pour des miroirs ou des surfaces *glossy*.
La seconde passe est la passe de calcul d'une image (dite "passe de l'oeil"). Celle-ci s'effectue par lancer de rayons[244, 111, 5, 80, 53, 169, 162, 161, 175, 78, 124, 141, 142, 154, 77], et permet de prendre en compte les surfaces complexes. Nous avons naturellement implémenté tout un ensemble d' optimisations[240, 81, 82, 6, 113, 79, 153, 74, 205, 253, 242, 90, 4, 10, 248, 204, 230, 114, 89, 221, 250, 36, 106, 249, 67, 115, 96, 35, 134, 209] du lancer de rayons (boîtes englobantes hiérarchiques, BSP, voxellisation, etc.). L'ensemble du logiciel repose par ailleurs sur le modèle d'illumination de Ward[236], largement décrit au chapitre 1.

4.2.2 Première passe : passe de radiosité

4.2.2.1 Calcul des facteurs de forme

La résolution de l'équation de radiosité passe par une lourde étape de calcul, celle des facteurs de forme. Dans cette étape, la plupart du temps reste consacrée au calcul du terme de visibilité, chargé d'estimer la manière dont se voient deux surfaces. Il existe pléthore de méthodes pour estimer ce facteur de configuration[59, 146, 194, 40], certaines étant reconnues pour leur précision de calcul (lancer de rayons par exemple), d'autres pour leur vitesse (hémicube, plan unique, complexe de visibilité, etc.)[6]. Nous avons retenu la méthode de l'hémicube pour deux raisons. Tout d'abord, comme nous cherchons un logiciel rapide, nous ne pouvons utiliser le tracé de rayons, qui malgré de nombreuses optimisations n'atteint jamais la vitesse des méthodes projectives comme le Z-Buffer par exemple. L'hémicube présente l'énorme avantage d'être représentable sous la forme de 5 plans, sur lesquels se projettent successivement les facettes de la scène. Il devient donc possible d'utiliser le Z-Buffer, ou le A-buffer pour traiter le facteur de visibilité. Ceci est d'autant plus intéressant que beaucoup de machines aujourd'hui possèdent le Z-Buffer câblé, ce qui devrait théoriquement réduire les temps de calculs de manière considérable. Le second avantage est en fait lié au problème principal de l'hémicube : l'aliassage. Comme nous utilisons des techniques projectives pour calculer le facteur de visibilité, nous proposons d'employer le A-Buffer pour résoudre les problèmes d'aliassage, sachant qu'il est extrêmement rapide (quasiment aussi rapide qu'un Z-Buffer non câblé voire qu'un Z-Buffer câblé suivant certaines conditions (voir paragraphe 4.2.2.1.2)), mais très complexe à implémenter. C'est pour cette raison que nous détaillons, un peu plus loin, notre façon de le programmer (une description différente est également disponible dans [239]).

[5] Comme nous disposions d'une machine bi-processeurs, nous avons choisi d'utiliser cette fonctionnalité, en lançant un thread pour l'interface graphique sur un processeur, tandis que les calculs sont réalisés sur l'autre.

[6] Nous ne référençons pas ici tous les articles sur le calcul des facteurs de forme tant ils sont nombreux. De plus, de nombreux tours d'horizon sont disponibles comme dans [194, 40].

4.2.2.1.1 Hémicube antialiassé

L'hémicube nécessite donc la projection successive de toutes les facettes de la scène, sur chacune de ses cinq faces, divisées en **cases élémentaires**. Pour chacune de ces faces, on trie suivant leur profondeur chaque élément projeté pour chacune des cases, afin de déterminer le plus proche. Cette étape est réalisable par une méthode projective comme le Z-Buffer (câblé de préférence et décrit au paragraphe 4.2.2.1.2) ou le A-Buffer.

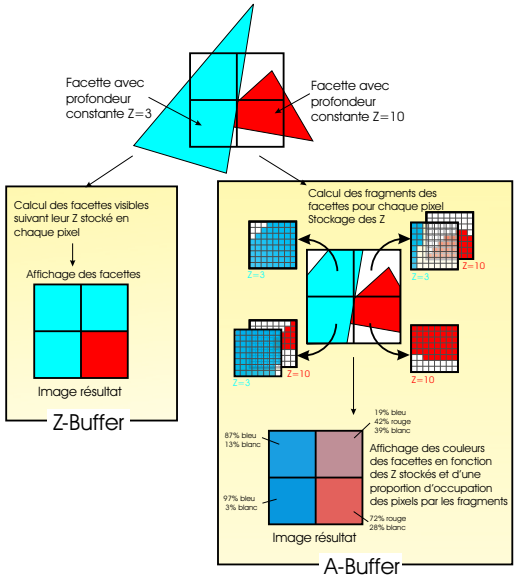

FIG. 4.1 – Comparaison entre le Z-Buffer et le A-Buffer. Dans le cas du A-Buffer, on remarque que la couleur finale (ou le facteur de forme si on travaille sur un hémicube de 1 cm de côté) est affiché au prorata du pourcentage d'occupation du pixel par la facette.

Le Z-Buffer[31] classique travaille avec un écran (ou une face d' hémicube) subdivisé en pixels (ou en cases élémentaires), tandis que le A-Buffer[30, 63, 64, 183, 107] utilise la même structure de données, en lui adjoignant une information supplémentaire de proportion d'occupation du pixel (ou de la case élémentaire), pour chaque élément visible s'y projetant (voir figure 4.1) : ceci crée donc une liste de fragments par pixel. Cette information revient à subdiviser le pixel (ou la case élémentaire) en n éléments plus petits, qui sont généralement fonction de l'architecture de la machine. L'implémentation habituelle du A-Buffer s'effectue en effet sur 32 bits, soit 4×8 sous-pixels (ou sous-cases élémentaires). Nous avons choisi de tirer partie de l'architecture même de l'*Octane*, qui est une station 64 bits et pour laquelle notre A-Buffer a des pixels(cases) subdivisés en 8×8 sous-pixels (sous-cases). L'avantage immédiat que l'on constate dans l'utilisation de ce A-Buffer, est que nous allons calculer des hémicubes dont la résolution initiale est en fait 64 fois supérieure à la résolution choisie, pour un temps de calcul supplémentaire infinitésimal. En pratique, nous utilisons des hémicubes de résolution 512×512, soit en fait de taille 4096×4096, ce qui nous affranchit de beaucoup de phénomènes d'aliassage : nous n'évitons pas cependant l'aliassage, qui survient lors de la projection d'une facette trop petite ou très éloignée, sur une des faces de l' hémicube (la taille projetée de cette facette peut alors être plus petite[7] qu'une sous-case élémentaire d'un hémicube, ce qui nous ramène au problème classique).

Méthode du A-Buffer

[7] La surface projetée doit être inférieure à $2.4 \cdot 10^{-7}$ cm^2 (taille d'une sous-case élémentaire d'un hémicube), pour que le A-Buffer commette des erreurs visibles d'aliassage.

Dans un premier temps, nous construisons 5 A-Buffer correspondant aux 5 faces de l'hémicube. Chacun de ces A-Buffer dispose d'une même position d'observation (qui est le centre du patch sur lequel se trouve l'hémicube), d'une focale de valeur unitaire, d'une direction qui varie suivant la face concernée (voir figure 4.2), et d'une grille divisée en $n \times n$ cases élémentaires, elles-mêmes codées sur 64 bits pour simuler une subdivsion en 8×8 sous-cases (comme expliqué précédemment). Dans un second temps, toutes les facettes sont passées dans le repère de la face de l'hémicube, auquel nous nous intéressons (nous n'explicitons pas ici cette opération qui est triviale). Nous projetons ensuite chacune de ces facettes sur la grille de cette face, en les triant suivant leurs distances au centre du patch, et remplissons au fur et à mesure les cases élémentaires qu'elles occupent dans la grille[8], par un facteur d' occupation et une profondeur Z. Chacun de ces facteurs d'occupation est calculé par une succession d'opérations plus ou moins complexes, que nous proposons d'expliquer maintenant (le Z de chaque case est calculé par interpolation des Z des 3 sommets de la facette comme en Z-Buffer).

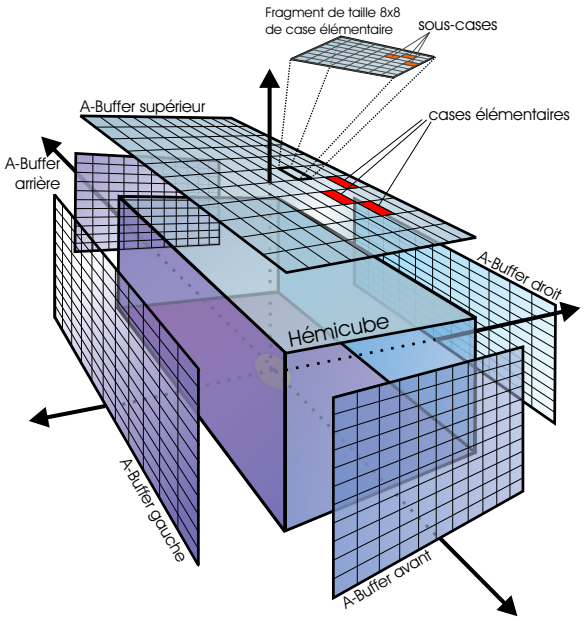

FIG. 4.2 – Construction des 5 A-Buffer, pour chacune des faces de l'hémicube

Le calcul de la proportion d'occupation pour une facette seule s'effectue en deux phases, plus une pré-phase qui consiste à précalculer, de manière définitive, l'ensemble des **fragments primaires** possibles, en connaissant l'entrée et la sortie d'un pixel (ou case). La figure 4.3 montre quelques fragments calculés pour des segments différents. Chacun de ces fragment est stocké dans un tableau de taille 64×64 (64 entrées pour 64 sorties possibles pour un segment traversant un pixel), et sous la forme d'un entier long non signé, de 64 bits. Certains fragments, comme ceux se trouvant aux coins des facettes par exemple, sont en fait des combinaisons de plusieurs fragments primaires, comme nous l'expliquerons plus loin.

Pendant la première phase, nous traçons les bords de la facette un par un sur la grille de la face de l'hémicube, en calculant la direction d'évolution la plus rapide en x ou en y pour chacune des 3 arêtes, ainsi qu'un facteur de déplacement le long de cette arête. Ce facteur nous permet de faire évoluer la position courante de déplacement sur l'arête, tout en détectant si on intersecte une ligne verticale ou horizontale de la grille. En fait cela revient à estimer les entrées et les sorties à

[8] La partie de pixel qu'occupe une facette, est généralement connue sous le nom de *fragment*.

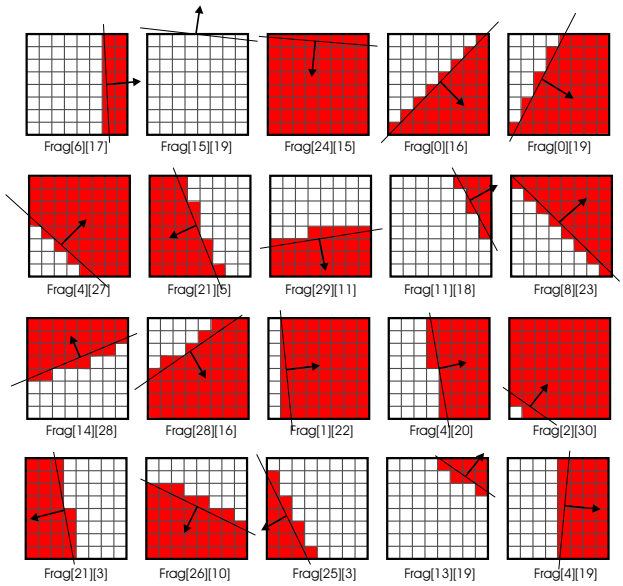

FIG. 4.3 – Exemples de fragments précalculés et stockés dans une table, dont le premier index indique le numéro de sous-pixel (ou de sous-case), par lequel le segment rentre dans le pixel (ou dans la case élémentaire), tandis que le second index précise le numéro par lequel ce segment sort de ce pixel. Les flèches indiquent l'intérieur de la facette, qui est toujours orientée dans le même sens (indirect ici).

l'intérieur d'une case élémentaire de la grille(voir figure 4.4).

Cette opération est très rapide, car il s'agit seulement d'additions et de multiplications principalement (à l'exception de deux divisions par arête pour calculer la pente, et de quelques tests pour connaître le sens de parcours). Par ailleurs, pendant cette première étape, on constate qu'il ne suffit pas de calculer les fragments des pixels traversés par les arêtes d'une facette, pour obtenir les fragments définitifs de chacun de ces pixels.

En effet, les fragments correspondant aux coins d'une facette, par exemple, ne sont pas précodés dans notre table, puisque ceux-ci sont le résultat d'un calcul en fonction d'une entrée dans le pixel et d'une sortie : il ne peut donc y avoir de point d'inflexion comme sur le fragment bleu de la figure gauche 4.5, ni de sous-cases vides comme sur le fragment rouge de la figure droite 4.5. C'est pour cette raison que ces fragments sont le résultat d'une combinaison de plusieurs fragments précalculés dans notre table, comme le montre la figure 4.6.

Par ailleurs, on pourrait penser que cette opération de combinaison n'est utile qu'aux sommets d'une facette. Cependant, on remarque bien que le fragment rouge de la figure 4.5 n'a pas été calculé pour un sommet. Ainsi, pendant le remplissage d'une facette, et le calcul des fragments de ses arêtes, une opération booléenne doit être effectuée chaque fois qu'un nouveau fragment est calculé (voir figure 4.7).

Les processus que nous venons de voir, permettent de calculer les fragments pour une seule facette se projetant sur une face de l'hémicube. Ceci constitue la première partie du A-Buffer, tandis que la seconde étape, que nous allons maintenant décrire, explique comment traiter plusieurs facettes à la fois d'une part, et, d'autre part, comment calculer la contribution de chacun des fragments, lorsque plusieurs appartenant à des facettes différentes, se projettent sur le même pixel.
Chaque fragment de facette, calculé pour un pixel (ou une case d'hémicube ici), est inséré dans la liste correspondante des pixels (ou des cases) de la grille du A-Buffer. En effet, le A-buffer, contrairement au Z-Buffer, conserve l'ensemble des morceaux des facettes se projetant sur chaque pixel, dans une liste triée suivant la profondeur calculée pour chacun de ces morceaux. La liste est alors

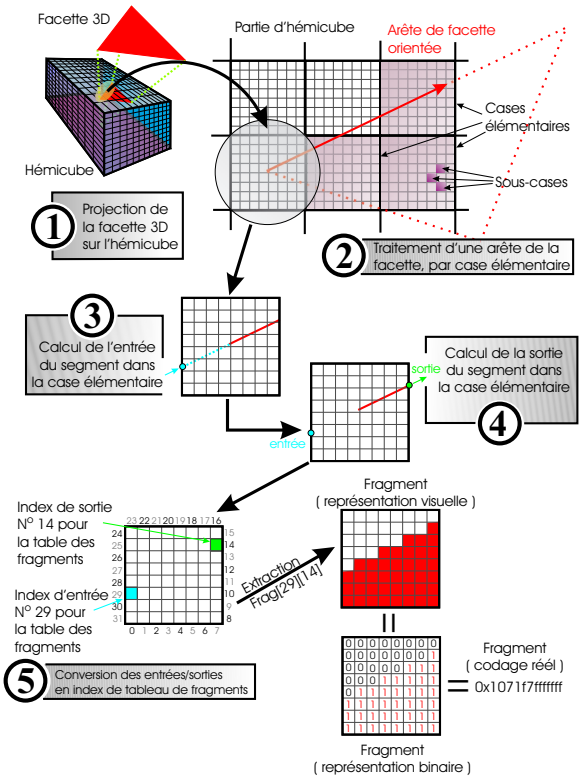

FIG. 4.4 – Schéma de calcul d'un fragment pour une portion d'arête appartenant à une facette, sur un hémicube.

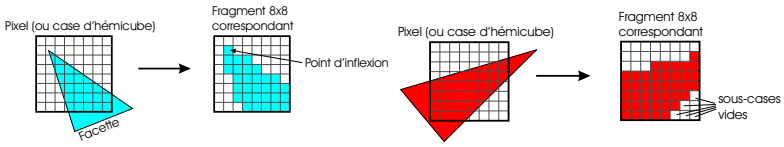

FIG. 4.5 – Exemples de fragments non précalculés par la table.

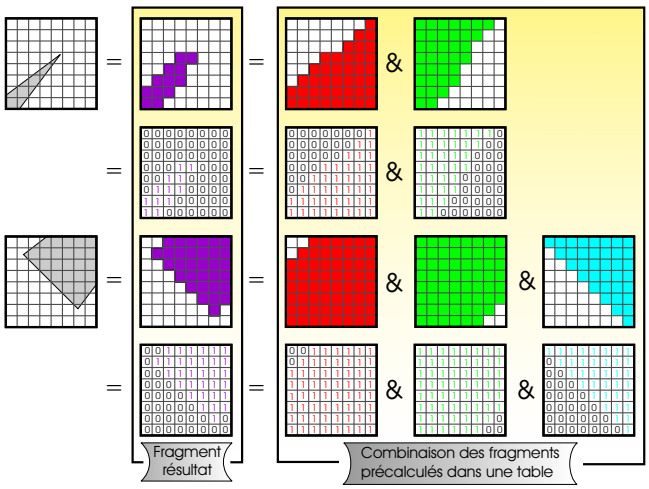

FIG. 4.6 – Exemples de fragments résultats de la combinaison logique de fragments précalculés

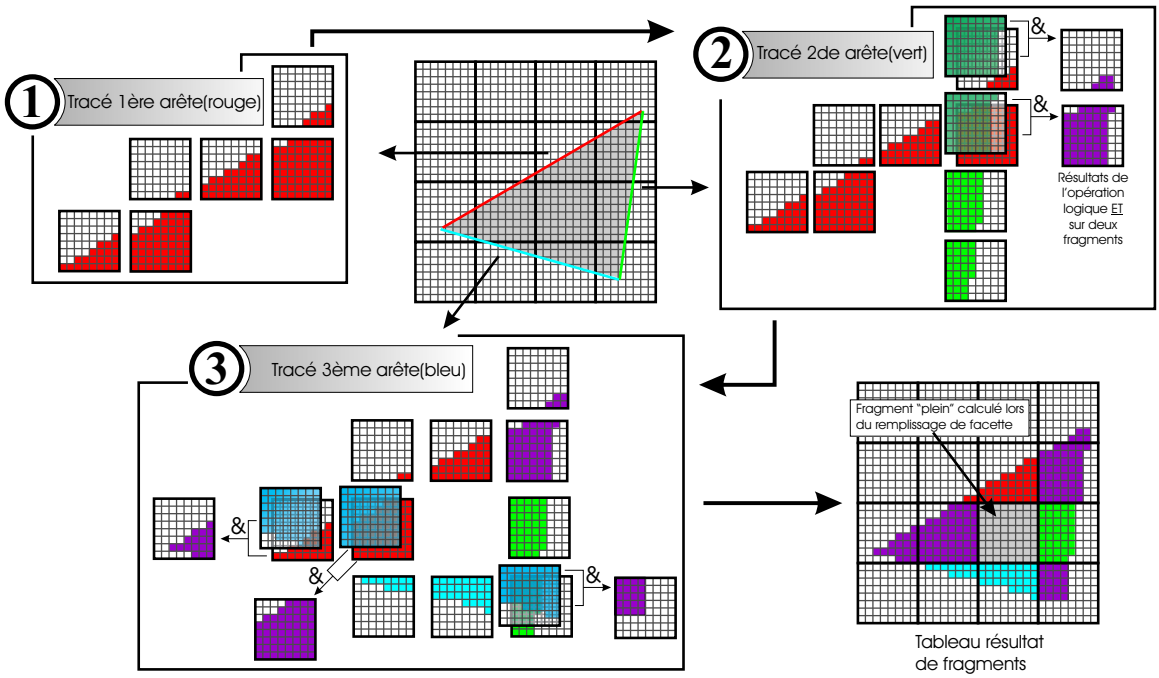

FIG. 4.7 – Processus total de calcul des fragments d'une facette. Certains fragments de pixel (aux sommets de la facette notamment) sont calculés par un ET logique entre les différents fragments intervenant dans ce pixel. Lorsque l'étape de remplissage de la facette a lieu, les pixels à l'intérieur de la facette possèdent un fragment dit "plein", c'est-à-dire un entier dont tous les bits sont à 1.

parcourue en chaque pixel (ou case d'hémicube), et les fragments de facettes sont affichés au prorata de leur proportion d'occupation du pixel (ou case d'hémicube) : plusieurs opérations logiques sont une fois de plus réalisées (voir figure 4.8 et algorithme 13) pour connaître la contribution de chacun de ces fragments.

Algorithme 13 Algorithme de calcul de la couleur d'un pixel depuis une liste de fragments

```
Procédure Calcul_Valeur_Pixel(ABuffer *grille)
  Pour Tous les pixels i de la grille Faire
    valeur_pixel = 0 ;
    cumul_frag = 0x0 ;
    Pour Tous les fragments de la liste triée pour ce pixel Faire
      //grille[i].liste_fragment→frag contient la valeur du fragment
      //grille[i].liste_fragment→couleur contient la couleur du fragment
      Si grille[i].liste_fragment→frag Alors
        contribution_frag = Contribution_Fragment(grille[i].liste_fragment→frag) ;
        contribution_frag *= (1/64) ;
        valeur_pixel += (grille[i].liste_fragment→couleur×contribution_frag) ;
        //Mise à jour du fragment cumulé actuel de ce pixel
        cumul_frag |= grille[i].liste_fragment→frag ;
      Fin si
    Fin
  Fin
Fin Procédure
```

La procédure d'évaluation du nombre de bits à 1, permettant de calculer cette contribution

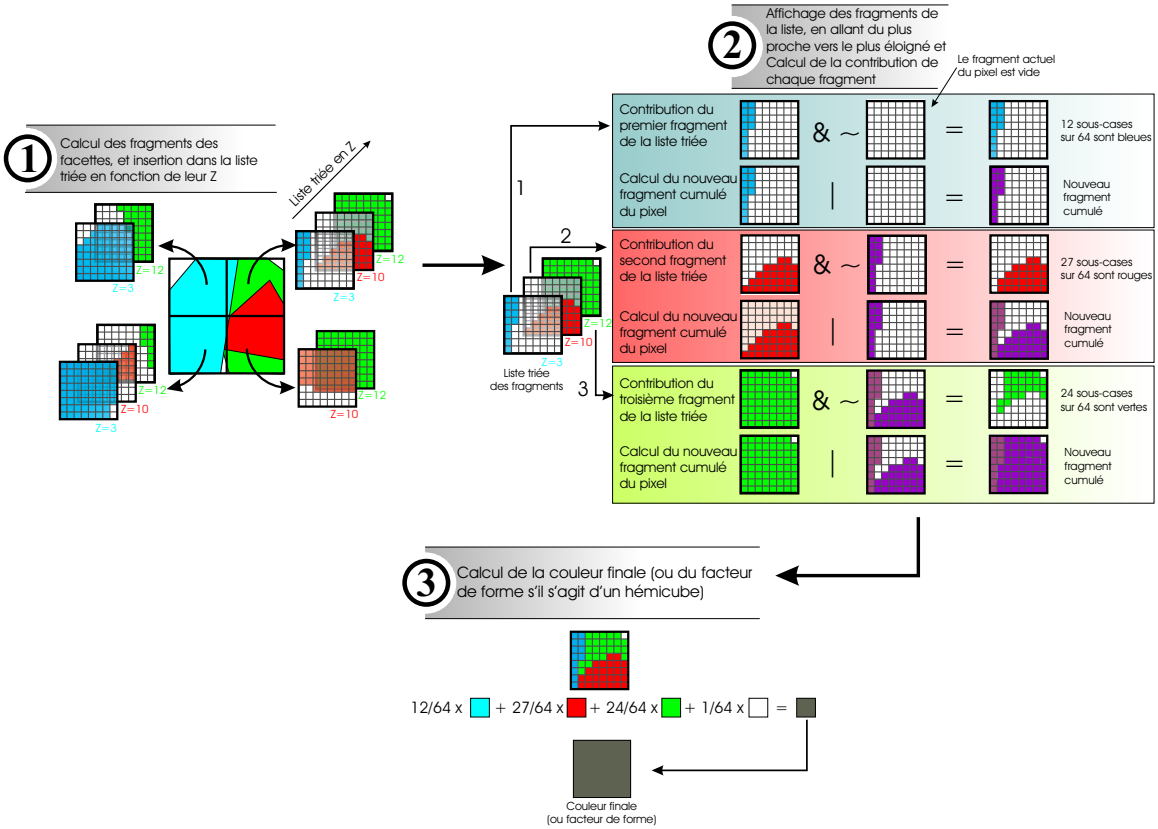

FIG. 4.8 – Calcul et affichage de la couleur (ou du facteur de forme) finale, en calculant la contribution de chacun des fragments se projetant sur un pixel (ou une case d'hémicube)

est donnée par l'algorithme 14. En principe, cela consiste à effectuer 64 décalages de bits pour détecter chacun des bits à 1 dans le fragment, et à incrémenter le facteur d'occupation chaque fois qu'un 1 est rencontré. Ces décalages très nombreux (64 par fragment) peuvent être optimisés, si on précalcule dans une table statique, l'ensemble des occupations pour toutes les combinaisons possibles de 1 et de 0 sur une ou plusieurs lignes du fragment. À l'aide de cette table, on procéde alors à des décalages successifs ligne par ligne (par rangée de 8 bits et non plus un par un), sachant que la taille d'une telle table est de l'ordre de $2^{8\times m}$ octets, avec m le nombre de lignes. Par exemple, si on ne précalcule qu'une ligne entière, il faudra procéder à 8 décalages successifs (au lieu de 64) pour connaître l'aire occupée par le fragment. Dans *Phoenix*, nous procédons en 4 fois, soit une table statique de taille 65Ko environ, pour stocker deux lignes du fragment.

Algorithme 14 Algorithme de calcul de la contribution d'un fragment

```
Procédure entier Contribution_Fragment(entier_long frag)
  //On lit les 16 premiers bits et on récupère le nombre de bits valant 1
  //Table_Bits_1 est une table précalculée d'entiers, contenant le nombre de bits à 1,
  //pour des valeurs allant de 0 à 65536 (2 lignes de fragments)
  taille_1 = Table_Bits_1[ frag & 0xffff ] ;
  //On lit les 16 premiers bits et on récupère le nombre de bits valant 1
  taille_2 = Table_Bits_1[ (frag & 0xffff0000) » 16 ] ;
  //On lit les 16 premiers bits et on récupère le nombre de bits valant 1
  taille_3 = Table_Bits_1[ (frag & 0xffff00000000) » 32 ] ;
  //On lit les 16 premiers bits et on récupère le nombre de bits valant 1
  taille_4 = Table_Bits_1[ (frag & 0xffff000000000000) » 48 ] ;
  retourne( taille_1 + taille_2 + taille_3 + taille_4 ) ;
Fin Procédure
```

Le A-Buffer, programmé sous cette forme, est extrêmement rapide, car la plupart des opérations à effectuer sont représentées par des opérations booléennes au temps de calcul négligeable. Cependant, lors de la création d'une liste de fragments en un pixel, nous avons une opération de parcours et d'insertion de fragments dans une liste triée suivant des valeurs de profondeurs. Cette opération pourrait être très lourde à gérer, si certaines optimisations n'étaient pas réalisées. En effet, supposons que nous ayons une scène comportant plusieurs centaines de milliers de facettes, il est probable, suivant la configuration de cette scène, que beaucoup de facettes risquent de se projeter sur le même pixel. Supposons que ce soit le cas, et que, par exemple, 5000 polygones, se projettent sur le même pixel. Nous avons alors à gérer une liste de 5000 fragments pour un seul pixel ! Cette situation n'est pas acceptable en terme de temps machine, ni d'espace mémoire utilisé, surtout si elle est reproduite pour tous les pixels de la grille. Nous proposons donc plusieurs optimisations. Tout d'abord, lorsqu'un élément doit être inséré dans la liste des fragments, nous calculons le masque cumulé des fragments précédant l'élément à insérer. Ainsi, si ce masque cumulé est "plein" (tous les bits du masque sont à 1), alors nous ne procédons pas à son insertion, puisque le fragment à insérer sera de toute façon masqué par les autres. Ensuite, si nous pouvons insérer cet élément (il reste donc une partie de ce fragment qui est visible, malgré les autres fragments se projetant avant lui), nous calculons le fragment cumulé en tenant compte de cette insertion (le fragment cumulé est la somme des fragments précédant l'élément inséré, lui y compris). Si le masque ainsi cumulé devient "plein" avant d'avoir atteint la fin de la liste, nous détruisons alors la totalité du reste de la liste, qui représente des éléments non visibles. Ces deux optimisations garantissent deux choses : nous ne pouvons avoir une liste de plus de 64 éléments (une facette par sous-pixel (ou sous-case) donc, ce qui est en plus hautement improbable, en pratique), et nous ne conservons pas les fragments inutiles, c'est-à-dire ceux qui seront masqués par d'autres fragments les précédant dans la liste. L'illustration de ces deux optimisations est fournie par la figure 4.9.

Nous pouvons néanmoins étendre la seconde optimisation à un cas plus général. En effet, nous calculons ici le fragment cumulé de tous les éléments précédant le nouveau fragment inséré[9]. S'il est

[9] Le calcul du fragment cumulé des éléments précédant l'élément à insérer est très rapide, car cette opération est

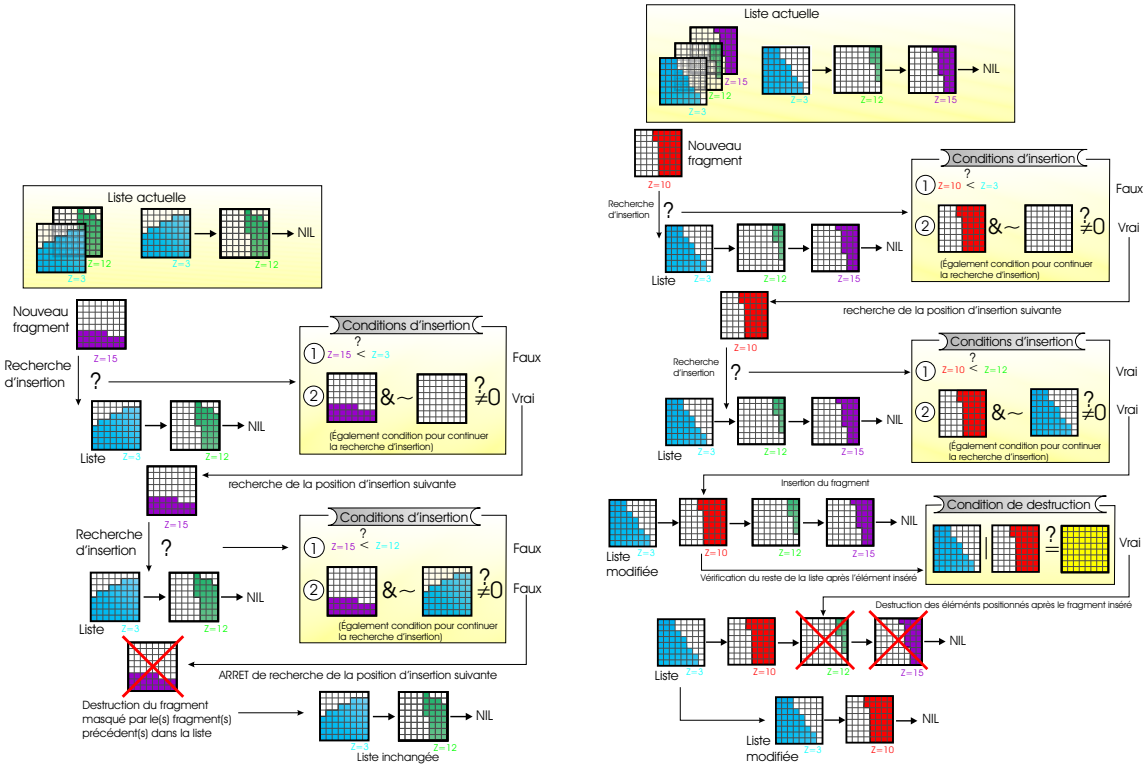

FIG. 4.9 – Les deux optimisations effectuées sur la création des listes de fragments pour un pixel. À gauche, le nouveau fragment est détruit avant d'être inséré, car il sera masqué pas un fragment le précédant dans la liste. À droite, l'insertion d'un nouveau fragment, et sa combinaison logique avec les fragments le précédant, produit un masque "plein" (tous les bits du fragment cumulé sont à 1). Ceci a pour effet la destruction de tous les fragments suivant l'élément inséré, puisqu'ils sont masqués.

plein, nous détruisons la liste qui le suit, sinon nous ne faisons rien. Nous pourrions donc proposer de continuer de parcourir la liste des fragments et de continuer à calculer le masque cumulé, afin de vérifier si, plus loin dans la liste, certains fragments ne sont pas masqués maintenant par cette insertion (le fragment inséré peut masquer un élément se trouvant 6 ou 7 places plus loin dans la liste par exemple). Cependant, nous n'avons pas implémenté cette optimisation sous cette forme, car elle nécessite un parcours supplémentaire de la totalité de la liste pour chaque pixel (ou case d'hémicube), et chaque nouvelle insertion, pour finalement ne détruire les éléments que un par un. Cette opération est donc très coûteuse, pour un gain d'espace mémoire généralement faible (sauf rares exceptions). Néanmoins, si on souhaite garder le A-Buffer calculé pour d'éventuelles optimisations futures ou d'autres réutilisations, nous pouvons réaliser cette troisième optimisation, après avoir créé le A-Buffer en totalité, ce qui nous évite de parcourir la liste en totalité pour chaque insertion (nous ne le faisons plus qu' une seule fois pour chaque pixel).

FIG. 4.10 – Exemple d'image de résolution 768×576 calculée par Phoenix, en 36 minutes (dont 23 de lancer de rayons en seconde passe). Cette scène comporte 155000 facettes et 10 rayons sont lancés par pixel. Lorsque la surface est *glossy*, 100 rayons sont ici relancés pour l'intégration de l'angle solide, soit 1000 rayons pour chaque pixel recouvert par la plaque d'aluminium au centre de l'image.

Le temps total de rendu d'un hémicube comportant 5 A-Buffer pour une scène complexe d' intérieur avec 155000 facettes, est de l'ordre de 2 à 13 secondes au maximum (lorsque toutes les facettes de la scène sont devant la position d'observation). Ce qui signifie donc qu' il faut environ 5 secondes à Phoenix pour calculer 155000 facteurs de forme. L'algorithme de rendu par radiosité progressive étant dépendant du seuil d'erreur que l'on fixe ($10^{-4}\,W/sr/m^2$ le plus souvent dans notre logiciel), et donc du nombre d'itérations de radiosité, il faut à Phoenix $n \times 5$ secondes pour calculer une image photoréaliste (le temps de résolution de la matrice de radiosité est négligeable), n étant le nombre d'itérations. Pour une image comme celle de la figure 4.10, il faut compter environ 13 minutes de rendu total, plus 23 minutes pour le lancer rayons, en sachant qu'on a lancé ici 10 rayons par pixel, et 100 par angle solide pour la surface rugueuse à BRDF complexe (plaque d'aluminium sur la figure 4.10).

réalisée en même temps que le test d'insertion sur sa profondeur : nous n'avons donc pas à parcourir la liste en sens inverse pour le connaître.

4.2.2.1.2 Accélérations hardware

La motivation initiale du choix de l'hémicube pour le calcul des facteurs de forme reposait sur un passage du livre [40] page 105, dans lequel il est précisé que l'utilisation du hardware des SGI permet d'accélérer le calcul des facteurs de forme de façon drastique. En effet, comme le Z-Buffer est câblé sur les SGI, et que ces stations sont capables d'en calculer plusieurs par seconde, il devient possible d'obtenir des temps de calcul proches du temps réel, en théorie (typiquement pour l'image 4.10, on aurait pour 100 itérations, 500 Z-Buffer à calculer, l'hémicube comportant 5 faces). Sachant que ces stations sont capables de calculer plusieurs millions de polygones par seconde, il devient à priori évident que ce calcul est temps réel. Cependant nous allons voir, que ce temps de calcul varie énormément en fonction de la résolution de la grille du Z-Buffer, ce qui peut fournir des temps d'exécution plus importants qu'il n'y paraît en première approximation. De plus, l'architecture parfois inadéquate de la station graphique employée peut rendre ce calcul plus long qu'une simple implémentation logicielle n'utilisant pas le hardware dédié (voir comparatifs plus loin).

Afin de pouvoir calculer une image (ou une table d'index) en utilisant le hardware des stations graphiques, il est indispensable de parvenir à générer une telle image sans ouvrir de fenêtre (sinon il faudrait accepter d'avoir en permanence une fenêtre ouverte et visible de taille écran pour tous les calculs) : cette opération s'appelle l'**offscreen rendering**[10]. Sur les Silicon Graphics, une telle opération n'est pas aussi simple qu'il y paraît, car aucune littérature n'est disponible sur ce sujet : même le livre de référence sur OpenGL[145] ne consacre que quelques lignes à ce problème, et elles n'apportent rien qu'on ne sache avant de les lire. Nous avons donc essayé toutes les méthodes existantes[11] pour calculer une image par offscreen-rendering, sachant que nous recherchions la méthode la plus rapide possible, en terme de temps d'exécution. Pour ce faire, nous avons testé 3 librairies différentes : OpenGL, la librairie publique Mesa[12](implémentation multi-plateforme des fonctions OpenGL avec les mêmes noms, mais avec quelques distinctions sur la gestion des *buffers*) et *Open Inventor*.

Tout d'abord, avant de comparer ces librairies d'un point de vue vitesse, il convient de comprendre un phénomène précis, concernant l'offscreen-rendering. En effet, suivant la technique qui est employée, il est fréquent que le fait de ne pas ouvrir de fenêtre empêche l' utilisation du hardware dédié de la station : tous les calculs sont alors réalisés par le processeur, ce qui rend l'opération beaucoup moins intéressante. Cette propriété est connue en programmation avec OpenGL, sous un paramètre booléen appelé *direct*, qui prend la valeur *GL_TRUE* ou *GL_FALSE*, suivant que l'on désire utiliser ou non une connexion directe vers la carte graphique, ou passer par le serveur X. Malheureuseument, ce booléen doit être systématiquement placé à *GL_FALSE*, si on veut pouvoir utiliser l'offscreen-rendering[13] sauf dans un cas unique : les *P-Buffers* sur lesquels nous reviendrons plus loin.

Lorsque la scène est calculée sur la carte graphique (élimination de parties cachées et stockage des index dans un tableau), nous devons procéder à la lecture du *buffer* résultat (la table d'index ici), via une fonction OpenGL appelée *GlReadPixels()*. Si l'exécution de cette fonction est très rapide avec l'utilisation directe du hardware (et donc en ouvrant une fenêtre), il lui faut de deux à quatre secondes en offscreen-rendering, pour récupérer un buffer de taille 1024×1024 (et une à deux secondes pour un buffer de taille 1024×512) ! Ceci signifie donc que si nous gardons cette

[10] Nous n'avons pas trouvé l'équivalent français de ce terme. Nous préférons donc garder le terme anglais original, plutôt que de tenter de fournir une traduction approximative, et dénuée de sens pour l'utilisateur qui connaît cette technique.

[11] Il existe cependant une librairie que nous n'avons pas testée : il s'agit de la libraire SGI *Performer*, qui nécessitait un temps de développement supplémentaire que nous n'avions pas.

[12] http ://www.mesa3d.org/

[13] Si les valeurs des paramètres pour l'offscreen-rendering et notamment celle du booléen *direct*, sont placées à des valeurs erronées, que la carte graphique ne peut supporter, cela produit alors un message d'erreur du serveur X, qui détruit le processus en cours.

résolution de 1024 × 1024 pour notre hémicube, il faut à *Phoenix* entre 6 et 12 secondes pour récupérer les buffers calculés par offscreen-rendering[14]. En utilisant notre A-Buffer, *Phoenix* calcule des facteurs de forme pour les cinq faces de l'hémicube avec une résolution de 4096 × 4096 en 13 secondes au total. Si nous appliquons la même résolution pour l'offscreen-rendering, nous passons alors entre 96 et 320 secondes pour obtenir la même qualité que notre A-buffer précédent. De plus, le temps d'exécution de notre A-Buffer peut descendre à 2 secondes, si le nombre de facettes placées devant l'observateur est réduit (les autres facettes ne sont pas traitées, car elles sont directement éliminées par le volume de vision dont elles ne font pas partie).
Pour résumer, notre A-Buffer est donc entre 7 et 160 fois plus rapide qu' un offscreen-rendering utilisant la librairie OpenGL, pour une scène de 155000 triangles et exactement dans les mêmes conditions expérimentales. En moyenne, le facteur d'accélération est de 90 en faveur de notre A-Buffer.

On constate donc que la lecture du *frame buffer* d'une SGI est très lourde et pénalise fortement les temps de calcul, lorsque l'utilisation de l'offscreen-rendering ne permet pas l' emploi du hardware de ces stations[15]. En effet, lors d'un appel à la fonction *GlReadPixels()*, le contenu du *frame buffer* utilise un type particulier de *buffers* graphiques X11 appelés **pixmaps** , qui sont à l'origine même du ralentissement considérable du processus de récupération de l'image calculée. SGI a donc tenté de pallier ce défaut majeur, en implémentant une version hardware des *pixmaps*, pour permettre à la fonction de lecture des *frame buffers* de s'exécuter en temps réel : les **P-Buffers**. Ces P-Buffers présentent l'avantage énorme de pouvoir utiliser le booléen *direct* en mode *GL_TRUE*, ce qui nous assure l'utilisation du hardware dédié des SGI, aussi bien sur le calcul de l'élimination des parties cachées, que sur l'exécution de la fonction *GlReadPixels()*. En pratique, nous avons implémenté les *P-Buffers* dans *Phoenix*. Les temps de calcul OpenGL passent alors à 20 secondes pour lire et calculer 100 *frame buffer* de taille 1024 × 1024 (curieusement, le temps varie de façon non significative pour des buffers de taille 1024 × 512), soit un facteur d'accélération de 10 environ. A priori, les P-Buffers permettent le calcul de 20 itérations de radiosité[16], pendant que notre A-Buffer n'en effectue que 2. Malheureusement, notre A-Buffer fournit une fois de plus une précision dans le calcul des facteurs de forme, 16 fois supérieure aux P-Buffers employés ici. Ainsi, si nous voulons la même précision avec les *P-Buffers*, il faut calculer 16 buffers par face d'hémicube pour obtenir une résolution de 4096 × 4096 [17]. En pratique, dans *Phoenix*, les *P-buffers* mettent 1600 secondes pour calculer 100 itérations de radiosité, alors que le A-Buffer ne met que 1000 secondes environ.

Les deux autres librairies utilisées pour l'offscreen-rendering sont *Mesa* et *Open Inventor*. Dans le cas de *Mesa*, nous obtenons des temps de calculs du même ordre que les P-buffers 1024 × 1024. Ceci tient au fait que le temps de rendu par Z-buffer d'une scène de 155000 facettes est très rapide, même sans hardware, mais surtout, il n'est pas besoin d'utiliser la fonction *GlReadPixels()* car *Mesa* gère les buffers d'offscreen-rendering en les allouant directement dans la RAM sous la forme d'un pointeur : son accès est donc instantané, quelle que soit sa taille. Cependant, comme il n'y a aucune utilisation du hardware, le temps de calcul augmente de façon linéaire avec le nombre de polygones de la scène (tout comme notre A-Buffer, mais qui reste plus rapide grâce à son sur-échantillonnage 64 bits).
Pour *OpenInventor* (emploi de la classe *SoOffScreenRenderer*), les temps de calcul sur la même scène sont les plus mauvais de tous. Cela provient clairement du fait que *OpenInventor* est une surcouche d'*OpenGL*, et possède donc les mêmes limitations que cette dernière, avec en plus sa

[14] le temps de calcul pour l'élimination des parties cachées par Z-Buffer câblé est bien inférieur au temps nécessaire à la lecture du résultat, et il est ici négligeable pour 155000 facettes.

[15] La plupart des cartes graphiques Silicon Graphics ne permettent pas l'emploi du hardware en offscreen-rendering, et c'est notamment le cas de notre Octane SI. Tous les tests de ce paragraphe ont été réalisés sur deux autres Octane, une comportant une carte Maximum Impact(MXI), l'autre plus récente possédant une carte MXE.

[16] Les P-Buffers calculent 100 buffers en 20 secondes, soit 20 hémicubes.

[17] Il est en effet impossible d'allouer des *frame buffers* de taille supérieure à 1024 × 1024. Il faut donc en calculer 16 pour obtenir une résolution de 4096 × 4096.

	320 buffers de taille 1024×1024 (= 20 buffers de taille 4096×4096)
OpenGL pixmaps (sans hardware)	20min30s
OpenInventor (sans hardware)	24min50s
Mesa 3.0 (sans hardware)	8min45s
OpenGL P-Buffers (avec hardware)	1min4s
A-Buffer	45s

FIG. 4.11 – Comparaison de notre A-buffer avec différentes techniques de rendu en offscreen-rendering pour le calcul de 320 buffers de résolution 1024×1024 (soit 20 A-Buffers 64 bits), pour une scène comportant 155000 facettes

	100 hémicubes de taille 4096×4096 ($\simeq$ 100 itérations de radiosité)
OpenGL pixmaps (sans hardware)	8h54min
OpenInventor (sans hardware)	9h30min
Mesa 3.0 (sans hardware)	3h35min
OpenGL P-Buffers (avec hardware)	26min30
A-Buffer	20min

FIG. 4.12 – Comparaison de notre A-buffer avec différentes techniques de rendu en offscreen-rendering pour 100 hémicubes de résolution 4096×4096, pour une scène comportant 155000 facettes

propre couche d'interprétation.

Il s'agit là de mesures expérimentales, que nous considérons comme fondamentales pour le rendu réaliste. En effet, beaucoup d'articles ou de livres considèrent que le temps de calcul en radiosité n'est pas un obstacle, puisque l'on peut utiliser le hardware de certaines machines pour accélérer les temps de calcul de façon importante. Nous démontrons ici, grâce à ces mesures, que cette affirmation est fausse pour des scènes dont la taille est inférieure à 200000 facettes. Bien sûr, les *P-Buffers* accélèrent fortement les calculs, si nous les laissons calculer des hémicubes de résolution 1024×1024, et que nous laissons le A-Buffer calculer des hémicubes de 4096×4096. Cependant, si nous voulons comparer notre A-Buffer aux P-Buffers de taille 1024×1024, nous devrons descendre la résolution de l'hémicube pour notre A-Buffer à 128×128 (chaque case étant elle-même suréchantillonnée en 8×8 sur un entier long de 64 bits (voir paragraphe 4.2.2.1)). Les temps de calcul d'un tel hémicube par A-Buffer restent de toute façon bien inférieurs à ceux des P-Buffers.

Il reste cependant une distinction très importante à effectuer. Le temps de calcul avec des P-Buffers est toujours extrêmement dépendant de la lecture par la fonction *GlReadPixels()*, mais reste très peu sensible au nombre de polygones (jusqu'à un certain point bien sûr). Ceci signifie que pour une carte graphique capable de traiter 10 ou 15 millions de polygones par seconde, le temps

	facteur d'accélération
OpenGL pixmaps (sans hardware)	1.0 (référence)
OpenInventor (sans hardware)	0.93
Mesa 3.0 (sans hardware)	2.5
OpenGL P-Buffers (avec hardware)	20.15
A-Buffer	26.7

FIG. 4.13 – Facteur d'accélération entre les méthodes d'offscreen-rendering (A-buffer compris)

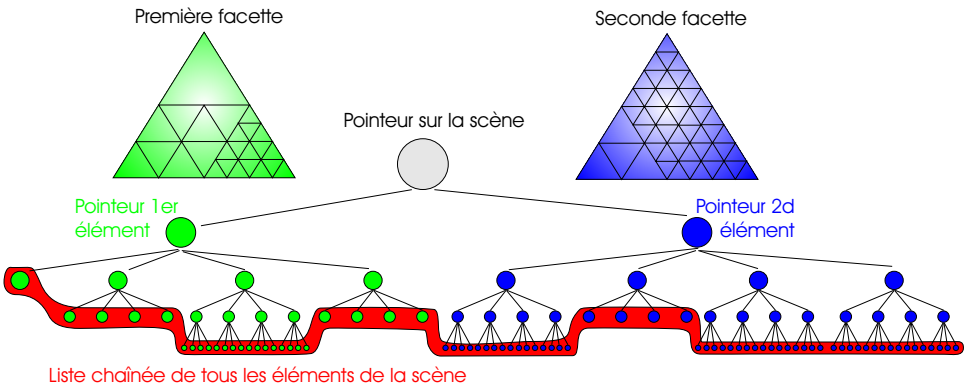

FIG. 4.14 – Structure hiérarchique de représentation des surfaces d'une scène par quadtree. Une liste chaînée (en rouge) de tous les éléments (nœuds terminaux) de la scène est construite pour un accès immédiat.

de calcul pour 20 hémicubes de résolution 1024×1024, sur une scène de 12 millions de facettes par exemple, reste de 20 secondes. Par contre, pour notre A-buffer, nous avons un phénomène différent : la lecture du résultat est bien instantanée et indépendante du nombre de polygones (comme pour les P-buffers), mais le nombre de facettes à traiter influence considérablement les temps de calcul. En fait, pour 10 ou 15 millions de polygones, nous aurons des temps de calcul de l'ordre de 1100 secondes pour 20 hémicubes de résolution 128×128 [18]. Les tableaux 4.11, 4.12 et 4.13 résument les résultats obtenus, pour une scène comportant 155000 facettes (figure 4.10).

Nous proposons en annexe B le code source en langage C et *OpenGL*, pour utiliser l'offscreen-rendering, suivant la méthode des *pixmaps* et des *P-buffers*.

4.2.2.2 Technique de radiosité

4.2.2.2.1 Radiosité progressive et surfaces diffuses

Comme expliqué précédemment, nous utilisons la méthode de la radiosité progressive[38] pour la résolution de l'équation de radiosité. Nous avons implémenté les optimisations classiques du *positive overshooting*[62] et du terme *ambiant*[38], pour accélérer la vitesse de convergence, et traiter les résidus d'énergie. Nous n'avons pas apporté de modification particulière à cette technique, si ce n'est une structure de données un peu particulière, pour éviter d'avoir à parcourir systématiquement tous les arbres des surfaces (nous utilisons une représentation hiérarchique par *quadtrees* des surfaces de la scène), alors que seules les feuilles terminales nous intéressent (voir figure 4.14).

Typiquement, lors d'une itération de radiosité progressive, nous calculons seulement une colonne de facteurs de forme, soit les facteurs de forme entre un patch émetteur et tous les éléments (feuilles terminales des arbres de radiosité) de la scène. Cette opération étant très fréquente et la taille de ces arbres souvent grande, il est très important de ne pas perdre de temps à accéder aux éléments terminaux, employés pour emmagasiner l'énergie.
Bien sûr, nous aurions pu retenir la radiosité hiérarchique qui permet d'avoir un maillage souvent plus optimal, pour une scène donnée. Nous avons conservé la radiosité progressive pour sa facilité d'implémentation d'une part (quelques lignes de code suffisent), et d'autre part, à cause de notre A-buffer qui permet de calculer des facteurs de forme très rapidement et de manière relativement précise (nous n'avons donc pas nécessairement besoin d'un maillage optimal). Nous avons par ailleurs implémenté un maillage adaptatif (voir paragraphe 4.2.2.2.5) selon la méthode de Cohen et al.[37] et avec le mappage bidimensionnel de Heckbert[99], lorsque nous souhaitons atteindre la quintessence de la précision visuelle. Si l'apport d'un maillage optimal n'est pas forcément indispensable, la précision apportée dans les calculs par le maillage des discontinuités[19] peut s'avérer un

[18] Un A-buffer 64 bits de résolution 128×128 équivaut à un Z-buffer 1024×1024 comme expliqué précédemment.
[19] *Discontinuity Meshing*[60, 215, 55, 232, 98, 213, 75, 54, 128, 206, 76, 127] dans la littérature scientifique : cette

outil puissant, que nous n'avons pas implémenté pour des raisons de temps : cela reste néanmoins une des caractéristiques futures à ajouter à *Phoenix*.

4.2.2.2.2 Rendu des surfaces spéculaires

Nous avons choisi une approche radicalement différente de celle qui consiste à réémettre immédiatement l'énergie reçue par une surface spéculaire. En effet, lorsque nous souhaitons visualiser la solution rapidement, nous n'utilisons que peu des itérations de la radiosité progressive (disons 20 par exemple). Si nous réémettons tout de suite l'énergie depuis les surfaces spéculaires, cela peut fortement ralentir les calculs, pour une visualisation rapide. Ainsi, les deux sphères réfléchissantes de l'image 4.10 possèdent plusieurs dizaines de patchs recevant de l'énergie dès la onzième itération (les surfaces émettant pendant les 10 premières étant des sources de lumière orientées vers le plafond), et demandent ainsi à réémettre leur énergie tout de suite : si nous considérons chaque réémission spéculaire comme une itération de radiosité, nous risquons d'obtenir une image très insatisfaisante car la contribution de ces surfaces n'est pas nécessairement importante par rapport à d'autres surfaces, et nous atteindrons vite le seuil des 20 itérations, en ayant émis que peu de l'énergie totale. Ceci est encore aggravé lorsque plusieurs miroirs se voient les uns les autres, comme c'est d'ailleurs le cas pour nos deux sphères, qui s'illuminent entre elles. Typiquement, pour la scène 4.10, si nous réémettons tout de suite l'énergie spéculaire, il faut attendre la $83^{\text{ème}}$ itération avant de réémettre une surface choisie par l'algorithme de radiosité progressive[20], tandis que si nous attendons que cette surface spéculaire soit sélectionnée, parce que la quantité d'énergie emmagasinée est suffisante, nous nous apercevons que la première réémission spéculaire n'intervient que beaucoup plus tardivement (à partir de la $125^{\text{ème}}$ itération de radiosité pour la première sphère de l'image 4.10).
Nous considérons donc que la méthode qui consiste à réémettre immédiatement l'énergie reçue par les surfaces spéculaires, est dommageable à la vitesse de convergence de la résolution de l'équation de radiosité, lorsqu'on ne souhaite pas émettre toute l'énergie présente dans la scène. Pour cette raison, nous avons créé une structure de données spécifique qui, pour une surface spéculaire, stocke la quantité d'énergie à réémettre, ainsi que la direction incidente de cette énergie. En effet, contrairement aux surfaces diffuses, la radiosité réémise par une surface spéculaire est directionnelle, et dépend directement de la direction d'incidence. Lorsque le cumul de cette énergie pour un ensemble d'angles solides appartenant à une surface spéculaire, est suffisant pour que cette surface soit choisie par le critère de radiosité progressive, nous réémettons la totalité de l'énergie reçue, en prenant garde de réémettre chaque quantité de radiosité dans son angle solide associé. La figure 4.15 illustre cette méthode sur un exemple.

La redistribution spéculaire de l'énergie s'effectue en deux grandes étapes. La première consiste à calculer le volume (l'angle solide en fait) dans lequel la facette spéculaire réémet son énergie. À partir du point d'émission original, nous calculons le point de réémission virtuel[168, 169] et traçons la pyramide de réémission comme pour les étapes 5 et 6 de la figure 4.15. Nous clippons alors toutes les facettes se trouvant dans ce volume[21] pour ne conserver que la partie qui reçoit de l'énergie. Les facettes se trouvant en dehors du volume ne recoivent pas du tout d'énergie depuis la facette spéculaire. La seconde étape détermine le facteur de forme entre la facette spéculaire et les facettes du volume, en utilisant un hémicube centré sur le point de réémission. Le calcul s'effectue alors de la même manière que pour une surface diffuse, par A-Buffer. La quantité d'énergie reçue par les surfaces à l'intérieur du volume est calculée au prorata de ces facteurs de forme.

---

technique permet de calculer directement les frontières d'ombre et de pénombre.

[20] En effet, l'algorithme de radiosité progressive accélère les temps de calcul en réémettant d'abord les surfaces dont la radiosité est la plus forte. Si nous réémettons immédiatement l'énergie reçue par une surface spéculaire, nous violons cette règle, et perdons ainsi un des avantages de cette technique.

[21] Nous utilisons la face spéculaire elle-même comme un des plans de clipping de ce volume, afin de ne pas traiter les facettes qui se trouveraient derrière elle.

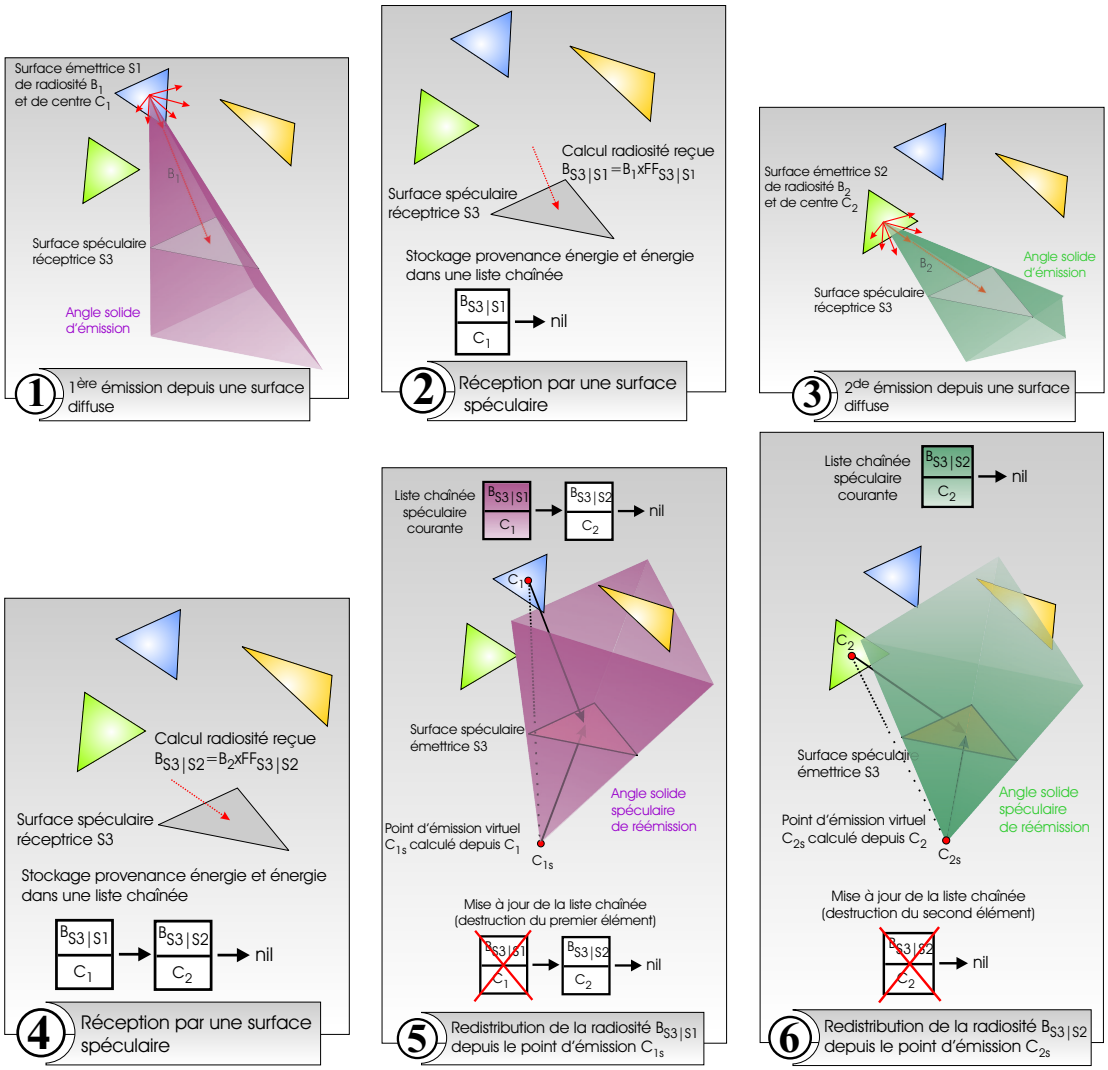

FIG. 4.15 – Processus de stockage dans une liste, de l'énergie reçue et de la direction d'émission, pour une surface spéculaire depuis une surface quelconque (étapes 1 à 4). Lorsque la surface spéculaire est sélectionnée pour réémettre sa radiosité, les radiosités stockées dans la liste sont réémises suivant les mêmes angles solides associés, et la liste est détruite (étapes 5 et 6).

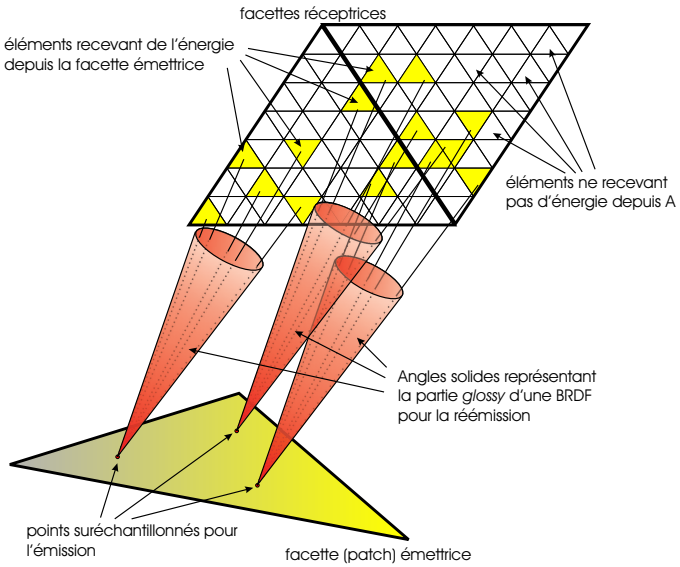

FIG. 4.16 – Exemple de facette rugueuse réémettant son énergie vers deux autres facettes, par la méthode qui consiste à tirer des rayons depuis la surface émettrice, vers les autres facettes. En ne lançant que quelques rayons à travers chaque angle solide (en rouge), certains éléments appartenant aux facettes réceptrices ne reçoivent pas d'énergie. Seuls certains éléments (en jaune) sont ainsi intersectés. Ceci produit des phénomènes d'aliassage au moment du rendu, visibles sous la forme de discontinuités. Plus les facettes réceptrices sont éloignées de l'émetteur, plus ce phénomène est amplifié.

4.2.2.2.3 Rendu des surfaces isotropes et anisotropes

En utilisant le modèle de Ward[236], nous pouvons prendre en compte des surfaces avec des BRDF complexes, autre que purement diffuses ou miroirs. Il devient donc possible de simuler des surfaces isotropes et anisotropes, comme des métaux ou des surfaces vernies par exemple. Dans nos scènes à regénérer, nous avons utilisé une plaque en aluminium, comme le montre l'image 4.10. Rappelons qu'il s'agit ici de surfaces avec une BRDF que nous approximons comme la somme de trois termes (voir chapitre 1) : le terme diffus, le terme spéculaire parfait et le terme *glossy*. Nous nous intéressons ici uniquement au cas *glossy*, les deux autres termes de la BRDF ayant déjà été décrits précédemment. Nous avons donc recherché différentes méthodes pour calculer la réémission de l'énergie reçue par une surface anisotrope. La plupart sont liées à de forts problèmes d'aliassage, comme la méthode de Monte Carlo [194, 226], ou ont des difficultés à traiter des BRDF très directionnelles [101, 92], tandis que d'autres nécessitent une implémentation complexe[241, 196]. Nous avons choisi de conserver la technique de lancer de rayons de Monte Carlo, mais dans une forme que nous pensons originale. Lorsqu'une surface (un patch) anisotrope doit réémettre son énergie, parce que l'algorithme de radiosité progressive l'a sélectionnée, nous générons alors plusieurs points aléatoirement sur la surface émettrice suivant la méthode proposée par [18](pages 249-253).

Pour chacun de ces points, nous ne calculons pas des rayons suréchantillonnés dans l'angle solide déterminé par les paramêtres α_x et α_y du modèle de Ward, mais nous calculons l'angle solide exact, englobant l'ensemble des rayons qu'on devrait théoriquement tirer. En effet, si nous calculons directement ces rayons, nous risquons de manquer plusieurs surfaces (éléments) de la scène, surtout si elles sont éloignées de l'objet émetteur (voir figure 4.16). C'est pourquoi, lorsque ce volume englobant a été calculé, nous déterminons quelles sont les surfaces qui pourraient être intersectées par lui en se contentant de tester, en première approximation, les surfaces dans la direction de réflexion spéculaire autour de laquelle est centré le volume. Ce test (il s'agit de simples produits scalaires) crée une liste de surfaces, pour lesquelles on va estimer la quantité d'énergie reçue depuis la surface *glossy* émettrice. Nous générons alors aléatoirement et suivant la même

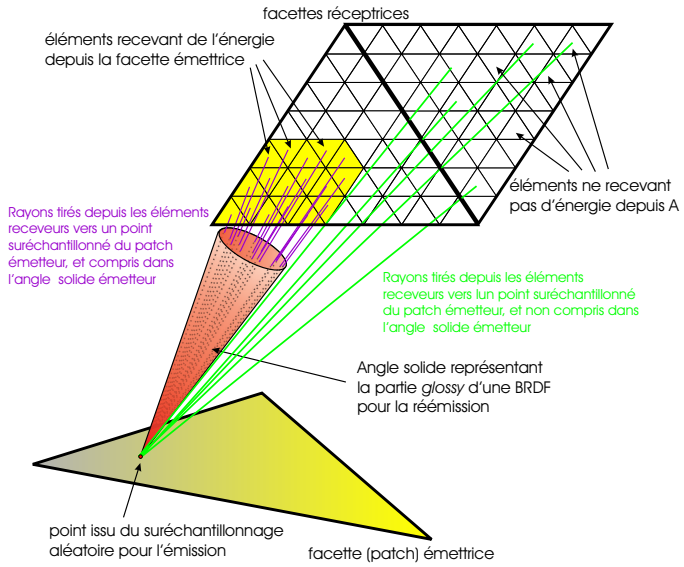

FIG. 4.17 – Exemple de facette rugueuse réémettant son énergie vers deux autres facettes, par la méthode qui consiste à tirer les rayons depuis les surfaces réceptrices, vers la surface émettrice. Les rayons qui ne sont pas compris dans l'angle solide défini depuis un point issu du suréchantillonnage sur la surface émettrice, ne reçoivent pas d'énergie depuis ce point. Cette méthode limite ainsi énormément l'aliassage produit par la technique de la figure 4.16. Nous avons volontairement ici, pour des raisons de clarté, limité le nombre de points issus du suréchantillonnage à un (au lieu de trois sur la figure 4.16).

technique que précédemment, des points sur la surface réceptrice, et nous calculons les rayons entre ces points "receveurs" et les points "émetteurs", pour déterminer grâce à l'équation 4.1 quels sont les rayons compris dans l'angle solide.

Soit $\vec{v}$ le vecteur passant par le point émetteur et le point receveur issus du suréchantillonnage aléatoire, $\vec{d_r}$ le vecteur réfléchi, $\vec{a}$ le vecteur entre le centre d'une des ellipses servant de base à l'angle solide, et un des deux points les plus proches sur cette ellipse (la norme de ce vecteur est donc le demi-petit axe) et $\vec{b}$ le vecteur entre le centre de l'ellipse et le point le plus éloigné sur cette ellipse (la norme de ce vecteur est donc le demi-grand axe). Le vecteur $\vec{v}$ est à l'intérieur de l'angle solide, si et seulement si le point résultat de l'intersection de la droite soutenue par $\vec{v}$ avec le plan passant par l'ellipse, appartient à l'ellipse. $\vec{v}$ doit donc vérifier :

$$\frac{(\vec{v}\cdot\vec{a})^2}{||\vec{a}||^4}+\frac{(\vec{v}\cdot\vec{b})^2}{||\vec{b}||^4}\leq(\vec{v}\cdot\vec{d_r})^2 \tag{4.1}$$

Si un rayon appartient à l'angle solide, nous calculons le facteur de visibilité entre la surface émettrice et la surface réceptrice, par simple tracé de rayons[22]. Il devient ainsi possible d'estimer le facteur de forme entre la surface *glossy* émettrice et la surface réceptrice, en étant nettement moins sensible au problème d'aliassage, comme le montre le schéma 4.17.

Lorsque tous les facteurs de forme entre la surface émettrice et les surfaces potentiellement réceptrices ont été calculés, nous pouvons calculer la quantité d'énergie reçue par chacune de ces surfaces (éléments) de façon classique, comme le produit de ces facteurs de forme par la radiosité de l'émetteur. Le pseudo-algorithme 15 résume notre méthode.

[22]Cette opération nécessite impérativement des optimisations de lancer de rayons, comme les BSP ou les octrees, pour éviter de devoir calculer trop d'intersections à chaque fois.

Algorithme 15 Algorithme d'émission d'énergie depuis une surface rugueuse (partie *glossy* d'une BRDF)

Procédure Emission_Energie_Surface_Rugueuse
  Utiliser table précalculée pour générer
  des points P_{e_i} aléatoirement sur le patch émetteur ;
  Calculer les frontières elliptiques des angles solides de réémission
  sous la forme de 4 vecteurs $\vec{v_i}$;
  Calculer les deux demi-axes de l'ellipse a et b (norme des vecteurs) ;
  Pour Tous les points suréchantillonnés P_{e_i} **Faire**
    Pour Tous les éléments j de la scène **Faire**
      visible = faux ;
      Pour les 4 vecteurs $\vec{v_i}$ **Faire**
        $visible = \vec{n} \cdot \vec{v_i}$;
        Si visible == vrai **Alors**
          Utiliser table statique pour générer des points P_{r_k}
          aléatoirement sur l'élément receveur ;
          Pour chacun des points P_{r_k} **Faire**
            Calculer vecteur $\vec{u_k} = P_{e_i} - P_{r_k}$;
            Calculer l'intersection entre cette droite et le plan passant
            par une ellipse appartenant à l'angle solide
            Vérifier si l'intersection est dans l'ellipse (equation 4.1) ;
            Si oui calculer le facteur de forme point-élément par Monte-Carlo ([18], pages 78-79) ;
            sinon le facteur de forme pour ce rayon est nul ;
          Fin
          sortir de la boucle des 4 vecteurs ;
        Fin si
      Fin
    Fin
  Fin
  Calculer le facteur de forme patch-élément par Monte-Carlo ([18], pages 78-79)
  $B_j = B_i \times F_{ik}$
Fin Procédure

4.2.2.2.4 Rendu des surfaces texturées

Le rendu des surfaces texturées est très important en synthèse d'images, car il augmente très souvent le réalisme de façon impressionante. Typiquement, nous avons utilisé pour nos images des livres réels, tels que "Computer Graphics" de Foley et Van Dam (référence [65]), ou "Thermal Radiation Heat Transfer" de Siegel (référence [190]). L'aspect extérieur de ces ouvrages ne peut être pris en compte autrement que par des notions texturelles, à moins d'utiliser un maillage excessivement fin avec une réflectance propre à chacun de ses éléments, ce qui pourrait produire notamment des problèmes d'aliassage. Nous avons donc retenu l'approche classique de Cohen et al.[37, 40], qui consiste à utiliser une discrétisation de la surface en éléments, pour lesquels nous calculons une réflectance, qui est la moyenne des intensités de pixels recouverts par chacun d'entre eux. Cette réflectance permet à la méthode de radiosité progressive d'approximer l'énergie reçue et réémise par la surface texturée. En phase finale de rendu réaliste (par tracé de rayons par exemple), lorsqu'un rayon traversant l'écran atteint une surface texturée, la radiosité de celle-ci est alors pondérée par la valeur réelle de l'intensité de pixel calculée directement depuis la texture (par mappage de texture[23]). On trouve aussi le détail de cette méthode dans [194], page 224.

4.2.2.2.5 Subdivisions adaptatives

Un problème très connu en radiosité est celui de la taille des éléments de surface, et plus particulièrement du comportement de la fonction de radiosité sur chacun d'entre eux. Ainsi, lorsqu'une facette dispose d'un gradient de radiosité élevé, cela signifie que le comportement de la fonction de radiosité est à priori inconnu sur sa surface. Il existe beaucoup de méthodes pour calculer le maillage optimal d'une scène[40, 194], et la plus précise reste celle du maillage des discontinuités[60, 215, 55, 232, 98, 213, 75, 54, 128, 206, 76, 127] qui consiste à déterminer analytiquement les frontières d'ombre et de pénombre sur toutes les surfaces. Cependant, nous n'avons pas ajouté cette fonctionnalité puissante dans *Phoenix*, en raison de l'implémentation complexe et du temps nécessaire à sa réalisation[24]. Nous avons retenu une approche plus simple, et plus rapide à programmer, qui consiste à subdiviser adaptativement les surfaces dont le gradient de radiosité est élevé. Cette technique, largement inspirée de l'article de Cohen et al.[37] est néanmoins relativement coûteuse en temps machine, et nécessite l'emploi d'une technique supplémentaire de remaillage[13] afin d'éviter les problèmes de *T-Sommets*[25], qui apparaissent lorsqu'une surface a été subdivisée, tandis que ses voisines sont restées à un niveau inférieur de subdivision. Une description détaillée de ce problème est donnée dans [40], page 214.
Malgré ces deux défauts majeurs, nous obtenons des temps de calcul pour une image qui restent raisonnables (voir image 4.10), pour une précision dans les ombres comparable à l'image réelle (voir chapitre 6).

4.2.3 Seconde passe : rendu de l'image finale

4.2.3.1 Problématique

Cette second passe est nettement plus simple à implémenter que la première. En effet, nous n'avons ici finalement qu'un seul objectif : produire une image à partir des informations de radiosité stockées sur chaque facette. Afin de pouvoir prendre en compte certains phénomènes comme les réflexions spéculaires, les surfaces rugueuses ou encore d'éventuels objets transparents, nous avons choisi d'utiliser le tracé de rayons comme méthode de calcul de l'image finale. Le principal problème du tracé de rayons est sa lenteur excessive pour traiter des scènes complexes. En effet, le lancer de rayons simple consiste à calculer pour chaque pixel l'intersection d'une droite passant pas ce pixel

[23] *Texture Mapping* est le terme anglais correspondant.

[24] Nous pensons néanmoins qu'une telle technique serait extrêmement bénéfique à notre logiciel, notamment comme prépasse pour obtenir un maillage optimal, mais surtout apporter une précision dans les calculs d'ombre et ainsi accélérer la génération d'une image synthétique. Ce sont précisément ces raisons qui nous poussent à ajouter cette fonctionnalité très prochainement.

[25] Nous ne connaissons pas l'équivalent français du terme *T-Vertices*, mais nous proposons la traduction directe de *T-Sommets*.

et l'observateur, avec tous les objets de la scène. En pratique, un tel algorithme met des jours à calculer une image pour une scène de 200000 facettes. Il est donc indispensable de mettre en place des optimisations pour pouvoir produire une image le plus rapidement possible (nous verrons au chapitre 6 que nous avons, en plus, besoin de calculer plusieurs fois la même image), a fortiori si plus d'un rayon est tiré à travers chaque pixel, comme c'est le cas ici (pour traiter l'aliassage).

4.2.3.2 Lancer de rayons stochastique

Le principe de base du lancer de rayons stochastique consiste à émettre plusieurs rayons depuis l'observateur à travers chaque pixel, et à intégrer ensuite les différentes couleurs récupérées pour obtenir l'image finale. Plusieurs optimisations existent dans la littérature scientifique[26], pour tenter de réduire le nombre d'intersections à évaluer pour un rayon. Certaines de ces méthodes travaillent dans l'espace 3D de la scène, comme celles des boîtes englobantes [113, 221, 250, 240, 6], celles des octrees[81, 82, 74, 242], des BSP[68, 96, 209] et celles des grilles 3D uniformes [69, 70, 204, 205, 253, 4, 114, 36, 35, 134, 249] ou non uniformes[79, 115, 106]. D'autres techniques se concentrent plutôt sur l'optimisation du nombre d'intersections dans l'espace image[240, 208, 173, 174] en utilisant une prépasse en Z-Buffer pour précalculer des tables d'index permettant de connaitre directement l'élément à intersecter pour chaque pixel[240, 208]. Dans le même ordre d'idée, [173, 174] proposent l'utilisation du A-Buffer[27] pour déterminer si plusieurs rayons doivent être relancés pour un pixel donné (si le fragment est plein, un seul rayon est nécessaire[28]). Lorsque plusieurs rayons sont lancés pour un pixel, il suffit de choisir la première intersection trouvée entre chacun de ces rayons et un des éléments stockés dans la liste de ce pixel, et d'intégrer ensuite les couleurs obtenues (par moyennage par exemple). Cette optimisation est de très loin la plus rapide de toutes, en terme de temps de calcul. Enfin, une autre optimisation consiste à réduire le temps de calcul d'une intersection, comme ici pour l'intersection d'un rayon avec un triangle[10, 248, 230].

En résumé, *Phoenix* utilise le lancer de rayons stochastique comme méthode de rendu de deuxième passe (souvent appelée "passe de l'oeil"). Les optimisations développées et employées sont la voxellisation régulière d'espace[29], alliée aux boîtes englobantes hiérarchiques, à une prépasse en A-Buffer et à l'intersection optimale d'un rayon avec un triangle[143].

4.3 Les outils d'analyse photométrique dans Phoenix

4.3.1 Présentation générale et objectifs

outils

L'objectif principal de cette thèse est la génération d'une image de synthèse, depuis une image réelle prise avec une caméra. Il s'avère, à cause de la méthode que nous avons choisie, qu'il est nécessaire de calculer plusieurs fois la même image pour réduire l'erreur entre l'image synthétique et l'image naturelle. Au fil des recherches et du développement des algorithmes nécessaires à cette regénération, nous avons constaté que l'analyse manuelle de plusieurs centaines de surfaces (sans compter le fait que lorsqu'une surface pose des problèmes il faut pouvoir retrouver son numéro dans le fichier de données) , avec des outils classiques de différence d'images, de calcul de moyenne, de

[26] Un excellent tour d'horizon des méthodes antérieures à 1990 est disponible dans le livre [80], pages 201-262, et [59] propose également un état de l'art superficiel de toutes ces méthodes mais jusqu'en 1999.

[27] Nous utilisons également notre A-Buffer comme passe de précalcul pour le tracé de rayons. Cependant, nous ne lançons des rayons dans un pixel que s'il existe un élément au moins qui ne soit pas diffus. Si tous les éléments sont diffus, nous avons directement la couleur du pixel en appliquant la méthode du A-Buffer décrite précédemment aux fragments stockés pour ce pixel.

[28] Dans ce cas précis, *Phoenix* ne lance aucun rayon puisque le A-Buffer permet de calculer immédiatement la couleur du pixel.

[29] Nous avons codé les octrees et les BSP. En pratique, pour nos scènes, ces méthodes s'avèrent moins rapides que la voxellisation régulière, qui permet l'utilisation d'un algorithme de Bresenham 3D extrêmenemt efficace. [80] page 223, ainsi que [70] précisent d'ailleurs que les BSP ou les octrees ne sont pas systématiquement les plus rapides.

maximum, etc., est particulièrement fastidieuse et répétitive. Il nous a donc paru indispensable de réaliser une interface graphique, facile à utiliser et qui nous fournisse en temps réel sans interrompre les calculs, l'évolution des réflectances surface par surface, groupe d'objets par groupe d'objets, leurs histogrammes, etc. Nous en avons également profité pour développer une interface où l'on puisse visualiser le résultat du rendu réaliste en temps réel (voir figure 4.18). Cette interface graphique est fondée sur le principe du *multi-threading* qui permet la visualisation des étapes du rendu réaliste par radiosité en temps réel, tout en se déplaçant dans la scène interactivement sans interrompre les calculs. Cependant, si certaines analyses plus poussées doivent être effectuées, il est aussi possible de mettre en pause le calcul courant et de le reprendre ensuite là où il a été laissé.

4.3.2 L'interface graphique

4.3.2.1 Comparer les images

Un des points fondamentaux de *Phoenix* est qu'il utilise la différence entre l'image réelle et l'image synthétique pour corriger les réflectances et faire des hypothèses sur leurs propriétés. Nous avons donc commencé par créer un onglet appelé *Analysis/Synthesis* (voir figure 4.18) destiné à nous permettre de visualiser rapidement les erreurs sur les surfaces et en déduire si notre algorithme réalise les bons choix quant aux propriétés de réflectance des surfaces. La figure 4.19 montre l'outil de différence d'image effectué ici sur une scène de bureau. On remarque que plusieurs techniques de comparaison sont possibles (soustraction, addition, etc.). Par ailleurs, l'utilisateur a le choix entre effectuer un rendu inverse de manière automatique en fonction d'un nombre d'itérations maximal, ou l'effectuer étape par étape, pour analyser les calculs réalisés par *Phoenix*. Par ailleurs, les images sont systématiquement rendues par lancer de rayons avec 10 rayons par pixel, et stockées sur disque. Si l'utilisateur souhaite augmenter la résolution, il peut à tout moment modifier le nombre de rayons par pixel, et recalculer une nouvelle image.

4.3.2.2 Analyser les résulats

L'onglet *Advanced Analysis* permet d'analyser le comportement des réflectances des surfaces au fur et à mesure des itérations de rendu inverse. On peut ainsi visualiser la structure de la scène, avec ses groupes d'objets et ses objets, et pour chacun d'entre eux visualiser par de simples cliques de souris, les histogrammes, les valeurs statistiques (moyenne, maximum, minimum, ...), l'évolution des réflectances. La figure 4.20 montre cet onglet en détail.

4.3.3 Le format de fichier

Le format de fichier lu par *Phoenix* est extrêmement simple, puisqu'il s'agit du format *Inventor* développé par *Silicon Graphics*. Nous n'avons donc besoin que de la description géométrique des objets, avec leur appartenance à un groupe (cette notion[30] est décrite plus précisément au chapitre 6). Un exemple de fichier est fourni en annexe C.

4.3.4 Quelques accélérations spécifiques au rendu inverse

Lors du rendu inverse d'une image, nous effectuons le calcul d'une image sous certaines conditions qui restent inchangées : la structure de la scène elle-même n'est pas modifiée pendant l'analyse photométrique (aucun objet ne bouge), ce qui signifie que les facteurs de forme sont constants d'une image à l'autre. Pour cette raison *Phoenix* conserve la quantité maximale stockable de facteurs de forme (mais pas les facteurs de forme calculés pour le spéculaire) en mémoire, afin d'éviter leur réestimation à la prochaine itération du rendu inverse. Si le facteur de forme à évaluer n'est pas disponible en mémoire vive lors d'une itération, nous le recalculons. Cette optimisation est très

[30] Déterminer le groupe d'un objet est très simple sous Inventor, puisqu'il suffit d'ajouter le mot *Group { }* dans le fichier et de mettre à l'intérieur des accolades, les objets (*Separator* en Inventor) appartenant au même groupe.

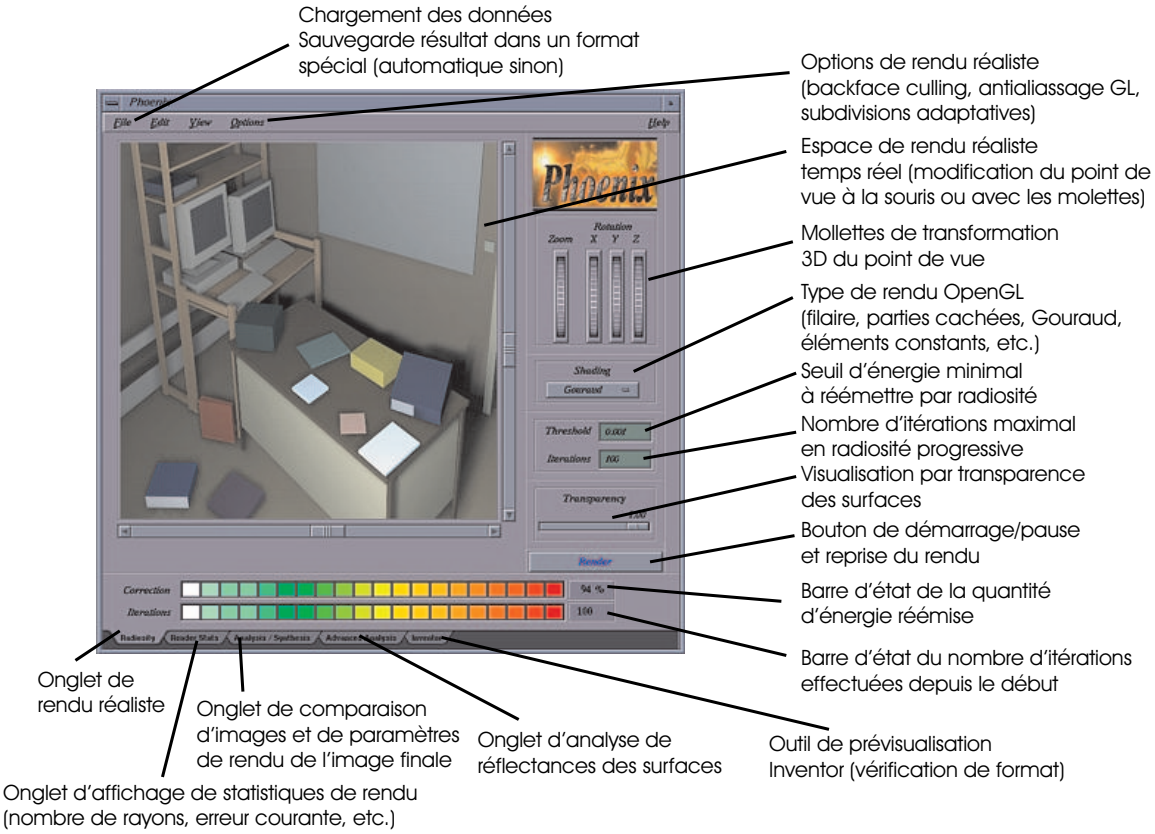

FIG. 4.18 – L'onglet principal de *Phoenix* est activé. Il permet de voir la scène évoluer en temps réel suivant les étapes de radiosité progressive, et la correction des réflectances. Plusieurs types de rendu sont disponibles dans les menus afin de visualiser éventuellement les subdivisions adaptatives, de gérer la transparence, de bouger le point de vue en temps réel sans gêner les calculs (*multi-threading*). Les onglets disponibles en bas de la fenêtre permettent de sélectionner différents outils pour comparer et/ou analyser les surfaces.

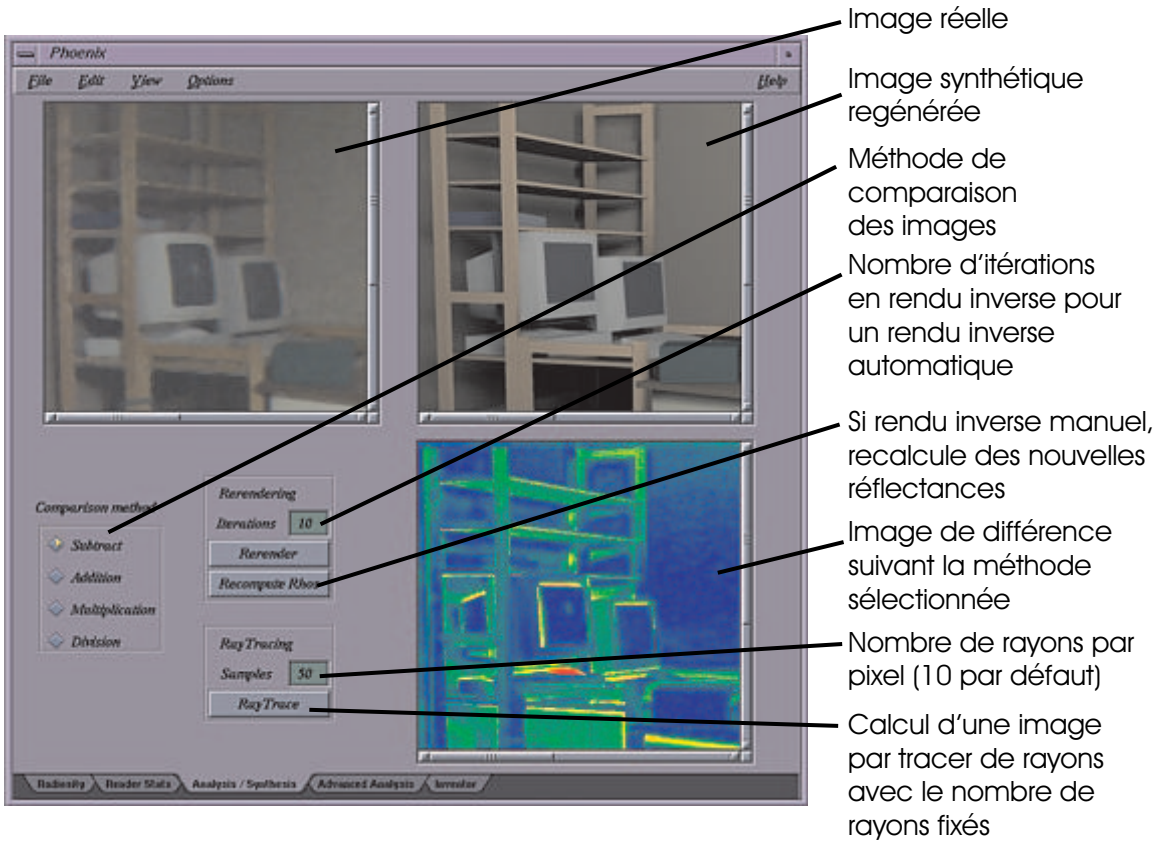

FIG. 4.19 – L'onglet de comparaison d'images de *Phoenix* est activé. Il permet de comparer l'image réelle avec l'image synthétique suivant plusieurs méthodes. Cet onglet offre également la possibilité de calculer une image en rendu inverse de manière automatique, ou de manière manuelle (pour une analyse plus poussée des calculs effectués par *Phoenix* sur les réflectances), ainsi que de choisir le nombre de rayons à lancer par pixel, pour calculer une nouvelle image (10 par défaut).

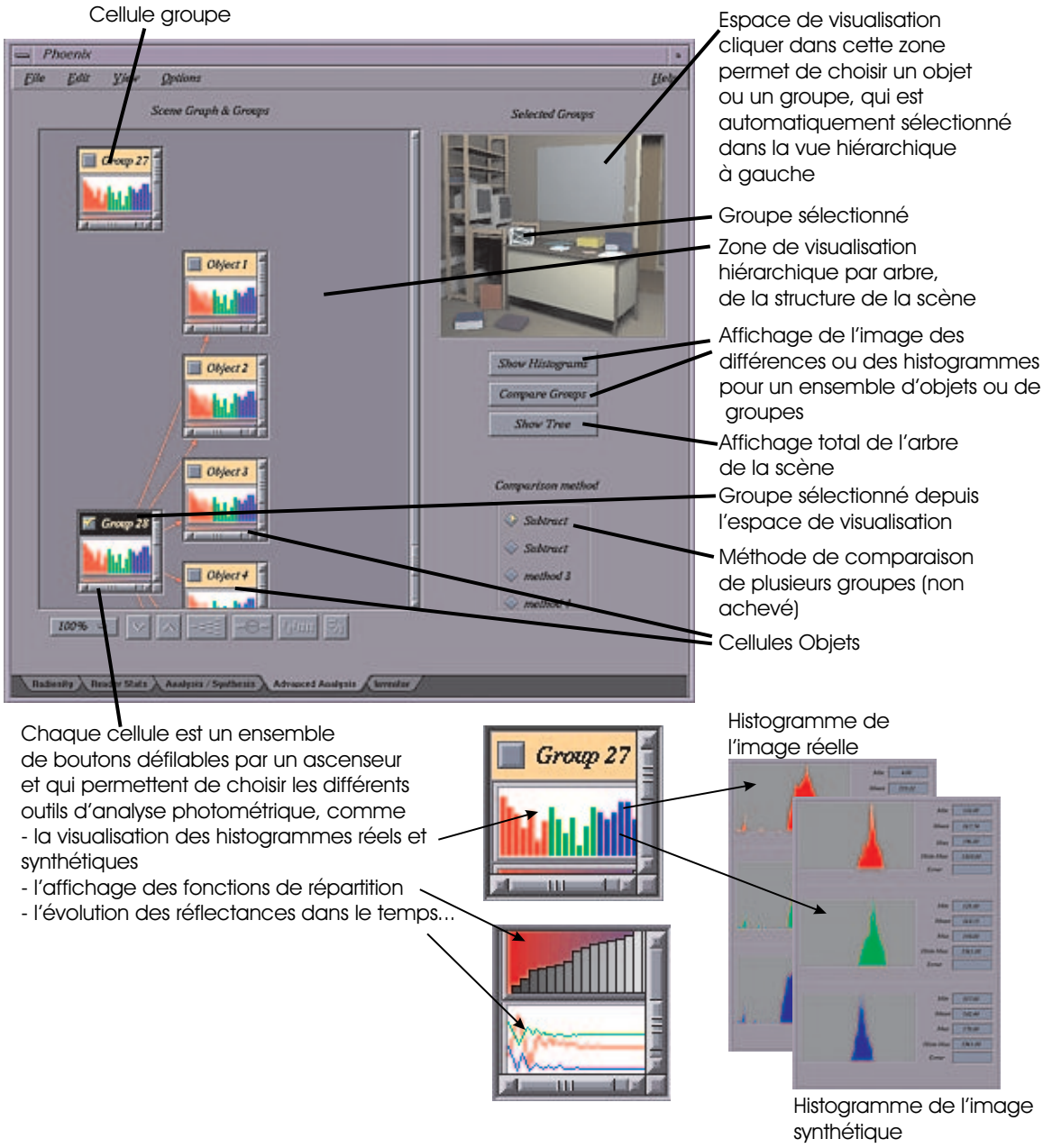

FIG. 4.20 – L'onglet d'analyse des réflectances de *Phoenix* est activé. La structure de la scène est représentable sous la forme d'un arbre où les nœuds pères sont les groupes et les fils les objets. L'utilisateur peut choisir en cliquant sur l'image (en haut à droite), un groupe (avec le bouton droit de la souris) ou un objet (avec le bouton gauche) : la cellule correspondante de l'arbre s'active automatiquement et se centre dans la zone d'affichage de l'arbre. On peut alors cliquer sur la cellule de l'arbre et faire défiler l'outil qui nous intéresse : si on sélectionnne par exemple l'outil *histogramme* (comme ici), *Phoenix* affiche les histogrammes de ce groupe pour l'image réelle et l'image synthétique.

pratique lorsque l'on souhaite se livrer à des calculs de réflectances sur des scènes de taille raisonnable, et pour lesquelles nous n'avons pas nécessairement besoin d'émettre la totalité de l'énergie dans la scène : il devient alors possible de stocker la quasi-totalité des facteurs de forme[31].
Un second paramètre qui ne change pas d'une image à l'autre est la position de l'observateur. Ainsi, lorsqu'une image a été calculée par lancer de rayons, nous connaisssons les intersections réalisées et pouvons en stocker certaines : en effet nous ne stockons que les intersections primaires qui restent inchangées dans tous les cas, mais pas les intersections secondaires issues de réflexions spéculaires ou de réflexions *glossy*.
Ces deux optimisations nous assurent un gain de temps exceptionnel pour le calcul d'une nouvelle image, puisque lors d'une nouvelle itération de rendu inverse (à partir de la seconde) ce calcul devient quasi-instantané si aucune surface *glossy* n'est présente (en raison du nombre de rayons réémis depuis chaque intersection d'une surface rugueuse).

4.4 Comparaisons avec des logiciels existants

Nous avons pu comparer notre logiciel avec un seul (*Lightscape*) à l'heure actuelle, pour deux raisons. La première est que *Lightscape* est le seul à disposer d'un format de fichier proche du nôtre. En effet, bien qu'il possède son propre format (fichiers d'extension "*.lp*"), *Lightscape* accepte d'importer tout un ensemble d'autres fomats (*Autocad* et *3DS* par exemple) pour lesquels il existe un convertisseur[32] qui permet de transformer nos fichiers de données *Inventor* (*VRML* en fait) vers ces formats (et vice-versa). Si nous ne disposons pas de telles outils pour réaliser cette conversion de nos données vers un autre format (comme c'est le cas pour *Radiance* par exemple), l'investissement nécessaire à cette conversion de formats devient alors beaucoup trop important, pour qu'elle puisse être réalisée en un temps raisonnable.

La seconde raison est que *Lightscape* est probablement le plus complet et le meilleur des logiciels de calcul d'illumination globale à l'heure actuelle. Il est donc particulièrement intéressant de pouvoir se comparer à l'élite des logiciels de rendu. Il est important cependant de préciser, que nous ne cherchons pas du tout ici à comparer *Phoenix* à *Lightscape* dans sa globalité. En effet, *Ligtscape* possède un nombre de fonctionnalités incomparables aux nôtres et une approche extrêmement physique du rendu réaliste (on peut stipuler par exemple des sources de lumière de façon très physique avec des notions de directionnalité, de puissance en candela, et même des sources fluorescentes). Globalement, nous pouvons dire que *Lightscape* reste de toute façon très nettement -et incomparablement- supérieur à *Phoenix* dans l'étendue de ses possibilités de rendu réaliste (y compris l'interface graphique qui permet de modifier les objets interactivement de manière géométrique comme photométrique par exemple).
Cependant, ce qui nous préoccupe ici, c'est la vitesse à laquelle *Lightscape* est capable de rendre une scène similaire à la nôtre. Nous avons donc employé une scène relativement simple, comportant environ 15000 polygones et 17000 sommets. Cette scène intitulée "*gallery*" est une scène téléchargeable depuis le site *Lightscape*(www.lightscape.com) et possède 8 textures de peintures, chacune de résolution 256×256 environ.

Les temps de calcul obtenus sur un PC Pentium III 600Mhz en passe de radiosité simple puis en passe de lancer de rayons distribués, sont récapitulés par les tableaux 4.25 et 4.26.

On constate que *Lightscape* est beaucoup plus rapide (4 fois plus) que *Phoenix* si on garde une résolution de 4096×4096 pour les hémicubes. Néanmoins, la qualité des images obtenues est à notre avis bien inférieure à celle obtenue avec *Phoenix*, comme le montrent les images (voir images 4.24 et 4.21).
De plus, le temps que met *Phoenix* à calculer cette image diminue de façon considérable si on réduit la résolution des hémicubes à 1024×1024, puisque *Phoenix* devient alors environ 1.5 fois plus rapide que *Lightscape*. Cette résolution s'avère d'ailleurs suffisante pour obtenir une image

[31] Nous avions la chance de disposer d'une station avec 1200Mo de RAM, ce qui nous permet de stocker facilement 200 itérations de radiosité, pour une scène de 200000 facettes.

[32] Typiquement, nous avons employé *CrossRoads* de Keith Rule, téléchargeable gratuitement depuis *http ://home.europa.com/∼keithr/Crossroads/index.html*.

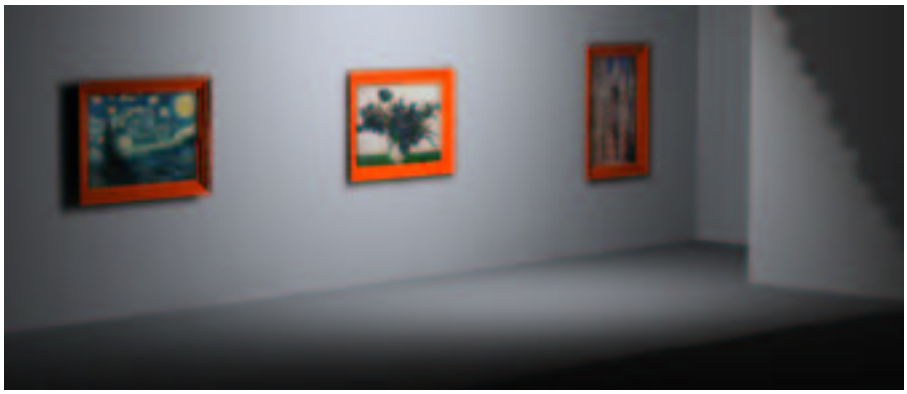

FIG. 4.21 – Image calculée par *Lightscape* en 5 minutes. On remarque de fortes discontinuités dans les ombres, issues d'un problème de subdivisions insuffisantes.

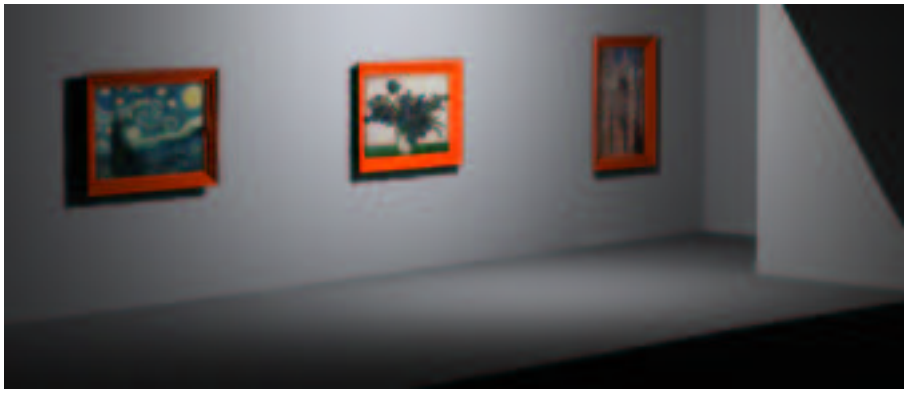

FIG. 4.22 – Image calculée par *Lightscape* en 52 minutes environ, en utilisant un maillage adaptif fin. On remarque que les fortes discontinuités de l'image 4.21 ont quasiment disparu.

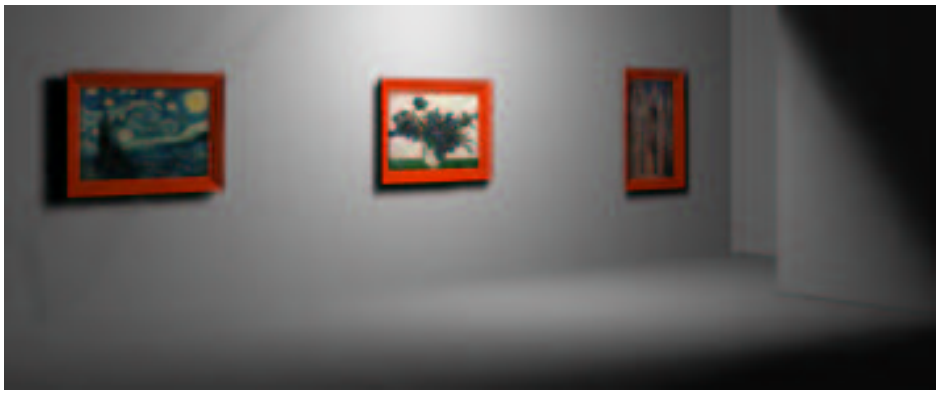

FIG. 4.23 – Image calculée par *Phoenix* en 3min17s environ, avec des hémicubes de résolution 1024×1024. Quelques problèmes d'aliassage apparaissent sur le mur gauche.

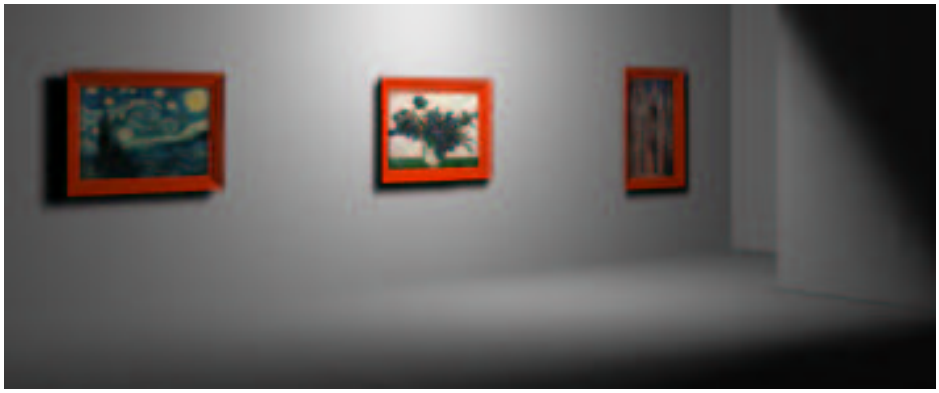

FIG. 4.24 – Image calculée par *Phoenix* en 21 minutes environ, avec des hémicubes de résolution 4096×4096. On constate que la position de l'observateur et la photométrie générale ne sont pas tout à fait les mêmes que pour les images produites par *Lightscape*. Ceci est dû au fait que, d'une part, nous n'employons pas du tout la même fonction de conversion des luminances en intensités de pixel, et, d'autre part, notre façon de décrire une caméra est radicalement différente de *Lightscape* (nous avons donc été obligé de retrouver manuellement un point de vue équivalent).

	temps de calcul	mémoire utilisée
Lightscape 3.2 (sans subdivisions)	5min	30 Mo
Lightscape 3.2 (avec subdivisions)	52min12	45 Mo
Phoenix(avec hémicubes 1024×1024)	3min17s	61 Mo
Phoenix(avec hémicubes 4096×4096)	20min48s	170 Mo

FIG. 4.25 – Comparaison de *Phoenix* avec *Lightscape* en calcul de radiosité pur sans phase de lancer de rayons, et pour 300 itérations de radiosité, avec un maillage fixe de 18000 éléments.

	Lancer de rayons
Lightscape 3.2	1min44s
Phoenix	11s

FIG. 4.26 – Comparaison de *Phoenix* avec *Lightscape* uniquement en phase finale de lancer de rayons sur une scène de 15000 polygones, avec 10 rayons par pixel, et une résolution d'image de 800×500.

de qualité acceptable, mais des artéfacts d'aliassage apparaissent dans l'image (voir image 4.23). Cependant, ces problèmes restent nettement moins génants que ceux issus du rendu réalisé par *Lightscape* en 5 minutes : les artéfacts en "escalier" sur le mur droit de l'image 4.21 nuisent fortement au réalisme de la scène.
Pour résoudre ce problème avec *Lightscape*, il faut choisir un critère de subdvision plus fin pour l'énergie stockée sur deux éléments voisins. Si nous réalisons cette opération afin d'obtenir des ombres optimales (voir image 4.22), le temps que met *Lightscape* pour calculer une image passe à 52 minutes (pour une subdivision en 55500 éléments, et 57800 sommets), et occupe une mémoire totale de 45 Mo, mais produit une image de haute qualité.

Par ailleurs, *Lightscape* utilise 30 Mo[33] pour calculer la scène tandis que *Phoenix* emploie au mieux sur cette scène 61 Mo de mémoire. Ceci est probablement dû à nos structures de données pas forcément optimales, mais surtout au A-Buffer que l'on sait très gourmand en mémoire vive.

Les résultats pour le lancer de rayons ont été obtenus en forçant volontairement *Phoenix* à utiliser le lancer de rayons pour calculer cette scène. En temps normal, celle-ci ne comportant que des surfaces diffuses, *Phoenix* l'aurait directement estimée par A-Buffer, ce qui aurait donné un temps de calcul inférieur à 3 secondes. Ici, notre logiciel met 11 secondes en raison de sa prépasse en A-Buffer, qui lui permet d'effectuer très peu d'intersections pour chaque pixel.

En conclusion, nous pouvons dire que *Phoenix* est un logiciel de rendu photoréaliste rapide, voire plus rapide que des logiciels haut-de-gamme, sous certaines conditions (diminution de la résolution de l'hémicube). Cependant, la structure de *Phoenix* a été conçue pour le rendu à base d'images réelles, ce qui le rend nettement plus lourd à gérer d'un point de vue mémoire vive.

4.5 Conclusion et extensions futures

Nous avons donc mis au point un logiciel assez conséquent [34] de rendu classique, mais performant, et surtout adapté au rendu inverse. *Phoenix* est donc capable de calculer des images photoréalistes (voir figure 4.10 et le chapitre 6), et en un temps raisonnablement court. Bien qu'il ne souffre d'aucune comparaison, en terme de vitesse et de qualité de rendu, avec d'autres logiciels de rendu réaliste comme *Lightscape* ou *Radiance* par exemple, il possède un panel moins large de possibilités. En effet, *Phoenix* ne sait pas encore prendre en compte les objets transparents, les milieux participatifs, et son maillage adaptatif reste relativement rudimentaire. C'est pourquoi il est envisagé de lui ajouter une prépasse destinée à la création d'un maillage de discontinuités pour améliorer la qualité des ombres notamment, ainsi que d'autres techniques lui permettant de prendre en compte les caustiques et autres phénomènes complexes. Par ailleurs, il est prévu d'implémenter la radiosité hiérarchique car les temps fournis par *Lightscape* restent très impressionnants. Il serait sans doute intéressant de comparer *Phoenix* à lui-même, avec une implémentation à base de radiosité progressive d'une part et, d'autre part, avec une approche en radiosité hiérarchique. Enfin, la possibilité d'intégrer des objets en temps réel dans l'image regénérée[203, 203, 57] et l'emploi d'un complexe de visibilité[59] est également à l'étude bien que cela ne constitue pas une priorité

[33] En fait, lors du calcul de l'image, *Lightscape* affiche une occupation totale de la mémoire de 3.98 Mo. Cependant, lorsqu'on emploie un outil d'analyse de mémoire et de tâches, on se rend compte que la mémoire totale occupée est en réalité de 30 Mo. Nous n'avons pas l'explication de cette différence, et rien dans la documentation de *Lightscape* n'est précisée à ce sujet.

[34] *Phoenix* comptabilise en effet environ 70000 lignes de codes, incluant la partie rendu pur et le rendu inverse.

immédiate.
Par ailleurs, l'interface graphique de notre logiciel et son format de fichier très simple lui assure une utilisabilité aisée par des personnes non expertes, désireuses de calculer rapidement une image de synthèse photoréaliste par rendu réaliste direct, ou depuis une image réelle.

Enfin, nous verrons au chapitre 6 que *Phoenix*, dans sa forme actuelle, s'avère un outil très puissant pour la regénération d'images de synthèse depuis des images réelles, et en utilisant finalement très peu de paramètres (le modèle 3D de la scène, la position de la caméra, et une seule image réelle capturée avec une caméra classique sans contrainte particulière sont nécessaires au recouvrement des réflectances de toutes les surfaces de la scène).

Deuxième partie

Rendu à base d'images réelles

Chapitre 5

Etat de l'art

Le premier précepte était de ne recevoir jamais une chose pour vraie que je ne la connusse être telle.
René Descartes

5.1 Introduction

Dans cet état de l'art, la part des travaux réalisés par notre laboratoire sera présentée dans le chapitre 6. Nous proposons donc ici, un tour d'horizon de toutes les techniques existantes pour retrouver les paramètres de réflectance de surfaces réelles. Il est important de bien distinguer chacune de ces méthodes, car aucune ne réalise vraiment la même image sous des conditions similaires aux nôtres.
En effet, on constate que si certains utilisent des appareils plus ou moins sophistiqués pour retrouver les réflectances (paragraphe 5.2), d'autres choisissent plutôt de tenter de les extraire d'images réelles, soit avec plusieurs (paragraphe 5.3), soit avec une seule (paragraphe 5.4). Ces deux dernières catégories sont encore séparables entre celles qui prennent en compte les inter-réflexions (paragraphes 5.3.2 et 5.4.2) et celles qui n'en tiennent pas compte et qui sont généralement limitées au recouvrement de la réflectance d'un objet unique (paragraphes 5.3.1 et 5.4.1).
Enfin, nous citons quelques-unes (il en existe une pléthore) des autres méthodes qui existent pour réaliser du rendu à base d'images réelles[1], mais qui ne procèdent pas à la recherche des paramètres de réflectance des surfaces (paragraphe 5.5).
Nous verrons dans la discussion, pourquoi nous pouvons considérer que notre approche est plus puissante que toutes celles décrites ici, bien qu'étant moins physique que certaines.

5.2 Mesurer directement les réflectances

L'idée de mesurer les réflectances avec un appareil spécifique mais peu coûteux et de pouvoir exploiter directement les mesures dans une équation de réflexion de la lumière, représente les motivations principales de Ward dans son article [236]. Il propose en effet un appareil capable de mesurer des paramètres de BRDF qu'il peut ensuite directement injecter dans son modèle d'illumination (voir chapitre 2, paragraphe 2.4.8).
Son outil de mesure de BRDF est simple, pour peu que l'on puisse se procurer l'ensemble des parties indispensables à sa constitution. Tout d'abord, il est nécessaire de placer son appareil dans une salle noire pour simuler un *corps noir*[2], et éviter ainsi que des éléments extérieurs viennent perturber les mesures (comme des inter-réflexions de la lumière). Ensuite l'objet à mesurer doit

[1] Encore une fois, nous utiliserons de préférence les termes anglais *Image-Based Rendering* ou *Rerendering* ou *Inverse Rendering* pour caractériser cette technique et ne pas nous couper de la littérature infographiste.
[2] Type de surfaces n'ayant que des propriétés d'absorption de la lumière et aucune réémission.

être posé au centre d'une surface autour de laquelle un bras mécanique peut se déplacer pour changer les directions incidentes de la source de lumière, qui éclaire les surfaces. Face à l'objet et à la caméra placée à côté et équipée d'une lentille *fish-eye*[3], est placé un hémisphère semi-argenté chargé de récupérer la lumière réfléchie ou réfractée par l'objet dont on cherche les propriétés. Les mesures s'effectuent alors sur l'image capturée et plusieurs équations permettent d'en déduire les angles nécessaires, ainsi que les valeurs de réflectance bidirectionnelle (une comparaison est réalisée entre un échantillon parfaitement diffus et la surface à mesurer). Afin de retrouver tous les paramètres $\alpha_x, \alpha_y, \vec{x}, \rho_d, \rho_s$ (décrits au chapitre 2, paragraphe 2.4.8), on procède alors à la résolution d'un système d'équations linéaires.
Par ailleurs, Ward valide son appareil et son modèle d'illumination en réalisant des images photo-réalistes, issues des mesures qu'il a réalisées sur des surfaces réelles. L'avantage de cette méthode est que, d'une part, son appareillage semble coûter peu cher par rapport à un goniophotomètre[4], et que, d'autre part, il fonctionne pour des objets variés, isotropes ou non (dont l'éclat spéculaire n'est pas trop fin cependant, comme sur des surfaces très polies), et même transparents. Les mesures réalisées présentent également l'énorme avantage de pouvoir être exploitées directement dans un modèle de BRDF. L'inconvénient est qu'il ne traite que les objets un par un, et donc que les mesures de plusieurs dizaines d'objets peuvent s'avérer longues et fastidieuses.

Une simplification du dispositif de Ward fut apportée par Karner, Mayer et Gervautz [110] qui proposèrent un appareillage sans récepteur particulier, tandis que la méthode précédente nécessitait tout de même une demi-sphère semi-argentée peu courante.
Le protocole d'expérimentation est donc le suivant : l'objet à mesurer est placé à côté d'une surface étalon standard parfaitement diffuse. La source de lumière doit être ponctuelle (ou simulée comme telle, si on la place suffisamment loin de l'objet à mesurer) pour éviter les angles multiples d'incidence[5], et doit disposer d'une distribution radiale symétrique d'énergie, pour que toutes ses valeurs d'exitance soient connues. Sa position est mesurée manuellement. Plusieurs images de l'objet sont alors capturées par une caméra CCD, de position connue et placée de sorte à toujours voir la ligne de séparation des deux surfaces.
Karner et al. utilisent ensuite la même technique que Ward pour mesurer la réflectance bidirectionnelle de la surface en comparant la surface mesurée à la surface étalon : la différence réside dans le fait que Ward soustrayait des mesures réalisées, la luminance produite par la source de lumière seule sur la sphère semi-argentée, tandis que Karner et al. soustraient des valeurs de luminances mesurées avec la source de lumière sur l'étalon et la surface, les valeurs mesurées dans les mêmes conditions aux mêmes endroits mais sans la source de lumière (pas besoin de corps noir donc).
En exploitant aussi le modèle d'illumination de Ward, Karner et al. recherchent les mêmes paramètres[6] $\alpha_x, \alpha_y, \rho_d, \rho_s$ en effectuant alors une minimisation aux moindres carrés, entre la BRDF regénérée et celle mesurée dans les images. Par ailleurs, les auteurs précisent que, souvent, une seule image est nécessaire pour retrouver la BRDF d'une surface, mais que plusieurs images améliorent considérablement les résultats.
Les avantages de la méthode proposée dans cet article sont sa simplicité et sa validité scientifique, puisque les auteurs ont comparé les résultats obtenus par leur méthode, avec ceux réalisés par un goniophotomètre. Néanmoins, les restrictions d'utilisation restent les mêmes que pour Ward (à l'exception du corps noir) et d'autres viennent même s'y ajouter, puisqu'on ne peut traiter que des surfaces planes et opaques (pas de calcul de transmittance possible).

[3] Le terme français le plus proche que nous ayons trouvé, est *caméra à très courte focale*, mais nous pensons que le terme anglais reste plus parlant.

[4] Le goniophotomètre ou goniomètre (*gonioreflectometer* en anglais) est un appareil complexe capable de mesurer directement les BRDF des surfaces et l'indicatrice des sources de lumière (pour plus de précision, se reporter à l'ouvrage de Desvignes [52], page 250). Cet appareil est extrêmement cher, et les quelques laboratoires en disposant effectuent des mesures sur des matériaux, à la demande. Ces mesures sont bien évidemment payantes et également très coûteuses.

[5] Ward utilise une sorte de filtre pour créer un faisceau rectiligne.

[6] Curieusement, dans l'article, rien n'est dit au sujet du vecteur $\vec{x}$ indiquant la direction des stries sur la surface. On suppose donc qu'elle est connue.

Contrairement aux deux travaux précédents, le souci de Baribeau, Rioux et Godin [12] n'est pas le prix du matériel, puisqu'ils proposent l'utilisation d'un laser polychromatique pour récupérer les réflectances des surfaces. Ces réflectances sont alors modélisées par le modèle de Torrance et Sparrow [220, 219], et sont paramétrées sous la forme de trois valeurs : la réflectance diffuse, la réflectance spéculaire et un coefficient de rugosité. L'équation de Torrance nécessite le calcul du terme de Fresnel, qui est ici approximé par une constante, fonction de la longueur d'onde. Ensuite, Baribeau et al. appliquent un algorithme de minimisation pour trouver les meilleures valeurs possibles pour ces trois paramètres.
L'avantage de leur méthode est qu'elle permet de calculer les réflectances de plusieurs objets à la fois : l'exemple donné dans l'article est d'ailleurs une scène avec un ensemble de crayons posés sur une table. De plus, le laser permet une reconstruction 3D de la scène en même temps que l'analyse des réflectances. Cependant, cette technique a plusieurs points faibles. Tout d'abord, elle ne fonctionne que pour les scènes non texturées et pas trop complexes, mais surtout, elle ne prend pas en compte les phénomènes d'inter-réflexions pouvant survenir. Il n'est donc pas évident qu'elle puisse produire quelque chose d'utilisable dans le cas d'un objet rouge posé sur une feuille blanche par exemple ou dans le cas d'une surface rugueuse réfléchissante. Dans tous les cas, rien ne permet de le dire à la lecture de l'article.

Dana et al. [43] proposent une technique originale pour mesurer les réflectances sur des échantillons de surfaces réelles. Cette méthode est basée sur un robot qui tient et oriente l'échantillon à mesurer, un photomètre, une lampe halogène avec un filtre de Fresnel (pour produire une faisceau de lumière parallèle), et une caméra tri-CCD couleurs reliée à un système d'acquisition d'images. L'ensemble de ce dispositif leur permet de mesurer et de construire les BRDF et BTF[7] de tous les objets. Ils s'en sont d'ailleurs servis, pour constituer une grande base de données possédant les BRDF de plusieurs milliers de surfaces, et disponible sur Internet[8].

5.3 Estimer les réflectances à partir de plusieurs images

5.3.1 Méthodes sans gestion des inter-réflexions

Dans cette partie, nous décrivons les techniques existantes pour calculer les réflectances d'un objet depuis plusieurs images. En fait, il s'agit ici d'extraire les propriétés de réflexion des surfaces depuis une ensemble d'images de ces surfaces, sous plusieurs angles différents et éventuellement sous plusieurs conditions d'éclairement : les valeurs de la BRDF ne sont plus directement extraites de l'image ou d'un dispositif particulier, comme cela était le cas dans le paragraphe précédent.

Kay et Caelli [112] proposent une méthode pour calculer les réflectances des surfaces, en utilisant le modèle de Torrance-Sparrow [220, 219], une carte de profondeur et quatre images obtenues avec des sources ponctuelles. Dans un premier temps, la surface est analysée et les pixels de sa projection sur l'écran sont classés en trois catégories : les pixels non spéculaires, les pixels spéculaires et les pixels inconnus[9]. Dans un second temps, chacune des zones ainsi détectée[10] sur la surface et formée par un ensemble de pixels, est traitée dans un processus automatique dont la première étape est la simplification du modèle de Torrance-Sparrow : le terme de Fresnel est considéré comme une constante, ainsi que le facteur d'atténuation géométrique car l'angle entre la lumière incidente et la direction d'observation ne dépasse pas 30 degrés[11]. Ceci réduit donc l'équation de Torrance à trois termes : k_d, k_s et c les coefficients respectifs de diffusion, de spécularité et de rugosité.

[7] Les auteurs font la distinction entre une réflectance uniforme et une réflectance non uniforme, caractérisée par une texture, et appelée BTF pour Fonction Bidirectionnelle de Texture (*Bidirectional Texture Fonction*).
[8] L'adresse web de ce site est : *http ://www.cs.columbia.edu/CAVE/curet/*.
[9] Dans leur article, Kay et Caelli parlent de *rank-deficient regions*.
[10] Une technique de détection des composantes diffuses et spéculaires, est également disponible dans [11].
[11] La justification de cette approximation se trouve dans [211].

En inversant le processus de formation d'images par stéréo-photométrie [210], l'équation de Torrance-Sparrow est réductible à trois problèmes de minimisation distincts pour chacune des zones : une méthode linéaire aux moindres carrés est appliquée pour retrouver le coefficient diffus k_d des zones non spéculaires, une méthodes non-linéaire des moindres carrés séparables avec régularisation est utilisée pour k_d, k_s et c dans les zones spéculaires, et une technique d'interpolation basée sur les résultats précédents est employée pour calculer k_s et c dans les zones non spéculaires, ainsi que k_d, k_s et c dans les zones inconnues.
Si l'avantage majeur que présente cette méthode est de pouvoir retrouver les réflectances des surfaces texturées ou uniformes tout en s'appuyant sur un modèle physique de BRDF, les inconvénients sont nombreux. Tout d'abord, l'algorithme ne sait traiter que des objets convexes et géométriquement simples. Il est également limité par des hypothèses géométriques fortes sur le terme de Fresnel et sur la fonction d'atténuation géométrique. La technique ne permet pas non plus de prendre en compte plus d'un objet à la fois et elle néglige, de ce fait, les inter-réflexions qui pourraient exister : il devient donc difficile de pouvoir traiter des surfaces miroirs par exemple. Enfin, rien ne permet d'établir clairement si le modèle est facilement extensible aux surfaces anisotropes.

L'approche choisie par Lu et Little [133] est orientée sur la recherche de réflectances de surfaces sans modèle de réflexion. Ainsi, tandis que Kay et Caelli emploient Torrance-Sparrow, Lu et Little ont choisi d'exploiter directement les intensités de pixel de quelques points singuliers de la surface. Une séquence d'images noir et blanc est donc capturée (19 images au total), en utilisant un dispositif spécifique qui tourne autour de l'objet que l'on cherche à analyser (rotation totale de 90 degrés). Par ailleurs, la source de lumière se trouve dans l'axe optique (l'observateur est placé à l'infini pour réduire le problème à une projection orthographique), ce qui permet de rendre la BRDF uniquement dépendante de la direction incidente.
Pour chaque angle incident(et donc réfléchi), une fonction d'intensités de pixel est construite avec la normale dans l'axe de l'observateur pour extraire des points caractéristiques, où l'intensité de pixel est maximale (en effet, si la source de lumière, la direction d'observation et la normale à la surface sont alignées, l'intensité réémise ne peut qu'être maximale en ces points). Cette fonction est utilisée ensuite pour caractériser la réflectance de la surface.
On voit immédiatement que la simplicité de cette technique est son point fort. Malheureusement, l'ensemble des restrictions qu'elle pose, la limite à des utilisations simples. Tout d'abord, on ne peut toujours traiter qu'un seul objet à la fois et les images sont en noir et blanc, ce qui simplifie grandement le problème. Par ailleurs, il faut utiliser un dispositif particulier et les conditions expérimentales sont très strictes. Pour reconstruire la fonction de réflectance, les objets doivent aussi avoir obligatoirement une réflectance uniforme (ce qui exclue également les miroirs et autres surfaces très réfléchissantes), on ne peut donc pas traiter d'objets texturés, et Sato [182] signale que la méthode n'est pas extensible à ce type de surfaces.

Un des précurseurs et spécialistes dans le domaine du recouvrement de réflectances dans des images est, sans aucun doute, Ikeuchi. Outre le fait qu'il a également travaillé sur les fonctions de réflectance [144], on trouve plusieurs articles de cet auteur sur le seul domaine de l'analyse de ces fonctions dans des images réelles, souvent en collaboration avec Y. Sato. La première contribution scientifique sur ce sujet date d'ailleurs de 1991 [179], et est décrite plus loin pour des raisons de présentation (il n'utilise qu'une seule image). On s'intéresse ici à une technique décrite par Sato et Ikeuchi [181], utilisant un appareil spécifique de capture d'images d'une part et d'autre part une combinaison linéaire du modèle de Lambert et du modèle de Torrance-Sparrow comme approximation de BRDF (cette technique a également été développée dans [180] en l'appliquant à des scènes où la source de lumière se déplace).
Un ensemble d'images couleurs de l'objet à analyser est donc capturé, à l'aide d'un appareil effectuant une rotation à 360 degrés autour de lui. Cet appareil permet également d'enregistrer des cartes de profondeurs de l'objet depuis chaque position : un modèle 3D de l'objet sous forme de facettes triangulaires est donc reconstruit, en fusionnant ces cartes suivant la méthode de Levoy et Turk [223]. Depuis ce modèle 3D, et en le reprojetant sur l'image à l'aide de l'algorithme du *Z*-

Buffer pour obtenir les facettes visibles, Sato et Ikeuchi extraient de sa projection les intensités de pixel et séparent la composante diffuse de la composante spéculaire[12]. À l'aide de ces composantes, Sato et Ikeuchi calculent les paramètres du modèle de Lambert pour en déduire le coefficient diffus k_d, et les paramètres k_s et c du modèle de Torrance-Sparrow (encore une fois très simplifié dans les expérimentations, puisque les termes de Fresnel et d'atténuation géométrique sont supposés constants : la direction incidente coïncide avec la direction de réflexion).
La méthode proposée par Sato et Ikeuchi présente l'avantage d'être capable de retrouver les réflectances (et la géométrie 3D) des surfaces à partir d'un ensemble d'images, mais elle souffre des plusieurs défauts qu'elle a en commun avec celle de Lu et Little. Bien qu'elle puisse traiter les objets à réflectance non-uniforme, elle ne peut prendre en compte les objets très texturés puisque chaque image capturée doit tout de même présenter des régions uniformes. Elle ne peut traiter, bien sûr, qu'un seul objet à la fois et nécessairement convexe. Enfin, la réflexion spéculaire n'est prise en compte que s'il y a un éclat spéculaire sur l'objet : la manière dont l'algorithme pourrait traiter un miroir n'est pas précisée dans l'article, et il est à priori peu probable que cette méthode puisse traiter de tels objets puisqu'ils interagissent avec leur environnement et sont donc probablement hautement -et indirectement- texturés par lui.

Cherchant à résoudre ce problème de surface hautement texturée, Sato, Wheeler et Ikeuchi proposent une nouvelle technique [182], mais toujours basée sur le principe précédent de séparation des composantes diffuse et spéculaire. Les ambitions de cette méthode restent les mêmes qu'auparavant, puisque les auteurs cherchent à obtenir un modèle 3D ainsi que ses propriétés de réflectance.
Dans un premier temps, plusieurs images couleurs de l'objet posé sur un bras mécanique, sont capturées avec une caméra CCD, et des cartes de profondeurs associées sont obtenues avec un dispositif spécifique[13]. Au total, 120 images couleurs et 12 cartes de profondeur furent nécessaires pour l'analyse de l'objet.
La reconstruction 3D de l'objet sous forme de maillage triangulaire est réalisée par une technique développée par Sato et al. dans [243], qui vise à extraire l'isosurface du volume 3D des cartes de profondeurs, après une étape d'alignement et de fusion de celles-ci. La seconde étape de modélisation consiste à estimer les normales aux points, comme les vecteurs perpendiculaires aux vecteurs propres de la matrice de covariance du voisinage 3D du point sélectionné[14]. L'analyse photométrique commence alors, par une séparation des composantes diffuse et spéculaire dans la séquence d'images couleurs. Comme la position de l'observateur et de la source de lumière sont connues, on peut directement estimer les coefficients diffus k_d pour toutes les images (une image de coefficients est en fait calculée : elle contient la texture notamment). Concernant la partie spéculaire, comme elle n'est observable que sous certaines conditions particulières et comme certaines discontinuités peuvent apparaître d'une image à l'autre, Sato et al. proposent de sélectionner, dans un premier temps, un ensemble de points en fonction de l'intensité de leur projection dans l'image. Dans un second temps, pour cet ensemble de points, des valeurs de spécularité (k_s) et de rugosité (σ dans le modèle de Torrance) sont estimées (résolution d'un système d'équations linéaires) puis interpolées. Ceci construit une image de coefficients de spécularité, qui, combinée avec l'image des réflectances diffuses et les normales calculées précédemment, permet de calculer une nouvelle image de synthèse.
Cette méthode permet donc de retrouver des réflectances non-uniformes et texturées de surfaces de manière robuste, dans une séquence d'images couleurs. Cependant, il est précisé que les interréflexions de l'objet (concave) sont négligées et n'influent pas sur l'image. Ceci n'est vrai que sous certaines conditions d'illuminations. Typiquement dans cet article, il s'agit d'une tasse et il est

[12] Cette idée est motivée par le fait que Kay et Caelli [112] signalent qu'estimer les deux composantes à la fois produit des résultats instables.

[13] Le dispositif est en réalité un laser à triangulation par nappe (a *light-stripe range finder* en anglais) dans l'article.

[14] L'article explique en fait que les normales sont les vecteurs propres de cette matrice de covariance. Ceci est inexacte, puisque ces vecteurs sont dans le plan qui passe par le voisinage 3D du point, dont on cherche la normale. Le vecteur recherché est donc bien orthogonal à ces vecteurs propres.

tout à fait probable que l'anse de la tasse puisse générer une ombre sur celle-ci (à moins d'être dans des conditions d'éclairement spécifiques, ce qui n'est pas précisé). Dans ce cas où l'objet s'auto-ombrerait, la procédure d'analyse serait faussée dans les zones ombrées. Par ailleurs, si la tasse était spéculaire comme un miroir, l'influence de l'énergie réémise par celle-ci sur l'anse (ou le contraire) ne serait plus négligeable[15]. Dans la mesure où cette technique ne prend jamais en compte l'illumination globale, ni les inter-réflexions, il est évident que beaucoup d'objets réels ne peuvent être traités en l'utilisant. Les autres défauts de cette méthode réside dans l'utilisation d'un dispositif spécifique (bras mécanique et système de rotation 3D), dans le nombre d'images à capturer (plus d'une centaine ici) et dans le nombre d'objets analysables à la fois (un seul).

Utilisant le même principe de rotation de la caméra autour de l'objet pour l'analyser, Marschner [137, 136] propose de retrouver les réflectances des surfaces de plusieurs façons possibles, en fonction du type d'image.
Considérant dans un premier temps, les objets à BRDF quelconques (mais isotropes), et partant d'un ensemble d'images de l'objet (plusieurs dizaines), et d'une représentation 3D obtenue soit par un scanner 3D soit en mesurant le modèle réel, Marshner utilise directement la définition de la BRDF pour en calculer les valeurs. Il estime en effet, le rapport de la luminance reçue par les pixels de l'image sur l'éclairement reçu par l'objet depuis la source de lumière. Depuis les directions incidentes et réfléchies connues, et pouvant donc calculer les valeurs de la BRDF pour ces directions, il est possible de construire une BRDF, en fonction de ces trois termes. Marschner applique sa technique à la réillumination de l'objet sous de nouvelles conditions. Une extension de cette technique est également disponible dans [138], où il a choisi de représenter les BRDF des objets étudiés par le modèle de Lafortune et al. [121]., et d'appliquer son algorithme de réillumination sur des visages humains, dont il a mesuré la BRDF. Marschner y a également simplifié son dispositif de capture et a étendu sa méthode aux objets concaves.
Par ailleurs, Marschner propose également une seconde méthode ([136], pages 41-72) pour retrouver la BRDF d'un objet texturé, mais en l'approximant comme un modèle Lambertien, sur lequel on applique un terme de modification de réflectance de la surface pour simuler des éclats spéculaires : le meilleur terme de lissage correspondant aux images réelles est trouvé en utilisant le modèle isotrope de Ward [236].
Plusieurs inconvénients majeurs découlent de la première technique, à commencer par le protocole d'expérimentation pour la saisie de données et la sophistication du dispositif employé (voir [136], page 88). Ce protocole est, en effet, lourd et ne fonctionne que pour un objet seul, nécessairement convexe et sur lesquel les changements d'orientation des normales ne sont pas trop abruptes (tous ces défauts ont été corrigés dans [138]). Il s'affranchit ainsi de la remarque précédente sur l'auto-ombrage dans l'article de Sato et al. [182](on remarquera que dans [138], l'utilisation d'un logiciel de tracé de rayons lui permet de résoudre le problème des ombres[16]). Cependant, la méthode ne peut toujours pas traiter les surfaces texturées, ni les surfaces anisotropes. Et comme pour les articles précédents, on voit mal comment le fait de négliger les inter-réflexions (même pour un objet convexe), lui permet de traiter des BRDF générales, puisqu'un simple objet miroir (une boule de Noël par exemple) présente tellement d'inter-réflexions avec l'environnement, qu'il est impossible d'en extraire les propriétés sans en tenir compte. Enfin, le nombre d'images nécessaires pour traiter un seul objet, comme une sphère ou un cylindre, est de trente.
La seconde technique parvient à simuler des objets légèrement spéculaires par une approximation lambertienne de sa BRDF, modulo un terme de régularisation pour calculer la composante spéculaire. L'avantage de la méthode est qu'elle permet de prendre en compte les objets fortement texturés et concaves. Les inconvénients sont presque les mêmes que précédemment, excepté que le côté texturel du traitement force à réaliser en plus une approximation pour trouver le terme spéculaire de la BRDF, tandis qu'une telle approximation n'est pas nécessaire dans la première méthode.

[15] On pourrait également trouver des situations où l'objet, s'il était diffus uniquement, créerait des zones de mélanges de couleurs sur lui-même. Ainsi une anse rouge vif sur une tasse blanche produirait probablement un phénoméne fort d'inter-réflexions (*color bleeding* en anglais).

[16] L'algorithme n'utilise cependant pas d'illumination globale. On ne voit donc pas comment la méthode peut prendre en compte les effets de *color bleeding*.

Enfin, la dernière technique que nous décrivons ici pour retrouver les réflectances est radicalement différente des précédentes. Wong et al. [218, 246, 247] proposent une technique originale, en calculant la réflectance de chaque pixel. En effet, le plan image est considéré comme un objet à part entière, constitué de plusieurs éléments de surface ayant chacun leur propre BRDF, représentée comme une somme d'harmoniques sphériques. Plusieurs images sous des conditions d'illumination distinctes et avec des points de vue différents sont calculées (s'il s'agit d'images réelles, on stocke juste la position de l'observateur et la direction incidente de la source de lumière placée à l'infini). La BRDF d'un pixel (ou élément de surfaces) est alors estimée comme le rapport de l'intensité du pixel sur celle de la source de lumière, pour chaque direction incidente et réfléchie. Pour changer le point de vue de la scène et réilluminer un objet dont la BRDF a été calculée, avec une source de lumière autre que directionnelle (les rayons de lumière dans ce cas précis sont tous parallèles), Wong et al. ont besoin d'une carte de profondeur qui doit être fournie également en entrée.
L'avantage de cette méthode est sa simplicité et la possibilité qu'elle offre de réaliser rapidement des effets visuels satisfaisants, notamment sur les changements de conditions d'éclairement. De plus, le recouvrement des BRDF est indépendant de la complexité de la scène, puisque tous les calculs s'effectuent sur l'image. Néanmoins, plusieurs inconvénients restent présents. Tout d'abord, la technique n'a pas été testée sur des images réelles, et bien que Wong et al. précisent que cela ne pose aucun problème, aucune reconstruction depuis une photo n'est montrée dans aucun des trois articles [218, 246, 247]. Par ailleurs, l'ajout d'objets nouveaux semble impossible, puisqu'à aucun moment, on ne connaît les conditions d'éclairement d'origine. La méthode montre également des résultats sur des scènes très simples (pas plus de deux objets), et il est probable qu'elle ne fonctionnerait pas correctement dans des scènes complexes, où les objets seraient soumis à des inter-réflexions fortes, car l'illumination globale n'est pas gérée[17]. Enfin, on constate des phénomènes d'aliassage importants dans les images, pour lesquels nous n'avons pas trouvé de raison immédiate : une explication intuitive tendrait à imaginer que la discrétisation spatiale de l'image (la surface examinée en fait) en intensités de pixel n'y est pas étrangère.

5.3.2 Méthodes avec gestion des inter-réflexions

Les méthodes utilisant l'illumination globale sont souvent plus puissantes que les méthodes du paragraphe précédent, car elles peuvent simuler les inter-réflexions pour raffiner la recherche de paramètres de réflectance.

L'un des premiers à employer l'illumination globale pour des applications de réalité augmentée fut Debevec [45]. Son objectif est d'insérer des objets de synthèse dans une scène réelle, en prenant en compte leurs influences éventuelles sur l'environnement (et vice-versa). Par ailleurs, l'idée sous-jacente qui motive cet article, est de pouvoir réaliser une telle opération sans modéliser toute la scène, car cette opération est longue et fastidieuse, et le calcul d'une image par illumination globale très coûteux. À l'aide d'une sphère parfaitement réfléchissante placée à côté de la zone d'insertion, un ensemble d'images omnidirectionnelles de la scène, est capturé, selon des temps d'exposition différents. Debevec reconstruit ensuite la fonction de conversion des luminances en intensités de pixel suivant la méthode décrite dans [46]. La scène est alors découpée en trois parties[18]. La première est la *scène distante* qui représente l'environnement et ne contient que des luminances (pas de BRDF donc), car celui-ci est éloigné des objets *locaux* qui nous intéressent. Cette *scène distante* servira comme source de lumière, en émettant de l'énergie depuis ses textures et en ne réémettant pas l'énergie reçue. La seconde partie est la *scène locale*, où les objets virtuels vont s'insérer, et qui va être influencée par cette insertion. Cette scène locale est modélisée géométriquement en 3D. Enfin, la dernière partie contient uniquement les objets de synthèse, eux-mêmes. Dans la mesure

[17] On remarquera que dans les trois articles, une théière miroitée est calculée sous des conditions nouvelles d'illumination. Si la méthode permet de changer les conditions d'éclairement de cet objet, il est peu probable qu'on puisse changer le point de vue sur cet exemple, car l'aspect fortement spéculaire de l'objet nécessiterait une connaissance totale de l'environnement pour pouvoir calculer les nouvelles réflexions.

[18] Cette opération est manuelle, et c'est à l'utilisateur de définir ces parties.

où seule la partie locale de l'image réelle est modélisée, la recherche des paramètres de réflectance ne s'effectue que sur les objets qui y sont définis (cette approximation part du principe que l'insertion future des objets de synthèse n'influence pas l'environnement). Le calcul des BRDF des objets s'effectue par un processus itératif, mais manuel (si les objets ne sont pas parfaitement diffus). L'utilisateur fixe donc les valeurs de réflectance[19] des objets réels de la *scène locale*, et calcule ensuite une image par illumination globale, en utilisant *Radiance* [237], et en négligeant donc l'énergie devant être réémise par la *scène distante*. Si l'image est satisfaisante, le processus s'arrête là, sinon l'utilisateur doit modifier les paramètres de réflectance des objets pour obtenir une meilleure image (sauf dans le cas d'objets parfaitement diffus où un calcul automatique a été réalisé). Lorsque l'image regénérée est satisfaisante, il devient possible d'insérer et/ou retirer des objets virtuels ou réels.
Cette méthode est vraiment intéressante, car elle permet une multitude d'applications. Le fait de recouvrer les réflectances des surfaces dans la zone d'intérêt permet d'accélérer les temps de calcul, et l'utilisation de l'illumination globale pour resynthétiser des scènes permet un réalisme maximal. Cependant on regrette que la procédure de correction des réflectances soit manuelle, car elle peut être longue et pénible, si beaucoup d'objets non diffus sont présents, et elle peut être subjective puisque l'utilisateur décide du moment où l'image est suffisamment proche de l'originale. Par ailleurs, la modélisation très approximative de l'environnement limite les applications : l'utilisateur ne peut se permettre de grands changements de points de vue (l'impression de perspective serait faussée), et il est difficile d'insérer des objets dont la contribution énergétique influerait sur l'environnement (comme des sources de lumière par exemple ou d'autres objets influant indirectement sur la *scène distante*).

On a clairement compris que le processus de recouvrement des réflectances dans l'article précédent constituait son principal point faible. Yu et al. proposent donc de remédier à cet inconvénient, en développant une technique retrouvant des BRDF quelconques dans une séquence d'images [251, 252], avec quelques restrictions cependant. Tout d'abord, et cela est un peu antinomique avec les ambitions de l'article précédent, la scène doit être modélisée en totalité, ce qui peut constituer une lourde tâche[20]. Ensuite, on ne peut retrouver les BRDF des objets, que si au moins un éclat spéculaire de l'objet est visible sur une des images de la séquence, sinon l'objet est supposé parfaitement diffus.
Les données pour cet algorithme sont les suivantes : un modèle géométrique 3D de la scène comprenant la caméra, plus d'une centaine d'images, la position et l'intensité des sources de lumière. La première étape est de retrouver la fonction de conversion des luminances en intensités de pixel : ceci s'effectue grâce à [46], qui calcule 40 cartes de luminances depuis les 150 images de départ (dont 12 spécifiquement pour obtenir des éclats spéculaires de toutes les surfaces de la scène dans les images). Dans le cas où toutes les surfaces seraient diffuses, et comme on a la radiosité de toutes les surfaces de la scène, on peut directement inverser l'équation de radiosité pour obtenir les réflectances[21]. La seconde étape est donc le recouvrement des BRDF, qui peuvent être anisotropes puisque les auteurs utilisent le modèle de Ward [236]. Sachant qu'au moins un éclat spéculaire est visible dans l'ensemble des images, une minimisation par la méthode des moindres carrés, est employée pour minimiser l'erreur entre l'image réelle et l'image regénérée par *Radiance* [237] et retrouver trois paramètres pour une surface isotrope (ρ_d, ρ_s, α) et cinq pour une surface anisotrope $(\rho_d, \rho_s, \alpha_x, \alpha_y, \vec{x})$. Les images résultats montrent que la méthode fonctionne de manière très satisfaisante, sur des images réelles et synthétiques plus ou moins complexes, mais les auteurs précisent cependant qu'aucune preuve formelle de convergence de l'algorithme n'est apportée. Par ailleurs, étant donné que l'on a une représentation géométrique 3D et photométrique totale de la scène,

[19] Bien que cela ne soit pas explicitement dit dans l'article, on imagine qu'il s'agit des paramètres issus du modèle de Ward [236], car les auteurs utilisent *Radiance* (*Radiance* [238] est un logiciel puissant de rendu réaliste par illumination globale développé par Greg Ward, et dont on peut trouver une version complète téléchargeable sur le site internet : http ://radsite.lbl.gov/radiance/HOME.html .) pour calculer une image de synthèse.

[20] Notre technique dispose de la même limitation.

[21] Si nous ne disposons que d'une seule image et ne voyons pas toute la scène, cette opération ne peut que conduire à une approximation des réflectances des surfaces. Dans certains cas, elle peut même s'avérer impossible.

toutes les applications sont possibles : ajout d'objets, changement de conditions d'éclairement, changement des propriétés des surfaces, etc.
Bien que cet algorithme soit capable de retrouver des BRDF quelconques (isotrope ou non, surface texturée ou non) dans des scènes réelles, on peut tout de même en objecter certains aspects. Tout d'abord, le nombre d'images nécessaires à la reconstruction photométrique est conséquent : 150 pour la scène d'intérieur montrée dans l'article et comportant une dizaine d'objets. Ensuite, l'algorithme est incapable de traiter des scènes comportant des objets spéculaires avec un éclairage indirect. Ainsi, une pièce possédant des miroirs, des plaques *glossy* et éclairée par un halogène orienté vers le plafond, se verrait simulée comme un environnement parfaitement diffus. Plus grave, tout objet spéculaire (y compris des miroirs parfaits) sera simulé comme un objet diffus, du moment qu'un éclat spéculaire n'est pas visible dans les images. Pour résoudre ce problème, il faut prendre d'autres images sous des angles spécifiques, ce qui alourdit fortement le protocole de construction des données, qui devient très contraignant. Enfin, l'algorithme met tout de même 3 heures pour retrouver toutes les BRDF des surfaces de la scène d'intérieur réelle (excluant le temps passé à construire la scène), et 30 minutes pour une scène synthétique comportant 6 objets plans.

La technique développée par Loscos et al. [129, 130, 131] part d'une idée d'Alain Fournier [66], et permet notamment de calculer les réflectances diffuses de surfaces depuis des images réelles. En fait, les motivations principales des auteurs sont, d'une part, de pouvoir insérer de manière très rapide des objets de synthèse, et, d'autre part, de pouvoir changer les conditions d'illumination de la scène (ajout d'une nouvelle source de lumière par exemple), tout en utilisant du matériel vidéo grand public. Tout d'abord, une représentation géométrique 3D de la scène est créée [22]. Un ensemble d'images est capturé (dont une de chaque source de lumière), sous des conditions d'illumination différentes, et de telle sorte que toutes les parties ombrées d'une image sont visibles au moins une fois, et directement éclairées dans l'une des autres images. La raison de cette contrainte réside dans le fait que Loscos et al. procèdent à des découpes successives de textures dans les images réelles pour ensuite les replaquer sur les surfaces 3D virtuelles. Si les textures étaient découpées sans tenir compte des ombres éventuellement créées par des objets voisins, elles seraient replaquées sur les surfaces 3D avec ces ombres, empêchant ainsi de retirer la source de lumière qui les a générées, et donc de modifier les conditions d'illumination (l'ombre resterait tout de même présente). La fonction de conversion des luminances en intensités de pixel, est calculée par la méthode de Debevec [46], adaptée au paramètre EV des caméras standards (voir définition dans [131]), et fournit donc les cartes de luminances correspondant aux images. Par ailleurs, comme les sources de lumière sont déplacées dans la scène lors de la capture, chacune des images doit être normalisée par un facteur d'échelle propre, afin que la réflectance soit bien la même d'une image à l'autre, pour une même surface visible. Ce facteur est estimé en minimisant l'erreur aux moindres carrés entre l'image que l'on cherche à corriger et une image référence, calculée par illumination globale depuis une approximation des réflectances des surfaces visibles. Cette approximation est directement dérivée de l'équation de radiosité, adaptée pour un pixel, et ne tenant compte que de l'éclairage direct.
Une fois les cartes de luminances obtenues, et la "normalisation" des images réalisées, l'estimation des réflectances peut commencer. Elle s'effectue en trois étapes. La première consiste à calculer une réflectance pour chaque pixel de chaque image, en inversant directement l'équation de radiosité adaptée aux pixels. Un terme de visibilité est donc calculé, entre le point 3D associé à un pixel, et la source de lumière. L'illumination indirecte est initialisée à une luminance ambiante pour tous les pixels[23], et est calculée comme la moyenne des intensités de l'image, divisée par le facteur de réflectance total[24]. La radiosité du pixel est initialisée à la luminance correspondante dans la carte de luminances. La seconde étape consiste à attribuer à chacune des réflectances précédentes, une valeur de *confiance*, qui reflète le niveau de confiance que l'on a dans cette réflectance (cette valeur est typiquement faible pour les zones éclairées indirectement), puis à filtrer les résultats.

[22] Cette construction peut se réaliser, par exemple, en utilisant *Facade [49]*.
[23] Il s'agit ici d'une importante approximation.
[24] Le facteur de réflectance total est fourni par l'utilisateur.

La troisième étape utilise les réflectances estimées à la première étape depuis chaque image de luminances, pour obtenir une seule réflectance par pixel : chaque nouvelle réflectance est donc calculée comme la réflectance moyenne dans les images, pondérée par les valeurs de confiance de la seconde étape. Ces réflectances servent à initialiser la génération d'une nouvelle image, rendue par radiosité hiérarchique [191]. Contrairement à [129], [131] propose de modifier itérativement les réflectances des surfaces, en remplaçant dans l'équation inversée de radiosité, la valeur constante et très approximative d'illumination indirecte, par la nouvelle, obtenue lors du rendu précédent : le processus s'arrête lorsque les réflectances sont stables[25].
Cette technique est intéressante, car elle permet de retrouver les réflectances de surfaces, dans des images réelles selon un processus itératif, tout en conservant l'information texturelle initiale. Par ailleurs, les applications montrées permettent de retirer et/ou ajouter des objets réels ou non, et de modifier les conditions d'illumination de la scène de façon interactive. Cependant, on peut objecter plusieurs aspects de cette méthode. Tout d'abord, le protocole de saisie des images est assez contraint puisqu'il faut arriver à acquérir chaque image de telle sorte que chaque surface apparaisse au moins une fois totalement éclairée, et le nombre d'images à prendre est tout de même au nombre de 10 pour une scène très simple[26]. On ne peut traiter que des surfaces diffuses, et étant donné que Loscos et al. découpent les textures dans l'image, toute surface spéculaire ou *glossy* sera considérée comme une texture : il s'agit là d'un problème très classique déjà mentionné par Sato dans [182]. D'ailleurs, même si les applications restent très intéressantes, les objets qui peuvent être déplacés doivent être des objets n'influant pas trop sur l'environnement (Debevec [45] avait choisi de découper la scène en plusieurs parties pour prendre en compte ce problème). En effet, si on choisit, avec cette méthode, de déplacer la chaise dont les pieds spéculaires se reflètent sur le sol, les reflets resteront présents. On remarquera enfin dans [130, 131], sans doute pour des raisons visuelles, que la topologie de la scène où la porte est retirée, a été modifiée, puisqu'elle est remplacée par la texture du mur (ce qui pourrait signifier que la porte est posée contre le mur).

Plus récemment, Loscos et al. [132] proposent une extension de la méthode développée dans [56][27], et résolvent ainsi les problèmes inhérents aux ombres dans les textures présentes dans [56], ainsi que dans les deux articles précédents [131, 130]. Les motivations de cet article diffèrent également légèrement, puisqu'il n'est question ici que de pouvoir modifier les conditions d'illumination de la scène. Loscos et al. proposent également une méthode pour améliorer les réflectances obtenues pour les surfaces texturées.
Partant d'un ensemble d'images réelles (12 dans l'article) capturées avec une caméra, d'un modèle géométrique 3D simple de la scène (obtenu d'une façon similaire à [131, 130]), Loscos et al. cherchent à calculer les textures non ombrées[28] pour pouvoir modifier l'illumination de la scène : il s'agit en fait de s'affranchir de la limitation précédente, qui forçait à prendre des images sous des conditions d'éclairement différentes, pour éviter des ombres sur les textures. On remarquera que, bien que les sources de lumière employées soient identiques, il est possible, dans le cas de sources différentes, de retrouver leurs intensités en résolvant un système d'équations linéaires, comme dans [56].
Dans un premier temps, les réflectances des surfaces (et non pas des pixels comme dans les deux articles précédents) sont estimées comme les moyennes des intensités de pixel dans la projection de ces surfaces (cette méthode provient des articles [66] et [56]). Un algorithme de radiosité progressive est alors employé pour créer une nouvelle image de synthèse. Le calcul des textures non ombrées s'effectue par l'application successive de plusieurs facteurs d'échelle, calculés selon un ensemble d'heuristiques. Il est alors nécessaire d'estimer les frontières d'ombre/pénombre (uniquement par éclairage direct) pour en extraire la partie de texture ombrée. La lumière manquant à cette partie de texture y est ensuite ajoutée, en la modulant par l'éclairement *bloqué* issu du calcul

[25] Les auteurs précisent que seules 3 ou 4 itérations suffisent à obtenir des réflectances stables.
[26] On imagine que s'il y avait beaucoup de sources de lumières dans l'image réelle, la capture de ces images serait très longue.
[27] Nous décrivons l'article [56] plus loin, car il n'utilise qu'une seule image pour retrouver les réflectances, tandis que l'article [132] qui nous intéresse ici en utilise plusieurs.
[28] Les auteurs parlent en fait de *unoccluded illuminated textures*.

de radiosité, pour créer une *texture intermédiaire* (et donc une réflectance *intermédiaire*). Cette opération provoque une surestimation de la radiosité de la surface ombrée (la texture semble trop illuminée), qui doit donc être corrigée. Cette correction est appliquée en recherchant d'abord une surface voisine éclairée directement, et en calculant un facteur d'échelle : une réflectance *corrigée* est créée comme le produit de la réflectance de la surface voisine par le rapport entre le facteur de forme de la surface éclairée vis-à-vis de la source de lumière, sur celui de l'élément ombré vis-à-vis de la même source. La texture finale non ombrée, est obtenue en multipliant la texture intermédiaire par le rapport entre la réflectance *corrigée* et la réflectance intermédiaire. Cette texture peut alors être reprojetée sur la surface à laquelle elle appartient, en étant simplement pondérée par le rapport de la radiosité avec ombres sur la radiosité sans ombres.
L'intérêt principal de cette technique est de pouvoir calculer une texture non ombrée, permettant de changer toutes les sources de lumière de la scène réelle, et d'en ajouter d'autres, virtuelles. L'insertion de nouvelles sources de lumière est très rapide. Néanmoins, plusieurs inconvénients apparaissent. En premier, le fait que les ombres prises en compte ne soient que celles issues de l'éclairage direct : un halogène dirigé vers le plafond génèrera une multitude d'ombres dans la scène de façon indirecte. Les textures non ombrées auront ces ombres, car elles ne font pas partie des ombres *directes*. Dans le mème ordre d'idée, des inter-réflexions diffuses violentes produiraient probablement des résultats erronés pour l'extraction des textures, qui contiendraient ces inter-réflexions. Par ailleurs, le calcul des zones d'ombre nécessite une connaissance parfaite de la géométrie des sources de lumière, pour extraire les textures ombrées de manière correcte. Les surfaces spéculaires ou avec des BRDF complexes ne peuvent être prises en compte, puisque l'on procède à des découpes de textures dans l'image (problème déjà existant dans les deux articles précédents). Enfin, la modélisation approximative de la scène empêche tout changement de point de vue pour le moment, et les auteurs signalent que l'extension de la technique à l'insertion/suppression d'objets réels et/ou virtuels n'est pas triviale, et n'a pas encore été réalisée.

5.4 Estimer les réflectances depuis une seule image

Les méthodes précédentes présentaient l'avantage, pour les plus sophistiquées, d'être capables de retrouver les BRDF quelconques de surfaces, depuis un ensemble d'images réelles, parfois fort important (plus d'une centaine). Ces images sont parfois soumises, en plus, à des contraintes particulières, que ce soit sur le type de surface, la géométrie de la scène ou la photométrie. Les techniques qui vont suivre, et elles sont nettement moins nombreuses, permettent de retrouver les paramètres de réflectance des surfaces, mais depuis une seule et unique image, quelquefois omnidirectionnelle cependant. Tenter de reconstruire les BRDF de surfaces, avec une seule image, est un défi nettement plus important qu'avec une séquence, puisque la fonction de réflectance est dépendante des directions incidentes et réfléchies, et n'est donc pas souvent mesurable dans une seule image, pour toutes les surfaces. Ainsi, contrairement à certaines techniques précédentes, le recouvrement des BRDF n'est pas nécessairement exact : il doit servir à simuler une image de synthèse, aussi proche que possible de l'image réelle, de manière à créer l'illusion d'une reconstruction photométrique parfaite. Encore une fois, nous pouvons distinguer les techniques qui vont utiliser les inter-réflexions entre les objets, et celles qui ne s'en serviront pas, et qui sont généralement restreintes à un seul objet.

5.4.1 Méthodes sans gestion des inter-réflexions

Le premier article à s'intéresser au recouvrement de réflectances dans une seule image, est celui de K.Sato[29] et K.Ikeuchi [179]. L'idée est ici de pouvoir estimer les réflectances d'un objet unique, en se servant d'une représentation 3D et d'une image réelle (photographie) de cet objet. L'algo-

[29] À ne pas confondre avec Y.Sato, auteur également de plusieurs articles avec Ikeuchi, et sur le même domaine.

rithme se sert donc d'un estimateur de profondeurs[30], qui fournit trois images X, Y, Z en fonction des pixels i, j de l'image. Ces cartes permettent ainsi d'estimer la normale en tout point de l'objet, examiné dans l'image de luminosités (obtenue par capture avec une caméra TV). Ikeuchi et Sato proposent de se servir du modèle de Torrance-Sparrow [220, 219], pour retrouver la réflectance diffuse et la réflectance spéculaire de l'objet. Ce modèle est, comme pour les articles précédents l'utilisant, une version simplifiée, puisque le terme de Fresnel et le terme d'atténuation géométrique sont supposés constants (si la direction de réflexion et la direction incidente ne dépassent pas 60 degrés[31], ces deux termes sont constants, sinon il n'est pas tenu compte des valeurs calculées). Les autres hypothèses sont que la source de lumière est à l'infini (pour obtenir des directions incidentes identiques) mais de direction inconnue (elle est également retrouvée par l'algorithme), l'observateur est loin de l'objet (la projection se restreint donc à une projection orthographique), et le matériau de l'objet est uniforme.
Dans un premier temps, Ikeuchi et al. recherchent le paramètre lambertien par une minimisation d'erreur aux moindres carrés, entre les valeurs de pixels observées et les valeurs nouvellement estimées. La direction de la source de lumière est également directement déductible de cette minimisation. À l'aide de l'image des profondeurs et du paramètre lambertien obtenu, il est possible de séparer les intensités de pixel en trois catégories : les pixels d'inter-réflexions ou spéculaires, les pixels lambertiens, les pixels d'ombre. De la même façon que pour la partie lambertienne, une méthode de minimisation est utilisée pour calculer les paramètres de spécularité du modèle de Torrance. Ceci permet d'affiner les critères de tri des pixels, en 4 parties (pixels d'inter-réflexions, pixels spéculaires, lambertiens, ou d'ombres) et de calculer de nouvelles images avec des conditions d'illumination nouvelles et optimales.
Bien que la méthode ait été testée sur des objets synthétiques ou réels (notamment un visage), les défauts inhérents à cette méthode sont nombreux. Tout d'abord, le nombre d'hypothèses de départ est bien trop important pour pouvoir appliquer la technique à des images réelles de scènes d'intérieur : analyse restreinte à un objet seul, terme de Fresnel et géométrique constants, source de lumière à l'infini, observateur éloigné de l'objet, matériau uniforme, etc. Par ailleurs, le fait de négliger les phénomènes d'inter-réflexion avec l'environnement empêche un traitement correct d'une surface miroir par exemple, et dont la réflectance est pourtant uniforme.

Beaucoup plus récemment, I.Sato[32], Y.Sato et Ikeuchi ont proposé une méthode pour trouver la réflectance bidirectionnelle d'un objet (toujours selon le modèle de Torrance-Sparrow), dans une image réelle, et sur lequel il existe une ombre projetée [177]. En utilisant une image omnidirectionnelle de l'environnement, et une représentation 3D des objets ombrés et des objets générant les ombres en question, Sato et al. recherchent la distribution de luminances dans les zones d'ombre (appelée l'*image d'ombres*) et calculent les BRDF. L'algorithme procède en 6 étapes distinctes, dont la première est l'initialisation de la réflectance de la surface ombrée, à une réflectance diffuse (k_d est initialisé à la valeur de luminance la plus forte, et $k_s = \sigma = 0$). Dans un second temps, la distribution des luminances est calculée en utilisant les paramètres k_d, k_s, σ et les intensités de pixel se trouvant dans l'image d'ombres. Une troisième étape estime les nouveaux k_d, k_s, σ de la surface ombrée, en minimisant par la méthode de Powell, l'erreur entre l'image regénérée et l'image réelle. Lors de la quatrième étape, la distribution de luminances est alors réévaluée avec les nouvelles réflectances en utilisant un système d'équations linéaires. Si les réflectances n'ont pas encore convergé, la cinquième étape procède à la réitération depuis la troisième. Enfin, l'algorithme procède à la comparaison des distributions de luminances dans l'image regénérée, avec celles de l'image réelle. Si cette comparaison n'est pas satisfaisante (les deux distributions sont encore éloignées), le nombre d'échantillons de la source de lumière est augmenté, et le processus redémarré depuis la seconde étape.
Cette technique est intéressante, car elle permet de calculer les BRDF de surfaces, dans des images réelles complexes, tout en adaptant sa recherche en fonction des résultats (suréchantillonnage des sources de lumière). Cependant, le fait de focaliser sa recherche de réflectance, en suivant les

[30] *A range finder* en anglais.
[31] Ceci est en contradiction avec [112] et [211] qui signalent que l'angle limite est de 30 degrés.
[32] À ne pas confondre, ni avec Kosuke Sato, ni avec Yoichi Sato.

ombres générées par les sources de lumière pose plusieurs problèmes. Tout d'abord, cela signifie que si l'objet n'est pas ombré, il est impossible de retrouver sa BRDF. Ensuite, comme les inter-réflexions ne sont pas prises en compte, les ombres issues de l'éclairage indirect seront négligées, tout comme les phénomènes de *color bleeding*, qui peuvent fortement perturber les calculs de réflectance. Enfin, les textures ne sont pas traitées, ce qui provoque de fortes discontinuités autour de la zone d'ombre regénérée, entre l'image réelle et l'image synthétique[33].

5.4.2 Méthodes avec gestion des inter-réflexions

Ces techniques sont, pour nous, les plus ambitieuses, car elle partent d'un minimum de données tout en utilisant la quintessence du rendu réaliste (l'illumination globale). Nos travaux s'inscrivent dans le même cadre que les articles de cette catégorie, et décrits plus loin. On notera, par ailleurs, que les solutions existant pour un tel problème sont bien moins nombreuses que pour les précédents.

Une des techniques pionnières, en matière de rendu réaliste à base d'images réelles, est sans conteste celle apportée par Alain Fournier [66]. Cet article fait aujourd'hui référence en la matière, et a inspiré beaucoup de chercheurs. L'idée novatrice de départ, consiste à partir d'un modèle 3D et d'une image de scène, et en connaissant la position des sources de lumière et de l'observateur, à calculer une nouvelle image de synthèse la plus proche possible de l'image d'origine. Ainsi, Fournier calcule une approximation des réflectances des surfaces présentes dans l'image, en moyennant les intensités de pixel, où se projette chacune d'entre elles. Cette approximation est en fait la réflectivité moyenne, pondérée par le rapport entre la radiosité de la surface et la moyenne des radiosités des pixels voisins. Ce facteur de réflectivité est fixé arbitrairement par l'utilisateur. En utilisant un logiciel de calcul d'illumination globale, qui va estimer les radiosités des surfaces depuis leurs réflectances, il est possible de reprojeter les textures extraites de l'image réelle, sur ces surfaces 3D en les pondérant par ces radiosités. On obtient ainsi une nouvelle image de synthèse proche de l'image réelle. Par ailleurs, dans un premier temps, les intensités des sources de lumière sont initialisées à une valeur unitaire. Après le premier rendu par radiosité, l'intensité des sources peut être recalculée, en résolvant un système d'équations linéaires (inversion de l'équation de radiosité). L'image finale est calculée par lancer de rayons, pour éventuellement simuler des surfaces spéculaires.
Cette idée simple est très efficace pour regénérer des images réalistes, proches d'une image réelle de référence. Cependant, elle est limitée aux surfaces purement diffuses, et ne cherche pas à réaffiner les réflectances trouvées en première approximation, ce qui dans le cas d'inter-réflexions diffuses fortes (un objet rouge, rougissant un objet blanc par exemple) créera des images biaisées.

Une extension de cette méthode a été réalisée par Drettakis et al. [56], qui propose une version interactive. Une seule image est toujours nécessaire pour estimer les réflectances, et les données de départ sont les mêmes que pour Fournier. Cependant, Drettakis et al. proposent également une méthode de vision, pour calculer la calibration de la caméra et du modèle 3D sur l'image réelle. Une légère différence est également ajoutée sur le calcul des réflectances (devenu un peu plus simple), qui sont maintenant considérées comme les moyennes des intensités de pixels où se projettent les surfaces, pondérées par la radiosité ambiante et la réflectivité totale (toujours fixée arbitrairement ou initialisée à la moyenne de toutes les intensités de pixel de l'image). À partir des réflectances ainsi approximées, une nouvelle image est calculée par radiosité hiérarchique (Fournier et al. utilisaient la radiosité progressive classique), et l'affichage finale s'effectue par tracé de rayons (comme Fournier et al.). Drettakis et al. appliquent leur méthode à la réalité augmentée, en ajoutant des objets de synthèse à la nouvelle image ainsi créée.
Bien qu'une méthode de vision, partiellement automatique, soit proposée, et que la technique de calcul des réflectances diffère très légèrement de celle de Fournier et al., les inconvénients présents

[33] Un article précédent des mêmes auteurs [178], résolvait ce problème en considérant que les réflectances étaient connues (ils ne s'intéressaient qu'à la distribution de luminances).

dans l'article de ces derniers restent entiers : l'algorithme ne traite que des surfaces purement diffuses, et on retrouve le même problème que précédemment pour le calcul des inter-réflexions (les réflectances ne sont pas améliorées par un processus itératif, indispensable dans certains cas). Par ailleurs, l'emploi de textures, directement découpées dans l'image, empêche l'extension directe de la méthode aux surfaces spéculaires, situation déjà largement débattue pour les articles précédents.

5.5 Les autres techniques pour réaliser des images photoréalistes à partir d'images réelles

Nous ne tenterons pas ici de décrire, ni même d'énumérer de façon exhaustive, la kyrielle de méthodes existant pour produire des images photoréalistes, depuis des images réelles. D'une manière générale, toutes ces techniques utilisent plusieurs images, n'estiment pas les réflectances des surfaces, et ne savent pas traiter correctement les images où des miroirs sont présents. Nous proposons seulement quelques articles qui nous ont paru significatifs, et qu'il est important de connaître pour l'*Image-Based Rendering*.

L'ensemble des travaux réalisés par l'équipe de Berkeley reste, pour nous, la référence majeure (avec l'article de Fournier [66]), en matière de rendu à base d'images réelles. Debevec proposa, d'ailleurs, une première solution dans [49, 48, 47, 44], en simulant des scènes extérieures, avec des techniques issues de la vision et de la synthèse d'images. En développant une nouvelle technique de plaquage de textures[34] fondée sur l'interpolation des textures de plusieurs photos de l'objet à reconstruire, il parvient à générer des images réalistes de bâtiments architecturaux en extérieur. Ces travaux ont d'ailleurs été implémentés dans un logiciel commercial, du nom de *Facade*, et une application connue est la reconstruction de la cathédrale de Rouen, ou, plus récemment, certains effets spéciaux des films *Matrix* et *Mission Impossible 2*.

Sato et al. [176] proposent une méthode pour ajouter des objets de synthèse dans des images réelles. Outre le fait qu'ils utilisent, en partie, les travaux de Debevec [46] notamment pour la reconstruction des cartes de luminances, Sato et al. ne prennent pas en compte les inter-réflexions entre les nouveaux objets et l'environnement (aucun modèle 3D des objets présents dans l'image réelle n'est créé). Cependant, la génération des nouvelles images s'effectue aussi bien sur des scènes d'intérieur que d'extérieur, et les résultats sont assez convaincants.

Nimeroff et al. montrent dans l'article [150] comment regénérer une nouvelle image, depuis une séquence d'images synthétiques (9 au total) sous des conditions d'illumination extérieures naturelles (ciel, soleil, etc.). Ils recalculent ensuite une nouvelle image de synthèse en utilisant une combinaison linéaire des 9 images précalculées, et appliquent leur méthode aux modifications de conditions d'éclairement de la scène originale.

Havaldar et al. [94, 95] proposent de calculer de nouveaux points de vue, en partant d'un ensemble de photographies, en utilisant les *invariants projectifs* qui y sont présents (un processus semi-automatique leur apporte également l'extraction des contours, des sommets et des faces des objets). Ceci leur permet de construire la géométrie 3D des objets présents dans les images, sans connaître la position initiale de la caméra. À l'aide de cette géométrie et du *Texture Mapping*, il devient possible de synthétiser de nouvelles images, sous d'autres points de vue, pour des scènes d'extérieur, comme d'intérieur.

Enfin, McMillan et al. [139] proposent d'utiliser la fonction plénoptique de Adelson et Bergen [1], qui permet de représenter tout ce qui est visible depuis un point de vue donné. Ils ont développé une technique, pour reconstruire cette fonction depuis un ensemble d'images panoramiques,

[34] *Texture Mapping* en anglais.

ainsi qu'un nouvel algorithme d'élimination de parties cachées. À l'aide de ces outils, ils peuvent générer de nouveaux points de vue, absents du panoramique original.

5.6 Discussion

Nous pouvons effectuer plusieurs déductions des articles précédents. Tout d'abord, on ne peut pas vraiment les comparer entre eux, lorsqu'ils ne sont pas dans la même catégorie. En effet, le fait que le jeu de données initiales ne soit pas le même d'une méthode à l'autre, fausse cette comparaison. Certaines techniques choisissent de ne pas utiliser de logiciel spécifique pour placer les objets sur l'image non pas interactivement, mais manuellement. La justification réside principalement dans le fait que cette opération est souvent longue et fastidieuse. Néanmoins, tous les analystes d'images vous diront, qu'à la longue, cette méthode est moins usante que celle qui consiste à cliquer sur des dizaines de points dans des images différentes, pour les faire correspondre, et en déduire une modélisation tridimensionnelle de la scène. D'autant plus que, souvent, cette correspondance n'est pas aussi immédiate que ce que l'on veut bien faire croire : la machine propose souvent plusieurs correspondances possibles, et l'utilisateur doit préciser celles qui sont justes.

Par ailleurs, il est plus facile de retrouver les BRDF de surfaces quelconques, si l'on dispose d'un ensemble d'images très important. Les directions incidentes et réfléchies des faisceaux de lumière étant plus nombreuses, la recherche du comportement d'une surface face à cette lumière est plus aisément déterminable. Par ailleurs, les surfaces texturées s'en retrouvent également plus facilement traitables, puisqu'en combinant toutes les images réelles, on finit par en déduire où sont les éclats, et comment obtenir la texture originale sans spéculaire. De plus, lorsque l'on dispose de plusieurs vues différentes de la scène, il devient très simple de déterminer quelles sont les surfaces spéculaires, puisque l'apparence de celle-ci change avec l'angle de vue. Bien sûr, on comprend aisément que tout cet ensemble d'images va de pair avec l'objectif de produire des BRDF physiquement proches des réelles. Néanmoins, il est possible de parvenir à des images photoréalistes, sans pour autant modéliser de façon exacte la photométrie d'une scène. Par contre, une modélisation géométrique trop approximative, nuira fortement aux applications potentielles d'une reconstruction photométrique. Il sera ainsi plus délicat de déplacer le point de vue, car, par exemple, les textures produiront des effets de perspective curieux : un marqueur posé sur le rebord d'un tableau, et qui serait juste approximé par la texture du tableau, empêcherait tout rapprochement de cette zone, celle-ci n'ayant aucune perspective. Nous pensons donc qu'il est raisonnable d'avoir une modélisation géométrique, aussi fine que possible, de l'environnement que nous cherchons à reconstruire.

Souvent, l'inconvénient des techniques utilisant peu d'images, voire une seule, est qu'elles ne savent pas gérer les surfaces autres que diffuses. Bien que certaines proposent des solutions, pour approximer les BRDF non diffuses avec de telles hypothèses, on s'aperçoit vite que ces solutions sont restreintes à des conditions d'utilisation strictes, et limitées à l'analyse d'un seul objet. Ces méthodes répondent à une interrogation différente de celles qui cherchent à modéliser plusieurs surfaces à la fois, l'objectif étant généralement de construire un dispositif le moins onéreux possible (en tout cas beaucoup moins qu'un goniomètre), et capable de mesurer de façon précise le comportement d'une surface face à la lumière. Par ailleurs, le fait de négliger les inter-réflexions, dans quelque méthode que ce soit, constitue un défaut majeur de la technique. En effet, cela limite par conséquent le type d'objets reconstructibles photométriquement, car certains sont très sensibles à ces inter-réflexions (notamment les objets miroirs et les objets blancs par exemple). Nous ne pouvons donc considérer ces techniques comme équivalentes, ni même concurrentielles de celles qui traitent d'environnements complets, où les objets présentent de telles propriétés de réflexion. La seule méthode qui soit d'ailleurs vraiment complète dans ce domaine est celle de Yu et al. [251], qui sait traiter des objets miroirs (encore que les exemples présentés dans l'article soient très petits : il s'agit de sphères de quelques centimètres de diamètre). Cependant, même cette puissante méthode a besoin d'un flot de données colossal, et est soumise à des restrictions

fortes (au moins un éclat pour chaque objet spéculaire doit être visible dans les images).

Comme nous allons le présenter dans le chapitre suivant, notre méthode s'affranchit de tous les problèmes précédemment cités. Les ambitions sont donc de reconstruire photométriquement une scène, en retrouvant les réflectances des surfaces présentes, qu'elles soient diffuses, spéculaires parfaites, *glossy* ou texturées[35], et ce, avec une et une seule image de la scène, sa géométrie 3D, la position de l'observateur et celle des sources de lumière. Nous ne sommes pas non plus contraints sur l'orientation de ces sources de lumière qui peuvent générer un éclairage direct ou indirect de la scène (ce qui n'est pas le cas de beaucoup des techniques précédentes), même en se réfléchissant sur des objets invisibles dans l'image. Bien sûr, tout cela a un prix : nous ne prétendons pas simuler de façon exacte, une représentation physique de la BRDF d'un objet, et notre algorithme peut mettre plusieurs heures avant d'estimer correctement une scène[36]. Par ailleurs, certaines limitations inhérentes à la méthode sont également présentes, et nous expliquerons qu'elles sont principalement liées à l'ensemble très restreint de données, et plus particulièrement l'utilisation d'une image unique.
Notre objectif est donc "seulement", de créer l'illusion visuelle quasi-parfaite, d'une reconstruction totale d'une scène d'intérieur (nous n'avons pas eu l'occasion de tester notre méthode sur une scène extérieure) à partir d'une photographie, mais en conservant le mieux possible le comportement général des surfaces (un miroir ne doit pas être simulé par un plaquage de textures). Il doit également être possible de se déplacer dans la scène tridimensionnelle, tout en rajoutant ou retirant des objets, ou des sources de lumière (nous ne réalisons pas une "simple copie" d'image). Cela peut même s'effectuer en créant une BRDF pour un objet, différente de sa BRDF réelle. Ainsi, certains objets texturés peuvent être approximés par des surfaces purement diffuses, une surface *glossy* peut être reconstruite comme une surface spéculaire simple. L'objectif reste donc une illusion visuelle, mais conservant une réalité tridimensionnelle, pour les applications futures telles que l'interprétation d'images, la vision par ordinateur (par collaboration analyse/synthèse), la réalité augmentée ou la compression de séquences d'images par exemple.

[35] Nous ne traitons pas des objets transparents, bien que l'extension soit tout à fait envisageable. Par ailleurs, la prise en compte d'objets complexes, comme des hypertextures ou des milieux participatifs, n'a pas été étudiée.

[36] En pratique, nous verrons que notre algorithme se révèle, de toute façon, plus rapide que ceux développés par ses concurrents indirects, car ce temps de calcul inclut des propriétés qu'aucune méthode concurrente ne sait traiter. Sans ce type de surfaces (une plaque d'aluminium par exemple), *Phoenix* ne met que quelques minutes pour estimer toutes les réflectances des surfaces présentes dans la scène.

Chapitre 6

Notre Méthode

La fin de la peinture n'est pas tant de convaincre l'esprit que de tromper les yeux grâce à une parfaite imitation du naturel.
Roger de Piles

6.1 Introduction

Ces travaux sont à replacer dans le cadre de la stratégie de collaboration analyse/synthèse, qui a été mise en place dès 1986, par son fondateur A.Gagalowicz. Dès cette époque, il a regroupé au sein de son laboratoire des spécialistes en vision par ordinateur et en synthèse d'images, pour développer des techniques fondées sur l'utilisation de boucles de retour[1] et de modèles pour l'analyse d'images (mais aussi la synthèse). À partir de 1989, il publiait un article [71] dans lequel cette technique était exposée, et qui incluait la méthode d'analyse photométrique d'images réelles par boucle analyse/synthèse.
Une des parties très difficiles à réaliser était la production automatique d'un modèle de la scène vue dans les images. Dès cette époque, une grande partie de l'activité de son laboratoire a été concentrée sur ce point. Des références à ces travaux peuvent également être obtenues dans [21]. Un article [72], ainsi que deux chapitres de livre [217, 28] précisent notamment les techniques pour parvenir à la construction d'un tel modèle. Les premiers résultats de resynthèse d'images réelles ont été publiés en 1994 [167] et 1995 [73] dans le cadre d'une approximation purement lambertienne, utilisant pour le rendu réaliste un algorithme d'illumination globale à base de radiosité. Cette stratégie d'analyse/synthèse s'est développée d'une manière considérable, et la restriction de cette technique à l'analyse photométrique par boucles de retour a pris le nom d'*Image-Based Rendering* dans la communauté scientifique. Un des objectifs de cette thèse a été de réaliser et de mettre au point de manière efficace, pour tous les cas de surfaces, l'analyse photométrique et la reconstruction de scènes réelles. Des publications intermédiaires et relatives à ce sujet peuvent être consultées dans [25, 23, 24, 21, 26]. Nous allons à présent essayer de prouver dans ce chapitre que nous avons bien atteint les objectifs fixés.

6.2 Principe général du rendu inverse

La principale motivation de cette thèse est la génération d'une image de synthèse photoréaliste, depuis une image réelle. Depuis le début de la synthèse d'images, les contributions majeures en synthèse d'images ont développé de nouveaux algorithmes, pour générer des effets plus ou moins sophistiqués et plus ou moins physiques. Les images produites par ces techniques semblaient parfois réalistes, néanmoins il leur manquait toujours une dimension : celle qui permet de les comparer vraiment au monde réel. Bien qu'elle ait été largement investiguée en analyse d'images dans le cadre

[1] A.Gagalowicz parle dans ses différents articles de *feedback*.

de la collaboration analyse/synthèse notamment, comme cela a été présenté au chapitre précédent, cette notion n'a été introduite que récemment en synthèse d'images : il s'agit de l'**Image-Based Rendering**[2].

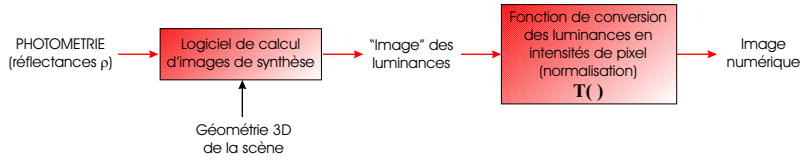

FIG. 6.1 – Processus de création d'une image de synthèse numérique

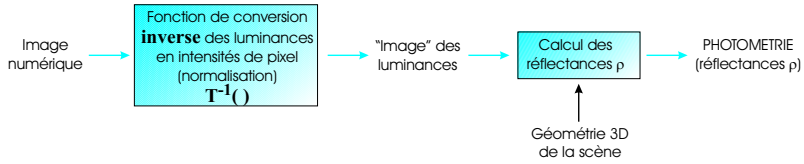

FIG. 6.2 – Processus d'analyse d'une image numérique

D'une manière générale, le rendu réaliste utilise un certain nombre de données, pour calculer une image de synthèse. Ainsi, un algorithme d'illumination globale peut créer une image photo-réaliste, s'il dispose de la géométrie 3D de la scène (comprenant l'observateur et les sources de lumière), les propriétés de réflexion des surfaces (réflectances) et les caractéristiques des sources de lumière (voir figure 6.1). Cette image est d'abord une image de luminances (en $Watt/sr/m^2$), qui est ensuite convertie en intensités de pixels, via une fonction qui peut être linéaire ou non, suivant la méthode que nous choisissons. Typiquement, cette fonction est monotone croissante, et non linéaire pour simuler une caméra standard [222, 46], comme dans notre problème. En *Image-Based Rendering*, nous cherchons à simuler la procédure inverse. Lorsque nous disposons d'une (et une seule) image réelle (ou de synthèse) et de la géométrie 3D de la scène, nous tentons de retrouver les réflectances des surfaces, éventuellement les propriétés des sources de lumière et la fonction de transfert caméra. Cette approche (voir figure 6.2) est similaire à celles déjà développées dans la littérature scientifique sous le nom d'analyse photométrique. C'est pourquoi nous avons introduit une notion de correction itérative et hiérarchique des réflectances estimées, décrite en détail au paragraphe 6.5.1. Cette technique est en fait exprimable comme une boucle analyse-synthèse, qui va d'une part successivement analyser l'image de synthèse regénérée en la comparant à l'image réelle (phase d'analyse pure), pour calculer de nouveaux paramètres de réflectance des surfaces et, d'autre part, regénérer ensuite une nouvelle image de synthèse (phase de rendu réaliste, voir figure 6.3).

L'avantage majeur de notre méthode innovante, est qu'elle est très puissante en raison des très nombreux types de surfaces qu'elle sait traiter (notamment des surfaces spéculaires rugueuses que seuls Yu et al.[251] simulent dans un environnement comparable au nôtre), de la faible quantité d'hypothèses de départ (en particulier, une seule image prise avec une caméra, là où il en faut parfois plus de 100 aux autres algorithmes), et elle reste simple et facilement programmable.
En effet, bien que nous puissions traiter des propriétés de réflectance très différentes (lambertiennes, spéculaires, rugueuses, texturées, etc.), nous avons fait le choix délibéré de créer un algorithme simple. Nous croyons que c'est souvent cette association puissance/simplicité, qui rend une technique utilisable et utilisée : ainsi, le modèle de réflexion de lumière qui soit le plus employé

[2]Plusieurs dénominations sont possibles pour caractériser cette discipline. *Image Based-Rendering, Inverse Rendering, Rerendering* sont les termes anglais généralement employés. *Rendu à Base d'Images Réelles, Rendu Inverse, Regénération d'Images* sont leurs homologues français, bien que nous pensions qu'il soit possible d'en créer d'autres.

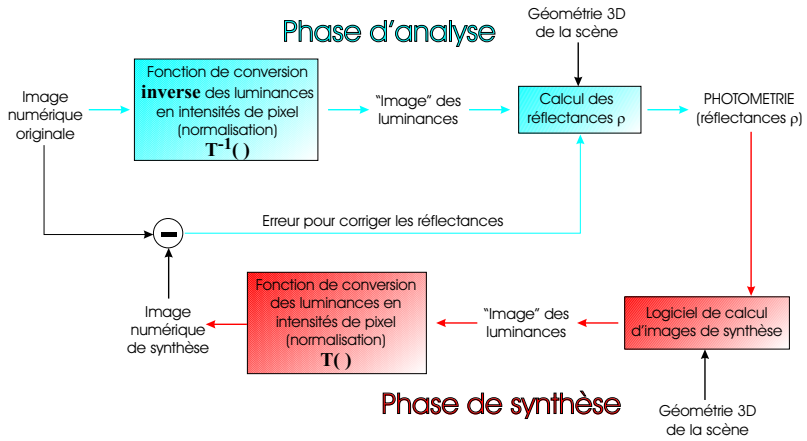

FIG. 6.3 – Processus d'analyse/synthèse d'une image numérique. Dans notre méthode, la fonction $T()$ est modélisée par la fonction de correction γ [222].

aujourd'hui, est toujours le modèle de Phong[159], qui, même s'il ne respecte pas toutes les lois de l'optique, permet de réaliser un grand nombre d'effets, et avec une simplicité maximale.
Bien sûr, nous ne prétendons pas ici réaliser un apport scientifique d'une valeur équivalente à Phong, mais une technique facile à programmer et dont les impacts sont immédiats. Ainsi, avec un ensemble restreint de données, il devient possible de regénérer une image de synthèse tridimensionnelle très proche de l'image réelle de référence. Cette image devient alors plus qu'un simple flux bi-dimensionnel, puisqu'il est possible de modifier la totalité de son contenu, du point de vue géométrique, et photométrique : les applications sont très nombreuses, et permettent, par exemple dans le cas de la synthèse d'images, l'insertion d'objets virtuels se fondant complètement avec leur environnement, l'ajout de nouvelles sources de lumière, la modification des comportements photométriques des surfaces présentes, etc.
Ce nouvel outil peut s'adresser aussi bien au monde de la recherche, pour d'autres extensions futures[3], comme au monde de l'industrie cinématographique, pour la création de mondes virtuels depuis des images réelles, ou l'insertion d'effets spéciaux. Cependant, cette technique repose sur un certain nombre d'hypothèses fondamentales, et sur un type de données particulières, qu'il convient maintenant d'énoncer.

6.3 Problématique et hypothèses de travail

6.3.1 Données et modèle d'illumination

Nos travaux partent de plusieurs données, indispensables à la reconstruction totale de la scène. Tout d'abord, nous devons disposer d'une image (et une seule[4]) de la scène capturée avec n'importe quelle caméra classique. Aucune restriction particulière n'est nécessaire sur le moyen employé pour obtenir l'image réelle[5].

La seconde donnée dont nous avons besoin est la représentation géométrique tridimensionelle totale de la scène, qui est présente sur la photographie à analyser[6]. Cette information est clai-

[3] La reconstuction de certains milieux participatifs (faisceaux de lumière à travers une fenêtre par exemple).
[4] Le fait d'utiliser plusieurs images dans notre algorithme, pour améliorer le résultat dans certains cas, est une extension envisageable, mais pas fondamentale. Ce point est débattu dans le paragraphe 6.5.2.5.
[5] Nous n'avons pas testé notre algorithme sur des images issues d'autres moyens de capture qu'une caméra standard. Néanmoins, à moins que la fonction de capture (qui transforme l'image en information numérique) employée par le scanner (par exemple) ne soit pas approximable par une autre fonction non linéaire, nous ne voyons aucune raison à l'échec potentiel de notre méthode sur de telles données.
[6] Nous présentons, en détails, la façon d'obtenir cette modélisation dans le paragraphe 6.3.2.

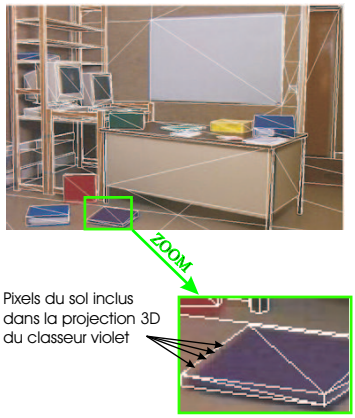

FIG. 6.4 – Problème de calage des données, occasionnant un calcul de réflectance moyenne erroné : la moyenne des intensités de pixel dans la projection du classeur violet, est perturbée par des pixels appartenant au sol marron.

rement la plus pénible à obtenir, car elle nécessite une opération de modélisation 3D qui peut être plus ou moins longue à réaliser. Cependant, plus cette modélisation sera fine, plus le modèle reconstruit sera bon. On notera par ailleurs, qu'une modélisation 3D trop approximative, ou trop détaillée provoque des erreurs de calcul parfois difficiles à traiter, dans l'analyse photométrique. En effet, dans la mesure où nous nous servons des pixels couverts par la reprojection sur l'image réelle des objets ainsi modélisés en 3D, notre algorithme peut être très sensible à ces approximations. Par exemple, sur l'image 6.4, on s'aperçoit que la reprojection en filaire[7] du modèle 3D du classeur violet posé sur le sol sur l'image réelle, comprend plusieurs pixels voisins appartenant au sol. Ainsi, lors de l'analyse photométrique, plusieurs points appartenant à un objet voisin (le sol ici en l'occurrence) risquent d'être pris en compte, pour l'objet violet : ceci est suceptible de perturber fortement les données, et donc les résultats. Il existe aussi un problème lié au raffinement de la modélisation géométrique. Par exemple, les livres sont modélisés par de simples polyèdres, impliquant que les couvertures sont souvent représentées par des faces sans épaisseur. Les images de synthèse réalisées depuis ces données sont par conséquent de qualité moindre en raison de l'approximation géométrique forte, réalisée sur ces objets (les livres paraissent peu réalistes).
Dans le même ordre d'idée, si l'espace recouvert par la reprojection d'un objet dans l'image réelle, n'occupe que très peu de pixels, nous ne disposons alors que de très peu d'informations pour reconstruire sa photométrie, et sommes donc très sensibles au bruit présent dans l'image réelle par exemple. Nous expliquons comment résoudre ces différents problèmes, souvent occultés dans la littérature infographiste, dans le paragraphe 6.4.1. Par ailleurs, ce modèle 3D de la scène est construit sous la forme d'un ensemble de groupes, contenant des objets, constitués de surfaces, elles-mêmes découpées en petits éléments[8]. Par ailleurs, nous entendons par objet, non pas un objet physique au sens réel du terme, mais une surface quelconque, plane ou non. Ainsi, par exemple, un cube sera constitué de 6 objets dans notre logiciel : ceci a pour but d'éviter des problèmes de lissage de Gouraud[9], qui risqueraient d'interpoler des couleurs et de provoquer un effet visuel non désiré (typiquement, si une des faces du cube est ombrée, et une autre face voisine est éclairée, on aura l'impression d'un halo aux sommets que les deux faces ont en commun). Par contre, si nous souhaitons avoir ce lissage, comme dans le cas d'une sphère par exemple, nous choisissons alors que cette sphère soit un objet unique. Cette notion de groupes d'objets est une opération manuelle, qui consiste à réunir les objets possédant théoriquement les mêmes propriétés photométriques. Ceci permet de traiter les objets non visibles directement dans l'image (la face arrière du bureau qui fait face au tableau dans l'image 6.4 par exemple), ou d'étendre le champ d'analyse d'un objet,

[7] Le terme anglais correspondant est *filler*.
[8] Comme dans le modèle de radiosité progressive[38], les éléments (*elements* en anglais) correspondent à des sous-surfaces (*patchs* en anglais), chargées respectivement d'emmagasiner et de réémettre l'énergie incidente.
[9] *Gouraud Shading* en anglais.

dont la surface de reprojection ne couvrirait que très peu de pixels, à plusieurs autres ayant une projection plus grande. Cette opération est très rapide et facile à réaliser.
La troisième donnée dont nous avons besoin est la position 3D de l'observateur, qui peut en fait être retrouvée de façon automatique par des algorithmes d'analyse d'images.
Enfin, nous avons besoin de connaître la position 3D des sources de lumière, ainsi que leur géométrie. A priori, toutes les sources de lumière classiques peuvent être prise en compte, du moment qu'elles émettent de l'énergie de façon uniforme, et que leur émission n'est pas d'un type trop particulier[10].

Le modèle d'illumination et le type de BRDF employés sont ceux proposés par Ward[236]. Ce modèle très largement décrit précédemment (notamment dans le paragraphe 2.4.8), a été retenu pour sa simplicité, sa puissance (il permet de générer des surfaces anisotropes) et sa validité scientifique (Ward a en effet validé son modèle avec des mesures physiques réalisées par un dispositif spécifique).

6.3.2 Construction du modèle 3D associé à une image réelle

Afin de procéder à l'analyse photométrique d'une scène réelle à partir d'une (ou plusieurs[11]) image de cette scène selon notre méthodologie, il convient donc de générer une représentation géométrique de cette scène en terme d'objets constitués de surfaces, associées à des positions de caméras.

La premiere étape consiste en la génération des modèles 3D des objets visibles de la scène. Pour ce faire, nous utilisons des mesures de chaque objet pris séparément, qui servent à construire des modèles polyèdriques simplifiés. Les objets très fins (miroirs, poster) sont modélisés par des boîtes d'épaisseur très faible. La précision des mesures est d'environ 1 mm, ce qui s'avère très suffisant pour un plan large sur une scène d'intérieur (voir figure 6.5).

Dans un second temps, nous procédons au calcul des paramètres régissant la caméra de prise de vue, pour une image donnée. Nous effectuons donc une *calibration* de caméra, qui consiste à obtenir :

- d'une part la position du centre optique de la caméra - rotation et translation -, exprimée dans un référentiel fixe (par construction, ce repère est lié à un objet de la scène choisi comme mire de calibration). Cette position, estimée par un déplacement rigide, constitue les paramètres extrinsèques de la caméra.
- d'autre part, les caractéristiques internes du capteur de prise de vue. Ces paramètres, dits intrinsèques (car ils sont en principe indépendants de la scène observée), permettent d'exprimer la projection d'un point 3D de la scène sous forme de coordonnées pixel. Notre modèle inclut la focale de la caméra, son aspect ratio (défini comme le rapport entre les tailles horizontale et verticale d'un pixel physique), et la projection du centre optique dans le plan image [12].

Les paramètres de calibration permettent, une fois obtenus, de caractériser complètement le processus de projection perspective d'un point de la scène sur le plan image, sous la forme de l'équation matricielle :

[10] Nous n'avons pas testé de sources de lumière "exotiques" n'émettant que dans le rouge par exemple. Néanmoins, comme nous utilisons directement les pixels pour calculer les réflectances des surfaces, il est probable que celles-ci verraient leur BRDF pondérée par un facteur rouge correspondant à la source. Si d'aventure, plusieurs sources de couleurs différentes étaient employées, un algorithme de recouvrement d'émittances de sources pourrait être nécessaire, comme celui proposé par Fournier[66] ou Drettakis[56].

[11] Nous n'avons pas implémenté d'algorithme permettant, pour le moment, d'utiliser plusieurs images dans la reconstruction photométrique, mais cela fait partie des extensions potentielles, pour des surfaces à la fois texturées et anisotropes.

[12] On a choisi ici de négliger les distorsions optiques car on travaille avec des focales relativement longues, supérieures à 60 mm. Le processus de calibration reste toutefois similaire si nous ajoutons les paramètres de distorsion, comme la distorsion radiale par exemple.

FIG. 6.5 – Exemple d'une scène réelle avec le modèle 3D reconstruit et reprojeté dessus

$$\begin{pmatrix} s_X \\ s_Y \\ s \end{pmatrix} = \begin{pmatrix} f & 0 & u_0 \\ 0 & f*r & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{R} & \mathbf{T} \\ \mathbf{0} & 1 \end{pmatrix} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} \tag{6.1}$$

X_w, Y_w et Z_w sont les coordonnées 3D du point projeté dans le repère initial, (sX,sY,s) sont les coordonnées pixels (au facteur projectif s près), f est la focale de la caméra, r le pixel ratio et (u_0, v_0) la projection du centre optique.

Pour obtenir ces paramètres, nous choisissons comme objet de calibration de référence (mire) un des objets de la scène. De préférence, cet objet doit occuper un volume englobant maximum, et sa géométrie doit être connue avec le plus de précision possible (ces deux conditions sont en général antinomiques). Dans notre cas (observation de scènes de bureau), c'est le bureau qui sert de mire[13]. On repère manuellement [14] dans l'image, les coordonnées des projections de points caractéristiques de la mire (dans notre cas, coins du bureau, pieds, etc.), que l'on associe à leurs cordonnées 3D. Nous obtenons ainsi autant de contraintes pour l'équation 6.1. En théorie, 5 points sont suffisants pour obtenir l'ensemble des paramètres de prises de vue. Dans la pratique, un minimum de 8 points bien répartis dans l'espace d'observation est raisonnable. La procédure de calibration en elle-même s'appuie sur l'algorithme de Dementhon et Davis [50], qui calcule par approximations linéaires successives, les paramètres extrinsèques de la caméra. Un module de minimisation d'erreur sur les paramètres intrinsèques vient en sur-couche de l'estimation extrinsèque, avec comme critère la distance entre les points projetés (obtenus par 6.1) et les points réels mesurés sur l'image. Cette minimisation est effectuée par la technique du *simplexe* de Nelder et Mead [88, 102]. La procédure totale est très rapide (une seconde sur un processeur MIPS $R12000$) et surtout très fiable.

[13] On peut aussi, si on a le contrôle sur les prises de vue, prendre deux images, une de référence où on ajoute à la scène un objet spécial servant de mire de calibration, et l'autre dépourvue de cet objet, mais avec exactement la même position de caméra.

[14] Il existe des procédures automatiques de détection de lignes ou de points dans les mires de calibration, mais elles sont peu adaptées sur un objet quelconque comme c'est le cas ici.

Finalement, une fois la caméra calibrée, il convient de positionner les objets dans la scène, afin que leur projection corresponde exactement à celle observée dans l'image. Pour cela, nous partons (si cela est possible) d'estimations initiales fournies par des mesures effectuées lors des prises de vue. Puis nous raffinons ces estimations dans un programme interactif[15], afin de compenser d'éventuelles imprécisions dans le processus de calibration. Typiquement, pour une image de résolution 512x512 obtenue avec une focale de 60 mm, il faut positionner un objet situé à δ mètres de la caméra avec une précision de δ mm, pour obtenir une précision image de l'ordre d'un pixel. Cette précision est relativement facile à obtenir pour des objets simples, mais peut s'avérer plus délicate, pour les objets complexes ou non polyèdriques.

Enfin, il convient de noter que la position et les caractéristiques de la source lumineuse ne peuvent être calculées à partir d'une image et sont donc obtenues par des mesures supplémentaires.

6.4 Retrouver les réflectances des surfaces

6.4.1 Différents problèmes

Nous proposons ici de présenter les différents problèmes sous-jacents à notre technique. Bien que nous revenions, par la suite, en détails sur chacun d'entre eux, il nous paraît important de préciser certains phénomènes, souvent occultés dans la littérature, ou de clarifier certaines idées préconçues sur d'éventuelles simplifications.

Le premier problème est la mise en correspondance des données tridimensionnelles avec l'image numérique. Comme nous travaillons sur les reprojections des objets 3D dans les images, il est fréquent que l'analyse des pixels contenus par cet objet, soit biaisée par un objet voisin (un calage approximatif en fait), ou par la caméra elle-même. Celle-ci peut en effet introduire du *bruit* dans l'image, et ainsi perturber les couleurs de l'image.
Un algorithme de filtrage est donc indispensable, sinon la méthode qui consiste à moyenner les pixels d'une zone, sur laquelle se projette approximativement le modèle 3D, pour calculer une réflectance, est nécessairement fausse (les réflectances moyennes vers lesquelles on cherche à faire converger l'algorithme, représentent alors la moyenne des intensités de pixel de plusieurs objets différents). La technique de filtrage n'est pas véritablement novatrice[16], mais nous croyons important que le lecteur sache qu'elle est indispensable au bon fonctionnnement de notre algorithme de recouvrement de réflectances. Par ailleurs, elle permet aussi de prendre en compte les inter-réflexions diffuses fortes, qui peuvent générer des tâches de couleur diffuses[17] (voir image 6.6). Bien que cela ne soit pas mentionné dans les articles de Fournier[66] ou Drettakis[56], leurs techniques pourraient également être sensibles à ce genre de phénomène. Cependant, le type de scènes qu'ils emploient sont peu sujettes à ce problème, car les surfaces à analyser sont grandes (elles contiennent beaucoup de pixels), mais surtout elles subissent peu d'inter-réflexions diffuses. Ainsi, par exemple, si une surface texturée blanche était rougie par un objet s'en approchant, un moyennage brutal des pixels recouverts par la surface blanche, provoquerait la création d'une réflectance complètement erronée. C'est une des raisons qui nous ont poussé à développer une méthode itérative de correction des réflectances. Par ailleurs, même dans le cas d'une technique itérative, l'objectif est souvent de caler les moyennes des pixels[18] appartenant aux surfaces dans l'image regénérée, sur celles de l'image de référence. Le calcul de ces moyennes est donc primordial et doit être le plus représentatif possible. Ce choix des moyennes réside principalement dans le fait que des calages photométriques sur les minima sont trop sensibles aux ombres, tandis que des calages

[15] Nous avons utiliser *Maya* d'Alias|Wavefront pour placer les objets interactivement dans la scène.
[16] Nous la décrirons néanmoins au paragraphe 6.4.2.
[17] Déjà précisé précédemment, ce phénomène est bien connu en radiosité et porte le nom de *color bleeding* en anglais. Nous utiliserons quelques fois, ce terme anglais dans les pages suivantes.
[18] Nous verrons plus loin que ce calage peut parfois nécessiter une analyse plus fine que celle de la moyenne des pixels de la région d'intérêt, notamment pour les surfaces spéculaires.

sur les maxima sont trop sensibles à l'illumination directe.

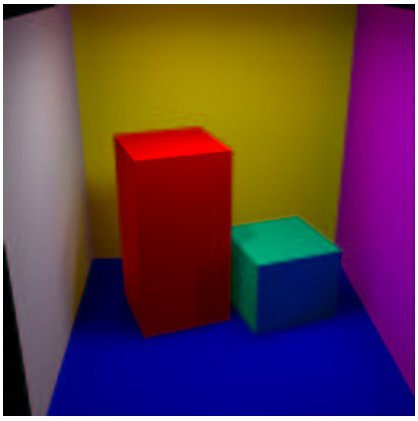

FIG. 6.6 – Phénomènes multiples de *Color Bleeding* : la face avant du petit cube vert est colorée par le sol, le mur gauche est jauni par le mur du fond.

Le second problème important est celui de la taille de la projection des objets 3D, et de leur visibilité dans l'image de référence. Ainsi, si les objets modélisés sont très petits dans la photographie, leur projection dans l'image représente alors peu de pixels. Le processus de moyennage des intensités de pixels dans ces zones, est donc extrêmement sensible à toutes les valeurs s'y trouvant. Nous pouvons, par conséquent, calculer des réflectances qui vont tout de même converger vers la réflectance moyenne estimée dans l'image d'origine, mais sans jamais obtenir la même couleur que l'image d'origine. Ce problème est souvent résolu par un filtrage adéquat, comme celui utilisé précédemment, mais il nécessite parfois l'intervention de l'utilisateur. Cette intervention a lieu au moment de la construction de la géométrie, lorsque l'utilisateur groupe les objets ayant théoriquement les mêmes propriétés photométriques. Ce processus reste très facile à réaliser, et suppose qu'au moins un objet du groupe est directement visible dans l'image. Nous pouvons ainsi tout de même estimer les réflectances des objets très petits, et donc mal représentés par leurs pixels, les pixels des autres objets du même groupe permettant de calculer cette réflectance de façon plus précise.
De plus, grâce à cette notion de groupe, il devient possible de créer une nouvelle vue (depuis l'image réelle 6.4) et de regarder des objets, ou des parties d'objets, invisibles directement auparavant. L'image droite de 6.7 montre un exemple de modification de point de vue, après estimation des réflectances : on remarque que l'on voit certaines zones invisibles dans l'image d'origine, comme le bord droit du parallélépipède vert, ou la plinthe et le mur, qui sont derrière le bureau, etc.

Enfin, le troisième problème est quelque peu différent des précédents. Il ne s'agit pas d'un traitement particulier à ajouter, mais plutôt de briser une idée séduisante, mais irréalisable. Comme nous cherchons des réflectances diffuses de surfaces dans certains cas (supposons que notre image ne contienne que des surfaces parfaitement diffuses, comme au paragraphe 6.5.2.1), une approche naïve pourrait consister à se dire que l'équation de radiosité est directement inversible. En effet, nous pouvons connaître les radiosités des surfaces (extraites depuis l'image) et les facteurs de forme sont directement calculables, puisque nous disposons de la géométrie. Néanmoins, le problème n'est pas aussi simple et ne peut être réduit à cette inversion, avec le peu de données que nous avons : s'il est vrai que nous pouvons calculer les facteurs de forme grâce à la connaissance de la géométrie, il est bien plus difficile de connaître les radiosités de toutes les surfaces. Ainsi, dans nos scènes réelles (et ce ne fut pas voulu), une des contributions énergétiques majeures provient de la lumière réémise indirectement par le plafond de la pièce, qui n'est pas visible dans l'image de référence (sa radiosité n'est donc pas estimable). Yu et al.[251, 252] mentionnent pourtant que le cas diffus est "trivial", car l'inversion de cette équation permet d'en déduire directement les réflectances. Mais comme nous n'avons qu'une et une seule image, cela n'est pas possible (sans

FIG. 6.7 – **À gauche** : Image de référence prise avec une caméra. **À droite** : Nouvelle vue générée depuis l'image d'origine (à gauche) : on constate que l'on voit des parties invisibles directement dans l'image comme la plinthe derrière le bureau par exemple. Ceci est rendu possible par la notion de groupe, qui rassemble les objets ayant les mêmes propriétés photométriques (ici la plinthe sur le mur gauche de l'image réelle gauche a été utilisée).

compter que nous ne pouvons estimer directement les radiosités, puisqu'il faut un minimum de 3 images à Debevec[46] pour retrouver la fonction de transfert caméra), car l'équation dispose d'un nombre important d'inconnues, et qui ne sont pas approximables.
Par ailleurs, il est rare que nous ayons besoin d'émettre la totalité de l'énergie présente dans la scène, pour calculer une image photoréaliste. Par contre, pour résoudre directement l'équation de radiosité inverse, il est indispensable de calculer toutes les réflectances des surfaces, ce qui nécessite l'estimation totale des facteurs de forme entre toutes les surfaces de la scène. Le temps de calcul d'une telle opération est prohibitif, et une telle quantité de données n'est pas non plus stockable (on ne peut donc pas utiliser la relation de réciprocité des facteurs de forme entre deux surfaces pour accélérer les calculs).

6.4.2 Extraire les intensités de pixels

Comme nous l'avons expliqué précédemment, la méthode de moyennage des pixels des zones nécessite un filtrage, pour éviter de biaiser cette moyenne. La première étape consiste à faire correspondre les pixels par zone avec chacune des surfaces.
Ceci s'effectue en deux temps. Tout d'abord, plusieurs cartes d'index sont calculées : elles permettent de connaître les objets se projetant sur l'écran, et d'identifier directement les pixels de l'image réelle leur appartenant. Chacun de ces index représente donc soit un numéro de groupe, soit un numéro d'objet, soit un numéro de patch, suivant le type de moyennage que nous souhaitons effectuer. En effet, il est parfois nécessaire de calculer, non pas une réflectance pour tout le groupe, mais une réflectance par patch[19] . Ces cartes d'index sont obtenues en procédant à une étape d'élimination des parties cachées, facilement réalisable puisque nous disposons à la fois de la géométrie 3D, et des informations propres de la caméra. Dans notre logiciel *Phoenix*, ce calcul est réalisé par OpenGL avec le hardware dédié des SGI, et en utilisant l'*off-screen rendering* décrit au chapitre 4, paragraphe 4.2.2.1.2. Cette extraction des pixels est illustrée par le schéma 6.8 et l'algorithme 16.
La seconde étape consiste à calculer les contours des objets (que nous avons automatiquement grâce aux cartes d'index), et à épaissir ces contours de 2 à 8 pixels, suivant le niveau de filtrage que nous souhaitons obtenir. On remarquera dans l'image en haut à droite du schéma 6.8, que certains objets n'ont pas subi cet épaississement. Il s'agit en fait d'objets dont l'épaisseur elle-même,

[19] Typiquement, lorsqu'un objet est texturé, nous devons calculer des réflectances pour chacun de ses patchs, suivant la méthode de rendu de textures par radiosité, proposée par Cohen et al.[37].

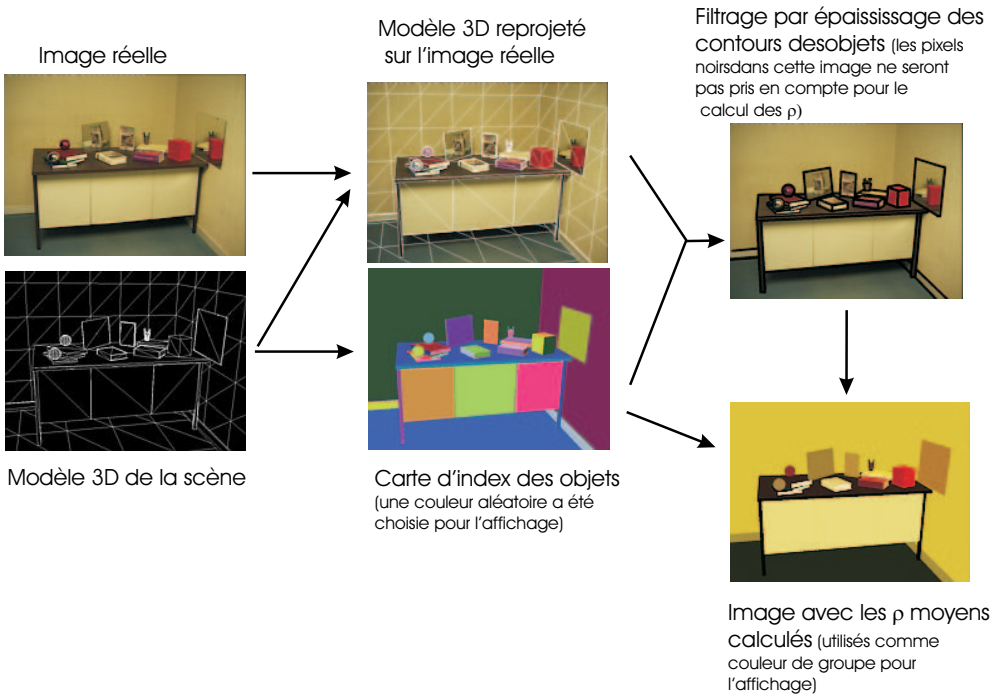

FIG. 6.8 – Processus d'extraction des pixels par zone, pour le moyennage et l'estimation des ρ_d en première approximation.

est trop faible pour supporter un tel traitement : il en résulterait, en effet, l'éradication totale du peu de pixels couvrant la zone, empêchant ainsi d'estimer la réflectance de la surface (image 6.9). Néanmoins, il est parfois possible de négliger tout de même ces objets, s'ils font partie d'un groupe où plusieurs pixels peuvent être utilisés pour analyser les réflectances. Ainsi, dans l'image 6.9, même si le rebord vertical du panneau frontal disparaît, sa réflectance peut encore être analysée, puisqu'il appartient au même groupe que le panneau lui-même, qui contient beaucoup de pixels visibles[20].

On remarque que le filtrage est réalisé sur les contours des objets, et pas sur celui des groupes. Il est en effet parfois préférable de laisser l'algorithme séparer lui-même certains objets de leur groupe, pour leur en créer un, propre. Ainsi, lorsqu'un objet appartenant à un groupe, est soumis à de fortes inter-réflexions diffuses, tandis que les autres ne sont pas perturbés, la méthode de moyennage par groupe va être biaisée sur tous les objets du groupe, qui subiront également ces inter-réflexions. Par exemple, un cube diffus uniformément blanc, dont l'un des côtés (objet en fait) serait rougi par une surface posée à proximité, aurait tendance à voir sa réflectance légèrement rouge propagée à tous les autres objets du groupe, auquel il appartient. Dans ce cas, les réflectances sur les objets diffèrant[21] d'un objet à l'autre, l'algorithme créera un groupe spécifique pour ce côté *perturbateur*. Bien sûr, cela contredit quelque peu la physique du cube lui-même, puisque dans la réalité, on peut raisonnablement penser que la réflectance du cube blanc est la même sur tous ses côtés : ici nous créerons un cube dont 5 côtés auront les mêmes réflectances (blanches), tandis que le sixième sera légèrement rouge. Ceci ne constitue pas véritablement un problème, puisque, comme nous l'avons expliqué plus haut, notre objectif n'est pas une représentation physique parfaite de la réalité, mais une illusion visuelle. Cependant, nous avons mis au

[20] En pratique, il est tout de même indispensable de procéder à ce test, car certains petits objets (comme les deux marqueurs dans le gobelet au fond à droite de l'image) constituent à eux-seuls un groupe, leur réflectance étant unique.

[21] Elles dépassent un seuil fixé initial.

Algorithme 16 Pseudo-Algorithme général d'extraction des pixels et de calcul de réflectance

```
Activer Buffer d'index OpenGL de la même résolution que l'image réelle ;
//Calcul d'un buffer d'index
Pour Tous les objets 3D Faire
  Couleur objet OpenGL = numéro objet (index) ;
  Eliminer les parties cachées par le Z-Buffer d'OpenGL et calculer une image (Buffer)
  où les index sont à la place des couleurs ;
Fin
Lire le Buffer précédent, contenant des index d'objets
Pour Tous les objets 3D j Faire
  //Initalisation du nombre de pixels appartenant a chaque objet
  objet3D[j].nbpixels = 0 ;
Fin
//Moyennage des zones de pixels pour calculer un ρd par objet
Pour Tous les pixels i de l'image réelle Faire
  Si Filtrage activé Alors
    objet3D[Buffer[i]].ρd += T^-1(Filtrage(pixel i)) ;
  Sinon
    objet3D[Buffer[i]].ρd += T^-1(pixel i) ;
  Fin si
  objet3D[Buffer[i]].nbpixels++ ;
Fin
Pour Tous les objets 3D j Faire
  objet3D[j].ρd / =objet3D[j].nbpixels ;
Fin
```

point une seconde technique de filtrage décrite plus loin, qui évite ce découpage d'objet et qui peut être activée suivant le désir de l'utilisateur (si celui-ci désire garder intacte la structure de la scène 3D).
Par ailleurs, la fonction appelée $T()$ de ce premier algorithme de filtrage est une fonction qui permet de convertir les luminances en intensités de pixel. Cette fonction n'est pas connue pour notre caméra[22], et comme nous ne disposons que d'une seule image, la méthode de Debevec[46] ne peut s'appliquer. Nous avons donc décidé d'approximer cette fonction de transfert, comme une fonction de correction γ, avec une valeur arbitraire initiale de γ égale à 2.2, suivant la proposition faite par Tumblin et al.[222]. Lorsque l'algorithme de recouvrement de réflectances a convergé, nous réestimons le paramètre γ en minimisant l'erreur totale entre l'image réelle et l'image de synthèse (le détail de cette technique est décrit au paragraphe 6.6).

Nous avons également développé un second algorithme de filtrage qui conserve la topologie originale du groupe, sans jamais créer de nouveaux groupes ni même séparer les objets, et qui permet, en plus, de s'affranchir du problème des ombres[23] ou des inter-réflexions diffuses utilisées pour calculer une réflectance.

Cette seconde méthode de filtrage, appelée *filtrage robuste*, est bien connue en statistiques, puisqu'il s'agit en fait de calculer une approximation plane de la distribution d'intensités de pixels sur une surface. Une fois cette équation estimée, une erreur est calculée pour chaque pixel : il s'agit de la distance entre ce pixel, et le pixel le plus proche sur ce plan. Il devient alors possible, de prendre les 50% de pixels ayant l'erreur la plus faible, pour en calculer la moyenne sur une surface. Contrairement à l'approche précédente, cette technique évite les problèmes d'ombres et d'inter-réflexions diffuses, puisque les *éclats colorés* provoqués par les réflexions diffuses présenteront une

[22] Une sony tri-CCD, DCR-VX1000E.

[23] Typiquement, lorsqu'un objet est ombré, son ombre est prise en compte dans le processus de moyennage des pixels, ce qui peut perturber la réflectance moyenne.

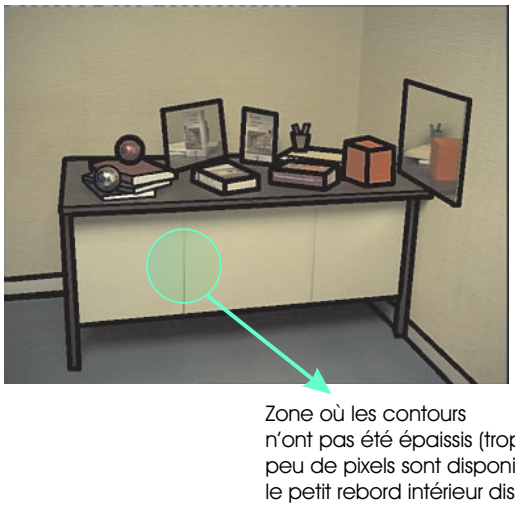

FIG. 6.9 – Problème potentiel d'élimination d'un objet lors du filtrage. Le rebord vertical, à l'intérieur du cercle, disparaîtrait, si les contours verticaux des deux panneaux du bureau étaient épaissis.

erreur élevée[24]. Nous pouvons donc écrire cette approximation plane, comme la minimisation d'une fonction quadratique :
L'équation du plan est de la forme $f(x,y) = ax + by + c$, et nous cherchons à calculer l'erreur :

$$\epsilon(x,y) = \sum_{x,y} (I(x,y) - ax - by - c)^2 \tag{6.2}$$

x, y étant les coordonnées pixel et $I(x,y)$ la fonction d'intensité pixel.

Nous avons donc une fonction quadratique, et pour qu'elle soit minimale, il faut que son gradient soit nul :

$$\begin{aligned}
\frac{d\epsilon}{da} &= \sum_{x,y} (I(x,y) - ax - by - c)(x) = 0 \\
\frac{d\epsilon}{db} &= \sum_{x,y} (I(x,y) - ax - by - c)(y) = 0 \\
\frac{d\epsilon}{dc} &= \sum_{x,y} (I(x,y) - ax - by - c) = 0
\end{aligned}$$

En calculant la matrice d'inertie, ceci revient à résoudre, pour a, b, c (n étant le nombre de pixel de la zone concernée) :

$$\begin{bmatrix} \sum_{x,y} x^2 & \sum_{x,y} xy & \sum_{x,y} x \\ \sum_{x,y} xy & \sum_{x,y} y^2 & \sum_{x,y} y \\ \sum_{x,y} x & \sum_{x,y} y & n \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} \sum_{x,y} x \cdot I(x,y) \\ \sum_{x,y} y \cdot I(x,y) \\ \sum_{x,y} I(x,y) \end{bmatrix} \tag{6.3}$$

Cette résolution s'effectue par inversion directe de la matrice d'inertie 3×3. En remplaçant ensuite a, b, c par les valeurs trouvées ici, dans l'équation 6.2, nous pouvons calculer les erreurs induites pour chaque pixel de la zone d'intérêt.

[24] À moins que cela ne représente la majorité des pixels.

Maintenant que les intensités de pixels ont été extraites de l'image d'origine, et moyennées par groupe d'objets pour estimer une première réflectance, nous pouvons commencer à appliquer notre algorithme de recouvrement de BRDF.

6.5 Algorithme de recouvrement de réflectances

6.5.1 Principe incrémental et hiérarchique

Notre technique est incrémentale et hiérarchique (voir figure 6.10). Incrémentale, car elle va faire évoluer progressivement les réflectances vers leur valeur optimale. Hiérarchique, car cette évolution est prévue pour simuler des surfaces de plus en plus complexes. Elle est donc itérative, et va procéder à des corrections successives des paramètres de réflectance des surfaces, en minimisant l'erreur entre la nouvelle image générée, et l'image réelle : elle se sert en effet des erreurs estimées pour chaque groupe d'objets ayant les mêmes propriétés photométriques, pour corriger leur BRDF. Notre algorithme applique successivement des hypothèses sur le type de surfaces qu'il est en train de traiter, jusqu'à ce que l'erreur entre l'image de synthèse et l'image originale soit en dessous d'un seuil fixé.

Ainsi, dans l'ordre, nous commencerons par une première approximation de l'image, en supposant que toutes les surfaces sont parfaitement diffuses. Si la différence entre l'image regénérée et l'image réelle pour une surface précise, présente des erreurs très importantes (supérieures à un seuil) sur toute sa superficie, elle sera considérée comme spéculaire parfaite, puis spéculaire *colorée*[25] . Si le résultat reste insatisfaisant (l'erreur est toujours supérieure à un seuil fixé), la surface est supposée diffuse et spéculaire simultanément, mais pas rugueuse : le paramètre α de rugosité du modèle de Ward est donc négligé, et la seconde partie de l'équation de Ward sur les éclats spéculaires n'est pas prise en compte. De la même façon qu'à l'itération précédente, si l'image obtenue ne produit pas le résultat attendu sur certaines des surfaces simulées diffuses et spéculaires (la différence entre les deux images présentent encore de grands écarts[26]), alors celles-ci seront calculées comme des surfaces *glossy*, possédant donc à la fois, un ρ_d, un ρ_s, et un facteur d'isotropie α. Eventuellement, la surface est étendue à une BRDF anisotrope, si certaines conditions ne sont pas remplies (trois paramètres au lieu d'un seul (α) sont alors évalués : une direction de stries $\vec{x}$ et deux facteurs d'anisotropie α_x, α_y). Nous verrons au paragraphe 6.5.2.4, comment déterminer cette anisotropie. Néanmoins, si les surfaces estimées par ces BRDF complexes ne réduisent toujours pas suffisamment les erreurs précédentes dans la zone correspondante de l'image réelle, alors nous sommes obligés de les simuler par des textures. Ce traitement texturel est très limitatif, car étant donné que nous n'avons qu'une seule image, il est extrêmement difficile, voire impossible, de créer une combinaison supplémentaire de propriétés de réflexion (c'est-à-dire une surface à la fois *glossy* et texturée par exemple). Nous discuterons de cette situation, au paragraphe 6.5.2.5.

6.5.2 Méthode détaillée par type de surface

6.5.2.1 Surfaces diffuses parfaites

Le cas le plus simple de surfaces à traiter, est celui des surfaces parfaitement diffuses. Lors de la première itération de rendu inverse, tous les objets de la scène sont considérés comme diffus. Une réflectance diffuse est donc estimée pour chaque groupe, comme la moyenne des intensités couvertes par la projection de ce groupe dans l'image réelle. Ce processus est différent de celui de Fournier[66] ou Drettakis[56], car nous travaillons ici sans nous préoccuper des textures. Il est en effet intéressant de noter, que beaucoup de surfaces texturées sont approximables par des surfaces

[25] Nous appelons une surface spéculaire colorée, une surface dont le ρ_s est différent de 0 et de 1 dans l'une des trois composantes R,V,B, mais dont le ρ_d est nul. Il s'agit en fait de surfaces dites *spéculaires non parfaites*.

[26] Nous discuterons plus loin, de la notion de *seuil*, très employée dans notre algorithme, et comment celui-ci influe sur les résultats.

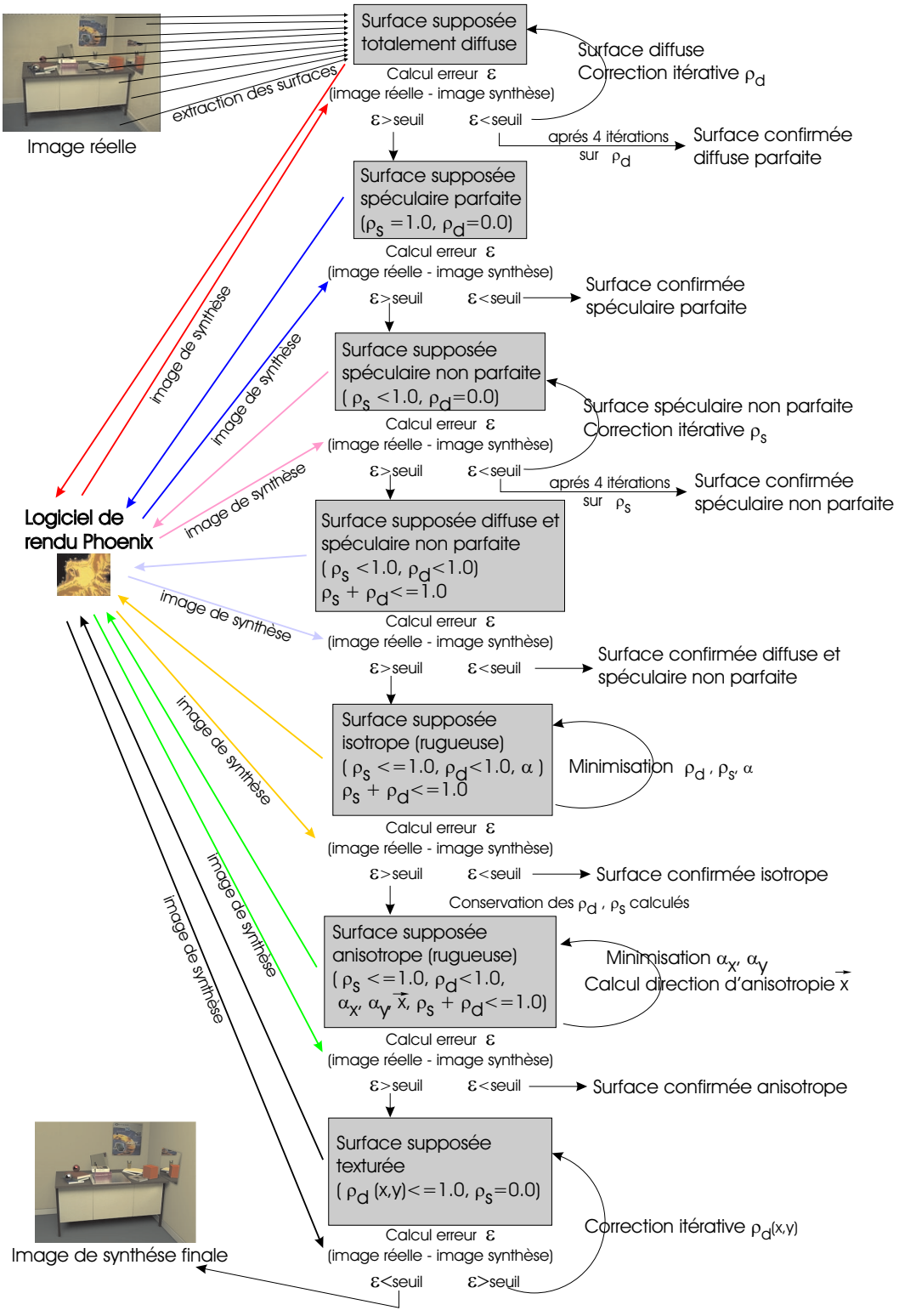

FIG. 6.10 – Algorithme général hiérarchique et itératif de recouvrement de paramètres de réflectance. Chaque surface de la scène est analysée séparément, suivant des hypothèses sur son type de réflectance (diffuse pure, spéculaire pure, etc.). Si l'hypothèse courante s'avère fausse (l'erreur entre l'image réelle et l'image de synthèse est grande), alors la surface est supposée d'un type plus complexe que celui choisi initialement (principe hiérarchique). Si l'hypothèse s'avère exacte, alors la réflectance de la surface est corrigée pour minimiser l'erreur entre les deux images (principe itératif).

purement diffuses. Ainsi, le brunzeel de l'image gauche de la figure 6.11 est constitué de bois fortement texturé, et néanmoins on constate dans l'image droite regénérée, qu'une simulation de ce matériau par une réflectance diffuse crée déjà une très bonne approximation visuelle.

FIG. 6.11 – Exemple d'une reconstruction d'une image réelle (à gauche) par une approximation diffuse de toutes les surfaces (à droite)

Par ailleurs, nous nous distinguons des méthodes précédentes car notre algorithme ne se contente pas d'estimer une réflectance moyenne puisqu'il cherche en effet à la corriger, jusqu'à ce que l'erreur entre les images soit la plus petite possible (suivant un seuil fixé).

Algorithme 17 Pseudo-Algorithme d'initialisation du rendu inverse

```
//Initialisation des réflectances des objets et d'un booléen indiquant
//si on doit créer un groupe pour cet objet (utilisé plus loin dans le spéculaire)
Procédure Initialisation()
  Pour Tous les groupes i Faire
    groupe[i].type = diffus ;
    groupe[i].fonction_erreur = erreur_diffuse() ;
    Calculer groupe[i].ρ_d comme la moyenne des luminances recouvertes par la projection de i ;
  Fin
  Calculer image de synthèse avec Phoenix ;
  Analyse_Reflectances(iter_rerendu, groupe, img_réelle, img_synthétique) ;
Fin Procédure
```

Algorithme 18 Pseudo-Algorithme de rendu inverse pour les surfaces diffuses

```
Procédure Regenere_Scene()
  Tant Que Erreur totale(img_réelle - img_synthèse) > seuil Faire
    Pour Tous les groupes i Faire
      Si groupe[i].type == diffus Alors
        Corriger_Réflectance_Diffuse(groupe[i]) ;
      Fin si
    Fin
    Calculer nouvelle image de synthèse depuis les nouveaux ρ avec Phoenix ;
    Analyse_Reflectances(iter_rerendu, groupe, img_réelle, img_synthétique) ;
  Fin
Fin Procédure
```

Algorithme 19 Pseudo-Algorithme d'analyse de réflectance de surfaces diffuses pures

Procédure Analyse_Reflectances(iter_rerendu, groupe, img_réelle, img_synthétique)
  Pour Tous les groupes i **Faire**
    //Calcul des erreurs pour le groupe i et ses objets j suivant les équations 6.4, 6.6
    groupe[i].fonction_erreur(img_réelle, img_synthétique, groupe) ;
  Fin
Fin Procédure

Algorithme 20 Pseudo-Algorithme d'analyse des surfaces diffuses pures

Procédure Corriger_Réflectance_Diffuse(groupe)
  Pondérer réflectance diffuse $\rho_{d_{groupe}}$ par groupe.ε ; //(équation 6.6)
Fin Procédure

Dans une première étape, nous initialisons les réflectances des surfaces, comme la moyenne des intensités de pixel (inversées par la fonction de transfert caméra), où elles se projettent dans l'image réelle, suivant l'algorithme décrit précédemment. À ce moment précis, l'algorithme de rendu réaliste dispose de toutes les données nécessaires au calcul d'une image de synthèse par illumination globale. *Phoenix* génère donc une image de la scène, qui va pouvoir être comparée à l'image originale. Cette comparaison s'effectue par le calcul d'une erreur par groupe, comme le rapport de la luminance moyenne d'un groupe dans l'image réelle, divisée par la luminance moyenne du même groupe dans l'image de synthèse. Ce ratio d'erreur est utilisé comme facteur de correction sur la réflectance du groupe.

Cette erreur $\widehat{\varepsilon_j}$ pour chaque objet j, s'écrit donc :

$$\widehat{\varepsilon_j} = \frac{\widehat{P_{org_j}}}{\widehat{P_{new_j}}} \tag{6.4}$$

où $\widehat{P_{org_j}}$ est la moyenne des pixels filtrés recouverts par la projection de l'objet j dans l'image réelle, et $\widehat{P_{new_j}}$ est la moyenne des pixels filtrés recouverts par la projection de l'objet j dans l'image synthétique.

Comme la réflectance diffuse ρ_{dj} et la radiosité moyenne $\hat{B}_j$ de l'objet j sont proportionnelles, le réajustement de ρ_{dj} peut s'écrire sous la forme $\frac{\rho_{j\,d_{k+1}}}{\rho_{d_{j_k}}} = \frac{\widehat{B_{org_j}}}{\widehat{B_{j_k}}}$, où k est le nombre d'itérations de rendu inverse, ρ_{dj_k} est la réflectance de l'objet j à l'itération k, et $\widehat{B_{j_k}}$ la radiosité moyenne de l'objet j à l'itération k.
Par ailleurs, dans le but de supprimer le biais inséré par les petits objets sur lesquels l'erreur est souvent importante (ils sont plus sensibles au bruit dans l'image, car le nombre de pixels que leur projection occupe est faible en général), les valeurs de ρ_{di} sont calculées pour les groupes i d'objets j, à chaque nouvelle itération :

$$\rho_{di_{k+1}} = \rho_{di_k} \times \widehat{\varepsilon_i} \tag{6.5}$$

$$\rho_{di_{k+1}} = \rho_{di_k} \times \frac{\sum_{j=1}^{n_i} f(\widehat{\varepsilon_j}) \cdot (\widehat{\varepsilon_j} \times m_j)}{\underbrace{\sum_{j=1}^{n_i} f(\widehat{\varepsilon_j}) \cdot m_j}_{\neq 0}} \tag{6.6}$$

$$\text{et } f(\widehat{\varepsilon_j}) = \begin{cases} 0 & \text{si } \widehat{\varepsilon_j} \geq (1+\lambda) \cdot md \\ 1 & \text{sinon} \end{cases}$$

avec : k, le numéro courant d'itération en rendu inverse
$\widehat{\varepsilon_i}$, l'erreur totale entre l'image réelle et l'image synthétique pour le groupe i
$\widehat{\varepsilon_j}$, l'erreur totale entre l'image réelle et l'image synthétique pour l'objet j
n_i, le nombre d'objets du groupe i
md la médiane (qui consiste à choisir la valeur située au milieu des erreurs classées par ordre croissant)
λ le coefficient de dispersion autorisée
m_j, le nombre de pixels dans la projection de l'objet j

Cette correction itérative des ρ_{dj} crée une amélioration nette des images, et converge en seulement 3 ou 4 itérations, pour retrouver les réflectances purement lambertiennes. Une illustration de cette convergence est fournie par la figure 6.12, ainsi que l'image des erreurs correspondant aux différences entre l'image réelle et l'image de synthèse. L'image réelle est ici, en fait, une image de synthèse générée par *Phoenix*. Des exemples sur des images réelles sont montrés dans les paragraphes suivants, ainsi qu'au chapitre 7. Cette méthode de traitement des surfaces diffuses, et de correction itérative des ρ_d, est décrite par les algorithmes 17, 18, 19 et 20.

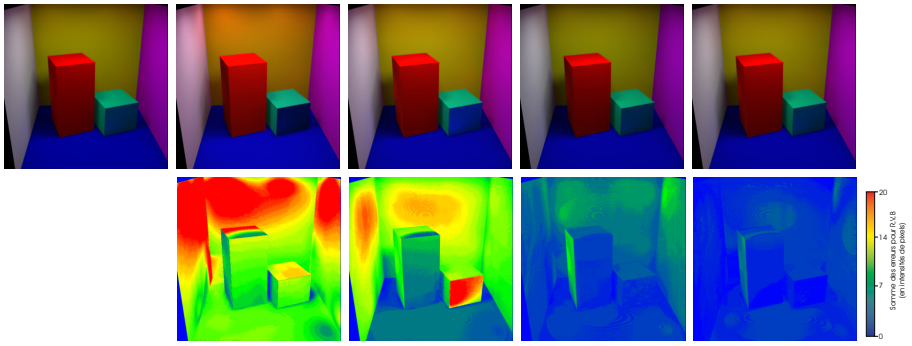

FIG. 6.12 – En haut : dans l'ordre, l'image de référence synthétique (en haut à gauche donc) est simulée par rendu inverse pendant 4 itérations (les 4 images suivantes en haut), et l'image des différences entre ces 4 itérations et l'image de référence est ensuite calculée (en bas). On constate une regression nette de l'erreur, au fur et à mesure des itérations.

6.5.2.2 Surfaces spéculaires

Cette partie expose une méthode totalement novatrice dans le domaine de la regénération d'images depuis une image réelle et a notamment fait l' objet d' un chapitre de livre[21].
À l'exception de Yu et al.[251], qui ont besoin de 150 images pour retrouver les BRDF, aucune autre méthode n'a été proposée pour retrouver les paramètres de réflectance pour des surfaces très spéculaires, comme des miroirs. Bien sûr, nous ne prétendons pas simuler de façon exacte la physique des objets, comme Yu et al., mais notre technique présente l'avantage de pouvoir être programmée très vite. Par ailleurs, au contraire de Drettakis[56] et des autres techniques basées sur l'extraction directe de textures sans analyse de l'image au préalable, nous pouvons regénérer des miroirs de toutes sortes en tenant compte d'éventuels changements de position de l'observateur. Les méthodes à base d'extraction directe de textures simulent les miroirs comme des surfaces

texturées, et le changement de la position d'observation n'influe pas sur l'aspect du miroir, ce qui constitue un défaut majeur.

Lors de la première série de simulations, nous avons considéré que toutes les surfaces de la scène étaient purement lambertiennes. Après quelques itérations pour tenter d'obtenir une réflectance diffuse plus ou moins optimale, nous calculons l'erreur globale produite sur chacune des surfaces de la scène, présentes dans l'image. Lorsque cette erreur est grande (globalement sur toute la surface), nous approximons alors cette surface, non plus comme une surface diffuse parfaite, mais comme une surface spéculaire parfaite (voir pseudo-algorithmes 21, 22, 23, 24), c'est-à-dire avec $\rho_d(r,g,b) = 0$ et $\rho_s(r,g,b) = 1$.

Algorithme 21 Pseudo-Algorithme de rendu inverse pour les surfaces diffuses OU spéculaires pures

```
Procédure Regenere_Scene()
  Tant Que Erreur totale(img_réelle - img_synthèse) > seuil Faire
    Pour Tous les groupes i Faire
      Si groupe[i].type == diffus Alors
        Corriger_Réflectance_Diffuse(groupe[i]) ;
      Fin si
      //Comme il s'agit d'un miroir parfait, nous n'avons rien à
      //modifier sur le ρs.
    Fin
    Calculer image de synthèse depuis les nouveaux ρ avec Phoenix ;
    //On doit procéder à l'analyse des images pour savoir si le groupe
    //doit devenir spéculaire
    Analyse_Reflectances(iter_rerendu, groupe, img_réelle, img_synthétique) ;
  Fin
Fin Procédure
```

Algorithme 22 Pseudo-Algorithme d'analyse du type de surface diffuse ou spéculaire pure

```
Procédure Analyse_Reflectances(iter_rerendu, groupe, img_réelle, img_synthétique)
  Pour Tous les groupes i Faire
    groupe[i].fonction_erreur(img_réelle, img_synthétique, groupe) ;
    Si groupe[i].type == diffus Alors
      Change_Groupe_Diffus_en_Spéculaire(groupe[i], iter_rerendu) ;
    Fin si
  Fin
Fin Procédure
```

Cette erreur globale pour une surface est exprimée comme la somme des valeurs absolues des erreurs, pixel à pixel, entre l'image réelle et l'image regénérée, et s'écrit donc pour les objets et les groupes :

Erreur totale moyenne pour un objet j

$$\widehat{\varepsilon_j} \quad = \quad \frac{1}{m_j} \cdot \sum_{k=1}^{m_j} (|P_{org_k} - P_{new_k}|)$$

Erreur totale moyenne pour un groupe i

Algorithme 23 Fonction modifiant des surfaces diffuses en surfaces spéculaires

```
Procédure Change_Groupe_Diffus_en_Spéculaire(groupe, iter_rerendu)
  //Pendant les 4 premières itérations, on essaiera de corriger ρ_d avant
  //de décider éventuellement de changer le type de la surface en miroir
  Si iter_rerendu ≥ 5 Alors
    Si groupe.ε ≥ seuil_diffus Alors
      groupe.type = miroir ;
      //La ligne suivante est utile pour les miroirs non parfaits
      groupe.iter_miroir = 0 ;
      //On recherche quels sont les objets du groupe qui sont des miroirs
      Rechercher_Groupe_Objet_Spéculaire(groupe) ;
    Fin si
  Fin si
Fin Procédure
```

Algorithme 24 Fonction de recherche des objets spéculaires dans un groupe

```
Procédure Rechercher_Groupe_Objet_Spéculaire(groupe)
  Pour Tous les objets j du groupe Faire
    Si groupe.objet[j].ε > seuil Alors
      //Si l'objet d'un groupe est spéculaire mais pas les autres,
      //il faudra le séparer de son groupe
      groupe.objet[j].separe = TRUE ;
    Fin si
  Fin
  Si groupe.ε > seuil Alors
    //Tout le groupe est spéculaire, donc tous ses objets sont spéculaires
    groupe.type = miroir ;
    groupe.fonction_erreur = erreur_spéculaire() ;
    Propager type groupe aux objets du groupe ;
  Sinon
    Pour Tous les objets j Faire
      //nbvisobj = nombre d'objets projetés visibles pour un groupe, dans l'image réelle
      Si groupe.objet[j].separe == TRUE et groupe.nbvisobj ≠ 1 Alors
        Créer un nouveau groupe séparé, contenant groupe.objet[j] ;
        Nouveau_groupe.type = miroir ;
        Nouveau_groupe.fonction_erreur = erreur_spéculaire() ;
        Retirer groupe.objet[j] de groupe ;
      Fin si
    Fin
  Fin si
Fin Procédure
```

De la même manière que pour les surfaces diffuses (équation 6.6), nous pouvons écrire l'erreur totale moyenne pour un groupe i comme :

$$\widehat{\varepsilon_i} = \frac{\sum_{j=1}^{n_i} f(\widehat{\varepsilon_j}) \cdot (\widehat{\varepsilon_j} \times m_j)}{\underbrace{\sum_{j=1}^{n} f(\widehat{\varepsilon_j}) \cdot m_j}_{\neq 0}}$$

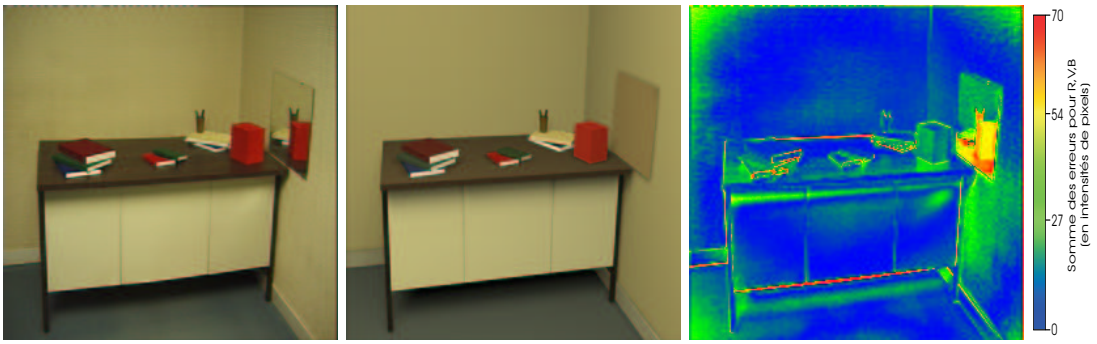

FIG. 6.13 – Comparaison entre l'image réelle (à gauche) et l'image synthétique (au centre) avec approximation diffuse de l'objet posé sur le mur droit. L'image d'erreurs (à droite), montre une erreur grande dans toute cette surface (un pixel rouge signifie qu'il y a 70 niveaux d'écarts en moyenne pour les 3 longueurs d'onde R, V, B entre un pixel de l'image réelle et un pixel de l'image de synthèse).

L'image droite de la figure 6.13 montre la différence entre l'image naturelle (à gauche) et l'image synthétique (au centre), dans le cas où l'objet sur le mur droit a été considéré, en première approximation, comme une surface diffuse parfaite. On constate que l'erreur sur cette surface est très grande, comparativement aux autres objets.

De la même façon, la série d'images de la figure 6.14 montre le même calcul, mais en considérant cette fois que l'objet en question posé sur le mur est un miroir parfait. Après un rendu de l'image sous les nouvelles conditions photométriques, on s'aperçoit que l'erreur sur cette surface a fortement diminué : si elle devient inférieure à un seuil donné, cet objet sera alors simulé comme un miroir parfait.

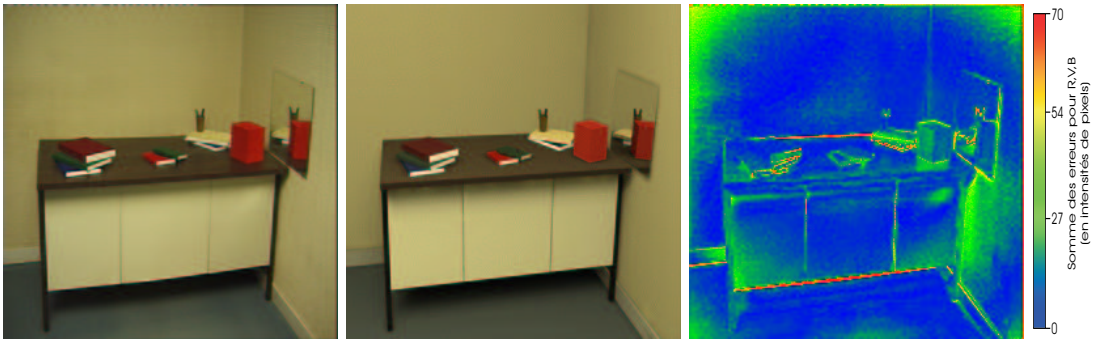

FIG. 6.14 – Comparaison entre l'image réelle (à gauche) et l'image synthétique (au centre) avec simulation spéculaire de l'objet posé sur le mur droit. L'image d'erreurs (droite), montre que l'erreur a énormément diminué (on a gardé la même échelle que pour la figure 6.13), mais le nombre d'itérations en radiosité a été augmenté.

L'algorithme 21 permet de prendre en compte les scènes contenant à la fois des objets parfaitement lambertiens, et des objets parfaitement spéculaires (miroirs). Cependant, il pourrait être intéressant de pouvoir prendre en compte des miroirs non parfaits, comme des boules de Noël colorées par exemple. Nous proposons donc une méthode, fondée exactement sur le même principe que pour les objets purement diffus, mais qui consiste à corriger ρ_s itérativement, en minimisant l'erreur sur la zone spéculaire entre l'image réelle et l'image de synthèse, tandis que ρ_d reste nul. Cette erreur guide, comme pour les surfaces diffuses, la correction des ρ_s. Nous pouvons donc appliquer directement l'équation 6.6 pour pouvoir modifier les ρ_s, ce qui donne :

$$\rho si_{k+1} = \rho si_k \times \widehat{\varepsilon_i} \tag{6.7}$$

$$\rho si_{k+1} = \rho si_k \times \frac{\sum_{j=1}^{n_i} f(\widehat{\varepsilon_j}) \cdot (\widehat{\varepsilon_j} \times m_j)}{\underbrace{\sum_{j=1}^{n_i} f(\widehat{\varepsilon_j}) \cdot m_j}_{\neq 0}} \tag{6.8}$$

$\widehat{\varepsilon_i}$ $\widehat{\varepsilon_j}$ étant les erreurs définies précédemment en 6.6 et 6.4, comme pour les surfaces diffuses. L'algorithme 21 peut donc être modifié, et devenir l'algorithme 25.

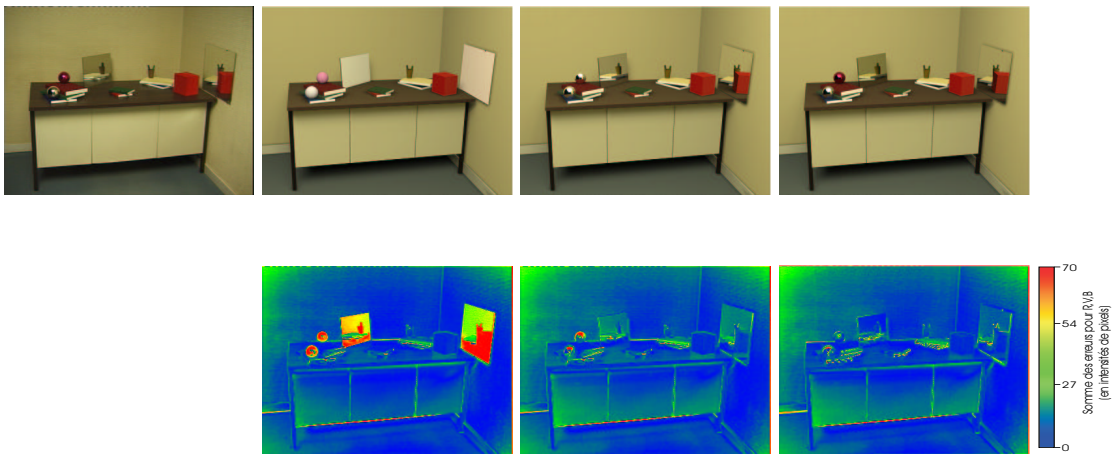

FIG. 6.15 – Simulation de rendu inverse hiérarchique, avec de gauche à droite, en haut, l'image réelle, l'image synthétique avec approximation 100% lambertienne (1ère itération), l'image synthétique avec diffus et spéculaire parfait (5ème itération) et l'image synthétique avec diffus et spéculaire non parfait (7ème itération). En deuxième ligne, les images d'erreurs (Image Synthétique - Image Réelle) correspondantes.

Les images 6.15 montrent une série d'objets estimés, au fur et à mesure des itérations de rerendering, les surfaces pouvant être soit parfaitement diffuses, soit spéculaires parfaites ou non. Les deux sphères sur la gauche de l'image sont des boules de Noël classiques. On constate aussi que le miroir central possède une légère teinte grisâtre. Les différences d'illumination dans ces boules, et particulièrement les éclats spéculaires, proviennent d'une modélisation très approximative de la source de lumière. Nous avons en effet approximé celle-ci par un petit plan, tandis qu'il s'agit d'un cylindre directionnel dans la scène réelle : ceci explique le fait que les éclats spéculaires soient plus petits (la source de lumière réelle étant concentrée sur le plafond) que dans l'image de synthèse, où la source de lumière a tendance à "étendre" son émission.

Algorithme 25 Pseudo-Algorithme d'analyse des surfaces diffuses OU spéculaires non parfaites

```
Procédure Regenere_Scene()
  Tant Que Erreur totale(img_réelle - img_synthèse) > seuil Faire
    Pour Tous les groupes i Faire
      Si groupe[i].type == diffus Alors
        Corriger_Réflectance_Diffuse() ;
      Sinon
        //Nous devons corriger la réflectance des objets spéculaires non parfaits
        Si groupe[i].type == miroir ET groupe[i].iter_miroir ≠ 0 Alors
          Corriger_Reflectance_Speculaire(groupe[i]) ;
        Fin si
      Fin si
    Fin
    Calculer image de synthèse depuis les nouveaux paramètres de BRDF, avec Phoenix
    Analyse_Reflectances(iter_rerendu, groupe, img_réelle, img_synthétique) ;
  Fin
Fin Procédure
```

Algorithme 26 Pseudo-Algorithme de correction des réflectances des objets spéculaires non parfaits

```
Procédure Corriger_Reflectance_Speculaire(groupe)
  //Le groupe est un miroir, il faut corriger son ρ_s
  Modifier ρ_{s_groupe} en utilisant l'équation 6.8 ;
  Propager ρ_{s_groupe} aux objets et aux patchs du groupe ;
  groupe.iter_miroir++ ;
Fin Procédure
```

Algorithme 27 Fonction modifiant une surface spéculaire parfaite en spéculaire non parfaite

```
Procédure Change_Groupe_SpecParfait_en_SpecNonParfait(groupe)
  Si groupe.ε ≥ seuil_spec Alors
    groupe.iter_miroir++ ;
  Fin si
Fin Procédure
```

Algorithme 28 Pseudo-Algorithme de modification du type de surface diffuse ou spéculaire

```
Procédure Analyse_Reflectances(iter_rerendu, groupe, img_réelle, img_synthétique)
  Pour Tous les groupes i Faire
    groupe[i].fonction_erreur(img_réelle, img_synthétique, groupe) ;
    Si groupe[i].type == diffus Alors
      Change_Groupe_Diffus_en_Spéculaire(groupe[i], iter_rerendu) ;
    Sinon
      Si groupe[i].type == miroir ET groupe[i].iter_miroir == 0 Alors
        Change_Groupe_SpecParfait_en_SpecNonParfait(groupe[i]) ;
      Fin si
    Fin si
  Fin
Fin Procédure
```

Lorsqu'un objet a clairement été identifié comme un objet spéculaire, notre technique présente un avantage supplémentaire considérable. En effet, lorsqu'un miroir est détecté, nous pouvons étendre notre méthode d'analyse des réflectances d'un objet à travers ce miroir. Par exemple, dans le miroir droit de l'image gauche de la figure 6.16, une des faces du cube rouge est visible à travers ce miroir, mais ne l'est pas directement dans l'image. Comme l'algorithme de resynthèse a simulé l'objet posé sur le mur comme un miroir parfait, nous pouvons calculer une nouvelle table d'index pour cet objet. Cependant, à la différence de la méthode décrite en 6.4.2, nous nous servons cette fois du lancer de rayons pour obtenir ces index, celui-ci étant bien plus adapté que le Z-Buffer des SGI pour le calcul des réflexions. De la même manière que pour les surfaces visibles directement, nous procédons à un filtrage sur les objets vus dans le miroir (épaississement des contours ou approximation plane), puis à la moyenne des intensités des pixels de la zone projetée dans le miroir, pour estimer la réflectance des surfaces vues indirectement. Dans le cas où le miroir ne serait pas parfait ($\rho_s(R,V,B) \neq 1.0$), cette moyenne est pondérée par le ρ_s du miroir : c'est également pour cette raison que cette analyse n'est effectuée qu'après plusieurs itérations de correction du ρ_s du miroir (sinon le ρ_s du miroir biaiserait la réflectance de l'objet vu indirectement à travers lui).

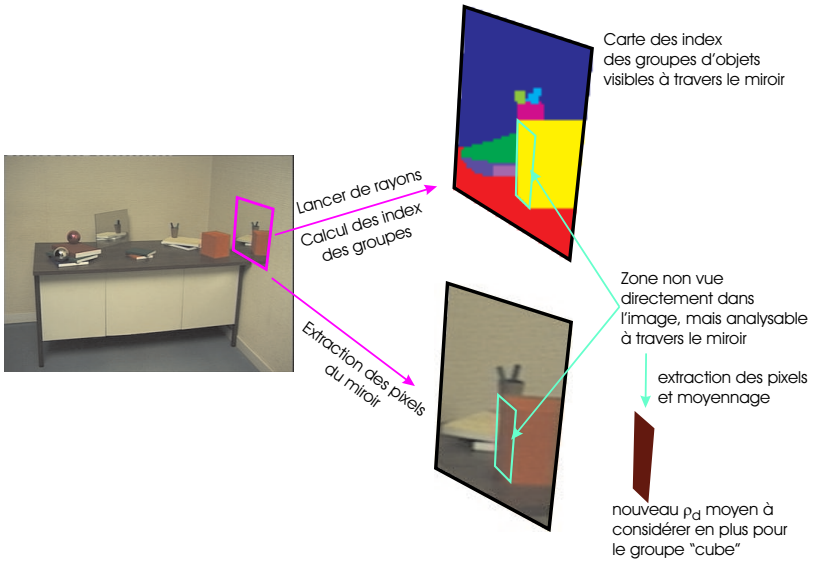

FIG. 6.16 – Exemple de zone, non visible directement dans l'image réelle, mais qui devient analysable indirectement, à travers le miroir fixé contre le mur. Cet objet appartenant au groupe "cube", il sera pris en compte dans la suite des calculs.

Si l'intérêt de cette idée n'est pas évident pour des objets non visibles directement dans l'image, mais appartenant à un groupe dont d'autres objets sont directement visibles, il existe cependant un cas nettement plus intéressant. Ainsi, dans l'image 6.17, la tranche du livre posé verticalement sur la table se reflète dans le miroir. Elle n'appartient à aucun groupe, puisque sa texture est unique[27]. Sa réflectance est déterminée dès que l'objet central a bien été analysé, et de manière stable[28] comme un miroir ; la projection indirecte du livre dans ce miroir est alors estimée, et nous pouvons déduire sa réflectance, des pixels recouverts par cette projection.
Cependant, si une surface dont la réflectance est indéterminée[29] est telle que sa réflexion dans un miroir, occupe plusieurs pixels de ce miroir dans l'image réelle, alors l'analyse de la réflectance de celui-ci doit prendre en compte ce phénomène : ces pixels ayant une réflectance indéterminée, ils

[27] Bien sûr, l'utilisateur peut toujours s'arranger pour déclarer l'appartenance de cet objet à un groupe dont des objets seraient visibles directement, mais ce n'est pas le cas ici.
[28] Ceci signifie que l'objet ne changera plus de propriété de réflectance, et restera de toute façon spéculaire.
[29] Ou "instable" : on ne sait pas encore si elle est diffuse, spéculaire, *glossy*, etc.

ne sont pas utilisés pour le calcul du ρ_s du miroir, car ils biaiseraient fortement le calcul de l'erreur entre l'image réelle et l'image synthétique. De plus, si, après cette élimination, le nombre de pixels restant occupés par le miroir dans l'image est trop faible (donc pas suffisamment représentatif) pour pouvoir calculer la réflectance du miroir, il est procédé à un découpage des textures dans l'image réelle, pour toutes les zones vues indirectement et précédemment indéterminées, dont nous avions écarté les pixels. Plus généralement, ce cas particulier peut arriver dans deux situations : soit deux miroirs se voient l'un l'autre dans l'image réelle (ce n'est pas le cas ici), soit nous estimons deux surfaces en vis-à-vis, comme des miroirs (assez fréquent lorsque la scène contient des objets texturés). Nous proposons d'expliciter supplémentairement ce phénomène, à l'aide d'un exemple.

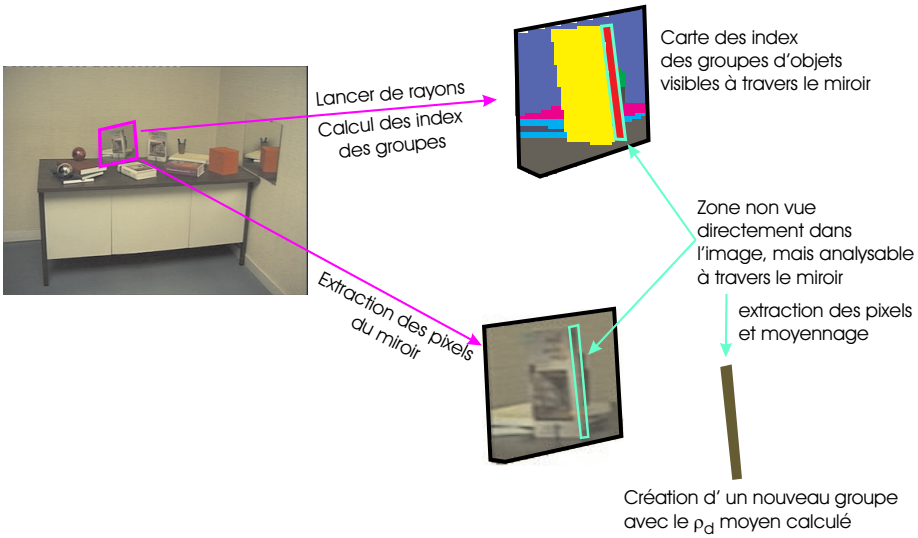

FIG. 6.17 – Exemple de zone, non visible directement dans l'image réelle, mais qui devient analysable indirectement, à travers le miroir fixé contre le mur. L'objet analysé n'appartient à aucun groupe, un nouveau groupe est donc créé, avec le nouveau ρ_d calculé pour la tranche du livre.

Supposons qu'un objet texturé (un livre par exemple) A, soit posé sur un miroir B. Supposons également que l'objet B, l'objet A, et la réflexion de A sur B, soient visibles dans l'image réelle. Lors de l'analyse des réflectances, les objets sont estimés comme parfaitement diffus. À cet instant, rien de particulier ne vient troubler les calculs, si ce n'est que l'algorithme détecte que ces surfaces ont une erreur très grande. Quatre itérations plus tard (nous avons tenté de simuler les objets comme des surfaces lambertiennes pures, en corrigeant ρ_d), l'erreur étant toujours grande sur les deux objets, ils sont considérés au même moment, comme des objets miroirs. Le problème est que, si c'est exact pour l'un (B), c'est faux pour l'autre (A) d'une part, et d'autre part, la réflectance de l'un influe sur l'autre et vice-versa : l'erreur risque donc d'être grande dans B simulé comme un miroir, alors que B est bien un miroir. À cet instant précis, nous introduisons une heuristique pour résoudre ce conflit : dans un premier temps, lors de l'analyse du miroir, l'algorithme évalue s'il existe des objets dont la réflectance est indéterminée (on ne sait pas s'il s'agit d'un objet diffus, spéculaire, *glossy*, etc.), et qui se projettent sur le miroir, en étant visibles dans l'image réelle . Si tel est le cas, la texture de l'objet A qui se reflète dans le miroir B, est directement extraite de B, et reprojetée sur l'objet A, et l'algorithme tente de minimiser l'erreur entre l'image réelle et l'image de synthèse, par lancer de rayons (l'éclairement produit par B sur A n'est pas pris en compte pour éviter de biaiser le calcul de la couleur réfléchie par la texture). Ceci fournit une première erreur. Dans un second temps, nous procédons à l'hypothèse inverse : B est considéré comme un objet texturé, et A comme un miroir. L'algorithme tente à nouveau de minimiser l'erreur entre l'image réelle et l'image de synthèse : ceci nous donne une seconde erreur. Dans

un troisième temps, les deux objets A et B sont considérés comme des miroirs, ce qui permet de calculer une troisième erreur. Nous choisissons alors l'hypothèse la plus probable, c'est-à-dire celle qui contient l'erreur moyenne la plus faible parmi les trois calculées[30]. Ici, c'est sans aucun doute la première hypothèse qui est retenue : B sera simulé comme un miroir, et A sera simulé comme un objet texturé, durant toutes les itérations visant à corriger le ρ_s de B (par la suite, A sera supposé *glossy*, puis de nouveau texturé si l'hypothèse précédente n'a pas fonctionné). Si le nombre de pixels de l'objet texturé A visibles dans le miroir est faible, A sera simulé comme un objet lambertien (c'est typiquement ce qui s'est produit pour l'image 6.17, qui a été regénérée au paragraphe 6.5.2.5). Si cette hypothèse produit toujours une grande erreur entre les deux images, les deux surfaces seront simulées comme n'étant pas des miroirs, mais des surfaces plus complexes faisant partie d'une des catégories décrites dans les paragraphes suivants.

6.5.2.3 Surfaces diffuses et spéculaires sans rugosité

Lorsque l'hypothèse précédente a échoué (surfaces spéculaires non parfaites, avec un ρ_d nul), l'algorithme sélectionne alors les surfaces dont l'erreur est restée importante, pour les considérer comme des surfaces à la fois diffuse et spéculaire, mais non rugueuses (voir algorithmes 29, 30 et 31).

Algorithme 29 Pseudo-Algorithme d'analyse des surfaces diffuses et/ou spéculaires

```
Procédure Regenere_Scene()
  Tant Que Erreur totale(img_réelle - img_synthèse) > seuil Faire
    Pour Tous les groupes i Faire
      Si ... Alors
        ... //Insérer la partie correspondante de la fonction Regenere_Scene() précédente
      Sinon
        //Surface à la fois diffuse et spéculaire ?
        Si groupe[i].type == diffus_spéculaire Alors
          Résoudre Analytiquement ρd et ρs suivant équation 6.11
        Fin si
      Fin si
    Fin
    Calculer image de synthèse depuis les nouveaux paramètres de BRDF, avec Phoenix
    Analyse_Reflectances(iter_rerendu, groupe, img_réelle, img_synthétique) ;
  Fin
Fin Procédure
```

Algorithme 30 Pseudo-Algorithme d'analyse du type de surface diffuse et/ou spéculaire

```
Procédure Analyse_Reflectances(iter_rerendu, groupe, img_réelle, img_synthétique)
  Pour Tous les groupes i Faire
    Si ... Alors
      ... //Insérer la partie correspondante de la fonction Analyse_Reflectances() précédente
    Sinon
      Si groupe[i].type == miroir ET groupe[i].iter_miroir ≠ 0 Alors
        Change_Groupe_SpecNonParfait_en_DiffusSpéculaire(groupe[i]) ;
      Fin si
    Fin si
  Fin
Fin Procédure
```

[30] On remarquera que nous n'établissons pas la quatrième hypothèse, qui pourrait consister à considérer les deux objets comme texturés : ceci produirait sans aucun doute l'erreur la plus faible, mais ne permettrait pas de simuler correctement les miroirs, ni les surfaces *glossy*.

Algorithme 31 Pseudo-Algorithme de modification des surfaces diffuses et spéculaires

```
Procédure Change_Groupe_SpecNonParfait_en_DiffusSpéculaire(groupe)
  //Pendant les itérations 5 à 8, la surface reste un miroir afin que l'on
  //essaie de corriger ρ_s, avant de décider de changer le type de la surface
  Si groupe.iter_miroir ≥ 5 Alors
    Si groupe.ε ≥ seuil_miroir Alors
      groupe.type = diffus_spéculaire ;
      groupe.fonction_erreur = erreur_diffus_spéculaire() ;
    Fin si
  Fin si
Fin Procédure
```

En pratique, dans notre monde réel, de tels objets sont rares. Bien que nous ayons tout de même implémenté une solution dans une telle hypothèse, notre algorithme n'a jamais arrêté son choix définitif sur de telles propriétés de réflexion, pour les objets présents dans nos images réelles. En effet, celui-ci a souvent estimé que l'erreur était soit suffisamment faible pour s'arrêter à une approximation lambertienne, soit réductible avec une hypothèse supplémentaire de rugosité de surface. Néanmoins, notre technique reposant tout de même sur des choix de seuils d'erreurs, l'utilisateur peut s'arranger pour que de telles situations arrivent. Si tel est le cas, la résolution du système est triviale, comme le montrent les équations suivantes.

L'équation d'illumination est ici de la forme :

$$I(r,v,b) \;=\; T(\rho_d \cdot B_d \;+\; \rho_s \cdot B_s) \tag{6.9}$$

avec :
ρ_d la réflectance diffuse du groupe
B_d l'éclairement du pixel
ρ_s la réflectance spéculaire du groupe
B_s la luminance du pixel
$T()$ la fonction de transfert caméra

Nous cherchons donc ρ_d, ρ_s, telle que l'erreur entre l'image réelle et l'image regénérée soit minimale, soit à minimiser :

$$(I_{synth} - I_{org})^2 \;=\; \sum_{i=1}^{nbg} (\rho_d \cdot B_d \;+\; \rho_s \cdot B_s - T^{-1}(I_{org}))^2 \tag{6.10}$$

avec nbg, le nombre de pixels dans la projection du groupe.

Cette minimisation a une solution directe, pour chaque longueur d'onde R, V, B, et de la forme :

$$\begin{pmatrix} \rho_d \\ \rho_s \end{pmatrix} = \begin{pmatrix} \sum\limits_{nbg} B_d \cdot T^{-1}(I_{org}) \\ \sum\limits_{nbg} B_s \cdot T^{-1}(I_{org}) \end{pmatrix} \begin{pmatrix} \sum\limits_{nbg} B_d^2 & \sum\limits_{nbg} B_d \cdot B_s \\ \sum\limits_{nbg} B_d \cdot B_s & \sum\limits_{nbg} B_s^2 \end{pmatrix}^{-1} \tag{6.11}$$

L'image 6.18 montre une exemple de recouvrement de paramètres de réflectance, à la fois diffus et spéculaire, mais sans rugosité. Typiquement, ici, nous nous sommes arrangé pour que le seuil qui sépare le spéculaire non parfait ($\rho_s \leq 1$ et $\rho_d = 0$), des réflectances diffuses et spéculaires sans rugosité, provoque la sélection du modèle de surfaces diffuses et spéculaires sans rugosité.

6.5.2.4 Surfaces à fonction de réflexion complexe (BRDF) ou surfaces rugueuses

Cas des surfaces isotropes

FIG. 6.18 – Simulation de rendu inverse avec à gauche l'image réelle, et à droite l'image de synthèse. Le plateau supérieur du bureau a été simulé comme une surface à la fois diffuse et spéculaire, sans rugosité.

Dans la suite de notre algorithme hiérarchique, nous supposons désormais les surfaces dont l'erreur est restée grande, comme des surfaces diffuses, spéculaires et rugueuses. Dans nos images réelles, nous avons ajouté une surface fortement anisotrope : une plaque d'aluminium[31]. Cette fois-ci, l'algorithme de rendu inverse doit obligatoirement procéder à une minimisation de fonction, car les paramètres sont trop nombreux pour pouvoir être calculés analytiquement, ou corrigés itérativement.

Dans le modèle de Ward[236], qui est donc notre modèle d'illumination et de réflexion de la lumière, nous nous apercevons que la fonction à minimiser dépend de 3 paramètres si nous voulons simuler une surface isotrope (ρ_d, ρ_s et α), et de 5 pour une surface anisotrope[32] (ρ_d, ρ_s, $\hat{x}$ la direction de stries de la surface et α_x, α_y les paramètres de rugosité).
Dans un premier temps, nous allons donc tenter de simuler la surface comme une surface isotrope. Il pourrait paraître plus logique de tenter de simuler directement une surface anisotrope, car l'isotropie n'est en fait qu'un cas particulier de l'anisotropie. Néanmoins, le nombre de paramètres à estimer pour le cas anisotrope est supérieur au cas isotrope, ce qui implique que la minimisation serait plus longue (à complexité équivalente) dans le cas anisotrope. C'est pour cette raison que nous utilisons les paramètres calculés en première approximation par l'hypothèse d'isotropie : en effet, on constate, notamment pour notre plaque d'aluminium, que la fonction d'erreur varie énormément suivant que la surface présente des propriétés de diffusion ($\rho_d \neq 0$) ou non. Ainsi, la figure 6.19 montre l'aspect de la fonction d'erreur avec des valeurs de ρ_d différentes de 0.0, et suivant des valeurs de ρ_s[33] et de α[34] variant respectivement entre $]0.0; 1.0[$ et $]0.001; 0.19[$ avec un pas de 0.018 et 0.1. La figure 6.20 montre la même fonction d'erreur, mais avec $\rho_d = 0.0$ et $\rho_s = 1.0$. On constate que l'aspect des fonctions d'erreurs diffère radicalement dans les deux cas. Ceci nous a donc incité à les traiter séparément, par deux algorithmes de minimisation[35], et à choisir l'hypothèse ($\rho_d = 0.0$ ou $\rho_d \neq 0.0$) qui donne l'erreur la plus petite.

[31] Ce choix fut en fait un pur hasard. Nous recherchions un objet pour notre scène qui soit fortement rugueux. Il s'est trouvé par la suite que cet objet est également anisotrope.

[32] On aurait pu penser que le nombre de paramètres à estimer serait supérieur, car ρ_d, ρ_s et $\hat{x}$ sont des vecteurs. Néanmoins, nous ne traitons pas les surfaces de type fluorescent, pour lesquelles la lumière incidente dans une longueur d'onde peut se réfléchir dans une autre longueur d'onde (un faisceau de lumière "rouge" qui se réémettrait dans le "vert" en atteignant cette surface par exemple). Ceci implique donc que nous pouvons traiter chaque longueur d'onde de façon séparée, en minimisant la fonction d'erreur dans les 3 canaux indépendamment les uns des autres (si cette méthode fournit des valeurs différentes pour ρ_s et ρ_d dans le rouge, le vert et le bleu, le paramètre α dans le cas isotrope ne varie pas d'une longueur d'onde à l'autre). Quant au vecteur $\vec{x}$ pour les surfaces anisotropes, il est nécessairement dans le plan de la facette supposée rugueuse, et peut donc être exprimé en fonction d'un angle de rotation autour de la normale.

[33] L'intervalle de ρ_s a été choisi de sorte que la surface ne peut pas être diffuse parfaite ($\rho_s = 0.0$). Le cas où $\rho_s = 1.0$ est traité séparément (voir figure 6.19).

[34] L'intervalle de α est choisi suivant le modèle de réflexion de Ward, qui précise que lorsque α atteint 0.2, la surface devient parfaitement diffuse, tandis qu'à 0.0 elle est spéculaire parfaite.

[35] On a retenu la méthode du simplexe[88, 102], qui dans les deux cas, converge vers un minimum global, en une cinquantaine d'itérations, soit 1min30s pour ρ_d et ρ_s et environ 1h50min pour α (avec 10 rayons relancés pour chaque rayon primaire touchant la surface isotrope).

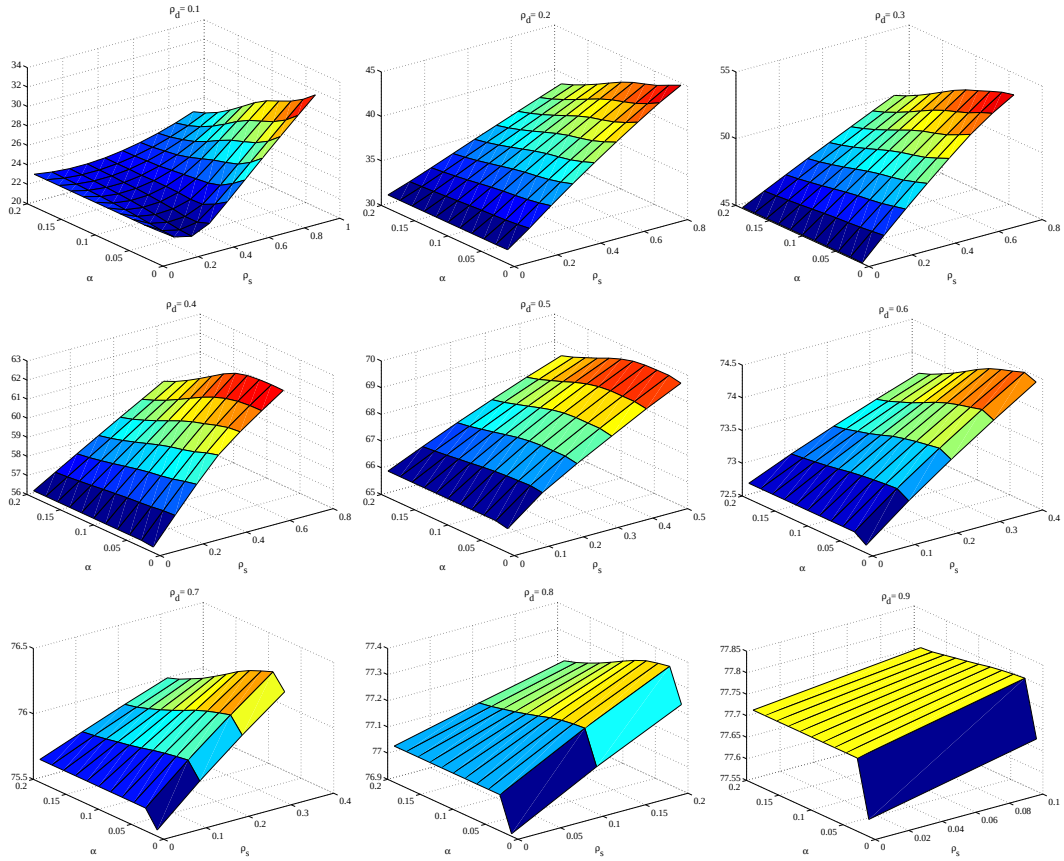

FIG. 6.19 – Fonction d'erreur (Image synthétique - Image réelle), pour une réflectance diffuse ρ_d fixée, et suivant les valeurs d'isotropie α, et de réflectance spéculaire ρ_s.

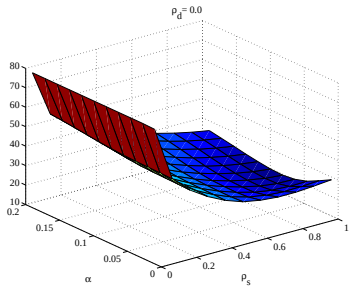

FIG. 6.20 – Fonction d'erreur (image synthétique - image réelle), pour une réflectance diffuse ρ_d nulle, et suivant les valeurs d'isotropie α, et de réflectance spéculaire ρ_s.

Le problème de cette méthode est le temps de calcul, prohibitif pour effectuer plusieurs dizaines de fois le rendu d'une surface rugueuse. Nous n'avons résolu ce problème que partiellement, en introduisant une méthode d'optimisation du tracé de rayons, spécifique à notre problématique : comme d'une image à l'autre les intersections des rayons *primaires* ne changent pas, nous pouvons stocker en mémoire les éléments intersectés pour chaque pixel de la zone d'intérêt[36], et sélectionner les pixels les plus significatifs. Malheureusement, le stockage des rayons secondaires ne peut être réalisé, car la distribution de ceux-ci varient en fonction de α. C'est pourquoi, afin de réduire au maximum le temps de calcul, le calcul d'une surface *glossy* n'est effectué qu'en tracé de rayons (on ne réémettra l'énergie qu'au moment du rendu final de l'image, lorsque toutes les réflectances des surfaces auront été retrouvées).

Si cette hypothèse génère une image de synthèse où l'erreur de la surface supposée rugueuse, par rapport à l'image réelle, est supérieure à 1%[37], alors nous tentons de la simuler de façon anisotrope. La figure 6.21 montre la scène, avec la plaque d'aluminium simulée de façon isotrope. On remarque que l'erreur y est globalement élevée dans la zone d'"étalement" de réflexion spéculaire : la surface sera donc simulée anisotrope.

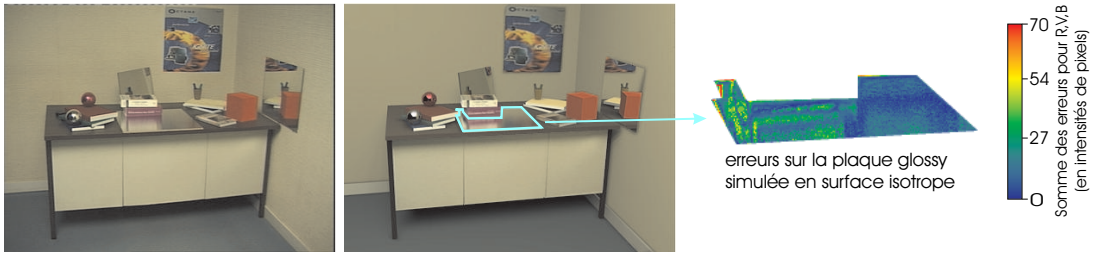

FIG. 6.21 – Approximation de la plaque d'aluminium anisotrope de l'image réelle (à gauche), par une surface isotrope dans l'image de synthèse (au centre). L'erreur entre ces deux images, pour la plaque d'aluminium est visible sur l'image de droite. On constate que cette erreur est grande sur les zones où la réflexion est censée "s'étaler". Les pixels rouges correspondent à une erreur très élevée, mais ne sont pas significatifs, car ils proviennent des bords de l'objet, et sont causés par le calage relativement approximatif du modèle 3D.

Cas des surfaces anisotropes

De l'hypothèse d'isotropie, nous conservons seulement ρ_d et ρ_s : on s'aperçoit en effet en regardant les courbes d'erreur d'isotropie, que ρ_s est décorrélé de α, car la fonction d'erreur est quasiment plane. Nous pouvons donc faire la supposition, que ρ_s et ρ_d ne diffèrent pas du cas isotrope au cas anisotrope.

La première idée "naïve" que nous avons eue pour traiter les surfaces anisotropes, fut de tenter une minimisation directe sur les paramètres d'anisotropie α_x, α_y et $\vec{x}$, en utilisant la méthode du simplexe. Il nous a donc paru indispensable de voir comment se comportait la fonction d'erreur entre les deux images, pour un angle θ de rotation (voir sa définition sur la figure 6.25) du vecteur d'anisotropie $\vec{x}$ (qui est en fait perpendiculaire à la direction d'"étalement" de la réflexion), et en fonction des valeurs de α_x et α_y.

Les figures 6.23 montrent les surfaces d'erreur calculées avec ces paramètres. On constate que plusieurs minima sont présents, mais surtout que si l'on regarde les images de synthèse correspondant à ces valeurs, elles sont visuellement insatisfaisantes : typiquement la barre verticale noire du livre blanc disparaît dans l'image de synthèse, alors qu'on s'aperçoit qu'elle est bien présente dans l'image réelle : si on regarde les 4 images correspondant aux 4 minima les plus importants,

[36] Nous ne réalisons, en effet, ce stockage que pour la surface incriminée, dont nous connaissons parfaitement la projection, grâce aux tables d'index calculées précédemment.

[37] Sur les grandes surfaces simulées isotropes, la différence entre l'image réelle et l'image de synthèse réside principalement dans la zone de "diffusion" de la réflexion spéculaire (zone à partir de laquelle la réflexion de l'objet sur la surface spéculaire devient de plus en plus floue) : comme le nombre de pixels concernés par cet "étalement" des réflexions est faible, nous avons volontairement choisi d'utiliser un seuil très bas, pour forcer, la plupart du temps, l'algorithme à ne pas s'arrêter à l'hypothèse d'isotropie.

FIG. 6.22 – Deux images réelles différentes, comportant la même surface anisotrope

on s'aperçoit qu'elles constituent une trop forte approximation de la surface anisotrope, car trop de détails ont été lissés (figure 6.24). À la vue de toutes ces courbes, et de ces constatations expérimentales, il est évident qu'un algorithme de minimisation, même sophistiqué, ne trouvera pas le minimum le plus intéressant pour une image photoréaliste.

Malgré le peu de pixels de la zone et la méthode même de calcul d'une surface anisotrope en synthèse d'images (voir chapitre 4), qui procède par des lancers de rayons aléatoires dans un cône de spécularité, et qui produit donc des perturbations dans la fonction d'erreur, nous proposons de réduire la minimisation précédente sur α_x, α_y et $\vec{x}$ à seulement α_x et α_y, sachant que nous allons directement calculer la direction d'anisotropie depuis l'image réelle (voir algorithme 32).

Algorithme 32 Pseudo-Algorithme de calcul de direction d'anisotropie

Procédure Calculer_Direction_Anisotropie(groupe, $\vec{x}$)
  Considérer la surface supposée anisotrope comme une surface spéculaire parfaite
  Calculer l'image de synthèse de la zone miroir par lancer de rayons
  Calculer la table des index des surfaces se réfléchissant dans le miroir parfait
  Calculer toutes les distances $d(S,P)$, entre les centres de gravité des objets se reflétant sur le miroir et le centre de gravité du miroir.
  Calculer les aires A des surfaces réfléchies sur le miroir
  Sélectionner la surface pour laquelle $\frac{A}{d(S,P)}$ est le plus grand
  Calculer l'image des différences entre l'image de synthèse précédente et l'image réelle pour la surface trouvée
  Pour Tous les directions d'anisotropie $\vec{x}$ échantillonnées sur la surface supposée anisotrope **Faire**
    Parcourir l'image des différences dans cette direction
    Calculer la moyenne des écarts types des différences dans cette direction
  Fin
  Choisir la direction $\vec{x}$ avec la moyenne la plus faible ($\vec{x}$ est la direction d'anisotropie recherchée)
Fin Procédure

Dans un premier temps, nous visualisons les zones d'étalement des réflexions des objets. Pour ce faire, nous calculons une première image de synthèse en supposant que la surface anisotrope est spéculaire parfaite. Nous estimons alors la différence, pour cette surface, entre l'image réelle et l'image regénérée (voir figure 6.25) avec ce miroir. Dans un second temps, nous calculons pour cette image de synthèse, sur le miroir parfait, la table d'index des objets se réfléchissant dessus. Nous choisissons alors la surface pour laquelle le rapport $\frac{Aire(surface)}{d(S,P)}$ est le plus grand possible ($d(S,P)$ représentant la distance entre le centre de gravité de la surface dont on voit la réflexion sur le miroir, et le centre de gravité du miroir). Ce choix est motivé par le fait que les objets éloignés

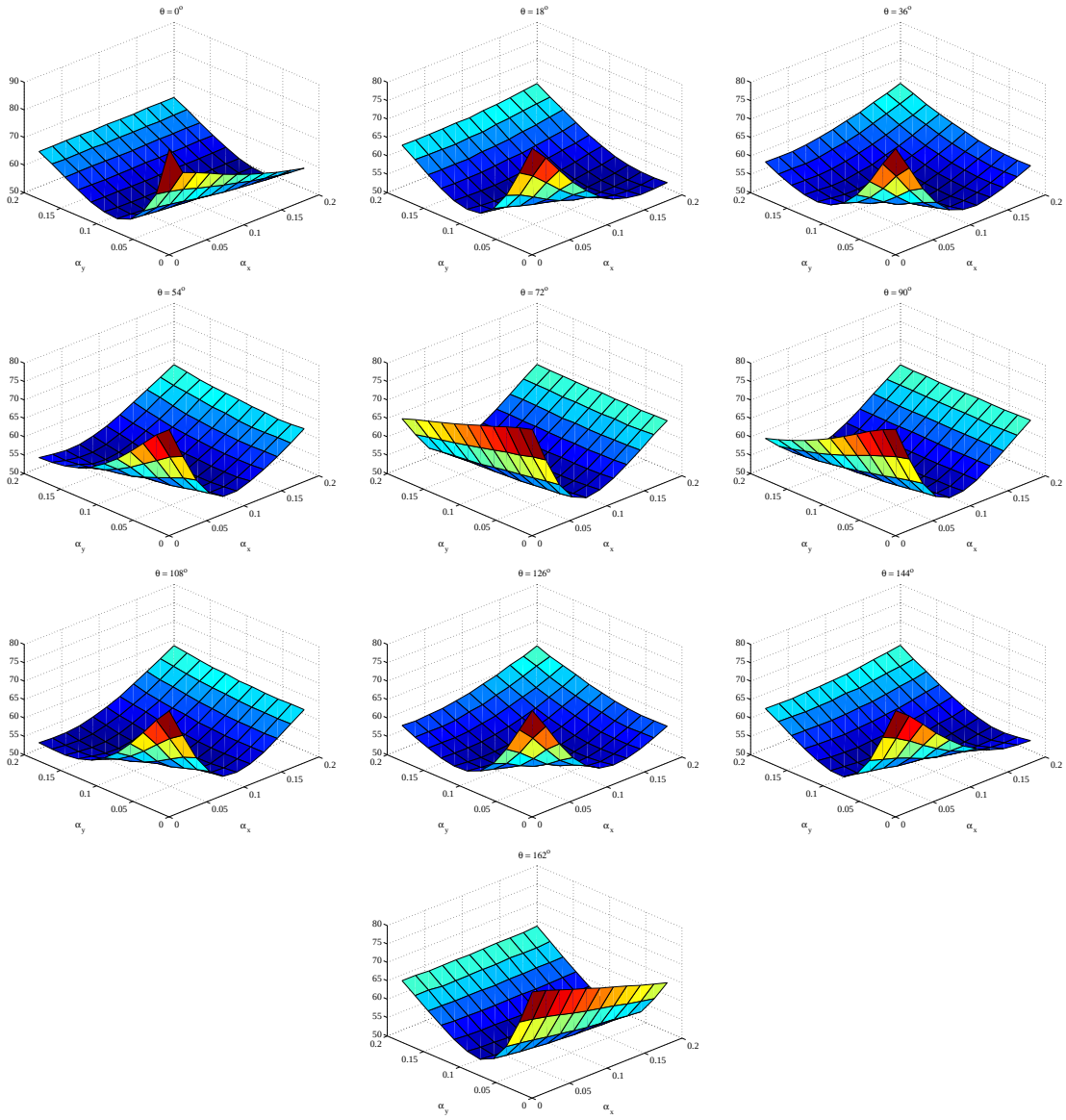

FIG. 6.23 – Fonction d'erreur (Image Synthétique - Image Réelle), pour différentes directions d'anisotropie $\vec{x}$ et suivant des valeurs d'anisotropie α_x, α_y (la réflectance diffuse ρ_d et la réflectance spéculaire ρ_s ayant été calculées par l'étape précédente d'isotropie).

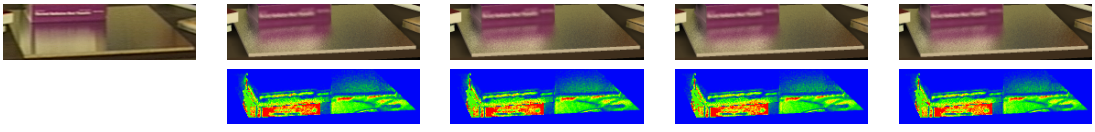

FIG. 6.24 – La première image est l'image de référence (réduite ici à la zone d'intérêt). Les 4 images suivantes (par ordre croissant de gauche à droite) ont une erreur minimale, pour la fonction d'erreur anisotrope 6.23. On constate que toutes ces images sont visuellement éloignées de l'image réelle (la ligne noire verticale de la tranche texturée du livre blanc a disparu par exemple). Ceci est confirmé par les images de différence correspondantes, en deuxième ligne (l'échelle des erreurs est la même que pour l'image 6.21).

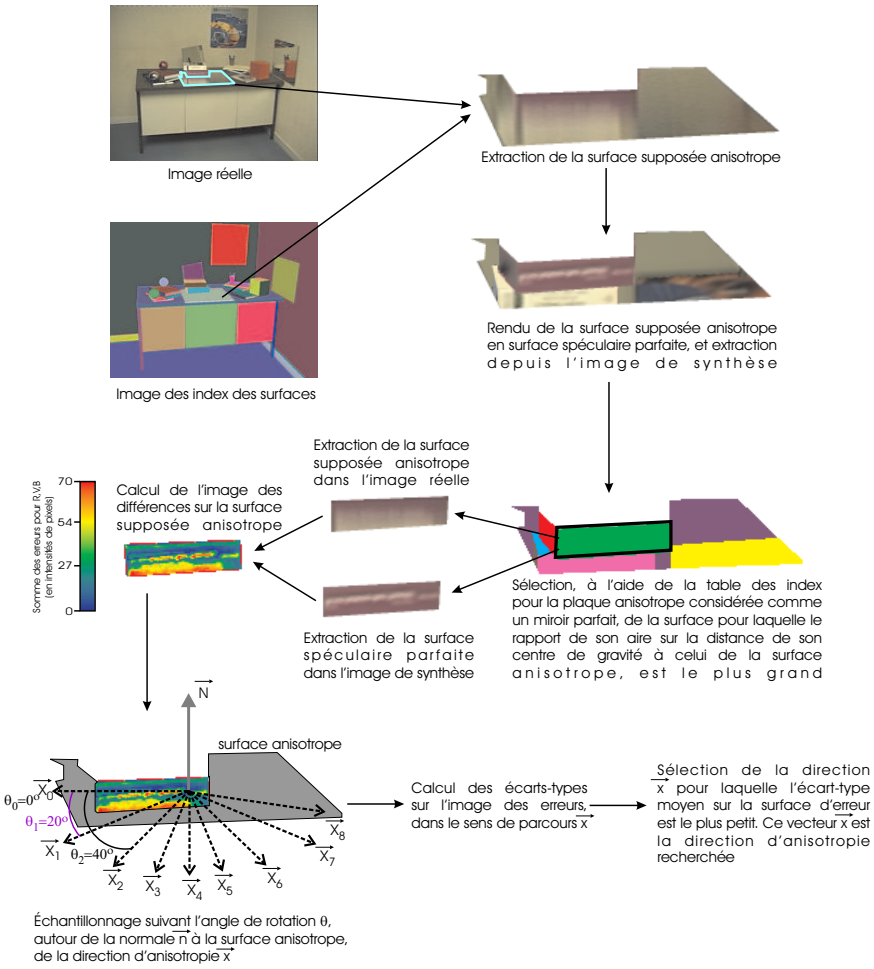

FIG. 6.25 – Processus de calcul de la direction d'anisotropie $\vec{x}$ sur une surface *glossy*

de la surface anisotrope, ont une réflexion trop étalée (et trop bruitée) pour pouvoir être discriminante sur la direction d'anisotropie (les rayons spéculaires relancés depuis la surface anisotrope intersectent l'objet, en des points très éloignés les uns des autres). Plus l'objet est proche, plus sa projection contiendra de pixels, et plus sa réflexion sur une surface anisotrope sera analysable pour en déduire une direction d'anisotropie.

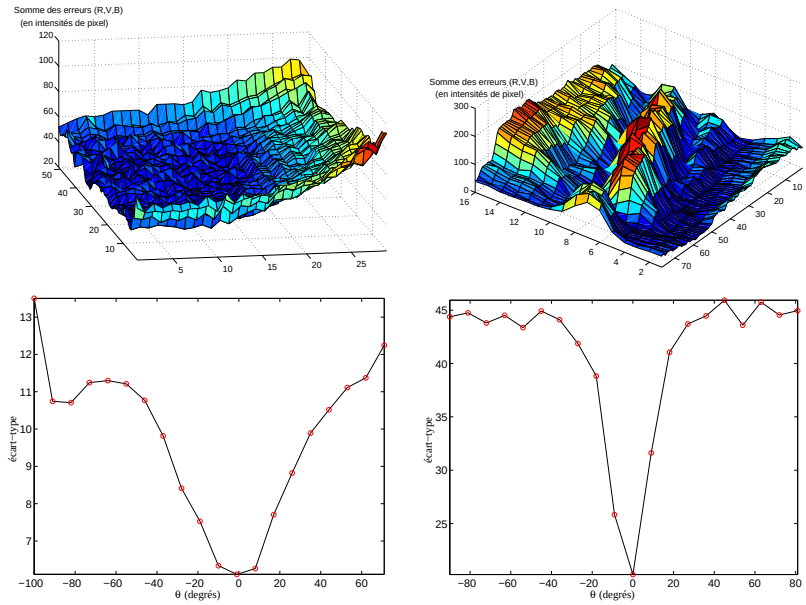

FIG. 6.26 – En haut à gauche : la surface 3D représente l'image des différences entre la réflexion virtuelle du cube sur la plaque supposée spéculaire parfaite et la réflexion du cube sur la même plaque (anisotrope) dans l'image réelle. En haut à droite, la même surface 3D a été calculée sur la réflexion du livre violet de l'image réelle droite de la figure 6.22. En bas, de gauche à droite, les courbes 2D des moyennes des écarts types, en fonction de la direction d'anisotropie (sens de parcours de l'image des erreurs).

Dans un troisième temps, nous échantillonnons la direction d'anisotropie, en créant des vecteurs $\vec{x}$, que nous considérons tour à tour comme les directions d'anisotropie, et suivant des angles de rotation autour de la normale à la surface supposée anisotrope. Pour chacune de ces directions, nous parcourons les pixels de la projection de la surface anisotrope dans l'image des différences, et nous en déduisons l'écart-type moyen. Les figures 6.26 montrent l'évolution de ces écarts-types en fonction de θ pour la plaque d'aluminium dans les images réelles de la figure 6.22. On voit clairement, que la courbe des écarts types en fonction de l'angle θ possède un minimum : la direction d'anisotropie est donc représentée par ce vecteur d'orientation θ. Par ailleurs, on constate que l'algorithme est peu sensible aux textures se réfléchissant sur la surface anisotrope : dans les deux images réelles, que ce soit pour le cube rouge uniforme (image gauche de la figure 6.22), ou les livres (image droite de la figure 6.22), les courbes sont discriminantes pour une direction d'anisotropie, ce qui est sain.

La figure 6.25 montre[38] le processus décrit précédemment pour une image réelle possédant, de plus, des textures. L'algorithme calcule d'abord une zone qui correspond à un objet repondant au critère $\frac{Aire(surface)}{d(S,P)}$. Ici, c'est le livre rose posé sur la plaque d'aluminium qui est sélectionné. La différence entre l'image réelle et la réflexion spéculaire parfaite de ce livre sur l'objet anisotrope est ensuite calculée, et la direction d'anisotropie suréchantillonnée. À partir de ces directions, la

[38] On remarquera la robustesse de la technique, puisque dans les deux images réelles radicalement différentes mais utilisant la même plaque d'aluminium, l'algorithme a calculé la même direction d'anisotropie ($\theta = 0$). De plus, nous proposons en annexe D d'effectuer le rendu inverse à partir d'une image synthétique générée par *Phoenix*, afin de comparer les véritables valeurs des BRDF ayant servi à produire cette image avec celles trouvées par notre algorithme de regénération d'images.

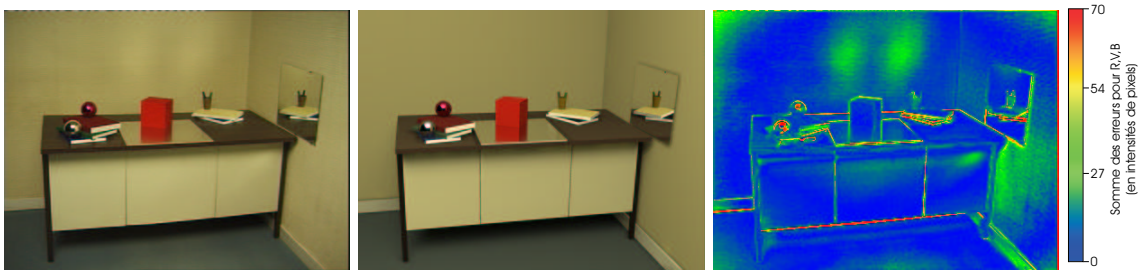

FIG. 6.27 – Simulation de rendu inverse avec à gauche l'image réelle, et à droite l'image de synthèse. Au centre des images, on voit la surface anisotrope en aluminium. On constate que l'erreur est restée importante uniquement sur les bords des objets, et ceci pour des raisons qui n'ont rien à voir avec notre technique d'analyse photométrique. L'explication de l'existence des grandes erreurs en dehors de ces contours sont données dans le paragraphe ci-après.

mesure de l'écart-type moyen permet de déduire une et une seule direction d'anisotropie $\vec{x}$.
Une fois la direction d'anisotropie trouvée, la quatrième et dernière étape de la recherche de la BRDF anisotrope consiste à minimiser la fonction d'erreur entre les deux images, sur α_x et α_y. En pratique, pour l'image de la figure 6.27, l'algorithme a trouvé 0 degré pour l'angle d'anisotropie (voir figure 6.26), ce qui signifie que la direction d'anisotropie est confondue avec le vecteur $\vec{X}$ du repère local à la surface.

Algorithme 33 Pseudo-Algorithme d'analyse du type de surface diffus, spéculaire, isotrope et anisotrope

```
Procédure Analyse_Reflectances(iter_rerendu, groupe, img_réelle, img_synthétique)
  Pour Tous les groupes i Faire
    Si ... Alors
      ... //Insérer la partie correspondante de la fonction Analyse_Reflectances() précédente
    Sinon
      Si groupe[i].type == diffus_spéculaire Alors
        Change_Groupe_DiffusSpéculaire_en_Isotrope(groupe[i]) ;
      Sinon
        Si groupe[i].type == isotrope Alors
          Change_Groupe_Isotrope_en_Anisotrope(groupe[i]) ;
        Fin si
      Fin si
    Fin si
  Fin
Fin Procédure
```

La fonction d'erreur en α_x et α_y à minimiser devient donc la première surface en haut à gauche de la figure 6.23. Cette fonction est facilement minimisable par la méthode du simplexe (que nous utilisons dans *Phoenix*), ou la méthode de Powell, et converge en une cinquantaine d'itérations (soit environ 2h de calcul, avec 10 rayons relancés pour chaque rayon primaire atteignant la surface anisotrope). Les images résultats obtenues pour la surface anisotrope sont celles de la figure 6.27, et celle de la figure 6.31 (qui contient des textures) au paragraphe suivant. On constate en ce qui concerne l'image 6.27, que l'erreur sur la plaque d'aluminium est restée élevée. Ceci a deux raisons principales fortement corrélées. En fait, lors de la construction de la scène, nous avons modélisé la source de lumière de façon très approximative (un ensemble de triangles formant un cercle), alors que celle-ci est extrêmement complexe (la surface émettrice est une vasque en aluminium, sur laquelle est posée une grille en acier, et autour de laquelle des panneaux noirs permettent de choisir la direction d'émission). Après expérimentations, nous avons trouvé ce qui produisait un tel écart entre les deux images : il s'avère que les panneaux directionnels de la source sont

Algorithme 34 Procédures modifiant les propriétés des surfaces isotropes ou anisotropes, en fonction des erreurs calculées

```
Procédure Change_Groupe_DiffusSpéculaire_en_Isotrope(groupe)
  Si groupe.ε ≥ seuil_diffus_spéculaire Alors
    groupe.type = isotrope ;
    groupe.fonction_erreur = erreur_BRDF() ;
  Fin si
Fin Procédure
Procédure Change_Groupe_Isotrope_en_Anisotrope(groupe)
  Si groupe.ε ≥ seuil_isotrope Alors
    groupe.type = anisotrope ;
    //La fonction d'erreur ne change pas par rapport à l'isotropie : erreur_BRDF()
  Fin si
Fin Procédure
```

légèrement surélevés par rapport à sa base émettrice, laissant ainsi filtrer un léger faisceau de lumière directe vers la plaque d'aluminium[39]. Ce faisceau provoque dans l'image réelle, d'une part, un suréclairement du mur droit et, d'autre part, la création de deux halos elliptiques sur le mur du fond par réflexion de la lumière directe à travers la plaque d'aluminium. Ces deux phénomènes sont responsables des erreurs présentes dans l'image droite de la figure 6.27. Comme nous regardons cette plaque d'aluminium, nous voyons également ces halos à travers la plaque, qui ont de fortes erreurs : ces erreurs se réfléchissant donc sur la plaque, notre algorithme estime -à juste titre- que l'erreur sur la plaque est élevée. De plus, comme notre méthode d'analyse ne prend en compte que la réflexion du cube, notre algorithme a trouvé la direction d'anisotropie optimale, ce qui explique pourquoi l'erreur est faible dans la réflexion du cube sur la plaque, mais pas ailleurs puisque la source de lumière est mal modélisée. Si cette erreur de modélisation de la source a une incidence forte sur l'image de synthèse de la figure 6.27, on constate qu'elle n'en a aucune visible à l'oeil nu en tout cas, sur l'image plus complexe 6.31.

[39] En fait, c'est encore plus subtile que cela : la lumière quittant le fond du cylindre émetteur, va se réfléchir sur la grille située à la sortie du cylindre. Cette lumière traverse en grande partie la grille métallique, tandis que le reste est réfléchi par elle vers la périphérie du cylindre. Cette périphérie n'étant pas parfaitement close, elle laisse s'échapper une partie de la lumière qui va ensuite illuminer la plaque d'aluminium.

Algorithme 35 Pseudo-Algorithme d'analyse intégrant les surfaces isotropes et anisotropes

```
Procédure Regenere_Scene()
  Tant Que Erreur totale(img_réelle - img_synthèse) > seuil Faire
    Pour Tous les groupes i Faire
      Si ... Alors
        ... //Insérer la partie correspondante de la fonction Regenere_Scene() précédente
      Sinon
        //Surface isotrope ?
        Si groupe[i].type == isotrope Alors
          //Minimisation de l'erreur en α, ρ_d et ρ_s
          Minimiser_Fonction_Isotrope_Simplexe(groupe[i]) ;
        Sinon
          //Surface anisotrope ?
          Si groupe[i].type == anisotrope Alors
            Calculer_Direction_Anisotropie(groupe[i], x⃗) ;
            //Minimisation de l'erreur en α_x, α_y avec le ρ_d et ρ_s du calcul isotrope et
            //le vecteur x⃗ estimé
            Minimiser_Fonction_Anisotrope_Simplexe(groupe[i]) ;
          Fin si
        Fin si
      Fin si
    Fin
    Calculer image de synthèse depuis les nouveaux paramètres de BRDF, avec Phoenix
    Analyse_Reflectances(iter_rerendu, groupe, img_réelle, img_synthétique) ;
  Fin
Fin Procédure
```

6.5.2.5 Surfaces texturées

Les surfaces texturées sont relativement faciles à traiter dans notre problématique. En effet, comme nous ne possédons qu'une seule image de référence, il est extrêmement difficile de savoir si un éclat spéculaire, une réflexion, ou une inter-réflexion présente dans la zone texturée de l'image, fait partie de la texture originale ou non. Yu et al.[251, 252] utilisent plusieurs dizaines d'images sous des angles différents, il devient alors beaucoup plus aisé pour eux d'estimer la texture non éclairée (sans éclat spéculaire par exemple), car on peut la reconstituer par déduction successive depuis plusieurs images réelles.

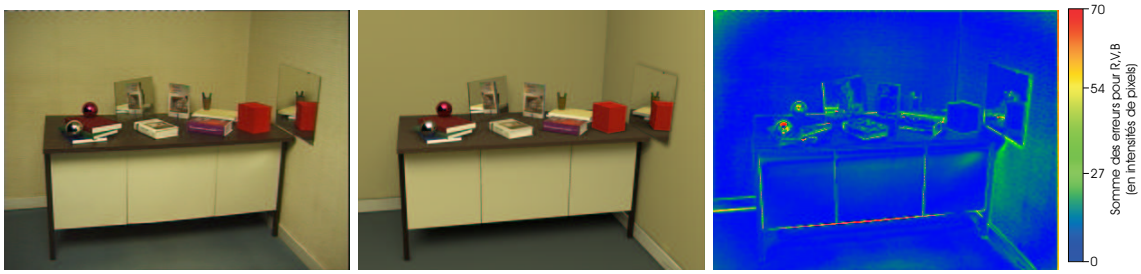

FIG. 6.28 – Simulation de rendu inverse avec à gauche l'image réelle, et à droite l'image de synthèse, par approximations lambertiennes, spéculaires, rugueuses et texturelles.

Nous ne savons pas estimer pour le moment, tout comme Drettakis et al.[56] ou Loscos et al.[132], ces éclats spéculaires sur des textures : nous découpons directement les textures dans

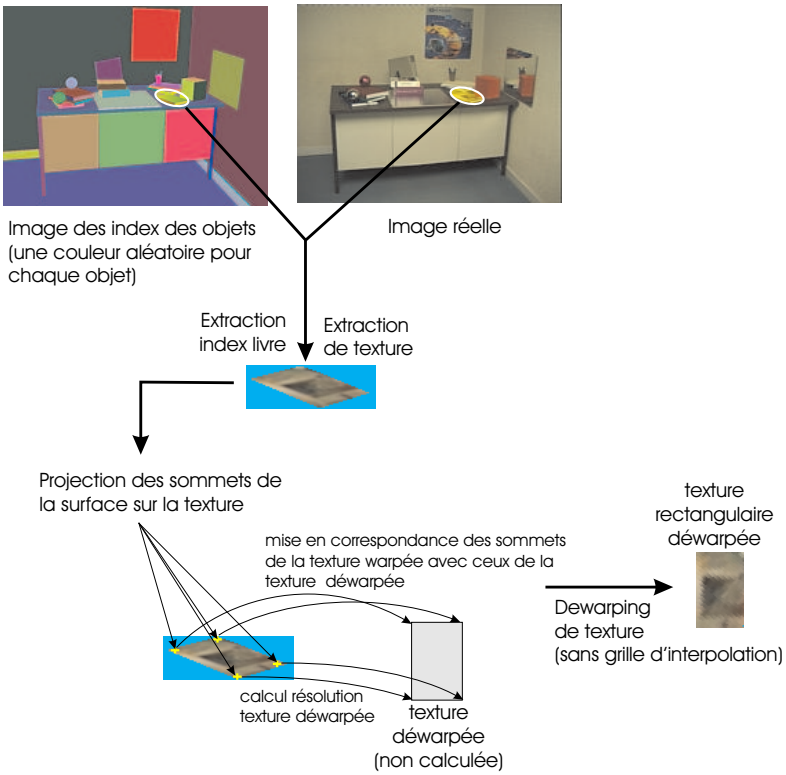

FIG. 6.29 – Processus d'extraction et de *dewarping* de textures

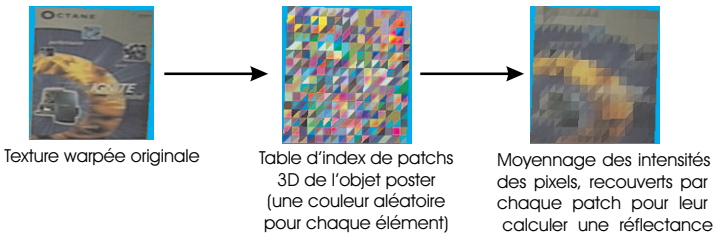

FIG. 6.30 – Calcul des réflectances diffuses des patchs pour une texture de l'image réelle. Le niveau de discrétisation de la surface en patchs a été ici volontairement exagéré pour rendre l' exemple plus explicite.

l'image réelle et recopions tel quel ces éclats spéculaires. Cependant, il est raisonnable de penser que si nous disposons de plusieurs images, il est possible d'avoir la texture sous des angles différents : ainsi il est probable que les éclats spéculaires se "déplacent" sur la texture et que nous puissions en déduire la texture sans ces éclats. Cette partie n'est que purement théorique, et nous ne l'avons pas encore implémentée, puisque nous ne disposons pour le moment que d'une seule et unique image par scène.

Algorithme 36 Procédures modifiant les propriétés des surfaces texturées

Procédure Change_Groupe_Anisotrope_en_Texture(groupe)
  Si groupe.$\varepsilon \geq$ seuil_anisotrope **Alors**
    groupe.type = texture ;
    //La surface est considérée comme une surface texturée, nous extrayons les
    //textures par objet, et appliquons une redressement (*dewarping*) pour pouvoir les utiliser
    Pour Tous les objets j **Faire**
      Calculer_Texture(groupe.objet[j], img_réelle) ;
    Fin
  Fin si
Fin Procédure

Algorithme 37 Pseudo-algorithmes de calcul des textures

Procédure Calculer_Texture(objet, img_réelle)
  //Les réflectances des patchs sont calculées depuis la texture originale déformée
  //sinon le redressement de la texture peut biaiser le calcul des ρ_d
  objet− >texture = texture extraite de l'image réelle grâce à la table d'index des objets ;
  Pour Tous les patchs k **Faire**
    Calculer par offscreen-rendering la réflectance diffuse moyenne du patch k (figure 6.30) ;
  Fin
  //Redressement (= *dewarping*) de la texture pour le rendu final
  Calculer les coordonnées des sommets des facettes de l'objet, projetés sur l'image ;
  Redresser objet− >texture en se servant des coordonnées précédentes pour produire une image rectangulaire (figure 6.29) ;
Fin Procédure

Algorithme 38 Pseudo-algorithmes de correction itérative des textures

Procédure Corriger_Texture(objet, img_réelle, img_synthétique)
  //Calcul de l'image des erreurs
  texture_tmp = texture extraite de l'image synthétique regénérée ;
  texture_ε = $\frac{\text{texture(objet,img\_réelle)}}{\text{texture\_tmp}}$;
  Calculer nouvelle texture nouv_text = objet− > texture × texture_ε ;
  //Calcul des réflectances des patchs pour l'équation de radiosité
  Pour Tous les patchs k se projetant sur nouv_text **Faire**
    Calculer ρ_{d_k} comme la moyenne des intensités de pixel
    recouverts par la projection de k ;
  Fin
  //Redressement de la nouvelle texture pour le prochain rendu
  Calculer les coordonnées des sommets des facettes de l'objet, projetés sur l'image ;
  Redresser nouv_text en se servant des coordonnées précédentes pour produire une image rectangulaire (figure 6.29) ;
Fin Procédure

Dans notre technique, nous supposons donc désormais que les surfaces pour lesquelles toutes les hypothèses précédentes ont échoué, sont texturées (voir algorithmes 39 et 40).
Nous procédons alors à l'extraction automatique des textures (grâce à des tables d'index calculées par patch, et non plus par objet ou groupe, voir algorithmes 36, 37 et figure 6.30, image de milieu), puis à leur redressement[40] (voir figure 6.29), suivant la méthode proposée par Wolberg[245], pages 53-54. Pour chacun des patchs, nous calculons une réflectance qui est la moyenne des intensités de pixel recouverts par leur projection. *Phoenix* génère alors une première image en utilisant directement la texture extraite (inversée par la fonction de transfert de la caméra). La réflectance de la surface est bien entendu sous-estimée (si elle est ombrée) ou surestimée (si elle n'est pas ombrée), car la réflectance extraite contient déjà la radiosité reçue par inter-réflexions. Nous employons alors un algorithme de principe similaire à celui décrit précédemment pour les surfaces diffuses. Dans un premier temps nous extrayons la texture de l'image générée par *Phoenix*. Dans un second temps, nous calculons l'image des erreurs relatives comme le rapport de la texture originale sur la texture extraite de l'image de synthèse. Dans un troisième temps, la texture qui a été employée pour générer cette image de synthèse est multipliée par l'image des erreurs pour obtenir une nouvelle texture (voir algorithme 38). Cette nouvelle texture est alors utilisée par notre logiciel de rendu réaliste pour générer une nouvelle image de synthèse, et ainsi de suite jusqu'à ce que l'erreur totale sur la zone texturée soit inférieure à un seuil fixé par l'utilisateur. En pratique, une telle technique converge en 4 itérations environ comme pour le cas diffus décrit au paragraphe 6.5.2.1. Il est à noter que pour les phases de rendu réaliste, chaque nouvelle texture est employée pour calculer des réflectances par patch, servant ainsi à la résolution de l'équation de radiosité. Pour le calcul de l'image finale par lancer de rayons, la nouvelle texture corrigée est rendue en employant la technique proposée par Cohen et al.[37].

Par ailleurs, il est clair que cette technique présente l'inconvénient de "recopier" les zones d'ombre texturées, tout comme les éclats spéculaires, mais il s'agit là d'un défaut inhérent au nombre d'images initiales (une seule).
Les images 6.28 et 6.31 ont été obtenues par cette technique.

Algorithme 39 Pseudo-Algorithme d'analyse des surfaces diffuses, spéculaires, isotropes, anisotropes et texturées

```
Procédure  Regenere_Scene()
  Tant Que Erreur totale(img_réelle - img_synthèse) > seuil Faire
    Pour Tous les groupes i Faire
      Si ... Alors
        ... //Insérer la partie correspondante de la fonction Regenere_Scene() précédente
      Sinon
        //Surface texturée
        Si groupe[i].type == texture Alors
          Pour Tous les objets j Faire
            Corriger_Texture(groupe[i].objet[j], img_réelle, img_synthétique);
          Fin
        Fin si
      Fin si
    Fin
    Calculer image de synthèse depuis les nouveaux paramètres de BRDF, avec Phoenix
    Analyse_Reflectances(iter_rerendu, groupe, img_réelle, img_synthétique);
  Fin
Fin Procédure
```

À ce stade de l'algorithme, nous sommes en mesure de regénérer des surfaces diffuses, spé-

[40] Ce "redressement" de textures est connu sous le nom de *texture unwarping* ou *dewarping* dans la littérature infographiste.

Algorithme 40 Pseudo-Algorithme général d'analyse du type de surface

```
Procédure Analyse_Reflectances(iter_rerendu, groupe, img_réelle, img_synthétique)
  Pour Tous les groupes i Faire
    Si ... Alors
      ... //Insérer la partie correspondante de la fonction Analyse_Reflectances() précédente
    Sinon
      Si groupe[i].type == anisotrope Alors
        Change_Groupe_Anisotrope_en_Texture(groupe[i]) ;
      Fin si
    Fin si
  Fin
Fin Procédure
```

FIG. 6.31 – Simulation de rendu inverse avec à gauche l'image réelle, et à droite l'image de synthèse, contenant des surfaces lambertiennes, spéculaires, texturées et à BRDF complexes.

culaires, avec des BRDF complexes, et texturées. Nous pouvons encore, néanmoins, améliorer légèrement l'image obtenue. En effet, notre technique utilise une fonction de transfert caméra, de type correction γ, et cette fonction a été initialisée avec $\gamma = 2.2$ suivant l'article de Tumblin et Rushmeier[222]. Nous proposons de minimiser l'erreur globale entre l'image de synthèse regénérée et l'image réelle, en trouvant le γ optimal qui permet de transformer l'image synthétique des luminances en intensités de pixel.

6.6 Minimiser la fonction de transfert caméra

La méthode qui consiste à minimiser la fonction de transfert caméra est très simple, puisqu'il s'agit de minimiser :

$$Err(\gamma) = (I_{\text{réelle}} - I_{\text{synthèse}})^2 = \sum_{i=1}^{n} (P_{org_i} - (L_{syn_i})^{\frac{1}{\gamma}})^2 \tag{6.12}$$

avec :

P_{org_i} l'intensité du pixel i dans l'image réelle
L_{syn_i} la luminance du pixel i dans l'image de synthèse
n le nombre de pixel de l'image réelle
γ le coefficient de correction gamma

Pour minimiser l'équation 6.12, il faut que son gradient soit nul :

$$\frac{dErr(\gamma)}{d\gamma} = 0 \tag{6.13}$$

$$\Leftrightarrow \quad \frac{2}{\gamma^2} \cdot \sum_{i=1}^{n} (P_{org_i} - (L_{syn_i})^{\frac{1}{\gamma}}) \cdot (L_{syn_i})^{\frac{1}{\gamma}} \cdot \log(L_{syn_i}) = 0 \tag{6.14}$$

$$\Leftrightarrow \quad \sum_{i=1}^{n} (P_{org_i} \cdot (L_{syn_i})^{\frac{1}{\gamma*}} \cdot \log(L_{syn_i})) = \sum_{i=1}^{n} (L_{syn_i})^{\frac{2}{\gamma*}} \cdot \log(L_{syn_i}) \tag{6.15}$$

$\gamma*$ peut être obtenu, soit en utilisant la méthode du gradient (voir équation 6.14). Il est aussi concevable de trouver $\gamma*$ en résolvant de manière implicite l'équation 6.15.

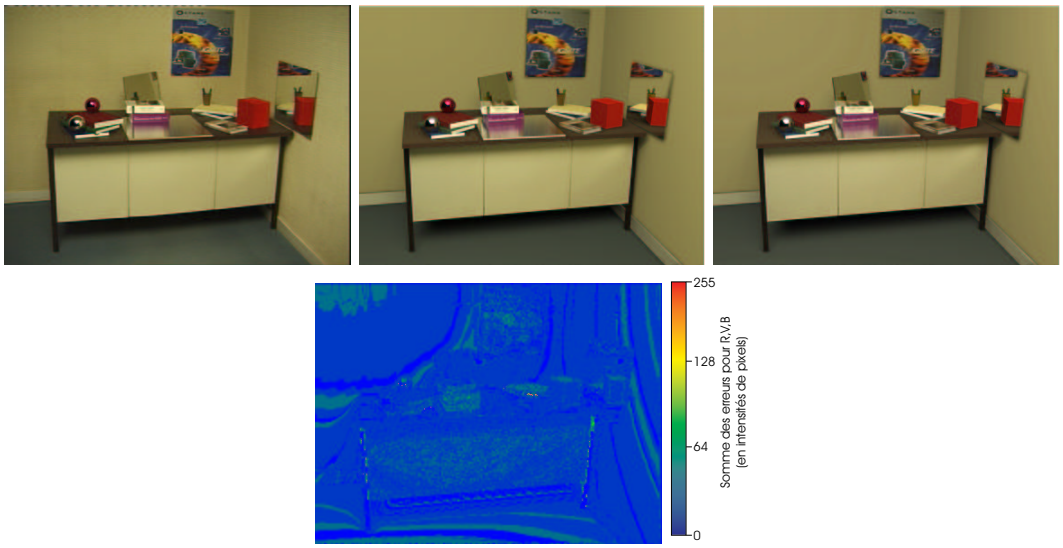

FIG. 6.32 – Correction du γ optimal, en minimisant l'erreur entre l'image réelle (à gauche) et l'image synthétique. L'image en haut au centre a été obtenue avec $\gamma = 2.2$, tandis que l'image en haut à droite a été calculée avec le γ optimal à 2.18, pour un gain total très faible de 0.06% en qualité. Enfin l'image du bas représente la différence entre les deux images de synthèse (on remarquera que l'échelle des erreurs n'est pas la même que pour les images précédentes).

Globalement, le résultat de cette minimisation (par la méthode du simplexe par exemple), est très difficilement perceptible visuellement. Sur la troisième image de la figure 6.32, nous obtenons une valeur optimale de γ à 2.18, au lieu du 2.2 employé pour l'image de synthèse regénérée (seconde image de la figure 6.32), et le gain total sur l'image avec le nouveau γ est de 0.06%. Cependant, il nous paraît tout de même intéressant de préciser qu'une amélioration sur γ est possible pour l'image de synthèse.

Chapitre 7

Résultats de regénération d'images et Applications

7.1 Exemples d'images regénérées

Nous présentons ici l'ensemble des résultats que nous avons obtenus en regénération d'images, depuis une image réelle.

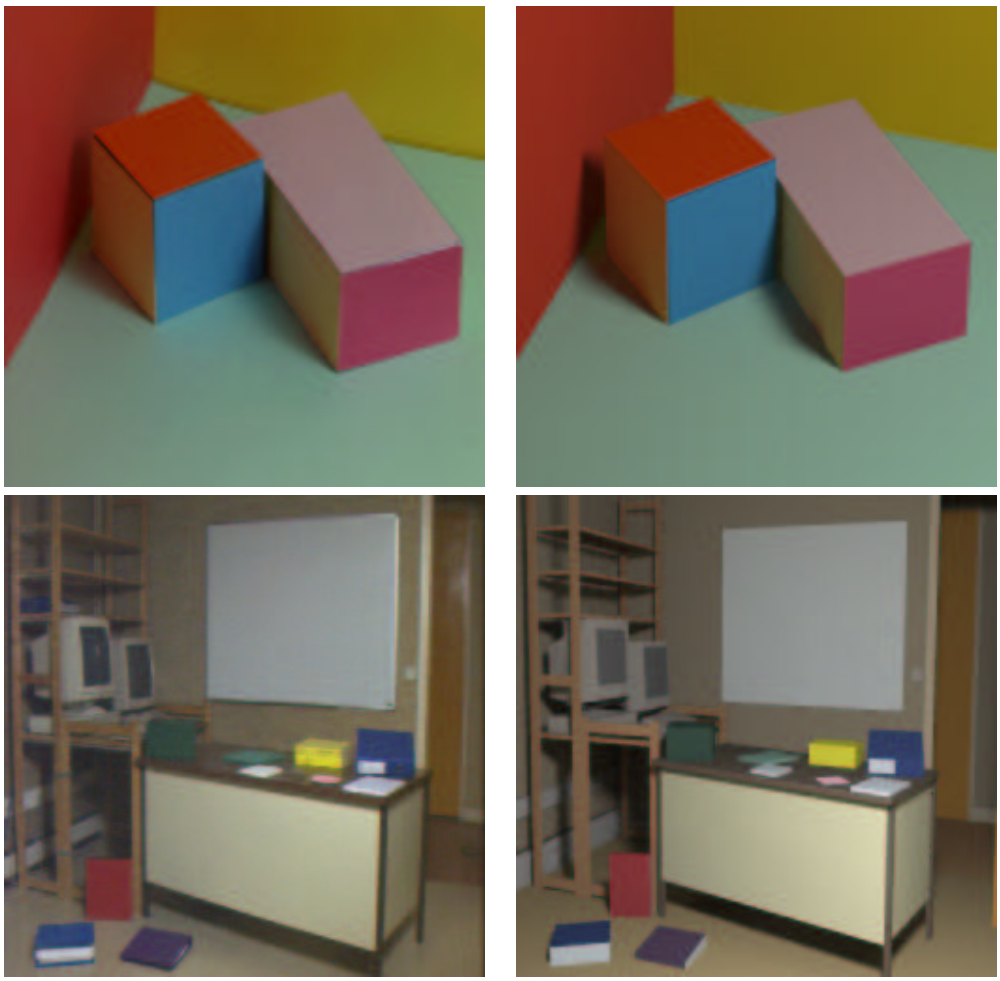

FIG. 7.1 – Scène totalement diffuse, regénérée en image de synthèse (à droite) depuis une image réelle (à gauche) capturée avec une caméra

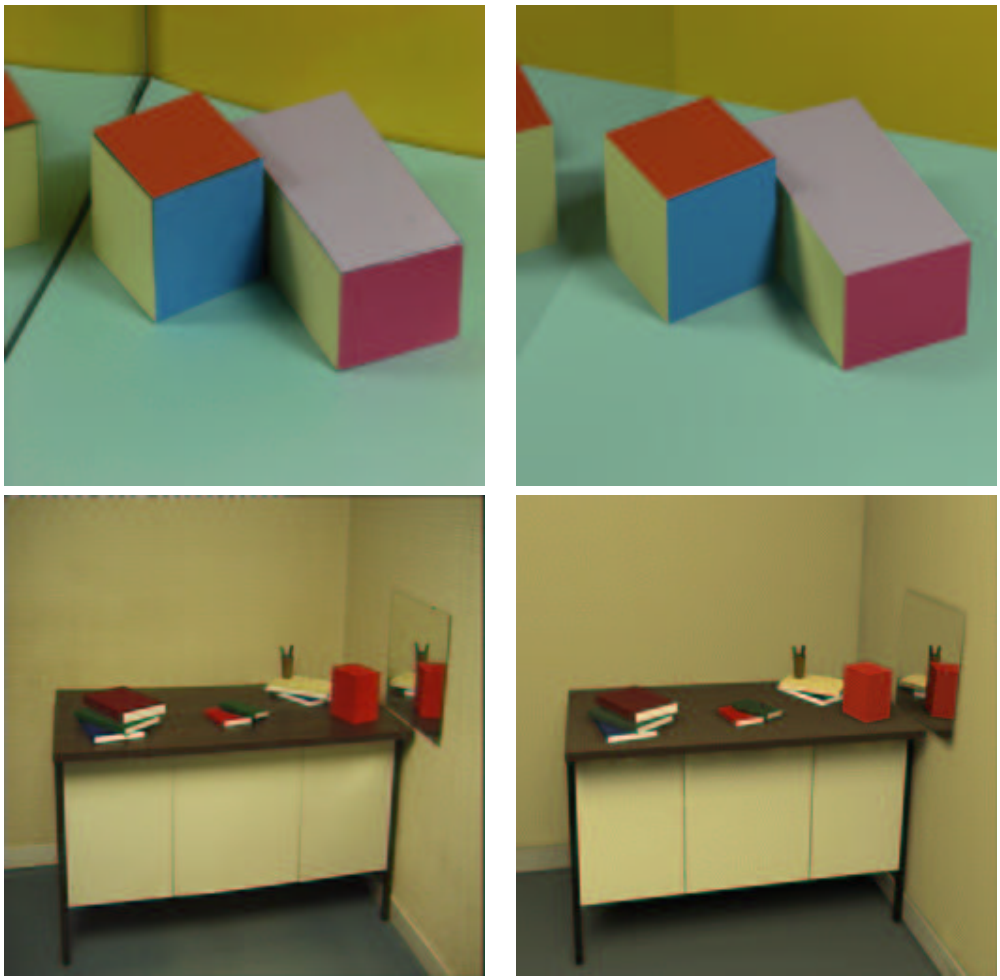

FIG. 7.2 – Scène possédant des surfaces soit totalement diffuses, soit spéculaires parfaites, et regénérée en image de synthèse (à droite) depuis une image réelle (à gauche) capturée avec une caméra

FIG. 7.3 – Scène contenant des objets totalement diffus, partiellement diffus, spéculaires parfaits, partiellement spéculaires, regénérée en image de synthèse (à droite) depuis une image réelle (à gauche) capturée avec une caméra

FIG. 7.4 – Scène contenant des objets multiples, diffus, spéculaires, texturés, anisotropes, etc. L'image originale (à gauche) a été regénérée en image de synthèse (à droite) depuis une seule image réelle capturée avec une caméra

7.2 Les applications

Le rendu à base d'images réelles possède de nombreuses applications, que ce soit dans le commerce électronique, les jeux vidéo, le cinéma, la compression de données, la vision par ordinateur, l'interprétation d'images, la robotique, etc. Nous proposons ici deux applications directes que nous considérons comme particulièrement intéressantes, bien que beaucoup d'autres soient possibles.

7.2.1 Réalité Augmentée

La réalité augmentée[103, 66, 56] est la plus immédiate des applications potentielles de notre technique. En effet, si l'illusion visuelle est quasi-parfaite entre une image réelle et une image de synthèse et que nous en disposons désormais d'une représentation tridimensionnelle[1], il devient alors possible d'ajouter ou de retirer des objets réels ou synthétiques, de modifier la position de l'observateur et les conditions d'illumination initiales. On peut ainsi s'arranger pour créer des effets spéciaux , en modifiant certains objets que ce soit d'un point de vue géométrique (voir image 7.5), ou photométrique (voir image 7.7). L'ajout d'objets complètement décorrelé de la scène est également une possibilité visuelle intéressante (voir image 7.8), tout comme l'ajout de nouveaux objets particulièrement réalistes. Bien sûr, il est également impressionnant de pouvoir changer le point de vue alors que nous ne disposions que d'une seule image réelle (voir image 7.6), tandis que la modification des sources de lumière permet de créer de nouvelles conditions d'illumination (voir image7.8).

FIG. 7.5 – Exemple d'application possible sur la soustraction d'objets existants. Le cube rouge a été retiré de l'image et on constate que le bureau flotte dans l'air (les pieds ont été raccourcis).

[1] C'est typiquement ce que nous obtenons avec *Phoenix*.

FIG. 7.6 – Exemple d'application possible sur la modification de la position d'observation. Après avoir été regénéré, on a déplacé le point de vue vers le centre de l'image de la figure 6.31.

FIG. 7.7 – Exemple d'application possible sur la modification des propriétés photométriques des objets. On remarque également l'insertion d'un nouvel objet.

FIG. 7.8 – Exemple d'application possible sur l'ajout de nouveaux objets et de modification des conditions d'illumination. Une lampe de bureau a été ajoutée, ainsi qu'un droïde près de la plaque d'aluminium, et l'illumination de la scène a été modifiée.

7.2.2 Compression de données

Nous n'avons pu tester cette application, faute de temps. Néanmoins, cette application est tout à fait réalisable en pratique. En effet, supposons que nous ayons capturé une séquence d'images, dans laquelle seule la caméra se déplace[2]. Si nous possédons toute la géométrie de cette scène d'intérieur et que nous avons réestimé les réflectances des surfaces, de façon à obtenir une image aussi proche que possible de l'originale, nous pouvons prétendre réaliser une compression de l'ensemble de cette séquence d'un facteur équivalent au nombre d'images de la séquence[3], car nous pouvons regénérer la totalité des autres images depuis les données que nous avons calculées. Ainsi, l'information à transmettre peut être réduite en : les données géométriques 3D de la scène, ses données photométriques, les paramètres intrinsèques de la caméra, celles-ci ne devant être transmises qu'au départ, et ensuite pour chaque image, les données extrinsèques de la caméra.

[2]On pourrait également prendre en compte des objets mobiles, mais cela est certainement nettement plus complexe, et il est difficile de se prononcer sur la faisabilité pratique d'une telle hypothèse.

[3]Les données nécessaires pour un rendu réaliste après estimation des réflectances sont de la taille d'une image de cette séquence environ. Il faut bien sûr également conserver la position de l'observateur pour chaque image, mais ceci est d'une taille négligeable par rapport à l'ensembles des autres données. Il est clair cependant que cette taille peut excéder celle d'une image simple, si toutes les surfaces sont estimées comme texturées, puisque nous devons également stocker la texture pour chaque surface. Néanmoins, il reste peu probable que la taille totale de toutes ces textures excèdent deux ou trois images de la scène, à moins qu'il ne s'agisse d'une scène extrêmement complexe.

Chapitre 8

Conclusion et extensions futures

8.1 Les objectifs accomplis

Les objectifs de cette thèse étaient multiples. Tout d'abord, nous devions développer un logiciel de rendu réaliste très rapide et qui soit capable de produire des images de qualité photographique. Ce logiciel que nous avons appelé *Phoenix* a représenté un gros investissement en recherche bibliographique pour choisir les algorithmes les plus performants, mais aussi en temps et en qualité de programmation : il ne sert en effet à rien d'avoir des algorithmes puissants mais implémentés de façon maladroite. Bien que cette partie ne soit pas à proprement parler innovante, elle était indispensable pour la création de nouveaux algorithmes de rendu à base d'images. Sans *Phoenix* avec sa vitesse de calcul et ses outils graphiques d'analyse de réflectances, nos nouvelles techniques n'auraient pu voir le jour.

Ce sont justement ces nouvelles méthodes de type collaboration analyse/synthèse qui ont représenté le second objectif de cette thèse : réaliser des algorithmes entièrement novateurs et utilisant des données aussi réduites que possible. Nous pensons avoir largement atteint cet objectif, puisque nous sommes en mesure de regénérer une image de synthèse comparable à une vraie image issue d'une caméra quelconque, en n'exploitant uniquement quatre données : le modèle 3D de la scène, la position et la géométrie des sources de lumière, la position de la caméra ainsi que ses paramètres intrinsèques, et une seule image de la scène. Nous avons donc développé tout un ensemble d'algorithmes capables de retrouver les BRDF de surfaces en se fondant sur le modèle de Ward[236]. À ce jour, *Phoenix* est capable de retrouver les paramètres de réflectance de surfaces totalement ou partiellement diffuses, totalement ou partiellement spéculaires, isotropes, anisotropes ou texturées. Nous ajouterons qu'à notre connaissance (voir chapitre 5), nous sommes les seuls à parvenir à regénérer des images aussi complexes avec une et une seule image réelle (surfaces spéculaires parfaites et surfaces anisotropes par exemple).

8.2 Quelques précisions

En conclusion, il est important de préciser deux points. Tout d'abord, la notion de seuils que nous avons employée régulièrement est très importante pour notre algorithme, car ce sont ces seuils qui guident les décisions que prend l'algorithme pour choisir le type de réflectance des surfaces. Cependant, nous ne leur avons pas consacré de paragraphe particulier, car ces seuils sont fixés par l'utilisateur une fois pour toutes. En effet, nous avons toujours gardé les mêmes seuils d'erreurs quelle que soit la scène que nous traitions (sauf une fois pour l'exemple du paragraphe 6.5.2.3). En pratique, ces seuils représentent l'erreur que nous sommes prêts à tolérer visuellement. Ainsi, un seuil relatif inférieur à 1% rend l'algorithme très sensible aux erreurs, et aura tendance à vouloir appliquer des textures partout (sauf sur les surfaces spéculaires parfaites). Par contre, un seuil inférieur à 5% permettra à l'algorithme de choisir plus judicieusement la meilleure approximation

possible de la réflectance réelle. Pour nos images, nous avons utilisé les seuils suivants : 5% pour passer de diffus à spéculaire parfait, de spéculaire parfait à spéculaire non parfait, de spéculaire non parfait à diffus et spéculaire sans rugosité, et de diffus et spéculaire sans rugosité à isotrope. Pour forcer l'algorithme à trouver la meilleure BRDF possible, nous avons par contre placé le seuil d'isotropie très bas (1%), tandis que l'hypothèse d'anisotropie peut tolérer jusqu'à 5% d'erreur avant de choisir de texturer la surface. Dans ce dernier cas, l'algorithme procède à des itérations successives pour réduire l'erreur entre les deux images, sachant qu'avec le bruit local présent sur la surface texturée (voir le poster sur l'image des erreurs de la figure 6.31), il est totalement utopique d'espérer atteindre 0%.
Finalement, nous obtenons des images de synthèse pour lesquelles l'erreur finale cumulée dans les trois longueurs d'onde R, V, B est de l'ordre de moins de 5%, ce qui correspond à des différences entre l'image réelle et l'image synthétique à peine perceptibles à l'oeil nu. Cette erreur est supérieure à 25% pour 1 pixel sur 4000, et est provoquée en général par des problèmes de calage approximatif du modèle 3D sur l'image réelle.
Par ailleurs, le deuxième point que nous souhaitons éclaircir repose sur le nombre d'itérations. D'une manière identique aux seuils, nous dirons que l'utilisateur arrête l'algorithme quand l'image lui paraît suffisamment proche de l'originale (il est aidé en cela par les différents outils de *Phoenix* qui lui permettent de visualiser avec précision les zones d'erreurs fortes dans les images regénérées).

8.3 Les extensions futures

Bien sûr, nous ne prétendons pas avoir résolu tous les problèmes dans le domaine du rendu réaliste à base d'images réelles, car de nombreux autres types de surfaces restent à traiter. La prochaine et immédiate extension consistera notamment à prendre en compte des surfaces possédant des éclats spéculaires (ce qui avec le modèle de Ward ne devrait pas poser beaucoup de problèmes), et à traiter des objets à la fois texturés et anisotropes (pour cette dernière possibilité, nous serons probablement contraints d'étendre le nombre d'images réelles nécessaires à l'analyse photométrique à deux ou plus). Une autre extension va consister à traiter des scènes possédant plusieurs sources de lumière d'intensité et de couleurs différentes. Il est également envisagé de traiter des surfaces transparentes et des milieux participatifs plus ou moins complexes (l'utilisateur aurait alors à modéliser une sorte de volume englobant de l'objet en question). Par ailleurs, il est prévu d'appliquer notre technique au calage géométrique de miroirs. En effet, nous avons constaté que l'étape de modélisation des miroirs, pour qu'ils reflètent correctement les objets lors du rendu inverse, est une opération très délicate : un déplacement de quelques millimètres du miroir provoque des changements radicaux dans les réflexions spéculaires. Nous proposerons donc une méthode pour caler automatiquement ce miroir en minimisant l'erreur entre sa projection dans l'image réelle et sa projection dans l'image de synthèse.
Enfin, concernant le rendu pur de *Phoenix*, nous allons tenter d'augmenter sa vitesse de calcul sur les surfaces *glossy*, car celles-ci sont très gourmandes en temps de calcul. En effet, lors des simulations de regénération d'images, nous nous sommes rendu compte que sur un temps de calcul total d'environ 4h30 pour l'image de la figure 6.31, 4h sont consacrées au calcul des surfaces isotropes et anisotropes[1]. Il est donc dans nos priorités de tenter de réduire ce temps d'au moins 75%, pour rendre notre logiciel utilisable par des professionnels des effets spéciaux ou des artistes par exemple.

[1] On remarquera à ce sujet qu'une optimisation immédiate consiste à supprimer le cas isotrope, en ne gardant que l'anisotropie, car l'isotropie est un cas particulier d'anisotropie. Le temps de calcul est alors réduit de presque 50%, car l'étape d'isotropie pour la figure 6.31 a tout de même duré 1h50 (voir paragraphe 6.5.2.4).

Annexe A

Traduction des grandeurs radiométriques et photométriques

Lors de cette thèse, nous avons souvent été confronté aux différences importantes entre un terme français et une terme anglais, pour définir des quantités radiométriques ou photométriques. Nous avons notamment pu constater l'emploi à tort de certains termes, comme la radiosité ou l'émittance, qui n'existent pas dans nos ouvrages français de physique. Il s'agit là en fait d'abus de langage la plupart du temps. Mais le plus génant est que plusieurs termes sont de faux amis. Ainsi, nous aurions pu penser que la traduction française de *radiance* serait simplement *radiance*, mais ce terme a disparu des livres de physique. Nous proposons donc ici un récapitulatif des grandeurs radiométriques et photométriques, ainsi que leurs homonymes anglais.

Radiométrie		
Grandeur	**Unités**	**Traduction**
Flux Energétique	W	Flux (radiant power)
Intensité Energétique	W/sr	Radiant Intensity
Luminance Energétique	$W/sr/m^2$	Radiance
Exitance Energétique (radiosité)	W/m^2	Radiant Exitance (radiosity)
Eclairement Energétique	W/m^2	Irradiance

Photométrie		
Grandeur	**Unités**	**Traduction**
Flux Lumineux	lm	Luminous Flux
Intensité Lumineuse	$cd\,(lm/sr)$	Luminous Intensity
Luminance Lumineuse	$cd/m^2\,(lm/sr/m^2)$	Luminance
Exitance Lumineuse	lm/m^2	Luminous Exitance (luminosity)
Eclairement Lumineux	lux	Illuminance

Annexe B

Exemple de code pour l'offscreen rendering

B.1 Offscreen-rendering par OpenGL sans hardware

```
#include <GL/gl.h>
#include <GL/glx.h>
#include <GL/glu.h>
#include <stdio.h>
#include <stdlib.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>

// Compiler avec: 'cc offscreen.c -o offscreen -lGLU -lGL -lX11'

main()
{
  // Variables pour l'offscreen rendering par PIXMAPS
  Display *dpy;
  XVisualInfo *vis;
  Pixmap pix;
  GLXContext cx;
  GLXPixmap px;
  // Attributs indispensables pour l'offscreen rendering
  int attribList[]={GLX_RGBA,GLX_RED_SIZE,1,GLX_GREEN_SIZE,1,
  GLX_BLUE_SIZE,1,GLX_DEPTH_SIZE,24,None};

  // fichier resultat
  FILE *file;
  // buffer de lecture du resultat
  unsigned char *pixels;
  // resolution de l'offscreen-rendering (max 1024x1024)
  int w=512, h=512;

  // Creation du display
  dpy=XOpenDisplay(0);

  // Creation du visual
  vis=glXChooseVisual(dpy,DefaultScreen(dpy),attribList);
```

```
// creation du contexte graphique, avec ici GL_FALSE
// pour le booleen direct, donc pas d'utilisation du hardware
cx = glXCreateContext(dpy,vis,0,GL_FALSE);

// Creation du pixmap, ou on va dessiner la scene
pix = XCreatePixmap (dpy, RootWindow(dpy, vis->screen), w, h, vis->depth);

// Connexion au du pixmap precedent au pixmap offscreen
px = glXCreateGLXPixmap (dpy, vis, pix);

// Y-a-t-il un probleme avec l'offscreen ?
if (!(glXMakeCurrent(dpy,px,cx)))
    {
    fprintf(stderr, "glXMakeCurrent failed (window)!\n");
    exit(0);
    }

// Toutes les commandes precedentes sont OK, toutes les instructions
// OpenGL vont desormais s'executer dans le buffer offscreen

// Initialisation OpenGL
glClearColor(255,255,255,0);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glEnable(GL_CULL_FACE);
glEnable(GL_DEPTH_TEST);
glShadeModel (GL_FLAT);

// Projection Perspective
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(45.0, (GLfloat)w/(GLfloat)h, 1.0, 2000.0);

// on passe tout dans le repere de la camera
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

// position de la camera
glPushMatrix();
gluLookAt (10, 8, 5, 0, 0, 0, 0, 1, 0);

// Trace des 6 faces d'un cube 3D
glColor3f(1, 1, 0);
glBegin(GL_POLYGON);
glVertex3f(-2,  2, -2);
glVertex3f( 2,  2, -2);
glVertex3f( 2, -2, -2);
glVertex3f(-2, -2, -2);
glEnd();
glColor3f(0, 1, 0);
glBegin(GL_POLYGON);
glVertex3f(-2, -2, 2);
glVertex3f( 2, -2, 2);
glVertex3f( 2,  2, 2);
```

```
  glVertex3f(-2,  2, 2);
  glEnd();
  glColor3f(0, 0, 1);
  glBegin(GL_POLYGON);
  glVertex3f(-2, -2, -2);
  glVertex3f(-2, -2,  2);
  glVertex3f(-2,  2,  2);
  glVertex3f(-2,  2, -2);
  glEnd();
  glColor3f(1, 0, 0);
  glBegin(GL_POLYGON);
  glVertex3f( 2,  2, -2);
  glVertex3f( 2,  2,  2);
  glVertex3f( 2, -2,  2);
  glVertex3f( 2, -2, -2);
  glEnd();
  glColor3f(0, 1, 1);
  glBegin(GL_POLYGON);
  glVertex3f(-2,  2, -2);
  glVertex3f(-2,  2,  2);
  glVertex3f( 2,  2,  2);
  glVertex3f( 2,  2, -2);
  glEnd();
  glColor3f(1, 0, 1);
  glBegin(GL_POLYGON);
  glVertex3f( 2, -2, -2);
  glVertex3f( 2, -2,  2);
  glVertex3f(-2, -2,  2);
  glVertex3f(-2, -2, -2);
  glEnd();
  glFlush();

  // allocation du buffer de recuperation du resultat offscreen
  pixels = (unsigned char *)malloc(sizeof(unsigned char)*w*h*3);

  // Lecture du buffer resultat
  glReadPixels(0,0,w,h,GL_RGB,GL_UNSIGNED_BYTE, pixels);

  // ecriture dans un fichier image au format PPM
  // (affichable avec xv3.10)
  file = fopen("offsc.ppm","wb");
  fprintf(file,"P6\n%d %d\n255\n", w, h);
  fwrite(pixels,w*h*3*sizeof(unsigned char),1,file);
  fclose(file);

  // Destruction de pixmap GLX
  glXDestroyGLXPixmap(dpy, px);
  // Destruction du pixmap
  XFreePixmap(dpy, pix);
  // Destruction contexte graphique
  glXDestroyContext(dpy, cx);
  // Fermeture du display
  XCloseDisplay(dpy);
}
```

B.2 Offscreen-rendering par OpenGL avec hardware (P-Buffers)

```
#include <GL/gl.h>
#include <GL/glx.h>
#include <GL/glu.h>
#include <stdio.h>
#include <stdlib.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>

// Compiler avec: 'cc pbuffers.c -o pbuffers -lGLU -lGL -lX11'

main()
{
  // Variables pour les P-Buffers
  Display *dpy;
  GLXFBConfigSGIX *fbconfig;
  int  screen;
  GLXContext cx;
  int fbconfig_count;
  GLXPbufferSGIX pbuffer;
  // Attributs indispensables pour l'offscreen rendering par PBUFFERS
  int attribList[]={GLX_RENDER_TYPE_SGIX, GLX_RGBA_BIT_SGIX, GLX_RED_SIZE, 4,
  GLX_GREEN_SIZE, 4, GLX_BLUE_SIZE, 4, GLX_DRAWABLE_TYPE_SGIX,
  GLX_PBUFFER_BIT_SGIX, GLX_DEPTH_SIZE, 24, None};
  // fichier resultat
  FILE *file;
  // buffer de lecture du resultat
  unsigned char *pixels;
  // resolution de l'offscreen-rendering (max 1024x1024)
  int w=512, h=512;

  // Creation du display
  dpy = XOpenDisplay(0);

  // recuperation de l'ecran
  screen = DefaultScreen(dpy);

  // Recuperation de la configuration du frame buffer
  fbconfig = glXChooseFBConfigSGIX(dpy,
                                   screen,
                                   attribList,
                                   &fbconfig_count);

  // La carte graphiqe est-elle capable d'utiliser des P-Buffers ?
  if(fbconfig == NULL || fbconfig_count <= 0)
    {
    fprintf(stderr,"Station ne possedant pas les P-Buffers.\n");
    exit(0);
    }

  // Creation du P-Buffer
  pbuffer = glXCreateGLXPbufferSGIX(dpy,
                                    fbconfig[0],
```

```
                                w, h,
                                NULL);

// La creation du P-Buffer a-t-elle fonctionnee ?
if(pbuffer == NULL)
  {
  fprintf(stderr, "Creation du pbuffer impossible.\n");
  exit(0);
  }

// Creation du contexte de rendu GLX
// On remarque que le booleen direct est place a GL_TRUE
// ce qui signifie que le hardware de la station est utilise
// pour tous les calculs 3D
cx = glXCreateContextWithConfigSGIX(dpy,
                                        fbconfig[0],
                                        GLX_RGBA_TYPE_SGIX,
                                        NULL, GL_TRUE);

// A-t-on reussi a creer le contexte graphique ?
if(!cx)
  {
  fprintf(stderr, "Creation du contexte graphique impossible.\n");
  exit(0);
  }

// Y-a--til un probleme avec les p-buffers ?
if(!glXMakeCurrent(dpy, pbuffer, cx))
  {
  fprintf(stderr,"Activation du contexte graphique impossible.\n");
  exit(0);
  }


// Toutes les commandes precedentes sont OK, toutes les instructions
// OpenGL vont desormais s'executer dans le buffer offscreen

// Initialisations Open GL
glClearColor(255,255,255,0);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glEnable(GL_CULL_FACE);
glEnable(GL_DEPTH_TEST);
glShadeModel (GL_FLAT);

// Projection Perspective
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(45.0, (GLfloat)w/(GLfloat)h, 1.0, 2000.0);

// on passe tout dans le repere de la camera
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

// position de la camera
```

```
glPushMatrix();
gluLookAt (10, 8, 5, 0, 0, 0, 0, 1, 0);

// Trace des 6 faces d'un cube 3D
glColor3f(1, 1, 0);
glBegin(GL_POLYGON);
glVertex3f(-2,  2, -2);
glVertex3f( 2,  2, -2);
glVertex3f( 2, -2, -2);
glVertex3f(-2, -2, -2);
glEnd();
glColor3f(0, 1, 0);
glBegin(GL_POLYGON);
glVertex3f(-2, -2, 2);
glVertex3f( 2, -2, 2);
glVertex3f( 2,  2, 2);
glVertex3f(-2,  2, 2);
glEnd();
glColor3f(0, 0, 1);
glBegin(GL_POLYGON);
glVertex3f(-2, -2, -2);
glVertex3f(-2, -2,  2);
glVertex3f(-2,  2,  2);
glVertex3f(-2,  2, -2);
glEnd();
glColor3f(1, 0, 0);
glBegin(GL_POLYGON);
glVertex3f( 2,  2, -2);
glVertex3f( 2,  2,  2);
glVertex3f( 2, -2,  2);
glVertex3f( 2, -2, -2);
glEnd();
glColor3f(0, 1, 1);
glBegin(GL_POLYGON);
glVertex3f(-2,  2, -2);
glVertex3f(-2,  2,  2);
glVertex3f( 2,  2,  2);
glVertex3f( 2,  2, -2);
glEnd();
glColor3f(1, 0, 1);
glBegin(GL_POLYGON);
glVertex3f( 2, -2, -2);
glVertex3f( 2, -2,  2);
glVertex3f(-2, -2,  2);
glVertex3f(-2, -2, -2);
glEnd();
glFlush();

// allocation du buffer de recuperation du resultat offscreen
pixels = (unsigned char *)malloc(sizeof(unsigned char)*w*h*3);

// Lecture du buffer resultat
glReadPixels(0,0,w,h,GL_RGB,GL_UNSIGNED_BYTE, pixels);
```

```
  // ecriture dans un fichier image au format PPM
  // (affichable avec xv3.10)
  file = fopen("pbuff.ppm","wb");
  fprintf(file,"P6\n%d %d\n255\n", w, h);
  fwrite(pixels,w*h*3*sizeof(unsigned char),1,file);
  fclose(file);

  // Destruction P-Buffer
  glXDestroyGLXPbufferSGIX(dpy, pbuffer);
  // Destruction contexte graphique
  glXDestroyContext(dpy, cx);
  // Fermeture du display
  XCloseDisplay(dpy);
}
```

Annexe C

Exemple de format de fichier employé par *Phoenix*

```
#Inventor V2.0 ascii

# Description de la camera
SceneCamera
{
   fields [ SFString Name, SFString RealImage,  SFVec2f Offset,
            SFString Resolution, SFVec3f Position, SFVec3f LookAt,
            SFVec3f VUP, SFFloat FOV, MFMatrix Transform]
   # nom de l'image resultat
   Name     "result.ppm"
   # nom de l'image reelle
   RealImage "mes\_cubes_\_org.ppm"
   # position centre optique de la camera
   Offset 0 0
   # resolution de l'image resultat (non utilise si rendu realiste a
   # base d'images reelles
   Resolution "512x512"
   # position de la camera
   Position 278 273 -800
   # position d'un point de direction de regard de la camera
   LookAt   278 273 -700
   # orientation de la camera
   VUP      0 1 0
   # angle d'ouverture de la camera (degres)
   FOV      40.0
   # Transformation geometrique eventuelle de la camera
   Transform [1 0 0 0
              0 1 0 0
              0 0 1 0
              0 0 0 1 ]
}

#  Objet source de lumiere
Group
{
   Separator
```

```
      {
      Material { diffuseColor 1.000000 1.00000 1.000000 }

      Coordinate3 { point [ 443.0 388.78 77.0,
                            443.0 448.78 182.0,
                            313.0 448.78 182.0,
                            313.0 388.78 77.0 ] }

      IndexedFaceSet { coordIndex [0, 1, 2, -1, 0, 2, 3, -1] }

      # Proprietes photometriques de la source de lumiere
      # N'est PAS employe si une image reelle a ete specifie avec RealImage
      # dans SceneCamera precedent
      RadiationProperties
       {
       fields [ MFColor diffuse_reflectance, # reflectance diffuse
                  MFColor specular_reflectance, # reflectance speculaire
                  MFVec3f aniso_direction,   # direction d'anisotropie
                  SFFloat glossy_ax,    # parametre alpha_x d'anisotropie
                  SFFloat glossy_ay,    # parametre alpha_y d'anisotropie
                  MFColor emittance ]    # emittance de l'objet
                  diffuse_reflectance  1.0 1.0 1.0
                  emittance           1.0 1.0 1.0
         }
      }
}


# Objet Sol (anisotrope)
Group
{
   Separator
      {
        Material { diffuseColor 0.0 0.0 0.9 }

        Coordinate3 { point [ 552.8 0.0   0.0,
                              0.0   0.0   0.0,
                              0.0   0.0 559.2,
                              549.6 0.0 559.2 ] }

        IndexedFaceSet { coordIndex [0, 1, 2, -1, 0, 2, 3, -1] }

        RadiationProperties
         {
         fields [ MFColor diffuse_reflectance, # reflectance diffuse
                  MFColor specular_reflectance, # reflectance speculaire
                  MFVec3f aniso_direction,   # direction d'anisotropie
                  SFFloat glossy_ax,    # parametre alpha_x d'anisotropie
                  SFFloat glossy_ay,    # parametre alpha_y d'anisotropie
                  MFColor emittance ]    # emittance de l'objet
                  diffuse_reflectance  0.2 0.2 0.2
                  specular_reflectance 0.6 0.6 0.6
                  glossy_ax            0.05
                  glossy_ay            0.02
```

```
         aniso_direction      1.0 0.0 0.0
         }
     }
}


# Petit cube (diffus pur)
Group
{
   Separator
      {
      Material { diffuseColor 0.0 0.8 0.3 }

      Coordinate3 { point [ 223 140 247,
                            65 140 296,
                            114 140 456,
                            272 140 406 ]}

      IndexedFaceSet { coordIndex [ 0, 1, 2, -1, 0, 2, 3, -1 ] }

      RadiationProperties
         {
         fields [ MFColor diffuse_reflectance, # reflectance diffuse
                  MFColor specular_reflectance, # reflectance speculaire
                  MFVec3f aniso_direction,   # direction d'anisotropie
                  SFFloat glossy_ax,    # parametre alpha_x d'anisotropie
                  SFFloat glossy_ay,    # parametre alpha_y d'anisotropie
                  MFColor emittance ]    # emittance de l'objet
                  diffuse_reflectance  0 0.76 0.26
          }
      }


   Separator
      {
      Material { diffuseColor 0.0 0.8 0.3 }

      Coordinate3 { point [ 223 0.09 247,
                            223 140 247,
                            272 140 406,
                            272 0.09 406 ]}

      IndexedFaceSet { coordIndex [ 0, 1, 2, -1, 0, 2, 3, -1 ] }

      RadiationProperties
         {
         fields [ MFColor diffuse_reflectance, # reflectance diffuse
                  MFColor specular_reflectance, # reflectance speculaire
                  MFVec3f aniso_direction,   # direction d'anisotropie
                  SFFloat glossy_ax,    # parametre alpha_x d'anisotropie
                  SFFloat glossy_ay,    # parametre alpha_y d'anisotropie
                  MFColor emittance ]    # emittance de l'objet
                  diffuse_reflectance  0 0.76 0.26
          }
```

```
    }

  Separator
    {
    Material { diffuseColor 0.0 0.8 0.3 }

    Coordinate3 { point [ 272 0.09 406,
                          272 140 406,
                          114 140 456,
                          114 0.09 456  ]}

    IndexedFaceSet { coordIndex [ 0, 1, 2, -1, 0, 2, 3, -1 ] }

    RadiationProperties
       {
       fields [ MFColor diffuse_reflectance, # reflectance diffuse
                MFColor specular_reflectance, # reflectance speculaire
                MFVec3f aniso_direction,   # direction d'anisotropie
                SFFloat glossy_ax,    # parametre alpha_x d'anisotropie
                SFFloat glossy_ay,    # parametre alpha_y d'anisotropie
                MFColor emittance ]    # emittance de l'objet
                diffuse_reflectance  0 0.76 0.26
        }
    }

  Separator
    {
    Material { diffuseColor 0.0 0.8 0.3 }

    Coordinate3 { point [ 114 0.09 456,
                          114 140 456,
                          65 140 296,
                          65 0.09 296  ]}

    IndexedFaceSet { coordIndex [ 0, 1, 2, -1, 0, 2, 3, -1 ] }

    RadiationProperties
       {
       fields [ MFColor diffuse_reflectance, # reflectance diffuse
                MFColor specular_reflectance, # reflectance speculaire
                MFVec3f aniso_direction,   # direction d'anisotropie
                SFFloat glossy_ax,    # parametre alpha_x d'anisotropie
                SFFloat glossy_ay,    # parametre alpha_y d'anisotropie
                MFColor emittance ]    # emittance de l'objet
                diffuse_reflectance  0 0.76 0.26
        }
    }

  Separator
    {
    Material { diffuseColor 0.0 0.8 0.3 }

    Coordinate3 { point [ 223 0.09 247,
                          272 0.09 406,
```

```
                        114 0.09 456,
                        65 0.09 296  ]}

      IndexedFaceSet { coordIndex [ 0, 1, 2, -1, 0, 2, 3, -1 ] }

      RadiationProperties
         {
         fields [ MFColor diffuse_reflectance, # reflectance diffuse
                  MFColor specular_reflectance, # reflectance speculaire
                  MFVec3f aniso_direction,   # direction d'anisotropie
                  SFFloat glossy_ax,    # parametre alpha_x d'anisotropie
                  SFFloat glossy_ay,    # parametre alpha_y d'anisotropie
                  MFColor emittance ]    # emittance de l'objet
                  diffuse_reflectance  0 0.76 0.26
          }
      }

   Separator
      {
      Material { diffuseColor 0.0 0.8 0.3 }

      Coordinate3 { point [ 65 0.09 296,
                        65 140 296,
                        223 140 247,
                        223 0.09 247  ]}

      IndexedFaceSet { coordIndex [ 0, 1, 2, -1, 0, 2, 3, -1 ] }

      RadiationProperties
         {
         fields [ MFColor diffuse_reflectance, # reflectance diffuse
                  MFColor specular_reflectance, # reflectance speculaire
                  MFVec3f aniso_direction,   # direction d'anisotropie
                  SFFloat glossy_ax,    # parametre alpha_x d'anisotropie
                  SFFloat glossy_ay,    # parametre alpha_y d'anisotropie
                  MFColor emittance ]    # emittance de l'objet
                  diffuse_reflectance  0 0.76 0.26
          }
      }
}
```

Annexe D

Résultats numériques sur une scène synthétique

Objectifs

Nous proposons dans cette annexe d'exposer les résultats visuels et numériques obtenus lors de la simulation d'une scène synthétique en rendu inverse [1]. Tout d'abord, nous avons calculé une image de synthèse assez simple (voir figure D.1) avec notre logiciel de rendu réaliste *Phoenix* (voir chapitre 4). À l'aide de cette image et du modèle géométrique 3D employé pour la générer (y compris les paramètres de la caméra), nous allons regénérer cette image par rendu inverse. Nous montrons ici que notre algorithme permet non seulement d'obtenir une approximation visuelle de l'image d'origine, mais qu'il permet aussi de retrouver les paramètres **exacts** des BRDF utilisées pour produire l'image originale.

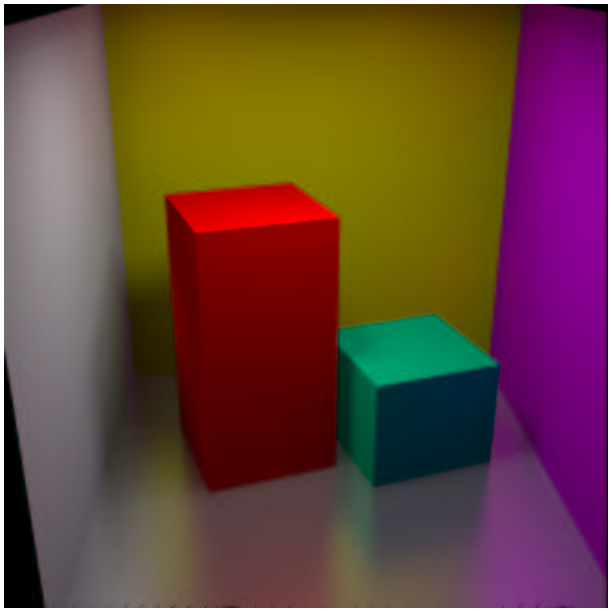

FIG. D.1 – Image de référence générée avec *Phoenix*. Le sol a été simulé comme une surface anisotrope.

[1] Bien que cela constitue une validation scientifique supplémentaire de notre méthode, il est généralement plus intéressant d'utiliser nos algorithmes sur des images réelles.

Résultats

Le premier niveau de hiérarchie (diffus pur) a permis de calculer les paramètres de réflectance après 4 itérations (les itérations suivantes ne modifiant pas la réflectance de façon significative[2]). Les valeurs pour ρ_d obtenues sont récapitulées plus loin dans le tableau de la figure D.4. Les niveaux suivants de la hiérarchie (spéculaire, spéculaire non-parfait et diffus/spéculaire sans rugosité) produisent des erreurs élevées sur la surface anisotrope, ce qui permet à l'algorithme d'essayer l'hypothèse d'isotropie.

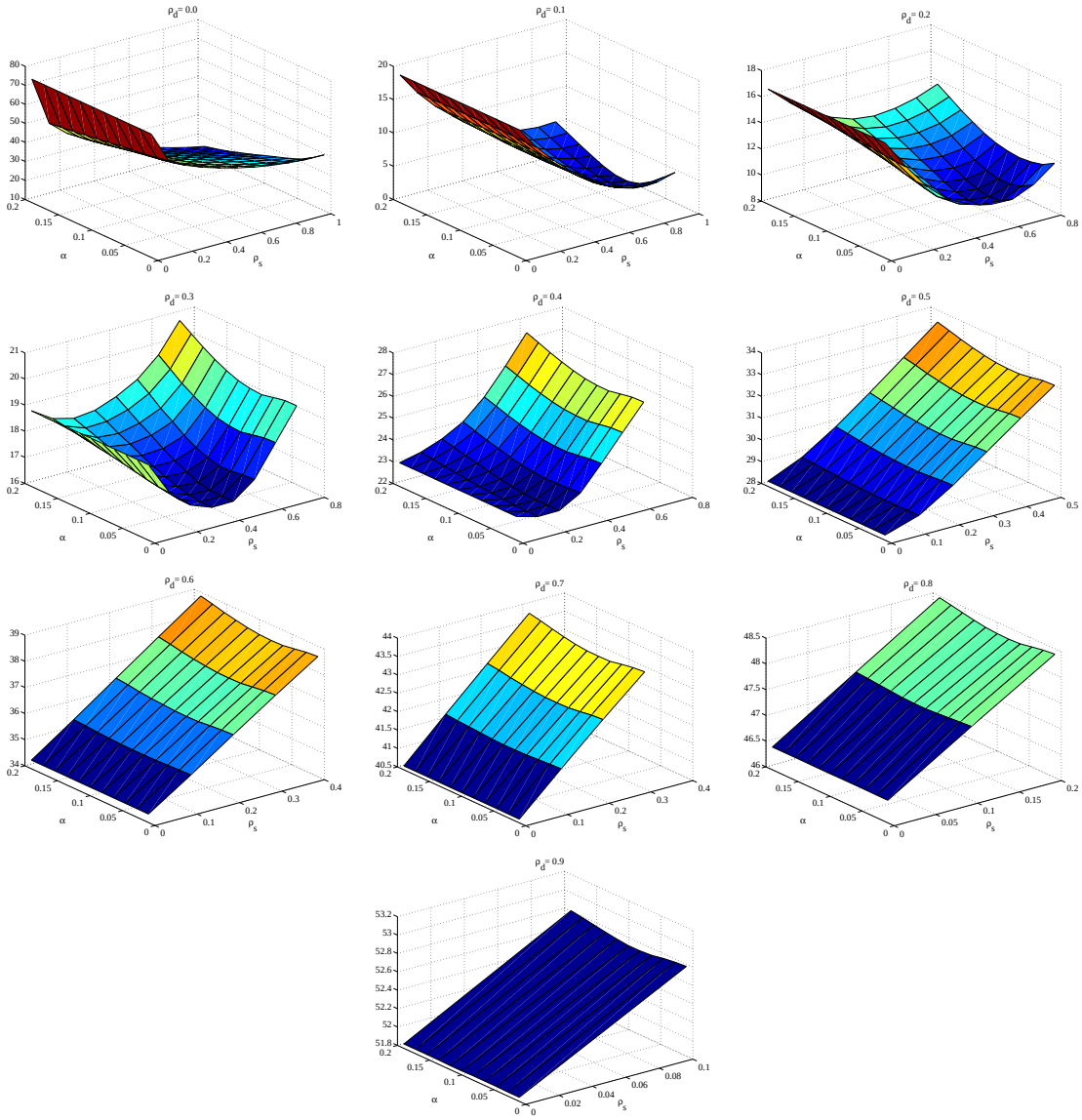

FIG. D.2 – Fonction d'erreur (Image Synthétique - Image Réelle), pour une réflectance diffuse ρ_d fixée, et suivant les valeurs d'isotropie α, et de réflectance spéculaire ρ_s.

Lors de cette hypothèse d'isotropie, l'algorithme de minimisation par la méthode du simplexe [88] trouve un minimum global pour la valeur d'isotropie $\alpha = 0.082$ pour des valeurs respectives

[2]Les variations sont de l'ordre de 10^{-4}.

de ρ_d et ρ_s de $(0.10013, 0.10045, 0.09981)$ et $(0.89909, 0.90102, 0.89903)$. Ces valeurs sont confirmées par la fonction d'erreur représentée par les courbes de la figure D.2, calculées ici pour la composante rouge. L'erreur produite par la différence entre l'image d'origine et l'image regénérée étant supérieure à 1% (voir image droite de la figure D.3), celle-ci force l'algorithme à rechercher une BRDF anisotrope[3].

FIG. D.3 – À gauche : sol anisotrope de l'image d'origine. Au centre : sol simulé de façon isotrope dans l'image synthétique. À droite : Image des erreurs issue de la différence entre l'image originale et l'image regénérée.

En échantillonnant pour la zone d'intérêt les directions d'anisotropie, et en calculant la moyenne des écarts types sur l'image des erreurs dans cette zone (voir image gauche de la figure D.6) pour chacune de ces directions (voir figure 6.25), nous pouvons construire la courbe de la figure droite D.6, qui a un minimum global en $\theta = 5$ degrés. Cette valeur est ici calculée dans l'image d'origine, et en la représentant dans le repère propre de la surface anisotrope (on obtient alors une valeur de $\theta = 2.8$ degrés), on s'aperçoit qu'elle constitue en fait une erreur d'estimation d'environ 0.8% par rapport à la véritable valeur ($\theta = 0$ degré). En minimisant la fonction d'erreur en α_x et α_y (voir courbes de la figure D.5) pour la direction d'anisotropie trouvée, on obtient α_x et α_y valant respectivement 0.0699 et 0.1101, soit les valeurs employées pour produire l'image d'origine.

Le tableau de la figure D.4 résume les différents paramètres des BRDF trouvées par rapport à l'image d'origine, et l'image gauche de la figure D.7 est l'image obtenue avec ces paramètres.

Surface	Paramètre	valeur réelle	valeur estimée	Remarque
mur gauche	ρ_d	(0.66, 0.66, 0.66)	(0.65916, 0.66075, 0.66037)	
	ρ_s	(0.0, 0.0, 0.0)	(0.0, 0.0, 0.0)	
mur droit	ρ_d	(0.69, 0, 0.95)	(0.69002, 0.0, 0.95901)	
	ρ_s	(0.0, 0.0, 0.0)	(0.0, 0.0, 0.0)	
mur du fond	ρ_d	(0.65, 0.65, 0.0)	$(0.64997, 0.65067, 4 \cdot 10^{-7})$	
	ρ_s	(0.0, 0.0, 0.0)	(0.0, 0.0, 0.0)	
plafond	ρ_d	(1.0, 1.0, 1.0)	(1.0, 1.0, 1.0)	Voir la note de
	ρ_s	(0.0, 0.0, 0.0)	(0.0, 0.0, 0.0)	bas de page[4].
grand bloc	ρ_d	(0.77, 0.0, 0.0)	$(0.77002, 2 \cdot 10^{-4}, 3 \cdot 10^{-6})$	
	ρ_s	(0.0, 0.0, 0.0)	(0.0, 0.0, 0.0)	
petit bloc	ρ_d	(0.0, 0.76, 0.26)	(0.0, 0.75802, 0.25912)	
	ρ_s	(0.0, 0.0, 0.0)	(0.0, 0.0, 0.0)	
sol	ρ_d	(0.1, 0.1, 0.1)	(0.10013, 0.10045, 0.09981)	
	ρ_s	(0.9, 0.9, 0.9)	(0.89909, 0.90102, 0.89903)	
	θ	0.0^o	2.8^o	Soit une erreur
	α_x	0.07	0.06999	de 0.77% pour θ.
	α_y	0.11	0.1101	

FIG. D.4 – Tableau récapitulatif des paramètres de réflectance estimés depuis une image originale synthétique.

[3] On remarquera que bien que le sol soit réellement anisotrope dans l'image d'origine, l'image centrale de la figure D.3 produite par l'hypothèse d'isotropie est déjà extrêmement proche de l'image d'origine.

[4] Le plafond n'étant pas visible dans l'image d'origine, l'algorithme l'initialise à (1.0, 1.0, 1.0) par défaut. En

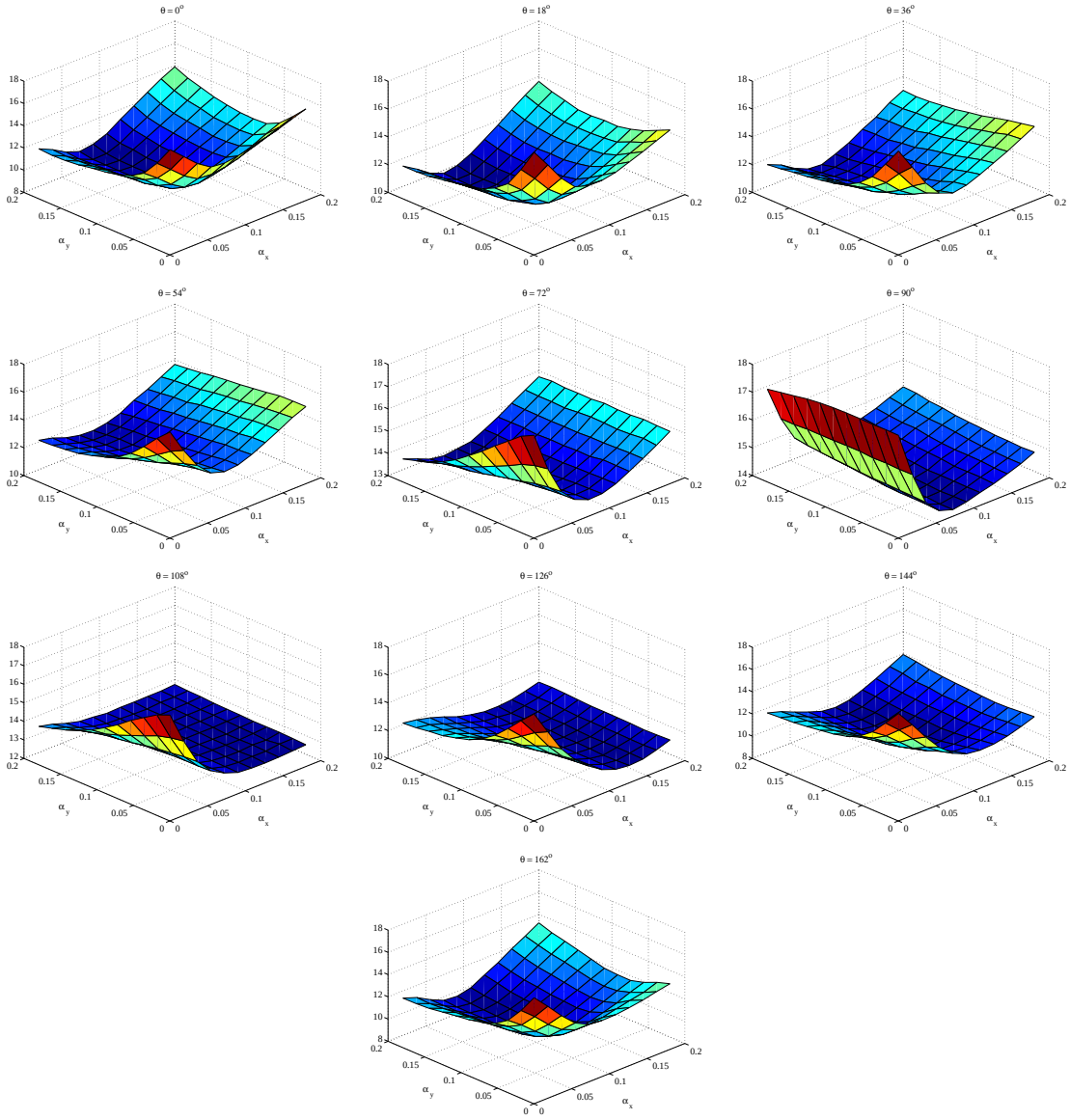

FIG. D.5 – Fonction d'erreur (Image Synthétique - Image Réelle), pour différentes directions d'anisotropie $\vec{x}$ et suivant des valeurs d'anisotropie α_x, α_y (la réflectance diffuse ρ_d et la réflectance spéculaire ρ_s ayant été calculées par l'étape précédente d'isotropie).

pratique, si un tel cas devait arriver, l'utilisateur devrait trouver un objet visible dans l'image réelle et ayant des propriétés similaires au plafond, pour l'inclure dans le même groupe. Dans le cas contraire, les valeurs de réflectance obtenues permettraient tout de même de produire une image très proche de l'image d'origine, mais toutes les réflectances seraient biaisées par l'initialisation arbitraire.

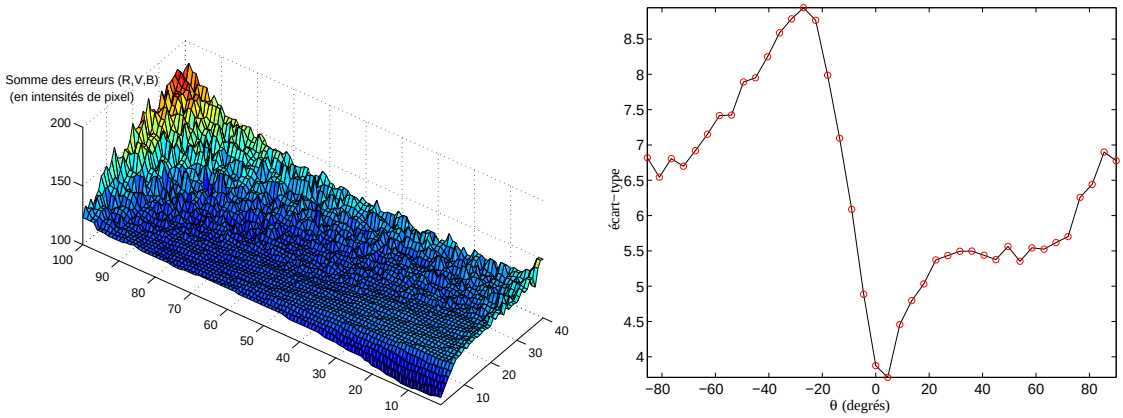

FIG. D.6 – À gauche : la surface 3D représente l'image des différences entre la réflexion virtuelle du petit cube vert sur le sol simulé de façon spéculaire parfait, et la réflexion du cube sur le même sol (anisotrope) dans l'image d'origine. À droite : la courbe 2D des moyennes des écarts types, en fonction de la direction d'anisotropie (sens de parcours de l'image des erreurs).

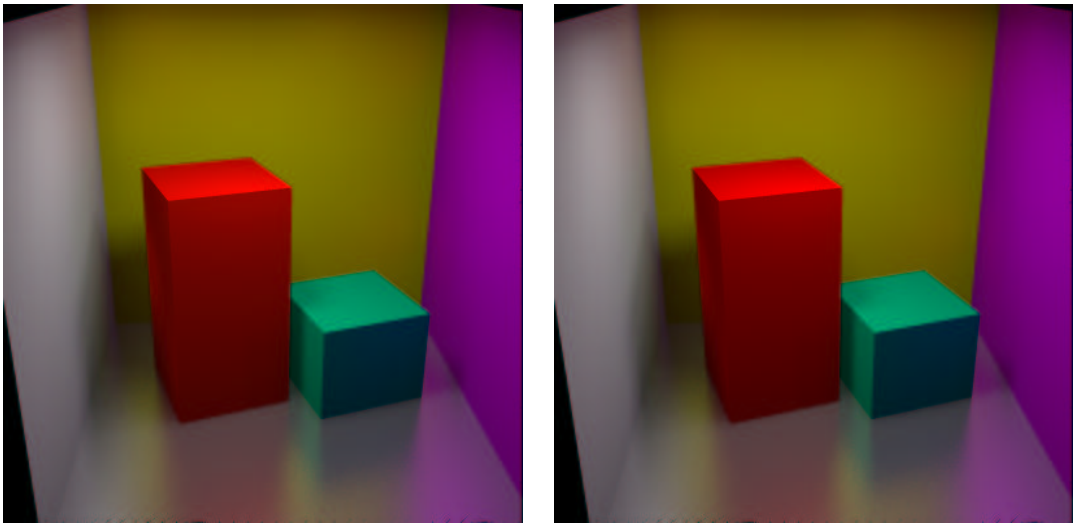

FIG. D.7 – À gauche, l'image synthétique d'origine. À droite, l'image synthétique regénérée avec les paramètres calculés du tableau de la figure D.4. L'image des erreurs issue de la différence entre ces deux images n'est pas montrée ici, car elle est uniformément "bleue" et présente donc peu d'intérêt.

Bibliographie

[1] E.H. Adelson and J.R. Bergen. *Computational Models of Visual Processing.* MIT Press, Cambridge, 1991.

[2] John M. Airey and M. Ouh-young. Two adaptive techniques let progressive radiosity outperform the traditional radiosity algorithm. Technical Report R89-020, Univ. of North Carolina Department of Computer Science, 1989.

[3] John M. Airey, John H. Rohlf, and Jr. Frederick P.Brooks. Towards image realism with interactive update rates in complex virtual building environments. In Rich Riesenfeld and Carlo Sequin, editors, *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, volume 24, pages 41–50, March 1990.

[4] John Amanatides and Andrew Woo. A fast voxel traversal algorithm for ray tracing. *Eurographics '87*, pages 3–10, August 1987.

[5] Arthur Appel. Some techniques for shading machine renderings of solids. *AFIPS 1968 Spring Joint Computer Conf.*, 32 :37—45, 1968.

[6] James Arvo and David B. Kirk. Fast ray tracing by ray classification. *Computer Graphics (Proceedings of SIGGRAPH 87)*, 21(4) :55–64, July 1987. Held in Anaheim, California.

[7] Ian Ashdown. *Radiosity : A Programmer's Perspective.* John Wiley & Sons, New York, 1994.

[8] M. Ashikhmin, S. Premoze, and P. Shirley. A microfacet-based brdf generator. In K. Akeley, editor, *Computer Graphics (ACM SIGGRAPH '2000 Proceedings)*, pages 65–75. Addison Wesley Longman, July 2000.

[9] L. Aupperle and Pat Hanrahan. A hierarchical illumination algorithm for surfaces with glossy reflection. In *Computer Graphics Proceedings, Annual Conference Series, 1993*, pages 155–162, 1993.

[10] Didier Badouel. An efficient ray-polygon intersection. *Graphics Gems*, pages 390–393, 735, 1990.

[11] R. Bajcsy, S.W. Lee, and A. Leonardis. Detection of diffuse and specular reflections and inter-reflections by color image segmentation. In *International Conference on Computer Vision*, pages 241–272, 1996.

[12] R. Baribeau, M. Rioux, and G. Godin. Color reflectance modeling using a polychromatic laser range sensor. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2) :263–269, February 1992.

[13] Daniel R. Baum, Stephen Mann, Kevin P. Smith, and James M. Winget. Making radiosity usable : Automatic preprocessing and meshing techniques for the generation of accurate radiosity solutions. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 51–60, July 1991.

[14] Daniel R. Baum, Holly E. Rushmeier, and James M. Winget. Improving radiosity solutions through the use of analytically determined form-factors. In Jeffrey Lane, editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 325–334, July 1989.

[15] P. Beckmann. Shadowing of random rough surfaces. *IEEE Transactions on Antennas and Propagation*, pages 384–388, May 1965.

[16] P. Beckmann and A. Spizzichino. *The Scattering of Electromagnetic Waves from Rough Surfaces*. Pergamon Press, New York, 1963.

[17] P. Beckmann and A. Spizzichino. *The Scattering of Electromagnetic Waves from Rough Surfaces*. Artech House, Norwood, 1987.

[18] Philippe Bekaert. *Hierarchical and stochastic algorithms for radiosity*. PhD thesis, Department of Computer Science, Katholieke Universiteit Leuven, Belgium, 1999.

[19] Jeffrey C. Beran-Koehn. A cubic tetrahedral adaptation of the hemi-cube algorithm. *Graphics Gems II*, pages 299–302, 1991. ISBN 0-12-064481-9. Held in Boston.

[20] Jeffrey C. Beran-Koehn. Delta form-factor calculation for the cubic tetrahedral algorithm. *Graphics Gems III*, pages 324–328, 575–576, 1992. ISBN 0-12-409673-5. Held in Boston.

[21] Jacques Blanc-Talon and Dan Popescu. *Advances in Computation : Therory and Practice*, chapter Advanced Computer Graphics and Vision Collaboration Techniques for Image-Based Rendering (Samuel Boivin and André Gagalowicz). Nova Science Books and Journals, New York, NY., 2001. To appear.

[22] James F. Blinn. Models of light reflection for computer synthesized pictures. In James George, editor, *Computer Graphics (SIGGRAPH '77 Proceedings)*, volume 11, pages 192–198, July 1977.

[23] Samuel Boivin. Recherche et développement d'algorithmes de rendu réaliste dans le cadre d'une coopération analyse/synthèse. In *Proceedings of AFIG'95*, pages 337–347, 1995.

[24] Samuel Boivin and Laroussi Doghman. A new rendering technique for the realistic simulation of natural scenes. In *Proceedings of IMAGECOM'96*, pages 302–307, 1996.

[25] Samuel Boivin and André Gagalowicz. Image-based rendering of diffuse, specular, and anisotropic surfaces from a single image. *accepté pour publication à SIGGRAPH 2001, à paraître*, –(–), August 2001.

[26] Samuel Boivin and André Gagalowicz. Image-based rendering from a single image. Rapport de Recherche, RR-4098, INRIA-Rocquencourt, Janvier 2001.

[27] G. Bruhat and A. Kastler. *Optique Sixième Edition*. Masson, Paris, France, 1992.

[28] V. Cantoni, M. Ferretti, S. Levialdi, R. Negrini, and R. Stefanelli. *Progress in Image Analysis and Processing II*, chapter Collaboration between Computer Graphics and Computer Vision (André Gagalowicz and Jean-Marc Vézien), pages 229–258. World Publishing, Singapore, 1992.

[29] Tolga K. Çapin, Cevdet Aykanat, and Bülent Özgüç. Progressive refinement radiosity on ring-connected multicomputers. In Thomas Crockett, Charles Hansen, and Scott Whitman, editors, *ACM SIGGRAPH Symposium on Parallel Rendering*. ACM, November 1993.

[30] Loren Carpenter. The a-buffer, an antialiased hidden surface method. *Computer Graphics (Proceedings of SIGGRAPH 84)*, 18(3) :103–108, July 1984. Held in Minneapolis, Minnesota.

[31] E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. Ph.D. thesis, University of Utah, Salt Lake City, UT, December 1974.

[32] Alan Chalmers and Derek Paddon. Parallel processing of progressive refinement radiosity methods. In *Eurographics Workshop on Rendering*, 1991.

[33] Alan G. Chalmers and Derek J. Paddon. Parallel processing of progressive refinement radiosity methods. In *Photorealistic Rendering in Computer Graphics (Proceedings of the Second Eurographics Workshop on Rendering)*, pages 149–159, New York, 1994. Springer-Verlag.

[34] Shenchang Eric Chen. A progressive radiosity method and its implementation in a distributed processing environment. Master's thesis, Program of Computer Graphics, Cornell Univ., January 1989.

[35] John G. Cleary and Geoff Wyvill. Analysis of an algorithm for fast ray tracing using uniform space subdivision. *The Visual Computer*, 4(2) :65–83, July 1988.

[36] Daniel Cohen. Voxel traversal along a 3d line. *Graphics Gems IV*, pages 366–369, 1994. ISBN 0-12-336155-9. Held in Boston.

[37] M. F. Cohen, D. P. Greenberg, D. S. Immel, and P. J. Brock. An efficient radiosity approach for realistic image synthesis. *IEEE Computer Graphics and Applications*, 6(3) :26–35, March 1986.

[38] Michael F. Cohen, Shenchang Eric Chen, John R. Wallace, and Donald P. Greenberg. A progressive refinement approach to fast radiosity image generation. In John Dill, editor, *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 75–84, August 1988.

[39] Michael F. Cohen and Donald P. Greenberg. The Hemi-Cube : A radiosity solution for complex environments. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 31–40, August 1985.

[40] Michael F. Cohen and John R. Wallace. *Radiosity and Realistic Image Synthesis*. Academic Press Professional, San Diego, CA, 1993.

[41] R. L. Cook and K. E. Torrance. A reflectance model for computer graphics. In *Computer Graphics (SIGGRAPH '81 Proceedings)*, volume 15, pages 307–316, August 1981.

[42] R. L. Cook and K. E. Torrance. A reflectance model for computer graphics. *ACM Transactions on Graphics*, 1(1) :7–24, January 1982.

[43] Kristin J. Dana, Bram van Ginneken, Shree K. Nayar, and Jan J. Koenderink. Reflectance and texture of real-world surfaces. *ACM Transactions on Graphics*, 18(1) :1–34, January 1999. ISSN 0730-0301.

[44] P. Debevec, C.J. Taylor, and J. Malik. Modeling and rendering architecture from photographs : A hybrid geometry- and image-based approach. Technical Report UCB/CSD-96-893, Computer Science Division, University of California at Berkeley, January 1996.

[45] Paul Debevec. Rendering synthetic objects into real scenes : Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In Michael Cohen, editor, *Proceedings of SIGGRAPH 98*, Computer Graphics Proceedings, Annual Conference Series, pages 189–198. Addison Wesley, July 1998.

[46] Paul E. Debevec and Jitendra Malik. Recovering high dynamic range radiance maps from photographs. In Turner Whitted, editor, *Proceedings of SIGGRAPH 97*, Computer Graphics Proceedings, Annual Conference Series, pages 369–378. Addison Wesley, August 1997.

[47] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs : A hybrid geometry- and image-based approach. In Holly Rushmeier, editor, *Proceedings of SIGGRAPH 96*, Computer Graphics Proceedings, Annual Conference Series, pages 11–20. Addison Wesley, August 1996.

[48] Paul E. Debevec, Yizhou Yu, and George D. Borshukov. Efficient view-dependent image-based rendering with projective texture-mapping. In George Drettakis and Nelson Max, editors, *Eurographics Rendering Workshop 1998*, pages 105–116. Eurographics, June 1998.

[49] Paul Ernest Debevec. *Modeling and Rendering Architecture from Photographs*. PhD thesis, University of California, Berkeley, 1996.

[50] D.F DeMenthon and L. Davis. Model-based object pose in 25 lines of code. In *Second European Conference on Computer Vision(ECCV)*, pages 335–343. Springer-Verlag, May 1992.

[51] Jeremy Denise. Etude et implémentation d'un modèle d'interaction lumière/matière. Master's thesis, Equipe Synthèse d'Images de l'Institut Gaspard Monge, Université de Marne-La-Vallée, January 1995.

[52] François Desvignes. *Rayonnements Optiques, 2ème Edition*. Masson, Paris, France, 1997.

[53] Mark A. Z. Dippé and Erling Henry Wold. Antialiasing through stochastic sampling. *Computer Graphics (Proceedings of SIGGRAPH 85)*, 19(3) :69–78, July 1985. Held in San Francisco, California.

[54] George Dretakkis and Eugene Fiume. A fast shadow algorithm for area light sources using backprojection. *Proceedings of SIGGRAPH 94*, pages 223–230, July 1994. ISBN 0-89791-667-0. Held in Orlando, Florida.

[55] George Drettakis and Eugene L. Fiume. Structured penumbral irradiance computation. *IEEE Transactions on Visualization and Computer Graphics*, 2(4), December 1996. ISSN 1077-2626.

[56] George Drettakis, Luc Robert, and Sylvain Bougnoux. Interactive common illumination for computer augmented reality. In Julie Dorsey and Philipp Slusallek, editors, *Eurographics Rendering Workshop 1997*, pages 45–56. Springer Wien, June 1997.

[57] George Drettakis and François X. Sillion. Interactive update of global illumination using a line-space hierarchy. *Proceedings of SIGGRAPH 97*, pages 57–64, August 1997. ISBN 0-89791-896-7. Held in Los Angeles, California.

[58] Steven M. Drucker and Peter Schroeder. Fast radiosity using a data parallel architecture. *Third Eurographics Workshop on Rendering*, pages 247–258, May 1992.

[59] Frédo Durand. *Visibilité tridimensionnelle : étude analytique et applications*. Ph.D. thesis, Université Joseph Fourier, Grenoble, France, July 1999.

[60] Frédo Durand, George Drettakis, and Claude Puech. Fast and accurate hierarchical radiosity using global visibility. *ACM Transactions on Graphics*, 18(2) :128–170, April 1999. ISSN 0730-0301.

[61] Martin Feda and Werner Purgathofer. Progressive refinement radiosity on a transputer network. In *Eurographics Workshop on Rendering*, 1991.

[62] Martin Feda and Werner Purgathofer. Accelerating radiosity by overshooting. *Third Eurographics Workshop on Rendering*, pages 21–32, May 1992.

[63] E. Fiume, A. Fournier, and L. Rudolph. A parallel scan conversion algorithm with anti-aliasing for a general purpose ultracomputer. *Computer Graphics (Proceedings of SIGGRAPH 83)*, 17(3) :141–150, July 1983. Held in Detroit, Michigan.

[64] Eugene Fiume, Alain Fournier, and Larry Rudolph. A parallel scan conversion algorithm with anti-aliasing for a general-purpose ultracomputer : Preliminary report. *Graphics Interface '83*, pages 11–21, May 1983.

[65] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics : Principles and Practices (2nd Edition)*. Addison Wesley, 1990.

[66] Alain Fournier, Atjeng S. Gunawan, and Chris Romanzin. Common illumination between real and computer generated scenes. In *Graphics Interface '93*, pages 254–262. Canadian Information Processing Society, May 1993. Held in Toronto, Ontario, Canada.

[67] Alain Fournier and Pierre Poulin. A ray tracing accelerator based on a hierarchy of 1d sorted lists. *Graphics Interface '93*, pages 53–61, May 1993. Held in Toronto, Ontario.

[68] H. Fuchs, Z. M. Kedem, and B. F. Naylor. On visible surface generation by a priority tree structures. In *Computer Graphics (SIGGRAPH '80 Proceedings)*, volume 14, pages 124–133, July 1980.

[69] Akira Fujimoto and Kansei Iwata. Accelerated ray tracing. *Computer Graphics : Visual Technology and Art (Proceedings of Computer Graphics Tokyo '85)*, pages 41–65, 1985. Held in New York.

[70] Akira Fujimoto, Takayuki Tanaka, and Kansei Iwata. Arts : Accelerated ray-tracing system. *IEEE Computer Graphics & Applications*, pages 16–26, April 1986.

[71] André Gagalowicz. Coopération entre l'analyse et la synthèse d'images. In *Proceedings of the AFCET Conference*, pages 1727–1758, Paris, France, November 1989.

[72] André Gagalowicz. Collaboration between computer vision and computer graphics. In *Proceedings of the ICCV'90 Conference*, pages 733–737, Osaka, Japan, December 1990.

[73] André Gagalowicz. Tools for advanced telepresence systems. *Computers and Graphics*, 10(1), 1995.

[74] I. Gargantini and H. H. Atkinson. Ray tracing an octree : Numerical evaluation of the first interaction. *Computer Graphics Forum*, 12(4) :199–210, October 1993.

[75] Neil Gatenby. *Incorporating Hierarchical Radiosity into Discontinuity Meshing Radiosity.* Ph.D. thesis, University of Manchester, Manchester, UK, 1994.

[76] Neil Gatenby and W. T. Hewitt. Optimizing discontinuity meshing radiosity. In *Fifth Eurographics Workshop on Rendering*, pages 249–258, Darmstadt, Germany, June 1994.

[77] Jon Genetti and Dan Gordon. Ray tracing with adaptive supersampling in object space. *Graphics Interface '93*, pages 70–77, May 1993. Held in Toronto, Ontario.

[78] D. Ghazanfarpour and J.-M. Hasenfratz. A beam tracing method with precise antialiasing for polyhedral scenes. *Computers & Graphics*, 22(1) :103–115, February 1998. ISSN 0097-8493.

[79] Michael Gigante. Accelerated ray tracing using non-uniform grids. *Proceedings of Ausgraph '90*, pages 157–163, September 1990. Held in Melbourne, Australia.

[80] Andrew Glassner. *An Introduction to Ray Tracing.* Academic Press, New York, NY, 1989.

[81] Andrew S. Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics & Applications*, 4(10) :15–22, October 1984.

[82] Andrew S. Glassner. Space subdivision for fast ray tracing. *Tutorial : Computer Graphics : Image Synthesis*, pages 159–167, 1988.

[83] G. Golub and J.M. Ortega. *Scientific Computing with an Introduction to Parallel Computing.* Academic Press, Boston, MA.(USA), 1993.

[84] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. Modeling the interaction of light between diffuse surfaces. In Hank Christiansen, editor, *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pages 213–222, July 1984.

[85] Steven J. Gortler, Michael F. Cohen, and Phillip Slusallek. Radiosity and relaxation methods : Progressive refinement is southwell relaxation. Technical Report CS-TR-408-93, Department of Computer Science, Princeton University, February 1993.

[86] P. Guitton, J. Roman, and C. Schlick. Two parallel approaches for a progressive radiosity. In *Eurographics Workshop on Rendering*, 1991.

[87] P. Guitton, J. Roman, and Christophe Schlick. Two parallel approaches for a progressive radiosity. In *Photorealistic Rendering in Computer Graphics (Proceedings of the Second Eurographics Workshop on Rendering)*, pages 160–170, New York, 1994. Springer-Verlag.

[88] Press W. H., Teukolsky S.A., Vetterling W.T., and Flannery B.P. *Numerical Recipes in C, The Art of Scientific Computing*, chapter 10.4 Downhill Simplex Method in Multidimensions, pages 305–309. Cambridge University Press, Cambridge, 1992.

[89] Eric Haines. Efficiency improvements for hierarchy traversal in ray tracing. *Graphics Gems II*, pages 267–272, 1991. ISBN 0-12-064481-9. Held in Boston.

[90] Eric A. Haines and John R. Wallace. Shaft culling for efficient ray-traced radiosity. In *Photorealistic Rendering in Computer Graphics (Proceedings of the Second Eurographics Workshop on Rendering)*, New York, 1994. Springer-Verlag. also available via FTP from princeton.edu :/pub/Graphics/Papers.

[91] David E. Hall and Holly E. Rushmeier. Improved explicit radiosity method for calculating non-lambertian reflections. *The Visual Computer*, 9(5) :278–288, March 1993.

[92] R. Hall. *Illumination and Color in Computer Generated Imagery.* Springer-Verlag, 1989.

[93] Pat Hanrahan, David Salzman, and Larry Aupperle. A rapid hierarchical radiosity algorithm. *Computer Graphics (Proceedings of SIGGRAPH 91)*, 25(4) :197–206, July 1991.

[94] Parag Havaldar, Mi-Suen Lee, and Gérard Medioni. Synthesizing novel views from unregistered 2-d images. *Computer Graphics Forum*, 16(1) :65–73, 1997.

[95] Parag Havaldar, Mi-Suen Lee, and Gérard Medioni. View synthesis from unregistered 2-d images. In Wayne A. Davis and Richard Bartels, editors, *Graphics Interface '96*, pages 61–69. Canadian Human-Computer Communications Society, May 1996.

[96] Vlastimil Havran, Tomás Kopal, Jirî Bittner, and Jirî Zára. Fast robust bsp tree traversal algorithm for ray tracing. *Journal of Graphics Tools*, 2(4) :15–24, 1997. ISSN 1086-7651.

[97] Xiao D. He, Kenneth E. Torrance, Francois X. Sillion, and Donald P. Greenberg. A comprehensive physical model for light reflection. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 175–186, July 1991.

[98] Paul Heckbert. Discontinuity meshing for radiosity. *Third Eurographics Workshop on Rendering*, pages 203–226, May 1992.

[99] Paul S. Heckbert. Adaptive radiosity textures for bidirectional ray tracing. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 145–154, August 1990.

[100] Nicolas Holzschuch, Francois Sillion, and George Dretakkis. An efficient progressive refinement strategy for hierarchical radiosity. In *Fifth Eurographics Workshop on Rendering*, pages 343–357, Darmstadt, Germany, June 1994.

[101] David S. Immel, Michael F. Cohen, and Donald P. Greenberg. A radiosity method for non-diffuse environments. In David C. Evans and Russell J. Athay, editors, *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 133–142, August 1986.

[102] Nelder J.A. and Mead R. A simplex method for function minimization. *Computer Journal*, 7 :308–313, 1965.

[103] P. Jancène, F. Neyret, X. Provot, J-M Vézien, C. Meilhac, and A. Vérroust. Res : Computing the interactions between real and virtual objects in video sequences. In *Second IEEE Workshop on Networked Realities*, pages 27–40, October 1995.

[104] J. P. Jessel, M. Paulin, and R. Caubet. An extended radiosity using parallel ray-traced specular transfers. In *Eurographics Workshop on Rendering*, 1991.

[105] Jean-Pierre Jessel, Mathias Paulin, and Rene Caubet. An extended radiosity using parallel ray-traced specular transfers. In *Photorealistic Rendering in Computer Graphics (Proceedings of the Second Eurographics Workshop on Rendering)*, pages 171–181, New York, 1994. Springer-Verlag.

[106] David Jevans and Brian Wyvill. Adaptive voxel subdivision for ray tracing. *Graphics Interface '89*, pages 164–172, June 1989.

[107] Norman P. Jouppi and Chun-Fa Chang. Z3 : an economical hardware technique for high-quality antialiasing and transparency. *1999 SIGGRAPH / Eurographics Workshop on Graphics Hardware*, pages 85–93, August 1999. Held in Los Angeles, California.

[108] James T. Kajiya. The rendering equation. In David C. Evans and Russell J. Athay, editors, *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 143–150, August 1986.

[109] K.F. Karner, E.A. Deutschl, and F.W.Leberl. Comparison of different light reflection models. Technical Report Report 399, IIG-Report-Series, November 1994.

[110] Konrad F. Karner, Heinz Mayer, and Michael Gervautz. An image based measurement system for anisotropic reflection. *Computer Graphics Forum*, 15(3) :119–128, August 1996.

[111] D.C. Kay. Transparency for computer synthesized images. Master's thesis, Program of Computer Graphics, Cornell Univ., January 1979.

[112] G. Kay and T. Caelli. Inverting an illumination model from range and intensity maps. *CGVIP : Image Understanding*, 59 :183–201, 1994.

[113] Timothy L. Kay and James T. Kajiya. Ray tracing complex scenes. *Computer Graphics (Proceedings of SIGGRAPH 86)*, 20(4) :269–278, August 1986. Held in Dallas, Texas.

[114] David Kirk. Improved ray tagging for voxel-based ray tracing. *Graphics Gems II*, pages 264–266, 1991. ISBN 0-12-064481-9. Held in Boston.

[115] Kryzsztof S. Klimansezewski and Thomas W. Sederberg. Faster ray tracing using adaptive grids. *IEEE Computer Graphics & Applications*, 17(1) :42–51, January - February 1997. ISSN 0272-1716.

[116] K.J. Koestner. A wave based relfection model for realistic imafe synthesis. Master's thesis, Program of Computer Graphics, Cornell Univ., August 1986.

[117] A. J. F. Kok, A. C. Yilmaz, and L. H. J. Bierens. A two-pass radiosity method for bezier patches. In *Eurographics Workshop on Photosimulation, Realism and Physics in Computer Graphics*, 1989.

[118] Arjan Kok. Grouping of patches in progressive radiosity. In Michael F. Cohen, Claude Puech, and Francois Sillion, editors, *Fourth Eurographics Workshop on Rendering*, pages 221–232. Eurographics, June 1993. held in Paris, France, 14–16 June 1993.

[119] Arjan J. F. Kok, Celal Yilmaz, and Laurens H. J. Bierens. A two-pass radiosity method for bezier patches. In *Proceedings Eurographics Workshop on Photosimulation, Realism and Physics in Computer Graphics*, pages 117–26, Rennes, France, June 1990.

[120] P. Kowaliski. *Vision et Mesure de la Couleur, 2ème Edition.* Masson, Paris, France, 1990.

[121] Eric P. Lafortune, Sing-Choong Foo, Kenneth E. Torrance, and Donald P. Greenberg. Non-linear approximation of reflectance functions. In *Computer Graphics (ACM SIGGRAPH '97 Proceedings)*, volume 31, pages 117–126, 1997.

[122] Paul Lalonde. An adaptive discretization method for progressive radiosity. In *Proceedings of Graphics Interface '93*, pages 78–86, Toronto, Ontario, Canada, May 1993. Canadian Information Processing Society.

[123] Michael Langer, Pierre Breton, and Steven Zucker. Parallel radiosity without form factors. Tr-cim-93-22, McGill Research Center for Intelligent Machines, McGill University, Montreal, Quebec, December 1993.

[124] Mark E. Lee, Richard A. Redner, and Samuel P. Uselton. Statistically optimized sampling for distributed ray tracing. *Computer Graphics (Proceedings of SIGGRAPH 85)*, 19(3) :61–67, July 1985. Held in San Francisco, California.

[125] Robert Lewis. Making shaders more physically plausible. In Michael F. Cohen, Claude Puech, and Francois Sillion, editors, *Fourth Eurographics Workshop on Rendering*, pages 47–62. Eurographics, June 1993. held in Paris, France, 14–16 June 1993.

[126] Dani Lischinski, Brian Smits, and Donald P. Greenberg. Bounds and error estimates for radiosity. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 67–74. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[127] Daniel Lischinski, Filippo Tampieri, and Donald P. Greenberg. Discontinuity meshing for accurate radiosity. *IEEE Computer Graphics & Applications*, 12(6) :25–39, November 1992.

[128] Daniel Lischinski, Filippo Tampieri, and Donald P. Greenberg. Combining hierarchical radiosity and discontinuity meshing. In *Computer Graphics Proceedings, Annual Conference Series, 1993*, pages 199–208, 1993.

[129] C. Loscos, M. C. Frasson, G. Drettakis, B. Walter, X. Granier, and P. Poulin. Interactive virtual relighting and remodeling of real scenes. Available from www.imagis.imag.fr/Publications RT-0230, Institut National de Recherche en Informatique en Automatique (INRIA), Grenoble, France, April 1999.

[130] C. Loscos, M. C. Frasson, G. Drettakis, B. Walter, X.Grainer, and P. Poulin. Interactive virtual relighting and remodeling of real scenes. In *Rendering Techniques '99*, pages 329–340, New York, NY, 1999. Springer Wien.

[131] Celine Loscos and George Drettakis. Low-cost photometric calibration for interactive relighting. In *Proceedings of the First French-British International Workshop on Virtual Reality*, Brest, France, July 2000.

[132] Celine Loscos, George Drettakis, and Luc Robert. Interactive virtual relighting of real scenes. *IEEE Transactions on Visualization and Computer Graphics*, 6(3) :289–305, 2000.

[133] J. Lu and J. Little. Reflectance function estimation and shape recovery from image sequence of rotating object. In *International Conference on Computer Vision*, pages 80–86, June 1995.

[134] J. David MacDonald and Kellogg S. Booth. Heuristics for ray tracing using space subdivision. *The Visual Computer*, 6(3) :153–166, June 1990. ISSN 0178-2789.

[135] Thomas J. V. Malley. A shading method for computer generated images. Master's thesis, University of Utah, Salt Lake City, June 1988.

[136] Stephen R. Marschner. *Inverse Rendering in Computer Graphics*. PhD thesis, Program of Computer Graphics, Cornell University, Ithaca, NY, 1998.

[137] Stephen R. Marschner and Donald P. Greenberg. Inverse lighting for photography. In *Proceedings of the Fifth Color Imaging Conference*. Society for Imaging Science and Technology, November 1997.

[138] Stephen R. Marschner, Stephen H. Westin, Eric P. F. Lafortune, Kenneth E. Torrance, and Donald P.Greenberg. Image-based brdf measurement including human skin. In Dani Lischinski and Greg Ward Larson, editors, *Eurographics Rendering Workshop 1999*. Eurographics, June 1999.

[139] Leonard McMillan and Gary Bishop. Plenoptic modeling : An image-based rendering system. In Robert Cook, editor, *Proceedings of SIGGRAPH 95*, Computer Graphics Proceedings, Annual Conference Series, pages 39–46. Addison Wesley, August 1995.

[140] Urs Meyer. Hemi-cube ray tracing : a method for generating soft shadows. *Eurographics '90*, pages 365–376, September 1990.

[141] Don P. Mitchell. Generating antialiased images at low sampling densities. *Computer Graphics (Proceedings of SIGGRAPH 87)*, 21(4) :65–72, July 1987. Held in Anaheim, California.

[142] Don P. Mitchell. Spectrally optimal sampling for distributed ray tracing. *Computer Graphics (Proceedings of SIGGRAPH 91)*, 25(4) :157–164, July 1991. ISBN 0-201-56291-X. Held in Las Vegas, Nevada.

[143] Tomas Möller and Ben Trumbore. Fast, minimum storage ray-triangle intersection. *Journal of Graphics Tools*, 2(1) :21–28, 1997. ISSN 1086-7651.

[144] S.K. Nayar, K. Ikeuchi, and T.Kanade. Surface reflection : Physical and geometrical perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(7) :611–634, 1991.

[145] J. Neider, T. Davis, and M. Woo. *OpenGL Programming Guide : The Official Guide to Learning OpenGL*. Addison Wesley, New York, 1995.

[146] Adelene Ng. Techniques for rapid computation of form factors in radiosity. Technical Report No. 680, Queen Mary and Westfield College, 1994.

[147] F. E. Nicodemus. Directional reflectance and emissivity of an opaque surface. *Applied Optics*, 4 :767–7735, 1965.

[148] F. E. Nicodemus. Reflectance nomenclature and directional reflectance and emissivity. *Applied Optics*, 9 :1474–1475, 1970.

[149] F. E. Nicodemus, J. C. Richmond, J. J. Hsia, I. W. Ginsberg, and T. Limperis. Geometric considerations and nomenclature for reflectance. Monograph 160, National Bureau of Standards (US), October 1977.

[150] Jeffry S. Nimeroff, Eero Simoncelli, and Julie Dorsey. Efficient re-rendering of naturally illuminated environments. In *Fifth Eurographics Workshop on Rendering*, pages 359–373, Darmstadt, Germany, June 1994.

[151] Tomoyuki Nishita and Eihachiro Nakamae. A new radiosity apporach using area sampling for parametric patches. *Computer Graphics Forum (Eurographics '93)*, 13(3) :385–398, September 1993.

[152] Nicolas Noe. *Etude de Fonctions de Distribution de la Réflectance Bidirectionnelle*. Ph.D. thesis, Ecole des Mines de Saint-Etienne, Saint-Etienne, France, September 1999.

[153] Masataka Ohta and Mamoru Maekawa. Ray coherence theorem and constant time ray tracing algorithm. *Computer Graphics 1987 (Proceedings of CG International '87)*, pages 303–314, 1987.

[154] James Painter and Kenneth Sloan. Antialiased ray tracing by adaptive progressive refinement. *Computer Graphics (Proceedings of SIGGRAPH 89)*, 23(3) :281–288, July 1989. Held in Boston, Massachusetts.

[155] M. Paulin and J-P. Jessel. Adaptive mesh generation for progressive radiosity : A ray-tracing based algorithm. In *Computer Graphics Forum*, volume 13(3), pages 421–432. Eurographics, Basil Blackwell Ltd, 1994. Eurographics '94 Conference issue.

[156] Fabio Pellacini, James A. Ferwerda, and Donald P. Greenberg. Toward a psychophysically-based light reflection model for image synthesis. In Kurt Akeley, editor, *Proceedings of SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, pages 55–64. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, July 2000.

[157] José-Philippe Pérez. *Optique Géométrique et Ondulatoire*. Masson, Paris, France, 1994.

[158] S. Philips, A. Worrall, C. Willis, and D. Paddon. Adaptive mesh refinement for the radiosity method. In *Proceedings of Compugraphics 1993*, pages 178–186, Alvor, Portugal, December 1993.

[159] B.T. Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6) :311–317, 1975.

[160] Thierry Priol, K. Bouatouch, and D. Menard. Parallel radiosity using a shared virtual memory. In *First Bilkent Computer Graphics Conference, ATARV-93*, Ankara, Turkey, July 1993.

[161] Werner Purgathofer. A statistical method for adaptive stochastic sampling. *Eurographics '86*, pages 145–152, August 1986.

[162] Werner Purgathofer. A statistical method for adaptive stochastic sampling. *Computers & Graphics*, 11(2) :157–162, 1987.

[163] Rodney J. Recker. Improved techniques for progressive refinement radiosity. Master's thesis, Program of Computer Graphics, Cornell Univ., January 1990.

[164] Rodney J. Recker, David W. George, and Donald P. Greenberg. Acceleration techniques for progressive refinement radiosity. *1990 Symposium on Interactive 3D Graphics*, 24(2) :59–66, March 1990. ISBN 0-89791-351-5.

[165] Rodney J. Recker, David W. George, and Donald P. Greenberg. Acceleration techniques for progressive refinement radiosity. In Rich Riesenfeld and Carlo Sequin, editors, *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, volume 24, pages 59–66, March 1990.

[166] Mark C. Reichert. A two-pass radiosity method driven by lights and viewer position. Master's thesis, Program of Computer Graphics, Cornell Univ., January 1992.

[167] A. Rosenblum. *Data Visualization*, chapter Modeling Complex indoor scenes using an analysis/synthesis framework (André Gagalowicz). Academic Press, 1994.

[168] Holly E. Rushmeier. Extending the radiosity method to transmitting and specularly reflecting surfaces. Master's thesis, Program of Computer Graphics, Cornell Univ., 1986.

[169] Holly E. Rushmeier and Kenneth E. Torrance. Extending the radiosity method to include specularly reflecting and translucent materials. *ACM Transactions on Graphics*, 9(1) :1–27, January 1990. ISSN 0730-0301.

[170] B. Le Saec and C. Schlick. A progressive ray tracing based radiosity with general reflectance functions. In *Eurographics Workshop on Photosimulation, Realism and Physics in Computer Graphics*, 1989.

[171] Bertrand Le Saec and Christophe Schlick. A progressive ray-tracing-based radiosity with general reflectance functions. Technical Report 90-17, Laboratoire Bordelais de Recherche en Informatique, Universite de Bordeaux I, 1990.

[172] Betrand Le Saec and Christophe Schlick. A progressive ray-tracing-based radiosity with general reflectance functions. In *Proceedings Eurographics Workshop on Photosimulation, Realism and Physics in Computer Graphics*, pages 103–16, Rennes, France, June 1990.

[173] David Salesin and Jorge Stolfi. The zz-buffer : a simple and efficient rendering algorithm with reliable antialiasing. *Proceedings of the PIXIM '89*, pages 451–466, 1989.

[174] David Salesin and Jorge Stolfi. Rendering csg models with a zz-buffer. *Computer Graphics (Proceedings of SIGGRAPH 90)*, 24(4) :67–76, August 1990. ISBN 0-201-50933-4. Held in Dallas, Texas.

[175] Hanan Samet. Implementing ray tracing with octrees and neighbor finding. *Computers & Graphics*, 13(4) :445–60, 1989.

[176] Imari Sato, Yoichi Sato, and Katsushi Ikeuchi. Acquiring a radiance distribution to superimpose virtual objects onto a real scene. *IEEE Transactions on Visualization and Computer Graphics*, 5(1) :1–12, January 1999.

[177] Imari Sato, Yoichi Sato, and Katsushi Ikeuchi. Illumination distribution from brightness in shadows : Adaptive extimation of illumination distribution with unknown reflectance properties in shadow regions. In *Proceedings of IEEE ICCC'99*, pages 875–882, September 1999.

[178] Imari Sato, Yoichi Sato, and Katsushi Ikeuchi. Illumination distribution from shadows. In *Proceedings of IEEE CVPR'99*, pages 306–312, June 1999.

[179] Kosuke Sato and Katsushi Ikeuchi. Determining reflectance properties of an object using range and brightness images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(11) :1139–1153, 1991.

[180] Yoichi Sato and Katsushi Ikeuchi. Temporal-color space analysis of reflection. *Journal of Optical Society of America*, 11(11) :2990–3002, November 1994.

[181] Yoichi Sato and Katsushi Ikeuchi. Reflectance analysis for 3d computer graphics model generation. *Graphical Models and Image Processing*, 58(5) :437–451, 1996.

[182] Yoichi Sato, Mark D. Wheeler, and Katsushi Ikeuchi. Object shape and reflectance modeling from observation. In Turner Whitted, editor, *Computer Graphics, Proceedings of SIGGRAPH 97*, pages 379–388. Addison Wesley, August 1997.

[183] Andreas Schilling and Wolfgang Straßer. Exact : Algorithm and hardware architecture for an improved a-buffer. *Proceedings of SIGGRAPH 93*, pages 85–92, August 1993. ISBN 0-201-58889-7. Held in Anaheim, California.

[184] Christophe Schlick. A customizable reflectance model for everyday rendering. In *Fourth Eurographics Workshop on Rendering*, pages 73–84, Paris, France, June 1993.

[185] Christophe Schlick. A Survey of Shading and Reflectance Models. *Computer Graphics Forum*, 13(2) :121–131, June 1994.

[186] Peter Schroder, Steven J. Gortler, Michael F. Cohen, and Pat Hanrahan. Wavelet projections for radiosity. *Computer Graphics Forum*, 13(2) :141–151, June 1994.

[187] Peter Schroeder. Numerical integration for radiosity in the presence of singularities. In Michael F. Cohen, Claude Puech, and Francois Sillion, editors, *Fourth Eurographics Workshop on Rendering*, pages 177–184. Eurographics, June 1993. held in Paris, France, 14–16 June 1993.

[188] Peter Schroeder, Steven Gortler, Michael Cohen, and Pat Hanrahan. Wavelet projections for radiosity. In Michael F. Cohen, Claude Puech, and Francois Sillion, editors, *Fourth Eurographics Workshop on Rendering*, pages 105–114. Eurographics, June 1993. held in Paris, France, 14–16 June 1993.

[189] Min-Zhi Shao and Norman I. Badler. A gathering and shooting progressive refinement radiosity method. Technical Report MS-CIS-93-03, Department of Computer and Information Science, University of Pennsylvania, 1993.

[190] Robert Siegel and John R. Howell. *Thermal Radiation Heat Transfer, 3rd Edition.* Hemisphere Publishing Corporation, New York, NY, 1992.

[191] F. X. Sillion. A unified hierarchical algorithm for global illumination with scattering volumes and object clusters. *IEEE Transactions on Visualization and Computer Graphics*, 1(3) :240–254, September 1995. ISSN 1077-2626.

[192] Francois Sillion. Radiosity with non-diffuse reflectors. *ACM SIGGRAPH '93 Course Notes - Making Radiosity Practical*, pages Chapter 5, 1–25, 1993.

[193] Francois Sillion. Clustering and volume scattering for hierarchical radiosity calculations. In *Fifth Eurographics Workshop on Rendering*, pages 105–117, Darmstadt, Germany, June 1994.

[194] Francois Sillion and Claude Puech. *Radiosity and Global Illumination.* Morgan Kaufmann, San Francisco, 1994.

[195] Francois X. Sillion. *Simulation de l'éclairage pour la synthèse d'images : réalisme et interactivité.* PhD thesis, Spécialité Informatique, Université de Paris-Sud, Centre d'Orsay, 1989.

[196] François X. Sillion, James R. Arvo, Stephen H. Westin, and Donald P. Greenberg. A global illumination solution for general reflectance distributions. *Computer Graphics (Proceedings of SIGGRAPH 91)*, 25(4) :187–196, July 1991. ISBN 0-201-56291-X. Held in Las Vegas, Nevada.

[197] Francois X. Sillion and Claude Puech. A general two-pass method integrating specular and diffuse reflection. In Jeffrey Lane, editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 335–344, July 1989.

[198] Jaswinder P. Singh, Annop Gupta, and Marc Levoy. Parallel visualization algorithms : Performance and architectural implications. *IEEE Computer*, 27(7) :45–55, July 1994.

[199] B. G. Smith. Geometrical shadowing of a random rough surface. *IEEE Transactions on Antennas and Propagation*, 15(5) :668–671, September 1967.

[200] Brian Smits. *Efficient Hierarchical Radiosity for Complex Environments.* Ph.D. thesis, Cornell University, Ithaca, NY, 1994.

[201] Brian E. Smits, James R. Arvo, and David H. Salesin. An importance-driven radiosity algorithm. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 273–282, July 1992.

[202] Stephen Spencer. The hemisphere radiosity method : a tale of two algorithms. In *Eurographics Workshop on Photosimulation, Realism and Physics in Computer Graphics*, pages 127–135, 1990.

[203] Marc Stamminger, Annette Scheel, Xavier Granier, Frederic Perez-Cazorla, George Drettakis, and François Sillion. Efficient glossy global illumination with interactive viewing. *Computer Graphics Forum*, 19(1) :13–25, March 2000. ISSN 1067-7055.

[204] Nilo Stolte and René Caubet. Discrete ray tracing high resolution 3d grids. *Winter School of Computer Graphics 1995*, February 1995. Held in held at University of West Bohemia, Plzen, Czech Republic, 14-18 February 1995.

[205] Nilo Stolte and René Caubet. Discrete ray-tracing of huge voxel spaces. *Computer Graphics Forum*, 14(3) :383–394, August 1995. ISSN 1067-7055.

[206] W. Sturzlinger. Adaptive mesh refinement with discontinuities for the radiosity method. In *Fifth Eurographics Workshop on Rendering*, pages 239–248, Darmstadt, Germany, June 1994.

[207] W. Sturzlinger and C. Wild. Parallel progressive radiosity with parallel visibility computations. In V. Skala, editor, *Proceedings of the Winter School of Computer Graphics '94*, pages 66–74, Plzen, Czech Republic, January 1994.

[208] Kelvin Sung. Area sampling buffer : Tracing rays with z-buffer hardware. *Computer Graphics Forum*, 11(3) :299–310, September 1992.

[209] Kelvin Sung. Ray tracing with the bsp tree. *Graphics Gems III*, pages 271–274, 538–546, 1992. ISBN 0-12-409673-5. Held in Boston.

[210] H.D. Tagare and R.J.P. de Figueiredo. A theory of photometric stereo for a class of diffuse non-lambertian surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(2) :133–152, 1991.

[211] H.D. Tagare and R.J.P. de Figueiredo. A framework for the construction of reflectance maps for machine vision. *CGVIP : Image Understanding*, 57(3) :265–282, 1993.

[212] Filippo Tampieri. Accurate form-factor computation. *Graphics Gems III*, pages 329–333, 577–581, 1992. ISBN 0-12-409673-5. Held in Boston.

[213] Filippo Tampieri. *Discontinuity Meshing for Radiosity Image Synthesis*. Ph.D. thesis, Cornell University, Ithaca, NY, 1993.

[214] Seth Teller and Pat Hanrahan. Global visibility algorithms for illumination computations. In *Computer Graphics Proceedings, Annual Conference Series, 1993*, pages 239–246, 1993.

[215] Seth J. Teller. Computing the antipenumbra of an area light source. *Computer Graphics (Proceedings of SIGGRAPH 92)*, 26(2) :139–148, July 1992. ISBN 0-201-51585-7. Held in Chicago, Illinois.

[216] Pierre Tellier, E. Maisel, Kadi Bouatouch, and Eric Langueonou. Exploiting spatial coherence to accelerate radiosity. *The Visual Computer*, 10 :46–53, 1993.

[217] Daniel Thalmann. *Scientific Visualization and Graphics Simulation*, chapter Collaboration between Computer Graphics and Computer Vision (André Gagalowicz), pages 233–248. Addison Wesley, Chichester, England, 1992.

[218] Siu-Hang Or Tien-Tsin Wong, Pheng-Ann Heng and Wai-Yin Ng. Image-based rendering with controllable illumination. In Julie Dorsey and Phillip Slusallek, editors, *Rendering Techniques '97 (Proceedings of the Eighth Eurographics Workshop on Rendering)*, pages 13–22, New York, NY, 1997. Springer Wien. ISBN 3-211-83001-4.

[219] K.E. Torrance and E.M. Sparrow. Theory for off-specular reflection from roughened surfaces. *Journal of Optical Society America*, 57(9) :1105–1114, 1967.

[220] K.E. Torrance, E.M. Sparrow, and R.C. Birkebak. Polarization, directional distribution, and off-specular peak phenomena in light reflected from roughened surfaces. *Journal of Optical Society of America*, 56(7) :916–925, 1966.

[221] Ben Trumbore. Rectangular bounding volumes for popular primitives. *Graphics Gems III*, pages 295–300, 555–561, 1992. ISBN 0-12-409673-5. Held in Boston.

[222] Jack Tumblin and Holly Rushmeier. Tone reproduction for realistic images. *IEEE Computer Graphics and Applications*, 13(6) :42–48, November 1993.

[223] Greg Turk and Marc Levoy. Zippered polygon meshes from range images. In Andrew Glassner, editor, *Computer Graphics, Proceedings of SIGGRAPH 94*, pages 311–318. ACM SIGGRAPH, ACM Press, July 1994.

[224] Amitabh Varshney and Jan F. Prins. An environment-projection approach to radiosity for mesh-connected computers. *Third Eurographics Workshop on Rendering*, pages 271–281, May 1992.

[225] Eric Veach. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Dept. of Computer Science, Stanford University, dec 1997.

[226] Eric Veach. *Robust Monte Carlo Methods for Light Transport Simulation*. Ph.D. thesis, University of Stanford, December 1997.

[227] Christophe Vedel and Claude Puech. A testbed for adaptive subdivision in progressive radiosity. In *Eurographics Workshop on Rendering*, 1991.

[228] Christophe Vedel and Claude Puech. A testbed for adaptive subdivision in progressive radiosity. In *Photorealistic Rendering in Computer Graphics (Proceedings of the Second Eurographics Workshop on Rendering)*, pages 93–103, New York, 1994. Springer-Verlag.

[229] Josep Vilaplana. Parallel radiosity solutions based on partial result messages. *Third Eurographics Workshop on Rendering*, pages 259–270, May 1992.

[230] Douglas Voorhies. Ray-triangle intersection using binary recursive subdivision. *Graphics Gems II*, pages 257–263, 1991. ISBN 0-12-064481-9. Held in Boston.

[231] John R. Wallace. A two-pass solution to the rendering equation : A synthesis of ray tracing and radiosity methods. M.sc. thesis, Program of Computer Graphics, Cornell University, January 1988.

[232] John R. Wallace. Meshing and reconstruction. *ACM SIGGRAPH '93 Course Notes - Making Radiosity Practical*, pages Chapter 6, 1–30, 1993.

[233] John R. Wallace, Michael F. Cohen, and Donald P. Greenberg. A two-pass solution to the rendering equation : A synthesis of ray tracing and radiosity methods. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 311–320, July 1987.

[234] John R. Wallace, Kells A. Elmquist, and Eric A. Haines. A ray tracing algorithm for progressive radiosity. In Jeffrey Lane, editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 315–324, July 1989.

[235] Mingfu Wang, Hujun Bao, and Qunsheng Peng. A new progressive radiosity algorithm through the use of accurate form-factors. *Comput. & Graphics*, 16(3) :303–310, 1992.

[236] Gregory J. Ward. Measuring and modeling anisotropic reflection. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 265–272. ACM Press, July 1992.

[237] Gregory J. Ward. The RADIANCE lighting simulation and rendering system. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 459–472. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[238] Gregory J. Ward. The radiance lighting simulation and rendering system. In Andrew Glassner, editor, *Proceedings of SIGGRAPH 94*, Computer Graphics Proceedings, Annual Conference Series, pages 459–472. ACM Press, July 1994. ISBN 0-89791-667-0. Held in Orlando, Florida.

[239] Alan Watt and Mark Watt. *Advanced Animation and Rendering Techniques : Theory and Practice*. Addison-Wesley Publishing Company, 1992.

[240] Hank Weghorst, Gary Hooper, and Donald P. Greenberg. Improved computational methods for ray tracing. *ACM Transactions on Graphics*, 3(1) :52–69, January 1984.

[241] Stephen H. Westin, James R. Arvo, and Kenneth E. Torrance. Predicting reflectance functions from complex surfaces. *Computer Graphics (Proceedings of SIGGRAPH 92)*, 26(2) :255–264, July 1992. ISBN 0-201-51585-7. Held in Chicago, Illinois.

[242] K. Y. Whang, J. W. Song, J. W. Chang, J. Y. Kim, W. S. Cho, C. M. Park, and I. Y. Song. Octree-r : an adaptive octree for efficient ray tracing. *IEEE Transactions on Visualization and Computer Graphics*, 1(4) :343–349, December 1995. ISSN 1077-2626.

[243] Mark D. Wheeler, Yoichi Sato, and Katsushi Ikeuchi. Consensus surfaces for modeling 3d objects from multiple range images. In *DARPA Image Understanding Workshop'97 (New Orleans, LA)*, pages 1229–1236, May 1997.

[244] T. Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6) :343–349, 1980.

[245] George Wolberg. *Digital Image Warping.* IEEE Computer Society Press, Los Alamitos, 1990.

[246] Tien-Tsin Wong, Pheng-Ann Heng, Siu-Hang Or, and Wai-Yin Ng. Illumination of image-based objects. *Journal of Visualization and Computer Graphics*, 9(3) :113–127, 1998.

[247] Tien-Tsin Wong, PhengAnn Heng, Siu-Hang Or, and Wai-Yin Ng. Illuminating image-based objects. In *Pacific Graphics '97 (Seoul, Korea)*, October 1997.

[248] Andrew Woo. Fast ray-polygon intersection. *Graphics Gems*, pages 390–394, 1990. ISBN 0-12-286166-3. Held in Boston.

[249] Andrew Woo. Ray tracing polygons using spatial subdivision. *Graphics Interface '92*, pages 184–191, May 1992.

[250] Xiaolin Wu. A linear-time simple bounding volume algorithm. *Graphics Gems III*, pages 301–306, 1992. ISBN 0-12-409673-5. Held in Boston.

[251] Y. Yu, P. Debevec, J. Malik, and T. Hawkins. Inverse global illumination : Recovering reflectance models of real scenes from photographs. In A. Rockwood, editor, *Computer Graphics (SIGGRAPH '99 Proceedings)*, volume 19, pages 215–224. Addison Wesley Longman, August 1999.

[252] Yizhou Yu. *Modeling and Editing Real Scenes with Image-Based Techniques.* PhD thesis, University of California, Berkeley, 2000.

[253] Borut Zalik, Gordon Clapworthy, and Crtomir Oblonsek. An efficient code-based voxel-traversing algorithm. *Computer Graphics Forum*, 16(2) :119–128, 1997. ISSN 0167-7055.

[254] Min Zhi Shao and Norman Badler. Analysis and acceleration of progressive refinement radiosity method. In Michael F. Cohen, Claude Puech, and Francois Sillion, editors, *Fourth Eurographics Workshop on Rendering*, pages 247–258. Eurographics, June 1993. held in Paris, France, 14–16 June 1993.

[255] Yong Zhou and Qunsheng Peng. The super-plane buffer : An efficient form-factor evaluation algorithm for progressive radiosity. *Comput. & Graphics*, 16(2) :151–158, 1992.