THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par *l'Université Toulouse III - Paul Sabatier*
Discipline ou spécialité : *Informatique*

Présentée et soutenue par Anthony Pajot
Le 26 avril 2012

Toward robust and efficient physically-based rendering

JURY

Rapporteurs : Kadi Bouatouch, Pr., Université de Rennes 1
Nicolas Holzschuch, DR, INRIA Rhône-Alpes

Examinateurs : George Drettakis, DR, INRIA Sophia-Antipolis
Xavier Granier, CR, INRIA Bordeaux Sud-Ouest

Invité : Pierre Poulin, Pr., Université de Montréal

Ecole doctorale : Mathématiques Informatique Télécommunications de Toulouse
Unité de recherche : Institut de Recherche en Informatique de Toulouse - UMR 5505
Directeur(s) de Thèse : Mathias Paulin, Loïc Barthe

Acknowledgments

À la question “jusqu’à quelle longueur ça peut faire, la section remerciements ?”, mon directeur de thèse m’a répondu “pas plus long que la thèse en elle-même !”.

Eh ben je peux vous dire que c’est court, parce que des remerciements, il y en a à faire ! Et des sincères hein, pas des remerciements dans le vent pour faire joli et parce que lorsque nous étions enfants on nous a dit d’être polis.

Déjà, je voudrais remercier ceux dont j’ai été le moins proche, mais qui ont eu un rôle essentiel : les personnes qui ont jugé mon travail. Rendez-vous compte : trois semaines avant la soutenance, un pavé de 240 pages qui arrive avec peu d’images et limite plus d’équations que de phrases. Et en plus ils ont dû se lever à pas d’heure et traverser la France pour venir assister à une présentation. Et le pire de tout ca, ils n’ont presque pas pu profiter du pot de thèse, leurs avions partant juste après. Franchement, à leur place, j’aurais dit non. Donc merci beaucoup à vous pour le temps consacré à l’examen de mon travail !

J’écris cette section presque un an après la soutenance (des...euh...j’étais très occupé, voyez-vous), et malgré ce petit laps de temps, les gens que j’ai pu rencontrer pendant les 3 ans et demi de la thèse et ce qu’ils m’ont apporté reste toujours aussi fort.

Il y a ceux dont c’est le métier, mais qui l’ont toujours fait pour tout autre chose que par pur professionnalisme ou parce que c’est leur rôle : mes encadrants, Mathias et Loïc. Pendant 3 ans et quelques, ils ont supporté mon opiniâtreté, ma mauvaise foi et mon sens inné de l’exagération. Et en plus de ca, Mathias a accepté que je ne capte pas grand chose au début de ma thèse et que je change de voie, Loïc a accepté qu’il ne comprenne pas grand chose au début de ma thèse (et même après), et malgré cela est toujours revenu à l’attaque, parce que bon, Monte-Carlo ca ne pouvait pas être QUE du bruit malgré ce que je lui montrais ! Bon, à un moment ils en ont dû en avoir marre, parce qu’ils m’ont envoyé pendant 3 mois à l’autre bout du monde, chez un pauvre Montréalais qui avait passé 6 mois chargés de bi...boissons à bulle et de VTT à Toulouse, et qui m’a acceuilli de la manière la plus gentille du monde. Pierre, vous autres québécois vous ne savez peut-être pas faire du bon pain pantoute, mais ostie de criss qu’est-ce-que c’était bien ces trois mois de sauna..euh, d’été à Montréal dans ton lab ! Et qu’est-ce-que j’étais content que tu viennes à ma soutenance ! Après mon retour, certainement parce que je n’avais pas changé, mes encadrants ont pris une troisième personne pour mieux tenir le choc, David. Le pauvre ! Et à côté de ca, VTT, montagne, Magic, tout était pretexte à discuter (ou à me faire cracher mes poumons ou mes points de vie) dans une ambiance légère, décontractée, mais où le travail avançait et les cerveaux fumaient gaiement !

Après, il y a ceux qui m’ont soutenu sans raison particulière. Ma famille par exemple. Toujours là quand j’en avais besoin même si je ne le disais pas. Toujours prêts, un support présent mais qui ne s’impose pas. Papa, Maman, Laurence et Arnaud, merci, vous avez été parfaits !

Et après il y a les amis. Oulala, je vais rester bref (mais si mais si, je suis bref là), mais ca pourrait être long...

Déjà, il y a les personnes qui travaillaient dans la même salle que moi, ceux des salles d’à-côté avec qui on allait manger... Et au delà de ça, les moments partagés en dehors de ce cadre ! Trois semaines d’un voyage inoubliable aux États-Unis après un an...un mois...bon, une très grosse semaine de boulot à

deux (couchés à 2h du matin tous les soirs pour avoir un talk SIGGRAPH !) ! Une semaine d'un voyage "toi aussi devient breton, apprends à ignorer la pluie" à Terre-Neuve. Et les sessions de karting avec la feuille de temps affichée dans la salle avec les meilleurs temps surlignés et Samir qui voit ses "poulains" commencer sérieusement à revenir sur lui ? Et les parties de Magic endiablées pendant les rares et très courtes pauses que l'on faisait tous les jours entre 16h et 18h ? Le cours de physique au milieu des vaches (littéralement) à Roffiac ? Les panneaux "help" pendant l'énoncé de l'exercice "bâteau" du puit de potentiel quantique ? Les discussions sur tout et n'importe quoi les midis (notamment celles sur "pourquoi on repeindrait pas la salle en rose ?" qui rencontra une vive opposition) ? Les spectacles d'impros ? Etc. etc. En fait, toutes ces choses qui font que même si la thèse en elle-même ne va pas, que ca avance pas, que ca rame, que ca en est désespérant, ben on va au labo avec la banane et une énergie à chaque fois renouvellée ! Donc un grand, un immense merci à tous ceux qui ont rendu ces moments si savoureux, je suis ravi de vous cotôyer ou de vous avoir cotôyés : Andra, Dorian, François, Guillaume, Jonathan, Laure, Marie, Mathieu G., Mathieu M., Nelly, Marion, Monia, Olivier, Philippe, Pignouf, Rodolphe, Roman, Samir, Sylvain, Shaouki, Vincent, et ceux dont j'aurais pu oublier le prénom !

Puis il y a les amis d'avant la thèse aussi, Antoine, Benoit, Cédric, Julien, Lucas, et les amis d'encore avant, Lorraine et Ronan, qui m'ont fait rencontrer une bande de fous-furieux adorables à Montréal ! Décarie, vraiment, c'était super cool !

Et puis il y a d'autres rencontres et d'autres moments, telles que les semaines à Roffiac ou à Odeillo avec les physiciens du Laplace, merci Richard et Stéphane de nous avoir donné cette opportunité. Vraiment, les vaches qui regardent le tableau rempli d'équations, c'était un grand moment !

Et enfin, il y a aussi les personnes rencontrées en dehors de la thèse, qui sans le savoir vous font oublier les tracas, et dont certains deviennent des amis. Benoit, Carine, Laurent et tous les fadas montagnards, merci pour tous ces bons moments passés en montagne ou ailleurs, qui permettaient de revenir le lundi matin avec le sourire jusqu'aux oreilles ! Et vivement les prochains ! Et merci aux gens des clubs d'impro et de sport que j'ai cotôyé durant cette période et que je continue à cotôyer avec joie !

Pour résumer, merci à toutes les personnes que j'ai croisé durant cette période, et qui directement ou indirectement, m'ont permis de relever ce défi !

À tous ceux et celles qui m'ont poussé à aller plus loin.

Contents

Partie I Introduction 1

Version française 3

English version 7

Partie II Context 11

Chapter 1 Physical modelisation of light transport for rendering 13

1.1 Image, pixel, rendering process . . . 13
1.2 Radiometry . . . 14
1.2.1 From spectral radiance to energy: radiometric integrals . . . 15
1.2.2 From energy to spectral radiance: differential formulation . . . 17
1.2.3 Relation between incident and outgoing quantities . . . 18
1.2.4 Spectral distributions . . . 19
1.3 Colorimetry . . . 19
1.3.1 Chromaticity, luminance, gamut . . . 20
1.3.2 RGB, gamma correction, white point, sRGB . . . 20
1.4 From radiometry to colorimetry . . . 22
1.4.1 Spectrum $\rightarrow$ RGB conversion . . . 24
1.4.2 Spectrum $\rightarrow$ CIE XYZ $\rightarrow$ RGB . . . 24
1.5 Content of a HDR image, sensor . . . 26
1.5.1 Reconstruction filter . . . 27
1.5.2 Signal value, sensor response . . . 27
1.6 Light interactions . . . 28
1.6.1 Self-emission . . . 28
1.6.2 Absorption and scattering . . . 29
1.7 Macroscopic models of light interaction . . . 30

1.7.1 Absorption and scattering at surfaces 31
1.7.2 Absorption and scattering in participating media 34
1.8 Light transport equations 39
1.8.1 Light transport in empty space 39
1.8.2 Light transport with participating media 39
1.9 Summary and final equations 41
1.9.1 Hypotheses 41
1.9.2 Spectral rendering 42
1.9.3 Color-space rendering 42
1.9.4 Final equations 44
1.10 Conclusion 45

Chapter 2 Description of a scene for physically-based rendering **47**
2.1 Camera model 47
2.1.1 Optical model of lens systems 48
2.1.2 Putting lens systems and sensors together 51
2.1.3 Radiometric model 54
2.2 Light sources 55
2.2.1 Local light sources 55
2.2.2 Distant light sources 55
2.3 Geometry 56
2.3.1 Geometry representation 56
2.3.2 Efficient ray-tracing function 59
2.4 Materials 61
2.5 Participating media 62
2.6 Conclusion 62

Chapter 3 Mathematical tools for physically-based rendering **63**
3.1 Path-integral formulation 63
3.1.1 Path representation and light sources types 64
3.1.2 Path space, path space measure 64
3.1.3 R_p and f 66
3.2 Tools for numerical integration 67
3.3 Probability to have a section on probabilities = 1 67
3.3.1 Random variable 67
3.3.2 Probability density function, cumulative density function 69
3.3.3 Probability distributions and random variables, status of PRNGs 71

3.3.4 Equivalence of random variables, i.i.d. random variables, impact of PRNGs on independence . . . 72
3.3.5 Function of random variable(s), expected value . . . 72
3.4 Estimators, standard Monte-Carlo estimator . . . 73
3.4.1 Bias, variance, robustness . . . 74
3.4.2 Dirac-delta distributions and Monte-Carlo . . . 77
3.5 Variance reduction methods . . . 79
3.5.1 Importance sampling . . . 79
3.5.2 Multiple importance sampling . . . 80
3.5.3 Control variate . . . 81
3.5.4 Uniform samples placement . . . 81
3.6 Non-standard Monte-Carlo methods . . . 83
3.7 Sampling optical paths . . . 84
3.7.1 Local importance sampling of optical paths . . . 84
3.7.2 Russian roulette . . . 86
3.8 Conclusion . . . 86

Partie III Contributions 89

Chapter 4 A versatile software architecture for physically-based rendering 91
4.1 From equations to software architecture . . . 92
4.1.1 Image reconstruction and processing . . . 93
4.1.2 Physical simulation . . . 94
4.1.3 Rendering control . . . 97
4.2 Examples of components . . . 97
4.2.1 Integrator . . . 97
4.2.2 Simulator . . . 101
4.2.3 Adaptive screen sampling . . . 102
4.2.4 Accumulator . . . 103
4.2.5 HDR processor . . . 103
4.3 Robust and efficient rendering . . . 104

Chapter 5 Representativity for robust and adaptive multiple importance sampling 107
5.1 Introduction . . . 107
5.2 Related works . . . 110
5.2.1 Importance sampling strategies for rendering . . . 110
5.2.2 General variance reduction . . . 111

5.3 Variance-optimal sampling configurations 112
5.4 Representativity-based sampling 113
5.4.1 BSDF-based strategy representativity 114
5.4.2 Photon-map-based strategy representativity 115
5.4.3 Sampling configurations from representativities 118
5.4.4 General hints for defining representativities 119
5.5 Numerical analyses 120
5.5.1 Simple scenes 121
5.5.2 Chains of estimators 126
5.6 Examples of applications 131
5.6.1 Photon mapping final gathering 131
5.6.2 Photon-map guided path-tracing 131
5.6.3 Direct lighting in highly occluded environment 131
5.6.4 Other contexts than MIS-based estimation 131
5.7 Conclusion 132

Chapter 6 Sample-space bright spots removal using density estimation **133**
6.1 Introduction 133
6.2 Related works 135
6.2.1 Image-space bright spots removal 135
6.2.2 Sample-space bright spots removal 136
6.2.3 Outlier detection 136
6.3 Bright-spot removal using density estimation 137
6.3.1 Adaptive bandwidth 139
6.4 Lowering memory consumption 140
6.4.1 Algorithm overview 140
6.4.2 Approximate distribution representation (ADR) 141
6.4.3 ADR: analysis 142
6.4.4 Rule-of-thumb bandwidth (ROT) 143
6.4.5 ROT: analysis 143
6.4.6 Switching from ADR to ROT 144
6.5 Results 146
6.6 Conclusion 148

Chapter 7 Effective despeckling of HDR images **151**
7.1 Introduction 151
7.2 Tag-and-reconstruct despeckling filter 151

7.3 Results 153
7.4 Conclusion 158

Chapter 8 Robust adaptive screen-sampling for Monte-Carlo-based rendering 159
8.1 Introduction 159
8.2 Robust error-based adaptive sampling 160
8.3 Results 162
8.3.1 Robustness to outliers 162
8.3.2 Adaptive sampling evaluation 163
8.4 Conclusion 165

Chapter 9 Globally adaptive control variate for least-square kd-trees on volumes 169
9.1 Introduction 169
9.2 Related works 170
9.2.1 Participating media representations 170
9.2.2 Numerical integration 171
9.3 Error-guided kd-tree 173
9.3.1 Error measure 174
9.3.2 Cost minimization 175
9.3.3 Enforcing global error from local errors 176
9.3.4 Generalization 176
9.4 Globally adaptive control variate 177
9.4.1 Overview 177
9.4.2 Estimations 179
9.4.3 Convergence criterion 180
9.4.4 Kd-tree as control variate 180
9.4.5 Control variate and estimations interactions 182
9.4.6 Estimation error 182
9.4.7 Integrals over arbitrary finite-dimensional supports 183
9.5 Results 184
9.5.1 Numerical integration 184
9.5.2 Trees 190
9.6 Extensions 194
9.7 Conclusion 195

Chapter 10 Combinatorial bidirectional path-tracing for efficient hybrid CPU/GPU rendering 197
10.1 Introduction 197

10.2 Related works . . . 198
10.3 Combinatorial bidirectional path-tracing (CBPT) . . . 198
10.3.1 Base algorithm . . . 198
10.3.2 Discussion . . . 201
10.4 Efficient computation of combination data . . . 202
10.5 Implementation of CBPT . . . 204
10.6 Results . . . 205
10.6.1 Combination throughput . . . 207
10.6.2 CBPT . . . 208
10.7 Conclusion . . . 212

Partie IV Final summary and conclusions 213

Bibliography 219

Personal bibliography 227

Part I

Introduction

Version française

L'informatique graphique est utilisée dans des domaines nombreux et variés. Les besoins en réalisme et en complexité visuelle présentés pour le divertissement augmentent très rapidement, que ce soit les effets spéciaux dans les films, les environnements des jeux vidéos ou même les publicités. L'informatique graphique devient aussi un moyen d'aide à la décision : la prévisualisation virtuelle d'objets ou de bâtiments permet aux designers et architectes de prendre en compte des aspects visuels difficiles à se représenter mentalement, tels que l'influence de l'éclairage sur la perception d'une pièce. De même, les illustrations sont un moyen efficace pour mieux appréhender des idées complexes ou des concepts abstraits, ou tout simplement garder l'attention de l'auditoire. Que ce soit pour le divertissement ou l'aide, l'informatique graphique peut endosser plusieurs fonctions : démonstration, remplacement d'objets difficiles à construire par des maquettes virtuelles, simulation, ou bien encore visualisation de mondes virtuels. Pour chacune de ces fonctions, il existe des types de présentations qui répondent plus ou moins bien aux besoins : le rendu non-photoréaliste est adapté pour les schémas, les plans ou bien encore les illustrations, grâce à la grande liberté existante dans les styles graphiques utilisables. Ceci dit, même quand le photo-réalisme est désirable, un large panel de possibilités est disponible, en fonction des contraintes qui sont posées : un faible coût de calcul pour les jeux, un haut niveau de contrôle artistique pour les films, une grande précision au prix de calculs très coûteux dans le cadre du rendu prédictif (rendu dit physiquement *réaliste*), ou bien encore des coûts de calcul moindres *via* l'utilisation d'approximations dont l'impact est connu comme étant mineur pour les scènes traitées (rendu dit *basé* sur la physique).

Dans cette thèse, nous nous concentrons sur ce dernier type de rendu : nous visons le *photo-réalisme*, *sans nécessiter de trucages* de la part d'un créateur de contenu pour reproduire un effet d'éclairage particulier, dans un nombre de scènes *aussi grand que possible*, *i.e.* des scènes où les approximations faites ont un impact imperceptible. Pour cela, il est nécessaire de simuler le comportement de la lumière, en choisissant consciencieusement les approximations faites, afin de pouvoir traiter un maximum de scènes d'une manière qui soit proche du comportement réel de la lumière.

Les équations utilisées pour la simulation, que l'on dérive au Chapitre 1, prennent en compte le comportement de la lumière tel que décrit par la physique, ainsi que la manière dont une image peut être calculée à partir de cette modélisation, en incluant les phénomènes de perception humaine. La physique de la lumière est basée sur une modélisation macroscopique, décrite par la radiométrie, et sur des équations dites de *transport lumineux*. Ces équations font partie du domaine du transfert radiatif. Pendant la dérivation, nous montrons que prendre en compte tous les phénomènes lumineux, même en se limitant aux phénomènes macroscopiques décrivables par la radiométrie, implique entre autre une

extrême complexité et des temps de calculs prohibitifs. Nous explicitons donc les approximations qui sont couramment faites dans le cadre du rendu basé sur la physique. Ces équations nous permettent de calculer une image telle que vue par une caméra placée dans une scène comprenant des sources de lumières, des objets et des milieux continus tels que la fumée ou le brouillard. Ces entités doivent être décrites et représentées dans un formalisme compatible avec la simulation. Une brève revue de ces représentations est effectuée dans le Chapitre 2. Les équations qui doivent être résolues durant la simulation sont des intégrales, qui ne peuvent être calculées de manière analytique pour des scènes arbitraires. Des méthodes numériques doivent donc être employées. Bien que plusieurs approches existent, nous nous concentrons dans cette thèse à la méthode la plus couramment utilisée actuellement en rendu basé sur la physique, à savoir la méthode de Monte-Carlo, que nous présentons dans le Chapitre 3.

Utiliser ces outils mathématiques pour effectuer la simulation physique dans le cadre du rendu requiert une architecture logicielle adaptée. Notre première contribution, exposée dans le Chapitre 4, est une architecture logicielle inspirée de celle de moteurs de rendu connus, mais ayant une flexibilité accrue. Cette architecture est basée sur une décomposition fonctionnelle des équations de la physique, en prenant en compte la nature des algorithmes liés à la méthode de Monte-Carlo, ainsi que les algorithmes développés dans le domaine du rendu. Nous montrons qu'un grand nombre d'algorithmes faisant partie de l'état de l'art et touchant à divers aspects du rendu sont facilement implantables dans l'architecture que nous définissons.

Le cœur d'un moteur de rendu basé sur la méthode de Monte-Carlo est la partie liée à l'intégration numérique des équations du rendu, étant responsable de la simulation du transport de la lumière. De manière superficielle, faire du rendu avec la méthode de Monte-Carlo consiste à utiliser quelques "exemples" (appelés *échantillons*) de chemins que la lumière peut suivre dans une scène avant d'atteindre la caméra, et d'en extrapoler la valeur qui serait obtenue si tous les chemins étaient pris en compte. À notre connaissance, aucun algorithme de l'état de l'art ne peut traiter toutes les scènes possibles de manière rapide et précise : pour chacun de ces algorithmes, il existe des scènes où l'algorithme donne de mauvais résultats, ou nécessite des temps de calculs trop importants. Cette sensibilité est un manque de *robustesse*, qui se traduit par des temps de calcul qui ne sont *pas consistants*. Pour chaque algorithme, les raisons pour lesquelles il n'arrive pas à traiter telle ou telle scène sont souvent techniques, et inconnues des utilisateurs principaux du moteur, à savoir les artistes. De leur point de vue, ils sont donc confrontés à des temps de calcul qui ne varient pas qu'en fonction de la complexité globale de la scène, mais peuvent aussi devenir extrêmement longs pour des configurations qui leur semblent arbitraires et imprévisibles.

Plutôt que de s'attaquer à la définition d'une méthode de simulation complètement robuste, nous choisissons dans cette thèse de développer un ensemble de méthodes nous permettant d'améliorer la robustesse du processus de rendu *global* : nous développons donc des méthodes ciblant plusieurs parties fonctionnelles d'un moteur de rendu. Plus précisément, nous avons développé des méthodes améliorant la robustesse de cinq parties d'un moteur :

- Intégration : le choix des chemins a un fort impact sur la robustesse d'un moteur utilisant la méthode de Monte-Carlo, et ce quel que soit l'algorithme précis utilisé pour la simulation. Dans le Chapitre 5, nous montrons comment améliorer la robustesse d'un grand nombre d'algorithmes

de simulation liés au rendu, tout en n'introduisant qu'un faible surcoût en temps de calcul.

- Reconstruction de l'image : en résumé, l'image finale est obtenue en moyennant l'énergie qui contribue à chaque pixel. Quand la méthode d'intégration, qui calcule cette énergie à partir d'échantillons, n'est pas parfaitement robuste, quelques valeurs d'énergie peuvent être très supérieure à la valeur exacte du pixel, donnant des pixels isolés très brillants (*points brillants*). Ce problème est dû au fait que rajouter ne serait-ce qu'une valeur très grande à un calcul de moyenne donne une sur-estimation très marquée de cette moyenne. Ces valeurs anormalement hautes sont appelées *valeurs aberrantes*. Le problème des points brillants peut donc être vu comme une conséquence du manque de robustesse de l'estimateur de la moyenne aux valeurs aberrantes. Dans le Chapitre 6, nous améliorons la robustesse de cet estimateur aux valeurs aberrantes en les détectant et en ne les ajoutant pas au calcul de la moyenne.

- Traitement d'images : les images calculées par un moteur de rendu sont traitées avant d'être affichées, premièrement pour qu'elles soient utilisables sans perte d'information sur un écran ou une imprimante (cette opération est appelée *correction de tons*). Ces traitements peuvent être basées sur des méthodes non-robustes, telle que diviser la valeur de tous les pixels par la valeur maximale sur les pixels. Cette méthode, ainsi que toutes celles basées sur du filtrage non robuste (qu'il soit Gaussien ou pas), donnera de mauvais résultats dès que des points brillants seront présents. Comme déjà dit ci-dessus, ces points brillants peuvent être causés par le fait que les méthodes d'intégration et de reconstruction d'images ne sont pas parfaitement robustes, mais ils doivent être traités avant l'application d'une quelconque méthode de traitement d'images. Nous présentons donc au Chapitre 7 une méthode pour recalculer la valeur des pixels étant des points brillants à partir des pixels avoisinants, évitant ainsi la présence de points brillants lors des différents traitements effectués par la suite sur l'image.

- Choisir les pixels pour lesquels des échantillons sont calculés : comme chaque échantillon ne contribue qu'à quelques pixels, ne devont choisir ces échantillons – et donc les pixels où les ajouter – de manière à ce que l'image soit aussi proche que possible d'une image "parfaite", en concentrant les échantillons là où l'erreur est la plus grande. Ceci est appelé de l'*échantillonnage adaptif*, et la majorité des méthodes d'échantillonnage adaptatif souffre d'un problème d'"arrêt prématuré" : un pixel pour lequel l'erreur est trouvé faible une fois n'est jamais reconsidéré. Nous montrons dans le Chapitre 8 que ceci peut amener à des artefacts très visibles sur l'image, et donc à un manque de robustesse, indépendamment de la robustesse des méthodes utilisées dans les autres parties fonctionnelles du moteur. Nous montrons une solution extrêmement simple à ce problème, qui peut de plus être appliquée à toute méthode d'échantillonnage adaptatif. De plus, nous montrons des moyens simples pour estimer correctement l'erreur d'un pixel même en présence de valeurs aberrantes, et comment réduire l'erreur aussi bien dans les zones ombragées que dans les zones fortement éclairées.

- Stabilité des temps de calcul : même si des méthodes robustes sont utilisées pour toutes les parties fonctionnelles d'un moteur de rendu, la description de la scène doit aussi être adéquate, afin

d'assurer une stabilité globale de l'ensemble du moteur de rendu. Dans le cas particulier des milieux participants, leur représentation peut avoir un impact non-néligeable sur les temps de calcul : certaines scènes requièrent dix ou cent fois plus de temps à calculer que d'autres, uniquement parce que les représentations des milieux participants ne sont pas adaptées aux opérations nécessaires pour faire du rendu basé sur la physique. Nous présentons donc dans le Chapitre 9 une représentation unifiée, qui peut être obtenue depuis n'importe quelle autre représentation, et qui assure des temps de calculs faibles et stables même pour des milieux complexes. Cette contribution requiérant une méthode d'intégration numérique extrêmement robuste et efficace, nous développons et présentons une telle méthode, qui peut être utilisée dans de nombreux cadres autres que celui de l'approximation des milieux participants.

Enfin, un moteur robuste qui soit également efficace serait très avantageux : les temps de calcul seraient ainsi stables et faibles pour un nombre très important de scènes. Afin de tendre vers cela, nous proposons comme dernière contribution dans le Chapitre 10 d'utiliser la puissance de calcul des ordinateurs actuels à son maximum, en incluant la grande capacité de calcul des cartes graphiques, pour la partie simulation du moteur du rendu, qui est la plus coûteuse.

Nous pensons que les contributions présentées dans ce document constituent une étape vers des méthodes de rendu robustes et rapides. Nous pensons aussi que ces contributions constituent une bonne indication de la thèse que nous défendons : des méthodes robustes devraient être utilisées pour chaque partie fonctionnelle d'un moteur de rendu, afin de s'assurer qu'aucune scène ne puisse demander de temps de calcul anormalement longs, tout ceci sans nécessiter une méthode de simulation gérant parfaitement toutes les configurations lumineuses possibles.

English version

Computer graphics is at the core of many domains nowadays. On the entertainment side, requirements in visual complexity and realism increase at a fast pace, being for special effects in movies, video games, or even advertisements. Computer graphics is also becoming a decision-helping tool: synthetic previews of objects or buildings allow architects or designers to take into account visual aspects that would have been hard to obtain before, and illustration can be of great help to explain difficult or abstract ideas, or simply catch the attention of an audience. In these two contexts, computer graphics can have multiple functions: demonstration, replacement of hard-to-build real objects by synthetic objects, simulation, and virtual world visualization. With respects to these functions, specific kind of graphics are often more adapted than others. Non-photorealistic rendering is advantageously used for sketches, blueprints or illustrations, as the freedom given by the different possible stylizations allows the user to focus on exactly what he wants to show. Even when photo-realism is desirable, several ways are possible, depending on the constraints: a low computational cost for games, a high level of artistic control for movies, very high accuracy at the cost of extremely large rendering times for predictive rendering (physically-accurate rendering), or lower rendering times at the cost of introducing minor approximations for illustration or simulation in scenes where the ignored effects do not have a large influence (physically-*based* rendering).

In this thesis, we focus on physically-based rendering: we aim at photo-realism, without requiring any tweaking from a content creator to reproduce the light behavior, in as many common-world environments as possible, *i.e.* environments where the few effects that are ignored do not have a large impact. We therefore need to simulate the behavior of light, choosing carefully the approximations which are done, to not restrict too much the kind of scenes we can handle.

The equations used for the simulation, derived in Chapter 1, take into account the physics of light, and how an image is obtained from this physical modelisation, including perception. The physics of light is based on radiometry and so-called light transport equations, which are part of the large field of radiative transfer. We show in the derivation that taking into account all of the phenomenons existing in radiometry-based light physics implies an extreme complexity, which would translate in extremely large computational times when dealing with large scenes, amongst other problems. We therefore explicit the approximations that are commonly done in physically-based rendering. The derived equations allow us to compute the image as seen from a camera for a scene containing light sources, objects and continuous media such as fog or smoke (called *participating media*). All these entities have to be described and represented in way suitable for the simulation, and we perform a brief non-exhaustive review of these representations in Chapter 2. The final equations obtained are integrals, which can not be computed

analytically for arbitrary scenes. Instead, numerical methods have to be used. Although several approaches exist, in this thesis we focus on the most commonly used method nowadays in physically-based rendering, the Monte-Carlo method, presented in Chapter 3.

Using these physical and mathematical aspects for rendering requires an adequate software architecture. As a first contribution, we describe in Chapter 4 a highly flexible and versatile software architecture, extending those of existing rendering engines. This software architecture is based on a functional decomposition of the equation of physics, taking into account the nature of the Monte-Carlo method as well as the algorithms that have been developed until now. We show that a large number of current state-of-the-art algorithms, related to various aspects of physically-based rendering, can be easily expressed in the abstract architecture we define.

The main part of a Monte-Carlo-based rendering engine is the numerical integration method, which is responsible for light transport simulation. To summarize, rendering with the Monte-Carlo method consists in using some “examples” (called *samples*) of the paths that can be taken by the light through a scene before reaching the camera, and to extrapolate the value that would be obtained if taking into account all of the possible paths. To our knowledge, there is no single method which can handle all of the possible scenes with good results for each: there are always some kind of scenes for which it fails, leading to extremely large rendering times. This sensitivity is a lack of *robustness*. This lack of robustness translates into computation times which are *not consistent*. All Monte-Carlo-based light simulation algorithms used nowadays have scenes where they behave very poorly. This is due to technical reasons not known by artists or everyday users, making them face inexplicably large rendering times in some cases. From a user point-of-view, rendering times should only depend on the global complexity of the scene, not on a particular lighting configuration or objects representations for which the specific rendering method of his 3D software behaves poorly.

Instead of trying to define an absolutely robust light transport simulation method, we choose in this thesis to develop a set of methods improving the robustness of the *global* rendering process: instead of focusing on making the integration part completely robust, we also develop other methods, targeted at other functional parts of the rendering engine, to handle the remaining lack of robustness. More specifically, there are five areas for which we have designed new methods to enhance the global robustness of an engine:

- Integration: the choice of the paths has a large impact on the robustness of Monte-Carlo-based rendering, whatever the actual algorithm and underlying mathematical formulation is. In Chapter 5, we show how to improve the robustness of many existing integration methods designed for rendering, while having a very low computational overhead.

- Image reconstruction: rapidly put, the final image is obtained by averaging the energy that contributes to each pixel. When the integration method, which computes this energy from samples, is not perfectly robust, some values can be much larger than the actual value, leading to very bright pixels in the image (*bright spots*). This problem arises because introducing a few very large and uncommon values leads to a large over-estimation of the actual average. These “abnormal” values are called *outliers*. The bright-spot problem can therefore be seen as a lack of robustness of the

average computation with respect to outliers for image reconstruction: a single outlier can lead to a completely wrong pixel value. We improve this robustness in Chapter 6 by detecting outliers and removing them from the average computation.

- Image processing: images from a renderer are post-processed before being displayed, first of all to match the display capabilities of screens or printers (an operation called tone-mapping). Some of these processings can be based on non-robust methods, such as taking the maximum over the pixels and dividing all the pixels by this value. This method, and many others such as the ones based on (Gaussian) filtering, will give poor results as soon as bright spots are present. As presented above, these bright spots can be caused by the non-absolute robustness of the integration and image reconstruction methods, and must be handled properly before further image processing. We therefore present in Chapter 7 a method to recompute the value of these pixels from neighboring pixels, avoiding the presence of bright spots in subsequent image processing steps.

- Choosing pixels where to add samples: as each sample contributes to only a few pixels, we have to choose the samples – and therefore the pixels where to add samples – so that the image is as close to the "perfect image" as possible, focusing samples where the error is larger. This is called adaptive sampling, and most methods suffer from a "premature stopping" problem: a pixel which is once found as close-enough from the exact value is never reconsidered. We show in Chapter 8 that this can lead to highly visible artifacts for some scenes, and therefore to a lack of robustness, even if the other functional parts of the rendering engine are robust. We show a very simple solution to that problem, which can be used with any existing adaptive sampling algorithm. Additionally, we show simple ways to correctly estimate the error of a pixel in presence of outliers, and how to reduce the error in shadowed zones as efficiently as in well lighted zones.

- Stable computation times: even if the rendering process uses robust methods, the scene description must also be adapted for stable computation times, to ensure a globally robust engine. In the specific case of participating media, their representation can have a large influence on computation times: some scenes require tens or hundreds times more time to compute than others, just because the participating media representations are not adequate. We therefore present in Chapter 9 a unified representation, which can be computed from any other representation, and which ensures fast and stable rendering times even for complex participating media. This contribution requires a highly robust, accurate and fast numerical integration method. We thus develop and present such a method, which can be used in a large number of contexts, related to participating media approximations or not.

Finally, a robust engine which is also efficient would be highly profitable: it would give stable and low rendering times for as many scenes as possible. In this attempt, we propose as final contribution in Chapter 10 to use the processing power of nowadays computer at their maximum, including the large processing power of the graphics processing units (GPUs), for the integration part, which is the most costly one.

We think that the contributions presented in this document constitute a step toward robust and fast rendering, and a good indication of the thesis we defend: robust methods should be used for each of the functions composing a rendering engine, to ensure that all scenes can be handled equally well, without requiring a universal light transport simulation method.

Part II

Context

1

Physical modelisation of light transport for rendering

1.1 Image, pixel, rendering process

An image, which is what we want to obtain, is a regular grid of pixels. A pixel is a purely computer-related notion. It has no area – and therefore no notion of center –, and no physical equivalence. Cameras compute the value of each pixel of a photograph by first obtaining a *measure* of the energy arriving during a given time at the surface of a 2D sensor (such as CCD sensors, or standard films) from every direction. Then, these measures are processed to mimic perception, in order to get images as close as possible to what is seen by a human. It is important to note that the measure obtained is not the actual value of energy received: it depends on the type of sensor, their sensibility, and their internal functioning.

It is therefore possible to distinguish three types of data: the density of energy at each point of the sensor for each direction and each wavelength, the measure M made by the sensor, and the final image. In cameras, it is not possible to directly obtain the energy density, only M is accessible. To the contrary, in rendering, no actual sensor is used, but the energy density is a quantity which can be obtained through physical simulation [DBB02, PH04]. Once it is known, M can be obtained by using a specific sensor model (film, or a model of the human eye). M can then be represented using an image whose pixels value can be as large as desired. These images are called *high dynamic range* (HDR) images [RWPD05]. As standard computer displays can not display such images, a dynamic-range reduction must be done to switch to values that can be handled by displaying devices, using *tone mapping* models which can take into account the way the human brain interprets the signals sent by the human eye. The final image is called a *low dynamic range* (LDR) image.

These three steps (computing the energy density, computing M, and finally computing the final LDR image) can be seen as a rendering process. In this thesis, we focus on the two first parts, which are solely based on physics, and take the last step as granted.

We now introduce the physical foundations of physically-based rendering.

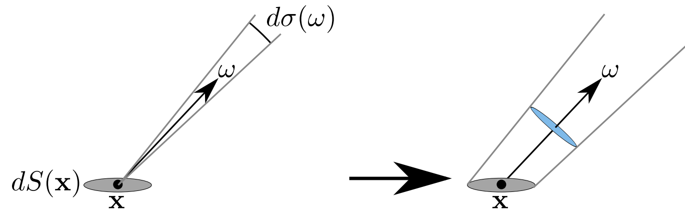


Figure 1.1: A ray's associated beam is the union of all the cones with an aperture corresponding to a solid angle $d\sigma(\omega)$ leaving from the points of an infinitesimal surface $dS(\mathbf{x})$. All the cones are identical. Left: the cone associated to the central point of $dS(\mathbf{x})$. Right: the ray's associated beam, obtained by taking the union of all the cones.

1.2 Radiometry

For each point (x, y) of the sensor, energy arrives from any direction at any time, for each wavelength. As billions of billions photons contribute to this energy, it is too costly to simulate them directly. Luckily, rendering can be done with simplifying assumptions which allow us to avoid photon-per-photon simulation. The main assumption is that we consider geometric optics: wave optics effects such as interference and polarization are ignored. Under this assumption, it is possible to consider several photons which have the same trajectory at the same time, and process them as a coherent light beam, also called *light ray*. The quantitative aspects of this physical modelisation is *radiometry*.

A light ray leaves from a surface in a given direction. It is characterized by a starting point $\mathbf{x}$ and a propagation direction ω. When traveling in a medium with a constant refraction index, the propagation direction remains constant. Otherwise, it can be bent, for instance when traveling in hot air.

A ray has an associated beam (*cf.* Figure 1.1), which is the union of cones oriented along ω, leaving from every point of an infinitely small surface $dS(\mathbf{x})$ centered around $\mathbf{x}$, of area $d\mathcal{A}(\mathbf{x})$. The beam cross section is perpendicular to ω, and each cone's aperture is an infinitesimal solid angle of size $d\sigma(\omega)$ centered around ω. The energy content of the ray is described by a per-wavelength measure of the density of energy inside the beam, called *spectral radiance*. As both the area and the solid angle are infinitesimal, it is in practice similar to a simple straight line.

The spectral radiance of a ray can have two forms, depending on whether the ray leaves from a point (*outgoing* spectral radiance, $L_{\lambda,o}$), or arrives on it (*incident* spectral radiance, $L_{\lambda,i}$). These two quantities are related to other incident or outgoing units: (spectral) irradiance, power and energy. As the equations are the same for incident and outgoing quantities, we illustrate for incident quantities, and the equations use a generic spectral radiance L_λ, as well as generic derived quantities. The only condition is to use only outgoing quantities or only incident quantities in the equations.

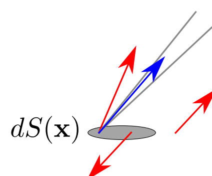


Figure 1.2: The blue photon hits the surface inside $dS(\mathbf{x})$ and its direction belongs to the cone associated to the hit point, so it contributes to the beam's radiance. The red photons do not contribute, either because they do not cross $dS(\mathbf{x})$ or because their incident direction does not belong to the solid angle.

1.2.1 From spectral radiance to energy: radiometric integrals

1.2.1.1 Spectral radiance

Spectral radiance is the density of energy per-unit-area per-solid-angle per-second and per-wavelength-unit which is inside a beam, being associated to a ray or not. Its unit is $[J.m^{-2}.st^{-1}.s^{-1}.nm^{-1}]$. Wavelenghtes are here measured in nanometers to better fit the visible spectrum ($380nm - 780nm$). In the remaining of this section, we only work on areas and solid angles, so the fact that the densities are also expressed per-second and per-wavelength-unit is implicit.

The incident spectral radiance of a ray hitting a surface at point $\mathbf{x}$, arriving from direction ω at time t is noted $L_{\lambda,i}(\mathbf{x}, \omega, t)$. Note that the ray has direction $-\omega$. Incident spectral radiance is independent from the surface: it is a measure made "just before" photons hit the surface. Incident spectral radiance can be visualized intuitively by considering only photons of wavelength belonging to a small interval $d\lambda$ centered around λ: incident spectral radiance is the density at the beam cross section of such photons which hit the surface $dS(\mathbf{x})$, and whose incident direction belongs to the solid angle of size $d\sigma(\omega)$. Figure 1.2 shows an example of photons taken into account or not when computing the radiance.

Symmetrically, the outgoing spectral radiance of a ray leaving a surface at point $\mathbf{x}$ at time t is noted $L_{\lambda,o}(\mathbf{x}, \omega, t)$. Outgoing spectral radiance is independent from the surface: it is a measure made "just after" photons leave the surface.

From spectral radiance at a point $\mathbf{x}$ on a surface with normal $\mathbf{N_x}$, it is possible to obtain the density of energy received or emitted on an infinitely small surface centered around $\mathbf{x}$. This is called spectral *irradiance*.

1.2.1.2 Spectral irradiance

Incident spectral irradiance $E_{\lambda,i}$ is the density of energy arriving at a point $\mathbf{x}$ from all the directions which lie inside a solid-angle Ω. Its unit is $[J.m^{-2}.s^{-1}.nm^{-1}]$. It consists in summing the contribution of all the rays arriving at $\mathbf{x}$ whose incident direction belongs to Ω, at time t.

Symmetrically, outgoing spectral irradiance $E_{\lambda,o}$ is the density of energy emitted by a surface $dS(\mathbf{x})$ centered at $\mathbf{x}$, summed on all directions of Ω.

As spectral radiance is a density with respect to solid-angle and area, we have to multiply the per-solid-angle energy density of the incident beam by the actual solid angle covered by it ($d\sigma(\omega)$) to obtain

the density per-unit-area. Moreover, as the ray comes (for incident irradiance) or leaves (for outgoing irradiance) at an angle $\theta = \mathbf{N_x} \cdot \omega$, the energy density is spread on a larger area, by a factor $|cos(\theta)|$. This spreading is similar to the one responsible for lower temperatures when the sun is at more grazing angles. This leads to the following contribution of spectral radiance arriving at or leaving from $\mathbf{x}$ from direction ω (the intricate notation comes from the differential formulation (Section 1.2.2):

$$d_\Omega E_\lambda(\mathbf{x}, \omega, t) = L_\lambda(\mathbf{x}, \omega, t) d\sigma\left(\omega\right) \left|\mathbf{N_x} \cdot \omega\right|. \tag{1.1}$$

Spectral irradiance $E_\lambda(\mathbf{x}, \Omega, t)$ is obtained by summing these contributions for all the directions in Ω:

$$\begin{aligned} E_\lambda(\mathbf{x}, \Omega, t) &= \int_\Omega d_\Omega E_\lambda(\mathbf{x}, \omega, t) \\ &= \int_\Omega L_\lambda(\mathbf{x}, \omega, t) \left|\mathbf{N_x} \cdot \omega\right| d\sigma\left(\omega\right). \end{aligned} \tag{1.2}$$

It is often the case that spectral irradiance is computed for all the directions, *i.e.* $\Omega = \mathcal{S}^2$ where $\mathcal{S}^2$ is the 2D unit sphere. In this case, the Ω argument is dropped, and we have:

$$E_\lambda(\mathbf{x}, t) = \int_{\mathcal{S}^2} L_\lambda(\mathbf{x}, \omega, t) \left|\mathbf{N_x} \cdot \omega\right| d\sigma\left(\omega\right). \tag{1.3}$$

1.2.1.3 Forgetting the wavelength: radiance and irradiance

Radiance and Irradiance are the same as their spectral counterparts, except that the contribution to the total energy of each wavelength is summed. As the spectral versions of each quantity is a per-wavelength density, the contribution of a single wavelength has to be multiplied by the infinitely small wavelength interval $d\lambda$ it spans. We therefore have:

$$L_\Lambda(\mathbf{x}, \omega, t) = \int_\Lambda L_{\lambda,i}(\mathbf{x}, \omega, t) d\lambda \tag{1.4}$$

$$E_\Lambda(\mathbf{x}, \Omega, t) = \int_\Lambda E_{\lambda,i}(\mathbf{x}, \Omega, t) d\lambda. \tag{1.5}$$

From irradiance, it is possible to compute the per-second energy density which is received or emitted by a surface S: *power*.

1.2.1.4 Power

Power Φ, also called *flux*, is an instantaneous measure at time t of the energy received or emitted by a surface S per unit of time, from a solid-angle Ω. It is expressed in $[J.s^{-1}]$, or $[W]$ as Joules per second is also noted *Watt* ($[W]$).

Similarly to directions for incident irradiance, incident power is obtained by summing the contribution of all the infinitesimal surfaces centered at each point of the surface S. The energy received per unit of time on the infinitesimal surface $dS(\mathbf{x})$ centered at point $\mathbf{x}$ is simply the density per unit-area

multiplied by the area of $dS(ptx)$, $d\mathcal{A}(\mathbf{x})$. This leads to a contribution for point $\mathbf{x}$ equal to:

$$d_S\Phi(\mathbf{x},\Omega,t) = E_\Lambda(\mathbf{x},\Omega,t)d\mathcal{A}(\mathbf{x}). \tag{1.6}$$

Therefore, the energy received per time-unit at time t for surface S from a solid-angle Ω at each point of S is:

$$\Phi(S,\Omega,t) = \int_S E_\Lambda(\mathbf{x},\Omega,t)d\mathcal{A}(\mathbf{x}). \tag{1.7}$$

Similarly to irradiance, it is often the case that all the directions are considered, leading to:

$$\Phi(S,t) = \int_S E_\Lambda(\mathbf{x},t)d\mathcal{A}(\mathbf{x}). \tag{1.8}$$

1.2.1.5 Energy

Finally, the total energy Q_i (measured in $[J]$) received by a surface during a time interval T is given by summing the contributions $d_TQ(S,\Omega,t) = \Phi(S,\Omega,t)dt$ at each time t:

$$Q(S,\Omega,T) = \int_T \Phi(S,\Omega,t)dt \tag{1.9}$$

1.2.2 From energy to spectral radiance: differential formulation

Radiometric units are often presented from global to local, leaving from energy to reach radiance. From the equations above, which leave from rays, the basic elements of light transport as we want to simulate it, we obtain

$$Q(S,\Omega,T) = \int_T \int_S \int_\Omega \int_\Lambda L_\lambda(\mathbf{x},\omega,t)\,|\mathbf{N_x}\cdot\omega|\,d\lambda d\sigma(w)d\mathcal{A}(\mathbf{x})dt \tag{1.10}$$

which links energy, the most global radiometric quantity, to its finest quantity, spectral radiance. The differential formulation helps to better see that all the intermediate quantities are densities. As a matter of fact, the flux at a given time t' is obtained by dividing the energy received or emitted during the interval dt' (centered around t') by the length of the interval, dt'. The function which computes the total energy during dt' is derived from the expression of Q by just removing the sum over all the time values of T, only considering the interval associated to the specific time t'. We note this function d_TQ, explicitly telling that we remove the sum on T:

$$d_TQ(S,\Omega,t') = dt' \int_S \int_\Omega \int_\Lambda L_\lambda(\mathbf{x},\omega,t')\,|\mathbf{N_x}\cdot\omega|\,d\lambda d\sigma(w)d\mathcal{A}(\mathbf{x}). \tag{1.11}$$

This energy is then converted to an instantaneous emission rate by dividing by dt':

$$\Phi(S,\Omega,t') = \frac{d_TQ(S,\Omega,t')}{dt'}. \tag{1.12}$$

Here, we have used t' instead of t to avoid the confusion between the variable of integration t and the specific time value we are interested in, t'. This distinction is not made in the remaining of this section.

In most radiometry presentations, t is directly used, and arguments are not put, leading to a much more concise expression:

$$\Phi = \frac{dQ}{dt}. \tag{1.13}$$

Applying similar methods and notation shortening, we have:

$$E_\Lambda(\mathbf{x}, \Omega, t) = \frac{d_S\Phi(\mathbf{x}, \Omega, t)}{d\mathcal{A}(\mathbf{x})} \tag{1.14}$$

$$E_\Lambda = \frac{d\Phi}{d\mathcal{A}(\mathbf{x})} \tag{1.15}$$

for irradiance, and

$$L_\Lambda(\mathbf{x}, \omega, t) = \frac{d_\Omega E_\Lambda(\mathbf{x}, \omega, t)}{|\mathbf{N_x} \cdot \omega|\, d\sigma(\omega)} \tag{1.16}$$

$$L_\Lambda = \frac{dE_\Lambda}{|\mathbf{N_x} \cdot \omega|\, d\sigma(\omega)} \tag{1.17}$$

for radiance.

1.2.3 Relation between incident and outgoing quantities

Some relations exist to link incident and outgoing quantities.

Radiance: *In free space*, a beam's spectral radiance is conserved at every point:

$$L_{\lambda,o}(\mathbf{x}, \omega, t) = L_{\lambda,i}(\mathbf{x}, -\omega, t). \tag{1.18}$$

This equality is valid for any point $\mathbf{x}$ which is not on a surface, and every direction ω. On surfaces and in mediums where interactions occur (being gazes or mediums with non-constant refraction indices), the relation is more complex and depends on the local interactions. As the equality is valid for any wavelength, the radiance is also conserved at any point in free space:

$$L_o(\mathbf{x}, \omega, t) = L_i(\mathbf{x}, -\omega, t). \tag{1.19}$$

Irradiance: Rendering models consider that for any time t, thermodynamical equilibrium is reached. This means that each surface point's temperature is constant during an infinitely small interval dt centered around t. A practical impact of this is that the energy leaving the surface at t only depends on the energy that arrives during an infinitely small amount of time. When a surface point absorbs electromagnetic energy (or more precisely, the infinitesimal surface around this point absorbs electromagnetic energy), the temperature at this point increases. When it emits the same amount of energy (possibly in different wavelengths), its temperature decreases by the same amount. Therefore, at thermodynamical equilibrium, a point must emit as much energy as it receives in order to maintain a constant temperature.

This means that, for any point, the sum of the received energy for all wavelengths must equal the sum of the emitted energy for all wavelengths. This is the same for densities, so we have the following identity:

$$E_o(\mathbf{x}, \mathcal{S}^2, t) = E_i(\mathbf{x}, \mathcal{S}^2, t), \tag{1.20}$$

where $\mathcal{S}^2$ is the unit sphere.

Note that this equality is valid only when considering all the wavelengths when computing irradiance from its spectral counterpart, and does not hold for spectral irradiance. In particular, this is not valid when limiting Λ to the set of visible wavelengths ($380nm - 780nm$). For instance, surfaces often emit energy in infrared wavelengths, which are then considered as radiant heat by neural sensors in human skin, and responsible for the warm feeling near a hot surface even if not touching it.

Power and energy: as power and energy are defined as the integral of irradiance, the two following equalities apply:

$$\Phi_o(S, \mathcal{S}^2, t) = \Phi_i(S, \mathcal{S}^2, t) \tag{1.21}$$
$$Q_o(S, \mathcal{S}^2, T) = Q_i(S, \mathcal{S}^2, T) \tag{1.22}$$

These three last equalities motivate the fact that irradiance, power and energy often appear without an indication about their ingoing or outgoing nature.

1.2.4 Spectral distributions

It is often useful to use spectral radiance or irradiance distributions as one single object. From now on, we will denote L or E (without λ or Λ subscripts) such spectral distributions. They are defined as follows:

$$L(\mathbf{x}, \omega, t)(\lambda) = L_\lambda(\mathbf{x}, \omega, t)$$
$$E(\mathbf{x}, \Omega, t)(\lambda) = E_\lambda(\mathbf{x}, \Omega, t)$$

For any operator f handling spectral quantities, a natural extension to handle spectral distributions is defined by applying f on each wavelength separately. For instance, $(L_1 + L_2)(\lambda) = L_1(\lambda) + L_2(\lambda)$.

1.3 Colorimetry

Radiometry is the framework for electromagnetic energy, and deals with wavelengths and energy. Part of the energy received on a sensor is transformed into an image through a process of *perception*. This process transforms a distribution of energy with respect to wavelengths into abstract entities, the *colors* [WS00].

For instance, a thermal sensor will convert the distribution of energy in the infrared wavelengths to colors through a specific perception process, ignoring all other wavelengths. In the case of the human vision system, cones and rods on one side, and the brain on the other side, will transform a distribution of energy density with respect to the visible wavelengths ($380nm - 780nm$) into a "mental image" (although it is not formed of pixels). Cones and rods are the sensors, and the brain interprets the measurements, the whole being the *human vision system* (HVS). Put together, this leads to an image, the complete process being perception. In the case of a camera sensor, CCD or CMOS sensors will transform this distribution into electric voltage for each pixel, which is then interpreted by specific equipments (software or hardware) to obtain the color of each pixel of the image.

Colorimetry deals with representing colors and giving quantitative tools linked to human perception, as well as switching from energy distributions with respect to the visible wavelengths to colors. It has been established that three type of cones are present in the human eye. Therefore, three numbers are enough to describe any color that can be perceived by the HVS. Note that the transformation from a spectral distribution to a color is not a bijection: several spectral distributions can lead to the same perceived color.

These colors are represented through various so-called color spaces. Some of these spaces can represent all the colors, they are called *reference spaces*. Some others can only represent a portion of these colors, but manipulation of colors in these spaces might be easier. In general, all these spaces are shown relatively to a reference space called *CIE xyY*.

1.3.1 Chromaticity, luminance, gamut

The CIE xyY color space is obtained by splitting a color in two distinct and independent characteristics: the luminance Y (which is the perceived brightness of a color), and two chromaticity values, x and y. It is often the case to represent this space using a constant brightness value, which leads to the color schema in Figure 1.3.

Manipulations in the xyY color space are not easy: addition of two colors is not done just by adding each components in a component-wise way, as chromaticity components could be outside of the graph. This is why reference spaces are not used as working color spaces.

Working color spaces can not represent all the colors. For such a color space, the extent of the colors it can represent is called *gamut*, or *color profile*. Imaging devices such as printers or screens use specific color spaces, with associated color profiles. When printing an image which is displayed on a screen, the colors must first be converted into the color space of the printer, using a reference space such as xyY as an intermediate to minimize distortion if the color profiles do not match (*i.e.* , there are colors that can be represented by a screen, but which can not be reproduced by the printer).

1.3.2 RGB, gamma correction, white point, sRGB

RGB color spaces are defined as triangle-shaped spaces inside the chromaticity schema, and are based upon the fact that human eyes have sensors sensitive to red, green and blue colors. To define a RGB color space, it is enough to associate a (x, y) coordinate to each of the RGB coordinates $(1, 0, 0)$, $(0, 1, 0)$ and

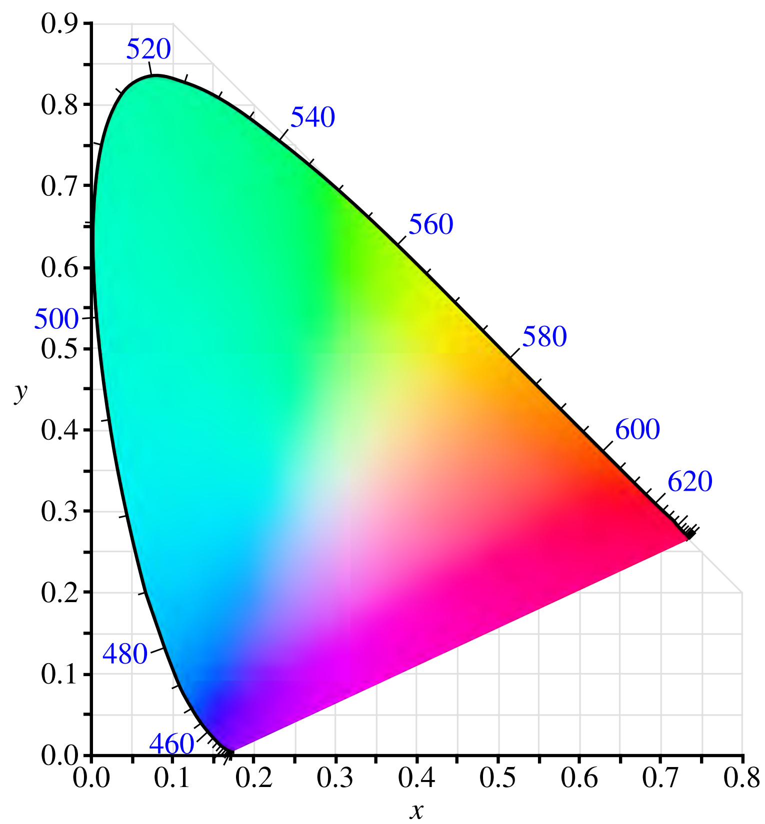


Figure 1.3: CIE xy schema, for a constant luminance Y. This color schema is displayed using a device which does not use a reference space. Therefore, it can not recreate all the colors, which makes the displayed or printed version inexact. Note that the boundary of the graph is made of the colors that are obtained from spectra where a single wavelength is present. Source : Wikipedia.

$(0, 0, 1)$, all the other colors being obtained by interpolating between these colors. As long as all the interpolation weights are positive (the weights being in fact the RGB coordinates), the obtained color chromaticity belongs to the triangle. When the weights sum is 1, the color belongs to the triangle. Otherwise, the luminance is different.

Display devices use RGB color spaces, but at the time of the confection, cathode-ray-tube (CRT) displays were used. In these systems, the process of displaying an image is non-linear in term of brightness: an input value twice as large leads to a perceived brightness which is much more than the double. A good approximation of the alteration is that the perceived brightness follows a power law of the form $B = V^{\gamma}$. To achieve a correct perception, it is therefore necessary to gamma-correct the RGB values, by applying a transformation of the form $V_{out} = V_{in}^{1/\gamma}$. In image formats such as JPEG, the gamma-corrected values are directly stored, as well as the γ value used. Therefore, when using an image as a texture for rendering, where no gamma-correction is needed, *the original values must be retrieved first*, by applying the inverse transform. A γ value is therefore necessary to further specify a RGB color space.

Moreover, a display produces perceived white (equal energy for all wavelengths) for a given chromaticity value, and a camera associate a given chromaticity for a given spectral distribution. The chromaticity value for which white should be displayed is called *white point*. If an image captured with a camera is displayed on a screen with different white point, colors will not appear as expected. A white point is therefore necessary to fully describe a RGB color space.

To summarize, a RGB color space is defined by the (x, y) chromaticity values associated to each of the $(1, 0, 0)$, $(0, 1, 0)$ and $(0, 0, 1)$ RGB coordinates, the gamma correction to apply, and the white point. When two devices do not use the same RGB color spaces, it is necessary to switch from one to another, which can lead to artifacts if one of the characteristic differs largely.

To avoid this and the existence of a lot of different RGB color spaces for printing or screening (one per imaging device), a specific standard RGB color space has been designed: sRGB. This color space is used by most displays and cameras, and is shown in Figure 1.4. Note that the gamut is quite reduced. The $D65$ point is the white point of sRGB. The particularity of sRGB comes from the fact that the gamma-correction used is not characterized by a single γ value, but is a more intricate function which matches more closely the actual CRT behavior.

1.4 From radiometry to colorimetry

As the simulation phase gives spectral quantities, it is necessary to switch from these spectral data to colors. We focus on two color spaces for which this conversion is possible. The human eye uses cones to translate spectral energy into electric impulses, rods being used for night vision as they are much sensitive, at the price of not allowing the brain to perceive colors. It has been established through perceptual studies that three type of cones are used [WS00], each having different sensitivity with respect to wavelengths. The first type is sensitive to wavelengths around $700nm$ (which corresponds to reddish colors), the second type is more sensitive to wavelengths around $546nm$ (green) and the last one is sensitive to wavelengths around $435nm$, leading to blue.

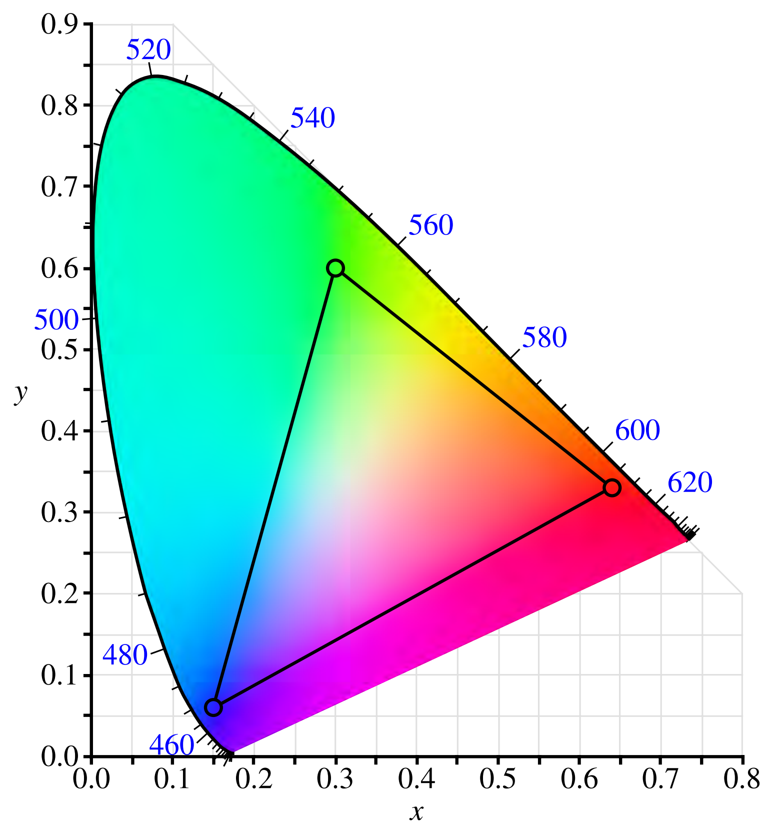


Figure 1.4: The interior of the black triangle is the sRGB gamut, *i.e.* the set of colors that can be represented in the sRGB color space. Source : Wikipedia.

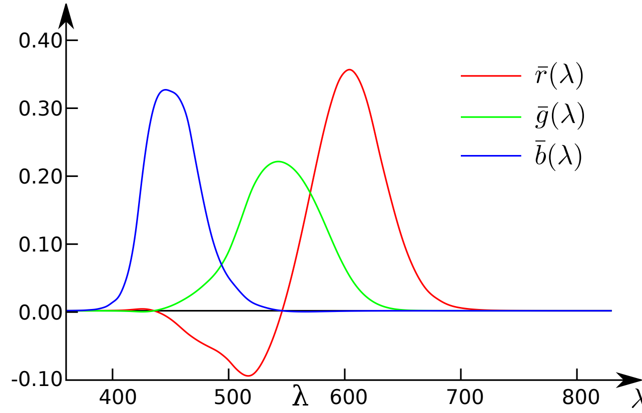


Figure 1.5: CIE RGB color matching functions. Note that the red matching function takes negative values, which shows that these functions are not sensitivity models. Source : Wikipedia.

1.4.1 Spectrum $\rightarrow$ RGB conversion

These observations, together with the will to define a RGB color space, has lead to the definition of specific color-matching functions, which make it possible to compute RGB coefficients for the CIE RGB color space from spectral data. These functions, shown in Figure 1.5, give the contribution of each wavelength to the three main colors (called primary colors), namely red, green and blue.

The associated color space is the CIE RGB color space. The gamut of this space is shown in Figure 1.6. The three main points correspond to spectra with a single wavelength present, the one of the corresponding cone sensitivity peak.

These response functions $\bar{r}(\lambda)$, $\bar{g}(\lambda)$ and $\bar{b}(\lambda)$, are defined over the interval $[380nm, 780nm]$, and make it possible to obtain the R, G, and B components from a spectral density $S(\lambda)$ with unit $[W.m^{-2}.sr^{-1}.m^{-1}]$, which is spectral radiance integrated over time. The three components are obtained as:

$$R = \int_{380nm}^{780nm} S(\lambda)\bar{r}(\lambda)d\lambda \quad (1.23)$$

$$G = \int_{380nm}^{780nm} S(\lambda)\bar{g}(\lambda)d\lambda \quad (1.24)$$

$$B = \int_{380nm}^{780nm} S(\lambda)\bar{b}(\lambda)d\lambda. \quad (1.25)$$

1.4.2 Spectrum $\rightarrow$ CIE XYZ $\rightarrow$ RGB

The conversion method above has problems, as CIE-RGB can not represent all colors. Moreover, display devices do not necessarily use CIE-RGB as color space. Therefore, if the gamut of the device's color space is not included in the one of CIE-RGB, the conversion will lead to a degradation. It is therefore preferable to represent images with a complete color-space, such as CIE-XYZ, and then convert from this space to the final color space (such as sRGB or any other color space, even subtractive ones based on CMYK, for printers) only when required.

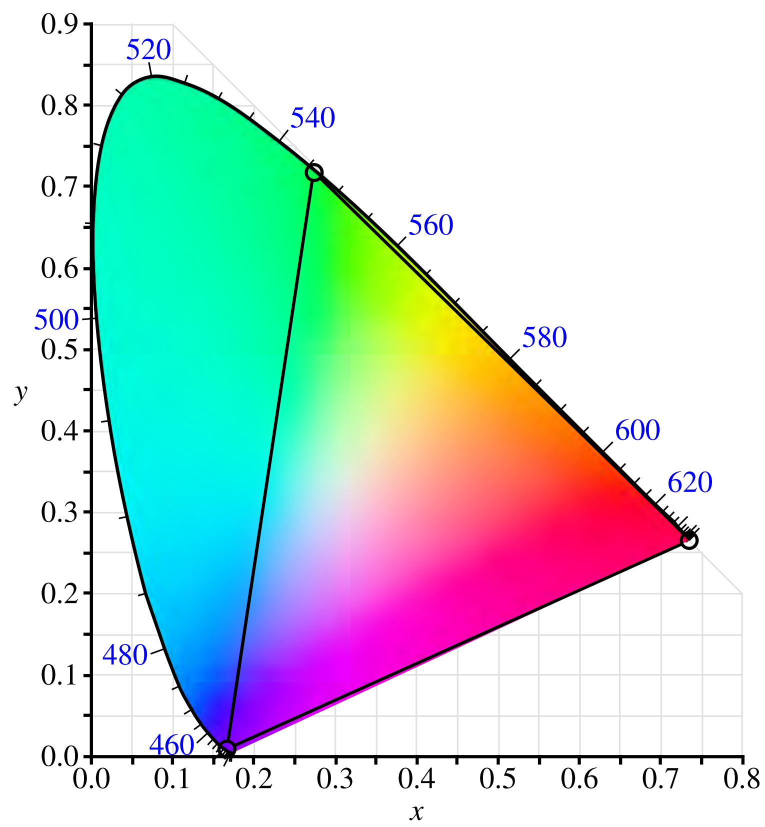


Figure 1.6: CIE RGB color space gamut. This color space includes the sRGB color space. Source : Wikipedia.

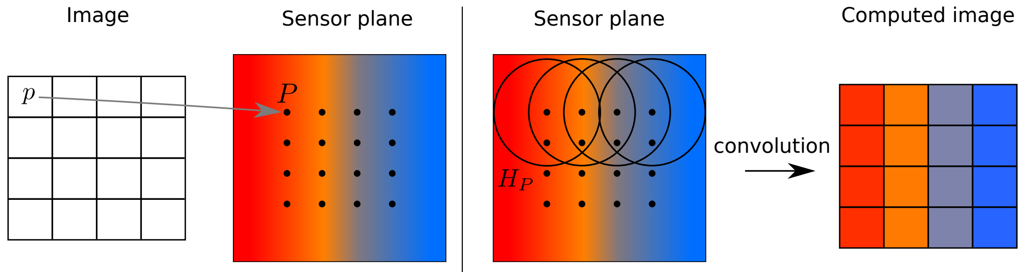


Figure 1.7: Image and sensor relationship. Left: we associate to each pixel p a point P on the image plane (the spatial extent of the sensor). Right: The value of pixel p in the image is equal to the convolution of a filter centered at P with the signal on the sensor.

This is why the usual conversion pipeline is a bit more complicated than the one presented above. The conversion from a spectrum to CIE-XYZ is done similarly to the conversion from spectra to CIE RGB, through the use of color-matching functions $\bar{x}, \bar{y}, \bar{z}$. Note that these matching functions do not exhibit negative values, as every spectra can be encoded to a valid XYZ triplet.

Then, conversion from CIE-XYZ to the final color space is done through methods which are specific to the final color space. For RGB color spaces, this is usually done through matrices and clamping of negative values.

1.5 Content of a HDR image, sensor

In rendering, the high dynamic range image represents what is measured by a sensor (such as CCD sensors or the retina) [RWPD05]. This sensor is often assumed to be planar, and its spatial extent is called *image plane*. The high dynamic range image can be seen as a punctual approximation of a 2D continuous signal S representing what is measured on the sensor. S has values in a specific color space, such as CIE-XYZ or sRGB. In rendering, S is the signal obtained by simulating the response of the sensor to the incident radiance, taking into account its sensitivity and other characteristics (directional sensitivity, *etc.*). The pixel values are then computed such that a continuous signal rebuilt from them is as close as possible from the original signal.

A way to do it is to associate a point P of coordinates (x_P, y_P) on the sensor to each pixel p, so that the set of points on the sensor form a regular grid and cover the active zone of the sensor. The value I_p of the pixel p is then computed as the convolution of the signal with a reconstruction filter h_P centered at point P on the image plane:

$$I_p = \int_{s(h_P)} h_P(\Delta_x, \Delta_y) S(x_P + \Delta_x, y_P + \Delta_y) d\mathcal{A}(\Delta_x, \Delta_y) \tag{1.26}$$

where $s(h_P)$ is the support of the filter h_P. Coordinates $(\Delta_x = 0, \Delta_y = 0)$ correspond to the center of the filter. The complete process is illustrated in Figure 1.7. Note that the image plane needs to be at least as large as the union of all the filter supports for a correct computation of border pixels.

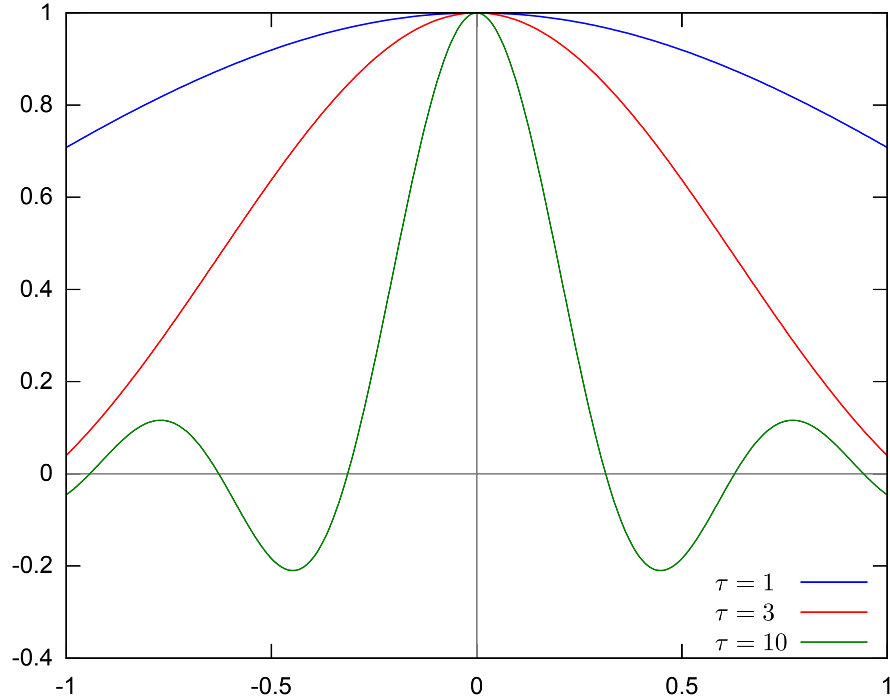


Figure 1.8: 1D Lanczos-sinc filter function on $[-1, 1]$, for increasing values of the τ parameter.

1.5.1 Reconstruction filter

The filter should be normalized (*i.e.* its integral over $s(h_P)$ is equal to 1) to avoid that the range of values of an HDR image depends on the size of the filter support. It is often the case that the filter's support of neighboring pixels overlap. Note that the larger the filter support, the more blurry the computed image is. Some known filters in the reconstruction literature are the box filter (the simplest, it is uniform over its support), the triangle filter, or the Gaussian filter. A good approximation of an optimal reconstruction filter is the Lanczos-sinc filter. Its support has a rectangular shape. Letting u' and v' be the (u, v) coordinates scaled between -1 and 1 in the support, $h(u, v)$ is given by:

$$h(u, v) = \frac{sin(u' \times \tau)}{u' \times \tau} \times \frac{sin(u')}{u'} \times \frac{sin(v' \times \tau)}{v' \times \tau} \times \frac{sin(v')}{v'}. \tag{1.27}$$

τ is a user parameter which controls the global shape of the filter: lower values will make it similar to a Gaussian filter, while larger values will make it oscillate, as illustrated in the 1D-case by Figure 1.8.

1.5.2 Signal value, sensor response

The value $S(x, y)$ of the signal at any point of the image plane (x, y) is obtained by summing the contribution of each light ray hitting the image plane, taking into account the sensor response. On chemical films, photons create splotches, therefore a beam of energy can contribute to a region of the film instead of only contributing to a single point. However, in rendering, it is common to consider that a beam contributes only to the point where it hits the sensor, and we will assume it is the case in the

rest of this document. Depending on the sensor, a directional sensitivity and a wavelength dependency can also be present. However, we assume that the sensor response does not exhibit non-linear effects: its response does not depend on the rays that have been taken into account before. The effect of this behavior would not be negligible if the sensor to simulate was easily saturated, its response decreasing with the amount of energy already absorbed, but we assume that our sensor does not have such properties. As we additionally consider that no cross-wavelength interactions occur and that the response is the same for each point of the sensor, the response function can be modeled as a simple direction-dependent spectral distribution weighting function $R(\omega)$.

A sensor produces measures in a given color space. We take CIE-XYZ as it is a complete space, which avoids color-loss (Section 1.4.2). Therefore, $S(x, y)$ is a XYZ triplet. Each component of this color space value is obtained by summing the spectral contributions for each incident direction and each time value. For instance for the X component:

$$S_X(x,y) = \int_T \int_{\mathcal{S}^2} \int_\Lambda \bar{x}((R(\omega) dE_i(x,y,\omega,t))(\lambda) d\lambda dt \tag{1.28}$$

where the fact that the measure is done after the photons have hit the sensor's surface is taken into account by considering $dE_i(x', y', \omega, t)$ instead of $L_i(x', y', \omega, t)$. Y and Z are computed using similar equations.

If the direction has no effect (*i.e.* , $R(\omega) = 1\ \forall\omega$), the expression can be further simplified:

$$\begin{aligned} S_X(x,y) &= \int_T \int_{\mathcal{S}^2} \int_\Lambda \bar{x}((R(\omega) dE_i(x,y,\omega,t))(\lambda) d\lambda dt \\ &= \int_T \int_\Lambda \bar{x}(\lambda) E_{\lambda,i}(x,y,\mathcal{S}^2,t) d\lambda dt \end{aligned}$$

and similarly for the Y and Z components. If the response function is independent from the direction, spectral irradiance can be used directly instead of dealing with spectral radiance when computing the final signal value.

1.6 Light interactions

Now that measurement of energy and conversion to colors by a sensor is described, we have to know how it reached the sensor in the first place. Light reaching the sensor has been emitted by light sources, and gone through interactions with the elements of the scene: objects, gazes, *etc.*. Light can interact in three ways with matter: it can be self-emitted, absorbed or scattered. Each of these interactions are best described intuitively at the photon level, but are handled in a statistic fashion in radiometry.

1.6.1 Self-emission

In real world, photons can be created through several atomic processes. We present incandescence to illustrate a way of creating photons, as it is one of the most common type of emission used. As a matter of fact, incandescence is responsible for the emission of light by light bulbs and hot gazes.

The emission of photons by an electron is described by atomic models from the quantum theory. Here we use Bohr's atomic model. More advanced models such as the one proposed by Erwin Schrödinger provide a much deeper understanding and a much more aspirin-demanding formulation of the quantum nature of electrons and other particles. These models have profound consequences for the understanding of our world and the notion of reality, but they are not useful for radiometry, so we do not have to go deeper.

According to Bohr's model, electrons spin around the nuclei on orbitals, and have well-defined energy levels (there are several energy levels possible for each electron). An electron can change its energy level by absorbing or emitting energy. When an electron attached to a nuclei has increased its energy level because of thermal agitation (which is the source of incandescence), it then drops back to a lower level of energy by emitting a photon, whose wavelength is given by $\lambda = c \times h/\Delta E$, where c is the speed of light in empty space, h is the Planck's constant, and ΔE the difference of energy between the initial and final energy levels of the electron. As these energy levels are determined by the chemical composition of a material (being a gaze, a liquid or a solid), a given material can emit only in limited ranges of wavelengths. These bands are called "emission lines". When these emission lines are at least partly in the visible range, the material can be used as a light source. In rendering, this is modeled through a simple radiance emission function, $L_e(\mathbf{x}, \omega)$.

1.6.2 Absorption and scattering

An electron can increase its energy level by absorbing a photon, switching from a level of energy E_1 to a level of energy E_2. The only condition for that is that the photon's energy is precisely $E_2 - E_1$, or equivalently its wavelength is equal to $c \times h/(E_2 - E_1)$. Taking into account all the possible changes of level of energy which are present in a material, this leads to the absorption of some wavelengths, while others are not affected. This is responsible for the colors of materials: under a "white spectrum" illumination (equi-energy repartition in the visible spectrum), a material will absorb partially or totally the energy of some wavelengths, and therefore the spectrum measured after the interaction will not be the same.

An excited electron is not stable, it has to go down to a stable energy level. Doing so, it will emit a photon, exactly as described for incandescence, as this is the same phenomena. The difference is that we speak of self-emission when the electron was not excited by the absorption of a photon.

Two key points have to be considered. First, an electron can absorb a photon while being excited, which makes it jump to a higher level of energy. Second, the descent to a stable energy level can be made by one big jump, or several. Each jump will lead to the emission of a photon. For the case of light (electromagnetic energy in the visible range), we can distinguish four type of behaviors, with different ways of handling them.

- When only one photon was absorbed and one jump is done back to the stable level, the two photons have the same energy, we speak of *scattering*.
- When at least one of the absorbed photon was in the visible spectrum and no emitted photon is in

the visible range, we speak of *absorption*: there is a loss of light.

- When one or more of the emitted photon is in the visible spectrum, and has a different wavelength than any absorbed photon, we speak of *fluorescent* scattering. This kind of scattering is taken into account in bi-spectral rendering methods, but is not taken into account in this document, and the physical modelisation thereafter ignores it.
- Neither any absorbed photon nor any emitted photon is in the visible range: no light is implied. It is most common to simply ignore it, although it can lead to very subtle errors if the re-emitted photons are then transformed into light *via* fluorescent scattering in the visible range.

Note here the main difference implied by this modelisation between light (photons with a restricted wavelength range) and photons without this restriction: with light, some energy can be "lost" after an interaction even at thermal equilibrium because of absorption, while when considering the full wavelength range, no such loss occurs.

1.7 Macroscopic models of light interaction

The previous section introduces the microscopic mechanisms at use for light interactions. As often in physics, three levels of description exist: microscopic (photons, characterized by their wavelength), mesoscopic (electromagnetic waves, a set of photons of same wavelength, with notions of electric and magnetic fields, polarization, phase, *etc.*), and macroscopic (light beams, which are a set of electromagnetic waves of various wavelengths). Switching from a precise type of description to a coarser one is done by aggregating the behaviors of the individual constituents of the description (the photons that constitute an electromagnetic wave, or the electromagnetic waves that constitute a light beam), to obtain more practical models, but often restricted to certain types of situations.

As rendering deals with light beams (Section 1.2), the statistical description of light interaction is obtained by first building mesoscopic descriptions in terms of electromagnetic waves, and then deriving a per-light-beam macroscopic description. In rendering, two kind of situations are considered, leading to two different macroscopic models. These situations can be obtained from the *mean free path length*. It is the average length between two interactions of an arbitrary photon in a medium.

On the one hand, if this average length is short ($\leq \alpha \times \lambda$ with λ the wavelength of the photon and α a small number), then we consider that we deal with standard solid objects (a chair, a piece of wood, ...). In this case, as light almost never reaches the inner parts of the object, we represent only its surface, and model the light interaction at each point of the surface, ignoring the inner points. For a light beam, we consider that there is a single interaction between the light beam and the photons.

On the other hand, materials for which the mean free path length is large lead to *participating media*. These media are important in order to correctly simulate light transport in smoke, gas, semi-transparent liquids, *etc.*. For a given light beam, we can consider that there is an interaction at each point along the beam. Light interaction has therefore to be modeled at each point of the volume containing the participating medium.

Note that for some materials such as liquids or glass, both representations are needed for an accurate simulation, as refraction occurs at the surface of the liquid, and then scattering occurs inside the liquid for the transmitted beam.

Self-emission is the simplest of the three interactions to model: a simple function $L_e(\mathbf{x}, \omega)$ is enough to fully describe a light-emitting surface or volume. This function can be based on real-world data of actual illuminants or emitting gazes such as the sun or fire, or created from scratch by artists.

By contrast, absorption and scattering require separate descriptions and tools.

1.7.1 Absorption and scattering at surfaces

1.7.1.1 Bidirectional scattering diffusion function (BSDF)

As the material is very dense, it has a sense to consider that the object boundary (its surface) is enough to completely describe its light interaction properties, and to consider that this surface has an orientation at each point. This orientation is given at any point by the *normal vector* at that point.

Depending on the electromagnetic properties of the material (conductor, insulator, dielectric, *etc.*), several mesoscopic behaviors can be obtained, describing what happens to an electromagnetic wave reaching an infinitesimally small planar portion of the object's surface. Three behaviors are very important in rendering (Figure 1.9):

- specular reflection: the electromagnetic wave is reflected in a direction symmetric to the incident direction with respect to the normal,
- specular transmission: the electromagnetic wave is transmitted through the surface according to Snell-Descartes law,
- diffuse: the electromagnetic wave interacts many times inside a very small region of the object, making it eventually go out in a random direction, independent of the incident direction.

These three behaviors are geometric. The quantitative properties (the amount of energy which is still present after the interaction) are given by special solutions of the Maxwell equations for the considered case, for instance the Fresnel equations in the case of specular reflections and transmissions.

Switching to macroscopic models is done by considering in a statistical way the electromagnetic waves that compose the light beam (using spectral distributions), and the surface on which the light beam falls. For instance, we can consider that the surface hit by the light beam is composed of many infinitely small planar surfaces, described by *microfacets distributions*, which, for each normal orientation, give the density of microfacets effectively having this normal. Note that this does not invalidate the fact that the surface normal at point $\mathbf{x}$ can be considered to be unique, as this is the “macroscopic“ (average) normal, while the microfacets normals are mesoscopic ones. This normal is needed to be able to define irradiance (Section 1.2.1.2).

The microfacets density can then be used to weight the contribution of each mesoscopic normal to the final energy emitted in a given direction. Summing over all the normals yields the total amount of

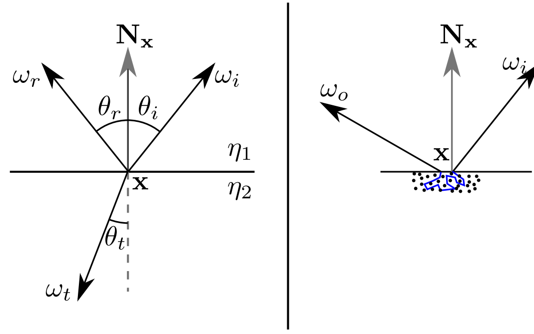


Figure 1.9: Illustration of the specular and diffuse behaviors of an electromagnetic wave, at the mesoscopic level. Consider a point $\mathbf{x}$ on a locally planar surface, with normal $\mathbf{N_x}$. The electromagnetic wave hits the surface at $\mathbf{x}$, with incident direction ω_i. Left: specular behavior. The electromagnetic wave is split into two parts: one is reflected (ω_r), the other is transmitted (ω_t). $\theta_r = \theta_i$, and θ_t is given by using the Snell-Descartes law, with the given indices of refraction η_1 and η_2. Right: diffuse behavior. The electromagnetic wave slightly penetrates inside the medium (the black disks) and is scattered multiple times, eventually leaving the medium at a nearby point, with a direction practically independent from ω_i. The exit point is so close from $\mathbf{x}$ that we consider it is at $\mathbf{x}$.

energy for each wavelength. A clear and complete illustration of this process can be found in [WMHT07] in the case of dielectric materials, to model light interaction properties of rough glasses.

These models lead to *bidirectional scattering distribution functions* (BSDF). For a more complete introduction to this vast subject and examples of such models, see chapters 9 and 10 of [PH04]. Note also that some models take into account the fact that even in dense objects, light can travel quite far (for instance for skin, marble, *etc.*). More complex models and equations have to be used, leading to the definition of *bidirectional sub-surface scattering distribution functions*. For an example of such models, see [JMLH01].

1.7.1.2 BSDFs And radiometry

In the radiometric context, BSDFs are used to transform spectral distributions of incident *irradiance* of individual light beams (given by $d_{\Omega_i} E_i$) to spectral distribution of outgoing spectral radiance, taking into account the attenuation due to scattering at the surface [1]

This is the main gateway between incident and outgoing quantities. To mathematically introduce BSDFs, outgoing radiance has to be further split, by considering that many incident light beams contribute to the radiance leaving along one direction ω_o, through scattering. As we only consider radiometric effects which are linear with respect to radiance, the final outgoing radiance is the sum of the contribution of each incident light beam. Therefore, one could compute the outgoing radiance associated to the light beams which belong to a solid-angle Ω_i. Instead of $L_o(\mathbf{x}, \omega_o)$, we should therefore write $L_o(\mathbf{x}, \omega_o, \Omega_i)$,

[1]If fluorescence was handled, some non-piecewise operations would occur between spectral distributions, which explains why we make BSDFs operate on spectral distributions instead of spectral quantities.

with:

$$L_o(\mathbf{x}, \omega_o, \Omega_i) = \int_{\Omega_i} d_{\Omega_i} L_o(\mathbf{x}, \omega_o, \omega_i). \tag{1.29}$$

As each radiance contribution $d_{\Omega_i} L_o(\mathbf{x}, \omega_o, \omega_i)$ is given by applying the BSDF $f_s(\mathbf{x}, \omega_i \leftrightarrow \omega_o)$ to a incident spectral irradiance distribution, we have:

$$\begin{aligned} d_{\Omega_i} L_o(\mathbf{x}, \omega_o, \omega_i) &= f_s(\mathbf{x}, \omega_i \leftrightarrow \omega_o)\,(d_{\Omega_i} E_i(\mathbf{x}, \omega_i)) & (1.30)\\ L_o(\mathbf{x}, \omega_o, \Omega_i) &= \int_{\Omega_i} f_s(\mathbf{x}, \omega_i \leftrightarrow \omega_o)\,(d_{\Omega_i} E_i(\mathbf{x}, \omega_i)). & (1.31) \end{aligned}$$

The linearity assumption of geometric optics and the fact that we do not deal with fluorescence implies that the value of the BSDF $f_s(\mathbf{x}, \omega_i \leftrightarrow \omega_o)$ is in fact a linear function, *i.e.* it can be considered as a piecewise weight called *reflectance*: for a spectral distribution D,

$$f_s(\mathbf{x}, \omega_i \leftrightarrow \omega_o)\,(D)(\lambda) = f_s(\mathbf{x}, \omega_i \leftrightarrow \omega_o)\,(\lambda) \times D(\lambda). \tag{1.32}$$

Applying this equality to Equation (1.30), we obtain the mathematical definition of a BSDF[2]:

$$\begin{aligned} f_s(\mathbf{x}, \omega_i \leftrightarrow \omega_o) &= \frac{d_{\Omega_i} L_o(\mathbf{x}, \omega_o, \omega_i)}{d_{\Omega_i} E_i(\mathbf{x}, \omega_i)} \\ &= \frac{d_{\Omega_i} L_o(\mathbf{x}, \omega_o, \omega_i)}{L_i(\mathbf{x}, \omega_i)\,|\mathbf{N_x} \cdot \omega_i|\, d\sigma(\omega_i)}. \end{aligned} \tag{1.33}$$

To be really strict, note that there is one BSDF defined at each point $\mathbf{x}$, so it is a 4D function (ω_i and ω_o are the only parameters). We call the complete $f_s(\mathbf{x}, \omega_i \leftrightarrow \omega_o)$ function, where each parameter can be changed, a *spatially-varying* BSDF.

Scattering equation: Putting all the pieces together, we obtain the fundamental equation of light interaction at a point $\mathbf{x}$, when no subsurface light transport is considered:

$$L_o(\mathbf{x}, \omega_o) = \int_{\mathcal{S}^2} f_s(\mathbf{x}, \omega_i \leftrightarrow \omega_o)\, L_i(\mathbf{x}, \omega_i)\, |\mathbf{N_x} \cdot \omega_i|\, d\sigma(\omega_i). \tag{1.34}$$

Physical plausibility: BSDFs have to meet certain requirements to be physically plausible.

First, they must be symmetric:

$$f_s(\mathbf{x}, \omega_i \leftrightarrow \omega_o) = f_s(\mathbf{x}, \omega_o \leftrightarrow \omega_i). \tag{1.35}$$

This symmetry justifies the $\leftrightarrow$ notation.

Then, they must not add energy to the system (they must be *energy-conserving*). This translates on a condition on power: for any surface S receiving a power $\Phi_i(S)$, the outgoing power $\Phi_o(S)$ re-emitted by S must be lower or equal to $\Phi_i(S)$. As this must be true for any surface S, it ends up that for any

[2] The division is done in a piecewise manner (Section 1.2.4)

point $\mathbf{x}$, $E_{\Lambda,o}(\mathbf{x}) \leq E_{\Lambda,i}(\mathbf{x})$. Further exploiting the fact that there is no interactions between different wavelengths, we have that, for any wavelength, $E_{\lambda,o}(\mathbf{x}) \leq E_{\lambda,i}(\mathbf{x})$. Putting all the pieces together and considering a constant incident spectral radiance distribution equal to 1 without loss of generality (all the equations being linear with respect to spectral radiance), we obtain that the BSDF must satisfy, for any point $\mathbf{x}$ and any wavelength λ:

$$\int_{\mathcal{S}^2}\int_{\mathcal{S}^2} f_s\left(\mathbf{x}, \omega_i \leftrightarrow \omega_o\right)(\lambda)\left|\mathbf{N}_{\mathbf{x}} \cdot \omega_i\right| d\sigma\left(\omega_i\right)\left|\mathbf{N}_{\mathbf{x}} \cdot \omega_o\right| d\sigma\left(\omega_o\right) \leq 2\pi. \tag{1.36}$$

Ensuring these conditions even when using practically-motivated assumptions is important to avoid very visible artifacts. For a more in-depth analysis, see Chapter 5 of Veach's thesis [Vea97].

1.7.2 Absorption and scattering in participating media

We now derive local equations to quantitatively describe the behavior of radiometric quantities at a single point $\mathbf{x}$. This section is just a brief introduction on the foundations of radiometry in participating media. More information linked to rendering can be found in chapter 12 of [PH04], and a complete treatment of this topic, linked to radiative transfer, can be found in [Cha53].

1.7.2.1 Absorption and scattering coefficients

As the photons of a light beam go through a medium, they can interact with particles of the medium. Some of them are absorbed, some other are scattered, therefore changing direction, while the rest stays in the beam. Therefore, as the beam travels through the medium, it gets attenuated. The quantitative aspects of these two causes of attenuation are modeled using per-distance probabilities (one per wavelength), called absorption and scattering coefficients, noted respectively $\sigma_a(\lambda)$ and $\sigma_s(\lambda)$. These coefficients are such that the number of photons of wavelength λ absorbed or scattered over a distance dt is equal to $N(\lambda)\sigma(\lambda)dt$, where σ is either σ_a or σ_s, and $N(\lambda)$ the number of photons with wavelength λ in the beam at the beginning of the segment of length dt. As the coefficients are per-distance probabilities, their unit is m^{-1}, and their value can be arbitrarily large: a medium where 100% of the photons inside a beam interacts within $1cm$ has a per-distance probability of $1/0.01m = 100$.

Assuming as before that interactions do not change the wavelength of a photon, and as spectral radiance is linear with the number of photons, we can write the difference of the radiance L of a beam caused by the interaction within the segment of infinitesimal length dt beginning at point $\mathbf{x}$:

$$\begin{aligned} L(\mathbf{x}+dt\omega,\omega) &= L(\mathbf{x},\omega) - L(\mathbf{x},\omega)\times\sigma\times dt \\ L(\mathbf{x}+dt\omega,\omega) - L(\mathbf{x},\omega) &= -L(\mathbf{x},\omega)\times\sigma\times dt \\ \frac{dL(\mathbf{x},\omega)}{dt} &= -L(\mathbf{x},\omega)\times\sigma \end{aligned} \tag{1.37}$$

where we have used the natural extension to spectral distributions of spectral radiance and operators acting on it (Section 1.2.4).

The absorption and scattering coefficients can vary both in positions and directions (imagine disk-shaped particles all aligned along a given direction, they will stop more photons coming along that direction than along others). Therefore, they are described by functions $\sigma_a(\mathbf{x}, \omega)$ and $\sigma_s(\mathbf{x}, \omega)$, and Equation (1.37) becomes:

$$\frac{dL(\mathbf{x}, \omega)}{dt} = -L(\mathbf{x}, \omega) \times \sigma(\mathbf{x}, \omega). \tag{1.38}$$

As the light beam is attenuated both by absorption and scattering, and assuming these interactions are independent, the total attenuation is the sum of the absorption and scattering contributions. Letting $\sigma_t(\mathbf{x}, \omega) = \sigma_a(\mathbf{x}, \omega) + \sigma_s(\mathbf{x}, \omega)$, we get the differential equation describing light attenuation of a light beam inside a participating medium:

$$\frac{dL(\mathbf{x}, \omega)}{dt} = -L(\mathbf{x}, \omega) \times \sigma_t(\mathbf{x}, \omega). \tag{1.39}$$

σ_t is called the *attenuation* or *extinction* coefficient.

Homogeneity, medium isotropy: When the scattering and absorption coefficients are constant throughout the medium, the medium is said to be *homogeneous*, otherwise it is *heterogeneous*. When the coefficients do not vary with directions, it is said to be *isotropic*, otherwise it is *anisotropic*.

1.7.2.2 Phase Functions

When a radiance beam coming from incident direction ω_i is scattered by a particle at point $\mathbf{x}$ of the medium, the outgoing radiance distribution is described by the *phase function* at point $\mathbf{x}$, $\rho(\mathbf{x}, \omega_i \leftrightarrow \omega_o)$, where ω_o is an outgoing direction. It can be seen as a sort of BSDF for continuous media. As BSDFs, they have to be symmetric, *i.e.* $\rho(\mathbf{x}, \omega_i \leftrightarrow \omega_o) = \rho(\mathbf{x}, \omega_o \leftrightarrow \omega_i)$. However, it has two important differences with BSDFs: no cosine term is considered here, as there is no projection on a macroscopic surface, and it does not have to take into account absorption, as it is already done by the absorption coefficient.

The first difference means that the radiometric definition of ρ at point $\mathbf{x}$ is

$$\rho(\mathbf{x}, \omega_i \leftrightarrow \omega_o) = \frac{d_{\Omega_i} L_o(\mathbf{x}, \omega_o, \omega_i)}{L_i(\mathbf{x}, \omega_i) d\sigma\left(\omega_i\right)}. \tag{1.40}$$

Note that this expression is similar to the one of the BSDF (Equation (1.33)) when considering a surface perpendicular to the incident beam's direction ω_i.

Solid-angles in participating media: the expression of solid-angles in spherical coordinates requires the measurement of two angles with respect to a frame with a "normal" vector. Here, the normal vector is chosen so that it is aligned with ω_i. Therefore $d\sigma\left(\omega_i\right) = d\theta d\phi$ is constant for any direction ω_i.

The second difference means that the sum of the radiance of all outgoing beams is equal to the radiance of the incident beam, or equivalently:

$$\int_{\mathcal{S}^2} \rho(\mathbf{x}, \omega_i \leftrightarrow \omega_o) d\sigma\left(\omega_o\right) = 1 \tag{1.41}$$

for all incident directions ω_i.

Mathematically, this is equivalent to saying that when both $\mathbf{x}$ and one of the direction are fixed, ρ is a probability density function expressed with respect to the solid-angle measure. As the phase function is symmetric with respect to the directions, it does not matter which one is fixed.

If the phase function is constant with respect to both ω_i and ω_o, it is said to be isotropic. As this term can be ambiguous with the medium isotropy, we will always explicitly say that we use an isotropic phase function when it is the case. Note that due to the normalization constraint, an isotropic phase function at point $\mathbf{x}$ is given by:

$$\rho(\mathbf{x}, \omega_i \leftrightarrow \omega_o) = \frac{1}{4\pi}. \tag{1.42}$$

Medium anisotropy and phase function anisotropy: Scattering coefficients and phase functions do not operate at the same level: scattering coefficients model the interaction of a radiance beam with a bunch of particles considered as a continuous medium (using statistical models), while a phase function describes the interaction of a radiance beam with a single particle. Therefore, a medium can be isotropic while the phase function is not, and *vice versa*.

1.7.2.3 Self-Emission, In-Scattering, Out-Scattering

We now focus on the change in radiance of a beam arriving at $\mathbf{x}$ with direction ω_o, *i.e.* we want to compute the radiance of the beam at a point infinitesimally further away along the beam's trajectory, located at $\mathbf{x}' = \mathbf{x} + dt \times \omega_o$. This means computing $L_o(\mathbf{x}', \omega_o)$.

Figure 1.10 shows the setup we use in this section, and illustrates the mechanisms described here.

Self-emission: The first cause of change is self-emission, which is modeled using a per-unit-distance radiance function $L_{ve}(\mathbf{x}, \omega_o)$. Therefore, the change in radiance due to self-emission is given by:

$$\Delta_e(\mathbf{x}, \omega_o) = L_{ve}(\mathbf{x}, \omega_o) \times dt. \tag{1.43}$$

Scattering: Scattering in participating media has two antagonist effects on a radiance beam $L_o(\mathbf{x}, \omega_o)$ at any given point $\mathbf{x}$: some energy arriving at $\mathbf{x}$ from other directions is scattered back along ω_o – *in*-scattering –, and some energy of the beam is scattered in other directions – *out*-scattering. This leads to a change in radiance between $L_o(\mathbf{x}, \omega_o)$ and $L_o(\mathbf{x}', \omega_o)$.

In-scattering: The amount of radiance from an incident radiance beam $L_i(\mathbf{x}, \omega_i)$ which is scattered ”at“ point $\mathbf{x}$ (more precisely, along a segment of infinitesimal length dt) is given by $\sigma_s(\mathbf{x}, -\omega_i) \times dt \times L_i(\mathbf{x}, \omega_i)$ (as ω_i is an incident direction, it leaves from $\mathbf{x}$, hence the minus sign in σ_s). Here, $\sigma_s(\mathbf{x}, -\omega_i) \times dt$ is the probability that a photon is scattered in the interval of length dt. This probability can also be

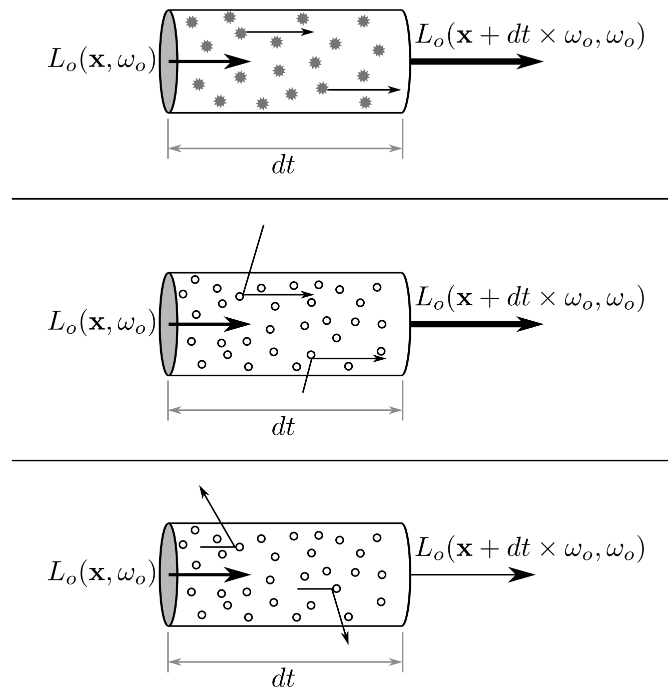


Figure 1.10: Top: Self-emission. Radiance is added to a beam by particles which emit along the same direction. Middle: In-scattering. Light interacts with the particles of the medium, and can be scattered in the beam direction, adding radiance to the beam. Bottom: out-scattering. Symmetrically to in-scattering, some radiance of the beam can be scattered to other directions, leading to a decrease in the beam's radiance.

interpreted as a proportion. The amount of radiance which is scattered from ω_i toward ω is therefore:

$$d_{\Omega_i} L_o(\mathbf{x}, \omega_o, \omega_i) = \sigma_s(\mathbf{x}, -\omega_i) \times dt \times L_i(\mathbf{x}, \omega_i) \times d\sigma\,(\omega_i) \times \rho(\mathbf{x}, \omega_i \leftrightarrow \omega_o). \tag{1.44}$$

Summing all the contributions from each infinitesimal solid-angle yields the total in-scattering contribution:

$$\begin{aligned} \Delta_i(\mathbf{x}, \omega_o) &= \int_{\mathcal{S}^2} d_{\Omega_i} L_o(\mathbf{x}, \omega_o, \omega_i) \\ &= \int_{\mathcal{S}^2} \sigma_s(\mathbf{x}, -\omega_i) dt L_i(\mathbf{x}, \omega_i) d\sigma\,(\omega_i)\, \rho(\mathbf{x}, \omega_i \leftrightarrow \omega_o). \end{aligned} \tag{1.45}$$

Out-scattering: According to the definition of the extinction coefficient, the amount of radiance which is lost due to scattering in other directions or absorption is given by:

$$\Delta_o(\mathbf{x}, \omega_o) = \sigma_t(\mathbf{x}, \omega_o) \times dt \times L_o(\mathbf{x}, \omega_o). \tag{1.46}$$

Note that the scattering from $-\omega_o$ to ω_o (*i.e.* , energy forwarding through scattering) is taken into account by in-scattering.

Two of the above terms add energy (Δ_e and Δ_i), while Δ_o removes some. Putting it all together, $L_o(\mathbf{x}', \omega_o)$ is given by:

$$L_o(\mathbf{x}', \omega_o) = L_o(\mathbf{x}, \omega_o) + \Delta_e(\mathbf{x}, \omega_o) + \Delta_i(\mathbf{x}, \omega_o) - \Delta_o(\mathbf{x}, \omega_o). \tag{1.47}$$

Putting Δ_e and Δ_i in a single term, we can express the radiance added to the beam as a function of a *source term* noted L_s, which gives the per-unit-distance density of added radiance at point $\mathbf{x}$ along direction ω_o:

$$\Delta_e + \Delta_i = dt \times L_{ve}(\mathbf{x}, \omega_o) + dt \times \int_{\mathcal{S}^2} \sigma_s(\mathbf{x}, -\omega_i) L_i(\mathbf{x}, \omega_i) \rho(\mathbf{x}, \omega_i \leftrightarrow \omega_o) d\sigma\,(\omega_i) \tag{1.48}$$

$$= dt \times \left(\underbrace{L_{ve}(\mathbf{x}, \omega_o) + \int_{\mathcal{S}^2} \sigma_s(\mathbf{x}, -\omega_i) L_i(\mathbf{x}, \omega_i) \rho(\mathbf{x}, \omega_i \leftrightarrow \omega_o) d\sigma\,(\omega_i)}_{L_s(\mathbf{x}, \omega_o)} \right). \tag{1.49}$$

This yields:

$$L_o(\mathbf{x}', \omega_o) = L_o(\mathbf{x}, \omega_o) + dt \times L_s(\mathbf{x}, \omega_o) - \Delta_o(\mathbf{x}, \omega_o) \tag{1.50}$$

$$= L_o(\mathbf{x}, \omega_o) + dt \times L_s(\mathbf{x}, \omega_o) - dt \times \sigma_t(\mathbf{x}, \omega_o) \times L_o(\mathbf{x}, \omega_o) \tag{1.51}$$

$$\frac{L_o(\mathbf{x}', \omega_o) - L_o(\mathbf{x}, \omega_o)}{dt} = L_s(\mathbf{x}, \omega_o) - \sigma_t(\mathbf{x}, \omega_o) \times L_o(\mathbf{x}, \omega_o). \tag{1.52}$$

Expanding $\mathbf{x}'$, the left-most term is the definition of $\frac{dL_o(\mathbf{x}, \omega_o)}{dt}$, therefore radiance change can be

expressed in terms of an integro-differential equation:

$$\frac{dL_o(\mathbf{x}, \omega_o)}{dt} = L_s(\mathbf{x}, \omega_o) - \sigma_t(\mathbf{x}, \omega_o) \times L_o(\mathbf{x}, \omega_o). \tag{1.53}$$

1.8 Light transport equations

The purpose of light transport equations is to be able to compute the effect of the three basic interactions described above (emission, absorption, scattering) when light travels in a scene, based on what happens at each point, which is described by the local equations derived above.

The formal goal of this section is to derive equations which allow us to compute the incident radiance $L_i(\mathbf{x}, \omega_i)$ at any point $\mathbf{x}$ from any incident direction ω_i. These equations assume that light is incoherent (each electromagnetic wave has a random polarization and a random phase), and that it travels in straight lines in absence of interaction. A consequence is that no non-zero magnetic field should be present, and that media with continuously varying refraction indices are not handled.

Ray-tracing function: As we assume that light travels in straight lines when no interactions are present, rays are particularly well suited to describe the geometrical aspects of light transport. Let $tr(\mathbf{x}, \omega)$ be the function which gives the nearest surface point seen from $\mathbf{x}$ when looking in direction ω. When no such point is present (no surface crosses the ray), $tr(\mathbf{x}, \omega)$ is a point at infinity. tr is called a *ray-tracing* function.

1.8.1 Light transport in empty space

As light travels in straight line and no interaction occurs, radiance is conserved locally at each point not on a surface (Equation (1.18)). Therefore, the incident radiance $L_i(\mathbf{x}, \omega_i)$ at a point $\mathbf{x}$ from direction ω_i is equal to the radiance leaving from the nearest point on a surface along direction $-\omega_i$, *i.e.*

$$L_i(\mathbf{x}, \omega_i) = L_o(tr(\mathbf{x}, \omega_i), -\omega_i). \tag{1.54}$$

Letting $\mathbf{y} = tr(\mathbf{x}, \omega_i)$, $L_o(\mathbf{y}, -\omega_i)$ is given by adding the contributions from the emission part at $\mathbf{y}$, $L_e(\mathbf{y}, -\omega_i)$, and the scattering part. This last one is given by Equation (1.34), derived in Section 1.7.1. This leads to a monument of rendering, the *light transport equation* (LTE) or rendering equation [Kaj86]:

$$L_o(\mathbf{y}, \omega_o) = L_e(\mathbf{y}, \omega_o) + \int_{\mathcal{S}^2} f_s(\mathbf{y}, \omega_i \leftrightarrow \omega_o) L_i(\mathbf{y}, \omega_i) |\mathbf{N_y} \cdot \omega_i| \, d\sigma(\omega_i), \tag{1.55}$$

Finally, we get:

$$L_i(\mathbf{x}, \omega_i) = L_e(\mathbf{y}, -\omega_i) + \int_{\mathcal{S}^2} f_s\left(\mathbf{y}, \omega_i' \leftrightarrow -\omega_i\right) L_i\left(\mathbf{y}, \omega_i'\right) \left|\mathbf{N_y} \cdot \omega_i'\right| d\sigma\left(\omega_i'\right). \tag{1.56}$$

Note that Equation (1.56) is recursive, as L_i depends on L_i. Convergence to a finite value is ensured only when all the BSDFs conserve energy.

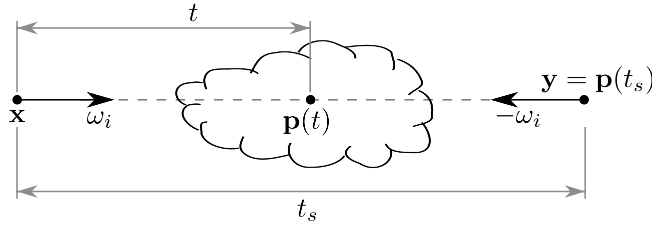


Figure 1.11: Geometrical terms used in the radiative transfer equation.

1.8.2 Light transport with participating media

Similarly to the light transport equation derivation, we set $\mathbf{y} = tr(\mathbf{x}, \omega_i)$. With this convention, the incident radiance $L_i(\mathbf{x}, \omega_i)$ can be computed by taking into account all the interactions that occur on the ray going from $\mathbf{x}$ to $\mathbf{y}$. From a radiance propagation point of view (which is the one taken for the local equations derivation), we instead consider all the interactions that occur on the ray going from $\mathbf{y}$ to $\mathbf{x}$.

Computing $L_i(\mathbf{x}, \omega_i)$ therefore consists in solving Equation (1.53) in the interval from $\mathbf{y}$ to $\mathbf{x}$, the boundary condition being that the outgoing radiance at $\mathbf{y}$ is either given by Equation (1.55) if $\mathbf{y}$ is on a surface, or the emission of a distant light if it is at infinity.

The resolution of Equation (1.53) is rather technical and does not bring any insight on light transport, so it is not reproduced here but it can be found in books focused on radiative transfer. However, the solution of Equation (1.53) is the key to simulating light transport in participating media, and is called the *radiative transfer equation* (RTE):

$$L_i(\mathbf{x}, \omega_i) = L_o(\mathbf{y}, -\omega_i)\, Tr(\mathbf{y} \to \mathbf{x}) + \int_0^{t_s} Tr(\mathbf{p}(t) \to \mathbf{x}) L_s(\mathbf{p}(t), -\omega_i) dt \tag{1.57}$$

where $\mathbf{p}(t) = \mathbf{x} + t\omega_i$, t_s is the distance between $\mathbf{x}$ and $\mathbf{y}$ ($t_s = \infty$ is $\mathbf{y}$ is at infinity), and $Tr(\mathbf{p_1} \to \mathbf{p_2})$ is a function detailed below. See Figure 1.11 for the geometrical terms. Similarly to the LTE, this function is recursive. Note that the bounds of integration over the t coordinate are in general tightened to the interval for which extinction is potentially non-zero.

If a surface is hit ($t_s < \infty$), $L_o(\mathbf{y}, -\omega_i)$ is given by the light transport equation (Equation (1.55)). If no surface is hit when leaving from $\mathbf{y}$ along direction ω_i (*i.e.* $t_s = \infty$), then $L_o(\mathbf{y}, -\omega_i)$ is either 0 if no distant light such as a sky model is present, or the radiance emitted by the light otherwise.

Transmittance and optical thickness: $Tr(\mathbf{p_1} \to \mathbf{p_2})$ is called the *transmittance*. It corresponds to the total attenuation along a non-infinitesimal segment of length d going from $\mathbf{p_1}$ to $\mathbf{p_2}$. Its expression is the solution of the differential equation given by Equation (1.39) on the segment going from $\mathbf{p_1}$ to $\mathbf{p_2}$, with $Tr(\mathbf{p_1} \to \mathbf{p_1}) = 1$ as boundary condition (no attenuation on a zero-length segment):

$$Tr(\mathbf{p_1} \to \mathbf{p_2}) = \exp\left\{-\int_0^{d} \sigma_t(\mathbf{p_2} - t\omega, \omega) dt\right\} \tag{1.58}$$

where ω is the normalized direction from $\mathbf{p_1}$ to $\mathbf{p_2}$. Note that transmittance is not symmetric for

anisotropic media. The inner integral is called *optical thickness*, and noted $\tau(\mathbf{p_1} \to \mathbf{p_2})$:

$$\tau(\mathbf{p_1} \to \mathbf{p_2}) = \int_0^d \sigma_t(\mathbf{p_2} - t\omega, \omega)dt. \tag{1.59}$$

The transmittance is such that given a radiance beam leaving from $\mathbf{y}$ along the direction ω, the radiance from this beam arriving at $\mathbf{x}$ is given by $L_o(\mathbf{y}, \omega)Tr(\mathbf{y} \to \mathbf{x})$.

Radiative transfer equation interpretation: Equation (1.57) just says that the incident radiance at $\mathbf{x}$ from incident direction ω_i is the sum of two contributions: the radiance leaving from $\mathbf{y}$, and the contribution of each intermediate point $\mathbf{p}(t)$, which adds radiance through self-emission and in-scattering. Each contribution is attenuated because of absorption and out-scattering at each point of the segment going from the origin of the contribution ($\mathbf{p}(t)$) to $\mathbf{x}$, this attenuation being given by the transmittance.

LTE as a special case of RTE: Note that in absence of participating medium, both the scattering and absorption coefficients can be considered as being equal to zero everywhere, as well as the self-emission density L_{ve}. In this case, Equation (1.57) is equivalent to Equation (1.56).

1.9 Summary and final equations

1.9.1 Hypotheses

It is crucial to remember that the equations derived in this chapter assume that the following hypotheses are verified:

- Polarization, interference, and other phenomenons relying on a precise wave description are not taken into account. In particular, light is considered incoherent, therefore lasers can not be accurately handled.
- Materials or mediums do not cause cross-wavelength and temporal interactions, such as phosphorescence or fluorescence.
- Light beams follow straight lines between interactions. More particularly, mediums have a constant refraction index, and no magnetic field should be present. For instance, refraction due to hot air can not be directly handled.
- Thermodynamical equilibrium is assumed: the energy that leaves a point at time t only depends on energy that arrived or was emitted during an infinitely small interval of time of size dt before t. More intuitively, outgoing radiance at t only depends on incident radiance at t.
- No sub-subsurface transport occurs inside objects.
- The sensor is planar, its response to a radiance beam does not change over time, a contribution hitting the sensor at coordinates (x, y) influences only the signal value at (x, y), and no inter-wavelength interactions occurs.

1.9.2 Spectral rendering

At this point, all the equations required to compute an image have been derived, using spectral distributions to describe light (hence the name of *spectral* rendering). We sum them up here as a "quick-check" reference.

Pixel value (Section 1.5):

$$I_p = \int_{s(h_P)} h_P(\Delta_x, \Delta_y) S(x_P + \Delta_x, y_P + \Delta_y) d\mathcal{A}(\Delta_x, \Delta_y). \tag{1.60}$$

I_p is a color, with unbounded values. The image obtained by gathering all the I_p values is a high-dynamic range image.

Signal value (Section 1.5.2):

These equations makes the link between colorimetry (Section 1.3) and radiometry (Section 1.2). There is one equation per color component C, with $\bar{c}$ being the color-matching function:

$$S_C(x, y) = \int_T \int_{\mathcal{S}^2} \int_\Lambda \bar{c}(R(\omega_i) dE_i(x, y, \omega_i, t)) d\lambda dt. \tag{1.61}$$

Physics of light (Section 1.7 and Section 1.8): From the point $\mathbf{x}$ in space corresponding to coordinates (x, y) on the sensor, $dE_{\lambda,i}(x, y, \omega_i, t) = dE_i(\mathbf{x}, \omega_i, t)(\lambda)$ (Section 1.2.4). The time parameter t is considered as an implicit parameter from now on.

Let $\mathbf{p}(t') = \mathbf{x} + t'\omega_i$ and $\mathbf{y} = tr(\mathbf{x}, \omega_i)$ (Section 1.8). Note that here t' is not a time-related coordinate, but an abscissa along a ray. Then we have:

$$dE_i(\mathbf{x}, \omega_i) = L_i(\mathbf{x}, \omega_i) d\sigma(\omega_i) |\mathbf{N}_\mathbf{x} \cdot \omega_i|, \tag{1.62}$$

$$L_i(\mathbf{x}, \omega_i) = L_o(\mathbf{y}, -\omega_i) Tr(\mathbf{y} \to \mathbf{x}) + \int_0^{t_s} Tr(\mathbf{p}(t') \to \mathbf{x}) L_s(\mathbf{p}(t'), -\omega_i) dt', \tag{1.63}$$

$$L_o(\mathbf{y}, \omega_o) = L_e(\mathbf{y}, \omega_o) + \int_{\mathcal{S}^2} f_s\left(\mathbf{y}, \omega_i' \leftrightarrow \omega_o\right) L_i\left(\mathbf{y}, \omega_i'\right) \left|\mathbf{N}_\mathbf{y} \cdot \omega_i'\right| d\sigma\left(\omega_i'\right). \tag{1.64}$$

1.9.3 Color-space rendering

Spectral rendering, summarized in Section 1.9.2, seems simple, but a major problem, not related to computation time, arises when one wants to use it: the specification of spectral data. The spectral data that are needed to compute the value of light transport equations are:

- Light emission models ($L_e(\mathbf{x}, \omega)$), and per-unit-distance self-emission models ($L_{ve}(\mathbf{x}, \omega)$).
- BSDFs ($f_s(\mathbf{x}, \omega_i \leftrightarrow \omega_o)$) and phase functions ($\rho(\mathbf{x}, \omega_i \leftrightarrow \omega_o)$).
- Absorption ($\sigma_a(\mathbf{x}, \omega)$) and scattering ($\sigma_s(\mathbf{x}, \omega)$) coefficients.

These data can be available in some specific cases, when rendering is more a targeted simulation tool and measures or models are available, than a digital image synthesis tool. In the latter case, rendering is used by artists, which have to specify all these data. Most of data can be analytically described by models, which can be parametrized by a few spectral distributions. For instance, for a simple car paint model, the artist would have to supply the spectral distributions of the diffuse layer and the reflexive layer, and the per-wavelength index of refraction of the paint layer. The BSDF model would then use these data based on Fresnel's law.

However, this process is long, requires measurements or a lot of trial-and-errors, and is not intuitive for artists. It is far easier for artists to specify colors: the color of the diffuse layer, the color of the paint layer, and a single parameter which controls the blending between these two layers. Experimentations are easy, and learning by trial-and-error rapidly yields satisfactory results.

It is not possible to mix spectral rendering and color-space rendering: switching back and forth between colors and spectra is not possible. As a matter of fact, the spectrum $\rightarrow$ color-space transformation is non-invertible: several spectra can lead to the same color. Therefore, if all the input are given in colors, it is not possible to just convert them to spectra without "wild-guessing", and then do spectral rendering. This ambiguous conversion is known as *metamerism*.

A common way to deal with that problem is to simply replace spectra by colors everywhere, expressed in a linear color space such as CIE-RGB. This is called *color-space rendering*. As the color-space is linear, all operations can be done component-wise as before with wavelengths. Mathematically, this is equivalent to "forwarding" the convolution with color-matching functions at the inner-most levels of the equations, instead of doing it in the end, when computing the signal value.

This introduces errors in many places as shown in [Bor91] for surface illumination, but in practice these errors are highly acceptable when exact simulation is not a target, and many commercial renderers aimed at artists are color-space renderers. We list some of these errors here:

- The component-wise multiplication of f_s with L_i in the LTE is not exact:

$$\int_\Lambda (\bar{c}(\lambda) \times f_s \times L_i) d\lambda \neq \left(\int_\Lambda (\bar{c}(\lambda) \times f_s) d\lambda \right) \times L_i^C \tag{1.65}$$

  where C is an arbitrary color component, $\bar{c}$ is its color-matching function and L_i^C is the C component of the color-space incident radiance. The same thing holds for the in-scattering term in radiative transfer (Equation (1.45)), as the phase function has a similar role as the BSDF.

- The color-space transmittance Tr_C is wrong:

$$\exp\left\{ - \int \left(\int_\Lambda \bar{c}(\lambda) \sigma_t(\lambda) d\lambda \right) dt \right\} \neq \int_\Lambda \hat{c}(\lambda) \exp\left\{ - \int \sigma_t(\lambda) dt \right\} d\lambda \tag{1.66}$$

- As linear color spaces are not complete color space, colors that can not be represented are not correctly handled, leading to further errors.

Moreover, color-space rendering prevents from taking into account wavelength-dependent behavior, such as refraction of white light by a prism that yields a rainbow.

As the color $\rightarrow$ spectrum conversion is ambiguous, it is hard to switch back and forth between spectral rendering and color-space rendering in a coherent way. Therefore, as artistic control is more important than absolute physical accuracy in digital image synthesis, color-space rendering is a method of choice in this domain.

As we aim at digital image synthesis, the rendering engine we have developed is a color-space renderer. However, it is not tightly tied to color-space rendering, and switching it to a spectral renderer would not be hard. Moreover, the methods presented in this document are not specific to color-space rendering.

1.9.4 Final equations

Similarly to Section 1.9.2, we now sum up the equations used in color-space rendering, and which will be used throughout the remaining of this document. For simplicity, we will add a few hypotheses.

Additionally to the assumptions summarized in Section 1.9.1, these equations further assume that data are provided in a linear color space (for instance CIE-RGB), the same for all data. Now, instead of being spectral distributions, all the quantities such as reflectances, incident radiances, emissions, *etc.*, are represented by a color, with component-wise operators similar to what has been used for spectral distributions. This means that, for instance, $L_e(\mathbf{x}, \omega)(R)$ is the red component of the color $L_e(\mathbf{x}, \omega)$ (sort of color-space radiance), and $(f_s(\mathbf{x}, \omega_i \leftrightarrow \omega_o) L_i(\mathbf{x}, \omega_i))(R)$ is equal to $f_s(\mathbf{x}, \omega_i \leftrightarrow \omega_o)(R) \times L_i(\mathbf{x}, \omega_i)(R)$.

Pixel value:

$$I_p = \int_{s(h_P)} h_P(\Delta_x, \Delta_y) S(x_P + \Delta_x, y_P + \Delta_y) d\mathcal{A}(\Delta_x, \Delta_y). \tag{1.67}$$

I_p is a color, with unbounded values. Its color space (called *storage color-space* from now on) can be the one used during computations (*computational color-space*, for instance CIE-RGB), or any other one. To illustrate, we will assume that the storage color-space is CIE-XYZ in the remaining of this section.

Signal value (Section 1.5.2): Additionally to sensor response modeling, this equation is now responsible for converting from the computational color-space (CIE-RGB) to the storage one (CIE-XYZ). Let C be the function which performs this conversion. All components can be computed at once:

$$S(x, y) = C\left(\int_T \int_{\mathcal{S}^2} R(\omega_i) dE_i(x, y, \omega_i, t) dt\right). \tag{1.68}$$

Physics of light (Section 1.7 and Section 1.8): Taking the same notations than for spectral rendering (Section 1.9.2), no apparent differences appear in the equations compared to spectral rendering. The big

implicit difference is that all quantities are colors.

$$dE_i(\mathbf{x},\omega_i) = L_i(\mathbf{x},\omega_i)\, d\sigma(\omega_i)\, |\mathbf{N_x}\cdot\omega_i|, \tag{1.69}$$

$$L_i(\mathbf{x},\omega_i) = L_o(\mathbf{y},-\omega_i)\, Tr(\mathbf{y}\to\mathbf{x}) + \int_0^{t_s} Tr(\mathbf{p}(t')\to\mathbf{x}) L_s(\mathbf{p}(t'),-\omega_i)dt', \tag{1.70}$$

$$L_o(\mathbf{y},\omega_o) = L_e(\mathbf{y},\omega_o) + \int_{\mathcal{S}^2} f_s\left(\mathbf{y},\omega_i'\leftrightarrow\omega_o\right) L_i\left(\mathbf{y},\omega_i'\right) \left|\mathbf{N_y}\cdot\omega_i'\right| d\sigma\left(\omega_i'\right). \tag{1.71}$$

1.10 Conclusion

Equations in Section 1.9.4 are the physical foundations of rendering, and the goal of a renderer is to solve them for a given scene. This scene is represented by different entities, which must be compatible with the requirements of physically-based rendering. For instance, the ray-tracing function, required for evaluating the light transport equations, must be implementable, or the camera must be modeled realistically. The next chapter is devoted to presenting the different abstract entities which are present in a scene, as well as examples of actual representations of these entities.

2

Description of a scene for physically-based rendering

Entities in a scene can be clustered in some general categories, and we briefly present ways to describe each entity.

- **Cameras**, to define the viewpoint, the type of camera, the type of sensor, *etc.*. This is where the response function from Section 1.5.2 is defined.
- **Light sources**, to define the L_e terms of the light transport equations (Section 1.8).
- **Geometry**, to define solid objects or the boundary of liquids. Geometries have to define ray intersection methods to build the ray-tracing function required in Section 1.8.
- **Materials**, to define the appearance of geometry by associating a BSDF (Section 1.7.1.1) to each point of the surface.
- **Participating media**, to get the absorption, scattering and self-emission coefficients at each point of a volume (Section 1.7.2).

2.1 Camera model

As the camera is an optical system which largely impacts the final image, it is necessary to describe it accurately. A general model is based on actual camera and lens systems. As shown below in Section 2.1.3, this model leads to a change of the equations to be evaluated during rendering. This generality and change of equations explains why we describe it in a precise way in this document, in contrast with the other entities.

Real-world cameras use complex cylinder-shaped lens systems which change the trajectory of light to focus it on sensors. They are commonly composed of chains of convergent or divergent lenses. An example of a (simple) lens system is shown in Figure 2.1. A lens system has two independent effects.

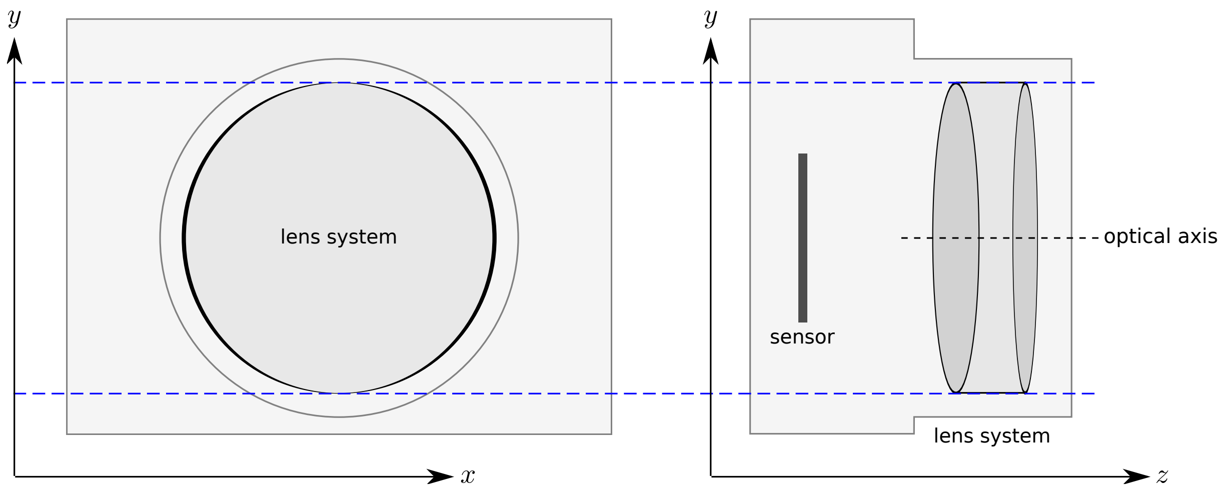


Figure 2.1: Side views of a camera, with a simple lens system composed of three chained lenses (two convergent and one divergent), and the sensor placed behind.

First, it has a *geometrical* effect: it changes the trajectory of light (radiance) beams. Second, it has a *radiometric* effect: as the light beam goes through several reflection/refraction processes, it gets attenuated. A camera model must represent these two aspects.

2.1.1 Optical model of lens systems

As lens systems have two sides, we take as convention that the side which faces the scene is the *front side*, and the other one the *rear side*. According to geometric optics, these lens systems can be described using a few properties, instead of considering all the lenses of the actual system. Here, we consider that we have perfect lens systems, without optical aberrations due to the fact that a real-world lens is not perfectly radially symmetric or perfectly uniform. With these hypotheses, some helpful properties can be obtained.

Stigma: One of the key property of a lens system for simulation is that astigmatism is not present: energy arriving from a point in the scene (which we call *object point*), is focused by the lens onto a single point, which we call *image point*. Such a pair of points is called a *stigmatic pair*. Stigmatic pairs are strongly related to in-focus points, as if the image point of a stigmatic pair is on the sensor, then the object point is in focus on the final image.

Nodal points: Compound lenses have a front and a rear nodal point, noted respectively N_f and N_r. As illustrated in Figure 2.2, a ray targeting one of this nodal point is refracted in the same direction as the incident ray, but as if leaving from the other nodal point. Note that for single symmetric lenses, these two points are at the same location. Each of N_f and N_r define a plane (noted P_f and P_r respectively), perpendicular to the optical axis and which contains the associated nodal point. P_f and P_r are called front and rear principal planes respectively. Principal planes have the property that, even though light

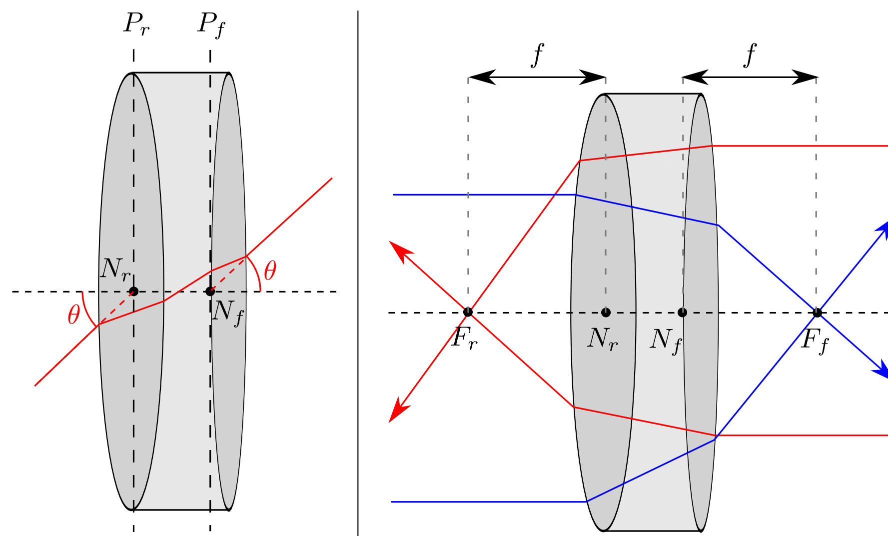


Figure 2.2: Left: Nodal points of a system of three chained lenses. Light rays targeting one of this nodal point goes out as if it was translated to the other nodal point. Right: focal points of the same lens system. All light rays which are parallel to the optical axis are focused on these points.

rays traversing the lens are refracted at the lens surface, it is equivalent to consider that they are refracted at the principal planes instead.

Focal points: Lens systems also have the property to focus light rays which are parallel to the optical axis on two singular points located on the optical axis. These points, noted F_f and F_r, are called respectively front and rear *focal points*. An example is shown in Figure 2.2. The distance between F_f and N_f (resp. F_r and N_r) is called front (resp. rear) *focal length*. For most lens systems, these lengths are equal. In the remaining we consider that they are equal, and we note both f. Similarly to nodal points and principal planes, each focal point has an associated plane, called focal plane, which contains the focal point and is perpendicular to the optical axis. This plane has an angle-related geometric property: for a given angle θ, all light rays entering the lens with an angle θ relatively to the optical axis end up traversing the same point of the rear focal plane. Moreover, the larger the angle, the further away from the focal point the point on the focal plane is. All the geometric properties presented above are summarized as geometric rules of lens systems by Figure 2.3.

Geometric rules: As shown in Figure 2.4 and Figure 2.5, the two rules of Figure 2.3 are sufficient to find the image point (if it exists) of any object point, as well as the direction and origin of any light ray at the exit of any lens system. The first application makes it possible to find stigmatic pairs, while the second is used to find the direction of a ray leaving from the camera and going into the scene, given a position on the sensor and a position on the front principal plane. Symmetrically, it can also be used to find the point on the sensor which is touched by a given incident light ray.

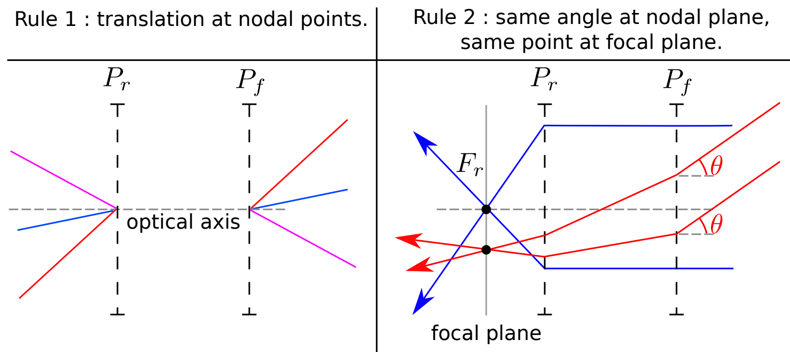


Figure 2.3: Base geometric rules for lens systems. Left: Rays which enter a lens system targeting a nodal point leave from the lens system as if translated to the other nodal point. Right: All rays entering the lens system with the same angle at a nodal plane go through the same point at the focal plane of the opposite side of the lens.

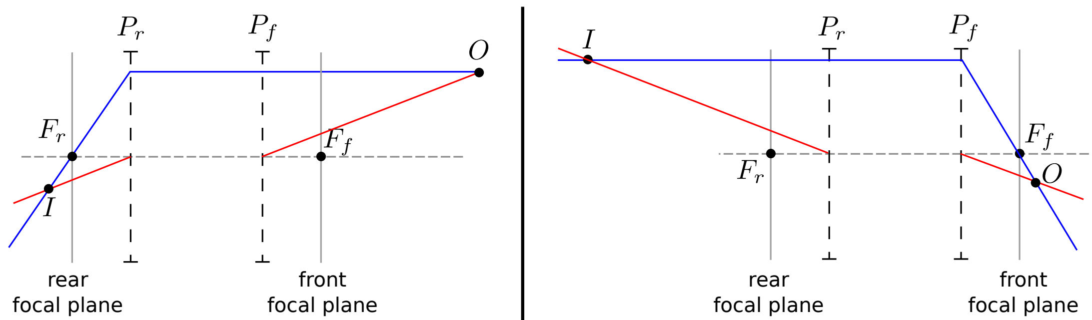


Figure 2.4: It is enough to find two trajectories to find the image point I from the object point O, as their crossing point is I. The first trajectory (red) is obtained by using the first rule of Figure 2.3, the second trajectory (blue) is obtained by using the second rule of Figure 2.3, either for the rear focal point (left) or the front focal point (right).

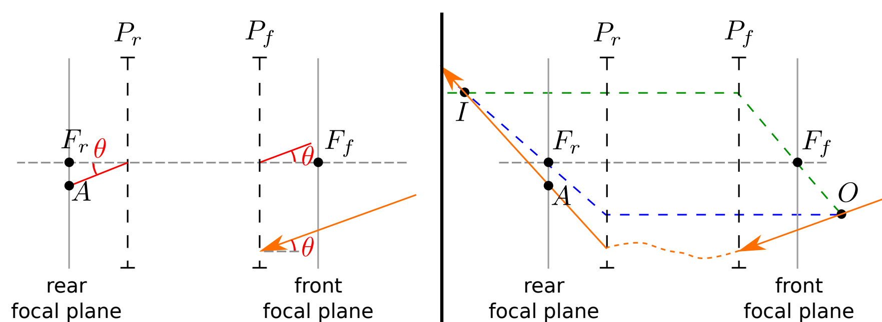


Figure 2.5: The two geometric rules of Figure 2.3 are used to find a light beam trajectory. More precisely, the trajectory inside the lens system is not found, but we find the light ray leaving from the lens (exit ray) associated to the ray which enters it (in orange, entry ray). The exit ray is found by computing the position of two of its points, for the given entry ray. Left: First, the angular property of focal planes and the properties of nodal points are used to find the point A of the focal plane associated to the entry ray's angle with the optical axis. In fact, the light ray parallel to the original ray which targets the front nodal point (in red) is only translated, so A is easily found. By the definition of the focal plane (second rule), we know that A belongs to the exiting segment, as the light beam and the red segment are parallel. Right: Second, we introduce a virtual object point O along the entry ray, and we apply twice the focal point rule to find its image point, once for each focal point (blue and green dashed curves). This yields a second point I, which belongs to the exit ray, as any light beam that goes through O must also go through I, as they form a stigmatic pair.

2.1.2 Putting lens systems and sensors together

The sensor used to obtain the measure image (Section 1.1) is put at a constant distance D_s from the rear focal plane. For real-world cameras, this distance depends on the camera model. The two possible cases for any point of the sensor are illustrated by Figure 2.6. If the point is "part of" a stigmatic pair, *i.e.* it is at the same position as an image point, only light rays coming from the associated object point will contribute to it. If the sensor point is not at the same place as an image point, light rays coming from different object points will contribute to it, leading to a incoherent contributions, and therefore blur in the final image. An important relation, illustrated in Figure 2.7, makes it possible to compute the focal length f which is needed to get objects in focus, from their projected distance D_o along the optical axis:

$$\frac{1}{D_o} + \frac{1}{D_s} = \frac{1}{f}. \tag{2.1}$$

Depth of field: All the points which are not in focus contribute to a region of the sensor, instead of contributing to a single point. For a given object point, when the size of the associated region on the sensor covers less than a pixel on the final image, the object still appears in focus. The size along the view axis of the perceptually in-focus zone is called *depth of field* (Figure 2.6).

Aperture: As all rays with the same incident angle cross at a common point on the focal plane, it is

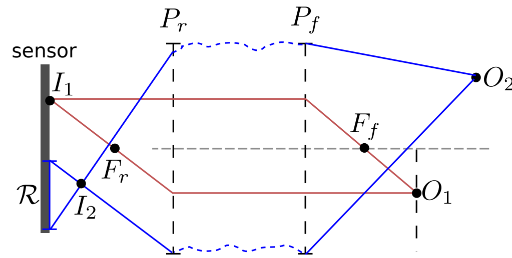


Figure 2.6: I_1, image point associated to O_1, is on the sensor, therefore O_1 is in focus in the image. By contrast, the image point I_2 of O_2 is not on the sensor, so O_2 is not in focus on the image. In fact, the rays leaving from O_2 contribute to a region of the sensor ($\mathcal{R}$), leading to blur.

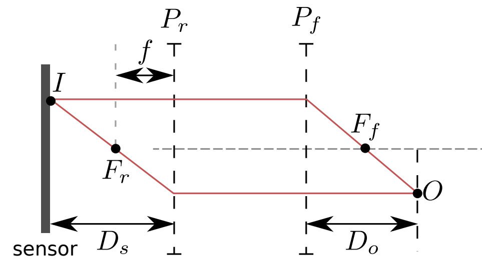


Figure 2.7: Conjugate relation: Points whose projected distance is equal to D_o are in focus on the image.

possible to filter incident angles by placing an occluder with an opening in it, centered at the focal point. This is illustrated by Figure 2.8. This occluder is called a *diaphragm* (although in real cameras it is placed inside the lens system, as the focal plane's position vary with the zoom level). The effect of the diaphragm is characterized by an *aperture angle*. This angle is the maximum angle of the incident light rays which are not blocked by the diaphragm. Note that lenses also put a limit on the maximum aperture angle, and is represented by *f-numbers*: the lower this number, the larger the maximum angle allowed. A modification of this aperture angle has two consequences. When increased, more light is collected during a same amount of time, leading to brighter images. But as more directions can contribute to a single sensor point, this also lead to an increase in blur of object points not in the focal plane, *i.e.* a reduced depth of field, as points of the sensor which are not part of stigmatic pairs receive contributions from more directions.

Bijection property: An important property is that the geometrical model should associate a single point $\mathbf{x_l}$ on the exit lens (of coordinates (u, v) on the lens), as well as a single beam direction ω_l, to a given "beam identifier“ on the sensor $(\mathbf{x_s}, \omega_i)$ (with $\mathbf{x_s}$ having associated 2D coordinates (x, y)). The reverse property should also be valid. This allows us to find where a radiance beam contributes on the sensor from any incident ray, and also allows us to compute both the direction ω_l of any ray leaving from the exit lens, and the incident direction on the sensor, ω_i, based on the coordinates (x, y) on the sensor and (u, v) on the exit lens. This "camera ray computation“ case, of particular importance for rendering, is shown in Figure 2.9.

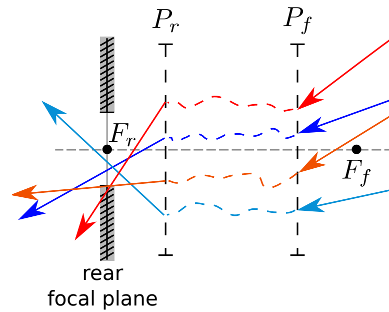


Figure 2.8: Diaphragm: All the rays whose incident angle on the front nodal plane is larger than a given threshold are blocked by the diaphragm (grayed dashed zone).

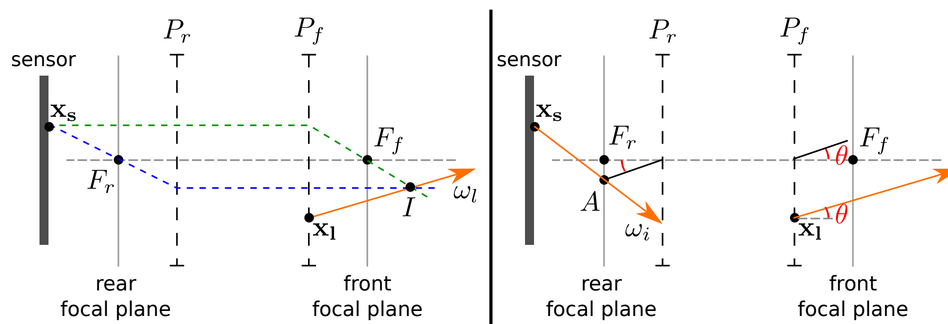


Figure 2.9: Left: First, ω_l is obtained from $\mathbf{x_s}$ and $\mathbf{x_l}$ by finding the image point I of $\mathbf{x_s}$. The camera ray's origin is $\mathbf{x_l}$, and its direction is the direction going from $\mathbf{x_l}$ to I. Right: Second, ω_i is found by using the angular rules of nodal points and focal planes.

2.1.3 Radiometric model

Scattering inside the lens system can affect the spectral distribution of radiance, or the color of the radiance beam for color-space rendering. To take it into account, one can use a radiometric model. A single point $\mathbf{x_l}$ on the exit lens and a single direction ω_l should correspond to a given coordinate (x, y) on the sensor and a given direction w_i. Therefore, we can use a $W(\mathbf{x_l}, \omega_l, t)$ function to transform incident radiance arriving on the lens to radiance arriving on the sensor. The t value allows us to model the effect of varying aperture, which is essential to correctly simulate motion-blur. This function has a similar notation as the notation of Veach in its Ph.D. thesis [Vea97] for its *importance function* W_e, as they are similar.

Putting together the radiometric model of a camera and the bijection property and adding some approximations, it is possible to reformulate Equation (1.68), an integral on directions incident on the sensor, to an integral over the points on the lens and which only uses radiance incident on the lens. A similar derivation for the case of the pin-hole camera can be found in Philip Dutré's Ph.D. thesis [Dut94], where all approximations which are made are detailed. The final model we obtain is common in physically-based rendering.

Let $\mathbf{x_s}$ be the point on the sensor for sensor coordinates (x, y), and $\mathbf{x_l}$ be the point on the lens for lens coordinates (u, v). Using the geometrical rules described above, we can compute ω_i, the incident coordinate on the sensor, and ω_l, the incident coordinate on the lens. Both are functions of (x, y) and (u, v), but this is not present in the equations for readability.

The base integral is (Equation (1.68)):

$$S(x,y) = C\left(\int_T \int_{\mathcal{S}^2} R(\omega_i) dE_i(x,y,\omega_i,t)dt\right). \tag{2.2}$$

Switching to an integral over the directions to an integral over lens points, we have:

$$S(x,y) = C\left(\int_T \int_{\mathcal{L}} R(\omega_i) L_i(x,y,\omega_i,t) \left|\mathbf{N_{x_s}} \cdot \omega_i\right| \left|\frac{d\sigma\left(\omega_i\right)}{dA(u,v)}\right| dA(u,v)dt\right) \tag{2.3}$$

where $\left|\frac{d\sigma(\omega_i)}{dA(u,v)}\right|$ is the determinant of the Jacobian matrix of the change of variable, and $\mathcal{L}$ is the set of (u, v) coordinates on the lens (in general, they are mapped from a unit square to a disk). Adding the camera radiometric model, we obtain:

$$S(x,y) = C\left(\int_T \int_{\mathcal{L}} R(\omega_i) W(\mathbf{x_l},\omega_l,t) L_i(\mathbf{x_l},\omega_l,t) \left|\mathbf{N_{x_s}} \cdot \omega_i\right| \left|\frac{d\sigma\left(\omega_i\right)}{dA(u,v)}\right| dA(u,v)dt\right). \tag{2.4}$$

Until now, no additional approximations have been performed compared to Equation (1.68). We now perform some crude approximations, but their impact is low on the final images while greatly simplifying the expressions. First, we assume that there is no directional sensitivity for the sensor: $R(\omega_i)$ is equal to one. Second, the $\left|\mathbf{N_{x_s}} \cdot \omega_i\right| \left|\frac{d\sigma(\omega_i)}{dA(u,v)}\right|$ term is simply ignored, which allows us to completely drop the

terms linked to the sensor. This gives us the simpler equation:

$$S(x,y) = C\left(\int_T \int_{\mathcal{L}} W(\mathbf{x_l},\omega_l,t)L_i(\mathbf{x_l},\omega_l,t)dA(u,v)dt\right). \tag{2.5}$$

This is the formulation we use in our rendering engine. Some rendering engines use more advanced formulations, to take into account effects such as chromatic aberrations or vignetting.

2.2 Light sources

Light sources are entities for which self-emission in the visible range occurs (Section 1.6.1). They can be put in two different categories: local light sources, and distant ones. For both cases, the actual emission function (L_e) can be directly set by the user, or can be obtained from models. The former case can be advantageously handled by textures, as they are generic tools. In the latter case, it depends on the light source type.

2.2.1 Local light sources

Local light sources have an associated geometry (even if punctual). Some examples are omni-directional punctual light sources, spots, or objects emitting lights (called area light sources).

Radiometric models: Radiometric properties can be obtained from an idealized emitter, called *black-body*. It has been developed by physicists during the nineteenth century and is very interesting in rendering. As a matter of fact, not only it lead to the discovery of quantum physics (therefore making possible the creation of modern computers where rendering is computationally feasible), it also helps describing light emission spectra. This material emits in all wavelengths, and perfectly absorbs any incident electromagnetic energy. Its emission spectrum solely depends on the temperature of the material, this spectrum being given by the Planck's law. Many common illuminants can be described as approximations of a black body, and therefore a single temperature can be given instead of specifying a full spectrum when describing spectral emission. Recently, Wilkie *et al.* proposed an improved model for rendering, based on black-body radiation [WW11]. Moreover, the CIE consortium gives the chromaticities of various common illuminants in various color-spaces for color-space rendering.

2.2.2 Distant light sources

Distant light sources are the ones where the source is so distant that considering only a direction is enough (*i.e.* $L_e(\mathbf{x},\omega) = L_e(\omega)\ \forall\mathbf{x}$). Some examples of such light sources are the earth sky (when considered as a light source by itself), the sun, or light coming from a distant environment.

Radiometric models: Models specific to the light source have to be used. For instance, Preetham *et al.* developed a well-known model for daylight emission spectra of the earth atmosphere [PSS99]. This model is heavily based on observed data, these data being fitted on an analytical model.

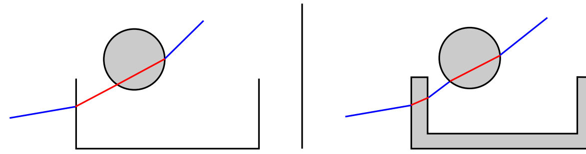


Figure 2.10: Impact of realistic objects on rendering of refraction, only considering geometrical aspects. It is illustrated for a scene with a glass and a sphere made of glass, both having the same index of refraction η different from 1 (cold air), and a ray arriving from the left. A blue segment indicates that the index of refraction tracked by the ray for the media in which it lies is 1. A red segment indicates that this index of refraction is η. Left: a glass is represented by 0-thickness borders, which makes the surface non-realistic. Right: same glass, with a non-zero thickness, giving a realistic surface. Having a realistic surface ensures a correct tracking of the index of refraction of the media the ray is in, and a correct trajectory of radiance beams. As shown by Veach in Chapter 5 of his thesis [Vea97], radiometric errors are present as well when non-realistic objects are used.

2.3 Geometry

Defining geometry is equivalent to defining surfaces. Several representations compatible with ray-tracing are available in the literature, we briefly present some of them in this section. We then give a very high-level overview of the methods used to implement an efficient ray-tracing function when many of these basic representations are used in a scene.

2.3.1 Geometry representation

Several basic representations are available to represent geometry. From the point of view of ray-tracing and physically-based rendering, the most important properties are the existence of accurate and efficient ray-surface intersection computation methods, and the closed 2D-manifold property. Closed 2D-manifold surfaces are qualified as *realistic* surfaces in the remaining of this section, and objects represented by a realistic surface are themselves realistic objects. The ray-surface intersection property is linked to computational time and precision, the closed 2D-manifold one is linked to correct handling of non-local effects such as refraction, where both entry and exit points must exist. As a matter of fact, if a ray enters a transparent non-realistic object with an index of refraction different from the previous media, it can leave it without hitting another boundary. As presented in Figure 2.10, non-realistic objects can lead to large errors. Therefore, representations ensuring realistic surfaces are preferable.

2.3.1.1 Polygon meshes

The most commonly used way to define a surface nowadays is by discretizing it in small triangular or quadrangular planar sections, called *polygons*. Polygons with more than four sides can be used as well, but it is harder to guarantee planarity (even with four sides it is not always easy for artists), and they are in general transformed to triangles before rendering takes place. Each polygon is delimited by *edges*, which are segments linking the *vertices* of the polygon. By making polygons share edges, it is possible to define a hole-free shape, without replicating data such as the vertices positions. Polygon meshes are

easy to create by artists, are easy to animate and can be very accurately and efficiently intersected with a ray [PH04]. Moreover, there exists so-called *tessellation* algorithms to transform most types of surface descriptions to polygon meshes, which places them as a sort of unified geometric representation. All these advantages explain their popularity.

However, it lacks flexibility or robustness in several domains: applying topological operations (such as creating holes in a correct way) on them is a research domain in itself, as well as ensuring smooth surfaces or adequate polygon density with respect to the amount of details present in the actual surface. As a matter of fact, this requires updating the topology of the polygon mesh, represented by the edges.

Moreover, it is possible for an artist to create non-realistic surfaces. Note that a non-realistic polygon mesh can be detected but it is difficult to automatically and accurately ”fill the holes“ (think of the glass example, filling the hole would not give a glass with thick borders, but a large rectangular piece of matter instead).

2.3.1.2 Implicit surfaces

In opposition to polygon meshes, where the surface is explicitly defined, implicit surfaces build on the resolution of equations. As a matter of fact, an implicit surface S is defined as the set of points which satisfy an equation, in general involving a function f and a constant value T in an equation of the form:

$$S = \{\mathbf{p} \in \mathbb{R}^3 / f(p) = T\}, \tag{2.6}$$

or, equivalently but exhibiting the root-finding nature of the underlying algorithms:

$$S = \{\mathbf{p} \in \mathbb{R}^3 / f(p) - T = 0\}. \tag{2.7}$$

f can be any function from $\mathbb{R}^3$ to $\mathbb{R}$. Most importantly, it can be the composition of several simpler functions. This leads to several modeling mechanisms, some of them very general (CSG modelisation) or more specific (metaballs) [MWB$^+$96].

When defined as a composition of simple functions, it has a low memory consumption, it is easy to animate, the topology changes that might occur are handled transparently, and the surface is automatically realistic. However, defining sharp edges remains difficult, and the control of the blending to match a desired behavior remains a challenge, even if important progress has been made recently.

The main difficulty with respect to ray-based rendering is computing the intersection of a ray with the surface. As a matter of fact, it requires finding the lowest root t of the equation:

$$f(\mathbf{o} + t \times d) - T = 0, \tag{2.8}$$

where $\mathbf{o}$ is the origin of the ray and d its direction.

Except for the simplest cases, this is not feasible analytically, and approximate methods have to be used. Two approaches are possible: either converting the surface to a polygon mesh (for instance marching cube [WMW86, LC87]), or finding the ray-surface intersection using root-finding methods

(ray-marching [Bar86, Lev88]). The first method comes at the cost of increased memory cost as well as precision and temporal artifacts problems, but allows for fast intersection as it is handled by ray-triangle intersection. The second method is more precise, but the per-ray cost can be relatively large, and it can also have precision problems if the function varies rapidly (or equivalently, has high-frequency content). A thorough state of the art can be found in [Kno07].

Moreover, when the function is a combination of several other functions, the intersection computation can not in general be expressed using simpler tests on sub-functions, independently of the combination: the function has to be considered in its generality. However, and as will be briefly presented in Section 2.3.2.3, this can be done when using sub-functions having compact supports, *i.e.* whose definition domain is not $\mathbb{R}^3$, but can be entirely bounded by a finite box (or any other bounding volume, such as a sphere).

2.3.1.3 Parametric surfaces

In opposition to implicit surfaces where the surface is defined as the set of roots of a function, parametric surfaces directly operate on an abstract 2D parameter space, and gives the 3D point associated to each parameters couple. Therefore, each point of the surface is given by applying a function f to a given couple of parameters (u, v):

$$\mathbf{p}(u, v) = f(u, v). \tag{2.9}$$

Examples of such surfaces include Bezier patches or bicubic patches. Ray-tracing such surfaces involves finding (u, v, t) such that

$$\mathbf{o} + t \times d = f(u, v). \tag{2.10}$$

Depending on f, this can be done analytically, or require the use of numerical methods. Note that some intersection methods between a ray and a triangle or a quadrangular rely on the fact that these geometry representations can be expressed as parametric surfaces, using barycentric coordinates as parameters. For instance for a triangle:

$$\mathbf{p}(u, v) = u \times \mathbf{p_0} + v \times \mathbf{p_1} + (1 - u - v) \times \mathbf{p_2} \tag{2.11}$$

where $\mathbf{p_0}, \mathbf{p_1}$ and $\mathbf{p_2}$ are the vertices of the triangle. Here, the parameter space is the 2D space for which $0 \leq u \leq 1$, $0 \leq v \leq 1$, and $0 \leq u + v \leq 1$.

Similarly to polygon meshes, no guarantee can be brought on the realistic nature of the objects created using parametric surfaces.

2.3.1.4 Voxels

Another useful representation is based on the notion of *voxel*. Instead of representing an object by its boundaries, a voxel representation directly represents the volume it occupies. As two objects can not occupy the same portion of space, it is possible to directly partition a 3D volume in little axis-aligned cubical zones (a voxel), each zone being unoccupied, or occupied by an object, with its attributes (color,

surface normal when it is a boundary voxel, *etc.*). This representation is receiving a great deal of interest as it allows to mix naturally volumetric elements such as participating media and objects in a single representation, and are very well suited for multi-resolution representations.

As voxels are very practical for visualization and are very well suited for parallel processing on graphics processing units (GPUs), rapid voxelization algorithms have been developed, to transform a polygon mesh to voxels. This allows users to use meshes for modeling and animation (which is far easier than voxels), and then switch to voxels for computations, maybe using a hybrid voxel/mesh representation to avoid the geometric aliasing problems of voxels while benefiting from their advantages.

Thanks to their very regular structures, voxel-based representations are very easy to ray-trace, and automatically lead to closed objects (a ray entering an object will eventually explicitly cross an exit point, as a voxel is a volume). However, they can require a lot of memory, as the more geometrical precision we want, the smaller the voxels must be, leading to potentially very large datasets.

For a state-of-the-art use of voxels in an out-of-core GPU visualization context as well as hybrid meshes/voxels representations, refer to Cyril Crassin's Ph.D. thesis [Cra11].

2.3.1.5 Point sets

The last type of geometry description we present are point sets. Here, a geometry is represented as a set of points belonging to the surface. These points are not linked together, unlike polygons where edges bring this topological information. This makes it easy to adapt the points set density to the local frequency content: a few points where it is regular, and a lot more where it is irregular. They are easy to obtain from 3D scans or by pre-processing another surface representation.

Building a surface from a point set can be done by fitting kernels of varying shapes (planes, spheres, *etc.*) [GG07, OGG09]. This makes them closely related to implicit surfaces, as they are expressed in terms of a function which can be evaluated at any point. Ray-tracing these surfaces can therefore be done using the same tools as for implicit surfaces, additionally exploiting the fact that the function has a particular structure. Note that depending on the methods, surfaces obtained from point sets may not be realistic.

2.3.2 Efficient ray-tracing function

Common scenes can contain several millions basic surface elements, which can be a mix of the above representations. As light transport equations make an extensive use of the ray-tracing function (Section 1.8), specific acceleration methods need to be developed. As a matter of fact, a direct ray-tracing algorithm would have a $O(N)$ complexity, where N is the number of surface elements: simply compute the intersection of the ray with all objects, and keep the nearest one.

As ray-tracing is very similar to a spatial search, structures to partition the space are used to perform this search more efficiently. Two kinds of structures can be found: spatial subdivision structures, and object subdivision structures. As we only deal with physically-based rendering, where rendering times can take up to several hours, we only indicate hints on construction times, but do not discuss it deeply.

The remaining of this part is a very-high level overview, for a much more precise description, see Chapter 4 of [PH04].

2.3.2.1 Spatial subdivision

Spatial subdivision structures create a spatial partition of space, dividing it in cells. An object can belong to several cells.

Regular grids: The simplest type of such a structure is the regular grid, which divides the space in cells of same size. Building grids is straightforward and very fast. The ray-tracing function then consists in browsing the few cells that are traversed, in order. In each such cell, the intersection of the objects lying in this cell with the ray is computed, stopping the traversal whenever a suitable intersection is found. The acceleration comes from the fact that the number of objects intersected with the ray should be much lower than N.

Kd-trees: Regular grids have robustness problems when objects of very varying sizes are present in a scene (think of a teapot in the middle of a stadium), because the spatial subdivision resolution has to be uniform over all the scene. To overcome this problem, kd-trees subdivide recursively the space, cutting a node in two along a selected axis. It can therefore adapt to the content of a scene, by having a larger depth where more precision is needed. This structure allows for a traversal with complexity $O(logN)$ by traversing the tree (maybe avoiding large portion of the scene if the ray does not cross the corresponding node), which is far more acceptable than $O(N)$ and makes ray-tracing a practical technique.

2.3.2.2 Object subdivision

For ray-tracing, object subdivision consists in creating groups or clusters of objects, and putting them in a (most often binary) hierarchy. The most used for ray-tracing is the *bounding volume hierarchy* (BVH). In this structure, the axis-aligned bounding box of each node is equal to the union of the bounding boxes of the two sub-nodes. Ultimately, the bounding box of a leaf node is the union of the bounding boxes of the contained objects. Basic ray-tracing using a BVH is a recursive process: from the root node, compute intersection in the left node if the ray hits the left node's bounding box, same thing for the right node, and keep the nearest intersection. If the node is a leaf, compute the intersection with all the objects in this leaf. Of course, this very basic version can be highly optimized to compute as few ray-object intersections as possible.

Several types of bounding volume hierarchies exist, depending on whether they use the semantic structure of a scene (given by a scene graph), or if they directly use the base representations (triangles, parametric surfaces, *etc.*) without considering semantic links. The latter ones are the most used, as they are the most efficient.

BVHs and extensions ([DHK08] for instance) acting directly on base representations are much easier to build efficiently than kd-trees [Wal07], provide the same $O(logN)$ ray-tracing complexity, and in

practice give performances that are similar to kd-trees on current generation computers. This explains why this structure is widely used nowadays.

2.3.2.3 Hybrids

Some structures or algorithms are based on a mix of space and object subdivision, or take from one side to improve the other side. For instance, [SFD09] attempts at improving the BVH performances when objects (in this case represented solely by triangles) have largely varying sizes, by splitting them. A way to generalize it to handle any type of basic representation is to use several non-overlapping boxes tightly bounding the object, and then build a BVH on these bounding boxes. When the bounding box of the object does not tightly bound it, using several tighter bounding boxes can lead to substantial improvements.

Another example is the *bounding interval hierarchy* acceleration structure [KW06], which is a mix between a kd-tree and a BVH.

As a side note, in parallel to this thesis, we developed an hybrid structure for implicit surfaces rendering [GPP$^+$10]. As it is completely unrelated to physically-based rendering, it will not be detailed in the contributions of this document. The goal was to develop an acceleration structure dedicated to ray-tracing thousands of metaballs (a specific type of implicit surfaces) on the GPU for interactive to real-time rendering. The structure we came up with, called *fitted BVH*, is an example of hybrid acceleration structure, as it is based on BVHs in spirit – with each node being tightly bound to its geometric content – , while ensuring that no node overlaps another one. Note that this structure can be trivially extended to handle arbitrary implicit surfaces of the form:

$$f(\mathbf{p}) = \sum f_i(\mathbf{p}) \tag{2.12}$$

where each f_i is a function with compact support. As a matter of fact, our structure uses the bounding box of the support of the particular f_i defining metaballs, but it could be any other function as long as the bounding box of the f_i's support can be computed.

2.4 Materials

Once the geometry has been defined, its appearance has to be described in some way. In physically-based rendering, this can be done through the use of materials. A material is a high-level description, relying on underlying physical models (plastic, lambertian, conductor metal, dielectric, *etc.*). Each model has an associated type of BSDF, whose parameters are the parameters of the material. For instance for a plastic material, an Ashikhmin-Shirley BSDF model [AS00] can be used, its parameters being the color of the diffuse (bottom) layer, the color of the reflective layer, the index of refraction of the reflective layer and two parameters describing the microscopic normals distribution. Similarly for a transparent material, a possible BSDF model is the rough transparent BSDF [WMHT07], whose parameters are the index of refraction inside the medium (for instance 1.5 if we want the object to be made of glass), the level of roughness, and so on. Note that it is possible to mix several BSDF models using *layered models*, such as

what is done in [PH04], or, more correctly by taking into account the interactions between the layers, in [WW07].

To summarize, the goal of a material is to compute a BSDF (model + parameters) for each point of a surface. The computed BSDF at $\mathbf{x}$ can then be used to obtain values of $f_s(\mathbf{x}, \omega_i \leftrightarrow \omega_o)$ for any ω_i and ω_o.

Real-world objects are rarely uniform: their color changes from point to point, as well as their roughness, their reflectiveness, *etc.*. In rendering, this translates in making the parameters of a BSDF vary spatially. These variations are controlled through *textures*. Each parameter of a material is obtained from a texture. This texture can be a constant, an image, a mathematical function, or the composition of other textures. This two last types lead to the field of procedural texturing [EMP$^+$02] and shader trees.

If the texture is an image or a 2D function, we see arising the need to retrieve a 2D coordinate from a 3D point on a surface. Making it in a correct way, with continuous 2D coordinates for a continuous piece of surface while avoiding distortions and visible seams, is a complex problem, known as parametrization. Some basic representations lend themselves to an easier parametrization than others. For instance, parametric surfaces, from their nature, have a natural 2D parametrization, while implicit surfaces are more difficult to parametrize in a satisfactory way.

2.5 Participating media

Finally, participating media can be described using various models: 3D grids of values (coming from measures or physical simulations based on the finite-elements method), which look like voxel grids, or 3D functions (for instance the Ebert's procedural cloud system, described in [EMP$^+$02]).

As solving the radiative transfer equation (Equation (1.57)) involves integrating these functions along a ray, there must be a method to get an accurate value. This is why, when the function is not directly integrable, participating media are often (but not always) discretized, for instance using a grid. This makes analytical computations possible, at the cost of introducing a discretization error.

2.6 Conclusion

We have identified five types of abstract entities required to represent any scene: cameras (Section 2.1), light sources (Section 2.2), geometries (Section 2.3), materials (Section 2.4) and participating media (Section 2.5). The nature of these entities is constrained by the light transport equations, as these entities must give the data which is used by these equations: materials have to compute BSDFs and therefore have parameters linked to the underlying BSDF models, participating media have to represent the absorption, scattering, and self-emission coefficients, the intersection between a ray and a geometry has to be computable in an efficient way, *etc.*. An important exception is the camera, which has a double-way interaction: light transport equations impose the existence of a radiometric model, but the signal value equation (Equation (1.68)) is changed to simplify its resolution (Equation (2.5)), at the cost of minor additional approximations.

The entities we just described give us a lot of freedom to specify the properties of a scene. Complex geometries, complex appearances, complex lighting and complex participating media make intractable the analytical resolution of Equation (1.26), using Equation (2.5) to compute the signal value. This is why numerical methods have to be used, which we present in the next chapter.

3

Mathematical tools for physically-based rendering

Computing the value of each pixel means computing the value of the integral in Equation (1.26), which itself implies computing the integral in Equation (2.5). Equation (1.26) is in general computed indirectly, as pixel filters often overlap, which makes a single signal value contribute to several pixels. There is therefore a decoupling: Equation (2.5) is computed for a given (x, y) sensor coordinate, and then the value of all the impacted pixels is updated in a suitable way, presented in Section 4.1.1.1.

In this chapter, we thus focus on the evaluation of Equation (2.5). This equation is a recursive integral, as the radiance is computed using the light transport equations, which are recursive. Analytical resolution not being possible for arbitrary scenes, numerical integration has to be used. In this case, it is better to have a single, non-recursive integral to handle, rather than a "tree" of integrals to solve for each evaluation of Equation (2.5). This is the reason why a so-called *path-integral formulation* has been developed (Section 3.1). This integral is then solved using a numerical method which has to be general enough to be able to handle it (Sections 3.2 to 3.7).

3.1 Path-integral formulation

Formulations of light transport equations on more global spaces than the unit sphere of directions and a length along a ray can be obtained by recursively expanding Equation (1.55) or Equation (1.57), and making some re-arrangements. The goal of these formulations is to obtain a pure integration problem for the estimation of the value of a pixel, using a single function instead of a recursive formulation. More formally, the goal is to go from the set of nested integrals summarized in Section 1.9.2 and Section 1.9.4 to a single non-nested integral over a more general space Ω:

$$I_p = \int_{\Omega} R_p(\overline{x}) f(\overline{x}) d\mu\left(\overline{x}\right) \tag{3.1}$$

where $\overline{x}$ is an *optical path*, which links a light to the sensor *via* an arbitrary number of *vertices*, which represent scattering events (on surfaces or in participating media). R_p contains all the terms that depend

on the pixel p, while f is independent from pixels, and contains the radiometric part of the integral. More precisely, f gives the radiance leaving from a point on a light which reaches the sensor, when the light follows the trajectory given by the vertices. Therefore, the value of each pixel can be obtained by summing the contribution of each such path. Ω is called the *path space*.

A very complete treatment of path-integral formulation is done in chapter 8 of Veach Ph.D. thesis [Vea97]. Here, we focus on aspects that are not completely handled in his thesis, namely handling local and distant light sources in a same framework, as well as participating media.

3.1.1 Path representation and light sources types

A path can be represented by different descriptions, depending on the light sources nature. In our case, we handle local light sources (area lights, spot lights, *etc.*, *cf.* Section 2.2.1), and distant light sources (environment maps, physical skies, *etc.*, *cf.* Section 2.2.1).

For local light sources, a path is just a tuple of points $(\mathbf{x_0}, ..., \mathbf{x_n})$ on the surfaces or in participating media, except for the first point which is on a light and the last on the camera's exit lens. The coordinates on the sensor can then be found from $\mathbf{x_n}$ and $\mathbf{x_{n-1}}$ using the bijection property as shown in Section 2.1.2. For distant light sources, a path is a tuple of one direction and a set of points on surfaces and participating media, *i.e.* $\overline{x} = (\omega_l, \mathbf{x_1}, ..., \mathbf{x_n})$, the direction ω_l pointing to the light when leaving from $\mathbf{x_1}$, and the last point being on the camera's lens. To avoid specific notations for each type of paths, when the light is distant, $\mathbf{x_0}$ will correspond to a point at infinity, with the direction from $\mathbf{x_1}$ to $\mathbf{x_0}$ being equal to ω_l. Each point has an associated *vertex*, which contains attributes such as the normal (when applicable), the BSDF, *etc.*, at that point.

With the unified representation for $\mathbf{x_0}$ presented above, an *edge* of a path is a segment between two adjacent vertices. For both types of paths, there is no need that adjacent points are mutually visible. More precisely, when two successive points of a path $\overline{x}$ are not mutually visible, no radiance reaches the sensor, and therefore $f(\overline{x}) = 0$.

Sensor attributes: The last vertex of a path is placed on the exit lens, not on the sensor, as the trajectory of the light beam inside the lens system is unknown. The path should therefore store attributes such as coordinates on the sensor, and other data that may be used by the sensing function (Section 1.5.2) and the camera model (Section 2.1) to compute the impact of the scattering that will occur in the lens system, before actually reaching the sensor.

3.1.2 Path space, path space measure

The Ω space contains all the paths, and is infinite-dimensional, as paths have no length limit. It is just the union of the paths of all lengths and of all types, each length and type defining a subspace $\Omega_{k,t}$ (k being the number of edges of a path, t being either "l" for local light sources path or "d" for distant light sources paths). For each $\Omega_{k,t}$ path subspace, there are plenty of ways to measure the size of an elementary set of paths, thus we have to choose one for each length and each type. A natural way to measure an elementary set of paths of $\Omega_{k,l}$ (*i.e.* a way to define $d\mu_{k,l}(\overline{x})$ for a local light source path of

length k) is to take the product of the elementary areas for points on surfaces and elementary volumes for points in participating media, and the measure associated to the sensor for $\mathbf{x_k}$ (it can be expressed with respect to the final image size, or with respect to its geometrical size). Letting $d_n(\mathbf{x_i})$ be the measure associated to a vertex ($d\mathcal{A}(\mathbf{x_i})$ if it is on a surface, $dV(\mathbf{x_i})$ if it is in a participating medium, and the sensor measure for $\mathbf{x_k}$), we have:

$$d\mu_{k,l}(\overline{x}) = d_n(\mathbf{x_0}) \times \cdots \times d_n(\mathbf{x_k}) \tag{3.2}$$

For distant light sources paths, the only thing which changes is the nature of the "link" with the light: for implementation purposes, it is easier to avoid points at infinity, and directly deal with the fact that the link with the light is a direction. A natural way to handle this is to define the measure as the product of the elementary solid angle defined by the direction and the elementary areas or volumes at each point, which leads to:

$$d\mu_{k,d}(\overline{x}) = d\sigma(\omega_l) \times d_n(\mathbf{x_1}) \times \cdots \times d_n(\mathbf{x_k}) \tag{3.3}$$

Then, $d\mu(\overline{x})$ is just $d\mu_{|\overline{x}|_E, type(\overline{x})}(\overline{x})$, where $|\overline{x}|_E$ is the number of edges of $\overline{x}$.

A way to unify all this in terms of notations is to use the point at infinity system for paths with a distant light source. By saying that $d_n(\mathbf{x_0})$ is equal to the solid-angle measure of the associated direction $d\sigma(\omega_l)$ when $\mathbf{x_0}$ is a point at infinity, we obtain a unified path space measure:

$$d\mu_k(\overline{x}) = d_n(\mathbf{x_0}) \times \cdots \times d_n(\mathbf{x_k}). \tag{3.4}$$

From $d\sigma(\omega)$ to $d\mathcal{A}(\mathbf{x})$ or $dV(\mathbf{x})$: In order to express the f function of the path-space integral from the light transport equations, it is necessary to switch measures from directions to areas or volumes. It can be shown that the following expressions hold, assuming that the direction ω leaves from a point $\mathbf{y}$ to reach a point $\mathbf{x}$:

$$d\sigma(\omega) = \frac{|\mathbf{N_x} \cdot \omega|}{||\mathbf{y} - \mathbf{x}||^2} d\mathcal{A}(\mathbf{x}) \tag{3.5}$$

$$d\sigma(\omega) = \frac{1}{||\mathbf{y} - \mathbf{x}||^2 \times dt} dV(\mathbf{x}). \tag{3.6}$$

The first one can be intuitively understood by reversing it, leaving from a solid-angle. The goal is to find the area of the surface around $\mathbf{x}$ which spans this solid-angle. As we deal with elementary elements, the surface can be considered as planar. The further away $\mathbf{x}$ is from $\mathbf{y}$, the larger the surface must be to span the same solid-angle as seen from $\mathbf{x}$, the ratio being given by $||\mathbf{y} - \mathbf{x}||^2$. Moreover, the less the surface around $\mathbf{x}$ is perpendicular to ω, the larger it must be to span the same solid angle, hence the $|\mathbf{N_x} \cdot \omega|$ factor.

The second one is obtained from the first, by considering an elementary volume extruded from an elementary disk centered at $\mathbf{x}$, perpendicular to ω, the length of the extrusion being infinitesimal, equal

to dt. As we deal with infinitesimal solid-angles and lengths, the extrusion can be considered as parallel to ω, and yields a cylindrical volume aligned with ω. Therefore, we have that $dV(\mathbf{x}) = d\mathcal{A}(\mathbf{x}) \times dt$, and we know that $d\sigma(\omega) \times ||\mathbf{y}-\mathbf{x}||^2 = d\mathcal{A}(\mathbf{x})$ (the dot product is equal to 1 because the elementary disk is perpendicular to ω). Therefore, $dV(\mathbf{x}) = d\sigma(\omega) \times ||\mathbf{y}-\mathbf{x}||^2 \times dt$, which leads to the second equation.

In the remaining of this document, $G(\mathbf{x} \leftrightarrow \mathbf{y})$ will denote the term to switch from $d_n(\mathbf{x})$ to $d\sigma(\omega)$, *i.e.* :

- if $\mathbf{x}$ is on a surface, $G(\mathbf{x} \leftrightarrow \mathbf{y}) = |\mathbf{N_x} \cdot \omega| \, / \, ||\mathbf{y}-\mathbf{x}||^2$,
- if $\mathbf{x}$ is on a volume, $G(\mathbf{x} \leftrightarrow \mathbf{y}) = 1/(||\mathbf{y}-\mathbf{x}||^2 \times dt)$,
- if $\mathbf{x}$ is a point at infinity, $G(\mathbf{x} \leftrightarrow \mathbf{y}) = 1$.

G is called *geometric term*.

3.1.3 R_p and f

In the context of color-space rendering, R_p contains the filtering part of the pixel value (Section 1.5) as well as the response of the sensor to the color-space radiance $f(\overline{x})$:

$$R_p(\overline{x}) = h_P(x_p - x(\overline{x}), y_p - y(\overline{x}))R(\overline{x}). \tag{3.7}$$

where (x_p, y_p) are the coordinates of the center of the filter associated to the pixel on the sensor, and $x(\overline{x})$ gives the x coordinate of the point on the sensor hit by the path, similarly for $y(\overline{x})$. $R(\overline{x})$ is the sensor's response for the contribution of $\overline{x}$, and depends on the incident direction on the sensor. If using the approximations of Equation (2.5), $R(\overline{x}) = 1$ for all paths. Note that all coordinates on the sensor should be computed coherently with the measure taken for the sensor (a pixel position such as $(630.45, 430.03)$ if the measure is based on the final image size, or geometrical coordinates in meters such as $(0.01, 0.0076)$ if the measure is based on the geometrical size of the sensor).

$f(\overline{x})$ contains all the rest, *i.e.* the emission by a light source from $\mathbf{x_0}$ (Section 2.2), the scattering events at each intermediate vertex (Section 1.7), the attenuation which occurs along each edge because of the participating medium (Section 1.8.2), and the attenuation due to the lens system (Section 2.1.3). To be coherent with the path-space measure, it also contains the geometric terms:

$$\begin{aligned} f(\overline{x}) = \quad & L_e(\mathbf{x_0} \to \mathbf{x_1}) \, Tr(\mathbf{x_0} \to \mathbf{x_1}) G(\mathbf{x_0} \leftrightarrow \mathbf{x_1}) \, W(\overline{x}) \times \\ & \prod_{i=1}^{|\overline{x}|_E} \left[f_s(\mathbf{x_{i-1}} \to \mathbf{x_i} \to \mathbf{x_{i+1}}) \, Tr(\mathbf{x_i} \to \mathbf{x_{i+1}}) G(\mathbf{x_i} \leftrightarrow \mathbf{x_{i+1}}) \right] \end{aligned} \tag{3.8}$$

where $|\overline{x}|_E$ is the number of edges of $\overline{x}$.

Note that here for simplicity reason, $f_s(\mathbf{x_{i-1}} \to \mathbf{x_i} \to \mathbf{x_{i+1}})$ can be either a BSDF or a phase function, depending on whether $\mathbf{x_i}$ is on a surface or in a participating medium.

3.2 Tools for numerical integration

Neither the light transport integrals nor the path space integral can be efficiently computed using deterministic quadratures methods because of their nature in the general case. The closed-form expression of the light transport integrals (Equation (1.55) and Equation (1.57)) is not known because they are recursive, and the path-space integral is on an infinitely-dimensional space. In addition, both integrals share very poor properties, the most impairing being the absence of continuity due to the visibility function.

This is why we resort to probabilistic methods, which explore the integration space by sampling. The method of choice in rendering is the Monte-Carlo method. The basic idea is to use a pseudo-random-number generator (PRNG) to generate numbers in $[0, 1)$ and transform them in directions, paths or whatever is useful to solve the integrals. All this can be described formally in the probability theory context, which we briefly review in Section 3.3. We next briefly present the standard Monte-Carlo estimator for integration (Section 3.4), and some methods to improve it (Sections 3.5 and 3.6). We finally present how the Monte-Carlo method can be used to solve the path-integration formulation (Section 3.7).

All these topics are treated in depth in many mathematics books ([KW86]) or rendering books or Ph.D. thesis ([PH04, Vea97]). Instead of writing yet-another thorough review, we prefer to focus on the foundations and make them clear, so that reading the details presented in the aforementioned documents can be done more easily. We also focus sometimes on details that can make rendering with Monte-Carlo methods tedious, and which must be taken into account as early as possible, such as Dirac-delta distributions.

3.3 Probability to have a section on probabilities = 1

3.3.1 Random variable

The notion of '*random variable* is at the root of Monte-Carlo methods, it is thus important to define it precisely.

Formally, a random variable X is just a *function* defined over a base space Ω_1 and with values in another space Ω_2, having some properties which makes it essential in every applications using probabilities. To put it rapidly, it allows to define a probability measure on Ω_2 from one defined on Ω_1. Thus, a random variable is just a "link" between two *probability spaces*, the base one being called the "sample space", and the other one the "state space". Note that although the name contains the word "random", a random variable in itself is a completely deterministic function: as will be detailed below, the randomness comes from the sample space.

A probability space is constituted of three things:

1. a *measurable* space Ω (for instance $[0, 1)$). Measurable means approximately that there exists a function which can give the "canonical size" of any part of this space, the *Lebesgue* measure (which correspond to length for $\mathbb{R}$, area for $\mathbb{R}^2$, *etc.*). Any other measure defined over a measurable space is a function of the Lebesgue measure.

2. a σ-algebra $\mathcal{F}$ on this space. Intuitively, $\mathcal{F}$ is the set of *events* that will arrive with a certain probability. Being a σ-algebra ensures the coherency of the notion of event: if an event is present, then its "negation" is also present. The full path space is an event, and thus the empty event is also present. For instance in the $[0, 1)$ case, $\mathcal{F}$ can be the set of all the intervals included in $[0, 1)$.

3. a probability distribution, which gives the probability with which an event in $\mathcal{F}$ will be observed if an observation is done. $\mathcal{P}$ must respect the Kolmogorov axiomatic to be a probability measure. There are 3 axioms:

   (a) $0 \leq \mathcal{P}(E) \leq 1$

   (b) $\mathcal{P}(\Omega) = 1$

   (c) The additivity of probabilities of disjoint events: if A and B are disjoint events, then $P(A + B) = P(A) + P(B)$.

   For instance, suppose we have a true uniform random number generator on $[0, 1)$, and $\mathcal{F}$ is the set of all the intervals included in $[0, 1)$. Then $\mathcal{P}(E)$ is the length of the interval E (which is the event "the random number generated (observed) is in E").

We can define more precisely what is a measure: a measure is a function defined on a σ-algebra $\mathcal{F}$, which gives a value in $\mathcal{R}^+$. A measure has some properties to ensure coherency, such that a part $E \in \mathcal{F}$ of Ω can't have a smaller size than any subset of E. The notion of distribution is used to describe all the types of probabilities that can be encountered, but in general (no Dirac probabilities) it is simply a measure on Ω whose value for the whole space is 1.

With this in mind, a random variable X is a *measurable* $\mathcal{F}_1/\mathcal{F}_2$ function that is defined from the *sample* space Ω_1 to the *state* space Ω_2 (again, it is just a transformation). Here, "measurable $\mathcal{F}_1/\mathcal{F}_2$" means that given the two σ-algebra $\mathcal{F}_1$ and $\mathcal{F}_2$, every antecedent of an element of $\mathcal{F}_2$ is an element of $\mathcal{F}_1$. The preimage of an event is defined by $X^{-1}(E) = \{\omega \in \Omega_1 | \omega = X^{-1}(\omega_2), \forall \omega_2 \in E\}$. In other words, every event of the state space corresponds to an event in the sample space. This automatically gives a probability measure on the (Ω_2, $\mathcal{F}_2$) space, this probability just being defined by $\mathcal{P}_2(E) = \mathcal{P}_1(X^{-1}(E))$. By definition of the antecedent of an element of $\mathcal{F}_2$, we see that $X^{-1}(\Omega_2) = \Omega_1$, and thus $\mathcal{P}_2(\Omega_2) = 1$, as we would expect from a probability measure. The fact that $\mathcal{P}_2$ satisfies the two others axioms, and is thus a probability distribution, can be demonstrated as well.

Let's take a little example to anchor this fundamental notion: the generation of the spherical coordinates of a 3D unitary direction in the upper hemisphere from two random numbers, using uniform mapping (will give X_u) and a more advanced scheme favoring directions closer to the vertical axis (will give X_c).

- X_u is defined as $X_u : (u_1, u_2) \rightarrow (2\pi \times u_1, \pi/2 \times u_2)$.
- X_c is defined as $X_c : (u_1, u_2) \rightarrow (2\pi \times u_1, \cos^{-1}(u_2))$.

These two random variables share the same source probability space:

- $\Omega_1 = [0,1) \times [0,1)$ (sample space of the two random variables) ;
- $\mathcal{F}_1$ is a convenient σ-algebra on Ω_1, for instance the σ-algebra containing the subsets of the form $[a,b) \times [c,d)$ and extended so that it conforms to the definition of a σ-algebra ;
- $\mathcal{P}_1 : E \rightarrow area(E)$.

They also share the same state space and associated σ-algebra, which are respectively the set of all the spherical coordinates of 3D unitary directions belonging to the upper hemisphere, and the equivalent of $\mathcal{F}_1$ on spherical coordinates.

But they define different target probability spaces, because the probability distribution derived from each one is very different, and they are themselves different, even if they produce the same type of "objects", here spherical coordinates of 3D unitary directions.

Now, we can define what is the *simulation* of a random variable: it is the process of "observing" an event in the sample space, and transforming it using the function defining the random variable (which results in an observation of the random variable). In computer science, it is common that the base probability space is the one of uniform random numbers in $[0,1)$, the base observations being done using a pseudo-random-number generator. All the "randomness" of the simulations comes from these base observations.

3.3.2 Probability density function, cumulative density function

Consider a probability space $(\Omega, \mathcal{F}, \mathcal{P})$. The probability distribution $\mathcal{P}$ is defined on the σ-algebra $\mathcal{F}$ of the domain, thus wanting to compute $\mathcal{P}(\omega), \omega \in \Omega$ has no meaning. To get a sense of the behavior of the probability measure at a given point, its density can be computed. Intuitively, the density at ω is the probability to observe samples in an elementary zone located around ω, divided by the size of the elementary zone (hence the name probability *density*). The "elementary" term makes arise the need to know how to measure the size of a zone, to know if it is elementary. And there are different ways of measuring a portion of a space, thus the density function will depend on the way we measure the space. Formally, the *probability density function* $p_\mu(\omega)$ (PDF), associated to $\mathcal{P}$ and to the measure μ on Ω, is defined as:

$$\int_{\mathcal{D}} p_\mu(\omega) d_\mu(\omega) = \mathcal{P}(\mathcal{D}) \tag{3.9}$$

where $D \in \mathcal{F}$.

For random variables with target space $\mathbb{R}$ (or any subset of it), the *cumulative density function* (CDF) is defined as the function that gives the probability associated to a special kind of events in the state space, namely the events of the form $X \leq x$. As a matter of fact, its definition is:

$$P_X(x) = \mathcal{P}(X \leq x) \tag{3.10}$$

where $\mathcal{P}$ is the probability measure on the state space. Thus, from this point of view, the CDF is just a restriction of the probability measure to some kind of events, but this restriction is enough to completely know the behavior of the random variable on the state space: it is a characterization. Note that the CDF

is measure-less, while the PDFs (one per measure) are not. To compute the PDF with respect to a given measure μ from the CDF, we leave from the definitions (we suppose that the target space is $\mathbb{R}$, it is the similar for bounded intervals):

$$\begin{aligned} P_X(x) &= \mathcal{P}(X \leq x) \\ &= \int_{-\infty}^{x} p_\mu(y) d\mu(y) \\ &= \int_{-\infty}^{x} p_\mu(y) \frac{d\mu(y)}{dy} dy \\ &= F(x) - F(-\infty) \end{aligned}$$

where F is a primitive of $p_\mu \frac{d\mu}{d}$.

By deriving with respect to x, we obtain:

$$\frac{dP_X(x)}{dx} = \frac{dF(x)}{dx}. \tag{3.11}$$

$F(-\infty)$ being constant with respect to x, its derivative with respect to x is zero.

Thus we have:

$$\begin{aligned} p_\mu(x) \frac{d\mu(x)}{dx} &= \frac{dP_X(x)}{dx} \\ p_\mu(x) &= \frac{dP_X(x)}{d\mu(x)}. \end{aligned} \tag{3.12}$$

3.3.2.1 Conversion of PDFs from a measure to another, change of variable

It is possible to convert a PDF $p_{\mu,X}$ of a random variable X defined on a state space Ω expressed in a measure μ to the same PDF $p_{\lambda,X}$ expressed in another measure λ. These functions will still represent exactly the same underlying probability distribution. As these functions are PDF, we have:

$$\begin{aligned} \int_\Omega p_{\mu,X}(x) d\mu(x) &= 1 \\ \int_\Omega p_{\lambda,X}(x) d\lambda(x) &= 1. \end{aligned}$$

From that, we get:

$$\int_\Omega p_{\mu,X}(x) \frac{d\mu(x)}{d\lambda(x)} d\lambda(x) = 1 \tag{3.13}$$

and thus:

$$p_{\lambda,X}(x) = p_{\mu,X}(x) \frac{d\mu(x)}{d\lambda(x)} \tag{3.14}$$

A change of variable creates a new random variable, on a different state space (for instance, changing from a direction to a point over a surface), with an associated PDF. The good thing is that the same formula holds for changes of variables, for instance to go from a PDF on directions to a PDF on points lying on a surface. Suppose we want to compute the probability density with respect to area $p_{\mathcal{A}}(\mathbf{y})$ to

have generated a point $\mathbf{y}$ ona surface from a point $\mathbf{x}$, knowing the probability density with respect to solid-angle $p_\sigma(\omega)$ with which the direction $\omega = \mathbf{x} \rightarrow \mathbf{y} = \frac{\mathbf{y}-\mathbf{x}}{||\mathbf{y}-\mathbf{x}||}$ has been generated. From Equation (3.14), we have:

$$p_{\mathcal{A}}(\mathbf{y}) = p_\sigma(\omega) \frac{d\sigma(\omega)}{d\mathcal{A}(\mathbf{y})}$$

and we know from Section 3.1.2 that $d\sigma(\omega) = G(\mathbf{x} \leftrightarrow \mathbf{y})\, d\mathcal{A}(\mathbf{y})$ if $\omega = \mathbf{x} \rightarrow \mathbf{y}$. Therefore:

$$p_{\mathcal{A}}(\mathbf{y}) = p_\sigma(\omega)\, G(\mathbf{x} \leftrightarrow \mathbf{y}).$$

3.3.3 Probability distributions and random variables, status of PRNGs

What does really mean "let $X \sim \mathcal{N}(0,1)$" ? If we remind us the definition of a random variable, it is a special function, which should have:

- a sample space Ω_1
- a state space Ω_2
- a source probability distribution $\mathcal{P}_1$ on a σ-algebra $\mathcal{F}_1$ of Ω_1
- a target probability distribution $\mathcal{P}_2$ on a σ-algebra $\mathcal{F}_2$ of Ω_2, resulting from the transformation of $\mathcal{P}_1$ by X

When saying that a random variable follows a given probability distribution (normal, exponential, ...) characterized by its PDF, it gives in fact the state space Ω_2 and the $\mathcal{P}_2$ distribution. The rest (Ω_1 and $\mathcal{P}_1$) are given by the simulation process. Thus, it does not characterize the random variable in itself, but its results. All the problem is then to find a random variable that produces these results (a simulation process, an algorithm). For instance, in computer science, pseudo random number generators are used to generate real numbers in $[0, 1)$, with a uniform repartition with respect to length. Thus, if we want a random variable following a Gaussian probability distribution, we have to find the function (the random variable) that maps $\Omega_1 = [0, 1)$ to $\mathbb{R}$ and the uniform probability distribution on $[0, 1)$ to the normal probability distribution, or, which is equivalent and much more tractable, the uniform PDF on $[0, 1)$ to the $\mathcal{N}(0, 1)$ PDF, the well-known $p(x) = \frac{1}{2\pi} \exp(\frac{-x^2}{2})$ (also expressed with respect to length).

Note that a PRNG is **not** a random variable. It is just a way to *observe* a random phenomenon described totally by its fully defined probability space $\mathcal{U} = ([0, 1), \mathcal{F}([0, 1)), \mathcal{P}_u)$ where $\mathcal{P}_u(E) = length(E)$, with $length(E)$ being the sum of the lengths of all the disjoint intervals of E. For instance, E can be $[0, 0.1] \cup [0.35, 0.7)$ and the length (and associated probability of the event represented by E) is 0.45.

3.3.3.1 Uniformity with respect to a measure

Sampling an element of any state space Ω uniformly with respect to a given measure μ in fact means defining a random variable X whose PDF with respect to μ is equal to $\frac{1}{\mu(\Omega)}$ for any element of Ω.

3.3.4 Equivalence of random variables, i.i.d. random variables, impact of PRNGs on independence

Let's say we have two random variables to map from $\mathcal{U}$ to a same target probability space (same state space, same probability distribution). In this case, we can say that these random variables are *equal*. If they do not have the same source probability space but share the same target probability space, they are *equal in distribution*. Thus, two random variables which do not perform the same mapping but share the same target probability space are just *equal in distribution*.. The good thing is that, in general, we do not care about this kind of details. Here, a distinction appears between algorithms and random variables: two (or more) random variables can use strictly the same algorithm, while not being equal: it depends on the source probability space of each random variable.

When speaking of a set of independent and identically distributed (i.i.d.) random variables $\{X_1, ..., X_n\}$, we just say that we take n "unlinked" random variables that have the same probability distribution, *i.e.* that are *equal in distribution*, not necessarily *equal*. This means that if we consider two random variables following a normal distribution, one using a rejection sampling algorithm and the other the Box-Muller one, they are i.i.d..

Formally, the independence of two random variables is characterized by the following properties on their PDFs. Two random variables X and Y are independent if and only if (we assume the Lebesgue measure on the respective spaces for each PDF)

$$p_{X,Y}(x, y) = p_X(x).p_Y(y) \tag{3.15}$$

On one computing thread, we often use a single PRNG (a single stream of pseudo random numbers) to generate all the observations of $\mathcal{U}$, which are then used to simulate every single random variable we want (thus, generate observations of these random variables). This may seem to remove any independence property between two any random variables X and Y simulated using the same stream, because the pseudo random numbers used to simulate X make it possible to know which ones will be used to simulate Y. But making this reasoning is not valid, because it supposes that we know we are using a PRNG, and that we know its state, which when considering X and Y is not the case. When we simulate X and Y, we just know that they take observations from $\mathcal{U}$ and transform them, each observation on $\mathcal{U}$ being supposed independent of the other. The last property must be ensured (or closely approximated) by the algorithm used in the PRNG, and is often tested through a statistical analysis of the generated sequences.

3.3.5 Function of random variable(s), expected value

A random variable X can be transformed by a function f defined on the state space Ω, to give another random variable $Y = f(X)$. For instance, consider a point $\mathbf{x}$, and a random variable D which generates directions distributed uniformly with respect to solid-angle from two random numbers. This is a random variable whose sample space is $[0,1)^2$ with a uniform probability distribution and whose state space is the set of 2D directions, $\mathcal{S}^2$, with a constant PDF with respect to solid-angle $p_\sigma(x) = 1/4\pi$. If we apply the function f which gives the incident radiance at $\mathbf{x}$ along a direction sampled from D, it gives another random variable $R = f(D)$ defined on the sample space of D $\left([0,1)^2\right)$ and whose state space is the space of incident radiance values (if we compute in CIE-RGB, it is the space of all CIE-RGB triplets for instance). As a matter of fact, it takes two uniform random numbers, and gives a radiance value.

We can also define a random variable as a function of several other random variables, for instance the random variable which returns the sum of n other random variables defined over compatibles state spaces for the sum.

The expected value $E[X]$ of a random variable X with state space Ω is used to represent the average value obtained when observing it. It is defined from the PDF of the random variable, and is defined as:

$$E[X] = \int_\Omega x.p_{\mu,X}(x)d\mu(x) \tag{3.16}$$

Note that the expected value does not depend on the measure taken for the PDF, and that the expected values of two random variables equal in distribution are equal.

The expected value of a function f of a random variable X can be defined with respect to the PDF of X:

$$E[f(X)] = \int_\Omega f(x)p_{\mu,X}(x)d\mu(x) \tag{3.17}$$

3.4 Estimators, standard Monte-Carlo estimator

An estimator is a way to compute the value of an unknown quantity from samples taken from one or more random variables. In rendering, the unknown quantity we want to compute is in general the value of an integral (being one of the light transport equations or the path integral (Equation (3.1)) for a given scene. A way to get an actual algorithm is to take a quantity for which we have a good estimator, and find a function of a random variable for which we can take samples from (*i.e.* we know how to simulate it). The expected value satisfies both conditions. In our case, a possible strategy is to find a function g of a random variable X whose expected value is the value of the integral we want to compute. More generally, for any integrable function f defined on a measurable space $\mathcal{D}$ and a random variable X with state space $\mathcal{D}$, we would like:

$$E[g(X)] = \int_\mathcal{D} f(x)d\mu(x) \tag{3.18}$$

with the only constraint that

$$f(x) \neq 0 \Rightarrow p_{\mu,X}(x) > 0 \ \forall x \in \mathcal{D}. \tag{3.19}$$

By definition of expected value, we have:

$$E\left[g(X)\right] = \int_D g(x) p_{\mu,X}(x) d\mu(x).$$

As we want

$$\int_D g(x) p_{\mu,X}(x) d\mu(x) = \int_D f(x) d\mu(x)$$

we obtain

$$g(x) = \frac{f(x)}{p_{\mu,X}(x)}. \tag{3.20}$$

Thus, $E\left[g(X)\right] = E\left[\frac{f(X)}{p_{\mu,X}(X)}\right]$ will converge to the value of the integral of any function f, as long as the constraint of Equation (3.19) is satisfied.

To estimate $E\left[g(X)\right]$, we use an estimator for the expected value, which is the well known average estimator:

$$E\left[X\right] \simeq \frac{1}{N} \sum x_i \tag{3.21}$$

where x_i are observations of N i.i.d. random variables $X_1, ...X_N$ at least equal in distribution to X. Thus, the final estimator (function of X and N) that we will use to estimate integrals is

$$F = \int_{\mathcal{D}} f(x) d\mu(x) = E\left[\frac{f(X)}{p_{\mu,X}(X)}\right] \simeq \frac{1}{N} \sum \frac{f(x_i)}{p_{\mu,X}(x_i)}. \tag{3.22}$$

This estimator is the standard Monte-Carlo estimator for integration, and is noted $< F >_{X,N}$ in the remaining for an integral F, a random variable X and a number of samples N.

Estimator template: Note that in the case of the standard Monte-Carlo estimator, each different random variable X as well as each different N value will give a different estimator of F. The "estimator" described by Equation (3.22) is thus more a model of estimator – an "estimator template" – than an estimator. This term will be used to represent a formula with so-called template parameters such as X or N, that gives estimators when all these parameters are set.

Estimator distribution: Estimators can be seen as functions of one or more random variables, and can therefore be considered themselves as random variables. They are thus distributed according to a probability distribution. In the case of the Monte-Carlo estimator, it has the same law as the average estimator of i.i.d. random variables, which can be shown to be a Student t-distribution when the number of samples is low. For estimators with large enough N values ($N > 30$), their distribution is actually very close from a normal law, thanks to the central-limit theorem. This property is crucial to compute the precision of an estimation, as will be detailed and used in Chapter 9.

3.4.1 Bias, variance, robustness

The qualities of an estimator are described by different notions, that can be "combined" between them to qualify an estimator. Three such notions will be presented here. The first one is about the systematic error, the second about the rate of convergence toward a stable value, and the third one about a correct computational behavior whatever the function to handle is.

Bias: The estimator used for the expected value is said to be *unbiased*. It means that the expected value of the estimator is the estimated quantity. For instance, for the expected value estimator with N samples:

$$\begin{aligned} E\left[\frac{1}{N}\sum_{i=1}^{N} x_i\right] &= \frac{1}{N}\sum_{i=1}^{N} E\left[X_i\right] \\ &= \frac{1}{N} N.E\left[X\right] \\ &= E\left[X\right] \end{aligned}$$

where we have used the fact that the X_i are identically distributed and at least equal in distribution to X (thus they have the same expected value) at the second line. Independence does not bring anything here.

There are 3 stages of "correctness" to describe an estimator with respect to the estimated value, which all refer to the notion of *bias*. The bias represents the systematic error committed by a given estimator, systematic in the sense that no matter what are the resources allocated for the estimation (time, memory, computing power, ..) there will always be an error in the computed value if the bias is not zero. Formally, the bias of an estimator $< \Theta >$ of an unknown quantity Θ is defined as

$$B(< \Theta >) = E\left[< \Theta >\right] - \Theta \tag{3.23}$$

From that, we get the following classification of estimators with respect to bias.

1. Unbiased: the expected value of the estimator is the estimated value.

2. Consistent: if the estimator is derived from an estimator template as defined above, then different estimators are obtained by changing one of the parameters of the template. If one of this parameter changes the bias of the resulting estimator in a monotonous way, we can consider a sequence of estimators $(< \Theta >_i)_{i\in\mathbb{N}}$ where $B(< \Theta >_{i+1}) \leq B(< \Theta >_i)\ \forall i \in \mathbb{N}$, with the parameter changing between each estimator of the sequence. The sequence of estimators $(< \Theta >_i)_{i\in\mathbb{N}}$ is said to be *consistent* if the sequence $B(< \Theta >_i)_{i\in\mathbb{N}}$ converges toward 0. When an estimator is said to be consistent, it in fact supposes that the estimator is obtained from an estimator template and that there exists a sequence of estimators taken from the estimator template that is consistent in the sense defined above. There is an important fact to note for computational simulation: a sequence of estimators can be considered only if it can actually be simulated. For instance, if the fixed parameter that makes evolve the bias is the memory consumption and the bias diminishes when

the memory increases toward a non-finite value, an estimator will never be consistent simulation-wise, because memory is not infinite.

3. biased: $B(<\Theta>) > 0$ and no sequence as described above exists to make it converge toward 0.

Variance: A second way to qualify an estimator (or more often an estimator template as described above) is its behavior with respect to variance. Although defined on random variables, we consider here variance as a measure of whether an estimator provides results close to its expected value (independently of bias). For an estimator $<\Theta>$ considered as a random variable, its variance $V[<\Theta>]$ is:

$$V[<\Theta>] = E\left[(<\Theta> - E[<\Theta>])^2\right]. \quad (3.24)$$

Note that another equation, more practical for computations, can be derived from Equation (3.24):

$$V[<\Theta>] = E\left[<\Theta>^2\right] - E[<\Theta>]^2. \quad (3.25)$$

The variance of an estimator is the average quadratic shift between an estimate and the exact value that should be computed by the estimator. In the case of an unbiased estimator, it is therefore the average quadratic shift between an estimate and the exact value of the estimated quantity. The lower the variance, the closer to the expected value any estimation is likely to be. As there may have more than one template parameter, it can be interesting to analyze the impact of each one on the variance of the resulting estimator (this is for instance what we do in Chapter 5). One way to study it is by evaluating analytically the variance of some of the estimators templates derived from the base one, by fixing all the parameters but one. In the general case for Monte-Carlo integration, we have:

$$\begin{aligned} V[<F>_{X,N}] &= V\left[\frac{1}{N}\sum_{i=1}^{N}\frac{f(X_i)}{p_{\mu,X}(X_i)}\right] \\ &= \frac{1}{N^2}\sum_{i=1}^{N} V\left[\frac{f(X_i)}{p_{\mu,X}(X_i)}\right] \\ &= \frac{N}{N^2} V\left[\frac{f(X)}{p_{\mu,X}(X)}\right] \\ &= \frac{1}{N} V\left[\frac{f(X)}{p_{\mu,X}(X)}\right]. \end{aligned} \quad (3.26)$$

- *Quadratic error:* By fixing the random variable and letting N vary, we therefore obtain that

$$V[<F>_{X,N}] \propto \frac{1}{N}. \quad (3.27)$$

A measure of probable error can be obtained using the standard deviation of the estimator, defined as

$$\sigma(<F>_{X,N}) = \sqrt{V[<F>_{X,N}]}. \quad (3.28)$$

As the standard deviation of $< F >_{X,N}$ is proportional to $\frac{1}{\sqrt{N}}$ *whatever the random variable used is*, dividing by two the error requires to use an estimator with four times more samples.

- *Importance sampling:* We can also study the variance with respect to the random variable X, and we get:

$$V\left[< F >_{X,N}\right] \propto V\left[\frac{f(X)}{p_{\mu,X}(X)}\right]. \tag{3.29}$$

Thus, whatever N is, we see that a correct choice of X will have a strong impact on the variance. More precisely, the more $\frac{f(X)}{p_{\mu,X}(X)}$ approaches constancy, the lower the variance of the resulting template estimator is, reaching 0 in the case of a perfect match $p_{\mu,X}(X) \propto f(X)$. This observation is at the roots of importance sampling which we detail afterward.

Robustness: Although the estimator template as presented in Equation (3.22) has a fixed f function, we can consider all the estimators template which have the same form. Robustness as we use it in this document is the property that whatever the function g to handle, an estimator of this form will not exhibit a very (arbitrarily) large variance. In rendering, this means that whatever the BSDFs, the lighting conditions, the geometric setup, *etc.*, rendering time for a given quality will remain stable, and no scene will lead to a very poor result. Chapter 5 is dedicated to improving the robustness of common estimators in rendering.

3.4.2 Dirac-delta distributions and Monte-Carlo

The Dirac-delta distribution is a mathematical construction which allows us to express that a given random variable X has a probability measure such that only one *value* x_0, which has a zero-measure, has a non-zero probability. Dirac-delta distributions are special because they are the base to associate a non-zero probability to an event with a zero measure. This is a way to express discrete events and probabilities in the context of continuous probability theory, and mix these notions. When only one zero-measure event has a non-zero probability, this is similar to determinism. Formally, the associated PDF, noted $\delta_\mu(x - x_0)$ at coordinate x, can be defined from its integral over an arbitrary domain $\mathcal{D}$:

$$\int_{\mathcal{D}} \delta_\mu(x - x_0)\, d\mu(x) = \begin{array}{l} 1 \text{ if } x_0 \in \mathcal{D} \\ 0 \text{ otherwise.} \end{array} \tag{3.30}$$

When several events with zero measure are possible, each associated Dirac PDF is multiplied by a weight. When only zero-measure events have a non-zero probability (case of a standard discrete probability distribution expressed in the context of continuous random variable), the weight of each Dirac is equal to the probability of each event, in the sense that would be given in a discrete probability distribution context.

If $x_0 \in \mathcal{D}$, we have:

$$\int_{\mathcal{D}} f(x)\delta_\mu(x - x_0)\, d\mu(x) = f(x_0). \tag{3.31}$$

From that we get the equality to change the measure of a Dirac distribution:

$$\int_{\mathcal{D}} f(x)\delta_\mu (x - x_0)\, d\mu(x) = f(x_0) \tag{3.32}$$

$$\int_{\mathcal{D}} f(x)\delta_\lambda (x - x_0) \frac{d\lambda(x)}{d\mu(x)} d\mu(x) = f(x_0) \tag{3.33}$$

$$\delta_\lambda (x - x_0) \frac{d\lambda(x)}{d\mu(x)} = \delta_\mu (x - x_0) \tag{3.34}$$

which is what is expected for a PDF.

When using Dirac-delta distribution in computations (which arises in rendering, when dealing with point lights, perfect mirror BSDFs, *etc.*), we keep only the changes of measures. As a matter of fact, Dirac-delta distributions can't be computed directly (we can't give any floating point value for $\delta_\mu (x - x_0)$), but they can simplify each other. Thus when writing $f(X)/p(X)$, we must be sure that if the expression of $f(X)$ makes appear a Dirac distribution (coming from X), it can be simplified by one of $p(X)$ with the same reference value, otherwise the expression can not be evaluated. With the simplification, only the changes of measure remains.

The main question is to find the measure in which the Dirac is at the "beginning", which often consists in knowing the analytical answer for the Dirac part and conclude from it. Consider the specular reflection problem (which involves a Dirac-delta distribution). We want to compute the following in a Monte-Carlo setting:

$$L_o(\mathbf{x}, \omega_o) = \int_{S^2} f_s(\mathbf{x}, \omega_o \leftrightarrow \omega_i)\, L_i(\mathbf{x}, \omega_i)\, |\mathbf{N}_\mathbf{x} \cdot \omega_i| \tag{3.35}$$

From physics, we know that for perfect mirrors, we have:

$$L_o(\mathbf{x}, \omega_o) = L_i(\mathbf{x}, r(\omega_o)) \tag{3.36}$$

where $r(\omega)$ is the reflected direction of ω. Thus from equation 3.31, we know that for specular reflections

$$f_s(\mathbf{x}, \omega_o \leftrightarrow \omega_i) = \delta_{\sigma_\mathbf{x}^\perp}(\omega - r(\omega_o)) \tag{3.37}$$

where $\sigma_\mathbf{x}^\perp$ is called the *projected solid-angle measure*, and is defined as $d\sigma_\mathbf{x}^\perp(\omega) = |\mathbf{N}_\mathbf{x} \cdot \omega|\, d\sigma(\omega)$.

Therefore, when estimating the integral using a formulation in solid angle and using a direction random variable D whose probability distribution is the Dirac-delta distribution above, we need to convert it from its nominal measure of projected solid-angle to the solid-angle measure. As the only direction

with a non-zero probability from D is $r(\omega_o)$, this yields:

$$\begin{aligned}
< L_o\left(\mathbf{x},\omega_o\right) >_{,D,N} &= \frac{1}{N}\sum_{i=1}^{N}\frac{f_s\left(\mathbf{x},\omega_o \leftrightarrow D_i\right)L_i\left(\mathbf{x},D_i\right)\left|\mathbf{N_x}\cdot D_i\right|}{p_\sigma\left(D_i\right)} \\
&= \frac{1}{N}\sum_{i=1}^{N}\frac{\delta_{\sigma_{\mathbf{x}}^{\perp}}\left(D_i - r(\omega_o)\right)L_i\left(\mathbf{x},D_i\right)\left|\mathbf{N_x}\cdot D_i\right|}{\delta_\sigma\left(D_i - r(\omega_o)\right)} \\
&= \frac{1}{N}\sum_{i=1}^{N}\frac{L_i\left(\mathbf{x},r(\omega_o)\right)\left|\mathbf{N_x}\cdot r(\omega_o)\right|}{d\sigma_{\mathbf{x}}^{\perp}\left(r(\omega_o)\right)/d\sigma\left(x\right)r(\omega_o)} \\
&= \frac{1}{N}\sum_{i=1}^{N}\frac{L_i\left(\mathbf{x},\omega_o\right)\left|\mathbf{N_x}\cdot r(\omega_o)\right|}{\left|\mathbf{N_x}\cdot r(\omega_o)\right|} \\
&= L_i\left(\mathbf{x},\omega_o\right)r(\omega_o).
\end{aligned}$$

All that to finally say that for implementation, if constant functions are the equivalent of heaven, then Dirac-delta distributions are clearly the evil. As a matter of fact, when implemented in a classic way, the evaluation of the Dirac-delta distributions can not be done easily: we have to manually take care in the algorithms implementations of where Dirac-delta distributions can arise, where two of them can simplify each other, *etc.*. Dirac-delta distributions are never actually evaluated, it is ratios of such distributions which are evaluated. And even then, when computing the ratio of two distributions in a given measure, the evaluation is approximated: it always returns 0, unless we tell that the parameter (for instance, the outgoing direction) has been sampled from it. In this case it returns the change of measure. This approximation is done for efficiency reason: there are (nearly) zero chances to give in parameter the correct value (the one for which the Dirac-delta distribution is not 0), unless it has been sampled from it. Thus, making the comparison each time (with all the reserve due to approximations introduced by the finite precision of floating point representations) would be a big waste of time.

3.5 Variance reduction methods

As described in Section 3.4.1, variance is a way to characterize the probable shift between any estimation and the actual value of the estimated quantity. It is therefore important to make variance as small as possible. As increasing the number of samples leads to a larger estimation time, other ways have to be explored, which reduce the variance without requiring more samples. We briefly present some method here. A more in-depth presentation specific to rendering can be found in chapter 2 of [Vea97], or chapters 14 and 15 of [PH04].

3.5.1 Importance sampling

The first way to reduce variance without increasing the number of samples, suggested in Section 3.4.1, is by using a random variable whose density mimics the function to integrate. This method, called importance sampling, is the most used in rendering. A perfect importance sampling (zero variance

estimator) can be obtained by using a random variable X whose PDF is proportional to the function: $p_{\mu,X}(x) = f(x)/\int_{\mathcal{D}} f(x)d\mu(x)$. The problem is that it requires $\int_{\mathcal{D}} f(x)d\mu(x)$, which is the quantity we want to estimate.

It is therefore more common to use approximate versions of f, where only a subset of the factors are considered. For instance in the case of the light transport equation (Equation (1.55)), one can simply consider only the BSDF term, which is fully known. In specific cases such as direct lighting, the incident radiance function can also be used for importance sampling, as it is totally described by the light sources.

3.5.1.1 Random variable creation

Finding a random variable X (an algorithm) generating samples on a domain $\mathcal{D}$ whose PDF with respect to measure μ, $p_{\mu,X}$, is the one we want to sample from is not an easy task. Some generic tools exist, with their advantages and their caveats. We briefly present two of them.

Rejection sampling: The idea of rejection sampling is to generate a candidate y from another, already known random variable Y defined on a superset of $\mathcal{D}$, whose PDF $p_{\mu,Y}$ has the property that there exist a $c > 0$ such that $p_{\mu,X}(x) < c \times p_{\mu,Y}(x)\ \forall x \in \mathcal{D}$. The candidate is sampled from Y, and a uniform random number $u \in [0, 1)$ is generated as well. y is then accepted as result of sampling X if

$$u < \frac{p_{\mu,X}(y)}{c \times p_{\mu,Y}(y)}. \tag{3.38}$$

If this is not the case, the process is repeated. Note that formally, we here consider the natural extension of $p_{\mu,X}$ which is 0 when $y \notin \mathcal{D}$.

This method is general, but requires to find the constant c, and may require a lot of candidates when $p_{\mu,X}$ and $p_{\mu,Y}$ highly differ from each other.

Analytical inversion: When the random variable state space is $\mathbb{R}$, it is possible to use the CDF to find a random variable X whose PDF is $p_{\mu,X}$. In fact, we want to find the transformation function g such that $X = g(U)$, where U is a uniform random variable over $[0, 1)$, and X is the random variable whose PDF is $p_{\mu,X}$. Letting P_X be the CDF of X, we have:

$$\begin{aligned} P_X(x) &= Pr\{X < x\} \\ &= Pr\{g(U) < x\} \\ &= Pr\left\{U < g^{-1}(x)\right\} \\ &= g^{-1}(x) \end{aligned} \tag{3.39}$$

and therefore $g(u) = P_X^{-1}(u)$, which means that $g = P_X^{-1}$. As shown in Chapter 14 of [PH04], this mechanism can be extended to random variables with multi-dimensional state spaces, at the cost of increased complexity. Sampling based on analytical inversion thus requires to be able to compute the CDF analytically, and invert it. This is not always feasible. That is why, for instance for complex BRDFs

models such as [AS00], importance sampling is not done with a random variable whose PDF is the normalized version of the real BSDF function. Most often, it is done with a random variable whose PDF is an approximation of the “perfect” PDF, approximation which can be analytically integrated and inverted.

The two methods presented above can be combined, for instance using analytical inversion on a simplified version of the function to get a close approximation of the actual PDF, and then using rejection sampling on the candidates generated from this approximation. This is for instance used in the Ziggurat algorithm, which efficiently generates random variables with a normal distribution [TM00].

3.5.2 Multiple importance sampling

Multiple importance sampling [VG95] is a method based on importance sampling, which consists in using a sum of importance sampling PDFs instead of a single one, combining the results in a way that minimizes the variance of the resulting estimator. We do not go in further details here, as a more detailed presentation is done in Chapter 5, where we present an improvement of multiple important sampling. Note that for vector-valued integrals, MIS can be used to limit the impact of the lack of correlation amongst components as presented above for standard importance sampling, by using one importance sampling PDF for each component.

3.5.3 Control variate

Several formulations of control-variate methods exist. Here we consider the formulation by Veach [Vea97], which consists in using an analytically-integrable function $\hat{f}$ mimicking f as a base, Monte-Carlo being used to integrate the difference, $f(x) - \hat{f}(x)$. Note that importance sampling and control variate do not seek the same thing. Importance sampling seeks a ratio $f(x)/p_{\mu,X}(x)$ as constant as possible, while control variate seeks a difference $f(x) - \hat{f}(x)$ as constant as possible. A bit more detailed presentation can be found in Chapter 9, where we present an adaptive control variate method for general numerical integration.

3.5.4 Uniform samples placement

The methods presented above act either on the random variable or on the function to integrate *via* the Monte-Carlo method. A key observation is that lower variance can also be obtained by correctly exploring the state space. This can be done by having uniform numbers which have a correct *practical* distribution, *i.e.* avoiding clutters of samples while some zones are completely unexplored for any estimation. This can most easily be formulated by considering that the random variable is the identity function on the space of uniform random numbers in $[0, 1)$, and that the function to integrate is the compound of the normal function f and the previous random variable, considered as a simple function on uniform random numbers.

For any function f and any random variable X whose sample space is a D-dimensional unit hyper-

cube $[0,1)^D$, we can rewrite the integration problem we want to solve:

$$\int_{\mathcal{D}} f(x) d\mu(x) = \int_{[0,1)^D} f(x(u)) \frac{d\mu(x(u))}{dL(u)} dL(u) \tag{3.40}$$

where L is the Lebesgue measure on $[0,1)^D$ (length if $D = 1$, area if $D = 2$, *etc.*). Noting that thanks to Equation (3.14), we have $p_{\mu,X}(x(U)) = p_{L,U}(U) \frac{dL(U)}{d\mu(x(U))}$, and that $p_{L,U}(U) = 1$ since we consider uniform random numbers on $[0,1)$, we obtain that $p_{\mu,X}(x(U)) \frac{dL(U)}{d\mu(x(U))}$, and therefore $\frac{d\mu(x(U))}{dL(U)} = \frac{1}{p_{\mu,X}(x(U))}$. We finally obtain a reformulation on the unit hypercube:

$$\int_{\mathcal{D}} f(x) d\mu(x) = \int_{[0,1)^D} \frac{f(x(u))}{p_{\mu,X}(x(u))} dL(u) \tag{3.41}$$

The new function to integrate is therefore $g(u) = \frac{f(x(u))}{p_{\mu,X}(x(u))}$.

3.5.4.1 Stratified sampling

Stratified sampling consists in applying the partition theorem for integration:

$$\int_{\mathcal{D}} f(x) dx = \sum_{i=1}^{S} \int_{\mathcal{D}_i} f(x) dx \tag{3.42}$$

where $\{\mathcal{D}_i\}_i, i \in \{1, \ldots, S\}$ is a partition of $\mathcal{D}$.

For Equation (3.41), it consists in partitioning the D-dimensional hypercube in strata $\mathcal{U}_i$, and then using a random variable U_i with a uniform probability density with respect to the Lebesgue measure for each stratum $\mathcal{U}_i$.

In each stratum $\mathcal{U}_i$, the probability density function has the form:

$$p_{L,U_i}(u_i) = \frac{1}{L(\mathcal{U}_i)}. \tag{3.43}$$

The estimator template corresponding to stratified sampling on the D-dimensional unit hypercube is therefore:

$$< F >= \sum_{i=1}^{S} \frac{1}{N_i} \sum_{j=1}^{N_i} g(U_{i,j}) \times L(\mathcal{U}_i). \tag{3.44}$$

In rendering, it is most often the case that the partition cuts the unit hypercube in an axis-aligned regular-grid scheme, the S cells having the same geometrical size, and therefore the same Lebesgue measure equal to $L([0,1)^D)/S = 1/S$. Moreover, an equal number of samples N_s is used in each stratum, leading to the following estimator:

$$< F >= \sum_{i=1}^{S} \frac{1}{N_s} \sum_{j=1}^{N_s} \frac{g(U_{i,j})}{S}. \tag{3.45}$$

This formulation has an advantage: it can be rewritten to look very similar to the standard Monte-Carlo estimator for f using $S \times N_s$ samples, where only the way the uniform random numbers are generated changes:

$$\begin{aligned} < F > &= \sum_{i=1}^{S} \frac{1}{N_s} \sum_{j=1}^{N_s} \frac{g(U_{i,j})}{S} \\ &= \frac{1}{S \times N_s} \sum_{i=1}^{S} \sum_{j=1}^{N_s} g(U_{i,j}) \\ &= \frac{1}{S \times N_s} \sum_{i=1}^{S} \sum_{j=1}^{N_s} \frac{f(x(U_{i,j})}{p_{\mu,X}(x(U_{i,j}))}. \end{aligned} \tag{3.46}$$

It is easy to derive the distribution of the stratified sampling estimator: each stratum has an independent estimator, which is a simple mean. It is therefore distributed according to the sum of each stratum's distribution, which is a Student t-distribution. When the number of samples per stratum N_s is sufficiently large, these distributions can be approximated by a normal law, whose sum is itself a normal law with summed average and summed variance. This makes it easy to obtain confidence intervals from this kind of estimators, which is important to assess convergence.

The specific formulation presented in Equation (3.46) is commonly used in rendering, but requires relatively square-shaped cells to correctly distribute samples in the unit hypercube. Therefore, for an optimal placement, a grid with the same number of cells N_c along each dimension is required, leading to $S = N_c^D$ cells. This rapidly becomes intractable when D increases, and has the disadvantage that it is not possible to use an arbitrary number of samples: it has to be a multiple of N_c^D. In Chapter 9, we show how to avoid these problems, and how to guarantee correctly shaped cells even for non-square domains for improved robustness in the context of general numerical integration.

3.5.4.2 Latin-hypercube sampling

Another well known method also plays with regular grid partitions of the D-dimensional space, but does not constraint the desired number of samples to have a specific form. Instead, from a desired number of samples for estimation N, it virtually creates a grid with N^D cells, and then select N cells in a way ensuring a proper distribution. A sample is then generated in each selected cells. The choice of the cell is done to ensure that no two cells have the same coordinates for a given dimension. For instance in 2D, the cells always have a different row *and* a different column than any other cell. This is done through permutations of the integers between 1 and N, $\pi_{i,N}$, one per dimension $i \in 1, \ldots, D$. The j-th cell coordinates in the grid are given by

$$(\pi_{1,N}(j), \ldots, \pi_{D,N}(j)) \tag{3.47}$$

where $\pi_{i,N}(j)$ is the j-th element of the permutation for the i-th dimension.

Latin-hypercube sampling leads to an unbiased estimator [MBC79], whose form is the same as the

standard Monte-Carlo estimator, the only change being the way each random number is generated.

Although having very interesting properties such as arbitrary number of samples, Latin-Hypercube sampling exhibits some defaults:

- Large parts of the integration space can be missed (think of the worst case, where all the permutations are equal, leading to samples only in the diagonal
- It is not easy to compute the precision of a single estimate done with N random numbers generated using Latin-hypercube sampling. As a matter of fact, even if there is a central-limit theorem for estimators based on Latin-hypercube sampling [Owe92], only an approximation of the variance of the normal law can be computed.

3.6 Non-standard Monte-Carlo methods

The methods presented in Section 3.5 lower the variance by changing the random variable (importance sampling), the function to integrate (control variate), or the estimator used (multiple importance sampling), but do not change the rate of convergence with respect to the number of samples. Recently, methods to improve this rate of convergence developed mainly for physics or finance have appeared in rendering. We briefly present Markov-Chain Monte-Carlo methods and adaptive methods.

Markov-Chain Monte-Carlo methods aim at perfect importance sampling using local samples perturbations. Two main type of methods have appeared in rendering: Metropolis sampling [MRR+53] and sequential methods. The first one has lead to Metropolis light transport [VG97] and its extensions [KSKAC02, SIP07]. The second kind of methods has been studied extensively by Fan in its Ph.D. thesis [Fan06].

Other methods focus on adaptively improving already known methods during the estimation. For instance, [DGMR05] aims at improving importance sampling based on the previous evaluations, using iterative optimization of parameters of blending between different known importance sampling functions. Therefore, they still use known importance sampling functions, but their combination is continuously improved, to yield a global PDF as close to the optimal one as possible. Similarly for control-variate, in Chapter 9, we develop a method which improves a specific control-variate function during evaluation, leading to largely improved performances and convergence rates for low-to-middle dimensional integrals.

All these methods closely link sampling and evaluation, as the next sample(s) potentially depend on the result of the evaluation of the previous sample(s). This possible dependency has to be taken into account when designing a versatile software architecture for physically-based rendering, as we do in Chapter 4.

3.7 Sampling optical paths

The vast majority of rendering algorithms spend their time sampling paths or pieces of paths. This is in general done using two tools: importance sampling and Russian roulette. We briefly describe these methods for the most common path sampling methods used in rendering algorithms.

3.7.1 Local importance sampling of optical paths

Paths are in general sampled in a random-walk fashion: beginning at a light or the camera, it is recursively expanded by sampling a direction and finding the next scattering point. Some methods leave from only one end, others create paths by sampling a sub-path leaving from the light, and another one leaving from the camera.

Importance sampling is used to sample points and directions, usually based on local information such as the reflection model at a point or the light emission properties. More advanced methods try to use as much information as possible (we present such a method in Chapter 5 in the framework of multiple importance sampling), but are more complex to use. These local importance sampling methods can be either based on formal derivations of the ideal random variable (*i.e.* a deriving the function which transforms uniform random numbers to what we want to sample, with the ideal probability density function), or based on random variables derived from approximations which try to match as closely as possible the ideal probability density function. We also tell when an given importance sampling method depends on a specific data.

- **Point on local light-source**: the goal is to first favor lights which emit the most energy, and then favor regions of the sampled light where the emission is the most important. For instance when using a texture to describe the spatial light emission distribution, favoring the brightest regions leads to a more adequate sampling. Ideally, the PDF with respect to area at a point should be proportional to the emitted irradiance at this point.

- **Direction from a local light-source**: from a point on the light source, favor directions for which the emission from the given point is largest.

- **Direction from a distant light-source**: favor directions for which the emission is largest. For instance for a sky model, the directions corresponding to the sun should be favored. Ideally, the PDF of a direction with respect to solid-angle as measured from any point in the scene should be proportional to the radiance received from this direction.

- **Direction from a BSDF**: from an incident direction ω_i, favor directions ω_o for which reflectance at point $\mathbf{x}$ ($f_s\left(\mathbf{x} \to \omega_\mathbf{i} \to \omega_\mathbf{o}\right)$) is largest. For instance for a glossy reflection model, directions which are in the glossy reflection zone associated to the incident direction should be favored. Ideally, the PDF with respect to solid angle of ω_o, $p_\sigma\left(\omega_o\right)$, should be proportional to the reflectance $f_s\left(\mathbf{x} \to \omega_\mathbf{i} \to \omega_\mathbf{o}\right)$. Most BSDF models used in computer graphics come with an associated sampling procedure, and some generic methods based on using approximations have been developed for the case where the set of BSDF parameters used in a scene are in finite and relatively low numbers, such as [LRR04]. Note that here, the PDF of w_o is a conditional PDF: in general, ω_i has been itself obtained using sampling.

- **Ray coordinate from a participating medium**: as the attenuation due to scattering is given by the transmittance $Tr(t)$, we should use a random variable whose PDF with respect to length is close or equal to $Tr(t)$. For some specific kind of mediums, it is possible to do it analytically, but

for most it is not possible, and specific methods are necessary. These methods are called '*free-path sampling* methods, because it aims at samplings lengths for which a beam does not interact with the medium, *i.e.* a length where it travels as in free-space. A kind of multiple-try method called Woodcock tracking has been developed long ago, and recently received attention for a wider use in rendering [SKTM11, YIC$^+$10]. Although efficient, it requires the computation of strict upper bounds of the maximum value of the coefficients on arbitrary regions, which can be a problem for highly varying mediums if no discretization is performed. Another option for free-path sampling is to use an approximation of the participating medium for which analytical sampling is possible. This approach has the advantage that it does not require any analytical property on the function, but can lead to large variance if the approximation is too coarse.

- **Camera parameters**: camera parameters include lens coordinates (for depth of field) and time coordinate (for motion blur). To minimize variance, the random variable used to generate camera parameters should have a PDF with respect to area $\times$ time (the measures used to integrate both on the lens and time interval) proportional to the radiometric response for these parameters.

3.7.2 Russian roulette

The path-space is infinite-dimensional. Thus, infinitely long paths should be sampled to ensure that the results are unbiased with respect to the full light-transport problem. As infinity is quite hard to reach, particularly toward the end, we need a way to stop making a path longer, without adding statistical bias. This is done through *Russian roulette*.

Russian roulette consists in replacing a random variable X by a "Russian-rouletted" version $X^{(r)}$, which has a probability equal to $1 - p$ to be equal to zero. In this case, when generating $X^{(r)}$, if zero is chosen, X does not have to be actually generated. If X is the radiance brought by a path which is generated randomly, this means that the path does not need to be generated, hence a large reduction of computation time. Formally, $X^{(r)}$ is defined as:

$$X^{(r)} = \frac{B_p}{p} \times X \tag{3.48}$$

where B_p is a Bernoulli random variable with probability p (it is worth one with probability p, and zero with probability $1 - p$): its PDF is given by $p_{B_p}(x) = p \times \delta(x - 1) + (1 - p) \times \delta(x - 0)$. Although

increasing variance, this skipping system does not add bias:

$$E\left[X^{(r)}\right] = E\left[\frac{B_p}{p} \times X\right] \quad (3.49)$$
$$= \frac{E\left[B_p \times X\right]}{p} \quad (3.50)$$
$$= \frac{E\left[B_p\right] \times E\left[X\right]}{p} \quad (3.51)$$
$$= \frac{p \times E\left[X\right]}{p} \quad (3.52)$$
$$= E\left[X\right] \quad (3.53)$$

where we have used the fact that B_p and X are independent at the third line, and that the expected value of a Bernoulli random variable with probability p is p at the fourth line.

In rendering, Russian roulette is most often used in algorithms which sequentially build paths of increasing lengths, performing one estimation for each path length. As it is costly to expand a path, it is better for performances to stop expanding a path as soon as the contributions are small.

3.8 Conclusion

The path integral formulation (Section 3.1) is at the core of most algorithms used to solve the light transport equations. This equation, as well as the light transport equations, can not be solved analytically, and exhibits properties which makes the use of deterministic methods difficult: infinite-dimensionality of the integration space, lack of continuity, *etc.*. This is why a probabilistic method, relying on random variables (Section 3.3) is most often used in rendering nowadays: the Monte-Carlo method (Section 3.4). In its standard form, this method, although general, has to be improved in order to be usable in practice for rendering (Section 3.5). More advanced methods, still based on the Monte-Carlo method, have also been proposed to get more effective algorithms (Section 3.6). Being the standard or more advanced Monte-Carlo methods, they both use common tools when applied to rendering (Section 3.7): a local generation of optical paths by sampling individual points and directions, and Russian roulette, to avoid as much as possible useless computations, without adding bias to the estimations.

Actual rendering algorithms are derived from particular formulations of the path-space integral or light transport equations, which lend to different types of resolutions, all relying on the Monte-Carlo method. These algorithms are used in a rendering engine. However, designing a software architecture flexible enough to handle most rendering algorithms is not easy. We now present the architecture we developed (Chapter 4), which aims at providing this flexibility, how current state-of-the-art methods fit in it, and then the technical contributions we propose to make the algorithms more robust and more efficient (Chapters 5 to 10).

Part III

Contributions

4

A versatile software architecture for physically-based rendering

All the technical contributions of this thesis have been developed in a coherent software architecture named *Flexray*, which stands for "FLEXible RAYtracer". The code of this rendering engine is freely available, under the terms of the GPL3 license: see `http://www.irit.fr/~Anthony.Pajot/code.php` for more details. Although at the moment of writing I am the only developer of Flexray, the goal is that other students continue its development in the future.

This software architecture is based on a component-based programming model, the components being inspired from the structure proposed in the book "Physically-Based Rendering" (PBRT in the remaining) [PH04].

In Flexray, any functional element for which a strong semantic can defined is a component interface. An actual component then implements this interface, potentially relying on other components interfaces. There are two possible behaviors with respect to data:

- *push*: a component can receive data to process from other components (for instance an image to store on disk), or it can send data to other components for processing. When a chain of components with this kind of relationship is set up, it can provide a pipeline. For instance, transform a set of accumulated spectral data to a CIE-XYZ image, transform it to RGB and tone-map it, and then display it on a screen. All these steps have different semantics, and each will have an associated component interface.

- *pull*: a component can explicitly require specific data from another component it is plugged to.

A rendering engine and the scene to render can then be described as a graph of components. This graph contains both the scene description (there are components for cameras, geometric primitives and all other constituents of a scene), and the algorithms used to compute the image. In Flexray, this graph is specified in an XML file. The part specifying the scene can be created through export scripts from 3D softwares such as blender, the set up of the rendering engine in itself being in an independent file for ease of use.

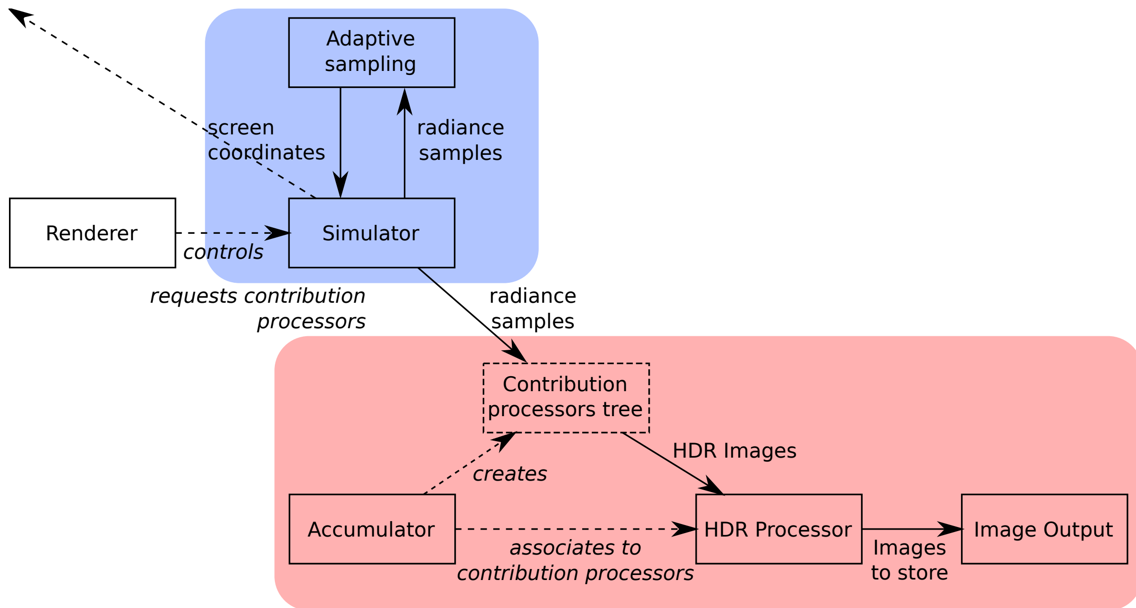


Figure 4.1: High-level relationships between components interfaces (in plain rectangles) and high level entities linked to the interfaces (in dashed rectangles) used in Flexray for a standard rendering. Dashed arrows indicate the nature of the link between components, plain arrows indicate data exchanges. The blue part is linked to the physical simulation, the red one to the image reconstruction and its processing.

Although not optimal in terms of performances (lots of abstractions, interfaces, virtual functions, *etc.*), this software choice of using components has proven to be very flexible and robust even to large changes. The best illustration we can give is that Flexray is being developed since 2008, and no major rewrite has been required during this time lapse, despite continuous extension/generalization, and despite the diversified nature of the technical contributions we present in this document.

The scene description part is mostly a direct adaptation from PBRT, except for light sources where a supplementary distinction between local and distant light sources has been made. By contrast, although inspired from PBRT, the rendering part presented in Section 4.1 has been largely generalized. As shown in Section 4.2, this architecture allows us to naturally integrate a large number of highly different algorithms, all taken from the current state-of-the-art in Monte-Carlo-based rendering.

4.1 From equations to software architecture

Similarly to PBRT, Flexray relies on a particular splitting of equations in two different parts: the image reconstruction and the radiance computation. From this decomposition, we exhibit the components interfaces which are needed to perform physically-based rendering. Figure 4.1 shows a general organization of the interfaces which are detailed below, for a standard rendering process (for which the goal is to obtain an image).

4.1.1 Image reconstruction and processing

The image reconstruction part builds the image from radiance samples. It is best described from the equations summarized in Section 1.9.4:

$$I_p = \int_{s(h_P)} h_P(\Delta_x, \Delta_y) S(x_P + \Delta_x, y_P + \Delta_y) d\mathcal{A}(\Delta_x, \Delta_y),$$

$$S(x,y) = C\left(\underbrace{\int_T \int_{\mathcal{S}^2} R(\omega_i) L_i(x,y,\omega_i,t) \left|\mathbf{N}_{\mathbf{x_s}} \cdot \omega_i\right| d\sigma\left(\omega_i\right) dt}_{L(x,y)}\right)$$

where $\mathbf{x_s}$ is the 3D point coordinates of (x, y) on the sensor.

Most of the components implementing the interfaces presented below are typical push-components: they receive data and send transformed data to other components.

4.1.1.1 Accumulator

The quantity computed by the accumulation part for each pixel is:

$$I_p = \int_{s(h_P)} h_P(x - x_p, y - y_p) S(x,y)) d\mathcal{A}(x - x_p, y - y_p). \tag{4.1}$$

Here, we split the estimation of $S(x, y)$ in two elements: the computation of the integrated radiance over time and lens coordinates ($L(x, y)$), and the transformation to the storage color-space, done by a function C (which can be identity if the storage color-space is the same as the computational color-space).

The accumulator part takes care of transforming $< L(x, y) >$ values to the storage color-space, and then accumulating them to obtain I_p. This is why we call it an *accumulator*. Each $< L(x, y) >$ is a N-samples estimate of $L(x, y)$, N being given with $< L(x, y) >$. Note that the accumulation must be done with care if the storage color-space is not linear: a suitable addition and scaling operator must be used. We obtain a N-samples estimate $< S(x, y) >$ of $S(x, y)$ from $< L(x, y) >$ by defining

$$< S(x,y) >= C\left(< L(x,y) >\right). \tag{4.2}$$

The x and y coordinates are not chosen by the accumulator part. In fact, it is even not possible to do so for some algorithms such as Metropolis light transport [VG97], which requires a total control on the sampling in order to be efficient.

Our system is made to be able to use several accumulators at the same time, through a “dispatching” component (a specific implementation of the accumulator interface) which simply sends the radiance samples to several accumulators. This is useful to test different accumulation strategies, all with the exact same radiance samples.

Standard accumulation: As sampling is not controlled, it is not easy to use the standard Monte-Carlo

estimator for Equation (4.1): the probability density of (x, y) is not easy to determine, as they are often unknown in advance when using some helping mechanisms such as adaptive screen sampling (Section 4.1.2.2). Instead, it is common to use a slightly biased estimator with a lower root-mean-square error (therefore leading to reduced noise), which does not require sensor position's probability densities:

$$< I_p >= \frac{\sum_{i=1}^{N_p} h_P(x_i - x_p, y_i - y_p) \times N_i \times < S(x_i, y_i) >}{\sum_{i=1}^{N_p} h_P(x_i - x_p, y_i - y_p) \times N_i}, \tag{4.3}$$

where x_i, y_i, N_i and $< S(x_i, y_i) >$ are given in parameters, and N_p is the number of (x_i, y_i) which belong to the support of the pixel's filter. Note that this estimator implicitly normalizes the reconstruction filter, removing the need to explicitly normalizing it.

When pixel filters overlap, a single $< S(x_i, y_i >$ value can contribute to several pixels. Therefore, the accumulation function loops over the pixels whose filter support include (x_i, y_i), and update their estimates. This operation is called *splatting*.

4.1.1.2 HDR processor

The accumulator handles radiance samples, and creates raw HDR images from them. The HDR processor interface allows us to process these images obtained from accumulation, for instance to make them suitable for display or storage. Specific implementations can either just convert these images to a desired color-space, apply an imaging pipeline such as the one described in [PH04] to switch to a low-dynamic range, perform post-processing on HDR images, *etc.*.

Similarly to the accumulation, a specific dispatching processor is used to have different processings applied to a given HDR image (for instance compute a tone-mapped image as well as the gradient of the HDR image at the same time).

4.1.1.3 Image output

Once the HDR images have been processed, they can be displayed, stored, printed, *etc.*, using an image output system. Several can be used at the same time through a dispatching image output.

4.1.2 Physical simulation

The task of evaluating $L(x, y) = \int_T \int_{\mathcal{L}} W(\mathbf{x_l}, \omega_l, t) L_i(\mathbf{x_l}, \omega_l, t) dA(u, v) dt$ (Equation (2.5)) is done by components which we call *simulators*.

The basic task of a simulator is to perform a "step" of simulation: it is the smallest operation which leads to the estimation of $L(x, y)$ values. Here, we can distinguish three kind of methods:

- the methods for which we can isolate as a step the sampling of a sensor coordinates and camera parameters and then the estimation of $L(x, y)$ using a single sample specified by the camera parameters (the most common in rendering, detailed below),

- the methods in which the values of (x, y) are obtained as a by-product of a more global sampling method, such as light tracing or Metropolis light transport,
- the methods which sample several (x, y) values at a time to benefit from correlation ([HOJ08, HJ09]) or increase workload (we present one such method in Chapter 10).

4.1.2.1 One-sample simulators

The methods of the first kind first sample sensor coordinates (x, y) using an arbitrary distribution (potentially using adaptive sampling), and then estimate the integral using a lens and time sample. This leads to the sampling of camera parameters ((u, v) positions on the lens), and a time coordinate t. Performing several estimates of this kind and accumulating them as described in Section 4.1.1.1, a complete image with motion blur and depth of field can be obtained. The one-sample simulator therefore relies on two sub-components: a *screen sampler* to obtain a (x, y, u, v, t) tuple (called a *screen sample*), and an *integrator* to compute $< L(x, y) >$ values from this screen sample.

Screen sampler: Generating the (x, y) coordinates on the sensor is in general done by using a uniform distribution, except if adaptive screen sampling (presented below) is used. (u, v) coordinates on the lens and a t value are then generated by the camera. The camera can use importance sampling to sample according to the radiometric attenuation due to the lens system (Section 2.1.3). This sampling also leads to the computation of the point $\mathbf{x_l}$ on the last lens of the lens system, and the direction ω_l at the exit of the lens system, according to the camera optical model (Section 2.1.1).

Integrator: The role of the integrator is to compute $< L(x, y) >$ using the given (u, v, t) values. Estimates $< L(x, y) >$ are of the form:

$$< L(x, y) >= \frac{W(u, v, t) \times L_i(\mathbf{x_l}, \omega_l, t) \, |\mathbf{N_{x_s}} \cdot \omega_i|}{p_{\mathcal{A}}(u, v) \, p_t(t)}. \tag{4.4}$$

Note that the probability density of (x, y) is not required as the accumulation phase takes care of it (Section 4.1.1.1), and the $p_{\mathcal{A}}(u, v)$ and $p_t(t)$ are given by the camera model. Integrators can be based either on the light transport equations or the path-space integral.

As shown in Section 4.2.1, most rendering methods are in fact integrators coupled with a generic screen sampler.

4.1.2.2 Adaptive screen sampling

Most of the simulation methods can benefit from a focused screen sampling: it is possible to use a kind of feedback to focus processing power on regions of the image where the quality is lowest, being caused by noise or geometric aliasing. This adaptation method is called *adaptive screen sampling*. Adaptive screen sampling components have in general a push and pull nature: they receive data from source components (in Figure 4.1 from simulators, because this is the most common case, but it can be from any other

source of information, such as morphological analysis of reconstructed images), and screen samplers or simulators ask them sensor coordinates. In the examples given in Section 4.2.3, we focus on adaptive-screen sampling methods which are based on information gathered from radiance samples given by the simulator.

4.1.2.3 Dealing with multiple contributions per step: contribution processors

Some integrators or simulators can lead to the computation of a contribution for more than one (x, y) pixel coordinate at each step. Moreover, these contributions may not be accumulated the same way. We deal with that by using *contribution processors* (CP), which are managed by accumulators, and used and combined by the simulators. This is a very flexible and general solution, which allows us to have possibly several accumulators linked to the simulator at the same time, as illustrated in Figure 4.2. We detail this figure below, but we can see that we define trees (or more generally directed acyclic graphes) of accumulators to compute the results we want. Each time, only the root node of this graphe is known by the simulator.

Each accumulator has its own type of contribution processor, the actual algorithms being executed inside the contribution processors. The accumulator in itself only manages these processors. For instance for the image reconstruction accumulator, the contribution processor accumulates the samples which are added as described by Equation (4.1). For a dispatcher, when a new contribution processor is requested, the dispatcher requests a CP to each "sub-accumulator" and returns a CP whose role is to dispatch the contributions to each of the sub-CPs. This yields a tree of CPs. Therefore, we can distinguish between a concrete contribution processor, which is linked to an accumulator and performs its own processing, and an *abstract* contribution processor, manipulated by the simulator, and whose entry point is the root contribution processor of the tree of CPs.

We now consider an example, presented in Figure 4.2. The top part of this figure represents the configuration of the system, as described by the user: we want to use a simulator based on bidirectional path-tracing (presented briefly in Section 4.2.1). As results, we compute a non-tonemapped EXR version and a tonemapped PNG version of the computed image, as well as a PNG image of the number of samples used per pixel.

The bottom part of Figure 4.2 presents which contribution processors are created, how they are linked between them, how the simulator interacts with each, and how they interact with the output components. The simulator requests three CPs to the dispatch accumulator, which is the root accumulator: the two for the radiance samples, and one for summing the two previous CPs, to obtain the final image. The request is done to the dispatch accumulator, which itself request CPs to the sub-accumulators it is linked to. As the sub-accumulators use output components which are not accumulators, each "leaf" CP has a simple link to the output component, to which images will be pushed when results have to be written.

During computations, the simulator pushes the radiance samples to the two first abstract CPs. When images have to be written, the third CP is computed as the sum of the two previous. The combination of CPs are done using a coherent combination approach: as all the abstract CPs have the same tree structure, it is possible to make a one-to-one correspondence between all the nodes of these trees, therefore com-

bining each CP of a tree with its equivalent in another tree. Each CP knows how to combine with another. For instance, there is nothing to do to combine dispatch contribution processors, and reconstruction CPs just have to add the pixel values.

4.1.3 Rendering control

As of now, no component controls the whole engine: the screen can be sampled, radiance samples can be added to accumulators, images can be produced and processed, but nothing coordinates all this. This is the role of the **renderer** part. It tells the simulator to perform a pre-process, perform steps and write images (which is done by asking the accumulators to compute images from the data they have, and which will be passed to the processors or outputs they are plugged to, thanks to the push nature of this pipeline), based on a time frequency, number of steps done, or any other suitable criteria. It also stops rendering when a given time limit or step limit is reached, or when the simulator tells that simulation is finished.

Note that by its formulation, the renderer part can in fact be used to control any process which can be expressed with steps.

4.2 Examples of components

We now show how common or recent methods in rendering or post-processing can fit in this framework, by giving examples for each component interfaces for which published work exists.

4.2.1 Integrator

As presented in Section 4.1.2.1, integrators have to compute $< L(x, y) >$ values from (x, y, u, v, t) screen sample.

Direct-lighting [PH04]: Direct lighting evaluates the light transport equation or the radiative transfer equation without recursion. Formally, it consists in replacing L_i by L_e everywhere. It is the same kind of computations OpenGL does, except that direct lighting uses a physically-based ground, and naturally handles soft shadows, motion blur, depth of field and participating medium, at the cost of much larger computation times. Specular or glossy reflection and refraction can be handled by evaluating the full light transport equation or radiative transfer equation when the BSDF has a non-zero glossy/specular component, only sampling directions for which the glossy/specular component is not zero. As it ignores diffuse inter-reflection, this method is biased.

Path-tracing [Kaj86]: This method is the simplest method which takes into account all of the light transport possibilities. This method works by splitting the path-space integral on a sum over path-spaces of different lengths (Section 3.1):

$$\int_{\Omega} R_p(\overline{x}) f(\overline{x}) d\mu\left(\overline{x}\right) = \sum_{k=1}^{\infty} \int_{\Omega_k} R_p(\overline{x}) f(\overline{x}) d\mu_k\left(\overline{x}\right). \tag{4.5}$$

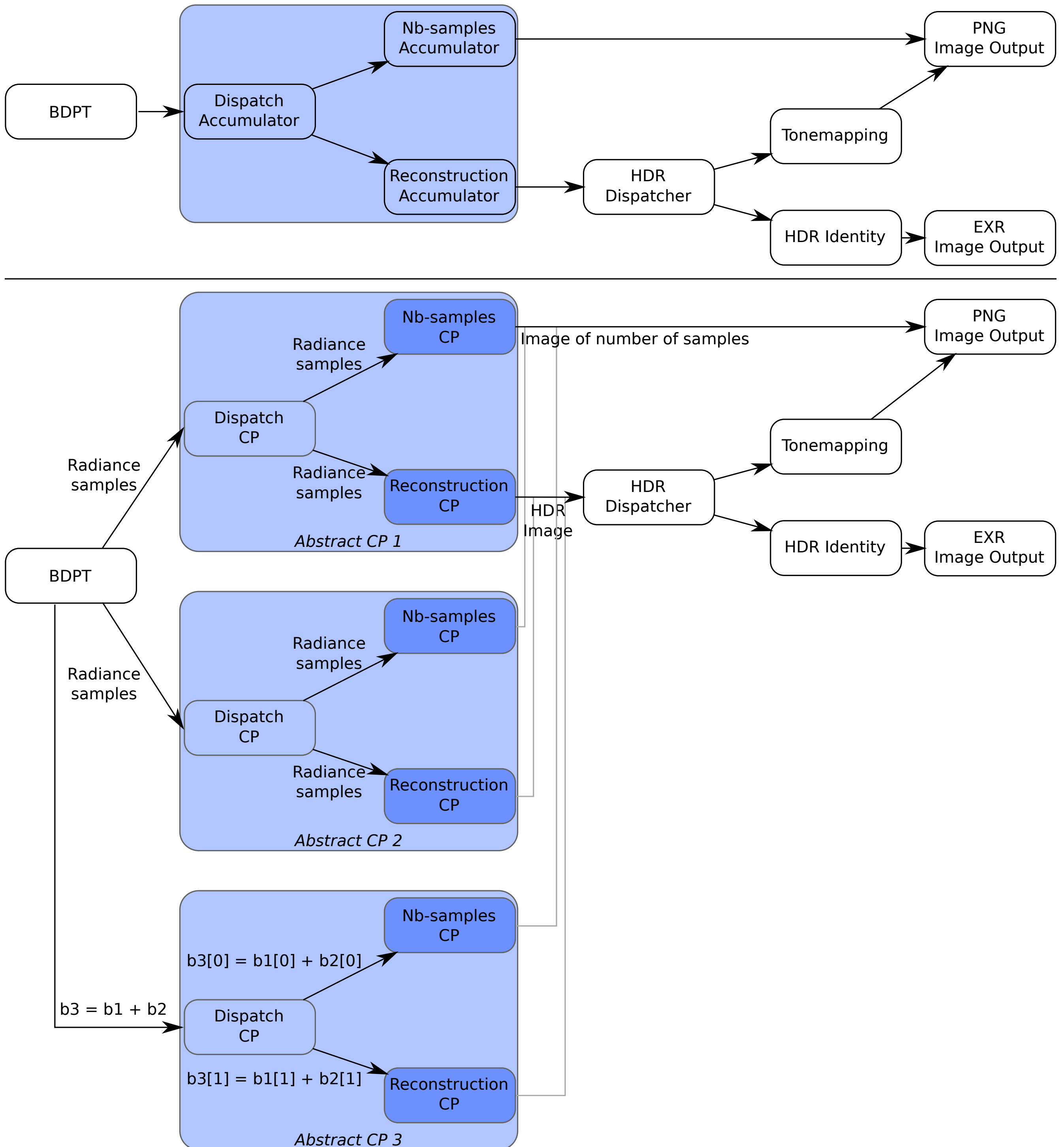


Figure 4.2: Top: Components organization as specified by the user. Bottom: Because bidirectional path-tracing (BDPT) requires three contribution processors for accumulation, it leads to three distinct CP trees, the last one being computed as the sum of the two first.

From a screen sample (x, y, u, v, t), it generates a path leaving from the camera. The key for efficiency is to reuse the path leaving from the camera that has been used for length $k-1$ as a base for length k: only a single vertex needs to be added to obtain the camera path. At each bounce, a vertex is sampled on a light, and the contribution of the complete path (camera path linked to the vertex sampled on the light) is added to the sum. Russian-roulette is used to stop expanding the path and terminate the estimation of the infinite sum.

Bidirectional path-tracing [VG94, LW93]: Path-Tracing has many qualities (simple, unbiased, parameterless), but can lead to very long rendering times to correctly compute some kind of light transports, such as the ones involving caustics. Instead of favoring exploration of the path space from the camera, bidirectional samplers are more symmetric, leaving from both the camera and the lights. This allows us to handle all kind of light transports efficiently. Bidirectional path-tracing uses such a bidirectional sampling scheme, with a specific estimator to give unbiased and low-variance results. It is presented in more details in Chapter 10, where we present an adaptation of this algorithm to be able to use at their maximum both CPUs and GPUs at the same time. Here, we briefly review the mechanism in a high-level fashion, to focus on the requirements this method puts on a software architecture meant to be as general and flexible as possible.

For a screen sample (x, y, u, v, t), a camera path $\overline{x}$ and a light path $\overline{y}$ are sampled independently, both being stopped by using Russian roulette (the camera path does not hit a light in general, and the light path does not end on the lens system as well). Then, all combinations between the light and camera paths are performed, which gives a large amount of complete paths. Most of these paths contribute to the screen sample, but when creating complete path using as partial camera path only the vertex on the lens system, the incident direction on the lens system changes, which makes that the complete path does not contribute to the (x, y) sensor coordinates. These contributions are called *light-tracing* contributions. Taking into account these contributions is very important for an efficient rendering of caustics, thus they can't simply be ignored. A camera sub-path and a light sub-path can therefore generate contributions for several sensor coordinates. A direct access to the contribution processors is therefore necessary, so that all the contributions generated during a single step can be added. Moreover, the light-tracing contributions are not accumulated the same way as normal samples, because the sensor coordinates have not been sampled the usual way. Instead, the value of a pixel is the sum of the filtered contributions, divided by the total number of light tracing contributions that have been added to the whole image. The final image is then obtained by summing the "normal" contribution processor, and the light-tracing one. This is a case where combining several contribution processors with different ways of being accumulated is necessary.

Note that in the case of bidirectional path-tracing and other algorithms which can generate contributions not accumulated the same way, any adaptive screen sampling scheme based on the radiance estimates should only consider the contributions for which the screen sampling is controlled. As this separation between contributions which can be used and others that can not is not easy to do in an automatic way, the simulators or the integrators directly push the contributions that can be used to guide screen sampling to the adaptive screen sampling scheme. Moreover, when several independent estimates

contribute to the same pixel, it it better for performances to perform a single splatting operation, but it is necessary to add each of these contributions independently to the adaptive sampling scheme, to provide more information.

Photon-mapping [Jen96]: Photon-mapping is a method which exploits the correlation present in the light field to compute the value of contributions: instead of using independent light field samples (being points on lights as in path-tracing, or optical paths leaving from the lights as in bidirectional path-tracing), an approximate light field is first computed, and then this approximation is used for all the contributions. The approximation consists in throwing N samples from lights, making them scatter in the scene, stopping scattering using Russian roulette. At each scattering event, a "photon" is stored, with energy and incident direction information. Once all the light samples have been processed, the set of all deposited photons gives a punctual information about the incident light field. This set is called *photon map*. The crucial point of this method is that this information is completely pixel-independent, and all view samples can use it, leading to increased coherence and lower computation times. As a photon map has to be built prior to rendering, a pre-processing phase is necessary before actual rendering takes place.

Once the photon map has been built, the light transport equation can be evaluated in two ways, several methods being available for the ray-integration part of the RTE ([JNSJ11]). For both ways, as the photon map already takes into account multiple scattering events from the lights, a direct-lighting-like estimation is enough, no path-tracing-like recursion is necessary. Only the specular (Dirac) components of the BSDF need to be explicitly handled by recursion. The first method, which we call *direct visualization*, evaluates the light transport equation by replacing the integral over directions by weighted sums on the photons which are nearby the evaluation point. The second way, called *final gathering*, consists in rebuilding an approximate $L_i(\mathbf{x} \to \omega_\mathbf{i})$ function from the photon map, and evaluating the integral over dimensions using this approximate function. These two ways use the mathematical framework of density estimation [Sil86] to perform interpolations in a mathematically optimal way. In the context of photon-mapping, density estimation consists in taking into account photons that are nearby the evaluation point, interpolating their data to obtain a rebuilt information at this point. As data is interpolated, this leads to blur in the images if the "nearby" criteria takes into account photons that are too far away. This blur is one of the way bias manifests in photon-mapping. On the other hand, if not enough photons are taken into account because the interpolation is done from photons in a too small region, noise occurs. Therefore, the wider the region, the more bias through blurring, the smaller, the more variance through noise, but less bias. Setting the size of the search region is therefore a bias *vs* variance compromise, and has received a lot of attention in the rendering community, as can be seen in the review in [SJ09]. This last work is another way of improving photon mapping, by working on the distribution of photons in the scene. This work is based on blue noise sampling, while other methods use advanced sampling method to add photons where they are most useful with respect to the current viewpoint [FCL05].

Note that photon-mapping is a biased method, even though the bias tends toward zero when the number of photons tends toward infinity: as the memory is finite, the number of photons is finite as well. This means that whatever the computer used, a strictly positive lower bound can be put on the bias.

4.2.2 Simulator

Light-tracing [DLW93]: A simple example of a method for which splitting screen sampling and integration is not natural is light tracing. It consists in creating partial paths from the light, and at each scattering event $\mathbf{x_k}$, creating a full path by sampling a point $\mathbf{x_l}$ on the lens system, sampled by the lens system. The full path is then $\mathbf{x_0}, \ldots, \mathbf{x_k}, \mathbf{x_l}$. The camera's optical model is used to derive the sensor coordinates from $\mathbf{x_l}$ and the incident direction $\mathbf{x_l} \rightarrow \mathbf{x_k}$. Note that $\mathbf{x_l} \rightarrow \mathbf{x_k}$ could be such that no valid coordinates exists, typically if $\mathbf{x_k}$ is not in the field of view of the camera. If the partial path is N vertices long (including the one on the light), this method generates N different contributions, at N different sensor coordinates, which are not sampled *a priori* but computed directly from the path.

Metropolis light transport [VG97]: A more elaborate example is Metropolis Light Transport and its extensions [KSKAC02]. In this method, a path is either sampled from scratch using whatever sampling method is desired (in general either similar to path-tracing or bidirectional path-tracing), or is obtained by modifying (mutating) an existing path (for instance, keeping only a part of it and sampling new points to "fill the hole", so that the new complete path is only a mutation of the previous one). A random choice is performed at each step to select between creation from scratch and mutation. The new path has a certain probability to be kept as current chain state. If the random choice leads to rejection, the old path is kept. This gives a Markov chain of paths (which can contain repetitions, due to rejection), whose stationary distribution depends on the rejection probability. When the rejection probability is well chosen, it allows us to sample paths using an arbitrary PDF over the path-space in the limit. In this method, the sensor coordinates are only known when the new full path is built. If the path has been built from scratch, it can be chosen from a screen sampler, but if it is mutated, it can either not be changed if the lens vertex and the vertex seen from the camera are not changed, or be changed in an arbitrary way otherwise.

Some other methods [CTE05, LFCD07] build on similar principles as Metropolis Light Transport, using mutations of paths, where the new sensor coordinates are sampled as part of the complete path sampling. For all these methods, the simulator semantic is particularly well suited, as screen sampling can not be separated from path integration.

Progressive photon-mapping, stochastic progressive photon-mapping [HOJ08, HJ09]: The aforementioned methods can not be naturally split in separate screen-sampling/integration part, but still estimate one sample per estimate, one step of the simulation being therefore a low-granularity step. This is not the case of progressive photon mapping and its extensions. This method basically consists in performing several photon-mapping passes, shrinking the size of the search region for photons at each pass. The basic algorithm consists in first sampling a large number of optical paths from camera, and then tracing N light paths, without storing the scattering events in a map. Instead, at each scattering event, the contributions to the eye paths generated before are computed. When a scattering event is in the search region of a vertex of a camera path, the contribution of this vertex and of the associated screen sample is updated in a progressive way. Once the N paths have been processed, new camera paths are sampled,

and the process repeats. This method requires to be able to handle several screen samples at the same time. In our software architecture, this process can be naturally described by successive steps, a step consisting in first sampling the camera paths, and then tracing the N light paths. Contrary to the simulation algorithms presented above, a step has here a large granularity: it encompasses a lot of processing. [KZ11] shows that progressive photon mapping is equivalent to accumulating several passes of standard direct-visualization photon mapping results, shrinking the size of the search region in a way derived in the paper. This method can be implemented in our system by making that a step performs one standard photon mapping pass, with a search region specified at each step according to the sequence given in the paper.

Progressive photon-mapping and its extensions are consistent estimators, because this time the bias tends toward zero when time tends toward infinity. Therefore, no strictly positive lower bound can be put on the bias given the computer that will be used to execute the algorithm.

4.2.3 Adaptive screen sampling

Two main kind of methods can be distinguished for adaptive screen sampling based on radiance samples: methods which rely on an information-theory measure of the quality of a pixel, and methods which rely on the statistical nature of the estimation, and use it to derive quantitative information about the committed error.

Information-theory approaches: Information theory is the mathematical framework to quantitatively measure quality of elements, based on *entropy*. This approach has led to many adaptive sampling algorithms, which basically add samples that contribute to a pixel while its quality measure is not good enough. In general, these algorithms do not make any difference between a very poor pixel and a quite good one: a pixel has to be over-sampled or not. This leads to over-sampling pixel sets of decreasing size, as the rendering progresses and fewer pixels do not have the required quality. A recent example of such methods, which also reviews other information-theory adaptive screen sampling methods, can be found in [XSXZ07].

Statistics-based approaches: Methods which can benefit from adaptive screen sampling are those that evaluate all pixels independently. Therefore, analysis based on each pixel contributions' distribution can give information about the state of the pixel estimation. For instance, [Pur87] and [TJ97] rely on confidence intervals to estimate when a pixel can be considered as converged. Statistics-based methods can often directly link the measured error or uncertainty to the number of samples that would be necessary to reduce this error to a given amount. Therefore, instead of binary oversampling/non-oversampling flags as in information-theoretic approaches, it is possible to assign probabilities to each pixel, so that the amount of samples computed for each pixel is proportional to the number of required samples. This way, the error tends to be made uniform over the image, leading to natural progressive rendering methods, which can be stopped at any moment while providing uniform-quality images.

Note that as described in [KA91], most adaptive sampling methods lead to bias for each pixel estimate. Most of them suffer from a “self-regression” problem as well: when each pixel probability or oversampling state is first computed, sampling occurs based on these first samples, and those pixels that are found as converged are not sampled anymore. If no samples are thrown to these pixels, they will never be updated, which can lead to large errors when the base quality or error estimate is wrong.

4.2.4 Accumulator

A standard accumulator is the splatting accumulator, presented in Section 4.1.1.1. As presented in detail in Chapter 6, the standard or weighted average estimators are not robust to statistical outliers. This problem leads to the appearance of *bright-spots*, which are very bright and visible pixels in the middle of zones with more normal values. This has led to the development of specific accumulation methods, called *sample-space* methods, such as [RW94] or [DWR10]. Chapter 6 is dedicated to presenting a new method for making the average estimator robust to outliers, which we apply to the case of contributions accumulation.

From a software architecture point of view, these sample-space methods are implemented as accumulators which act as filters, and push the samples which are judged as valid to sub-accumulators, such as the splatting one. Here, our complex contribution processor system allows us to test several sample-space bright-spot removal methods at the same time on the same radiance samples, using a dispatcher accumulator to push the samples from the simulation system to all of the bright-spot-removal accumulators at the same time. A single standard accumulator is then used to compute the final images. Figure 4.3 shows the accumulators configuration when testing two methods, one called DBOR ([DWR10]) and the other BSR (which we present in Chapter 6), and the contribution processors that are created and used when handling a single abstract contribution processor. Only the relations between accumulators require to be manually set, as they indicate what we want to do with contributions. Note that the creation of a BSR (or DBOR) contribution processor leads to the creation of an associated reconstruction contribution processor, to get separate images for each method in the end. Thus, even though the configuration of accumulators is a graphe, the resulting contribution processors are always organized in a tree.

4.2.5 HDR processor

Imaging pipeline (Chapter 8 of [PH04]): A standard imaging pipeline has to transform high-dynamic-range (HDR) images represented in a complete color-space such as CIE-XYZ to a RGB or sRGB low-dynamic-range (LDR) images which are suitable for storage in a format such as JPG or PNG. This is done in two steps: first, tone-mapping is done using models which can be either heuristics or based on models of human vision. This yields an image whose dynamic range is suitable for printing or screening. Then conversion to the final color space is done, with a special handling for colors that are not part of the gamut of this color-space (Section 1.3.1). In general, a simple clamping of the values in $[0, 1]$ is used.

HDR noise reduction: Monte-Carlo-rendered images often exhibit a high-frequency/very-low-amplitude

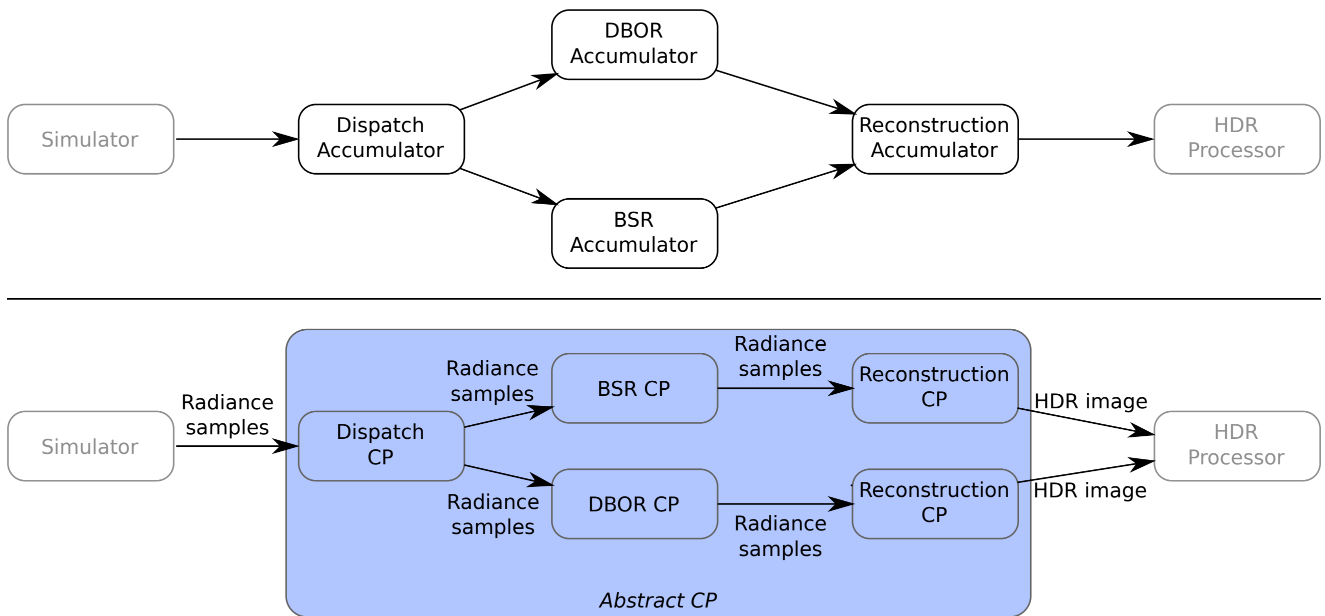


Figure 4.3: Top: organization of the accumulators to use two bright-spots removal methods at the same time: DBOR ([DWR10]) and our method, called BSR (Chapter 6). Bottom: Contribution processors created when a single abstract contribution processor is needed.

noise which is visually disturbing, even after a lot of samples have been computed. While it could be removed by adding more samples, it is sometimes possible to remove this noise in a visually non-disturbing way by using methods directly processing the image, called *image-space* methods, which can be based for instance on Gaussian filtering [TM98, Par07] or anisotropic diffusion [Wei96, McC99].

Image-space bright spot removal: Some HDR processing methods are not robust to spikes in the pixel values, which are typical of bright spots. It can therefore lead to very visible artifacts, or poor results. Although sample-space bright-spot removal methods can be implemented as accumulators to avoid them while controlling the error done by explicitly handling them, it happens that some bright spots remain. It is therefore necessary to be able to remove them as a post-process, ideally without introducing visible artifacts such as blur or deformations. We present an effective method to remove these bright spots in Chapter 7, where we also review some other existing methods.

4.3 Robust and efficient rendering

The most elegant way to have a robust and efficient rendering would be to develop a perfectly robust and efficient simulation method, which *never* gives unexpected results whatever the scene is, and which naturally performs a robust adaptive screen sampling.

However, it is in practice very difficult to build a single simulation method which ensures that no unexpected results will *never* occur whatever the scene is. First of all, it is difficult to find a single

method which can handle correctly *all* the scenes that can be created. Bidirectional methods (photon mapping, bidirectional path-tracing, *etc.*) are most suitable when it comes to caustics from local light sources, but completely drop down when distant light sources are involved. For this case, path-tracing is far more adapted, but caustics are not efficiently handled. Therefore, as no one-fits-all method exists, robust-but-not-perfect methods everywhere in the rendering architecture is a more pragmatic choice than trying to build a single simulation method with perfect results.

Therefore, additionally to as-robust-as-possible integration and adaptive screen sampling, robust and efficient methods are also necessary for accumulation and HDR image processing. This is why, in this thesis, we develop methods that are both robust and efficient, for all the parts of a rendering engine.

Statistical robustness is a very important property, but per-sample computation time robustness is important as well: no scene should lead to extremely large computation times because one of its elements can not be handled efficiently. For most elements of the scene description, this is already the case. A single element can lead to extremely large computations: participating media. As a matter of fact, sampling optical paths and computing transmittance imply integration on potentially very complex functions when handling any type of participating media representations. Finding a single representation ensuring efficient evaluation is therefore important to get this processing-time stability.

Robust integration: As said before, there is no one-fits-all method. We therefore try to improve existing methods, so that whatever the scene is, *there is a robust integration method for it*. This begins by low-level but important improvements: Chapter 5 presents a general method to improve any estimator based on multiple-importance sampling, which we apply to direction sampling for robust and efficient local sampling of optical paths. This allows us to greatly improve the robustness of any integration method which is not naturally robust (path-tracing, photon-mapping, *etc.*), with a very small per-sample computational overhead.

Robust and efficient accumulation: Even if a more robust path sampling largely improves the average behavior of a method, greatly reducing the cases of very poor behavior, variance remains, and can still lead to samples with very large values because of an inadequate importance sampling (small probability to sample a path whose contribution is large). These samples are not correctly handled by standard accumulation methods, because the standard (weighted-)average estimator is not robust to statistical outliers. These outliers lead to very bright pixels, called *bright spots*. Chapter 6 presents how to build a more robust average estimator based on online and adaptive selection of samples. Applied to the case of radiance samples accumulation, it naturally leads to a sample-space bright-spots removal method. This method has a small computational overhead while greatly improving results when large variance is frequent, additionally allowing us to know which samples have not been added at any moment, for error control for instance.

Robust and efficient HDR processing: Although delivering far better results than pure standard accumulation, our improved accumulation method can still lead to a few remaining bright-spots. For image production, these bright spots can have a large impact when the HDR processing pipeline contains non-

robust methods. Chapter 7 presents a simple and effective method to remove these bright-spots before any potentially non-robust method is used, without adding visually-disturbing artifacts such as blur or deformations.

Robust and efficient adaptive screen sampling: Focusing processing power where it is *actually* most needed can lead to substantial rendering time improvements, but an incorrect focusing can lead to zones which are never improved while more samples would be necessary. Chapter 8 presents a simple statistics-based method which always ensure that the error estimation on all the image is improved during rendering, leading to correct focusing of processing power, even in presence of statistical outliers in the samples used to estimate this error.

Computationally- and memory-efficient participating media representation: Inhomogeneous participating media can lead to extremely long computation times when no analytical free-path sampling and transmittance evaluation are available. A common solution is to discretize them in a structure which enables analytical computations. Chapter 9 presents a construction algorithm to build a discrete representation with a minimal number of nodes for arbitrary participating media, the control being done through an approximation error. This representation can then be used to perform analytical sampling and transmittance evaluations, independently of the base participating medium. As we ensure that a minimal number of nodes is used, we obtain lower memory footprints and faster evaluations than other discrete representations, without loss of generality. The construction algorithm relies on a numerical integration method which we have developed, whose goal is to provide efficient estimations and accurate precision information, without requiring any *a priori* information on the integrands. As it is built as an adaptive method, it exhibits sub-linear complexity with respect to variance.

Processing-power-efficient simulation: Some integration methods are naturally robust for a lot of scenes, but are often difficult to implement efficiently without performing any further approximations than those done when deriving the light transport equations (Chapter 1). Efficient means here that it can use as much processing power as possible. Nowadays, graphics processing units (GPUs) can be used to perform general computations, a very large processing power being available for parallel tasks. Hybrid algorithms, which use both the CPU and GPU at their maximum, are therefore suitable for efficient rendering. Chapter 10 presents “Combinatorial Bidirectional Path-tracing” (CBPT), an hybrid algorithm based on bidirectional path-tracing, which is a naturally robust integration algorithm for scenes without distant light sources. Our formulation has been designed so that no approximations are required, while allowing ten times faster computations than a pure CPU implementation.

5

Representativity for robust and adaptive multiple importance sampling

5.1 Introduction

The simulation part of a rendering engine focuses on solving either the LTE (Equation (1.55)) or the RTE (Equation (1.57)) for a given scene. For simplicity, we here focus on the LTE at a point $\mathbf{x}$:

$$L_o(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_{S^2} f_s(\mathbf{x}, \omega_i \leftrightarrow \omega_o) L_i(\mathbf{x}, \omega_i) |\mathbf{N_x} \cdot \omega_i| \, d\sigma(\omega_i) \tag{5.1}$$

where $L_o(\mathbf{x}, \omega_o)$ is the outgoing radiance at point $\mathbf{x}$ along direction ω_o, $L_e(\mathbf{x}, \omega_o)$ is the self-emitted radiance, $L_i(\mathbf{x}, \omega_i)$ is the incoming radiance, $f_s(\mathbf{x}, \omega_i \leftrightarrow \omega_o)$ is the bidirectional scattering distribution function (BSDF), and $\mathbf{N_x}$ is the normal at point $\mathbf{x}$.

In the general case, no analytical solutions to this equation are known, resorting to the use of numerical integration methods. As detailed in Chapter 3, the Monte-Carlo method is widely used because it does not need any analytical property in order to converge to the correct result. For a general integrand $f(x)$ defined over a space Ω, Monte-Carlo defines for $F = \int_\Omega f(x)dx$ the following estimator:

$$F \approx < F >_{X,N} = \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)}, \tag{5.2}$$

where N is a fixed number of samples used to compute the integral and the X_is are independent and identically distributed (i.i.d.) random variables defined over Ω and whose probability distribution function (PDF) is p. $< F >_{X,N}$ is unbiased if, for each value x where $f(x) \neq 0$, $p(x) > 0$.

In its basic form and for a fixed PDF, the estimator $< F >_{X,N}$ has a standard deviation in $O(N^{-1/2})$. When attempting at reducing the variance while sticking to the basic estimator, most of the work is done on the PDF. Importance sampling (Section 3.5.1) builds up on this: a PDF that better matches the

Full paper in IEEE Transactions on Visualization and Computer Graphics, co-authored by Loïc Barthe, Mathias Paulin and Pierre Poulin [PBPP11b]

integrand lowers the variance of its estimator. Each PDF specifically defined to focus on one part of the integrand leads to a *sampling strategy*.

In rendering, it is common to have one strategy to sample the light sources, which correctly matches the direct lighting part of the integrand, and one strategy to sample the BSDF, which correctly matches the glossy parts of the integrand. However building one strategy that takes into account both parts of the integrand at the same time is a difficult task, especially when no assumptions can be made on these two parts. Multiple importance sampling (MIS) [VG95] tries to lower the impact of this problem by combining several estimators, each using one strategy that correctly matches one part of the integrand.

If S strategies are available, each represented by a PDF p_i, the MIS framework defines two estimators, depending on whether one or several samples are drawn to evaluate the integral. The *one-sample* estimator:

$$< F >_{os}= w_i(X)\frac{f(X)}{c_i p_i(X)} \tag{5.3}$$

consists in first choosing a strategy p_i with a probability c_i, and then sampling it to evaluate the integral. w_i is a weighting function. The second estimator is the *multi-sample* estimator:

$$< F >_{ms}= \sum_{i=1}^{S}\left[\frac{1}{n_i}\sum_{j=1}^{n_i} w_i(X_{i,j})\frac{f(X_{i,j})}{p_i(X_{i,j})}\right], \tag{5.4}$$

where n_i is the number of samples generated using p_i. These estimators are unbiased as long as some constraints on w_i are satisfied, and any x for which $f(x) \neq 0$ can be generated by at least one PDF. Veach and Guibas [VG95] derive optimal weighting functions with respect to variance, for a *given* set of c_i or n_i values. This set of *a priori* fixed values is called *sampling configuration* throughout the rest of this chapter. These optimal weighting functions are known as the balance heuristic. In the case of multi-sample estimator, the balance heuristic is only near-optimal, and Veach and Guibas provide other heuristics that may behave better in some cases, such as the power heuristic or the maximum heuristic.

MIS does not give any hint about which sampling configuration would lead to the lowest variance for a given integrand, and thus which strategy should be preferably used. As illustrated in Figure 5.1 for rendering, different integrands require different sampling configurations in order to get an optimal estimator with respect to variance. When the same configuration is used for all these cases, the variance of the estimators would greatly vary from one case to another, meaning they are not *robust*. Finding optimal sampling configurations is a challenging problem, as shown in Section 5.3, and would require a huge amount of processing power to be solved with usual means.

The main contribution of this chapter is an approach that allows us to compute adequate sampling configurations at a negligible cost, leading to more robust MIS estimators, without introducing any bias. We develop the notion of *representativity*, which is an empirical measure of the match of a strategy with an integrand. As shown in Figure 5.2, we derive from these representativities both the probability assigned to each strategy when using a one-sample estimator, and the number of samples that should be taken from each strategy when using a multi-sample estimator.

As presented in Section 5.2, two different approaches have been used to obtain variance reduction.

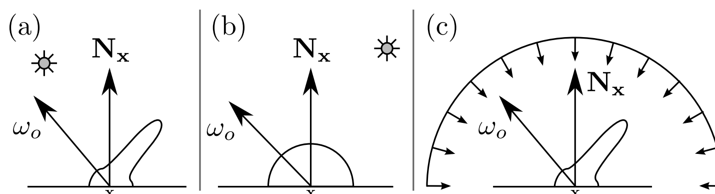


Figure 5.1: (a) and (b) Cases where sampling using the BSDF leads to high variance when estimating direct lighting; most directions generated from ω_o do not reach the light source. (c) A case where sampling from light sources fails: most points generated on large light sources such as environment lighting are not in the specular lobe associated to ω_o.

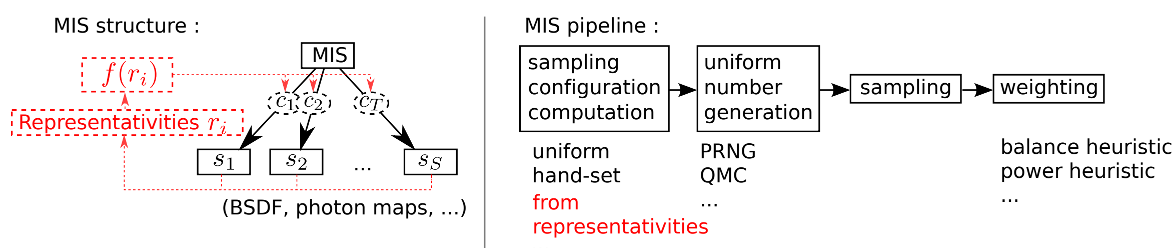


Figure 5.2: Our contribution (in dashed-red and italics) relative to MIS framework. Representativities allow us to automatically derive empirically good probabilities for each of the available strategies. The examples of importance sampling strategies are taken from the rendering domain. Our method is orthogonal to the uniform number generation methods, as well as the weighting heuristic that is used to compute the final estimate. Therefore, it provides another way to improve the robustness of the MIS estimators, while benefiting from better uniform number generation methods or better heuristics.

The first approach is to create very specific methods, where the form of the integrands and the strategies are known in advance. The second is to define general methods, that do not benefit from the optimality results provided by the MIS framework. As defined in Section 5.4, representativity is general and can be applied to any integrand and strategies. Moreover, it is designed to be used from within the MIS framework, and can be used with methods that do not change the MIS framework, such as Quasi-Monte-Carlo ones. More specifically, representativities are the results of the evaluation of a so-called *representativity function*. This representativity function has to be crafted for each strategy used by a MIS estimator. Once this is done, our method automatically computes the sampling configuration that is used by the MIS estimator. In order to apply this method to rendering, we design such functions for a strategy sampling from Ashikhmin-Shirley BSDF's, and a strategy sampling from photon maps. As our method is based on empirical models, we assess its validity in Section 5.5 by performing numerical analyses on various cases where classic MIS estimators lack robustness. Once its validity is assessed, we show in Section 5.6 that representativity-based sampling can be used in any context where several importance sampling strategies can be pertinent depending on the integrand, focusing on its potential uses in rendering. This work has been published as a full paper in the IEEE Transactions on Visualization and Computer Graphics journal [PBPP11b].

5.2 Related works

5.2.1 Importance sampling strategies for rendering

Extensive research has involved designing efficient sampling strategies for common BRDFs [LW94, LRR04], and designing BRDFs (Bidirectional Reflectance Distribution Functions) and BTDFs (Bidirectional Transmission Distribution Functions) that are well suited for importance sampling, while still providing high-quality results [AS00, WMHT07]. These strategies generate well-distributed samples where BSDF (Bidirectional Scattering Distribution Function, *i.e.* BRDF + BTDF) values are larger. Unfortunately, sampling using only the BSDF fails in situations where lighting comes from within a small solid angle, as illustrated in Figure 5.1.a and 5.1.b.

Importance sampling of environment maps has also been thoroughly investigated [ARBJ03, ODJ04]. Similarly, sampling a point on area light sources is a straightforward strategy for computing direct lighting from local light sources [PH04]. However, using only this strategy to estimate direct lighting contribution at a surface point fails when the BSDF is highly glossy and light sources are large (which is the case of environment maps), since the solid angle within which energy is scattered to the outgoing direction is very small (Figure 5.1.c).

Sampling from directional maps to better capture indirect lighting effects has been investigated by Jensen [Jen96, Jen95], and a robust method to sample directions based on particle footprints has been introduced by Hey and Purgathofer [HP02]. Pharr [Pha] uses photon maps to guide the final gathering of its improved photon mapping algorithm. In each case, they introduce a user-defined parameter that gives the probability to sample the BSDF instead of the map, and they do not deal with multiple maps.

5.2.2 General variance reduction

We are not aware of any published work focusing on the problem of automatically finding good sampling configurations. To our knowledge, only hand-set constants or uniformity are used.

Several methods other than MIS have been developed to create low-variance estimators, even for very complex integrands, such as Metropolis sampling [MRR+53] or sampling importance resampling [Tal05]. All these methods rely on pseudo-random uniform number sequences. An extensive work has been done to create uniform number sequences with very good discrepancy properties, leading to the Quasi-Monte Carlo (QMC) methods.

Metropolis sampling [MRR+53] aims to sample from any target distribution. It uses mutations to transform a base sample and probabilistically accepting it as a new base sample, or keeping the previous one if the mutated sample is rejected. Using adequate acceptance tests, the base samples obtained by this process are distributed according to the target distribution. Besides the correlation between the samples, Metropolis sampling requires an initialization phase to create the first base sample, and performs several steps to converge to the target distribution. These two tasks are prohibitively expensive when generating a small number of samples from a target distribution. Moreover, mutations must be carefully designed to get a faster convergence to the target distribution, which is a difficult task. More generally, Markov-Chain Monte-Carlo methods, to which Metropolis sampling belongs, are not adapted when taking small numbers of samples from a target distribution, because of their complexity and initialization costs.

Sampling importance resampling [Tal05] is another method to sample complex distributions by using simpler importance sampling strategies. It generates N samples from a single strategy s, and then filters the generated samples to create a distribution close to the function f we would like to sample from. If the PDF associated to s is quite different from f, the number of samples to generate using s can be large before reaching enough samples. This leads to useless computations, or increased variance when the number of samples to generate from s is fixed and too small.

In the domain of rendering, several methods have emerged, taking into account the form of the integrand given by the rendering equation [Kaj86]. Bidirectional importance sampling methods [BGH05, CJAMJ05, CAM08] have been mainly developed for direct lighting, where lighting is represented by environment maps. Recently, a similar method has been developed for indirect lighting represented by virtual point light sources [WA09]. These methods perform well for the cases they are designed for, but they are limited to the combination of only two importance sampling strategies (whether these strategies take several factors into account or not), whereas more sampling strategies could provide better results. Rousselle *et al.*[RCL+08] developed a method for sampling products of functions, but an intermediate hierarchical representation for each integrand must be built before sampling. Although handling an arbitrary number of functions in the product, the memory cost of their method is linear with the number of functions, and the efficiency of the sampling depends on the shapes of the functions.

(Randomized-)Quasi-Monte Carlo methods [Lem09] aim at replacing the pseudo-random number generators by deterministic sequences that have very good discrepancy properties, leading to a better exploration of the sample space. These methods have been used in computer graphics for a while [Kel96], with highly convincing results. Similarly to the other methods presented above, our method considers

that the uniform numbers used for sampling are given by a black-box system. Therefore, it is very easy to benefit from the QMC pattern's low discrepancy properties to further lower the variance of the estimators, at the cost of introducing bias if a non-randomized QMC sequence is used.

5.3 Variance-optimal sampling configurations

Our goal is to examine the possibility to use a formal approach in order to find variance-optimal sampling configurations.

We want to solve the following integral, using the Monte-Carlo (MC) method:

$$F = \int_{\mathcal{D}} f(x)dx. \tag{5.5}$$

The MC method evaluates F using N i.i.d. random samples $X_1, \ldots, X_N$ distributed over $\mathcal{D}$ following a PDF p. It is based on the following unbiased estimator of F:

$$< F >_{X,N} = \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)}, \tag{5.6}$$

Each tuple (f, X, N) defines an estimator, and setting one or two elements of this tuple defines a family of estimators.

When f is complex, we can use a family of estimator defined in MIS [VG95] to reduce variance:

$$< F >_{ms} = \sum_{i=1}^{S} \left[\frac{1}{n_i} \sum_{j=1}^{n_i} w_i(X_{i,j}) \frac{f(X_{i,j})}{p_i(X_{i,j})} \right] \tag{5.7}$$

where S is the number of strategies that can be used, n_i is the number of samples to generate using strategy s_i, w_i is its weighting function , p_i is its associated PDF, and each $X_{i,j}$ is a random variable distributed according to p_i, independent from the other $X_{i,j'}$, $j \neq j'$. Estimators of $< F >_{ms}$ are called *multi-sample estimators*.

As with an MC estimator, each tuple $(f, \{X_i\}, \{n_i\}, \{w_i\})$, $i \in \{1, \ldots, S\}$ defines an estimator. Optimal weighting functions $\{w_i^*\}$ with respect to the variance $V[< F >_{ms}]$ are solutions of the problem

$$\{w_i^*\} = \underset{\{w_i\}}{\operatorname{argmin}} \left(V[< F >_{ms}] \right). \tag{5.8}$$

This problem has not been solved exactly, but a very good approximate solution w_i^m is the balance heuristic [VG95]:

$$w_i^m(x) = \frac{n_i\, p_i(x)}{\sum_{k=1}^{S} n_k\, p_k(x)}. \tag{5.9}$$

To get still a better variance-wise estimator, it would be necessary to minimize with respect to $\{n_i\}$,

i.e. solve

$$\underset{\{n_i\}}{\operatorname{argmin}} \left(V \left[E_{bh,ms}[\{n_i\}] \right] \right) \tag{5.10}$$

where $E_{bh,ms}[\{n_i\}]$ is the estimator obtained by fixing the $\{n_i\}$ to the values given between brackets. This problem is a constrained discrete optimization problem, which is known to belong to the NP-complete complexity class, and thus cannot be efficiently solved [Wil03].

The family of one-sample estimators present in the MIS framework could help going further:

$$< F >_{os} = w_i(X) \frac{f(X)}{c_i p_i(X)}. \tag{5.11}$$

An optimal variance-wise weighting function is a slightly modified version of the balance heuristic w_i^m (Equation (5.9)):

$$w_i^o(x) = \frac{c_i \, p_i(x)}{\sum_{k=1}^{S} c_k \, p_k(x)}. \tag{5.12}$$

$\{w_i^o\}$ are the exact solution of the problem

$$\underset{\{w_i\}}{\operatorname{argmin}} \left(V \left[< F >_{os} \right] \right). \tag{5.13}$$

The family of one-sample estimators using the balance heuristic as weighting functions is denoted as $E_{bh,os}$.

Finding the estimator with minimal variance in $E_{bh,os}$, thus solving the problem

$$\underset{\{c_i\}}{\operatorname{argmin}} \left(V \left[E_{bh,os}[\{c_i\}] \right] \right) \tag{5.14}$$

is a continuous optimization problem, in theory easier to tackle than the minimization presented in Equation (5.10). Despite this, directly solving the problem in Equation (5.14) is not a viable option. First, it requires to compute accurate values of $V\left[E_{bh,os}[\{c_i\}]\right]$. This implies a large number of estimations of I using $E_{bh,os}[\{c_i\}_j]$, at each step j of the optimization process, which is very costly. Moreover, the average of all these estimations can already give a very good estimation of I, making the minimization useless. Furthermore, as the computation of the variance is itself an estimation with an inner variance, results obtained for a given sampling configuration could be far from the real value, hence misguiding the optimization process to miss the correct solution.

5.4 Representativity-based sampling

To render an image with photon-map-based importance sampling, we first trace photons from the light sources to build these maps. Then, we estimate the value of each pixel by tracing a number of camera paths. At each bounce of a camera path, we perform an estimation of the rendering equation using MIS. When using the one-sample estimator (Equation (5.3)), we first choose the sampling strategy according to the probability c_i of choosing each of the strategies. When using the multi-sample estimator

(Equation (5.4)), the number of directions to generate using each strategy is given by the n_i values. We generate each direction using the associated strategy, and recursively estimate the radiance arriving from this direction. The final estimation is the sum of the weighted contribution of each direction.

While weighting functions such as the balance heuristic or power heuristic are at the very end of MIS, representativities come up as the first step of the estimation, as it helps choosing amongst the available strategies (Figure 5.2). Therefore, the representativity of a strategy has to measure the appropriateness of a sampling strategy to evaluate an integrand: the more a sampling strategy reduces variance for the current integrand, the higher its representativity value is.

The sampling configuration should thus reflect the estimation of relevance by assigning probabilities that are function of the representativity of each sampling strategy for a given integrand. This implies that representativities should be unit-less comparable values, expressed here between 0 and 1. We design such functions in the context of global illumination. We want to use MIS-based estimators, with $n+1$ possible strategies: either sampling from the BSDF, or sampling from n different photon maps, where n can be arbitrarily large.

All the representativity functions that we now derive have two implicit parameters, which completely describe the integrand when the scene is fixed: the estimation point $\mathbf{x}$ and the outgoing direction ω_o.

5.4.1 BSDF-based strategy representativity

Importance sampling from BSDF is one of the most widely used strategy when simulating global illumination. For this reason, we derive a representativity function for strategies based on a BSDF.

Our representativity function for BSDFs is built upon the directionality of the BSDF, for the given outgoing direction. Indeed, a diffuse BSDF has a low directionality, all directions having the same scattering behavior. Conversely, an almost mirror-like glossy BSDF has a high directionality, since light is scattered only within a tiny cone of directions in the outgoing direction.

In our rendering engine, the BSDF model combines an Ashikhmin-Shirley anisotropic BRDF [AS00] $AS(\omega_i \to \omega_o)$ with parameters k_d and k_s, and a specular BTDF $ST(\omega_i \to \omega_o)$ with parameter k_t, with a Fresnel term $F(\omega_o)$ to weight between BRDF and BTDF:

$$\begin{aligned} f_s(\omega_i \to \omega_o) &= F(\omega_o)\, AS(\omega_i \to \omega_o) + \\ &\quad (1 - F(\omega_o))\, ST(\omega_i \to \omega_o). \end{aligned} \tag{5.15}$$

This BSDF has three components: the diffuse part does not guide more in any particular direction, the glossy part of the BRDF guides depending on the roughness terms (n_u, n_v) that are similar to Phong exponents [AS00], and the specular BTDF part guides completely in the unique contributing direction. Thus, the final representativity function is a composition of the directionality of each component.

The directionality of the diffuse part is set to a minimum value, corresponding to the uniform probability to sample any direction: $d_d = 1/(2\pi)$.

The directionality of the glossy part can be estimated by the aperture angle of the cone containing a proportion of the directions generated by importance sampling. This angle can be computed considering

that the importance sampling procedure creates directions whose angle to the perfect mirror reflection direction is decreasing as the random number u used to sample this direction increases. We define our directionality by considering the angle θ_h^n obtained for $u = 0.5$ for a given Phong exponent n, meaning that the cone contains half of the generated directions. This does not affect the sampling procedure in itself, which still considers all the contributing directions. We only use this angle for the directionality estimation. θ_h^n is obtained from the importance sampling formula:

$$\theta_h^n = \cos^{-1}\left(0.5^{\frac{1}{n+1}}\right). \tag{5.16}$$

The anisotropy is handled by taking $n = \max(n_u, n_v)$, because it is the most directional (*i.e.* narrower), and so, it makes the BRDF more representative.

As a small angle implies a high directionality, we can not directly use the computed angle, but we need to revert it, using the maximal angle that can be obtained ($\cos^{-1}(0.5) = \frac{\pi}{3}$, given by $n = 0$). We also need to ensure that the directionality is at maximum 1, with a minimal value corresponding to diffuse scattering. Combining all the terms, the directionality for the glossy part of roughness n corresponds to:

$$d_s = \frac{1}{2\pi} + \left(1 - \frac{1}{2\pi}\right)\left(\frac{\pi}{3} - \cos^{-1}\left(0.5^{\frac{1}{n+1}}\right)\right)\frac{3}{\pi}. \tag{5.17}$$

The specular BTDF is the most directional scattering behavior we can have, thus its directionality is maximal: $d_t = 1$.

For more complex BTDFs, such as the microfacet-based model of Walter *et al.*[WMHT07], similar derivations as the one used for d_s can be applied.

The final representativity function is obtained by weighting the three directionalities, according to the Fresnel term $F(\omega_o)$ and the normalized version of each component, obtained from the k_d, k_s, and k_t parameters as $\bar{k_d} = k_d/(k_d + k_s + k_t)$, respectively for $\bar{k_s}$ and $\bar{k_t}$. Taking the normalized version of each component ensures that the global albedo of the BSDF is not taken into account, as it affects only the final value of the integral but not its shape. This leads to:

$$R(\text{BSDF}) = F(\omega_o)\,(\bar{k_d}\,d_d + \bar{k_s}\,d_s) + (1 - F(\omega_o))\,\bar{k_t}\,d_t. \tag{5.18}$$

Note that representativities obtained from this function can be null only if there are no contributing directions, ensuring that no bias is added. This final representativity function meets all the requirements presented above, as representativities computed from it are always between 0 and 1, and are unit-less. Such representativity functions are called *single*.

5.4.2 Photon-map-based strategy representativity

When computing global illumination, strategies that match the incident radiance part of the integrand can greatly help reducing variance when incident light is highly non-uniform. This is most visible when caustics are present, as in this case the incident radiance term is the most important of the integrand. In our application, we choose to use photon maps to sample incident directions. More specifically, we have

several photon maps, each considering a different part of the radiance field (caustics, diffuse indirect lighting, *etc.*). Each of these maps can be sampled, leading to one strategy per map. Consequently, we derive a representativity function that can be used for all these map-based strategies.

Photons that are stored in the sampled maps provide a flux estimation whose value is not limited to the range $[0, 1]$. Moreover, it is not a good absolute measure of interest, as the flux depends on the light sources intensities. However, these flux values can be compared between photon maps, to help choosing amongst maps. We thus introduce a two-level representativity function for each photon map: the first-level representativity function helps choosing between sampling the BSDF term or sampling the incident radiance term. The second-level representativity function helps choosing one particular photon map amongst the available photon maps, and can therefore use all the available physical data.

Representativity function construction:
The representativities obtained from the first-level representativity function have to be unit-less and contained between 0 and 1 in order to be comparable with the BSDF representativity. Once computed for each photon map, we combine the first-level representativities to obtain the representativity of all the photon maps at once, gathered in a group. We call it *group representativity*.

The particular first-level representativity function that we now derive has the advantage of being fully and efficiently precomputable. For each map strategy, its first-level representativity is computed from the photons densities. We build an SAH-based kd-tree [PH04] from the photons in the associated map, with a given maximum number of photons N_{p_max} per leaf. For each leaf l, we estimate its density by computing the ratio $d(l) = n_p(l)/SA(l)$, where $n_p(l)$ is the number of photons in the leaf, and $SA(l)$ is the surface area of the bounding box of the leaf. We use the surface area because photons in the map are distributed on surfaces, and thus we want to keep comparable units (number of points over area). This per-leaf density is then converted to a representativity $r(l)$ by switching to a global probability model based on a Gaussian distribution. We avoid the use of $r(l) = d(l)/\max_{l'}(d(l'))$ in order to be robust to very high densities caused by one very small leaf containing photons.

The final leaf representativity is the value of the cumulative distribution function of the global Gaussian distribution:

$$r(l) = \mathrm{P}(X \leq d(l)), \quad X \sim \mathcal{N}(\mu, \sigma). \tag{5.19}$$

The average μ of the global Gaussian distribution is taken as the average density of the non-empty leaves (denoted as μ_d). Its standard deviation σ is computed from the standard deviation of the non-empty leaves densities (denoted as σ_d) and μ_d:

$$\sigma = \min\left(\sigma_d, \frac{\mu_d}{2}\right). \tag{5.20}$$

This clamping of the standard deviation eliminates non-negligible representativities for large leaves (in terms of surface area) with very few photons in it, whereas they would not be representative at all. Note that this first-level representativity function can be used for any map- or cache-based strategy, by replacing photons by the adequate term in the description above.

The group representativity of g is then defined as the maximum of each first-level representativity in

the group:

$$GR(g) = \max_{s \in g}(r(s)) \tag{5.21}$$

where $r(s)$ is the representativity of the leaf containing the estimation point $\mathbf{x}$ in map $m(s)$ associated to strategy s, or 0 if $\mathbf{x}$ is not contained in $m(s)$. The average or any other combination of the first-level representativities could also be used to compute the group representativity. We choose the maximum to be conservative and to avoid missing a probabilistically very good strategy even though the others in the group are not adapted at all, and thus have very low first-level representativities.

Choosing amongst several photon maps is done thanks to the second-level representativity function. Representativities obtained from this function are called *local representativity*, as opposed to the group representativity. We now derive such a local representativity function for a photon map m. We use the photons in leaf l containing $\mathbf{x}$, and define the local representativity as the average of the potential contribution of each photon p:

$$LR_g(m) = \frac{1}{n_r(l)} \sum_{\mathrm{p} \in l} [f_s(\mathbf{x}, \omega_i(\mathrm{p}) \to \omega_o)\, w(\mathrm{p})\, k(\mathbf{x}, \mathrm{pos}(\mathrm{p}))] \tag{5.22}$$

where $\omega_i(\mathrm{p})$ is the photon's incident direction, $w(\mathrm{p})$ is the photon's weight, and $k(\mathbf{x}, \mathrm{pos}(\mathrm{p}))$ is a kernel value based on the distance between the photon's position and the estimation point $\mathbf{x}$.

Representativity function usage:
When estimating an integral at $\mathbf{x}$ in a scene, we find the leaf containing $\mathbf{x}$ in the map associated with the strategy. This corresponds to descend in the kd-tree. The first-level representativity is the leaf's representativity, as defined by Equation (5.19). If point $\mathbf{x}$ is outside the kd-tree's global bounding box, the first-level representativity of the strategy using this map is set to 0. This computation does not add noticeable overhead compared to user-defined sampling configurations. The final group representativity is the combination of each strategy's first-level representativity, using Equation (5.21). The local representativity is obtained for each map by using Equation (5.22).

At this moment, one could argue that we have one parameter in the first-level representativity function we have designed: the maximum number of records per leaf N_{p_max}. However there are major differences between a user-defined sampling configuration and this parameter. First, N_{p_max} does not vary within a scene, and in practice, it does not vary between scenes either, but it is affected by the number of photons in a map. N_{p_max} is a tradeoff between the resolution of the representativities over the scene on one hand, and the accuracy of density estimation on the other hand. A larger N_{p_max} value leads to larger leaves, thus less-varying representativities. The more points there are, the more accurate density estimation is for uniform zones, but it does not adapt well to rapid density variations, typical of caustic effects for instance. In all our tests, N_{p_max} has been taken as the minimum of $n_p(m)/10000$ and 100 ($n_p(m)$ being the number of photons in map m), without any special tuning, the range of values producing good results being quite large in practice.

5.4.3 Sampling configurations from representativities

Grouping the photon maps can be generalized: all strategies which rely on absolute values should be clustered together, first- and second-level representativity functions being created for them. This leads to situation similar to the one depicted in Figure 5.3. Different sets of strategies can be created. The first one, $\mathcal{S}$, contains all the strategies whose representativity function is single, as the BSDF sampling strategy. The set $\mathcal{G}$ contains all the groups created for two-level representativity functions. When using photon maps, there is one such group. If using photon maps and radiance cache, $\mathcal{G}$ would contain two groups.

We now consider a fixed integrand. Once we have computed the representativities $R(s)$ for the strategies in set $\mathcal{S}$, the group representativities $GR(g)$ for the set $\mathcal{G}$ of groups g, and the local representativities for the strategies in the groups $LR_g(s)$, we can compute the strategy sampling probability $p(s)$ for all these strategies.

Letting

$$\begin{aligned} \text{norm} &= \sum_{g\in\mathcal{G}} GR(g) + \sum_{s\in\mathcal{S}} R(s) \\ \bar{GR}(g) &= GR(g)/\text{norm} \\ \bar{LR}_g(s) &= \frac{LR_g(s)}{\sum_{s'\in g} LR_g(s')} \end{aligned}$$

the final probability $p(s)$ of strategy s belonging to group g is finally given by:

$$p(s) = \begin{cases} R(s)/\text{norm} & \text{if } s \in \mathcal{S} \\ \bar{GR}(g) \times \bar{LR}_g(s) & \text{otherwise.} \end{cases} \tag{5.23}$$

As an example, consider the situation depicted in Figure 5.3. For this situation, letting $\text{norm} = GR(g_1) + GR(g_2) + R(s_1)$, probabilities are

$$p(s_1) = \frac{R(s_1)}{\text{norm}}$$

$$p(s_2) = \underbrace{\frac{GR(g_1)}{\text{norm}}}_{\bar{GR}(g_1)} \times \underbrace{\frac{LR_{g_1}(s_2)}{LR_{g_1}(s_2) + LR_{g_1}(s_3)}}_{\bar{LR}_{g_1}(s_2)}$$

and $p(s_3)$, $p(s_4)$, and $p(s_5)$ have a similar expression as $p(s_2)$.

These probabilities can be directly used with the one-sample estimator, keeping it unbiased as long as the representativity of a strategy is not 0 if this strategy can generate at least one contributing sample. In the case of a multi-sample estimator, a sufficient but not required way to ensure unbiasedness is to have a special strategy s_c (c for *complete*) that can generate any such sample, and ensure that the number of samples n_c assigned to s_c is at least one. In the context of rendering, the BSDF sampling strategy is a very good candidate for being a complete strategy. Nevertheless, we must also ensure that the distribution of n_i still follows as much as possible the probability distribution given by all p_i when

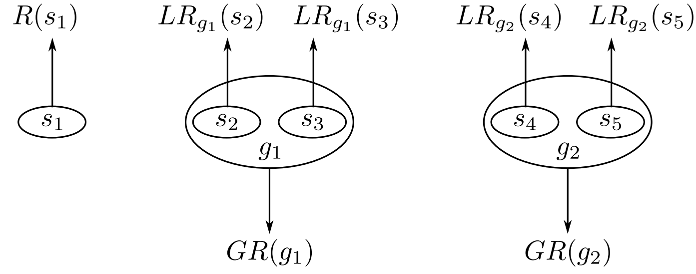


Figure 5.3: Example of a situation where a total of five sampling strategies are available, with one strategy whose representativity function is single (s_1), and two groups, each containing two strategies with two-levels representativity functions (s_2 and s_3 in g_1, s_4 and s_5 in g_2).

assigning systematically at least one sample to s_c. This implies changing the probabilities of strategy s_c (originally given by p_c) and of all other strategies p_i, giving new probabilities p_i^t, (t for *temporary*) to maintain the expected value $E[n_i] = p_i \times N$ for each strategy:

$$p_c^t = \max\left(\frac{(p_c \times N) - 1}{N - 1}, 0\right) \tag{5.24}$$

$$p_i^t = \frac{p_i \times N}{N - 1} \quad \text{if } s_i \neq s_c. \tag{5.25}$$

As is, $p_c^t + \sum_i p_i^t$ can be larger than 1.0 if $((p_c \times N) - 1)/(N - 1)) < 0$. We thus normalize the probabilities, leading to the final probabilities actually used to compute the number of samples assigned to each strategy:

$$p_c^f = \frac{p_c^t}{p_c^t + \sum_i p_i^t} \text{ and } p_i^f = \frac{p_i^t}{p_c^t + \sum_i p_i^t}. \tag{5.26}$$

To obtain each n_i (including n_c) while ensuring unbiasedness, we start by setting $n_c = 1$ and all other n_i to 0. We then sample $N - 1$ times the probability distribution defined by all p_i^f (including p_c^f), selecting each time a strategy s_i. Each time a strategy s_i is chosen, its associated n_i is increased by one.

5.4.4 General hints for defining representativities

Similarly to Metropolis mutations [MRR+53], a representativity function is an observational model, whose quality affects the rate of convergence of estimators. Ideal representativity functions should have the two following properties:

- be proportional to the relevance of information locally available for the strategy,
- be computed using only data from the strategy or the group of strategies it represents. The normalization is the only operation that considers all strategies at once.

For strategies with two-level representativity functions, second-level representativity functions should use as much information as possible to favor strategies that are better than other strategies within the same group.

5.5 Numerical analyses

We performed numerical analyses to assess the robustness brought by our method. For a number of very different cases, we compare the behavior of estimators obtained with our method to static sampling configurations. These cases are specifically designed to cover a wide range of common situations in rendering.

We used the photon map guided path-tracing system described in Section 5.6.2 to perform the tests, because its unbiasedness ensures that tests based on reference averages criteria are meaningful, such as the mean square error (MSE). All the estimators use the balance heuristic to obtain the final estimation value for a sample. Three strategies are available in our test implementation: sampling using a BSDF, sampling using a diffuse indirect map, and sampling using a map for specular paths.

Each optical situation leads to a different integrand to evaluate. For each integrand, 11 estimators have been considered: 10 test estimators using an increasing probability ρ_b of sampling a BSDF, and our automatic estimator. The test estimators we have chosen allow us to cover a wide range of possible sampling configurations, including the uniform one recommended by Veach and Guibas [VG95]. Each estimator can be near-optimal for an integrand, but behave poorly on others. For each integrand, we can then compare the best MIS estimator with our adaptive estimator.

For the test sampling configurations, the probabilities ρ_b to sample according to the BSDF range from 0.1 to 1.0 by steps of 0.1. The map-based strategies probabilities are computed as follows: if both maps are present, it is $(1 - \rho_b)/2$ for each one, otherwise it is $(1 - \rho_b)$ for the available map. If no maps are present at point $\mathbf{x}$, we set $\rho_b = 1$. Note that this is already a sort of adaptation to local estimation, but it is simple enough to be implemented in a basic photon-map-based path-tracer. The uniform sampling configuration is closely approximated by the case $\rho_b = 0.3$ when both maps are present, and is represented exactly by $\rho_b = 0.5$ when only one map is present.

Figures 5.5 to 5.10 present the numerical results we use to evaluate the efficiency of our method. Each curve in these figures corresponds to the MSE of an estimator for an increasing number of samples, from 2 to 1024, used to estimate the final value. Let E_n be an estimator using n samples, $\Theta(E_n)$ the random variable associated to the luminance of an estimation made by E_n, and $\hat{\theta}$ the luminance of the reference value computed with path-tracing. The MSE of E_n is computed as

$$\mathrm{MSE}(E_n) = V\left[\Theta(E_n) - \hat{\theta}\right]. \tag{5.27}$$

As our test scenes exhibit only one main color, luminance can be safely used. Nevertheless, extending this metric to color samples is straightforward.

In each case, the reference value has been computed as the average of 16 unbiased estimations computed with 2^{16} samples each, using a photon-map guided path-tracer with good sampling probabilities for each specific integrand, hand-tuned to reduce the variance of the final estimate. To avoid numerical instability, we did not directly use a 2^{20} samples estimation. The effective value of $\mathrm{MSE}(E_n)$ has been estimated by running m times E_n and computing the variance using the numerically-stable Knuth's algorithm [Knu98]. To avoid too long computations for no practical gain in precision, m is decreasing as

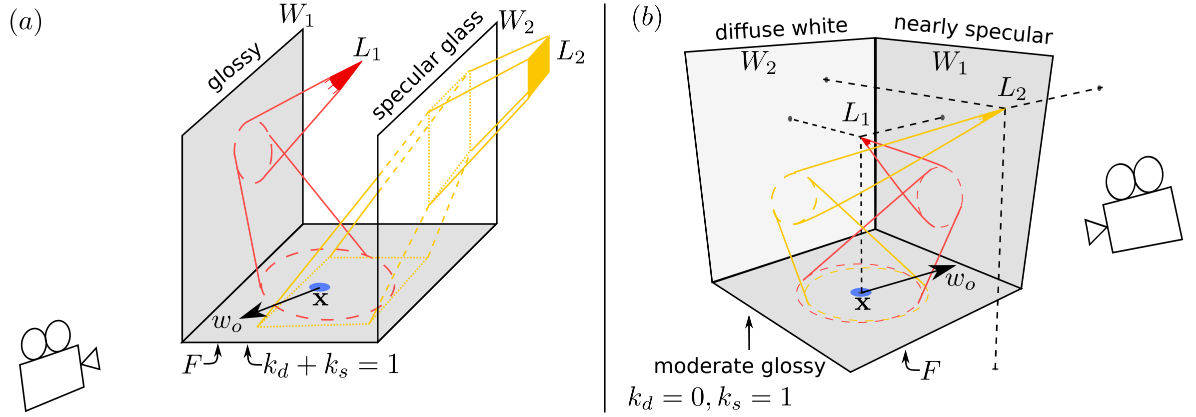


Figure 5.4: The configuration of the two simple test scenes. Light sources are in red and yellow, the blue circle at the middle of the floor indicates the estimation point.

n is increasing, since the variance of the sequence of estimators (E_n) should decrease as n increases.

5.5.1 Simple scenes

Figure 5.4 presents the simple scenes that were used to study the behavior of estimators obtained by using our method. We made this study for different integrands in well-controlled conditions and at a location where the estimation of the variance can be computed with high accuracy. To achieve this, we estimate the LTE (Equation (5.1)) with at maximum one indirect bounce. These scenes have been constructed to have no direct illumination at the estimation point, and analytical solutions for indirect bounces. Even if very simple, these scenes can lead to arbitrary high variance when using an inadequate sampling configuration.

Scene (5.4, *a*) description: L_1 is a spotlight and L_2 is an area light source. W_1 is a highly glossy wall, and W_2 is a perfectly specular glass wall. Both walls scatter light coming from respectively L_1 and L_2 toward the floor F. The floor's material is described by two parameters (k_d, k_s), with $k_d + k_s = 1$. k_d is the coefficient for the diffuse part, and k_s is the one for the glossy part. In the BRDF used for this scene, the glossy part is nearly perfect specular. The camera's position has been chosen so that the k_s part does not scatter any light directly to the camera through the observed point on floor F, all the energy coming from the k_d part. Increasing k_s leads to higher variance when using the BSDF strategy with a high probability. This variance can be made arbitrarily high for pure path-tracing. Note that in scene (5.4, *a*), for each configuration each light source power has been adapted to keep final values in the same order of magnitude (between 0.1 and 1.0).

Scene (5.4, *b*) description: It is composed of two spotlights L_1 and L_2, two walls and a floor. W_1 is a nearly perfect specular reflective wall, W_2 is made of a diffuse white material, and F is a moderately glossy floor (no diffuse component, *i.e.* $k_d = 0$ and $k_s = 1$). F is not highly specular so that directions

generated using photons coming from W_2 can have a non-null contribution. In this scene, all the light coming from L_1 does not contribute to the final value, since F's material does not scatter energy to the camera's direction. Thus, only a part of the light coming from L_2 and scattered by W_2 contributes to the final value. Here, the map-based strategy can lead to a high variance if the scattered energy coming from L_1 is much larger than the one coming from L_2. The higher the difference, the higher the variance, until reaching an upper bound given by the probability c_m to use the diffuse indirect map strategy. As a matter of fact, using pure BSDF-based path-tracing on this scene results in low variance. Provided there is a large difference of scattered energy in favor of L_1, the variance of the final estimation can be made arbitrarily high by increasing c_m. Thus, poor sampling configurations probabilities can lead to arbitrary high variance.

Tests setup: Each of the two scenes has parameters that can be set to make some estimators exhibit arbitrary high variance. The MSE (Equation (5.27)) of the LTE estimation has been computed for different values of these parameters. Figure 5.5 presents the results obtained for one-sample estimators, and Figure 5.6 the results obtained for multi-sample estimators. For scene (5.4, *a*), the k_s coefficient of the floor has been changed ($k_s = 0.0, 0.4, 0.8$) leading to a rapid increase in variance for "BSDF-oriented" strategies. For scene (5.4, *b*), the emitted intensity of L_2 is increased ($L_2 = 1, 100, 10000$, for each component of the spectrum). Note that in scene (5.4, *b*), the emitted spectrum of L_1 is 100 in all cases. For each scene variation, multi-sample estimators are obtained by the method described in Section 5.4.3, with N (the total number of samples) set to 16.

Discussion: We explicitly compute only the lighting caused by one indirect bounce, and the LTE restricted to direct lighting can be solved analytically thanks to the presence of Dirac functions, brought either by the spotlight, or by the specular transmission. This allows us to compute a reliable estimation of each estimator's variance, as there is only one LTE solution to compute using an MC estimator, at the point seen by the camera. This is this estimator's variance that we estimate here.

These simple scenes lead to very high variance when a non-adapted sampling configuration is used, and in each case the optimal sampling configuration is different. Figures 5.5 and 5.6 show that estimators with test sampling configurations have important variations in variance, meaning that they are not robust. Meanwhile, the sampling configurations obtained by using our representativity method leads to estimators with a low variance in every situation. The results confirm what was speculated above, and assess that estimators obtained using our method are robust. As a matter of fact, even if not the best, there are no estimators that behave consistently better than our estimator in all the cases.

When using one of these test configurations for a whole scene, there would be pixels with very low variance, but also pixels with very high variance. Using our method would lead to homogeneous results, with a rather low variance each time. This automatic robustness is one of the key advantage brought by our method. To further test the robustness of our method in very difficult cases, Figure 5.7 shows the MSE of the estimators obtained when increasing c_m in scene (5.4, *b*), with L_2's emitted intensity set to 10000. We can see that our method automatically leads to estimators whose variance does not vary much over the scenes, additionally keeping it low.

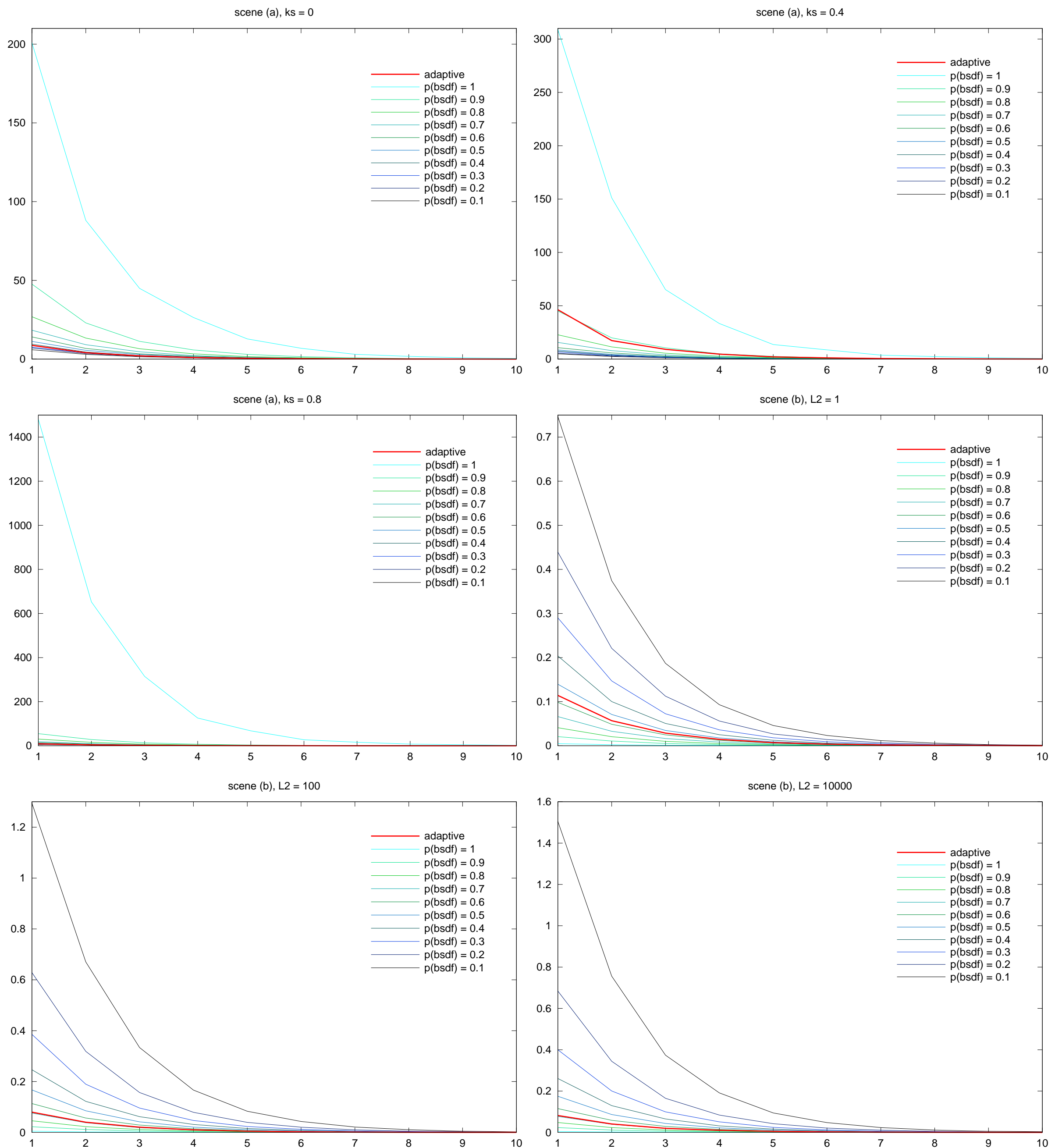


Figure 5.5: MSE of one-sample estimators, with a number of test configurations estimators, and the estimator obtained by using our representativity method in bold red. Results for scenes (5.4, *a*) and (5.4, *b*) are presented respectively in the top and bottom row. The x axis corresponds to the number of samples generated to perform one estimation of the LTE, from 2^1 to 2^{10}. The important thing to note is that no sampling configuration has consistently a better variance than ours. There are better configurations for each case, but these more adapted configurations change for every test.

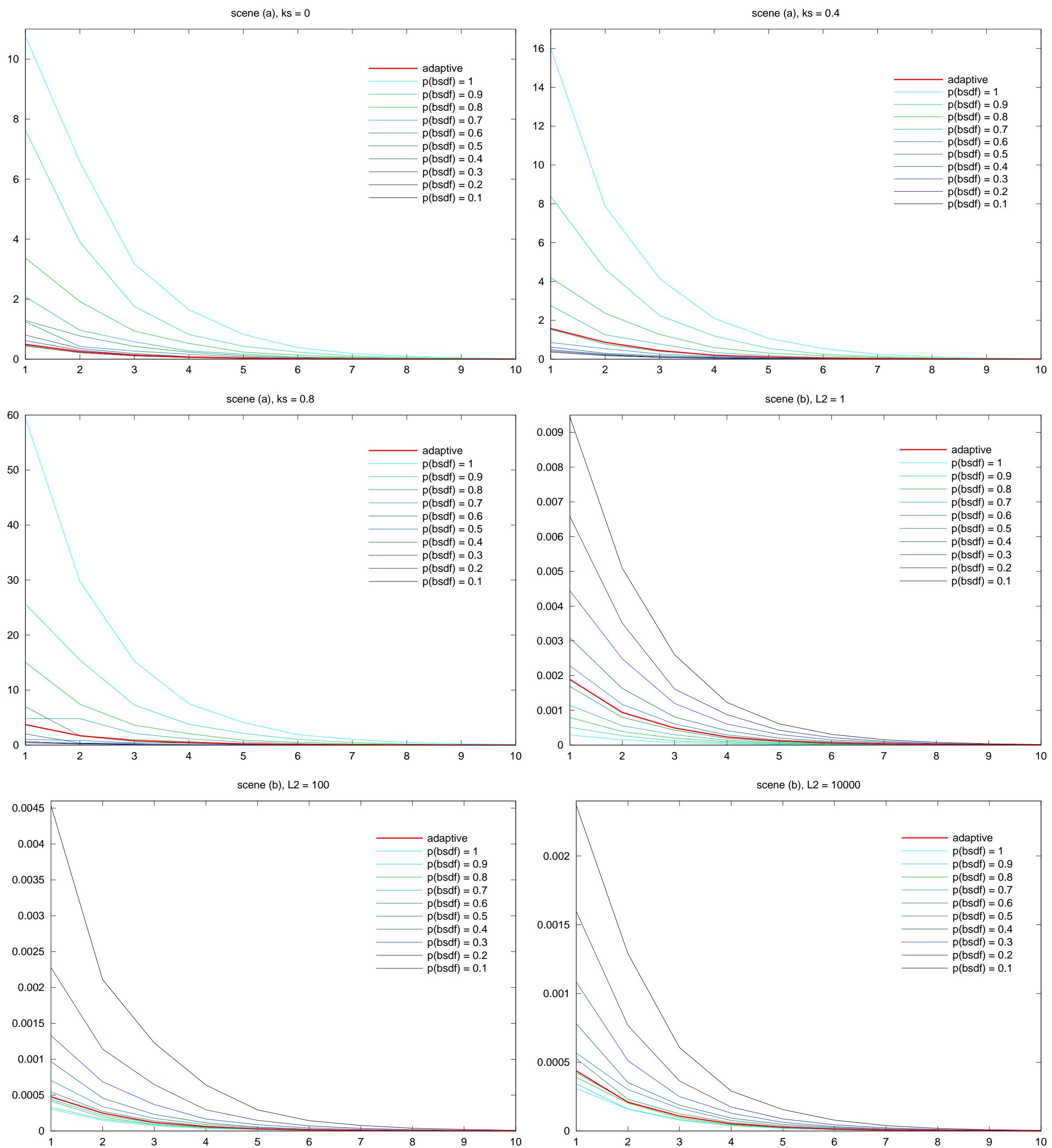


Figure 5.6: MSE of multi-sample estimators, with the same conditions and caption as in Figure 5.5.

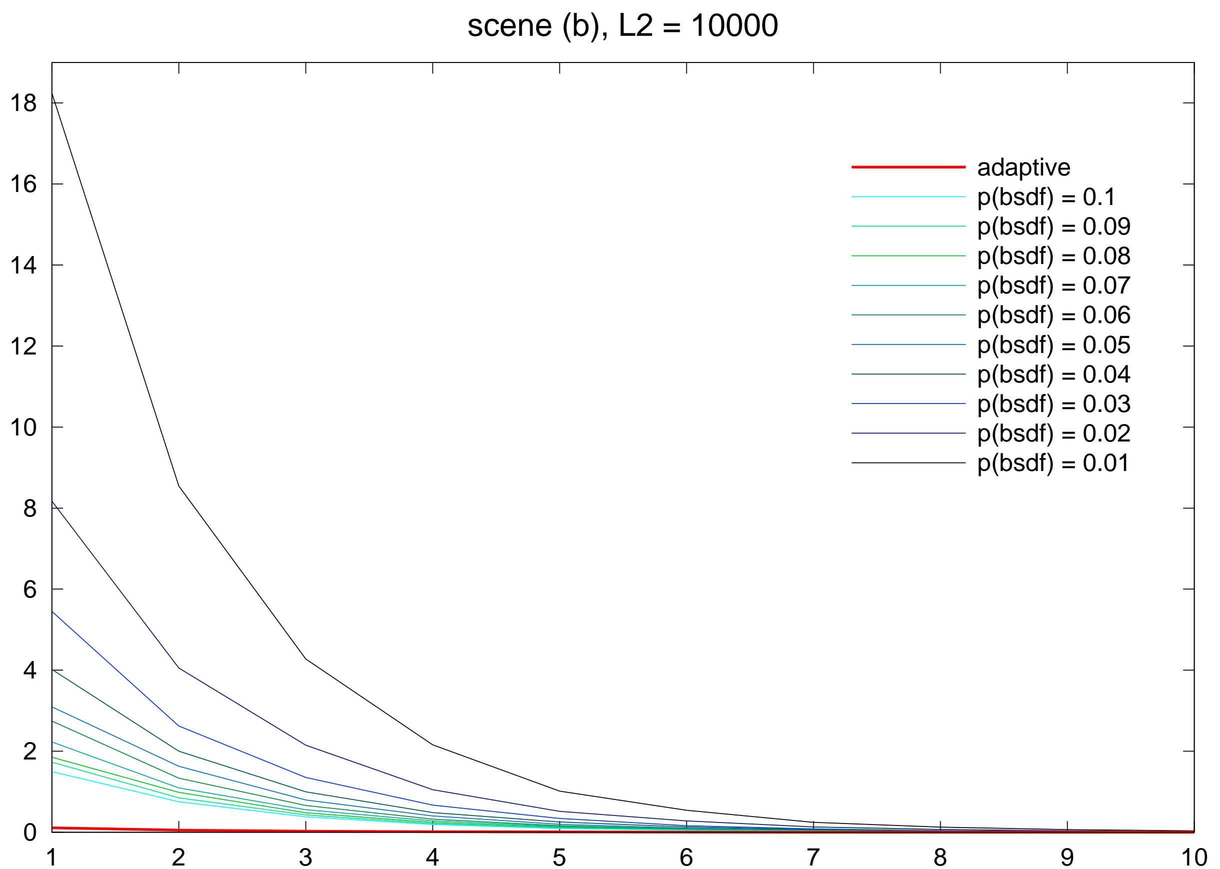


Figure 5.7: MSE of one-sample estimators with very low c_m probabilities (in $[0.01, 0.10]$), and MSE of our estimator in bold red. Same details as in the caption of Figure 5.5.

5.5.2 Chains of estimators

For a more complete study, path-tracing-like estimations have been performed on specific pixels of one scene, shown in Figure 5.8. This scene features many different optical configurations, involving specular and glossy caustics, diffuse and specular scatterings, strong indirect illumination, etc. Each test pixel features one (and sometimes more) specific lighting situation, thus leading to very different integrands at each step of the path-tracing algorithm. The goal is to examine the variance of the estimation for these pixels when using chains of estimators derived from our method, compared to chains of test estimators (each c_i being the same for all the estimators in a chain). Even if the number of test pixels can seem low, the results are representative of most lighting situations in any scene, and allow for careful and complete study. An additional series of tests over more than 6000 random pixels confirms the conclusions drawn from our chosen pixels.

Each test pixel has been chosen carefully, in order to control the lighting situations and to get meaningful results:

- Location 1 is a very hard case. At the first bounce (at the point seen by the camera), the diffuse BSDF scatters light coming from all directions, but incident lighting is strongly directional: it comes from the left (ring's caustic). For a direction sampler, it should sample the diffuse indirect map at the first bounce, and then the BSDF at the second bounce, since the ring's BSDF is highly glossy.

- Location 2 is a similar case, but with much lower caustic intensity, thus sampling the BSDF and sampling the indirect diffuse map are both a good choice, each one sampling different effects.

- Location 3 features a highly glossy reflection with a low diffuse part, with strong indirect illumination provided by the ring's caustic. This situation can lead to arbitrary high variance for pure BSDF-based path-tracing when no incident lighting comes from the reflection directions. As a matter of fact, the lower the diffuse part is relatively to the glossy part, the less it is sampled, and thus the less the only contributing directions are generated.

- Location 4 is a case of diffuse surface with low frequency indirect illumination, which is encountered in nearly every scene.

- Location 5 is a similar common case of a glossy surface with low frequency indirect illumination.

- Location 6 is a trickier version of Location 1. It adds one bounce to the "optimal" sampling chain: at the first bounce, the BSDF-based strategy should be used, at the second, the diffuse-indirect-map-based strategy should be used, and at the third, the BSDF-based strategy should again be used.

- Location 7 is a classic case of specular caustic.

- Location 8 is a classic case of specular transmission, where only the BSDF-based strategy can provide contributing directions because the Fresnel term is null, thus there are no glossy reflections.

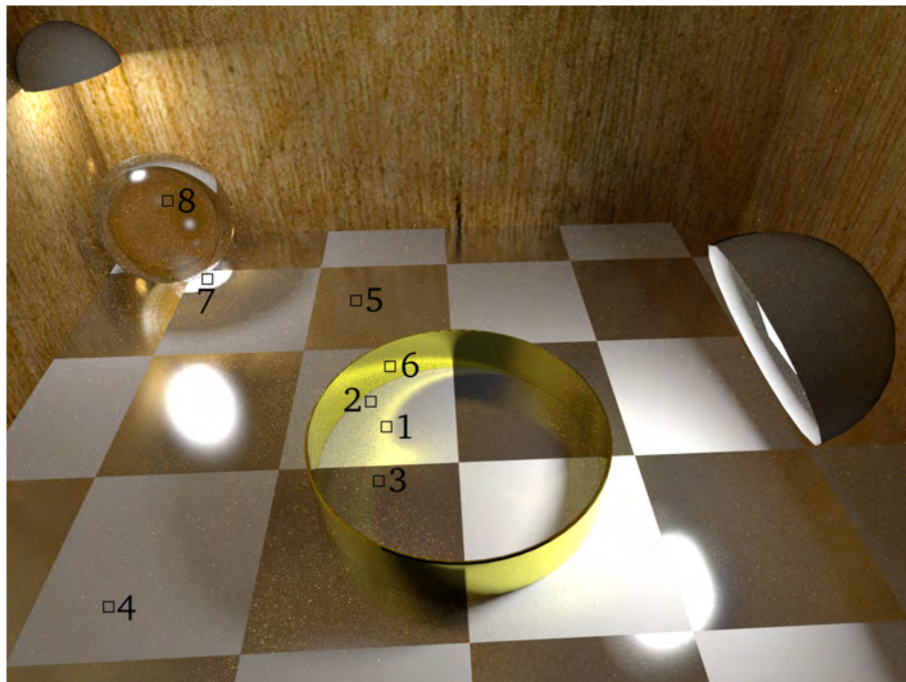


Figure 5.8: The test scene for chains of estimators, with labeled test pixels, corresponding to very different optical situations. The floor has a checkerboard pattern composed of diffuse cream tiles and glossy tiles. The ring is glossy with a large Phong-like roughness coefficient, and the sphere is a Fresnel weighted sum of a glossy reflection part and a specular transmission part.

For representativity-based sampling configurations, the two strategies using the maps are gathered in one group. The BSDF-based strategy is a single strategy using the representativity function defined in Section 5.4.1. Figure 5.9 presents the probabilities computed by our method for each of the three strategies for the observed points of the scene. This defines a single estimator, which depends on $\mathbf{x}$ and ω_o.

The goal of these test pixels is to show that for all these situations where test configurations can fail, our adaptive approach performs well. We might not be as good as the best fixed configuration for a given optical situation, but the best fixed configuration is different for each such situation, and therefore an adapted sampling configuration for one illumination situation can give very poor results in another situation, while our approach still gives a good configuration.

Discussion: Figure 5.10 shows the MSE obtained for each test pixel labeled in Figure 5.8. Our estimator gives better results than any test estimator in Location 1. The glossy caustic is well sampled by the diffuse indirect map, but using this map to sample a direction on the almost-mirror-like glossy ring leads to high variance. Conversely, pure-BSDF path-tracing (cyan curve) is well adapted for the ring, but not for the caustic. Our method favors the strategy using the map for the caustic, and the strategy using the BSDF on the ring. Location 3 is our worst case. The specular map strategy is not available as there are no specular paths in this part of the scene (they are gathered in the sphere's region), thus only the group representativity of the indirect map strategy is taken into account. Both the BSDF and indirect diffuse map strategies have high representativities, but sampling the BSDF would be the most appropriate strategy.

There are several important facts to note about these graphs:

- Our worst case (Location 3) still has a lower MSE than some test sampling configurations, even if the majority of estimators based on test configurations have a better behavior than our estimator.

- The best configurations for Location 1 are the ones with lower ρ_b, configurations with high ρ_b performing very poorly. Conversely, the best configurations for Location 3 are the ones with higher ρ_b, the other ones performing very poorly.

- On the eight test cases, our configuration is good to very good for five of them (Locations 1, 2, 4, 5, and 6), average for one of them (Location 8), and relatively poor for two of them (Locations 3 and 7).

- None of the test configurations perform well for all pixels, and none are consistently better than our representativity-based configuration.

In terms of computation time, there is no noticeable overhead compared to a path-tracer relying on photon maps to guide the sampling, whatever sampling configuration it uses (uniform or the tests ones).

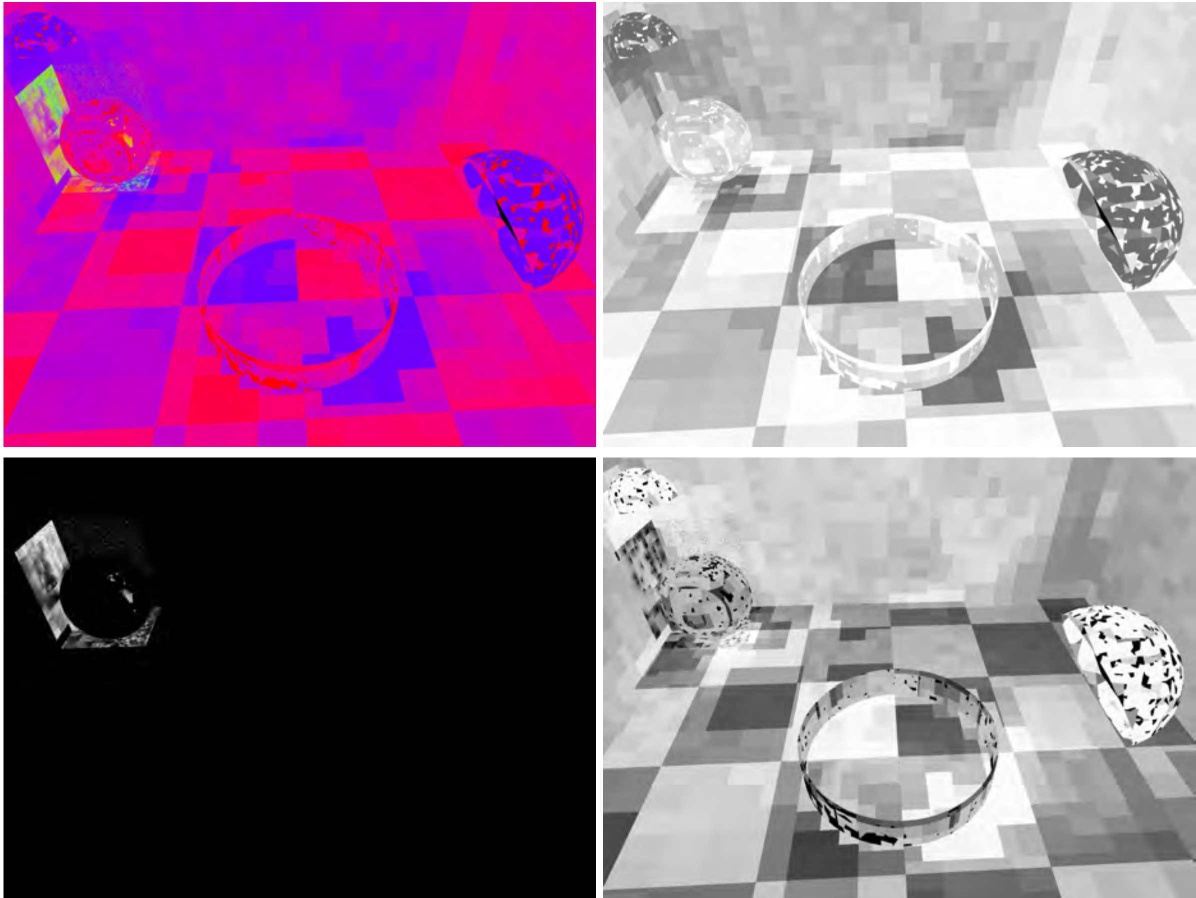


Figure 5.9: Probabilities (as pixel intensities) assigned by our method to each strategy for the visible points in the scene. Probabilities of sampling the BSDF are displayed in the top-right image, the ones for sampling the specular map in the bottom-left, and sampling the diffuse indirect map in the bottom-right. The top-left image presents a summary, encoded in the three RGB channels: R for sampling the BSDF, green for sampling the specular map, and B for sampling the diffuse indirect map. Note how sampling the BSDF is favored on the glossy tiles, sampling the diffuse indirect map is favored for the ring caustic on the diffuse tiles, and sampling the specular map is favored under the sphere and on the wall, a specular caustic being created by the right light source. The indirect map contains 1,000,000 photons, and the specular map 100,000 photons, N_{r_max} was set to 100.

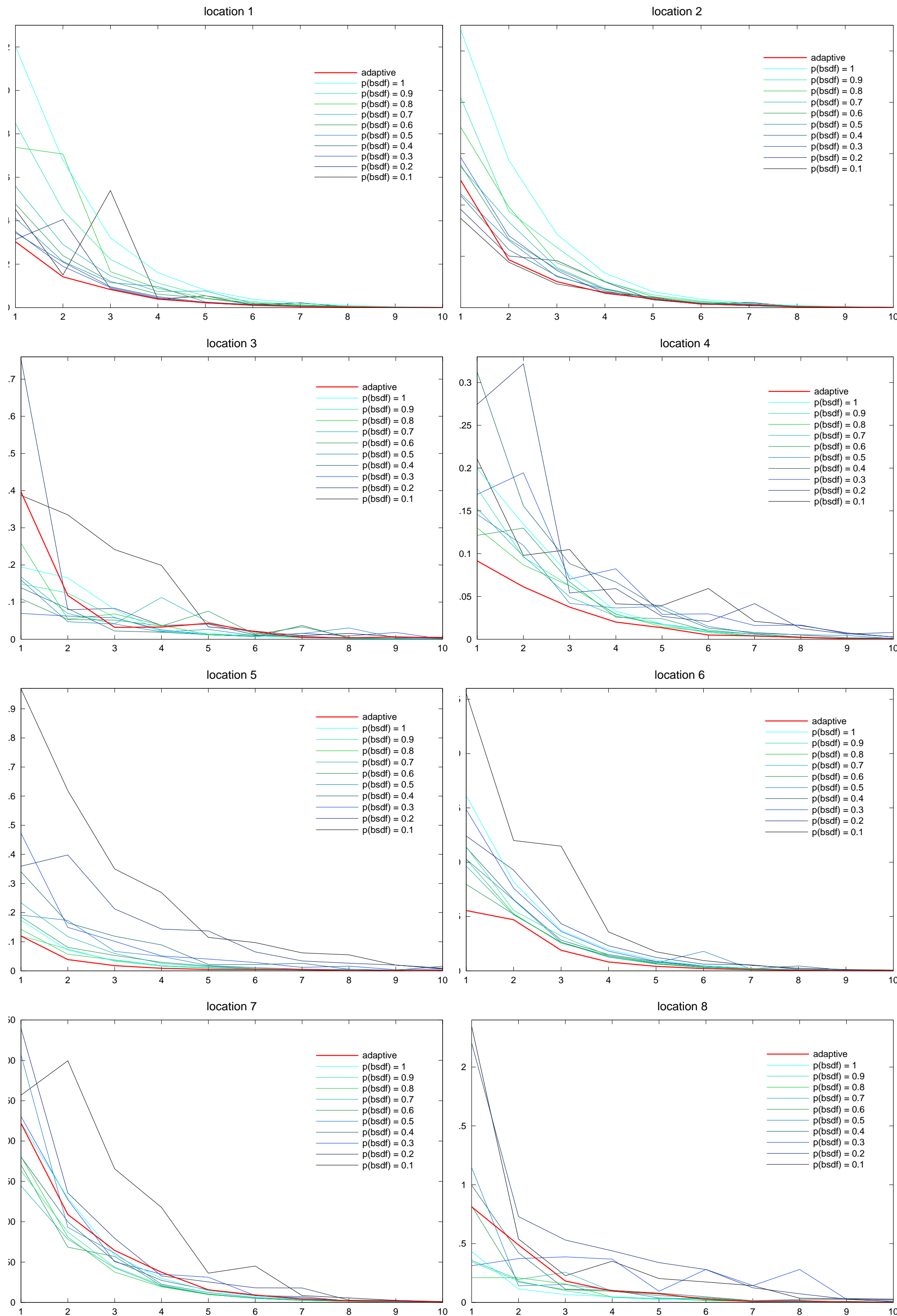


Figure 5.10: For each location of the test scene of Figure 5.8, MSE of our estimator are in red, and MSE of the other estimators are with a color gradient ranging from dark blue to cyan. Dark blue means $\rho_b = 0.1$, cyan means $\rho_b = 1.0$, that is pure BSDF-based path-tracing. The x axis corresponds to the number of samples generated to perform one estimation of the value of a pixel, from 2^1 to 2^{10}. MSE has been computed using a variable number of estimations. For power i, it is given by $\min(10^4, 2^{10-i} \times 10^3)$. The number of estimations is clamped to avoid numerical instability.

5.6 Examples of applications

5.6.1 Photon mapping final gathering

Using our method to perform final gathering is straightforward. The first-level representativities of all available maps are precomputed (we can think of different kinds of indirect maps, for instance separating glossy and non-glossy scatterings). At each gathering point, local representativities of each strategy are computed, and the base probability p_i for each strategy s_i is obtained using Equation (5.23).

We then use the multi-sample estimator with N samples, by distributing the N gathering rays over the strategies, as described in Section 5.4.3, assigning at least one sample to the BSDF sampling strategy to conservatively ensure unbiasedness, as we know that this strategy can always sample any contributing direction.

5.6.2 Photon-map guided path-tracing

Path-tracing with next event estimation can be interpreted as a recursive process rather than an integration method over a path space, where we evaluate at each level of recursion the direct and indirect parts of the rendering equation, using only one sample for the indirect part. In this formulation, the estimation of the indirect part can be computed by the one-sample estimator.

Using our representativity-based model, we compute the probabilities p_i to choose any of the available strategies. As the BSDF representativity, defined in Section 5.4.1, is non-zero for any non-completely absorbing BSDF, unbiasedness is ensured.

5.6.3 Direct lighting in highly occluded environment

Dedicated techniques exist to handle this problem [DWB$^+$06]. However, it is possible to use the same representativity functions as the one defined for photon maps in Section 5.4.2 to construct robust estimators for the direct lighting part of the rendering equation. These representativities would allow us to sample the most appropriate light sources even when there is a large number L of small light sources in a highly occluded environment. These estimators rely on $L + 1$ sampling strategies: one strategy per light source – consisting in sampling a point on the light source –, and the strategy that samples the BSDF. All light-based strategies are clustered in one group. By creating one small photon map containing only one-bounce photons per light source, the light strategies mostly have null probability at point $\mathbf{x}$ because of occlusion. The maps containing photons at $\mathbf{x}$ indicate which light sources contribute most to $\mathbf{x}$, accordingly affecting the strategies probability.

5.6.4 Other contexts than MIS-based estimation

Our representativity-based method can also be used in different contexts than MIS-based estimation, by building a PDF taking into account all strategies, making it for instance usable by any algorithm requiring to sample directions, such as all algorithms relying on local path sampling. This PDF can be used for samples of any type for which several importance sampling strategies are suitable (directions, points,

paths, etc.). For sample x (x being a direction, a point on a surface, a path, etc.), this PDF is defined by:

$$\mathrm{PDF}(x) = \sum_{i=1}^{N_s} p(s_i)\,\mathrm{PDF}_i(x) \tag{5.28}$$

where N_s is the total number of strategies, $p(s_i)$ is the probability of strategy s_i computed from its representativity, and PDF_i is the PDF associated to s_i.

Note that strategies can be ignored when creating an estimator. For instance for bidirectional path-tracing, this ability allows us to only consider strategies using importance-based maps when creating light sub-paths, and strategies using energy-based maps when creating camera sub-paths.

5.7 Conclusion

Multiple importance sampling (MIS) is a general and efficient way to perform integral estimation of complex functions using simple importance sampling strategies. We have shown that using measures of relevance of the strategies (given by representativity functions) improves the distribution of samples amongst the strategies, and therefore increases the robustness of estimators based on it. As we have shown that local path sampling can be improved using the exact same tools, it is possible to improve the robustness of most rendering algorithms by using the rendering-specific representativity functions that we derived for BSDFs, maps and caches, with negligible computational overhead.

However, even if more robust, there are still rare cases where very poor importance sampling still occurs: an important direction is sampled with a very low probability. This leads to a few samples with extremely large values, which create bright spots on the final image. As these bright spots require a large amount of samples to vanish, we enhance the robustness of the accumulation stage, by developing a method to avoid adding these large and rare samples to the image in the first place.

6

Sample-space bright spots removal using density estimation

6.1 Introduction

Standard accumulation (Section 4.1.1.1) relies on a weighted average (Equation (4.3)) to compute each pixel's value:

$$< I_p >= \frac{\sum_{i=1}^{N_p} h_P(x_i - x_p, y_i - y_p) \times N_i \times C(< L(x_i, y_i) >)}{\sum_{i=1}^{N_p} h_P(x_i - x_p, y_i - y_p) \times N_i}, \tag{6.1}$$

where x_i, y_i, N_i and $C(< L(x_i, y_i) >)$ are given for each sample to accumulate, and N_p is the number of (x_i, y_i) samples which belong to the support of the pixel's filter. $< L(x_i, y_i) >$ is obtained by Monte-Carlo simulation as an estimate of the form

$$< L(x_i, y_i) >= \frac{1}{N_i} \sum_{j=1}^{N_i} \frac{f(v_j)}{p(v_j)}. \tag{6.2}$$

Equation (6.1), combined with Equation (6.2), tells that the pixel value is computed as the estimated weighted mean of a positive quantity. This estimator is not robust to outliers, represented here by very large values of $\frac{f(v_i)}{p(v_i)}$. This means that even if a very large number of elements is used to compute each pixel, the presence of only one very large value can lead to an estimation much larger than the actual pixel value. Visually, this generates very bright pixels, denoted as *bright spots* (illustrated in Figure 6.1) that are still present even in high-quality images. Removing these bright spots is generally done as a post-process on the HDR or the tone-mapped image, but it often leads to blur.

The outliers which lead to bright spots are produced by the creation of a sample v whose probability density is low, and which yields a large or very large contribution (a large $f(v)$ value). If $p(v)$ is low, this means that few samples that can lead to bright spots are created when computing an image. Note that this property is true for most rendering algorithms, such as path-tracing [Kaj86], bidirectional path-tracing [VG94, LW93], or photon-mapping [Jen96]. However, this does not apply to Metropolis light transport

Full paper at Graphics Interface 2011, co-authored by Loïc Barthe and Mathias Paulin [PBP11c]

Figure 6.1: Examples of bright spots, which are very bright pixels surrounded by pixels whose value is much nearer from the real expected value.

[VG97] and similar algorithms, as their bright spots are caused by the accumulation of many samples at a single pixel, all the samples having exactly the same luminance. Note that for algorithms such as path-tracing or bidirectional path-tracing, $p(v_j)$ is not easily computable, as each estimate is in fact a (potentially weighted) sum of several other estimates. Moreover, as we work as an accumulator, the individual $\frac{f(v_j)}{p(v_j)}$ values can not be directly accessed without putting additional requirements on the simulation part, which we want to avoid as much as possible. We therefore work only on the $< L(x_i, y_i) >$ values.

The main contribution of this chapter is a method which detects and delays on the fly outlier values from a set of samples. More specifically for Monte-Carlo-based rendering, it avoids the presence of bright spots in the final image by detecting and delaying outstanding $< L(x_i, y_i) >$ values, without introducing blur. As our algorithm only depends on the $< L(x_i, y_i) >$ values, it suits our flexibility requirements and can be used as a "contribution-filtering" accumulator without modifying any other functional part of the rendering engine. A standard accumulator can then be used to reconstruct the final image from the filtered samples.

As shown in Section 6.2, many existing methods devoted to removing bright spots try to minimize their visual impact by filtering the final image [DSHL10, McC99, RW94, TM98, XP05], leading to visible smears or blur. Also, recently, a method using the joint image-color space to detect samples that can cause bright spots has been presented [DWR10]. Similarly to this method, our method is in essence an outlier detection method. This kind of methods is mostly used in data-mining applications. The methods proposed in this field assume a vast amount of data (several million samples), whereas in our case we only have a few tens or hundreds of samples for a given pixel. As presented in Section 6.3, we approximate the probability distribution of the luminance of the screen samples for each pixel using density estimation. We use this distribution to temporarily discard samples that are susceptible to be outliers, and definitively accept those that are surely not. This probability distribution is updated during the rendering, making our method progressive, and well suited to take advantage of adaptive sampling. As storing all the samples for each pixel would be too costly, we develop compact representations of the distribution in Section 6.4.2 and Section 6.4.4. These representations have different and complementary properties with respect to memory cost and precision. As shown in Section 6.4.6, these two representations can in fact

be used jointly, with parameters allowing the user to control the precision/memory cost ratio. Moreover, our method has a time cost which is independent of the scene, and which is shown to be a small fraction of the pure rendering time. Using our method, we obtain images in Section 6.5 that are greatly improved versions of the images obtained using standard accumulation, without neither having to use a very large number of samples, nor having images that exhibit smears or blur. This work has been presented as a full paper at the Graphics Interface 2011 international conference [PBP11c].

6.2 Related works

The estimator used in Monte-Carlo rendering systems to compute the value of each pixel, which is the mean value estimator, is not robust to outliers. Using over-sampling will not efficiently reduce the impact of an outlier, as a bright spot will be present unless a prohibitively large number of samples are computed for each pixel containing a bright-spot. This explains the existence of methods specifically developed to remove bright spots, which are based on filtering, either at the samples level, or directly on the final image.

6.2.1 Image-space bright spots removal

Image-space approaches directly process the final image, using anisotropic diffusion [McC99] or filtering. Xu *et al.* [XP05] recently proposed a technique based on bilateral filtering [TM98]. Bilateral filtering uses two filters to compute each pixel value. The *domain filter*, function of the *positions* of the current pixel p_1 and the neighbor pixel p_2, reinforces the influence of nearby pixels. The *range filter*, function of the *value* of both pixels $f(p_1)$ and $f(p_2)$, reinforces the influence of pixels that have nearby values, using a Gaussian filter centered at $f(p_1)$ and with a user-set standard deviation. Xu *et al.*'s method, instead of directly using $f(p_1)$ in the range filter, uses an estimate $\tilde{f}(p_1)$ of the true value of the current pixel, obtained by filtering the neighboring pixels. This modification makes the range kernel less sensitive to outlier pixels when reconstructing their value. This method is highly efficient, and requires a negligible amount of memory. However, even though providing much better results than previous approaches, bright spots caused by diffuse interreflections – which, in general, have lower values than bright spots involving caustics, as their $f(v)$ value is lower – still lead to visible smears, making high-frequency textures blurred. Moreover, for the range filter to perform correctly, its standard deviation must be chosen carefully, taking into account the typical orders of magnitude of the samples for the current scene, and the variance of the underlying integration algorithm. Yet more recently, Dammertz *et al.* [DSHL10] used a wavelet-based filtering to better approximate the hemispherical integrals that are typically computed in Monte-Carlo rendering from low-samples estimates, while avoiding the edges of features to avoid blurring these high-frequency elements. These features are detected based on geometric information, therefore requiring more data from the simulation part. It is very efficient for high-frequency/low-amplitude noise on scenes where the illumination changes slowly (typically mostly-diffuse scenes), but it fails when many high-frequency details are present, and no tests with bright-spots were presented. As the edge detection is partially based on the pixel values, it is likely that the bright spots would be considered as

single-pixel objects, therefore not being filtered at all.

6.2.2 Sample-space bright spots removal

Rushmeier *et al.* [RW94] changed the way image reconstruction is performed. Instead of using a constant-width filter for all samples, they compute a per-sample width, and use normalized filters to evaluate the contribution of the sample to each pixel. The width is computed so that all samples have comparable contributions, leading to lower high-frequency noise. However, when used on samples causing bright spots, this tends to blur the final image, as the filter width has to cover a very large number of pixels.

DeCoro *et al.* [DWR10] recently proposed to build a tree of the samples described in a joint image-color space, and accept upcoming samples only if the density of samples in the tree is sufficient to assess the new sample's correctness. This algorithm is simple, elegant, memory-efficient, and uses most of the correlation present when rendering an image. It works very well for well converged zones of the image, even for a very low number of samples per pixel. However, for parts where convergence is far from reached – typically parts where indirect illumination dominates –, more samples are delayed, leading to slightly darker zones. Moreover, a k-NN query is required per Monte-Carlo rendered sample, which can lead to an important computational overhead.

Although targeting similar final goals and using the same base toolkit – density estimation –, our approach has major differences with the method by DeCoro *et al.*. From a mathematical toolkit point-of-view, we use kernel-based density estimation on a 1D samples set, while they use k-NN queries on a 5D joint image-color space. From a resultant estimator point-of-view, they define a set of biased correlated average values estimators that are robust to outliers, while we define a biased average estimator that is robust to outliers, and use one such estimator per-pixel. Our estimator can therefore be used as a direct replacement for any average estimator – at the cost of introducing bias –, and can therefore target a wider range of applications.

6.2.3 Outlier detection

Outlier detection is a well studied approach in the statistical analysis domain. According to the survey by Hodge *et al.* [HA04], outlier detection methods can be split into three different categories, depending on the prior knowledge required on the data:

1. Methods requiring tagged data, of the form normal/abnormal, in order to build a model and then classify candidate data [Wet94]. It is impractical in rendering, as the user should tag enough samples per pixel before any automatic processing can be performed.

2. Methods requiring normal data tagged, and figure out abnormality [DF95]. For the same reason as above, no manual tagging must take place in the algorithm.

3. Methods that do not require any prior knowledge of the data [RRS00]. As no user tagging is required, this category is well suited for rendering.

Methods for outlier detection in a rendering context must fit in the third category, as we do not want to put any constraints on the algorithms used to compute each sample's value. Such existing methods presented in [HA04] target static distributions, where all the samples are available at once. As we want to be compatible with adaptive sampling, the number of samples can not be fixed in advance. The methods presented in the survey could be adapted by first building a model using a fixed number of samples, and then classifying the other samples without updating the model. As in rendering, each pixel is computed using very few samples (tens or hundreds of samples, compared to millions or billions for database applications), building a fix model can lead to a strong lack of robustness, we thus need a progressive method.

6.3 Bright-spot removal using density estimation

The simulation part computes many radiance samples $r_i =< L(x_i, y_i) >$ for a given screen position. Ideally, outliers could be detected by estimating the probability density function (PDF) of the samples that contribute to each pixel. The radiance samples with probability density lower than a given threshold could then be ignored, as they are not common and could be bright-spots. However, accurately estimating a PDF from a set of samples usually requires a large number of samples, typically several thousands or even millions. As typically only tens or hundreds of samples are available per-pixel, this solution can not be used.

Instead, we cluster the samples in groups, delaying samples which are not part of any group. This clustering uses density estimation on scalar values l_i, each being obtained from a radiance sample r_i. We obtain these l_i values by using the underlying structure of the radiance samples. In the case of path-tracing, each of the radiance samples is the sum of contributions from different light-transport mechanisms: direct-lighting, first-bounce indirect lighting, *etc.*. Each combination of these mechanisms has typical values that have different orders of magnitude. We can therefore differentiate the samples with respect to their contributing mechanisms by using the logarithm of the luminance of these samples (Figure 6.2).

Our algorithm detects outliers by finding the l_i values that are uncommon. To find them, we cluster the available values in groups, that we call *modes*. We define a mode as the biggest log-luminance interval $[a, b]$ in which the estimated PDF of the distribution defined by the l_i values is strictly positive, as illustrated in Figure 6.3. If a mode contains several l_i values, it is likely that the associated r_i are viable. We call such a mode an *extended mode*. Otherwise, if a mode contains only a single element, we can not conclude on the viability of the associated r_i sample. Such a mode is called *single*. As bright spots are caused by samples with very large luminance values, we only focus on dubious samples whose log-luminance is greater than the upper bound of the extended mode with largest values. This extended mode is called *last extended mode* from now on, illustrating that it is the last extended mode along the l_i axis. From this observation, our characterization of a dubious sample is: *a sample should not be added*

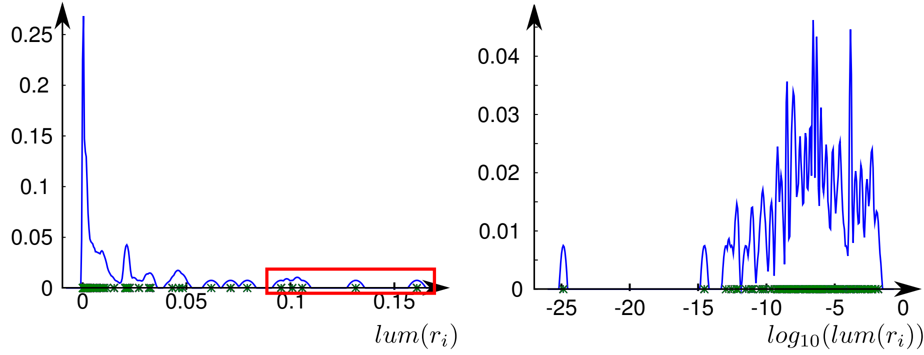


Figure 6.2: Left: density estimation (blue curve) based on the luminance of samples obtained by path-tracing (green crosses on the horizontal axis). Right: density estimation based on log-luminance. Note that the surrounded isolated values in the left image seem to have contributed to the same light-transport mechanisms, because they have similar orders of magnitude. They should therefore belong to the same mode. It is the case only when using log-luminance.

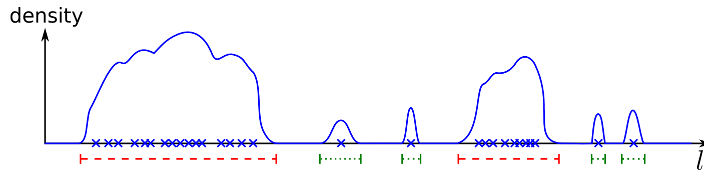


Figure 6.3: Modes are the biggest intervals of log-luminance values where the density is strictly positive. Red dashed intervals are extended modes. Green dotted intervals are single modes. The x-axis corresponds to the l values, the blue crosses representing the l_i values of the samples. The y-axis corresponds to the value of density obtained using kernel density estimation based on the l_i values. The blue curve is the density obtained. The last extended mode is the right-most extended mode.

to the final image if its associated l_i value generates a single mode greater than the last extended mode, or equivalently, if its l_i value is greater than the upper bound of the last extended mode, m.

Algorithm 1 presents a basic algorithm performing outlier filtering using our method. Note that it adds previously rejected samples that are finally classified as viable when the acceptance value m is updated, making our method computationally efficient.

As we need to know where the estimated PDF is strictly positive, our method requires to estimate the PDF of the distribution based on its samples l_i. The low number of samples we have is sufficient to retrieve this information, as it does not require a very accurate PDF estimation. Kernel density estimation [Sil86] is very well designed to build a PDF from sample values. From a set of samples $x_1, \ldots, x_n$, the estimated PDF $\overline{p}(x)$ at x is computed based on a per-sample bandwidth h_i:

$$\overline{p}(x) = \frac{1}{n} \sum_{i=1}^{n} K(x - x_i, h_i) \tag{6.3}$$

where K is a normalized kernel, in our case Epanechnikov kernel [Epa69].

Algorithm 1 Basic algorithm for kernel-density-estimation-based outlier rejection

```
Initialize samples log-luminance distribution S = {}
Initialize dubious list L = {}
for each rendered sample r, with log-luminance l do
   add l to S, updating or adding modes
   if the last extended mode has changed then
      recompute m, the upper bound of the last extended mode
      for each element s of L, with log-luminance y do
         if y < m then
            splat s
            remove s from L
         end if
      end for
   end if
   if l < m then
      splat r
   else
      add r to L
   end if
end for
```

6.3.1 Adaptive bandwidth

The most difficult task in kernel density estimation is estimating each kernel bandwidth, so that the reconstructed density is as close as possible to the original. On the one hand, the choice of a too low bandwidth leads to a highly oscillating density, with many spikes that are not present in the actual distribution. In terms of modes, many modes will be single, while they should not. On the other hand, a too large kernel bandwidth leads to a lot of smoothing, putting all the samples in one extended mode. For a better reconstruction, it is advised to compute a per-sample kernel bandwidth [Sil86]. From a reasonable base bandwidth h, an adequate reconstruction can be obtained using an adaptive bandwidth h_i for each log-luminance sample l_i:

$$h_i = nn \times \left(\frac{\exp\left\{ \frac{1}{n} \sum_{j=1}^{n} log\overline{p}(l_j) \right\}}{\overline{p}(l_i)} \right)^{1/2}, \tag{6.4}$$

where nn is the average distance to the nearest neighbor of each sample, and $\overline{p}(l)$ is the density at a point l estimated using nn as bandwidth for all samples (Equation (6.3)).

6.4 Lowering memory consumption

Computing adaptive bandwidths requires us to store all the samples, in order to be able to compute the nn value and the final bandwidth at each sample. Although this storage can be done when computing one pixel after another, all the samples cannot be stored in a more generic rendering context, when using

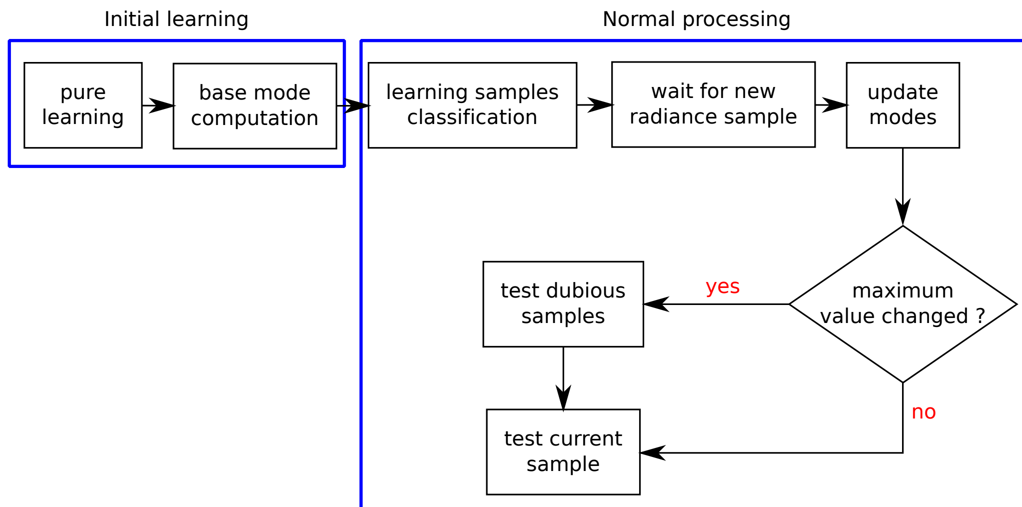


Figure 6.4: The general algorithm used in our bright spot removal method.

image-based adaptive sampling for instance. As we also want to be able to deal with these situations, we develop a more specific method based on Algorithm 1 to limit memory consumptions.

6.4.1 Algorithm overview

Figure 6.4 presents the general procedure used in our method for each pixel. Two main phases are performed: a pure learning-phase, which basically computes a base m value, and then normal processing, which updates it.

The pure learning phase just consists in storing the first N r_i values, N being fixed by the user. For non-black samples, the logarithm of the luminance l_i is also computed. After the N-th sample has been stored, modes are built using a specific representation, based on the N l_i values (Sections 6.4.2 and 6.4.4). m is computed, and each of the N samples is tested: if its l_i value is below m, the associated sample is splatted, otherwise it is put in the dubious list. Once this is done, the algorithm switches to normal processing.

In the normal processing phase, each time a new radiance sample r_i is computed, the modes representation is updated to take into account the sample's log-luminance value l_i. This update can have three different consequences on the representation: creation of a new single mode, modification of an existing extended mode, or creation of a new extended mode from an existing single mode. In the two last cases, the maximal value m is updated if the last extended mode has been modified or replaced. If m has been updated, all the samples in the dubious list are tested, as they may be below the new m value. This makes our method computationally efficient, as samples are not permanently rejected, but just delayed. After this update phase, the new sample is accepted if its l_i value is below m. If l_i is above m, r_i is added to the dubious list. Note that the dubious list size can be bounded.

6.4.2 Approximate distribution representation (ADR)

As outliers are samples that are far from the viable ones, having an exact value for the base bandwidth nn is often not necessary. To avoid consuming too much memory, we develop a compact approximate representation of the distribution, which keeps an exact representation of the single modes, while approximating the extended modes with only the lower and larger samples that belong to it. This representation allows us to compute a close approximation of nn, while drastically reducing the memory consumption. As a matter of fact, a large majority of the samples belonging to a pixel are in extended modes, as these modes represent log-luminance intervals where samples are the most probable to lie.

In an extended mode, the precise position of each sample is not useful in order to get a correct approximate value of the nn base bandwidth. Instead, we approximate the position of the M samples of an extended mode by M regularly spaced artificial samples. Therefore, for each extended mode, we just need the two extrema samples, and the number of samples in the mode.

Initial learning: It consists in storing a user-defined number N of samples, and then building an approximate distribution from these samples. Once the N samples have been stored, the list is sorted by increasing log-luminance values, the bandwidth of each sample is computed according to Equation (6.4) and each sample l_i is browsed sequentially to build modes by aggregation: when the kernels of two consecutive samples overlap, they belong to the same mode. Each time a mode is detected, its extrema samples and number of samples are stored. Once the modes are built, the maximum acceptable value m is set to the log-luminance of the maximum sample of the last extended mode.

Incremental update: Once the base modes have been computed, the representation is updated for each new radiance sample r, with a log-luminance value equal to l. If l is included in the interval of an extended mode, this mode's number of samples is just increased by 1. Otherwise, a single mode centered at l is created and added to the representation, and we compute its potential overlap with existing modes. If an overlap is found, it leads to either extending the bounds of an existing extended mode, or creating a new extended mode from two overlapping single modes. When l does not belong to any mode, its kernel bandwidth is computed using the samples of the approximated distribution, and overlap between this kernel and the kernel of each sample of the approximated distribution. Note that merging of modes can occur if the kernel of l overlaps both the mode before it and the mode after it. In this case, a new extended mode replaces the two existing modes.

6.4.3 ADR: analysis

Robustness: The third row of Figure 6.7 shows the results obtained when we removed bright spots using our approximate distribution representation (ADR). We have tested our method on several scenes, using path-tracing and 50 samples for the initial learning phase. All these scenes exhibit many bright spots when rendered using path-tracing, which makes them relevant to evaluate our method. Each pixel of each image can be considered as an independent test, as each pixel has its own independent estimator.

	ring		living room		computer room	
	sing.	ext.	sing.	ext.	sing.	ext.
100	2.3M	4.0M	1.7M	3.2M	568K	1.2M
HQ	3.1M	4.7M	4.0M	5.2M	1.8M	1.8M

Table 6.1: Total number of single and extended modes for various scenes when using ADR, for images computed with 100 samples per pixel and high quality ones (HQ), obtained using adaptive sampling and between 200 and 1000 samples per pixel. The ring images have been computed at a resolution of 800×600, the others at a resolution of 800×450 pixels.

Our method successfully delays most of the samples leading to bright spots without delaying the viable ones, consequently improving the image quality.

Progressiveness is an important property of our method, as it allows us to splat samples only when they are viable. We have tested the effectiveness of this property on a glossy caustic. The second column of Figure 6.8 presents the results obtained by ADR when adaptive sampling is used. It shows that ADR first delays many of the caustic samples, but finally updates its maximum value as they are considered correct, leading to an adequate caustic. Note that even for the very difficult case of the glossy caustic on the glossy part of the floor, it correctly separates caustic samples from bright spots.

Discussion: The approximation introduced in this mode representation allows us to use our density-estimation-based method without having to store all the samples. It is both robust and easy to parametrize (using 50 as initial learning size has been proved efficient on all our tests), and introduces a small rendering time overhead. However, as illustrated by Table 6.1, this representation can lead to the storage of a large number of modes, and so, a large memory cost, even if it is drastically reduced compared to a brute-force approach. This is due to the necessity of using all modes in the computation of the average nearest-neighbor base bandwidth nn, and the adaptive bandwidth.

6.4.4 Rule-of-thumb bandwidth (ROT)

To further reduce the memory consumption of the ADR representation, we propose an alternative method to compute each kernel bandwidth, which does not require us to store all the modes. This method, although less robust, can still lead to very good results when the outliers are easy to identify. In this case, our density estimation method performs efficiently even with very conservative kernel bandwidths. Therefore, we choose to use a constant bandwidth instead of the adaptive one used in ADR. This bandwidth is computed using the statistical properties of the underlying distribution. Such a bandwidth called *rule-of-thumb* (ROT) has been introduced by Deheuvels [Deh77, Tur93]. For Epanechnikov kernels, the ROT bandwidth is obtained as:

$$h_{rot} = (25 \times n)^{-1/5} \times \sigma, \tag{6.5}$$

where n is the number of samples in the distribution, and σ their standard deviation. Using this bandwidth, we just need to store the log-luminance of the largest sample of the last extended mode, and the

Figure 6.5: Close-up on the glass sphere of the ring scene, where the ADR method (left) performs better than the ROT one (right).

larger single modes. All the modes located before the last extended mode are no longer needed, as any sample that is below the last extended mode is automatically accepted.

Initial learning: Initial learning for this representation is very similar to the one of the ADR method. The only essential differences are during the base modes computation: there is no adaptive bandwidth to compute, and when a new extended mode is built, all the modes that have lower upper bounds are released, as they are not used anymore.

Incremental update: When a new sample is added, if it is lower than the current maximum acceptable value, nothing has to be updated. When it is larger, overlap is examined using the constant bandwidth given by Equation (6.5). If merging occurs and a new extended mode is created with values larger than the last extended mode, this new extended mode becomes the last one, and the lower modes are released. Otherwise, a new single mode is added.

6.4.5 ROT: analysis

We perform the same tests as in Section 6.4.3.

Robustness: the results, presented in the fourth row of Figure 6.7 show that this method, although a little less robust than ADR, succeeds in identifying the outliers in most cases.

Progressiveness: the same progressiveness test as for ADR (Section 6.4.3) has been performed. Figure 6.8 shows that similarly to the ADR method, the ROT method correctly found that caustic samples are acceptable, while still delaying samples producing bright spots.

Discussion: the results provided by this method are not as good as those provided by ADR (Figure 6.5).

	ring		living room		computer room	
	sing.	ext.	sing.	ext.	sing.	ext.
100	198K	482K	184K	360K	102K	331K
HQ	206K	482K	200K	360K	99K	331K

Table 6.2: Total number of single and extended modes for the various scenes using ROT. Note that the number of extended modes corresponds to the number of sampled pixels.

	ADR only		ROT only		$T=1, \delta=0$		$T=1, \delta=0.5$		$T=2, \delta=0$		$T=2, \delta=0.5$	
	sing.	ext.	sing.	ext.	sing.	ext.	sing.	ext.	sing.	ext.	sing.	ext.
100	2.2M	3.8M	190K	433K	710K	1.4M	421K	872K	1.8M	3.2M	1.3M	2.3M
HQ	6.6M	8.7M	316K	483K	1.3M	1.9M	709K	986K	4.8M	6.5M	2.6M	3.3M

Table 6.3: Left: number of modes when using only ADR or only ROT on the ring scene. Right: number of modes when using our hybrid method, in function of T and δ. As expected, fewer modes are present when the parameters favor ROT over ADR.

However, for most pixels, results are either identical or very similar, with a tendency for ROT to accept more samples than ADR. This comes from the constant bandwidth computed using ROT that is more conservative than the adaptive bandwidths computed with ADR, thus leading to larger modes.

6.4.6 Switching from ADR to ROT

ADR is robust but can consume a lot of memory, while ROT works well for most cases, with a lower memory consumption and lower computational cost. However, it performs poorly in more intricate cases. We therefore develop a method that allows us to use, per-pixel, either one or the other, depending on the difficulty to differentiate outliers.

Both the ROT and the ADR methods store the last single samples after their last extended mode. More precisely, as ROT is more conservative, it is likely that the larger single modes of ROT are included in the larger single modes of ADR. These modes can therefore be shared by both methods. Moreover, the computational cost of ROT is negligible. Thus, in order to reduce memory consumption, we want to use ROT whenever possible. We introduce two user parameters, δ and T, which allow us to control the transition from ADR to ROT. At the beginning, both methods are maintained, using ADR's maximum acceptable value to test the samples. For each sample leading to an update of the maximum acceptable value of one of the two methods, we compute the relative distance between the maximum of ROT m_{rot} and the maximum of ADR m_{adr}, defined as $|m_{rot} - m_{adr}|/m_{adr}$. If T such successive distances are less or equal to δ, ROT becomes the only method used for a pixel, and the data used for ADR is released.

These two parameters allow the user to specify a measure of similarity between the results. If the two methods give sufficiently close maximum values (δ) during a sufficiently long time (T) for a pixel, then ROT can safely be used alone, leading to a greatly reduced memory consumption for this pixel as the data for ADR is no longer needed. Experimentally, requiring equal maximum values ($\delta = 0$) during two successive steps ($T = 2$) proved a good compromise between memory consumption and robustness

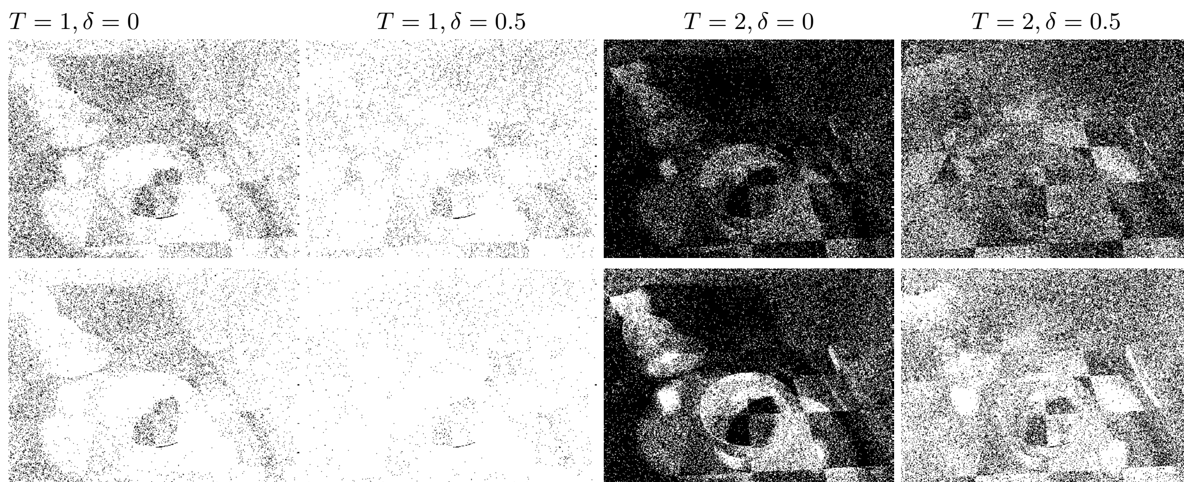


Figure 6.6: Top: in gray, pixels for which only ROT is used on the ring scene, after 100 samples per pixel have been computed. Bottom: the same after the computation of 500 samples per pixel. Note that ROT does not tend to be used everywhere, but rather in regions.

for all our test scenes.

Figure 6.6 shows where ROT is used as the only method in the ring scene, for various values of the parameters T and δ. The use of ROT has been observed after 100 samples per pixel have been computed, and after uniform over-sampling has been performed. These parameters effectively allow the user to choose between precision and memory, by defining the amount of similarity desired between the two methods, and verifying this similarity over a small or large time interval before using ROT only. As shown by Table 6.3 and as expected, the more ROT is used, the lower the memory consumption. The column labeled ADR+ROT of Figure 6.7 presents the results obtained when using both methods as indicated here, with $T = 2$ and $\delta = 0$. These images have been computed using the same radiance samples as those used when testing ADR and ROT.

6.5 Results

All the results shown below have been computed with an initial learning set size of 50 samples.

Robustness: Figure 6.7 shows the results obtained using ADR, ROT, and both at the same time with parameters $T = 2$ and $\delta = 0$, on three different scenes. We compare it to the image obtained using the standard average estimator, and the one obtained by the method from DeCoro *et al.*(called DBOR in the remaining). These images have been shot after 100 samples per pixel have been computed. Note that for each scene, the exact same radiance samples were used to compute each image. Each pixel of each image can be considered as an independent test of our method, as we define one independent estimator per-pixel. Each pixel of the images produced by DBOR can be considered as a reference, as their method employs virtually a much larger number of samples for their classification, through the use of the joint-

scene	ADR + ROT	DBOR
ring	1.3%	4.0%
living room	1.4%	2.3%
computer room	0.5%	1.1%

Table 6.4: Percentage of samples delayed in average by the ADR + ROT method and the one by DeCoro *et al.*, after 100 samples per pixel have been computed.

space: for each sample, they potentially use all the spatially nearby samples. We can see that results of all the estimators obtained using our method are really close to the ones from DBOR, demonstrating the robustness of our method, even though less samples are used for classification: for each sample, only the samples that contribute to the same pixel are used. Very few pixels are still bright spots, but our method removes less samples than DBOR in dim zones (which leads to slightly darker results for DBOR), or in large variance zones such as caustics. This confirms that our method can be used as a drop-in average estimator replacement even when the number of samples is relatively low. Table 6.4 confirms that the average number of samples delayed by our method is kept small, smaller than the one of DBOR while still removing most of the bright spots.

Progressiveness: as shown by Figure 6.8, progressiveness has been tested in a case where the sample distribution is hard to handle when using path-tracing: a glossy caustic on a glossy floor. As the glossy caustic is rarely sampled and its samples have a quite large value, it can be mistaken for outliers. Our method first considers these samples as outliers, but as more samples are computed, it detects they are viable, while still delaying high-value samples which are not part of the caustic.

Performance: the representations we use induce a small overhead in computation time (from 5 to 10 percent for the measurements we have done). Note that it is independent from the scene complexity as we only rely on the values returned by the integration algorithm, not on the scene itself. For a bright-spot-removal application, it is negligible compared to the time that would be required to remove the bright spots by pure over-sampling (to avoid blurring when filtering), and remains lower than the overhead caused by DBOR, which suffers from poor parallelization efficiency when used on a many-core architecture. As a matter of fact, the accesses to the samples tree have to be thread-safe. Our method does not suffer as much from thread-safety, as the accesses to different pixels representation can be made in parallel.

Comparison with image-space methods: specific image-space methods have been developed to handle bright-spots. Figure 6.9 shows that even if removing the bright spots, they still introduce a lot of artifacts in the resulting images, such as blur or deformations of objects.

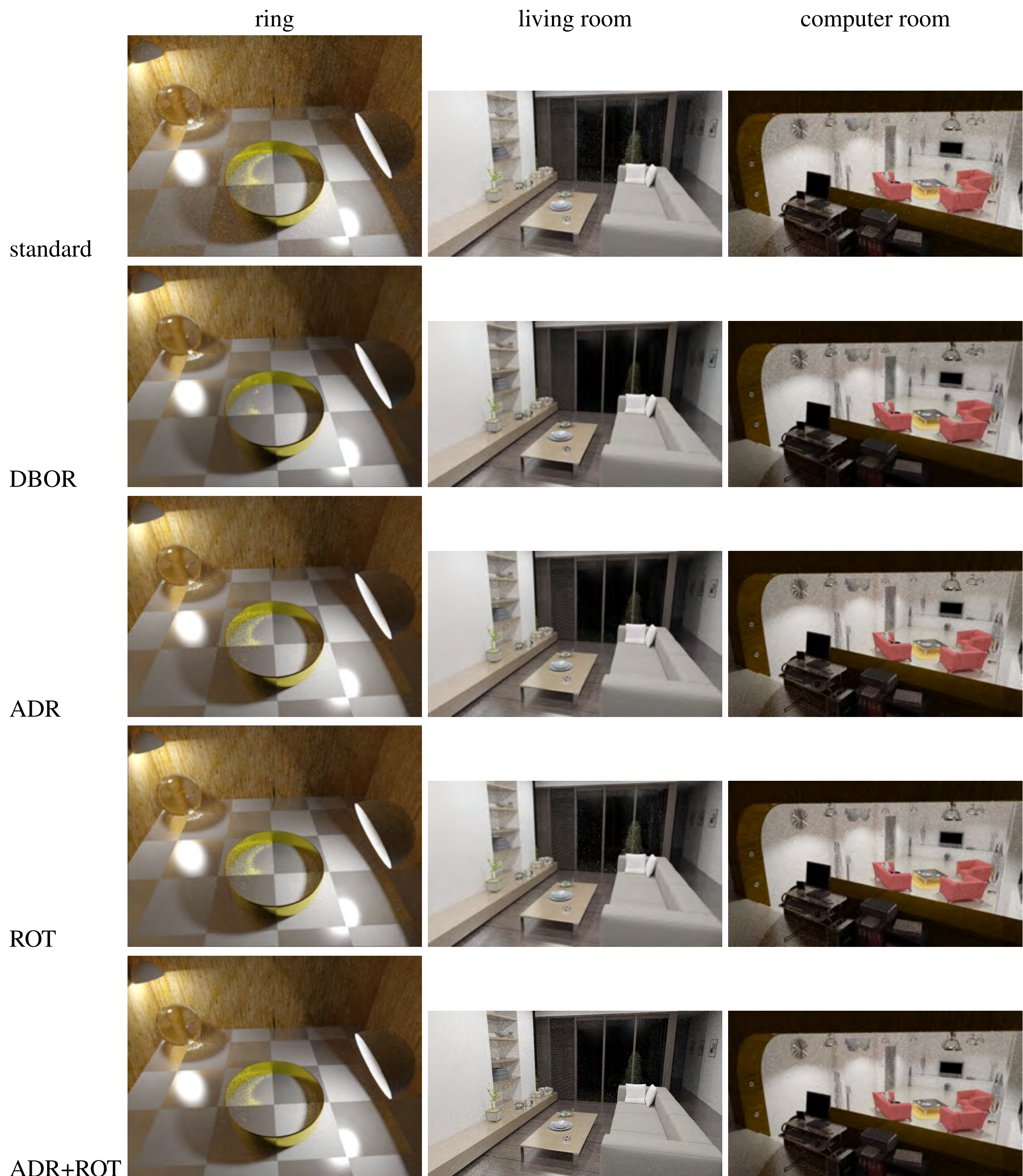


Figure 6.7: Images obtained using different average estimators to compute each pixel's final value from 100 samples obtained using path-tracing. First row: standard average estimator. Second row: average estimator as defined by the DeCoro *et al.* method, which uses inter-pixel correlation. Third, fourth and fifth rows: estimators obtained using one or both of the approximations we developed to lower the memory consumption. (See additional material for full size pictures.)

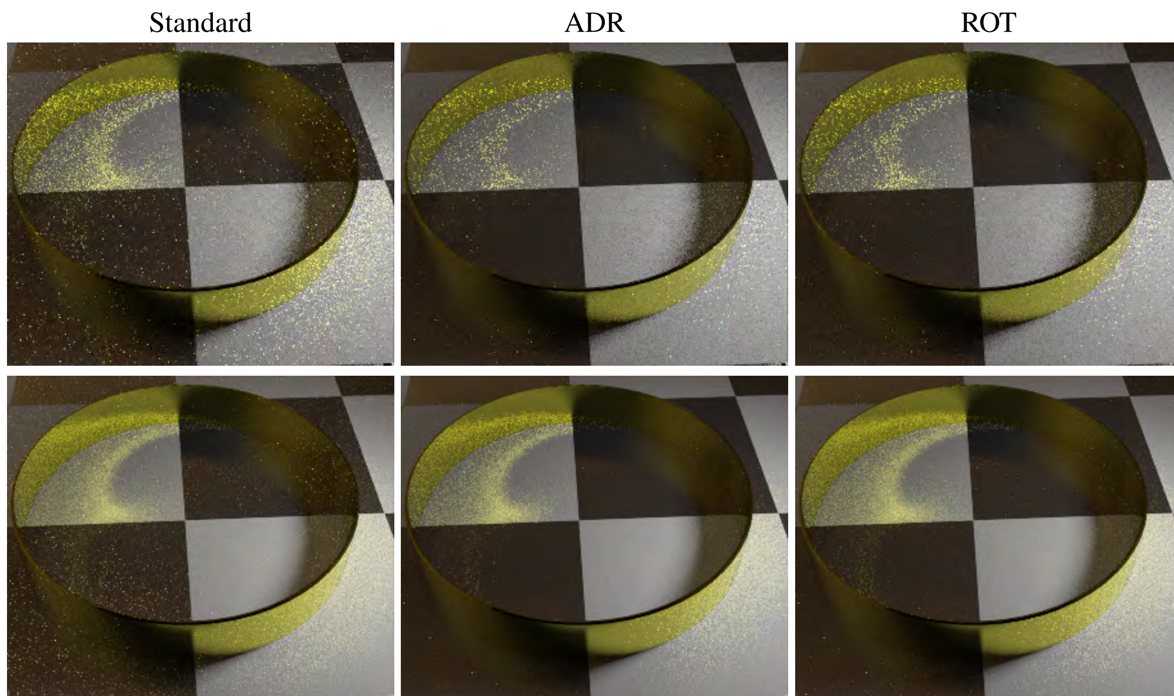


Figure 6.8: Top: images obtained with 100 samples per pixel. Bottom: images obtained with 1000 samples per pixel. Note that a large number of very visible bright spots are still visible on the brute path-tracing image.

6.6 Conclusion

Using kernel-density estimation and a proximity measure allows us to define a contribution-filtering accumulator which detects and delays outliers on the fly. From a usage point of view, when using one of the two representations we develop for bright-spot removal, our algorithm is only parametrized by the number of samples to use during the initial learning phase. The lower this parameter, the faster the first image obtained as learning is shorter, but results might be less accurate. Our tests, performed on various scenes, show that setting this parameter to 50 leads to good results. This shows that this parameter is in fact almost scene-independent, meaning that our algorithm is in practice parameter-less and can be directly applied on any new scene using $N = 50$. When using both representations at the same time, two additional parameters T and δ have to be set, to control the robustness/memory consumption ratio. In practice, we have found that $T = 2$ and $\delta = 0$ gave good results.

This method has been shown to be effective at removing most of the bright spots from an image which are produced by a non-perfectly robust simulation method. However, there are still some remaining bright spots. We process these few remaining bright spots using an image-space method presented in the next chapter.

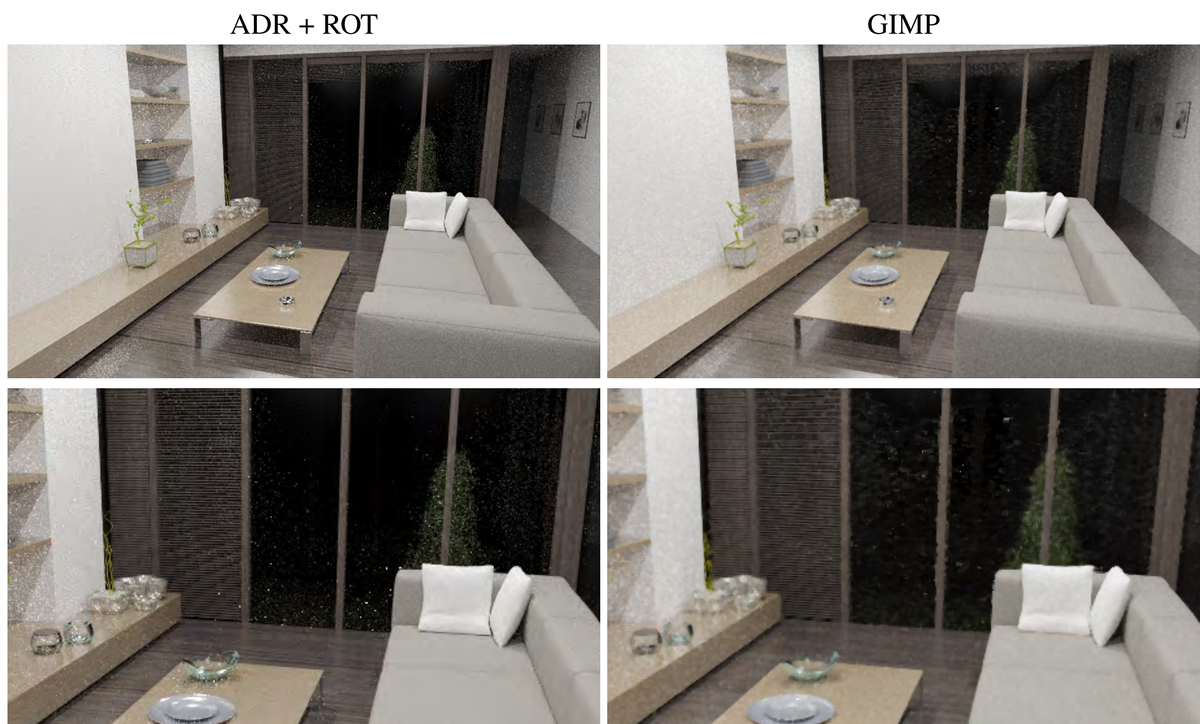


Figure 6.9: Left: image obtained using our method. Right: image obtained by applying the GIMP despeckle filter on the *tonemapped* image obtained using the standard average estimator. Top: full image, bottom: close-up. Note the amount of blur and the deformations at the edges of the objects added by the GIMP filter, in addition to the fact that it can not handle HDR images.

7

Effective despeckling of HDR images

7.1 Introduction

In our rendering engine, HDR processing handles perception-related issues such as tone mapping, and performs more elementary image processing tasks such as edge detection, sharpening, ... These operations relie on operators which whose robustness to bright spots is not guaranteed. We thus develop an HDR despeckling processor to remove the few bright spots that remain after the accumulation phase, adding neither blur nor artifacts. Ideally, a bright-spot free HDR image should be kept unchanged by HDR despeckling.

A popular image-based despeckling method is bilateral-filtering or more specific adaptations [XP05]. However, these methods induce noticeable artifacts, such as blur (Figure 7.3, 3rd row and Figure 7.4, 3rd row).

Our robust image-based approach consists in two steps: speckles detection, and reconstruction using pixels not tagged as speckles. This is highly different from filtering methods, which process all pixels the same way, reconstructing each pixel using all their neighbors. Our method does not introduce blur, naturally preserves edges and thin image features, yielding an artifact-free reconstruction. A preliminary version of this approach has been accepted as a technical sketch at SIGGRAPH Asia 2011 [PBP11a].

7.2 Tag-and-reconstruct despeckling filter

Speckles detection: The key insight of our method is that we do not detect speckles directly (which requires a characterization of speckles), but instead we detect non-speckled pixels. Therefore, we need only a single characterization of non-speckled pixels to handle very different types of speckles, not only bright-spots. Our characterization is based on coherence: if a pixel belongs to a coherent region, then it is not a speckle. For each pixel p, we therefore build a set S_p of pixels coherent with p. If the size of S_p is larger than a user-defined number of pixels N_c, then p is tagged as not being a speckle. We use $N_c = 10$.

Technical sketch at SIGGRAPH Asia 2011, co-authored by Loïc Barthe and Mathias Paulin [PBP11a]

We define a coherent region with respect to p as a set of connected pixels including p (spatial coherence), with all the pixels in S_p having a color perceptually close to the one of p (color-space coherency). This last point is assessed by using the perceptually-linear $L^*a^*b^*$ color-space. More specifically, a pixel q is found as similar to p from a color point-of-view if:

$$\sqrt{(a^*(q) - a^*(p))^2 + (b^*(q) - b^*(p))^2} < d \tag{7.1}$$

and

$$\frac{L^*(p)}{L^*(q)} < r. \tag{7.2}$$

The first criterion controls the perceptual chromatic distance between p and q, the maximum distance d being a user parameter. The second criterion, parameterized by a user-defined ratio r, controls that p is a rough local minimum of lightness, and is therefore in accordance with the local lightness in this zone of the image.

The key advantage of this region-based definition is that it does not presuppose anything about the image. Coherent regions can have arbitrary shapes: it can be a long thin feature or a block. This is crucial to correctly handle visual edges or textured zones. Additionally, it does not require any uniformity in the lighting. As we use local and relative properties, the images can contain very dim zones and very bright ones at the same time, they will be handled equally well.

The detection algorithm can be easily implemented with a flood-filling approach. Leaving from p, the pixels in its one-ring neighborhood are examined, excluding diagonal pixels. Pixels satisfying Equations (7.1) and (7.2) are accounted as being part of S_p, and put in a stack. Then, the pixel at the top of the stack is considered in a similar way, until either the stack is empty or S_p contains at least N_c pixels.

Note that the color-space criteria are not commutative, which means that S_p has to be built for all pixels: a pixel belonging to S_p is not necessarily not a speckle.

Reconstruction of the tagged pixels is performed with a simple Gaussian filter of width w_g (we use $w_g = 5$), but it only considers pixels that are not tagged as speckles. This helps avoiding creating new speckles from the ones we just removed. Note that this also implies that a pixel (x, y) is reconstructed without using its original value $I(x, y)$. From a theoretical point of view, this can be seen as a bilateral filter with a binary range filter:

$$I'(x,y) = \frac{\sum_{(x',y') \in \mathcal{N}(w_g)} I(x',y')G(x',y',w_g)H(x',y')}{\sum_{(x',y') \in \mathcal{N}(w_g)} G(x',y',w_g)H(x',y')}$$

where $\mathcal{N}(w_g)$ is the neighborhood of size w_g of pixel (x, y), G is the Gaussian filter centered at (x, y), and the range filter $H(x', y')$ is 0 if (x', y') is a speckle, 1 otherwise.

Efficient implementation: Parallelism over the pixels is used for each step of this algorithm. First, a $L^*a^*b^*$ version of the image is computed. Then, clustering is done for all pixels, and finally reconstruction of speckled pixels is performed. Note that the flood-filling algorithm requires per-pixel thread-

tagging to avoid counting a same pixel several times as part of S_p. Although this is not a problem on CPU, this would have to be handled properly for an efficient pure GPU implementation. However, on our test machine, processing times are largely below a second even for large images, as shown in Section 7.3. As a consequence, a CPU version is sufficient for our purpose of physically-based rendering.

Setting parameters: This technique has shown to be relatively sensitive to the N_c and r values, while using $d = 20$ always gave good results. For N_c, using $N_c = 10$ gave good results for all of our tests. $N_c = 5$ was not selective enough, and $N_c = 20$ was too restrictive. Depending on the image, we have used r values between 1.1 and 1.5, without being able to find a single value suitable for all cases.

7.3 Results

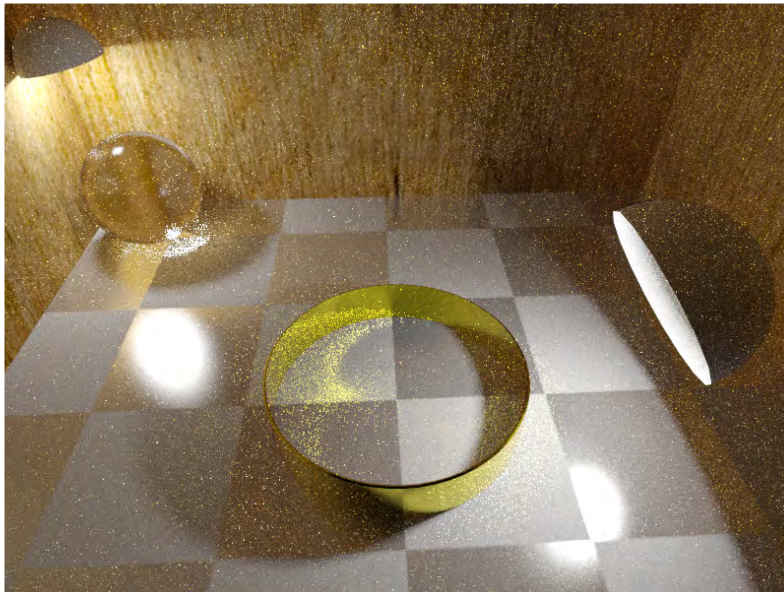
Test setup: Our tests have been performed on a Intel core i7 3.07Ghz, using all available cores and a not particularly optimized implementation. The base HDR images, shown in Figure 7.1, have been obtained either by adding bright-spots to a converged image, or from path-tracing. The first test allows us to verify that we correctly process the bright spots while not introducing visible blur or artifacts in converged zones. Images from path-tracing have been computed with a relatively low number of samples and without using our sample-space bright-spot removal method, on scenes where this rendering algorithm exhibits large variance, leading to numerous speckles. This allows us to verify that only low-amplitude noise remains. All the images in this section have been tone-mapped using a simple exposure-based tone-mapper, which does not lead to any noise reduction.

Results images: Figure 7.2 shows the speckles detected by our method on our test images. Figure 7.3 shows the three images before and after complete processing, as well as a comparison with results obtained using Xu *et al.*method. Close-ups presented in Figure 7.4 clearly show the effectiveness of our method. These figures illustrate the efficient removal of both high-frequency/large-amplitude and high-frequency/middle-amplitude noise, leaving only low-amplitude noise. In addition, no visible blur is added on textures, and edges and very thin features are well preserved (see the one-pixel-wide green plant at the bottom-right of the second scene's close-up in Figure 7.4). Comparatively, Xu *et al.*method adds a lot of blur as well as artifacts due to numerical instability caused by very large pixel intensity values, whatever the set of parameters we tried.

Computation times: Our method has a linear complexity with respect to the number of pixels. The HDR images in Figure 7.3 have been processed in 0.2, 0.07 and 0.05 seconds respectively.

Handling low-dynamic-range images: Our algorithm can also be used for low-dynamic range images. However, in this case value lightness would not be as large as in HDR images, making the lightness criterion less discriminant.

Synthetic addition of bright spots, 1600×900



Path-tracing, 800×600

Path-tracing, 800×450

Figure 7.1: Tone-mapped version of the HDR images used to test our method, together with the way they were obtained and their resolution.

Figure 7.2: Left: base HDR image. Right: Speckles detected by our method.

Figure 7.3: Results of our method, and comparison with Xu *et al.*method. First row: original HDR images. Second row: after despeckling by our method. Third row: Results obtained by Xu *et al.*method, with a window size of 4 and a range factor set to 0.01.

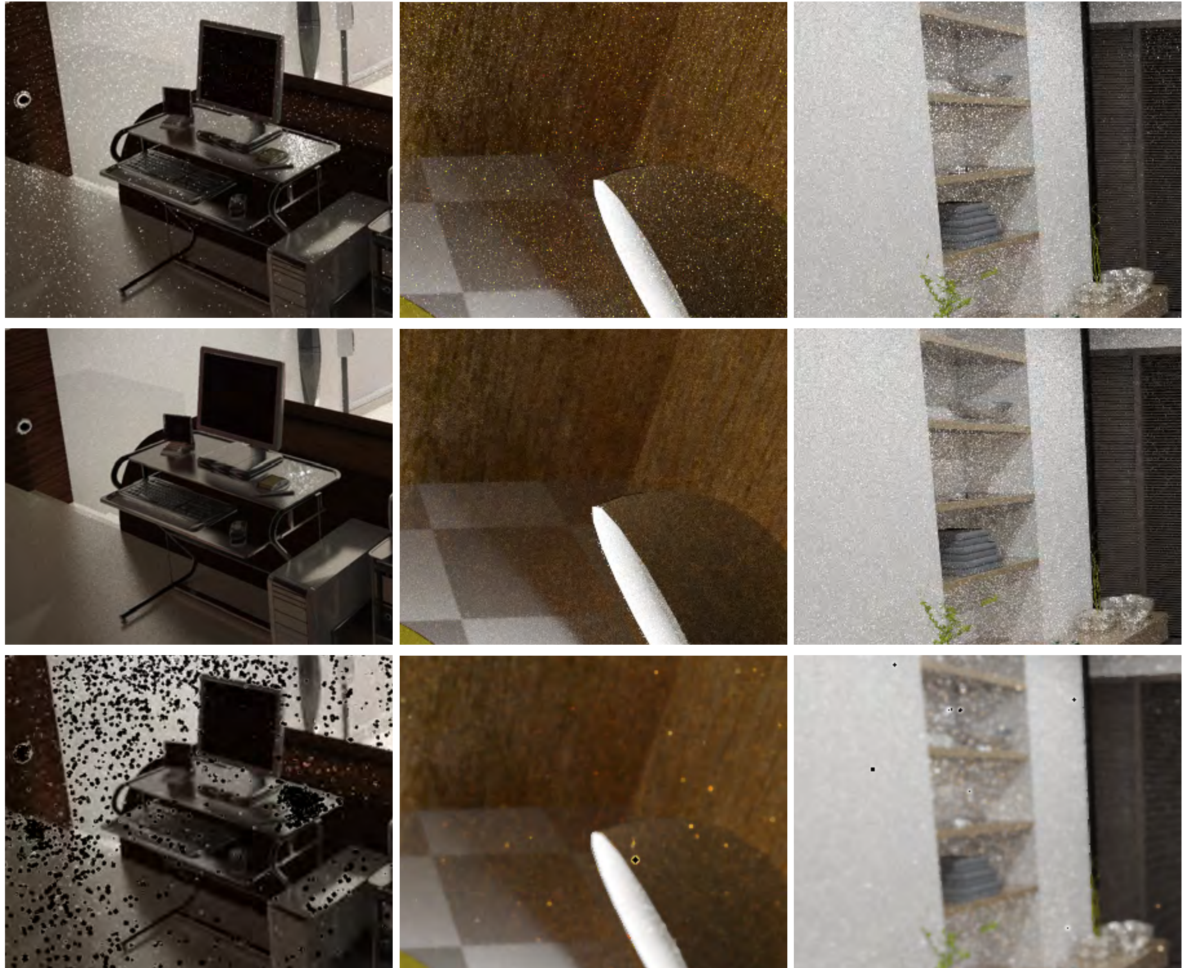


Figure 7.4: First row: close-up on original HDR images. Second row: after processing by our method. Third row: after processing by Xu *et al.*method.

7.4 Conclusion

Although correctly handling all the speckles, our method may tag high-frequency noisy features as speckles (as the not-well-converged large glossy reflection on the table of the first test image, see Figure 7.4) or small bright features such as some of the specular highlights of the plate in the third image of Figure 7.3. Moreover, finding a good value for all images for the lightness ratio is difficult. As the algorithm is fast, a GUI allowing the user to set r and select regions which should be ignored can be used for real-time adjustment. Although adequate for single-image production, this solution is not adequate for animation or fully-automated rendering. As a side note, the remaining low-frequency/low-amplitude noise can be removed while ensuring accurate results by averaging several HDR images obtained independently.

From the point of view of the engine architecture, methods to improve the robustness of each element of the "integration $\rightarrow$ accumulation $\rightarrow$ HDR processing" pipeline have been presented. Any integration method can use representativities (Chapter 5) to improve local path sampling and avoid arbitrarily large variance, accumulation can be made more robust to remaining poor importance sampling using our contribution-filtering accumulator (Chapter 6) between the simulation and standard accumulation, and the last remaining bright-spots can be effectively removed by the image-space approach we just presented. The last part of the rendering process for which we have identified a possible lack of mathematical robustness is adaptive sampling, which we consider in the next chapter.

8

Robust adaptive screen-sampling for Monte-Carlo-based rendering

8.1 Introduction

Focusing resources on parts of the image where the convergence is more difficult to reach leads to a large decrease of rendering time. This can be done efficiently by either adding samples to pixels using a probability proportional to a measure of the pixel's estimated error, or using a measure of quality of the pixel telling if it should be oversampled or not (such as entropy-based methods).

As presented in Section 4.1.1.1, each pixel value is a stochastic estimation (Equation (4.3)):

$$< I_p >= \frac{\sum_{i=1}^{N_p} h_P(x_i - x_p, y_i - y_p) \times N_i \times < S(x_i, y_i) >}{\sum_{i=1}^{N_p} h_P(x_i - x_p, y_i - y_p) \times N_i}. \tag{8.1}$$

A natural error measure is variance. It is used in adaptive sampling methods relying on confidence intervals, such as in [Pur87]. A problem is that tone-mapping can make bright and dim regions look similarly bright in the final image, so absolute variance can not be used directly. Relative error measures should be used instead.

Error measures are often computed from the previous $< S(x_i, y_i) >$ samples. This introduces two problems:

- As already examined in Chapter 6, the simulation part can produce outlier values in the estimates. We therefore need to ensure that our error estimate is robust to these outliers: an outlier should lead to an appropriately large error. Note that even if using our contribution-filtering accumulator to avoid the appearance of bright-spots in the image, we should seek to reduce the actual error, so that with enough time, all samples are added to the image, without creating bright-spots.

- Pixel sampling depends on the error estimate, which itself depends on pixel sampling. This leads to a poor estimate of the actual error for pixels whose initial error estimate is low, and can lead

Poster at SIGGRAPH Asia 2011, co-authored by Loïc Barthe and Mathias Paulin [PBP11b]

to highly visible artifacts. Instead of making rendering times smaller, adaptive sampling with a non-independent error estimate can lead to much longer rendering times than rendering without it because of a few very visible artifacts (see Figure 8.2c for an illustration), or even worse to highly visible bias if no samples are added to pixels found once as converged while it is not the case.

In this chapter, we address these two issues with very simple yet effective methods. First, we use a robust error estimator to obtain accurate error estimates. Second, we alternate between uniform and adaptive sampling, to consistently reduce the error estimate variance for all pixels. This yields a complete adaptive sampling algorithm that is simple to implement, robust, and correctly focuses processing power on pertinent parts of the image. Alternation in itself can be used to improve the robustness of any adaptive sampling algorithm, being based on statistical measures or not. This work has been presented as a poster at the SIGGRAPH Asia 2011 international conference [PBP11b].

8.2 Robust error-based adaptive sampling

Robust error estimate: A theoretical relative error measure for the pixel value I_p from the current estimate $< I_p >$ is

$$e_t(I_p) = \frac{V\left[< I_p >\right)]}{I_p^2}. \tag{8.2}$$

Computing the variance of $< I_p >$ is not easy: in fact, the filter weights are themselves random variables, as they depend on the sampled screen coordinates. To overcome this problem, we perform a slight approximation to compute our errors (it does not impact accumulation at all): we here consider a relative measure of error of the average of $< S(x_i, y_i) >$ values computed from the radiance values given by the simulator (Section 1.5.2), which is equivalent to considering that the filter is a constant filter over the original filter's support. As we use a relative error, there is no need to scale each radiance value by the filter's value, as this scaling factor would also be present in the expected value. Taking into account the fact that a single $< S(x_i, y_i) >$ value represents the sum of N_i single signal estimates, we define our reference error as:

$$e_r(I_p) = \frac{V\left[\left\{\frac{< S(x_i, y_i) >}{N_i}\right\}\right]}{N_p \times E\left[S(X, Y)\right]^2}, \tag{8.3}$$

where $V\left[\{< S(x_i, y_i) > /N_i\}\right]$ is the experimental variance of the $< S(x_i, y_i) > /N_i$ values. Letting

$$< E\left[S(X, Y)\right] >= \frac{1}{N_p} \sum_{i=1}^{N_p} \frac{< S(x_i, y_i) >}{N_i} \tag{8.4}$$

be the arithmetic average estimation of $E\left[S(X, Y)\right]$ from the set of samples $\{< S(x_i, y_i) > /N_i\}$, the experimental variance is given by:

$$V\left[\left\{\frac{< S(x_i, y_i) >}{N_i}\right\}\right] = \frac{1}{N_p - 1} \sum_{i=1}^{N_p} \left(\frac{< S(x_i, y_i) >}{N_i} - < E\left[S(X, Y)\right] >\right)^2. \tag{8.5}$$

Note that here we implicitly consider the luminance of the signal values, as we need a scalar value, and that the definition domain of X and Y, the screen coordinates random variables, corresponds to the support of the pixel's filter.

As the variance decreases linearly with the number of samples, sampling according to $e_r(I_p)$ tends to make the error uniform over the pixels. $E[S(X,Y)]$ being unknown, we need to estimate it. A natural estimator of $E[S(X,Y)]$ is $< E\left[S(X,Y)\right] >$. Using this estimate yields an arithmetic error estimate $e_a(I_p)$:

$$e_a(I_p) = \frac{V\left[\left\{\frac{<S(x_i,y_i)>}{N_i}\right\}\right]}{N_p \times < E\left[S(X,Y)\right] >^2}. \tag{8.6}$$

However, and as we have presented in details in Chapter 6 when few samples are much larger than the actual estimate, $< E\left[S(X,Y)\right] >$ largely over-estimate $E\left[S(X,Y)\right]$, and therefore $e_a(I_p)$ largely under-estimates $e_r(I_p)$, leading to under-estimated probabilities for pixels being bright spots.

Although the enhanced estimator of Chapter 6 could be used, we here define a much simpler robust average estimator to compute $E\left[S(X,Y)\right]$. Its bias is much larger than the one of Chapter 6, making it unpractical for accumulation, but it is adapted for robust adaptive sampling. This estimator is based on approximate median. For each pixel, we compute an approximate median M_s of the samples $< S(x_i,y_i) > /N_i$ which are in a neighborhood of width h (h being the width of the filter's support), and use it to compute a robust error measure $e_m(I_p)$:

$$e_m(I_p) = \frac{V\left[\left\{\frac{<S(x_i,y_i)>}{N_i}\right\}\right]}{N_p \times M_s^2}. \tag{8.7}$$

We compute M_s as the average of the medians computed on small chunks of N_c elements of the sequence $\{< S(x_i,y_i) > /N_i\}$. We use $N_c = 10$, which makes the per-pixel memory requirements low, while ensuring a robust and efficient error estimate. When M_s is 0, we resort to the standard $e_a(I_p)$ error measure. Taking all the samples in a neighborhood with the same size as the reconstruction filter allows us to naturally handle visual edges, being caused by geometry, textures, shadows, caustics, *etc.*

Figure 8.1 shows the ratio of errors $e_a(I_p)/e_r(I_p)$ (blue) and $e_m(I_p)/e_r(I_p)$ (red) in a context where the sampling quality ranges from adequate to poor, by varying both the number N_o of sampled outliers (different curves), and their value (abscissa). We can see that $e_m(I_p)$ remains a good approximation of $e_r(I_p)$ even for large outlier values, while $e_a(I_p)$ largely under-estimates the actual error even for low outlier values. Using $e_a(I_p)$ would lead to a nearly zero pixel selection probability, while the actual error is very large and therefore requires a lot of samples to vanish.

Alternating between uniform and adaptive sampling: Instead of using adaptive sampling and recomputing the probabilities every N_a samples, we alternate between adaptive sampling for N_a samples, and uniform pixel selection for N_u samples, with N_u larger than the number of pixels in the image. The error estimates are then updated once the $N_a + N_u$ samples have been computed. This ensures that all error estimates receive samples, while still focusing on pixels with larger errors.

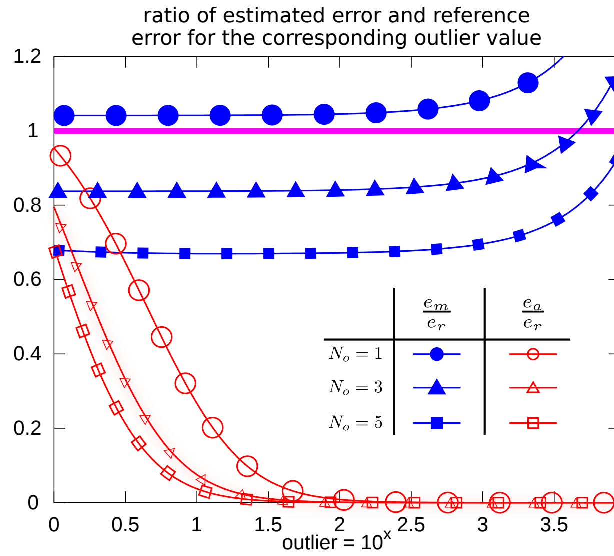


Figure 8.1: Ratio of estimated error and reference errors for increasing outlier values (abscissa) and increasing number of sampled outliers (curves). The magenta line depicts the perfect estimation.

Complete adaptive sampling algorithm:

1. a fixed number of samples (for instance two) are shot per pixel. As we use a neighborhood of pixels to evaluate the error at each pixel, only a few samples are required to begin using adaptive sampling.
2. Compute error estimates. Compute a maximum error such that 95% of the computed errors are below. This avoids focusing processing power on few pixels with very inaccurate and over-estimated errors. Set the pixel probabilities accordingly to the clamped errors.
3. Compute $N_a + N_u$ samples, using adaptive and uniform sampling. For each sample, update the data required for the computation of the error estimates. Loop back to step 2.

8.3 Results

8.3.1 Robustness to outliers

Figure 8.1 shows the result of our experiments assessing the robustness of our median-based error estimate, as well as the lack of robustness of the standard measure error. The experiments are based on a discrete random variable $X(e)$, containing 10^6 different real values. All the possible values of this random variable are obtained using a uniform pseudo-random number generator, which generates values between 0 and 1. Only 10 values, the outliers, are set to 10^e. $X(e)$ can be seen as a discretized version of the continuous random variable which is used when evaluating the light transport equations. When $e = 0$, $X(e)$ is close to a case where no low-probability direction ω yields a large contribution $f(\omega)$. The larger e, the larger the contribution of some of these low-probability directions. In our experiments,

the quantity to estimate is $E[X(e)]$ – similarly to rendering where we also estimate expected values – , and we use 20 samples to do so. In these 20 samples, we force N_o samples to be outliers. This yields an estimate $I(e, N_o)$. As we want to assess robustness to outliers, these N_o outliers are here to simulate from acceptable to poor sampling. As $X(e)$ is known, we can compute the reference error estimate $e_r(I(e, N_o))$, and observe the behavior of e_a and e_m relatively to the reference. The closer to it, the better. This relative behavior is shown in the graphs, for $e \in [0, 4]$ (in abscissa) and for N_o equals $1, 3$ and 5 (each curve).

Asymptotic behavior: Figure 8.1 suggests that our method overestimates the error more and more when the outlier increases above 10^4. In fact, when reaching 10^5, the error ratio is approximately 7. This is because the exact mean begins to be dominated by the outliers: there are 10^6 values of average 0.5, summing to 5×10^5, and there are ten outliers of value 10^5, summing to $10^6 > 5 \times 10^5$. Meanwhile, the approximate-median average remains unchanged because the number of outliers does not increase. That being said, e_a/e_r continues to converge toward 0 as the outlier value increases toward infinity, as the arithmetic average of the 20 sampled values increases much faster than $E[X(e)]$.

8.3.2 Adaptive sampling evaluation

Three different tests have been performed on specific scenes, in order to evaluate the impact of each element of our method.

Uniform sampling alternation: Figure 8.2 illustrates the robustness brought by alternating between uniform and adaptive sampling. The scene exhibits very large soft shadows which are prone to noise, and lead to poor error estimations when few samples are used. Alternating between adaptive and uniform sampling is crucial to have a correct adaptive sampling. In penumbra regions and when few samples are used, it is common for a pixel to have all its samples occluded, leading to a zero value and a zero error. With pure adaptive sampling, the penumbra remains mostly not sampled while with alternation, a sample eventually reaches the light and yields a correct error estimation.

Correct sample placement from relative variance: Figure 8.3 presents a scene containing complex geometric meshes with high-frequency features, a detailed asphalt texture on the floor, and detailed volumetric data (measured density of a car). The image is computed using direct lighting from a physical sky model. This scene exhibits strong dynamic reduction due to tone-mapping: the shadowed zones have significantly lower illumination values than the directly-lit parts of the car, while after tone-mapping these values are closer. The use of a relative error is therefore important to ensure a sampling adapted to the visual content of the image. This scene also exhibits several visual edges formed by textures, geometry or shadows. An accurate anti-aliasing thus requires an error evaluation based on all the samples in a neighborhood of each pixel.

In this test, we have compared our method to standard uniform sampling and Tsallis-entropy-based adaptive sampling [XSXZ07] (abbreviated “Tsallis entropy” in the remaining). In its standard form,

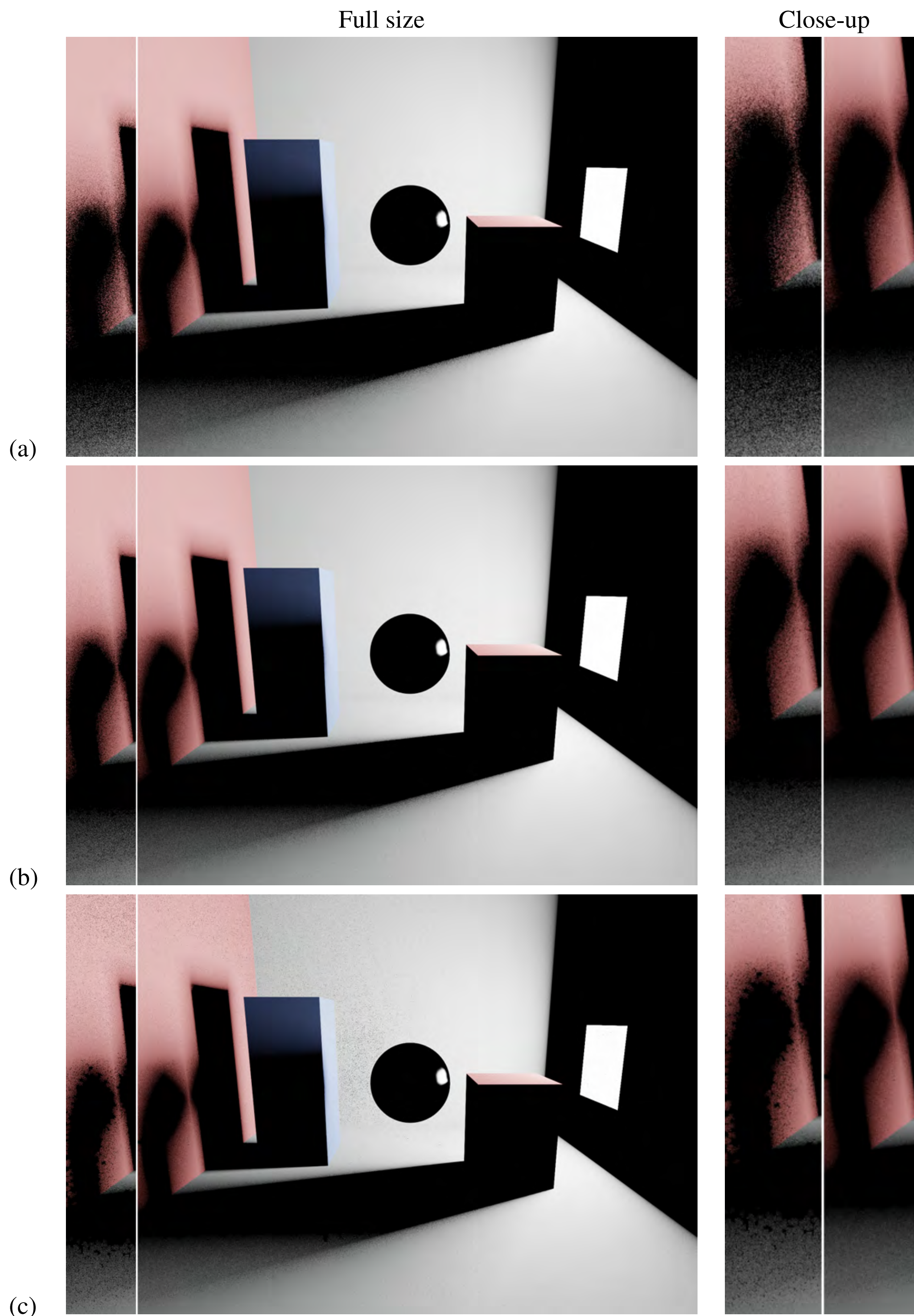


Figure 8.2: Same scene rendered without adaptive sampling (a), with adaptive sampling using alternation, setting $N_u = N_a$ (b), and with adaptive sampling but no alternate uniform/adaptive sampling (c). Every image has been computed at a resolution of 1600×1200 pixels, using the equivalent of 100 samples per pixel. “Full size” column: The left-most part of each image is the left border of the rendered image after the computation of 12% of the samples. Alternation largely improves the results in penumbra zones.

this algorithm is quite sensitive to the values of its entropic index parameter q. We have experimentally set it to 3. Moreover, the convergence threshold is not intuitive, and leads to a kind of multi-stage uniform sampling (uniform sampling on a set of pixels whose size decreases). Therefore, when stopping the rendering at an arbitrary moment, some zones of the picture will be perfectly converged, while some others will remain not converged at all. In contrast, our method favors uniformity in the visual appearance, allowing us to stop rendering at any time. We experimentally found that using a sequence of thresholds 0.999, 0.9999, *etc.*provides a similar behavior for Tsallis entropy on this scene, although it remains highly perfectible.

Figure 8.3 also presents noise-measure images. These images have been obtained in two steps. First we compute the difference between the (tone-mapped) test image and a (tone-mapped) reference image obtained with a large number of samples. For each pixel of this difference image, we compute the variance of the pixels values located in a 5×5 neighborhood. Using a difference with a reference image allows us to measure the actual variance produced by the noise. In addition, as the images are tone-mapped, it focuses on visually apparent noise. The final noise-measure images are obtained by normalizing the variance values with the maximum variance over the three images. These noise measures clearly show that the use of a relative variance measure yields to results that are comparable with Tsallis entropy (the noise is not distributed the same way but has a similar magnitude in average), and brings a clear improvement over uniform sampling.

Bright-spots and approximate median: The scene presented in Figure 8.4 is prone to bright-spots when rendered with path-tracing. The error based on standard average estimator ($e_a(I_p)$) distributes the processing power relatively uniformly over the pixels, while our method correctly focuses on the bright spots, even though we clamp the errors at step 2 of the algorithm.

8.4 Conclusion

Alternation between uniform and adaptive sampling is an easy way to make any adaptive sampling algorithm more robust. Combined with relative variance-based errors and approximate-median, it leads to an algorithm which behaves well in a wide variety of situations, correctly focusing on visual edges, visually apparent noise and correctly handling difficult cases such as those presented in Figure 8.2 and Figure 8.4. Our algorithm is very simple to implement, has a negligible per-sample overhead, leads naturally to uniform error over the image at any moment and does not have any sensitive parameter or hard-to-set thresholds. This is why we use it as adaptive sampling algorithm in our rendering engine.

So far, mathematical robustness has been improved to limit poor importance sampling, bright-spots in images, and ensure that adaptive screen sampling is correct in as many cases as possible. All these improvements have been done without considering the nature of the representations used to describe the scene. In the set of entities described in Chapter 2, there is one element which can have a large impact on the robustness of the computations: the representation of participating media, because integration has to be done to evaluate transmittance (Equation (1.58)). In the next chapter, we provide a way to represent arbitrary participating media, to ensure that they are handled in an efficient and stable way when using

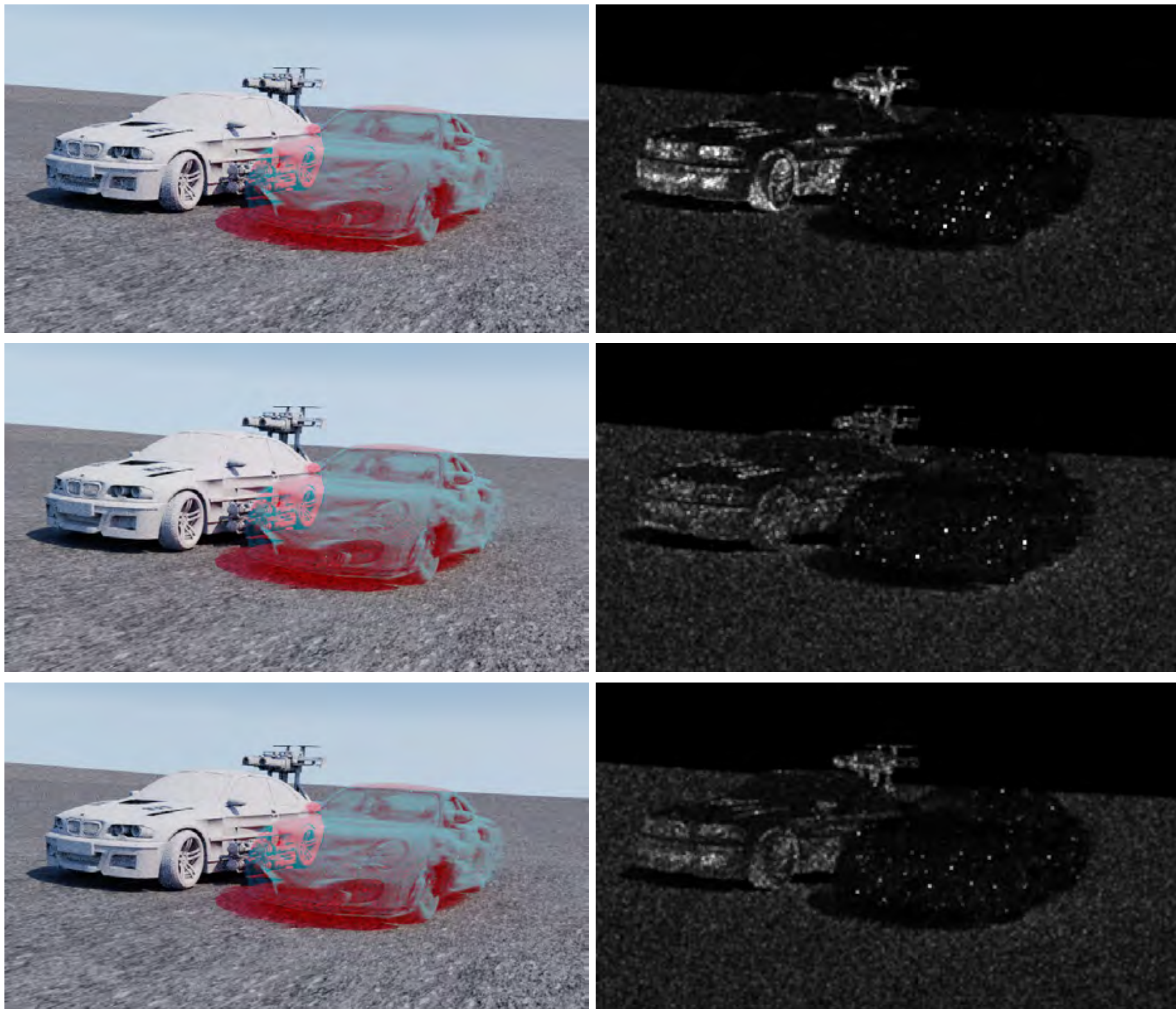


Figure 8.3: Comparison between uniform sampling, Tsallis entropy and our method after the equivalent of 12 samples per pixel have been shot, with a uniform sample taken every two adaptive samples, both for Tsallis entropy and our method. Each row is composed of the computed image on the left and the noise measure on the right (the whiter the more noise there is). First row: standard uniform sampling. Second row: with Tsallis entropy. Third row: with our method. Note that the results between our method and Tsallis entropy, both using alternation, are comparable, while our method naturally leads to uniform error for time-constrained computations.

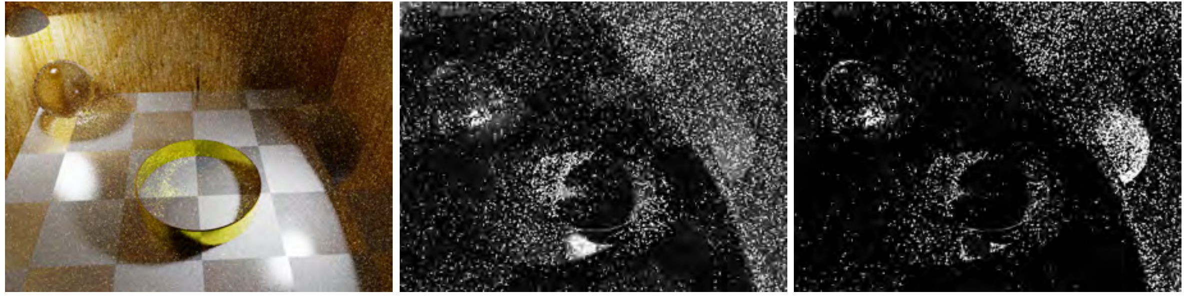


Figure 8.4: Left: image obtained after 10 uniform samples per pixel have been shot. Middle: adaptive sampling probabilities obtained using standard average estimator. Right: adaptive sampling probabilities obtained using approximated median.

Monte-Carlo integration.

9

Globally adaptive control variate for least-square kd-trees on volumes

9.1 Introduction

The evaluation of the radiative transfer equation (Equation (1.57)) or its generalization over path-space (Equation (3.8)) using Monte-Carlo integration requires the ability to perform free-path sampling (Section 3.7.1) and compute transmittance accurately along arbitrary rays on any type of participating media. However, while different models can be used for their representation, *e.g.* simulation results stored in grids [PH04], procedural models [Ebe97], analytical models [EMP$^+$02], *etc.*, none combines at the same time generality, low memory usage – allowing for instance the use of GPUs as co-processors –, and low computation times for sampling or transmittance evaluation. Such a representation, unified, efficient and versatile enough to represent any participating medium, would be especially well adapted to ensure accuracy and computation times stability for the processing of participating media in a rendering engine.

In general, existing unified representations split the medium in regions in which it is approximated by a given function (*e.g.* a constant). As reviewed in Section 9.2, these representations exhibit at least one of the two weaknesses:

- The approximation error is difficult to control, leading to costly trial-and-error cycles when a user tries to find the model parameters generating a desired result.
- Structures lack of adaptability, *i.e.* the number of subdivisions is far from optimal. This yields unnecessary large memory footprints, which, in case of large models, prevent the structure to fit on a GPU, and thus avoid more efficient accelerations.

Contributions: This chapter has two contributions. The **practical contribution** (Section 9.3) is an algorithm to build a kd-tree to represent arbitrary isotropic participating media with a minimal number of nodes with respect to a given reconstruction error. The kd-trees have therefore low memory footprints and low computational costs for physically-based rendering. This algorithm requires an efficient and robust numerical integration method that must fulfill the following constraints:

- Nearly no analytical properties on the functions to integrate (called *integrands*) are available. They should be bounded, but continuity is not guaranteed.
- No asumption can be made on the frequency content, the range of values, and the size of integration domains. Scale-dependent parameters such as grid resolutions should therefore be avoided.
- The accuracy of the estimation must be controllable. This is mandatory to get stable results from one run to another.
- 95% confidence intervals must be computed (Section 9.3.2).

As presented in Section 9.2, state-of-the-art integration methods fail to fulfill all these requirements. We thus propose, as **theoretical and main contribution** of this chapter, an appropriate numerical integration method (Section 9.4) called *globally adaptive control variate*. This method effectively couples deterministic quadrature rules and globally adaptive subdivision methods from standard Monte-Carlo estimation. During integration, our method locally refines an approximant kd-tree of the function based on the information obtained from the Monte-Carlo estimators. Combined with globally adaptive subdivision strategies for adaptive sampling, this yields an integration method with the following properties.

- It efficiently handles scalar and vector-valued integrands, integrated over arbitrary low-to-middle-dimensional domains.
- The estimator is unbiased and gives accurate confidence intervals.
- Its complexity is sublinear with respect to variance.
- It does not rely on any analytic property of the integrand.
- Computation times and memory consumption are very stable for a same integrand, and can be easily and accurately controlled.
- It is simple to implement, and has few and simple intrinsic parameters.

Our results (Section 9.5) show the high robustness, accuracy and efficiency of our numerical integration algorithm, as well as a much improved computational stability compared to other state-of-the-art algorithms. The results also illustrate the versatility of our participating media representation. For instance, it shows that we can compute a low-error approximation of a 4.6GB grid composed of 193 millions cells, with a 7.5MB multi-resolution kd-tree of $500,000$ nodes. We finally examine in Section 9.6 possible extensions or alternative uses of our method, which further exploit the generality of the algorithms developed in this chapter.

9.2 Related works

9.2.1 Participating media representations

Several unified representations already exist, all of them approximating the original medium using space subdivision strategies. Grids [PH04] are a natural representation: inside each voxel, the medium data can

be approximated by constant values, or any other reconstruction function. They are simple to implement, can be built rapidly, are very efficient for point-evaluation, and are easy to traverse along a ray for analytic integration and inversion. However, they lack robustness: as their resolution is constant, they can not automatically and locally adapt to frequency variations inside the medium. Small details are not captured, unless the grid granularity is increased, leading to a very large memory consumption and long traversals.

Octrees [Kno06] can be controlled directly by the quality: subdivision is applied until the quality is satisfactory. They provide a natural multi-level representation. However, although more adaptive than grids, they still exhibit adaptability problems as the splitting positions are fixed, and the size of the voxels is deterministic: very small details inside otherwise constant regions may require many "useless" subdivision levels, which leads to long traversal times for point-evaluation and rays.

9.2.2 Numerical integration

Methods targeting low-to-middle-dimensions numerical integration can be broadly split into two categories: deterministic and stochastic methods.

Deterministic methods: A large number of deterministic integration methods are based on so-called quadrature or cubature rules. Interested readers can refer to [DR07]. These methods build a precise analytically-integrable approximation of the integrand. Even though fast to evaluate, the integration with a high precision of high-frequency functions over large and high-dimensional domains can lead to extremely important memory consumptions. Additionally, an accurate estimation of the error is difficult to obtain. Another kind of deterministic methods is based on Quasi-Monte-Carlo (QMC) sampling, which we present below. For a given integrand, integration domain and precision, deterministic methods always give the same result. An important consequence is that failure cases (inaccurate result but small computed error) are difficult to detect automatically without already knowing the reference value.

Stochastic methods: These methods are based on the Monte-Carlo estimator. Even though very general, the standard Monte-Carlo estimator is computationaly inefficient for complex integrands. It remains thus the core of several improved methods. The numerical integration library CUBA, developed by physicists, contains two reference state-of-the-art methods called SUAVE and DIVONNE [Hah05]. These methods can be used to integrate, with an estimation of the error, any low-to-middle dimensionality integrands defined over the unit hyper-cube at a given absolute or relative precision, without requiring any a-priori information. An integrand with an arbitrary integration domain has thus to be mapped so that its integration domain is the unit hyper-cube.

The Monte-Carlo estimator evaluates an integral as the mean value of weighted integrand evaluations done at randomly generated samples of the integration domain. The efficiency of Monte-Carlo estimation can be measured by its variance, and the variance highly depends on the sample distribution. Standard methods to improve Monte-Carlo estimation adapt the sample distribution to lower this variance. Monte-Carlo estimation makes use of random numbers obtained through pseudo-random number generators (PRNG), generally distributed uniformly on the $[0, 1)$ interval. The practical uniformity of these random numbers is of importance for robustness, and a high-quality distribution can lead to major variance

reduction. QMC methods [Lem09] replace PRNGs by deterministic sequences, whose uniformity is largely improved compared to standard PRNGs. All methods in CUBA can use either PRNGs or QMC.

Two standard methods amongst others exist to improve the samples distribution: adaptive sampling and importance sampling. Adaptive sampling relies on the property of additivity of integrals, by splitting the integration domain into regions and estimating the integral in each region independently. By subdividing more where the integration error is larger, samples are focused on the regions where the integrand is harder to integrate. Importance sampling reduces the variance of an estimator by generating samples according to the integrand values: more samples where the integrand is larger (Section 3.5.1).

SUAVE combines these two methods. First, it uses a globally adaptive subdivision to achieve adaptive sampling. A globally adaptive subdivision splits the integration domain according to an integration error measure, in a progressive way, until the total error or variance is below a desired threshold. At each step, the region of the integration domain with the largest error is subdivided, and a new estimation is computed in each sub-region. Second, for each estimation, the VEGAS algorithm [Lep78], which performs importance sampling, is used to compute the integral and the integration error. VEGAS builds iteratively a separable approximation of the integrand using piecewise-constant functions, and uses it for sampling. Note that in our case of vector-valued integrals, correlation amongst the components of the integral should be present to get a correct sampling for all components at once.

The SUAVE algorithm has few parameters, and default values are given in the paper for the Mathematica implementation. However, our numerical tests presented in Section 9.5.1 show that SUAVE has major flaws. First, its computation times and memory consumptions are not stable: they vary a lot from an integrand to another, and even for a same integrand when using a PRNG to generate the uniform numbers used. Second, for some integrands, the method leads to extremely large computation times (in the order of ten minutes against ten seconds for our method) and unacceptably large memory consumption (ten gigabytes of memory against 47 megabytes for our method). Third, this method is not well adapted for arbitrary vector-valued integrals, especially when correlation between the components is low. Fourth, our tests on complex participating media functions, presented in Section 9.5.1, show that this method can have robustness problems with functions with large almost constant parts, which in our case lead to largely underestimated integrals. All these flaws are a mark of a global lack of robustness, being mathematical or computational, which is highly impairing when handling arbitrary vector-valued integrals.

DIVONNE relies on numerical optimization methods to find peaks of the integrand and sample them accordingly. This method has a lot of intrinsic parameters which depend on the integrand, making it difficult to use. More precisely, finding a set of parameters which work well for all integrands seems difficult, as for instance some of these parameters are linked to the smoothness of the integrand. An illustration of this problem is that the default parameters values given in [Hah05] give good results for some functions both in terms of accuracy and rapidity, but fails for others. More specifically, these parameters are well adapted for the Genz test functions [Gen87], which are used in [Hah05] to assess the accuracy and robustness of the method. However, it gives values between 1.08 and 1.45 after large computation times when estimating several times an integral of practical interest whose analytical value is known to be 1.7035, with a required precision of 0.001. As we want to compute arbitrary integrals without requiring

any prior information and therefore without having to set integrand-dependant parameters, we decided to avoid the use of DIVONNE.

Other approaches do not rely on *a priori* information. [PWP08] uses a control variate based on a low-order approximation of the function reconstructed in a grid. The use of a grid prevents the method from handling integrands with dimensionalities larger than 3 or 4. Moreover, the grid size must be computed *a priori*, which greatly reduces the robustness of the method with respect to very different integrands. A large number of Monte-Carlo integration methods rely on local exploration with samples mutations for improving the estimation [DGMR05, MRR$^+$53]. However, these methods use dependent samples to perform the estimation, which makes the precision difficult to compute.

9.3 Error-guided kd-tree

For clarity, our participating medium is defined by a scalar absorption coefficient σ_a and a scattering coefficient σ_s, into a function f at point p:

$$\begin{aligned} f \quad : \quad & \mathbb{R}^3 \to \mathbb{R}^2 \\ & p \to (\sigma_a(p), \sigma_s(p)). \end{aligned} \tag{9.1}$$

Section 9.3.4 shows how to trivially handle RGB and spectral coefficients.

Algorithm 2 Construction algorithm.

1: **proc** node = buildSubtree (cr, f) :
2: Node node
// If the cell representation is precise enough, create a leaf
3: **if** cr.error $< \epsilon$ **then**
4: node = makeLeaf (cr)
5: **return** node
6: **end if**
// Otherwise, find a good split plane for the node (Algorithm 3)
7: split = findBestSplit (cr.bbox, f)
// Does this split plane bring a noticeable improvement ? (Section 9.3.2)
8: **if** split.upperBoundCost $>$ cr.error **then**
9: split = splitInMiddle() *// errors have to be estimated for each child of split in splitInMiddle()*
10: **end if**
11: node = makeIntermediate (split)
// Build the sub-tree for each side
12: node.left = buildSubtree (split.leftCR, f)
13: node.right = buildSubtree (split.rightCR, f)
14: **return** node

In this discussion, our kd-tree approximates the medium in each node/leaf using a constant *cell representation* (CR). A CR is the reconstruction function used for the region covered by the node.

Our kd-tree construction algorithm operates by recursively splitting the 3D space containing the participating medium, using a standard top-down approach (Algorithm 2). Each node has an associated

CR, for which a quadratic reconstruction error is computed (Section 9.3.1). When this error is too large, we find the best split-plane (Algorithm 3) by minimizing a cost function based on the reconstruction error (Section 9.3.2). This yields a tree with leaves whose error is below a given threshold. By iteratively lowering this threshold, we can enforce global error constraints (Section 9.3.3). As each intermediate node has a cell representation with a known error, any approximation with a larger error than the one of the tree can be obtained by cutting through the kd-tree. Although the discussion in this section is specialized to the case of scalar coefficients approximated by constant CRs, Section 9.3.4 shows that it is easy to handle more general coefficients representations and CRs.

Algorithm 3 Best split-plane computation.

1: **proc** split = findBestSplit (bbox, f) :
2: Split bestSplit[3]
// On each axis, find the best split-plane (Section 9.3.1 and Section 9.3.2)
3: **for** each axis $\in \{x, y, z\}$ **do**
4: Split splits[N]
5: Δ =bbox.extent(axis) / $(N+1)$
6: **for** $i = 1$ to N **do**
7: splits[i].plane = (axis, bbox.min(axis) + $i \times \Delta$)
8: splits[i].computeLeftAndRightCR(f)
9: splits[i].estimateErrorEachSide(f)
10: splits[i].computeCost()
11: **end for**
12: bestInd = $\underset{i}{\text{argmin}}\,(\text{splits}[i].\text{cost})$
13: bestSplit[axis] = locallyMinimize (bestInd, splits, f)
14: **end for**
// Return the candidate with minimum cost over the three axes.
15: bestAxis = $\underset{i}{\text{argmin}}\,(\text{bestSplit}[i].\text{cost})$
16: **return** (bestSplit[bestAxis])

9.3.1 Error measure

We use quadratic error E to quantify the local reconstruction error committed by using CR h (constant, linear, *etc.*) instead of the original function f on region $\mathcal{R}$:

$$E = \int_{\mathcal{R}} (h(p) - f(p))^2 \, dp. \tag{9.2}$$

For the quadratic error, the optimal h minimizing E is the solution of a least-square problem. For instance if h is a constant, the optimal solution is the average of the function over $\mathcal{R}$. As Equation (9.2) cannot be computed analytically for arbitrary f and h functions, it is estimated using our adaptive control-variate method developed in Section 9.4.

9.3.2 Cost minimization

Cost function: A split plane separates a node n in left and right children. The left child is represented with CR l and the right child with CR r.

For absorption and scattering, the reconstruction error committed by using l and r is the sum of their reconstruction errors. In order to select a split plane minimizing the error both for absorption and scattering, we define the cost function as

$$cost(n, l, r) = \max(E_a^l + E_a^r, E_s^l + E_s^r) \tag{9.3}$$

where E_a^l (resp. E_s^l) is the absorption (resp. scattering) error on the left child, and E_a^r (resp. E_s^r) on the right child.

Minimization: Finding the best split-plane t along an axis is done by selecting t such that:

$$t = \underset{t'}{\operatorname{argmin}} \left(cost(n, l(t'), r(t')) \right) \tag{9.4}$$

where $l(t')$ and $r(t')$ are the CRs of the child nodes obtained when splitting at t'. We solve Equation (9.4) in three steps, using a local optimization method, as follows:

1. Evaluate the cost at regularly spaced planes t (Line 7 of Algorithm 3, $N = 3$ by default).
2. Select the plane with minimal cost.
3. Improve the plane location with a derivative-free local minimization process[Cha98] (Line 13 of Algorithm 3). This process iteratively reduces a search interval using a quadratic approximation of the cost function. It stops when the cost function variation over the current interval is below a user-defined threshold ϵ_f, or when the interval size is below a user-defined threshold ϵ_s. We suggest to set $\epsilon_f = 10^{-3} \times c_b$, where c_b is the minimum cost value found during the first step, and $\epsilon_s = 10^{-4} \times s$, where s is the size of the volume along the current axis.

This plane selection is performed on each axis, and we finally split along the axis with the lowest cost.

Dealing with nearly constant costs: In some nodes, the cost function can be almost constant, *i.e.* no split plane is best. In this case, we simply balance the tree by splitting at the middle of the longest axis. In order to detect this situation, we first compute an upper bound of the cost, from the upper bounds of the 95% confidence intervals of the error estimates (these intervals are computed by the integration method presented in Section 9.4). If this upper bound is larger than the maximal value between the parent's absorption and scattering errors, the cost function is considered as almost constant. This has proven very efficient to remove thin nodes which are not improving the reconstruction while generating poor sub-trees.

9.3.3 Enforcing global error from local errors

A simple yet robust way to ensure that the global reconstruction error is below ϵ is to build the tree iteratively using decreasing local errors ϵ_i.

We first build a tree with a local error $\epsilon_0 = \epsilon$. The global error is then the sum of the local errors of the leaves of the tree. If it is below ϵ, then the construction is finished. Otherwise the tree is refined until a new local error $\epsilon_1 = \epsilon_0/2$ is reached. The process iterates with $\epsilon_i = \epsilon_{i-1}/2$ until the global error is below ϵ.

9.3.4 Generalization

The complete discussion about the kd-tree construction has been specific to the case where the absorption coefficient as well as the scattering one are described by a single constant value inside a region.

Coefficient representation: Handling more complex representations such as RGB coefficients or spectral distributions is straightforward. Letting N be the number of components of this representation (3 for RGB), there are $2N$ components to represent absorption and scattering. The quadratic error E, given by

$$E = \int_{\mathcal{R}} (h(p) - f(p))^2 \, dp, \tag{9.5}$$

is computed for each of these $2N$ components, using our adaptive control-variate method. As this method directly handles vector-valued integrals, a single estimation is necessary. This gives E_i^l and E_i^r for component i. The cost is obtained as the maximum of the errors sum over these $2N$ components:

$$cost(n, l, r) = \max_{i=1..2N} (E_i^l + E_i^r). \tag{9.6}$$

Cell representation: More complex cell representations than a constant approximation can be used, as long as optimal parameters in the least-square sense can be found. We illustrate this point by using a linear representation based on plane equations, which further assumes that the different components (RGB, spectral values, *etc.*) of each coefficient (absorption and scattering) are correlated. We therefore have one plane equation for absorption, and one for scattering. For scattering, the coefficients approximation is given by:

$$\sigma_s(p) \simeq \max(0, \mathcal{N}_s \cdot p + o_s) \times \bar{\sigma_s}, \tag{9.7}$$

where $\mathcal{N}_s$ is the "normal" of the scattering plane, o_s an offset value, and $\bar{\sigma_s}$ is the average of the exact σ_s coefficients in the region. With this definition, we approximately solve the least-square problem by first computing $\bar{\sigma_s}$, and then finding the optimal plane equation for a set of sample points using the singular value decomposition (SVD) method.

9.4 Globally adaptive control variate

9.4.1 Overview

Most of the construction time of the kd-trees are spent computing values of the form:

$$I = \int_{\Omega} g(p)dp \tag{9.8}$$

where g is an arbitrary function from $\Omega \subset \mathbb{R}^D$ to $\mathbb{R}^N$. In the following, we say that g (and its integral) has N components, which are noted $g_1, \ldots, g_N$, and $g = (g_1, \ldots, g_N)$. In the following, Ω is assumed to be a compact (bounded) axis-aligned subset of $\mathbb{R}^D$, which is always the case for the kd-tree construction. We show how to handle unbounded and/or non-axis-aligned subsets of $\mathbb{R}^D$ in Section 9.4.7.

We present the globally adaptive control variate (GACV) algorithm, which computes an estimate $< I >$ of I. The precision of the estimation is controlled through confidence intervals, of the form $P(|< I > -I| < \epsilon) = 95\%$ for the N components of the integral, where ϵ is determined from relative and absolute precision requirements given by the user (Section 9.4.3). As shown in Algorithm 4, we use a globally adaptive subdivision strategy, similarly to the SUAVE algorithm. At each step of the algorithm, the region with largest error is subdivided along the longest axis into two equally-sized sub-regions, and an estimation is done for each sub-region. This is efficiently done through the use of a heap.

The subdivision process generates an estimations tree, whose leaves form a partition of Ω, and are used to obtain the final estimation. Section 9.4.2 presents the estimation of the integral inside a region $\mathcal{R}$ of Ω, which is the main contribution of our method. To obtain an accurate, robust, and efficient estimator, we use a combination of

- a new locally-refinable approximation of the integrand, detailed in Section 9.4.4,
- control-variate-based estimation,
- standard Monte-Carlo estimation,
- a version of stratified sampling suitable even for large D values and non-cubic regions (Section 9.4.5).

We additionally show that all the estimations of the leaves of the tree are unbiased and independent, and that their distribution can be made close to a Gaussian distribution. The globally Gaussian nature of the leaf estimations, and more specifically the variance information, allows us to derive the error used to choose the estimations to split, the variance of the complete estimation (Section 9.4.6) and the actual convergence criterion used (Section 9.4.3).

Global estimation and variance: As all the leaf estimations can be assumed to have a Gaussian distribution, the complete estimation and its variance are obtained by summing each leaf estimate and variance, and has itself a Gaussian distribution with mean value I. This means that our algorithm is theoretically *unbiased*. The Gaussian distribution of the result is confirmed to a large extent by our tests (Section 9.5).

Algorithm 4 Top-level integration routine, computes an estimate $< I >$ of I, and the size of the 95%-confidence interval w_C of this estimate.

1: **proc** ($< I >$, w_C) = integrate (g, Ω, ϵ_a, ϵ_r):
2: $\hat{g}$ ←initialize_acv(g, Ω, ϵ_a, ϵ_r)
// make a first estimate
3: $(< I > (\Omega), \sigma^2(\Omega)) \leftarrow$ estimate(g, $\hat{g}$, Ω)
// Add the estimate to an error-based heap for zone selection.
4: $E \leftarrow \max_{n=1...N}(\sigma_n^2)$
5: pushHeap $((\Omega, < I > (\Omega), \sigma_2(\Omega)), E)$
// For scale-independent error computations (Section 9.4.6)
6: errorScaling $\leftarrow 1$, stepNumber $\leftarrow 1$, stepUpdateError $\leftarrow 1$
7: **loop**
8: *// test convergence*
9: $\sigma^2_{max} = \frac{max(\epsilon_a, \epsilon_r \times |<I>|)^2}{16}$
10: **if** $\sigma^2 < \sigma^2_{max}$ for all components **then**
11: *// The estimate satisfies the convergence criterion, finished*
12: **return** $(< I >, 4 \times \sqrt{\sigma^2})$
13: **end if**
14: *// Take the estimate with largest error and improves it*
15: $(\mathcal{R}, < I > (\mathcal{R}), \sigma^2(\mathcal{R})) \leftarrow$ pop_heap()
16: $(\mathcal{R}_1, \mathcal{R}_2)$ ←split($\mathcal{R}$)
17:
// Remove the estimate from this region
18: $< I >=< I > - < I > (\mathcal{R})$
19: $\sigma_2 = \sigma^2 - \sigma^2(\mathcal{R})$
20:
// Potentially update the error computation (Section 9.4.6)
21: **if** stepUpdateError = nbSteps **then**
22: errorScaling $\leftarrow 1/\sigma^2_{max}$
23: recomputeHeap(errorScaling)
24: stepUpdateError $\leftarrow$ stepUpdateError $\times 2$
25: **end if**
26: **for** $i \in \{1, 2\}$ **do**
27: *// Perform the estimation (Section 9.4.5)*
28: $(< I > (\mathcal{R}_i), \sigma^2(\mathcal{R}_i)) \leftarrow$ estimate(g, $\hat{g}$, $\mathcal{R}_i$)
// Add the estimate for this region to the global estimate
29: $< I >=< I > + < I > (\mathcal{R}_i)$
30: $\sigma_2 = \sigma^2 + \sigma^2(\mathcal{R}_i)$
// Add this estimate to the heap, for further refinement
31: $E_i \leftarrow \max_{n=1...N}(\sigma_n^2(\mathcal{R}_i) \times$ errorScaling$_n)$
32: push_heap $((\mathcal{R}_i, < I >, \sigma^2), E_i)$
33: **end for**
34: stepNumber $\leftarrow$ stepNumber $+1$
35: **end loop**

Note that the two global sums (global estimate and variance) are updated on the fly in Algorithm 4. From an implementation point-of-view, using double precision for all the components of these sums is mandatory to avoid numerical instabilities.

9.4.2 Estimations

Control-variate-based integration makes use of the following identity:

$$I(\mathcal{R}) = \int_{\mathcal{R}} (g(p) - \hat{g}(p))dp + \int_{\mathcal{R}} \hat{g}(p)dp \tag{9.9}$$

where $\hat{g}$ is an analytically integrable function approximating g, and is called *control variate*. The more constant $g(p) - \hat{g}(p)$ is, the lower the number of samples required to estimate the first integral with Monte-Carlo. For our estimator, we use stratified sampling to enhance the robustness of the estimator, and further reduce variance. Using N_p independent passes of stratified sampling with S strata (we show how we choose N_p and the S strata in Section 9.4.5) yields our control-variate-based estimator:

$$\begin{aligned} < I_c > (\mathcal{R}) &= \frac{1}{N_p} \sum_{i=1}^{N_p} \left(\frac{V(\mathcal{R})}{S} \sum_{s=1}^{S} (g(P_{i,s}) - \hat{g}(P_{i,s})) \right) \\ &\quad + \int_{\mathcal{R}} \hat{g}(p)dp. \end{aligned} \tag{9.10}$$

Equation (9.10) is an effective way of mixing Monte-Carlo estimation for high-frequencies of g, and deterministic quadratures for its smooth part. As detailed in Section 9.4.4, our control variate is based on a kd-tree, each leaf being a first-order approximation of the function on the region covered by the leaf. In the litterature, the control variate is built before any integration is done [FCH+06]. The key difference of our method is that we refine this kd-tree locally based on the global subdivision scheme *during* integration, allowing us to tightly approximate the function where needed.

The major drawbacks of using control variate are:

- Negative integrals estimates can be obtained even for positive functions.
- It can perform worse than standard Monte-Carlo when $g - \hat{g}$ contains more variations than g (see Figure 9.1 for a 1D example).

To robustly address these two issues, we also compute, with the same samples and thus a minimal overhead, a standard Monte-Carlo estimation:

$$< I_m(\mathcal{R}) >= \frac{1}{N_p} \sum_{i=1}^{N_p} \left(\frac{V(\mathcal{R})}{S} \sum_{s=1}^{S} g(P_{i,s}) \right). \tag{9.11}$$

Note that here, the uniformity of the stratified samples is crucial for $< I_c >$ and $< I_m >$ to be equally well sampled.

Once the N_p passes have been performed, the final vector-valued estimation is obtained by taking, independently for each component, the estimation with lowest variance between standard Monte-Carlo

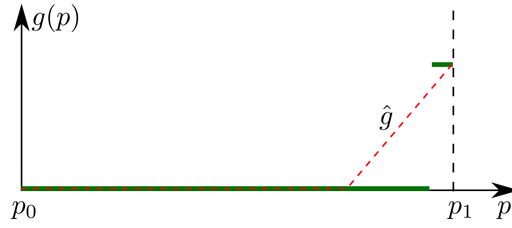


Figure 9.1: Illustration of a function g for which control variate-based estimation requires a lot more samples than standard Monte-Carlo: the function (green) is constant everywhere except near $p1$. The control variate $\hat{g}$ (red dashed) matches g for the left-most part, but leads to non-constant $g - \hat{g}$ for the right-most part, thus requiring more samples.

estimation and control-variate-based estimation. For a given component, if g is known to be positive for this component and the estimate of control variate is negative, the standard Monte-Carlo estimate is taken.

9.4.3 Convergence criterion

As $< I >_n$ is assumed to have a Gaussian distribution with mean value I_n and variance σ_n^2 for each component $n \in \{1, \ldots, N\}$, we have:

$$P(|< I >_n - I_n| < 2\sqrt{\sigma_n^2}) = 0.95. \tag{9.12}$$

Using a relative precision ϵ_r and an absolute precision ϵ_a to define the tolerance, our convergence criterion is:

$$2\sqrt{\sigma_n^2} < max(\epsilon_a, \epsilon_r \times |< I >_n|)), \tag{9.13}$$

$$\Leftrightarrow \sigma_n^2 < \frac{max(\epsilon_a, \epsilon_r \times |< I >_n|)^2}{4} \tag{9.14}$$

where, similarly to [Hah05], we use $< I >_n$ instead of I because I is not known.

As we do not know the exact variance of the Gaussian distribution of the global estimation, we use the global variance estimate to perform the convergence test.

9.4.4 Kd-tree as control variate

Our control variate is based on a kd-tree, each leaf representing the integrand with a first-order approximation. When a leaf is split into two parts, the split plane is placed at the middle of the longest axis. This allows us to associate a node of the kd-tree to each estimation.

The first-order approximation of the integrand inside a node of the kd-tree covering a region $\mathcal{R} \subset \Omega$, of extents $e_1, \ldots, e_D$ in each dimension, with centroid c, is based on two quantities ∇^+ and ∇^-

computed using finite differences:

$$\nabla^{+} g_n(c)_d = \frac{g_n(c + (0, \ldots, e_d/2, \ldots, 0)) - g_n(c)}{e_d/2} \quad (9.15)$$

$$\nabla^{-} g_n(c)_d = \frac{g_n(c - (0, \ldots, e_d/2, \ldots, 0)) - g_n(c)}{e_d/2} \quad (9.16)$$

where $e_d/2$ is the $d-$th component of the vector $(0, \ldots, e_d/2, \ldots, 0)$.

The value of the $n-$th component of the control variate at a point p belonging to $\mathcal{R}$ is then given by the integrand approximation:

$$\hat{g}_n(p) = g_n(c) + \nabla g_n(c, p) \cdot |p - c| \quad (9.17)$$

with $\nabla g_n(c,p)_d = \nabla^{+} g_n(c)_d$ if $p_d \geq c_d$, and $\nabla g_n(c,p)_d = \nabla^{-} g_n(c)_d$ otherwise, for the D components of $\nabla g_n(c,p)$. This interpolant function gives an adequate compromise between smoothness, memory storage, and evaluation cost. In the remaining, we call the N-components function defined from Equation (9.17) a *gradient interpolator*. The integral of this function over $\mathcal{R}$ for a component n is:

$$\int_{\mathcal{R}} \hat{g}_n(p) dp = V \times \left(g_n(c) + \sum_{d=1}^{D} (\nabla^{+} g_n(c) + \nabla^{-} g_n(c)) \times \frac{e_d}{8} \right), \quad (9.18)$$

where V is the volume of the region. The important point is that even though our approximation defines $N \times 2^D$ hyperplanes, we store only $2 \times N \times D$ values for ∇^{+} and ∇^{-}, and the integral can be computed in a linear time with respect to D.

Local selective refinement: The control variate is never globally refined: refinement is applied locally on the sub-tree containing the region of interest. As most methods, our refinement is based on an approximation error [Gon11]. As illustrated on the 1D case in Figure 9.2, our error is estimated using three gradient interpolators inside each leaf: the first covers the whole leaf and the two others cover the two halves of the region. Let I_1 be the integral of the coarsest gradient interpolator, and I_2 be the sum of the integrals of the two others. We refine the leaf if, for at least one component:

$$|I_1 - I_2| > \max(\epsilon_r^c \times \left|\hat{G}\right|, \epsilon_a^c) \quad (9.19)$$

where $\hat{G}$ is the integral of the global control variate computed before the refinement starts. We use $\epsilon_r^c = 10 \times \epsilon_r$ and $\epsilon_a^c = 10 \times \epsilon_a$, to get a sufficient approximation while not being yet as precise as required for the whole estimation.

Control variate evaluation and integral: We use the two interpolators, each covering half of the region covered by a node, as $\hat{g}$ function. The control variate integral in a leaf is the sum of the two interpolators integrals, and the complete control variate integral is the sum of each leaf integral.

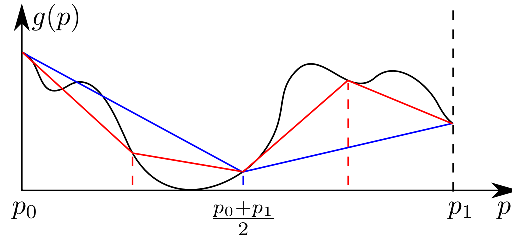


Figure 9.2: Illustration of leaves representation and refinement: suppose a leaf covers the interval $[p_0, p_1]$. Its coarsest interpolator is in blue, and the two more precise interpolators, in red, are obtained by cutting $[p_0, p_1]$ in two halves. Here, I_1 is the integral of the blue interpolator, and I_2 is the sum of the integrals of the red interpolators.

9.4.5 Control variate and estimations interactions

Non-combinatorial stratified sampling: The structure of our control variate provides directly the strata of our stratified sampling scheme. Indeed, the middle split strategy of our kd-tree ensures that for a node $\mathcal{N}$, the best $S = 2^d$ strata covering $\mathcal{N}$ are the nodes at d levels below $\mathcal{N}$. This way, the number of strata is constant whatever the value of D and strata are dominantly cube-shaped, providing an adequate samples distribution.

The value d thus determines the robustness of the estimator, while N_p determines its distribution. The combination of d and N_p allows us to determine the efficiency of the estimation, as $2^d \times N_p$ samples are used for each estimation. We use $N_p = 15$ and $d = 4$ in order to avoid an important loss of samples when refining an estimation while ensuring both an estimator distribution close to a Gaussian and a correct stratification.

Estimation-guided control variate refinement: The estimation of each stratum is an independent Monte-Carlo estimation. Better estimations are obtained with at least one control variate leaf per stratum. The control variate subtree with root $\mathcal{N}$ must thus have a minimum depth of d. If not, we refine it, systematically up to depth d, and selectively afterward. This approach handles difficult cases such as illustrated in Figure 9.3.

This largely improves the robustness of our control variate to arbitrary-frequency content, and reduces the sensitivity of GACV to the values of ϵ_r^c and ϵ_r^a, as refinement occurs locally based on Monte-Carlo estimation.

9.4.6 Estimation error

The globally adaptive subdivision selects the regions to split according to an error measure. We derive it from our convergence criterion (Equation (9.14)). At each step, we select the region in which the estimation is the least converged. For scalar integrands, this is done by selecting the regions according to the variance of their estimates.

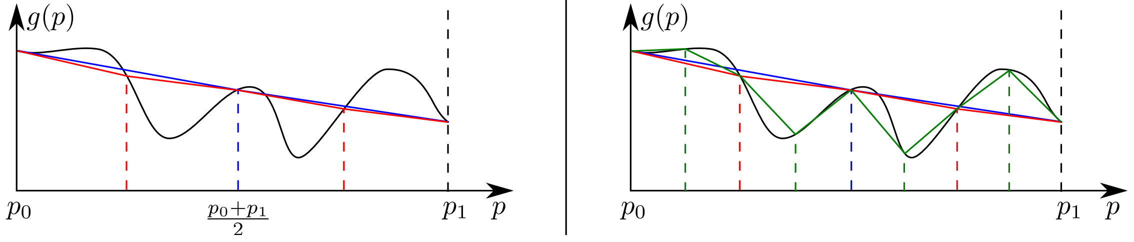


Figure 9.3: Left: illustration of a selective refinement failure: the computed error is far from the actual error. Right: after refinement when $d = 1$ (leading to the green leaves), the error is correctly detected as large, and selective refinement for this node will lead to a more correct control variate.

For vector-valued integrands, scale differences between components have to be taken into account. A criterion based on a scale-independent error is given by:

$$\frac{4 \times \sigma^2}{max(\epsilon_a, \epsilon_r \times |< I >|)^2} < 1. \tag{9.20}$$

As our global estimator is the sum of Gaussian estimators, one by estimation leaf, the global error is the sum of the error of each leaf relatively to the global estimate, given by:

$$\frac{4 \times \sigma^2(\mathcal{R})}{max(\epsilon_a, \epsilon_r \times |< I >|)^2} < 1. \tag{9.21}$$

We obtain the final scalar error from Equation (9.21) by taking the maximum over all the components.

As Equation (9.21) uses $< I >$ which changes at each step, we have to update the errors and rebuild the heap accordingly. As this is too costly, we do not update the errors at each step. Instead, they are updated and the heap is rebuilt very often at the beginning, since the relative scales between components can change dramatically, and less often after a few iterations because the relative scales estimations are stabilized. We thus update the errors and rebuild the heap at the second step, the fourth step, the eight-th step, and so forth.

9.4.7 Integrals over arbitrary finite-dimensional supports

As shown in [KSKAC02], Monte-Carlo estimation can be considered as computing an integral over the unit-hypercube of uniform random numbers in $[0, 1)$. Instead of computing the integral of a function g over an *arbitrary* support Ω,

$$I = \int_\Omega g(x) d\mu(x) \tag{9.22}$$

one can use a mapping $X : [0, 1)^D \to \Omega$ between numbers in $[0, 1)^D$ and elements of Ω, which has the properties of a random variable defined over the uniform random hypercube of dimension D. This random variable X defines a probability distribution on Ω, and for the measure of integration μ, it has an associated probability density function, $p_{X,\mu}$. Then, Equation (9.22) can be reformulated as an integral

over a bounded axis-aligned support:

$$I = \int_{[0,1)^D} \frac{g(X(u))}{p_{X,\mu}(X(u))du} \tag{9.23}$$

where u is a D-dimensional vector of numbers in $[0,1)$. This general reformulation allows us to handle arbitrary supports whenever D is a finite constant and X satisfies $g(x) \neq 0 \Rightarrow p_{X,\mu}(x) > 0$.

Combining GACV and importance sampling: Equation (9.23) also allows us to use GACV and importance sampling at the same time. If a well adapted sampling distribution X is known for g, integrating $\frac{g(X(u))}{p_{X,\mu}(X(u))}$ using GACV leads to a very effective combination of importance sampling and control variate.

9.5 Results

9.5.1 Numerical integration

In order to experimentally verify that GACV reaches our objectives (mathematical and computational robustness, efficiency, generality), we have developed a set of numerical tests and compare GACV against two variants of SUAVE, a deterministic using QMC (SUAVE-QMC) and a stochastic using a PRNG (SUAVE-PRNG).

We use SUAVE with the default numeric parameters given for the Mathematica implementation, and we limit the maximum number of samples to 100 millions. With SUAVE, we can use only the samples in the leaves of the estimation tree, or all the samples to compute the global integral value. However, although slightly faster, this last possibility has shown to largely decrease the accuracy of the method. We thus use the leaves samples only.

9.5.1.1 Tests functions

Genz test functions [Gen87]: Genz defined six families of functions to easily test the behavior of numerical integration methods with arbitrary-dimensional scalar functions. These families of functions are parameterized by two randomly chosen vectors. The first vector, denoted w, is non-affective, *i.e.* different values of w should lead to similar performances. The second vector, denoted c, is affective through its norm (called difficulty d): the larger the norm, the more difficult to integrate the function. For a given difficulty d, the components of w and c are first chosen randomly in $[0,1)$, and c is scaled so that its norm equals d. Note that for a same difficulty, different c vectors should lead to similar performances.

The test functions are defined as:

$$
\begin{aligned}
\text{Oscillatory}: \; f_1(x) &= \cos(c \cdot x + 2\pi w_1) \\
\text{Productpeak}: \; f_2(x) &= \prod_{i=1}^{D} \frac{1}{(x_i - w_i)^2 + c_i^{-2}} \\
\text{Cornerpeak}: \; f_3(x) &= \frac{1}{(1 + c \cdot x)^{D+1}} \\
\text{Gaussian}: \; f_4(x) &= \exp(-c^2 \cdot (x - w)^2) \\
C^0\text{-continuous}: \; f_5(x) &= \exp(-c \cdot |(x - w)|) \\
\text{Discontinuous}: \; f_6(x) &= \begin{cases} 0 & \text{if } x_1 > w_1 \\ & \text{or } x_2 > w_2 \\ \exp(c \cdot x) & \text{otherwise} \end{cases}
\end{aligned}
$$

In a similar fashion to [Hah05], we perform our tests following the process summarized in Algorithm 5. We use the same difficulty values as in [Hah05] ($d(f1) = 6, d(f2) = 18, d(f3) = 2.2, d(f4) = 15.2, d(f5) = 16.1, d(f6) = 16.4$), and compute integrals for 10 values of w and c for each of the six Genz functions, with $\epsilon_r = 10^{-3}$ and $\epsilon_a = 10^{-7}$. We test the computational stability of the stochastic methods – GACV and SUAVE-PRNG – by performing 50 independent estimations of each integral. Compared to [Hah05], we only perform the test for $D = 8$, a middle-dimensional integration problem (tests with $D = 3$ are presented below). In order to also evaluate the behavior of the integration methods on vector-valued integrands, we propose the use of a seventh test function family defined as $f_c = (f_1, \ldots, f_6)$.

Algorithm 5 Test of the behavior of Genz functions integration.

```
 1: for i = 1 to 10 do
 2:   // Scalar integrands test
 3:   for f ∈ {f_1, ..., f_6} do
 4:     (w(f), c(f)) ← generate(d(f))
 5:     for r = 1 to 50 do
 6:       computeIntegral(f, w, c, 0.001)
 7:     end for
 8:   end for
 9:   // Vector-valued integrands test
10:   f_c = (f_1, ..., f_6)
11:   w_c = (w(f_1), ..., w(f_6))
12:   c_c = (c(f_1), ..., c(f_6))
13:   for r = 1 to 50 do
14:     computeIntegral(f_c, w_c, c_c, 0.001)
15:   end for
16: end for
```

Participating media integration: In addition to specifically designed test functions, we use two 3-

components functions defining the scattering coefficients of two participating media represented by RGB triplets. The rendering of these participating media with ray-marching is shown in Figures 9.10 and 9.11.

The first participating medium, called *Porsche*, is the density of a Porsche car, transformed by a colored transfer function. This medium is represented by a grid of $559 \times 1023 \times 347$ nodes, with a total of 193 millions cells. It exhibits a lot of details and sharp non-axis-aligned features.

The second participating medium, called *cloud*, has been produced using Ebert's procedural system [Ebe97]. This model exhibits both smooth parts and very-high frequencies, localized on the border of the cloud. No analytical properties are available for this function.

Our test consists in integrating the scattering coefficients on the whole support of these media with three different relative precisions: 0.1, 0.01, and 0.001. For each precision, we evaluate the robustness and the accuracy of the three methods by computing the integral, 1000 times with GACV and SUAVE-PRNG, and only once with SUAVE-QMC as it is deterministic. The same color code is used as for the Genz test functions.

Plots description: In our figures, we use the following color code: blue for GACV, red for SUAVE-PRNG and green for SUAVE-QMC.

For results related to Genz functions, for each family of scalar functions f_1 to f_6, we consider ten different functions defined by ten different values of (w, c). For each of these functions, we perform 50 computations. In plots analyzing the integration of Genz scalar test functions (Figures 9.4 and 9.7), values are organized as 6 successive blocks of 500 values: one per function family. Each block contains 10 contiguous sets of 50 values, one per function. Finally, in Figures 9.4, 9.5, 9.7, and 9.8, plots illustrating the behavior of SUAVE-QMC are presented by duplicating 50 times the result of single evaluations.

9.5.1.2 Accuracy and mathematical robustness

Scalar integrands: The accuracy of our method is confirmed by the relative errors computed from reference values, obtained with standard Monte-Carlo. Figure 9.4 presents the $\log_{10}(|< I > -I| / |I|)$ plot for the 3000 scalar integrals computed on the Genz functions. All methods give results with a relative error of about 0.001, which matches our precision requirements.

There is a noticeable accuracy exception both for GACV and SUAVE-PRNG, which exhibit a very large variance for one or two of the ten integrands of the family of functions f_6 (Figure 9.4, right), as the returned value for the integral is zero for some of the 50 computations, while others are close to the reference. In the case of GACV, this is due to an insufficiently dense sampling at the first step. Indeed, all samples may fall in the interval where f_6 returns 0, depending on the w_1 and w_2 values. In this case, although the w parameter should not be affective, it in fact affects the size of the non-zero interval. It is important to notice that this failure case can be detected by the large variance of several independent lower-precision estimates.

Vector-valued integrands: Figure 9.5 presents the $\log_{10}(|< I > -I| / |I|)$ plot for the 500 vector-

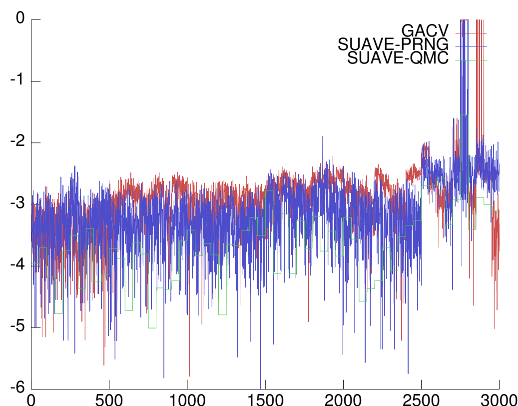


Figure 9.4: Plot of the relative error for each scalar integrand. Ordinates are in logarithmic scale.

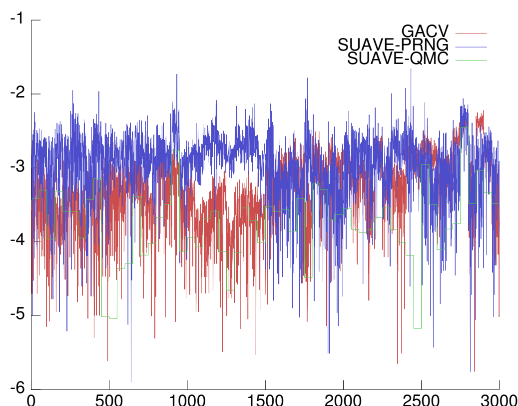


Figure 9.5: Plot of the relative error for the 6 components of each integrand. Ordinates are in logarithmic scale.

valued integrals, plotting each component sequentially (first 500 points: first component, *etc.*). Vector-valued integrands improve GACV robustness, by avoiding the failures appearing in Figure 9.4. In fact, since other components require splitting, the sampling is denser for the sixth component as well.

Figure 9.6 presents histograms of the 1000 integral estimations of the scattering coefficients on the Porsche and cloud media, for the three relative precisions. Results are displayed for the third component only as the same behavior is observed on each of them. The underestimation in the Porsche case is caused by regions where the integrand is zero almost everywhere, which often lead to zero estimates and variances. However, we can see that GACV performs more accurate estimations, and better handles this case than both variants of SUAVE.

Figure 9.6 also shows that the Gaussian distribution of the estimations of GACV is in practice well verified. This validates the confidence intervals we return, as they are based solely on the Gaussian nature

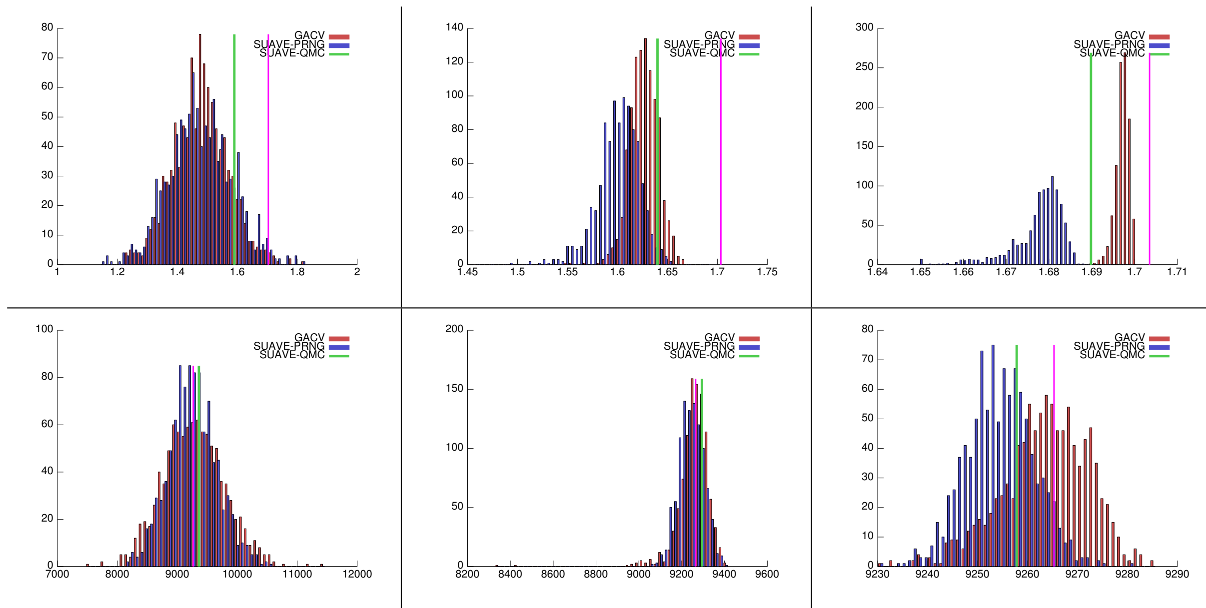


Figure 9.6: Histograms of the values obtained for 1000 estimations with decreasing ϵ_r values (and therefore increasing precision) on the Porsche medium (top row) and on the cloud medium (bottom row), showing only the third component of the computed integral. The green line is the value obtained by SUAVE-QMC, the magenta line is the exact integral value. Left: $\epsilon_r = 0.1$, middle: $\epsilon_r = 0.01$, right: $\epsilon_r = 0.001$.

of the distribution.

9.5.1.3 Efficiency and computational robustness

Scalar integrands: Figure 9.7 contains plots of the computation time, number of evaluations required and the memory usage when available, for each scalar integrand of the Genz test.

As we can see in Table 9.1, SUAVE-QMC is consistently faster than SUAVE-PRNG. It is also faster than GACV for five families of functions of the Genz test (families 2 to 6) with varying factors, but it is slower by an average factor of 100 for integrands of the first family. However, on an absolute scale, for the cases where SUAVE-QMC is faster, the absolute difference does not exceed a second, while the difference can exceed ten *minutes* in favor of GACV for some integrands, with computation times for GACV below 10 seconds, and in average below 0.5 seconds. Note that functions of the families f_2, f_4 and f_5 are separable on all the definition domain, and that functions of the f_6 family are separable on the non-zero domain. This separability strongly favors SUAVE as VEGAS uses separable functions for sampling.

Figure 9.7 also shows that GACV is very stable in terms of both computation time and number of evaluations for a same integrand, while SUAVE-PRNG exhibits large variations. The memory consumption of GACV is also stable when evaluating several times an integrand, except for the integrands where zero estimates are obtained. In this case, memory consumption is lower since these zero estimates are

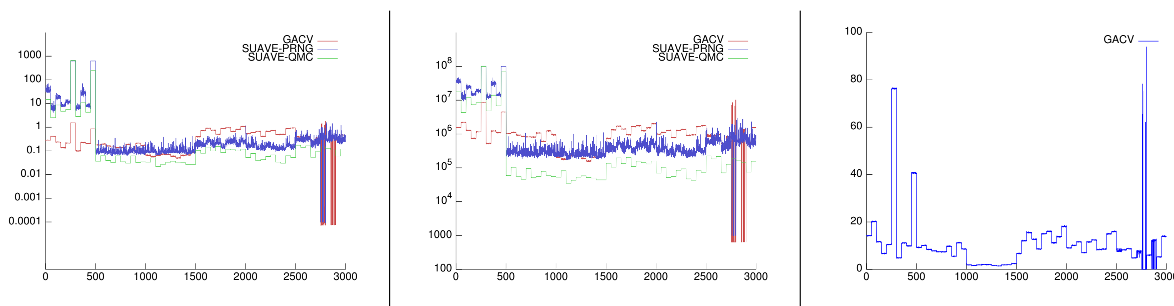


Figure 9.7: Left: Time, in seconds, required by the computation of each scalar integral of the Genz test. Ordinates are in logarithmic scale. Middle: Number of evaluations required to compute each integral. Ordinates are in logarithmic scale. Right: Memory consumption with GACV, in MB.

	f_1	f_2	f_3	f_4	f_5	f_6
GACV	0.45	0.16	0.06	0.76	0.64	0.34
SUAVE-PRNG	138.51	0.10	0.11	0.23	0.17	0.34
SUAVE-QMC	96.02	0.04	0.03	0.11	0.05	0.12

Table 9.1: Average time, in seconds, required by each method to compute an integral for each family.

handled using a single node in the estimation tree.

The peak memory consumption for this test is 93MB for GACV, with a non-memory-optimal implementation. Memory consumption data is not available for SUAVE, but we did note that the memory used by our simple test program went as high as ten *gigabytes* of memory with both SUAVE-QMC and SUAVE-PRNG for two of the integrands of the first family (*i.e.* , the 100 estimations each required 10GB of memory). This peak is confirmed by the number of integrand evaluations required by SUAVE for each integral computation: the 100 millions samples limit has been reached several times by SUAVE-PRNG and SUAVE-QMC. Increasing the maximum number of samples would allow a complete estimation, at the cost of increased computation time and memory consumption.

Vector-valued integrands: As in the scalar case, Figure 9.8 contains plots of the computation time, the number of evaluations required and memory usage when available, for each of the 500 vector-valued integral evaluations.

A robust method for vector-valued integrand should not require more function evaluations than the sum of function evaluations required to compute each component separately. In fact, each evaluation brings informations for all components at once, leading to an automatic use of the correlation amongst components. Table 9.2 shows that this is well verified in practice for GACV when integrating functions of the f_c family, while both SUAVE-PRNG and SUAVE-QMC do not benefit from any possible correlation amongst components and require a lot more evaluations than computing each component separately. This is a consequence of the use of importance sampling on weakly-correlated vector-valued integrands. The plots of number of evaluations and computation times assess that GACV is also computationally stable on vector-valued functions, and much faster than both versions of SUAVE.

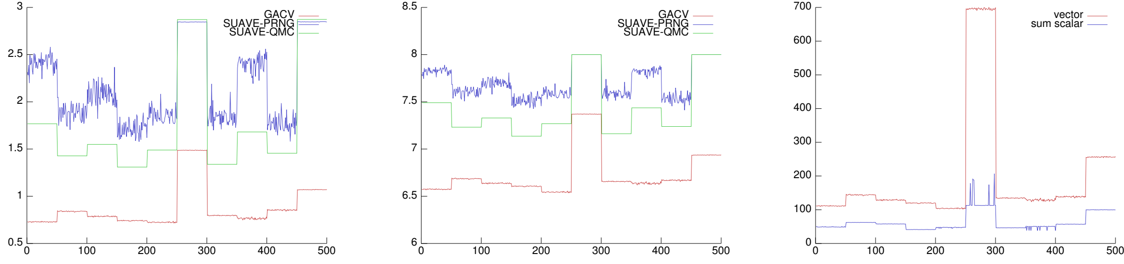


Figure 9.8: Left: Computation time of each vector integral of the Genz test, in seconds. Ordinates are in logarithmic scale. Middle: Number of evaluations required to compute each integral. Ordinates are in logarithmic scale. Right: Memory consumption with GACV when all components are directly computed (red), and sum of the consumptions when they are computed separately (blue), in MB.

GACV	0.69	0.71	0.68	0.89	0.67	1.74	0.88	0.81	0.74	0.79
SUAVE-PRNG	1.73	2.76	1.85	1.96	1.85	0.98	2.59	1.96	2.09	1.45
SUAVE-QMC	1.70	3.46	1.72	1.78	2.07	0.99	2.75	1.89	2.42	0.76

Table 9.2: Average ratio of number of evaluations required to compute f_c and $f_1, \ldots, f_6$ separately, for the ten different values of w and c.

The right-most plot in Figure 9.8 shows that the memory used by GACV for each f_c integration is larger and globally proportional to the sum of the memory used to evaluate each component. Indeed, the adaptive control variate tree stores more elements per gradient and value at centroid than each individual control variate tree when components are computed separately. The ratio of memory consumption depends on the correlation between components, the stronger the correlation, the lower the ratio. Note that in theory, this plot does not have any per-evaluation meaning, as $f_1, \ldots, f_6$ are evaluated independently of f_c. However, as our memory use is stable, it allows us to visualize the proportionality for a given family of functions.

The speed and computational time stability of GACV is also confirmed by our tests on participating media, shown in Figure 9.9. Both SUAVE-PRNG and SUAVE-QMC are only marginally faster than GACV at low precision, but become significantly slower when high precision is required. These tests also show that GACV is sublinear in computation time and number of evaluations with respect to variance: it requires only 200 more time to increase the precision by 100, while standard Monte-Carlo would require 10.000 more time.

9.5.2 Trees

For the two media cloud and Porsche, we compare kd-trees built using our method, against regular grids with constant cell representations (the most common representation), and against error-guided octrees with constant or linear cell representations, whose global error enforcement is done using the method described in Section 9.3.3. The average values and error computations are all done using our GACV integration algorithm (Section 9.4). All the errors are expressed relative to the participating media vol-

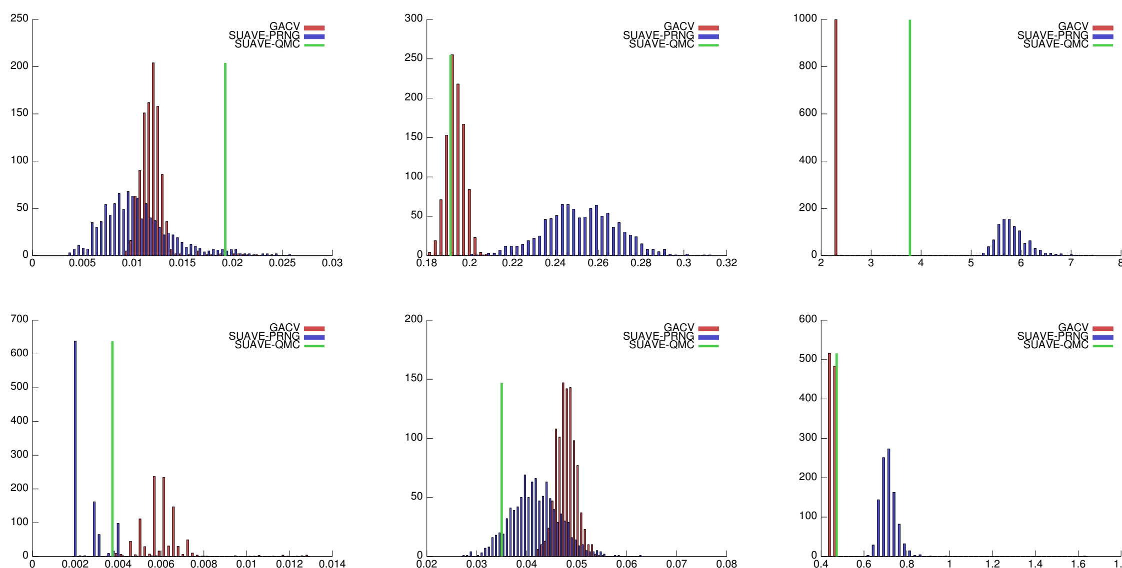


Figure 9.9: Histograms of time, in seconds, required to compute the integral of the RGB scattering coefficients for the whole Porsche (top row) and the cloud (bottom row) media, for decreasing relative error (increasing precision). The green vertical line is the time needed by SUAVE-QMC. Left: $\epsilon_r = 0.1$, middle: $\epsilon_r = 0.01$, right: $\epsilon_r = 0.001$.

umes, *i.e.* the actual error is the given error multiplied by the volume of the participating medium. This kind of error allows us to scale the participating medium without having to change the error in order to keep the same amount of details.

Table 9.3 clearly shows that regular grids are not adapted for highly-varying participating media. By contrast, octrees perform really well for media where no sharp features are present, their adaptability being enough to correctly handle these kinds of media. As expected, our method always requires less nodes, for similar errors. Note that although the total number of nodes are close, optimizing the cost leads each time to at least twice as much leaves in the octree as in the kd-tree. As the leaves require more memory, the final memory consumption is largely in favor of our method.

Table 9.4 highlights the impact on the number of nodes of the restricted adaptability of octrees, which leads to structures larger by a factor of 10 compared to our method.

Table 9.5 shows the construction times of the structures on a core i7 with 16GB of memory. Octrees are much faster to build, as splitting positions do not have to be found through minimization. For our method, in average for each node, 20 split planes have been tested, leading to 80 integrals evaluation per node (two child nodes per split-plane, with average and error computed for each child node). Repeated constructions of the trees have assessed the predictability of our method: the trees are stable in terms of the number of nodes, computation times, and global error reached.

Memory optimality: These results clearly show that, even if octrees are a good compromise between construction cost and memory footprint for smooth media, they break down for media with sharp fea-

target global error	0.001	0.005	0.05
grid, constant	N/A	$387 \times 512 \times 309$ (1401MB)	$97 \times 128 \times 78$ (22MB)
octree, constant	$2,348,183/16,437,283$ (385MB)	$258,412/1,808,885$ (42MB)	$4,976/34,836$ (0.81MB)
kd-tree, constant	$7,072,079/7,072,080$ (215MB)	$527,254/527,255$ (16MB)	$15,274/15,275$ (0.46MB)
octree, linear	$72,118/504,826$ (27MB)	$16,332/114,331$ (6.1MB)	$1,027/7,190$ (0.38MB)
kd-tree, linear	$280,368/280,369$ (17MB)	$62,944/62,945$ (3.8MB)	$3,756/3,757$ (0.23MB)

Table 9.3: Approximation of the cloud participating medium using various representations. The number of cells, or the number intermediate nodes/leaf nodes for various global error values is given, with in parenthesis the memory consumption in MB when the tree is used for rendering (the CRs of the intermediate nodes are removed to minimize memory consumption during rendering). Memory sizes are computed based on an optimized layout for all structures.

target global error	0.5
octree, constant	597 501/4 182 508 (98MB)
kd-tree, constant	246 762/246 763 (7.5MB)
target global error	5
octree, constant	2 351/16 463 (0.38MB)
kd-tree, constant	3 639/3 640 (0.11MB)

Table 9.4: Number of intermediate nodes/leaf nodes to approximate the Porsche participating medium, using an octree or a kd-tree built using our method. The same convention as in Table 9.3 is used to compute the structures size.

tures. Meanwhile, our method always leads to lower memory footprints and therefore more efficient traversals, the improvement being as large as one order of magnitude in the case of media with sharp features.

Compactness and computational efficiency: Figures 9.10 and 9.11 illustrate the multi-resolution aspect of the kd-trees we compute, and allow us to visually compare results obtained by our method for different errors to reference images obtained from the original media. Figure 9.10 clearly shows that the sharp details such as holes, are all conserved and accurately represented, even if the number of leaves in our structure is more than 700 *times lower* than the number of cells in the grid. It also shows that rendering times are much longer when using the grid for a same number of samples: although both structures

medium, global error	cloud, 0.001	Porsche, 0.5
octree, constant	1h40m	21m
kd-tree, constant	15h05m	1h15m
octree, linear	0h05m	N/A
kd-tree, linear	1h09m	N/A

Table 9.5: Computation times for the structures built to test our method.

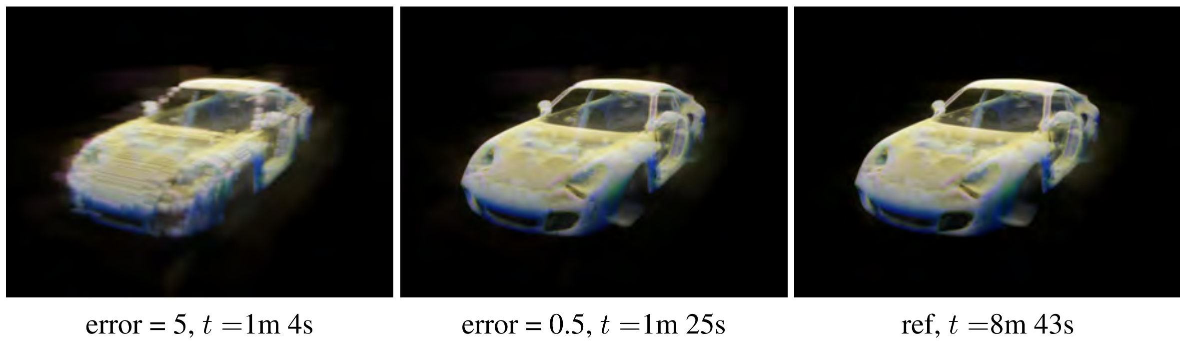


Figure 9.10: Left and Middle: Image obtained by using a constant kd-tree approximation, with a global error of respectively 5 and 0.5. Right: Reference image, obtained using the original grid. The rendering time is provided for each image. All the images have been computed with the same number of samples per pixel and at the same resolution.

provide analytical sampling and transmittance computations, a $559 \times 1023 \times 347$ grid is more costly to traverse than a kd-tree with 7000 or $500,000$ nodes.

Versatility: We use a single implementation to build the kd-tree and the octree, by considering the latter as a special case where the splitting plane is not found *via* minimization. As this choice can be done on the fly during construction, we can in fact produce hybrid octree/optimal structures whose number of nodes will be in-between, with a construction time which can range continuously from the one of the octree to the one of the optimal kd-tree. Changing the construction process on the fly allows us to build hybrid octree/kd-tree structures whose precision/construction time ratio is controllable. Finding strategies to build an as-good-as-possible tree given a fixed amount of time, or to build a tree with a maximal precision/construction time ratio would be a step toward an ideal unified discrete representation of any participating media.

9.6 Extensions

Approximate free-path sampling: Our method is originally targeted at approximating any participating medium with a controlled error, to use this approximation instead of the original medium. A coarser approximation can also be used to perform free-path sampling in the non-approximated medium. In contrast with methods which are dedicated to free-path sampling based on Woodcock tracking such as [YIC+10], our method does not sample according to the transmittance of the actual medium. This approximate sampling leads to additional variance, but avoids the necessity to be able to compute a sharp maximum value of the coefficients over an arbitrary region, which is challenging for highly-varying medium such as the cloud one. However, using an approximation for sampling can lead to bias, as the approximation can be zero while the actual medium is not. This bias can be corrected by always having a non-zero approximation, at the cost of additional variance. For this specific use-case, a structure similar

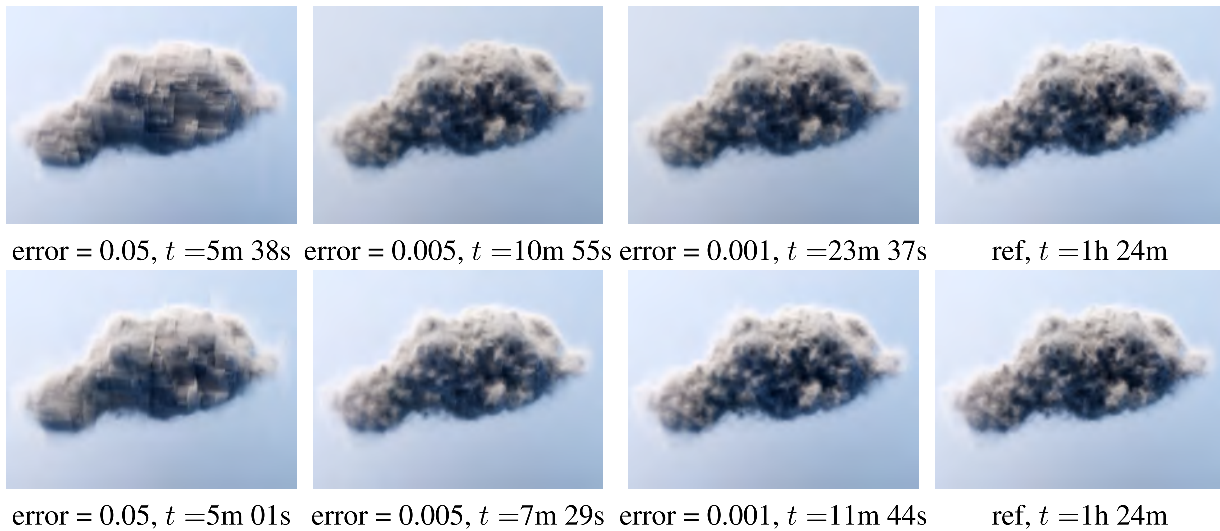


Figure 9.11: Columns 1 to 3: Images of the cloud computed from a constant (top row) or linear (bottom row) kd-tree approximation with decreasing global error, using analytic sampling. Last column: Reference image obtained using ray-marching. The rendering time is provided for each image. All the images have been computed with the same number of samples per pixel and at the same resolution.

to octrees can be most efficient, considered their reduced computation time.

Pre-integration: Pre-integration is an efficient way to visualize complex effects at interactive framerates [CNS^{+}11]. It is possible to use pre-integration with our structures, as they are basically a spatial subdivision structure. However, as the nodes can have arbitrary shapes, anisotropic representations should be preferred to avoid too strong artifacts for nodes which do not have a cubic shape.

Handling arbitrary functions: Although our method has been developed to handle participating media, it is trivial to generalize it in order to handle any kind of functions from a compact subset of $\mathbb{R}^n$ to $\mathbb{R}^d$. For instance in the computer graphics context, it can be used to handle 2D textures, or precomputed animation by adding time as supplementary dimension, at the cost of increased computational costs. Additionally, although the discussion has been limited to isotropic participating media, which are the most common in rendering, anisotropic participating media can also be directly handled by adding two supplementary angular dimensions.

9.7 Conclusion

Our representation allows us to handle any participating medium in a generic way. In Flexray, we use it to approximate any participating media, and we also use it for approximate free-path sampling, building a coarse approximation of any participating media which can not be sampled efficiently. Additionally storing the approximated media on disk avoids us having to recompute them for each render.

Adding this unified representation of participating media to the set of methods we have developed,

the LTE (Equation (1.55)), the RTE (Equation (1.57)) and the final pixel value (Equation (1.26)) can be computed using robust methods.

- Representativities (Chapter 5) improves the robustness of local path sampling for the LTE, and the boundary condition of the RTE. Other representativity functions could be develop to improve the RTE part as well, to also focus where incident radiance is most important along a ray.
- The participating media representation we just presented ensures analytical transmittance and free-path sampling, avoiding to have problems with representations not adapted to RTE evaluation using the Monte-Carlo method.
- The sample-space bright-Spot removal method (Chapter 6) improves the robustness of accumulation to compute the final pixel value, and the image-space method (Chapter 7) removes the last few remaining bright-spots, ensuring a correct HDR processing.
- Finally, our robust adaptive sampling algorithm (Chapter 8) focuses processing power on parts of the image where convergence is harder to reach, resulting in reduced rendering times for a same desired quality.

As a final step, we focus in the next chapter on the efficiency of the simulation part, which is the most costly part, to provide robust *and* efficient rendering.

10

Combinatorial bidirectional path-tracing for efficient hybrid CPU/GPU rendering

10.1 Introduction

In this chapter, we investigate the possibilities to use the processing power of GPUs to efficiently compute the light transport equation (Equation (1.55)), adding as few approximations as possible, and still allowing us to use a maximum of the flexibility of our engine: generic materials, generic light sources, generic camera models, a lot of textures, caching mechanisms, lazy loading, *etc.*. Pure GPU methods have to limit code sizes, memory use and code branchiness in order to reach maximum efficiency, therefore restricting to small datasets and fixed models (for instance, materials expressed using trees of textures are not easily feasible, while they are extremely useful). We therefore choose to use both the CPU and the GPU together, using the GPU for computations where restricted flexibility does not imply large restrictions.

We base our study on bidirectional path-tracing (BPT) [VG94] [LW93], which is a robust method for most scenes, although it has difficulties with distant light sources and some specific scenes (see [HOJ08] for more details on these scenes). For the many scenes where it is robust, bidirectional method (and therefore the method we present now) should avoid most of the bright spots, which means that when using it, standard accumulation can be used alone. As simulation is the most costly part, our method allows us to get robust *and* efficient rendering for most scenes illuminated with local light sources.

Contribution: We combine correlated sampling and standard BPT to efficiently use both CPU and GPU in a cooperative way (Section 10.3). The basic principle of BPT is to repeatedly sample an optical path leaving from the camera, and an optical path leaving from the light. Complete paths are then created by linking together each sub-path of the camera path with each sub-path of the light path. The last vertex of each sub-path are called linking vertices, and the segment between the two linking vertices is the linking segment. A complete path created this way contributes to the final image if the linking vertices

Full paper at Eurographics 2011, co-authored by Loïc Barthe, Mathias Paulin and Pierre Poulin [PBPP11a]

are mutually visible, and if some of the energy arriving to the light linking vertex is scattered to the camera path. Instead of combining two paths, we combine sets of camera and light paths, computing the values needed for linking on the GPU. As each camera path is combined with each light path, many more linking segments are available, allowing us to use the GPU at its maximum without increasing the cost of sampling the paths (Section 10.4). The only approximation we add is that geometry must be tessellated as a pre-process. We then interleave the CPU and GPU parts in order to obtain an algorithm where both the CPU and GPU are always busy (Section 10.5). This reformulation reduces the processing time by a factor varying between 12 and 16 compared to standard BPT (Section 10.6), allowing feedback in less than a minute even for complex scenes, and the computation of high-quality images in one hour. This work has been presented as a full paper at the Eurographics 2011 international conference [PBPP11a].

10.2 Related works

Using both the CPU and GPU in a cooperative way can provide a large gain of performance without restricting the flexibility of an engine, allowing us to provide high-quality results or rough previews significantly faster.

Attempts at isolating parts of algorithms to execute them on GPU [Lux10] or on hybrid CPU/GPU render farms [BBS$^+$09] are examined in rendering engines. [BBS$^+$09] focuses on path-tracing, which is easier to execute efficiently in a distributed way. For bidirectional path-tracing, *LuxRender* [Lux10] performs intersection tests on the GPU. The main problem that face LuxRender developers is keeping both the CPU and the GPU busy all the time. In general, the CPU is too slow to provide enough work to the GPU. More generally, it is not easy to adapt bidirectional algorithms such as BPT or progressive photon mapping and its extensions [HOJ08, HJ09, KZ11] to efficiently use the GPU to compute intermediate data, without restricting the size of the datasets nor the complexity of the materials. In fact, sampling, which must be done on CPU as it involves all the dataset and all the materials, would in general require much more time to be computed than the GPU part, leading to a negligible gain.

10.3 Combinatorial bidirectional path-tracing (CBPT)

10.3.1 Base algorithm

In BPT-based algorithms, a camera path $\overline{x} = (\mathbf{x_0}, \ldots, \mathbf{x_c})$ and a light path $\overline{y} = (\mathbf{y_0}, \ldots, \mathbf{y_l})$ are sampled. $\mathbf{x_0}, \ldots, \mathbf{x_c}$ are called *camera vertices*, $\mathbf{y_0}, \ldots, \mathbf{y_l}$ are called *light vertices*. For each vertex $\mathbf{x_i}$ or $\mathbf{y_j}$ located on the surface of an object, the parameters of the bidirectional scattering distribution function (BSDF) are computed using a shader tree. Complete paths are then created by linking sub-paths $(\mathbf{x_0}, \ldots, \mathbf{x_i})$ and $(\mathbf{y_0}, \ldots, \mathbf{y_j})$, for all the possible couples (i, j). The number of segments of each complete path is $i + j + 1$, and the linking segment is the segment $(\mathbf{x_i}, \mathbf{y_j})$. Let function $f_C(x, i)$ give the energy transmitted by x from $\mathbf{x_i}$ to $\mathbf{x_0}$, and $f_L(y, j)$ give the energy transmitted by y from $\mathbf{y_0}$ to $\mathbf{y_j}$. The

energy emitted from $\mathbf{y_0}$ which arrives to the sensor *via* the path $\overline{z} = (\mathbf{y_0}, \ldots, \mathbf{y_j}, \mathbf{x_i}, \ldots, \mathbf{x_0})$ is then:

$$\begin{aligned} f_{i,j}(\overline{x}, \overline{y}) \quad = \quad & W_{i,j}(\overline{x}, \overline{y}) f_L(\overline{y}, j) G_{\mathcal{A}}(\mathbf{y_j}, \mathbf{x_i}) f_C(\mathbf{x}, i) \times \\ & f_s\left(\mathbf{y_{j-1}} \rightarrow \mathbf{y_j} \rightarrow \mathbf{x_i}\right) \times \\ & V(\mathbf{y_j}, \mathbf{x_i}) \times \\ & f_s\left(\mathbf{y_j} \rightarrow \mathbf{x_i} \rightarrow \mathbf{x_{i-1}}\right) \end{aligned} \tag{10.1}$$

where $W_{i,j}$ represents the radiometric model of the camera for the complete path (Section 2.1.3), f_s is the BSDF, V is the visibility function (1 if not occluded, 0 otherwise), and $G_{\mathcal{A}}$ is the geometric term between the two points:

$$G_{\mathcal{A}}(\mathbf{y}_j, \mathbf{x}_i) = \frac{\left|\mathbf{N_{y_j}} \cdot \mathbf{y_j} \rightarrow \mathbf{x_i}\right| \left|\mathbf{N_{x_i}} \cdot \mathbf{x_i} \rightarrow \mathbf{y_j}\right|}{\left|\left|\mathbf{x_i} - \mathbf{y_j}\right|\right|^2}. \tag{10.2}$$

Equation (10.2) is similar to Equation (3.8), ignoring the transmittance terms, and taking into account the structure of the path.

We define the *basic contribution* $L_{i,j}(x, y)$ of such a complete path as:

$$L_{i,j}(\overline{x}, \overline{y}) = \frac{w_{i,j}(\overline{x}, \overline{y}) f_{i,j}(\overline{x}, \overline{y})}{p_{i,j}(\overline{x}, \overline{y})}. \tag{10.3}$$

$p_{i,j}(\overline{x}, \overline{y})$ is the density probability with which the two sub-paths have been sampled, and $w_{i,j}(\overline{x}, \overline{y})$ is the multiple importance sampling (MIS) weight [VG95].

In our implementation, we use the direct BSDF probability density function (PDF) p to sample directions for the camera path, the adjoint BSDF PDF p^* [Vea97] to sample directions for the light path, and the balance heuristic [VG95] to compute the MIS weights:

$$w_{i,j}(\overline{x}, \overline{y}) = \frac{p_{i,j}(\overline{x}, \overline{y})}{\sum_{s,t} p_{s,t}(\overline{x}, \overline{y})} \tag{10.4}$$

where each (s, t) couple is one of the possible techniques with which $\overline{z}$ could have been sampled. Computing this weight requires to compute $p(\mathbf{x_{i-1}} \rightarrow \mathbf{x_i} \rightarrow \mathbf{y_j})$ and $p^*(\mathbf{y_j} \rightarrow \mathbf{x_i} \rightarrow \mathbf{x_{i-1}})$ using the BSDF at $\mathbf{x_i}$, and $p(\mathbf{x_i} \rightarrow \mathbf{y_j} \rightarrow \mathbf{y_{j-1}})$ and $p^*(\mathbf{y_{j-1}} \rightarrow \mathbf{y_j} \rightarrow \mathbf{x_i})$ using the BSDF at $\mathbf{y_j}$.

When either i or j are less than 1, the corresponding terms are not based on the BSDF, but instead on the light or camera properties. If $j = -1$, it means that $\mathbf{x_c}$ is on a light, making a complete path by itself.

The data that depends on both $\mathbf{x_i}$ and $\mathbf{y_j}$ has to be computed per linking segment, and is the most time-consuming task when computing the contribution of a complete path. These data can be computed on the GPU very efficiently, in parallel for each linking segment. Unfortunately, producing a sufficient number of linking segments would require to sample and combine a very large number of pairs, leading to very large CPU costs, large memory footprints both on CPU and GPU, and very time consuming CPU-to-GPU memory transfers.

The key idea allowing us to use both CPU and GPU efficiently is to sample populations of N_C camera paths and N_L light paths independently on CPU, and then combine each camera path with each light path. This leads to the combination of $N_C \times N_L$ pairs of paths, and allows us to have largely enough linking segments to benefit from the processing power of GPUs without requiring larger sampling costs. Combining all camera paths with the same light paths introduces a correlation in the estimations, but does not lead to bias in the average estimator.

In practice, we have three kernels which compute, for each linking segment $(\mathbf{x_i}, \mathbf{y_j})$ in parallel:

- the visibility term $V(\mathbf{x_i}, \mathbf{y_j})$,
- the shading values involving the BSDF of the camera point: $f_s\left(\mathbf{y_j} \to \mathbf{x_i} \to \mathbf{x_{i-1}}\right), p(\mathbf{x_{i-1}} \to \mathbf{x_i} \to \mathbf{y_j})$, and $p^*(\mathbf{y_j} \to \mathbf{x_i} \to \mathbf{x_{i-1}})$, if $\mathbf{x_i}$ has an associated BSDF (*i.e.* it is neither on the camera lens nor on a light),
- the shading values involving the BSDF of the light point: $f_s\left(\mathbf{y_{j-1}} \to \mathbf{y_j} \to \mathbf{x_i})\right), p(\mathbf{x_i} \to \mathbf{y_j} \to \mathbf{y_{j-1}})$, and $p^*(\mathbf{y_{j-1}} \to \mathbf{y_j} \to \mathbf{x_i})$, if $\mathbf{y_j}$ has an associated BSDF.

If $\mathbf{x_i}$ or $\mathbf{y_j}$ does not have an associated BSDF, the probabilities (probability to have sampled the light, probability density to have sampled the point on the light, probability density to have sampled the direction from the camera, *etc.*), and the light emission and importance emission terms are computed on CPU, to keep the flexibility on camera and light models that can be used.

The final contributions of a pair (x, y) can then be split into two parts (two separate contribution processors). The first part is the sum of all the basic contributions that affect the image location intersected by the first segment of x. We denote it as the bidirectional contribution: $L^b(x, y) = \sum_{i>0, j\neq -1} L_{i,j}(\overline{x}, \overline{y}) + L_{c,-1}(\overline{x}, \overline{y})$ and we call *bidirectional image* the image obtained by considering only the bidirectional contributions. The second part contains all the contributions obtained by light-tracing, each affecting a different image location: $\{L_{0,0}(\overline{x}, \overline{y}), L_{0,1}(\overline{x}, \overline{y}), \ldots, L_{0,l}(\overline{x}, \overline{y})\}$. We call *light-tracing image* the image obtained by adding all the contributions from light-tracing, each multiplied by the number of pixels of the final image. In our implementation, light-tracing does not contribute to direct lighting, as it brings a lot of variance for this type of light transport. The final image is the sum of the bidirectional and light-tracing images.

As a result, a step of CBPT consists in:

1. sample a camera population $\{\overline{x}\}$ of N_C paths, and a light population $\{\overline{y}\}$ of N_L paths;
2. compute the combination data for these two populations on GPU;
3. compute the contributions of each pair of paths, splatting N_C values to the bidirectional image, and splatting the light-tracing contributions to the light-tracing image.

Note that as is, our algorithm does not directly handle motion blur, but it can be integrated in a straightforward manner by sampling each $(\{\overline{x}\}, \{\overline{y}\})$ population couple with a specific value of time, *i.e.* all the paths of the two populations have the same time value, and this value is different for each couple of populations.

10.3.2 Discussion

Setting N_C and N_L: Ideally, we would like to always be perceptually faster than standard BPT. Perceptually faster means computing more camera paths per second, with each camera path being combined with $N_L > 1$ light paths. This leads to a similar or faster coverage of the image, with each camera path bringing a lower-variance estimate than in standard BPT, leading to perceptually faster convergence. N_C and N_L can be computed to ensure faster perceptual convergence, by measuring the time t_b needed by BPT to sample, combine, and splat the contribution for a pair of paths, and the time $t_s(N_C, N_L)$ needed by CBPT to perform one step. As the combination is the most time consuming part of a step, $t_s(N_C, N_L)$ is roughly constant as long as the number of pairs $P = N_C \times N_L$ remains constant. Therefore, for a fixed P, an appropriate N_C value is such that

$$N_C > \frac{t_s(P)}{t_b}. \tag{10.5}$$

A lower N_C value will lead to lower-variance estimate of each path, larger value will lead to faster coverage, but also more correlation. A side-effect of Equation (10.5) is that if N_C, computed using this equation, is such that N_L would be < 1, this indicates that the machine on which CBPT is running is not fast enough to bring any advantage over standard BPT *for the chosen* P.

Light-tracing: The discussion above does not take into account light-tracing, and using Equation (10.5) generally gives N_L values that are small, leading to high-variance caustics. Light-tracing does not really take advantage of the GPU combination system, as each light sub-path is combined with only one vertex of a camera path, namely the vertex which lies on the lens of the camera. Moreover, contributions for different camera paths are in general very similar, or even equal when using a pinhole camera, as all the lens vertices are at the exact same location. We therefore choose to compute light-tracing using a standard CPU-based light-tracer.

At each step of CBPT, we sample N_T light paths ($\{\overline{y_{lt}}\}$) and compute their light-tracing contributions. In general, we choose N_T close to N_C to get approximately the same bidirectional/light-tracing ratio as standard BPT. This leads to the final algorithm for a step of CBPT, presented in Algorithm 6.

Algorithm 6 A complete step of CBPT.

sample($\{\overline{x}\}$)
sample($\{\overline{y}\}$)
upload($\{\overline{x}\}$, $\{\overline{y}\}$)
gpu_comp($\{\overline{x}\}$, $\{\overline{y}\}$)
combine($\{\overline{x}\}$, $\{\overline{y}\}$)
sample($\{\overline{y_{lt}}\}$)
compute_lt($\{\overline{y_{lt}}\}$)

Correlated sampling: Correlated sampling can take several forms, such as re-using previous paths in

order to improve the sampling efficiency [VG97, CTE05], or re-using a small number of well-behaved random numbers to compute different integrals [KH01]. In our method, the camera and light paths are all sampled independently using different random numbers, as in standard BPT. Therefore, complete paths are sampled in a correlated way, as they are created by linking the sub-paths in all possible ways. To avoid visible correlation patterns in the final image while ensuring a proper coverage of the image, the image-space coordinates that are used for each camera path are generated in an array, using a stratified scheme over the entire image or coordinates from an adaptive sampling scheme, with the equivalent of four samples per pixel. This array of samples is then shuffled. When sampling a camera population, each path uses the samples sequentially in the array, leading to paths that most likely contribute to different parts of the image. Therefore, correlation is present, but as it is spread randomly over the image, no regular patterns appear. This array is regenerated each time all the samples have been used.

10.4 Efficient computation of combination data

Our algorithm requires an efficient computation of the combination data on the GPU. In this section, we suppose that for each vertex of the two populations $\{\overline{x}\}$ and $\{\overline{y}\}$, we have the position, the BSDF parameters, and the direction to the previous vertex in the path. The size of this data is in $O(N_C + N_L)$. As there are typically few vertices in populations, the GPU memory requirements are very low for the population data. Combining populations exhaustively avoids uploading the $O(N_C \times N_L)$ linking segment array that would otherwise be necessary.

We now give some high- and low-level details on our implementation. Figure 10.2 shows how the techniques we use are put together.

High-level details: The computation is divided into three main steps: visibility (blue V rectangles in Figure 10.2), BSDF and PDF computations – called *shading* computations from now on – for camera vertices (green C rectangles), and shading computations for light vertices (red L rectangles). For each step, we divide the work into batches of fixed size, each having an associated memory zone on the CPU-side memory (the batch id where results are downloaded is indicated in the download rectangle). On the GPU-side, we use two buffers of fixed size to store the results of the batches (represented by respectively black and white rectangles inside each task). Using batches allows us to compute results of the current batch while downloading results of the previous batch to the CPU, leading to an increased efficiency. This also avoids the need for any array of size $O(N_C \times N_L)$ on the GPU-side, making the N_C and N_L values bounded only by the CPU-side memory capacity. In practice, this provides more space for the scene's geometry that is needed for the visibility tests.

As is, some shading computations will be done even though the linking vertices are not mutually visible. In fact, for the shading models we use [AS00, WMHT07], introducing an array to only compute the useful shading is much less efficient, as computing on CPU and uploading this array for each batch takes more time than directly computing all the shading values.

Low-level details: We use NVidia's CUDA language for GPU computations. The CPU-side work con-

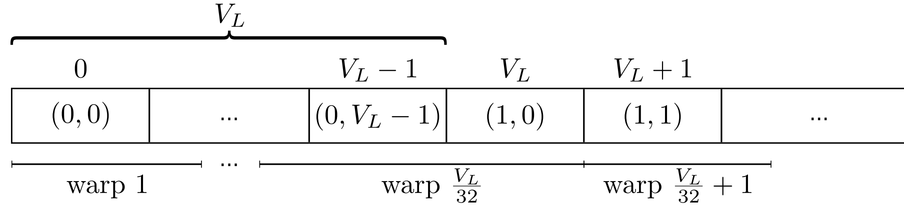


Figure 10.1: Threads organization for the shading of camera vertices. Each vertex is handled by blocks of V_L consecutive threads. At least $(V_L - 2)/32$ warps execute codes with the exact same BSDF parameters, as they all concern the same vertex, leading to high code coherency.

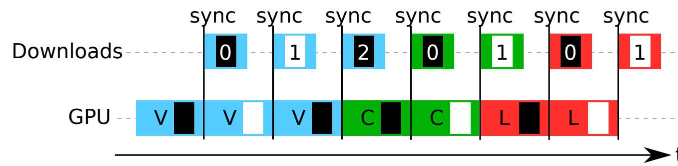


Figure 10.2: Temporal execution of our combination system, not temporally to scale for clarity. The meaning of each element is described in the main text.

sists only in synchronization, and is performed in a CUDA-specific thread, thus not interfering with the main computational threads. All the positions, directions, and BSDF data are stored in linear arrays (structure-of-array organization), that are re-used across populations to avoid memory allocations, and enlarged if needed. Each array is accessible through textures, because each of the values is used many times (once for each linking segment to which a vertex belongs), and generally in coherent ways (subsequent threads are likely to use the same data, or nearby data).

For visibility, we use an adapted version of the radius kd-tree GPU ray-tracing implementation by Segovia [Seg08], which gives a reasonable throughput and is well suited for individual and incoherent rays that are not stored in an array. The rays are effectively built from the thread index idx, by retrieving the camera and light vertices from their indices as (idx/V_L) and $(idx \bmod V_L)$ respectively, where V_L is the number of vertices in the light population.

The same indexing scheme is used for the camera shading computations, which makes a single BSDF processed by consecutive threads, as illustrated by Figure 10.1. Each thread handles one linking segment. This leads to a very good locality in the accesses to the textures containing the BSDF parameters, as well as a very good code coherency in the BSDF evaluation code. In fact, for most warps, the BSDF parameters are the same across all the threads, the only difference between consecutive threads being the directions. For light shading computations, the indexing is reversed (*i.e.* all the linking segments for one light vertex are processed in consecutive threads), to benefit from the same good properties than for the camera shading. All the results are written in linear arrays indexed by the thread index, leading to coalesced writes.

10.5 Implementation of CBPT

Using the combination data computation system described in Section 10.4, we implement CBPT as described in Algorithm 7. Note that population sampling and combinations are done in parallel on all available CPU cores. The main points to note about Algorithm 7 is that we process two couples of populations at the same time, in an interleaved way. As illustrated by Figure 10.3, this allows us to perform GPU processing, CPU processing, downloads, and uploads at the same time. As the computation by the GPU of the combination data does not need any upload and is the only process that performs downloads, there is no contention on the memory bus if the GPU is able to perform transfers in both ways at the same time. In Algorithm 7, *combine()* uses the data computed on GPU and downloaded into the CPU memory to compute the $f^b(\overline{x}, \overline{y})$ contribution for each pair of paths, pushes each of these contributions to the adaptive sampling scheme, and pushes the sum of the contributions for each camera path $\overline{x}$ to the contribution processor used to compute the bidirectional image (the number of contributions in a single $< L >$ value of Equation (4.3) is therefore N_L). As the number of splatted values is small, ensuring thread-safety in the accumulator(s) even with a large number of threads does not create a bottleneck. *compute_lt()* computes light-tracing on all available CPU cores.

Timings for each task of a step are reported in Algorithm 7 for a standard scene, and production-oriented parameters. These timings show the efficiency of our asynchronous computation scheme, as the total wall-clock time needed for one loop is 34.5ms, compared to 60.1ms if all computations had been done synchronously. It also shows that GPU work is done "for free", as the complete time to perform a step is equal to the sum of the times needed by each CPU task, ignoring the GPU one.

Algorithm 7 CBPT algorithm, with timings of each noteworthy element using $N_C = 2000$, $N_L = 15$, $N_T = 1500$, in a scene with 758K triangles and 1.5GB of textures. The time spent by the GPU to compute all the results is given in "async time". The total time needed to perform a step is 34.5ms.

for $t = 0$ to ∞ **do**
  sample($\{\overline{x}\}_t$) *// time:* 13.5*ms*
  upload_async($\{\overline{x}\}_t$)
  sample($\{\overline{y}\}_t$) *// time:* 0.1*ms*
  upload_async($\{\overline{y}\}_t$)
  sample($\{\overline{y_{lt}}\}$) *// time:* 10.1*ms*
  if $t > 0$ **then**
    sync_gpu_comp($t - 1$) *// time:* 0.1*ms*
  end if
  sync_upload(t)
  gpu_comp_async(t) *// async time:* 25.7*ms*
  if $t > 0$ **then**
    combine($\{\overline{x}\}_{t-1}$, $\{\overline{y}\}_{t-1}$) *// time:* 9.6*ms*
    compute_lt($\{\overline{y_{lt}}\}$) *// time:* 1.1*ms*
  end if
end for

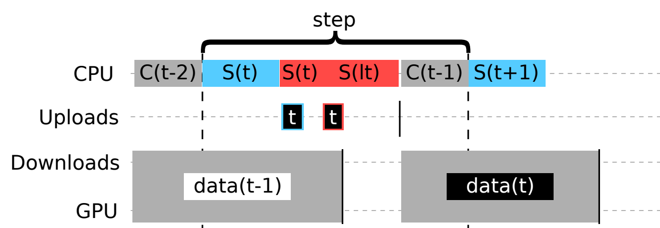


Figure 10.3: Temporal execution of CBPT, not temporally to scale for clarity. Exact timings are given in Algorithm 7. The block labeled C contains both *combine()* and *compute_lt()*. The colors white and black for the rectangles indicate which GPU-side buffer is used to read the population data and store the results.

10.6 Results

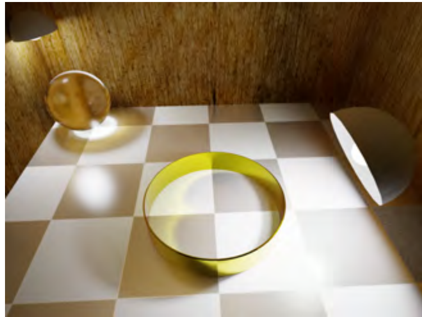
We now analyze the computational behavior of the combination system and CBPT. All the measures are done on an Intel i7 920 2.80GHz system, with an NVidia GTX480 GPU, and 16 GB of CPU-side memory. For our tests of CBPT, we use $N_C = 2000$, $N_L = 15$, and $N_T = 1500$ for all the scenes. These settings are not aimed at providing peak GPU performance, but rather at providing a good compromise between throughput of the GPU part and rendering quality. No adaptive sampling is used, but supporting it is straightforward.

We use three different scenes of various complexities, which are presented in Figure 10.4. We have chosen these challenging scenes for their high lighting complexity:

- The first scene, *ring*, is geometrically simple, but composed of many glossy surfaces. It produces many subtle caustics that typically lead to noticeable noise, for instance on the back wall from the glossy tiles of the floor.

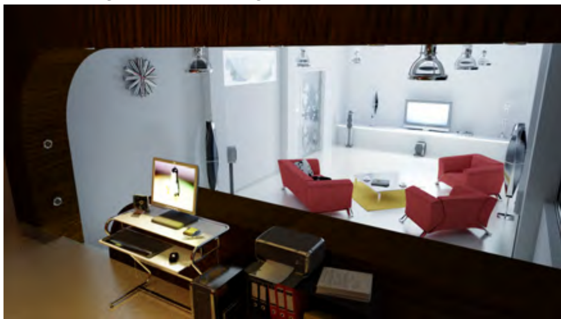
- The *comp* scene, rendered with two different lighting configurations, is much more involved than the *ring* scene. The *lights* version is lit by the ceiling lights, with indirect lighting caused by specular transmission of the light through the glass of the light fixtures. The front room and upper parts of the back room are only indirectly lit. In the *monitors* version, light comes only from the TV and computer monitor. Note the caustics on the wall due to refraction in the twisted pillars made of glass, as well as the caustics beneath the glass table. Nearly all the non-diffuse materials are glossy but not ideal mirrors, leading to very blurry reflections, which is especially visible on the floor.

- The *living* scene is lit by six very small area lights located on the ceiling above the table and the couch. It contains a lot of glossy materials (especially all the wooden objects), of which very few are specular. Note the caustics caused by the shelves on the left, and the completely indirect lighting in the hallway on the right.



ring, 7.4K triangles (2.5, 3.4, 463K)

comp lights, 758K triangles (3.6, 3.2, 570K)



comp monitors, 758K triangles (3.6, 3.6, 620K)

living, 400K triangles (3.7, 3.4, 620K)

Figure 10.4: The three scenes used to test CBPT. We indicate between parentheses the average length in segments of the sampled camera and light paths, as well as the average number of linking edges for each couple of populations in CBPT. Note that the average path lengths for BPT and CBPT are equal, as they use the same code. All the images have been rendered with CBPT. No post-process has been performed except tone-mapping, as our engine produces HDR images. The top-left image has been rendered at a resolution of 1600×1200 pixels in 1 hour. The three others have been rendered at a resolution of 1600×900 pixels, in 4 hours. As CBPT is based on standard Monte-Carlo methods, images at a resolution of 800×450 for the last three scenes can be obtained with a similar quality in 1 hour.

	ring			comp lights		
	GPU	CPU	÷	GPU	CPU	÷
vis	42.6	3.5	12.1	25.4	2.5	10.2
camera	266.7	11.8	22.6	281.7	15.6	18.1
light	266.5	17.3	15.4	280.2	13.0	21.6
	comp monitors			living		
	GPU	CPU	÷	GPU	CPU	÷
vis	25.6	2.9	8.8	32.2	2.1	15.3
camera	275.3	16.2	17.0	256.3	12.5	20.5
light	272.8	15.1	18.1	272.8	15.9	17.6

Table 10.1: Throughputs for visibility (*vis*), camera shading (*camera*), and light shading (*light*), when using the system described in Section 10.4, and when using the 4 physical cores of our processor, plus hyper-threading. The "÷" column gives the ratio of throughputs, corresponding to the actual speedups. Visibility is measured in *millions* of visibility tests per second, camera and light shadings are measured in *millions* of computations of (f_s, p, p^*) tuples per second (see Section 10.3 for the components of the tuple). All the measures take all the memory transfers into account.

10.6.1 Combination throughput

Table 10.1 gives the raw throughputs of visibility and shading values we obtain on CPU and GPU depending on the scene, and the speedup brought by our system. All the measures take all the memory transfers into account. As expected, only visibility thoughputs decrease with the scene's size.

The shading throughput on CPU is quite sensitive to the type of BSDFs (glossy or purely diffuse) that mostly compose the paths of a certain type, explaining the gap that is present for some scenes between the camera and light shading throughputs. This is mostly visible in *comp lights* because of the glass fixture surrounding the light sources. On the other hand, the GPU throughputs are much less affected by this. Despite the need to transfer the results back to GPU, we achieve a 15-20× speedup in average compared to CPU for shading only, consistently on all scenes.

The absolute timings in Table 10.2 give hints about the average time proportions needed by each element of the combination. These timings depend on the number of linking segments that have to be processed for each combination, which depend on the scene.

Figure 10.5 illustrates the impact of batch size on performance, for visibility and shading computations, on the *ring* scene. This allows us to evaluate the impact of different transfers/computation repartitions, and to find optimal batch sizes for the computer we use.

For the visibility computations, even on this geometrically very simple scene, the transfers are not a limiting factor, as the visibility results are packed in a very compact form. Therefore, using batches does not make any noticeable difference on performance as soon as the batches are large enough. Consequently, the major advantage brought by batches for visibility resides in the control we have on the memory-size requirements on GPU, without much impacting on performance.

For more memory-consuming results such as shading ones, the batch size has a large impact on performance, with the additional benefit of using less memory on the GPU. As a matter of fact, using

	ring	comp lights	comp monitors	living
vis	10.9	22.5	24.4	19.3
camera	1.7	2.0	2.2	2.5
light	1.7	2.0	2.2	2.3

Table 10.2: Average time needed to complete each step on GPU, for each scene, in milliseconds.

asynchronism brings a $1.75\times$ speedup, going from 160 millions to 252 millions of computations per second when transfers are done in parallel. Note that the optimal batch sizes are in practice only machine-dependent, as shading computations efficiency does not depend on the scene, and visibility computation efficiency is almost constant for any batch size larger than very small values.

10.6.2 CBPT

To quantify the efficiency of CBPT, we count the number of $L_{i,j}(\overline{x}, \overline{y})$ computations performed during a complete CBPT step, and divide it by the time needed to complete the whole step, including populations sampling and splatting. We call this efficiency measure *basic contributions throughput*. This allows us to have meaningful and consistent results whatever the average path length is in each scene.

Computational efficiency: Table 10.3 gives the basic contributions throughputs obtained using CBPT, and the speedups compared to standard BPT. We compute these values when using CPU-based light-tracing (in this case $N_T = 1500$), to get actual performance, and when not using it ($N_T = 0$), to get the bidirectional-only basic contributions throughput. The CPU version of BPT uses the same code to sample paths, and the same code to compute the $L_{i,j}(\overline{x}, \overline{y})$ values, except that all shading and visibility values are computed on CPU. Both CBPT and standard BPT uniformly sample the image, and do not use any adaptive sampling scheme.

The impact of light-tracing on throughputs is noticeable (around 20%), but the visual impact of a high variance light-tracing part is much more noticeable than the gain in bidirectional part when setting N_T to a very small value, particularly for very short rendering times. For longer rendering times and scenes where caustics are easily captured by light-tracing, N_T can be set to a smaller value, as it will visually converge faster than the bidirectional part.

As shown by timings in Algorithm 7, our reformulation allows us to keep both the CPU and GPU fully loaded, the GPU computation time being masked by the CPU one. The speedup we obtain with "production settings" is consistently greater or equal to $12\times$ on our test scenes. Even if our samples are correlated, the correlation is spread on all the image by our image-sampling process. This effectively avoids the appearance of any noticeable correlation pattern.

Visual comparison with standard BPT: Visually observing noise reduction is made easier when looking at non-converged images, where improvements are clearly visible. Figure 10.6 presents the images obtained by CBPT and BPT after a few seconds of rendering, and after at least 4 samples per pixel have

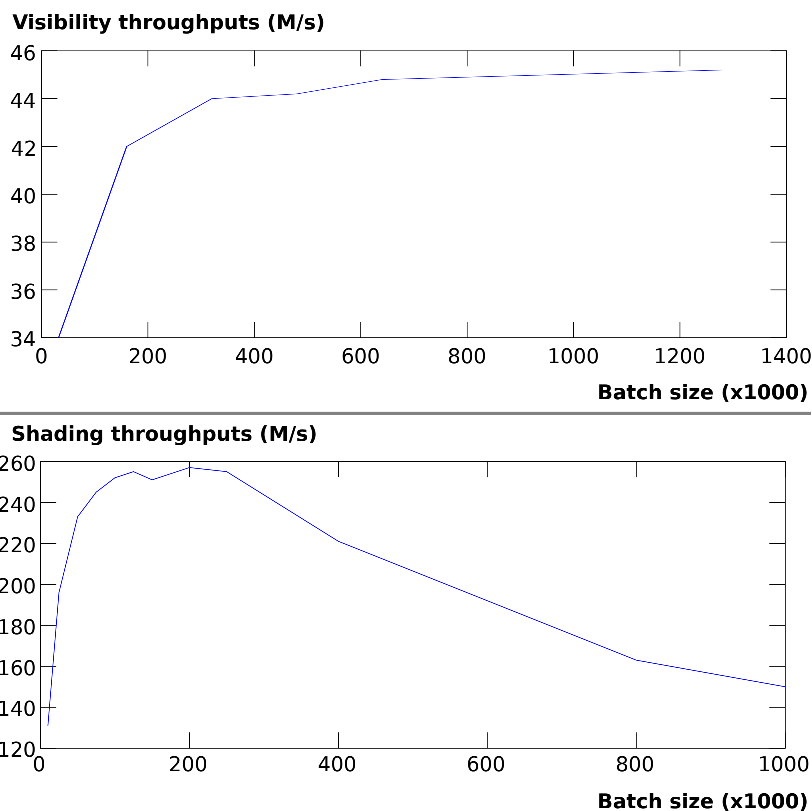


Figure 10.5: Top: Visibility throughput, in millions of tests per second, in function of the number of visibility tests to perform in each batch. Bottom: Shading throughput, in million of shading tuples computation per second, in function of the number of shading computations to perform in each batch.

	CBPT		BPT
	$N_T = 0$	$N_T = 1500$	
ring	20.9 (17.4×)	15.7 (13.1×)	1.2
comp lights	16.2 (16.9×)	12.7 (13.2×)	0.96
comp monitors	16.3 (14.8×)	13.1 (11.9×)	1.1
living	16.5 (21.7×)	12.5 (16.4×)	0.76

Table 10.3: Basic contributions throughput for CBPT and standard BPT, in *millions* of $L_{i,j}(x, y)$ values computed per second, and speedup in parenthesis.

	CBPT		BPT		(x,y)
	prev.	$\simeq 4$ spp	prev.	$\simeq 4$ spp	
ring	1.64	5.15	1.60	5.41	14.3×
comp lights	1.05	3.99	1.38	4.44	13.5×
comp monitors	1.36	4.12	1.61	4.77	12.9×
living	1.62	4.23	1.53	3.86	16.4×

Table 10.4: **Overall speedup measurement:** Average number of samples computed per-pixel for the bidirectional part of the images of Figure 10.6. This is equivalent to the average number of camera paths that have contributed to each pixel. The last column gives the ratio between CBPT and BPT of the number of *pairs of paths* contributing to the bidirectional part of each pixel, which is a good measure of the actual speedup brought by CBPT over standard BPT. For standard BPT, each camera path is combined with one light path, therefore the number of pairs of paths per-pixel is equal to the number of camera paths. For CBPT, as each camera path is combined with N_L light paths, the number of pairs is N_L times the number of camera paths per-pixel. In our tests, we use $N_L = 15$.

been computed by CBPT. As images were stored every 10 seconds, it can happen that more than 4 samples per pixel were actually computed, but both BPT and CBPT got the same computation time. The places where the improvements are most visible are on the diffuse walls, where light-space exploration is crucial to get low variance results, and in the glossy reflections. Table 10.4 gives the actual average number of samples per pixel for the bidirectional part of each image. As expected, the speedups obtained are similar to the ones obtained for the basic contributions throughputs, the little difference coming from the splatting, as BPT needs to splat many more values than CBPT for a same number of pair of paths. The main information of this table is that the images presented in Figure 10.4 would have required from 50 to 66 hours to be computed using standard BPT, *versus* 4 hours with CBPT.

Memory usage and scalability: Table 10.5 gives the memory usage both on CPU and GPU of CBPT. As expected, the size of the combination data on CPU and the populations memory size on GPU are related to the average path length. For populations, we use a conservative allocation scheme, reuse memory between populations, and refit memory zones regularly to keep the consumption low. This can lead to a consequent overestimation of the actual memory size needed, but drastically reduces the number of memory allocations, therefore providing a slight speedup. Despite this, memory requirements remain low for all our scenes on CPU (between 100 and 200MB), and very low on GPU (less than 100MB). Table 10.5 also shows that our method handles scenes much larger than the ones we used. Indeed, the scenes' kd-tree size are kept relatively low even for quite complex scenes (about 50MB). Therefore, scenes that contain several millions of polygons fits in the GPU memory. Moreover, the memory size of populations is negligible except for idiosyncrasies, as even with participating media, the paths remain short ($10 - 20$ vertices on average).

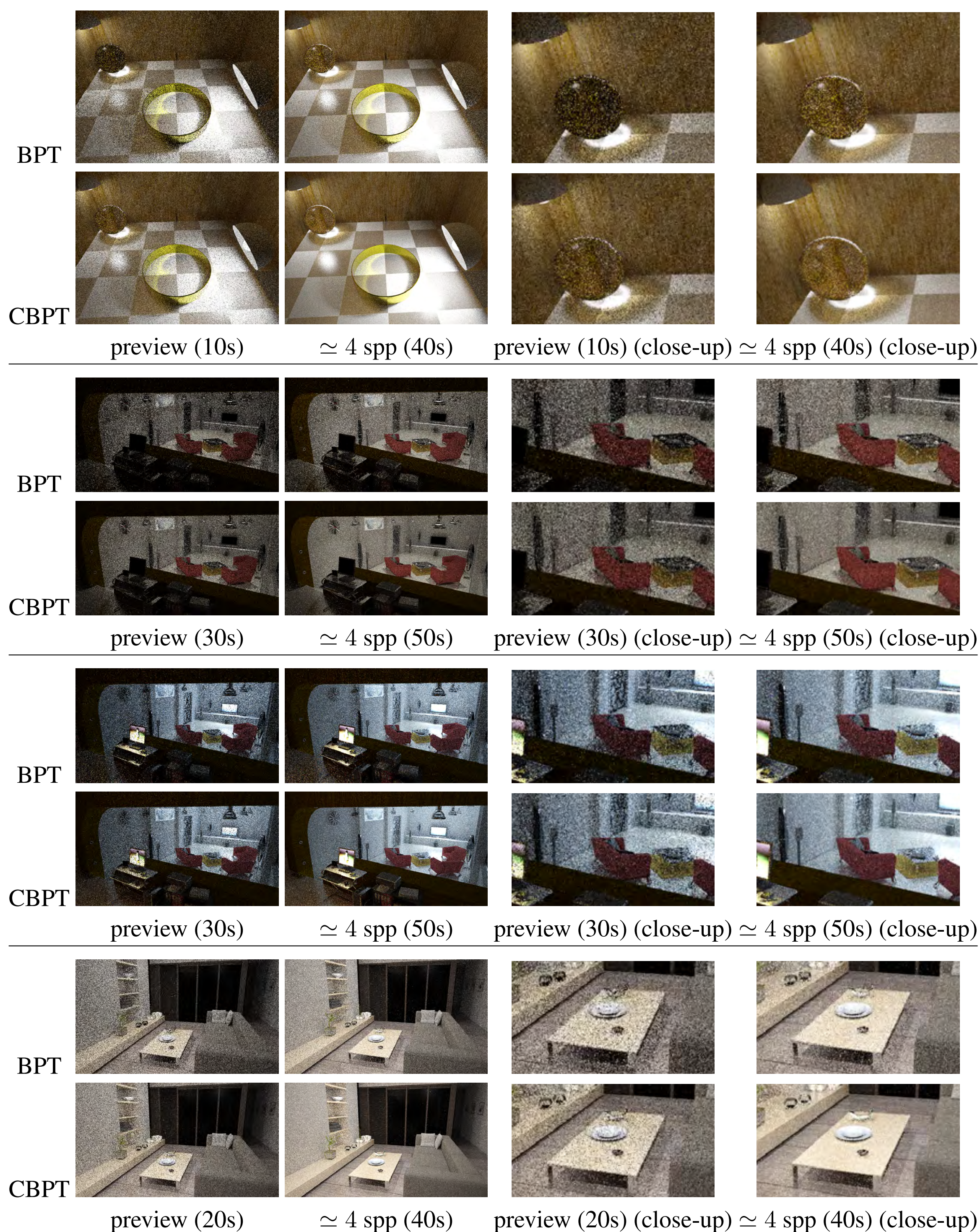


Figure 10.6: Results obtained by BPT and CBPT on our test scenes, after approximately 10 seconds of actual computations, and after CBPT has computed approximately 4 samples per pixel. Images are rendered at 800×450, except *ring* which is rendered at 800×600. Note that for all the scenes, mipmaps are lazily built when first accessed, explaining the 30 and 20 seconds of total rendering times for the preview configuration of the *comp*s and *living* scenes. The time spent building these mipmaps is negligible for the *ring* scene, but takes 16 and 8 seconds in the *comp*s and *living* scenes respectively, and are generally built when sampling the first paths. This also shows that our system can be seamlessly used together with all the usual ways of reducing the peak memory usage, as it does not impact the rendering engine architecture.

	CPU		GPU		
	pops.	comb.	kd-tree	pops.	comb.
ring	73.3	48.0	0.47	3.8	23.5
comp lights	91.2	66.0	56.1	4.8	23.5
comp monitors	95.0	72.3	56.1	4.8	23.5
living	60.8	60.3	58.5	5.0	23.5

Table 10.5: Memory usage for populations and combination data on CPU, and memory usage for the kd-tree, the populations data (position, BSDFs parameters, *etc.*), and all the batch buffers, in MB.

10.7 Conclusion

CBPT allows us to efficiently handle most scenes without participating media and distant light sources. We have shown that it is more than an order of magnitude faster than standard BPT on various test scenes, without affecting the size of the datasets or the flexibility of the underlying rendering engine in terms of shaders, and models of lights and cameras. The only added approximation is the need to tessellate the geometry beforehand. Participating media can be easily handled by using the unified representation developed in Chapter 9, computing the transmittance along the linking segments on GPU. In fact, it requires low memory, and can therefore fit in GPU memory even for complex participating media. However, care has to be taken in order to maximize efficiency: mechanisms such as stream compaction [BOA09] should be used, because in contrast to BSDF evaluations which are cheap, transmittance computations are more expensive, and should therefore be done only when necessary, packing the linking segments requiring it on adjacent threads.

Part IV

Final summary and conclusions

Physically-based rendering as presented in this dissertation is a compromise between realism and computation times. Many rendering dissertations focus their presentation on the numerical tools used to solve the light transport equations derived in Chapter 1, taking these equations for granted, without highlighting the asumptions that are made and their consequences. These equations put important requirements on the different elements used to described a scene, motivating our brief review in Chapter 2. In a similar way, it is not common to find a discussion of the mathematical foundations on which the Monte-Carlo method is built, while these foundations are crucial for a correct understanding of the higher level tools. This is why, instead of writing yet another Monte-Carlo review, which would be less precise or less complete than reference texts [Vea97, PH04], we prefer to present in an as clear as possible way what are the entities we manipulate in the equations, what assumptions have been done to get them, and what are the mathematical or computational objects at the heart of the numerical integration performed during rendering.

From a software point of view, this dissertation covers all parts of a Monte-Carlo-based rendering engine, with contributions in most parts. We first present a complete software architecture targeted at increased flexibility, which is an extension of PBRT [PH04]. This flexibility is assessed by considering the very different natures of the technical contributions we presented. Notably, CBPT (Chapter 10) can not be expressed easily in the context of PBRT. Arbitrary number of contribution processors can be used and combined, using an arbitrary organization of accumulators, thanks to our abstract contribution processor system. It allowed us to test our sample-space bright-spots removal method (Chapter 6) with other removal methods on the same radiance samples. Except for representativities which imply a low-level modification for local path sampling, the other contributions can be used together and with other algorithms in a black-box way.

Summary of contributions

This thesis focuses on two topics: mathematical and computation time robustness, and efficiency.

Robustness: The goal of this part is to get a consistent behavior with respect to quality and computation times. Two directions are explored.

First, we make the observation that a one-fits-all integration method is hard to develop: after more than thirty years of research, all algorithms developed still have problems with specific lighting configurations, these configurations not being the same for each algorithm. We therefore take a side approach: instead of developing yet another new integration method which would have its own caveats, we improve the mathematical robustness of all algorithms relying on local path sampling (this covers most of nowadays rendering algorithms) (Chapter 5), we perform a more robust accumulation (Chapter 6), and we provide speckles-free HDR images using our image-based despeckling, ensuring that no problems linked to bright spots arise in the HDR processing pipeline (Chapter 7). As a final step, we focus processing power on places where convergence is harder to reach, taking into account outliers and ensuring that error decreases on all the image (Chapter 8).

The second direction explored aims at improving computation time stability. Solving the light trans-

port equations in presence of participating media requires to sample free-paths in these media and to compute transmittance. The computation time of these operations largely depends on the participating medium representation. To ensure that computation times are consistent whatever the base representation is, we develop a multi-resolution unified representation, whose approximation error is tightly controlled, and which provides efficient sampling and transmittance computations (Chapter 9). The construction of this structure is made possible by the development of a new numerical integration method, designed to ensure robustness, accuracy and efficiency.

Efficiency: This part is a study of how to increase the efficiency of the integration part, without restricting the freedom of expression of the user. This flexibility requirement leads us to develop an hybrid CPU/GPU method, based on bidirectional path-tracing (Chapter 10). Our method allows us to obtain ten times faster rendering than with a standard pure CPU implementation.

Place of our work in a more global context

The contributions on robustness easily allow Monte-Carlo-based rendering engines to be more robust, *without imposing any strict technical choice*. From an algorithmic point of view, the integration method, which is the core of a rendering engine, can be chosen freely and can use representativities to improve sampling. Our accumulation system is in-between integration and standard accumulation. Adaptive sampling can be either our error-measure based on approximate-median relative error, or can be chosen freely, using alternation to ensure correct results. Finally, our HDR despeckler is in-between standard accumulation and the HDR processing pipeline. From a scene description point-of-view, our unified participating media representation allows us to approximate any other representation, as only point-evaluation is required: this representation can be used internally, while the user can still use any representation he wants. Globally, these contributions are generic tools that can be used in any rendering engine based on the Monte-Carlo method, and therefore a perennial step toward more robust rendering.

Finally, we believe that our hybrid CPU/GPU rendering method is a perennial step toward flexible and efficient rendering. At the moment of writing, most algorithms using the GPU run fully on it, severely restricting the flexibility and the size of the datasets they can handle. We demonstrated that it is possible to fully exploit the GPU while still using arbitrarily complex scenes, in terms of procedural shaders, textures, *etc.*. Considered that hardware architectures tend toward a large number of CPU-like cores and that memory bandwidth and synchronization will most likely be the major bottlenecks, our method can be reused advantageously, as it is designed to minimize both aspects while maximizing the amount of computations to perform.

Lessons learned from this thesis

Generic enhancement tools have many advantages over all-in-one solutions: they do not impose technical choices, and make it easier to develop more efficient algorithms. We therefore believe that this kind of research should be pursued. Robustness is often overlooked, efficiency for a subset of scenes being

preferred. However, robustness is the key to get an algorithm being practical for end-users. As a matter of fact, users are the best to find the one situation where a not-totally-robust algorithm fails.

Orthogonally to robustness, efficiency is reached by large-scale parallelism nowadays. However, it is likely that GPUs are replaced by generic CPU architectures with tens or hundreds of cores, offering the same computation power as GPUs, together with the same flexibility as nowadays CPUs. Although a lot of improvements can still be done with GPUs, we think that research for high-performance high-quality rendering should preferably target large-scale CPU-like parallelism.

Bibliography

[ARBJ03] S. Agarwal, R. Ramamoorthi, S. Belongie, and H.W. Jensen. Structured importance sampling of environment maps. In *SIGGRAPH '03*, pages 605–612, 2003.

[AS00] M. Ashikhmin and P. Shirley. An anisotropic phong light reflection model. *JGT*, 5:25–32, 2000.

[Bar86] A.H. Barr. Ray tracing deformed surfaces. In *SIGGRAPH '86*, SIGGRAPH '86, pages 287–296, 1986.

[BBS+09] B.C. Budge, T. Bernardin, J.A. Stuart, S. Sengupta, K.I. Joy, and J.D. Owens. Out-of-core data management for path tracing on hybrid resources. In *Eurographics '09*, 2009.

[BGH05] D. Burke, A. Ghosh, and W. Heidrich. Bidirectional importance sampling for direct illumination. In *EGSR '05*, pages 147–156, 2005.

[BOA09] M. Billeter, O. Olsson, and U. Assarsson. Efficient stream compaction on wide simd many-core architectures. In *HPG '09*, HPG '09, pages 159–166, 2009.

[Bor91] C.F. Borges. Trichromatic approximation for computer graphics illumination models. *SIGGRAPH '91*, 25(4):101–104, July 1991.

[CAM08] P. Clarberg and T. Akenine-Möller. Practical Product Importance Sampling for Direct Illumination. In *Eurographics '08*, pages 681–690, 2008.

[Cha53] S. Chandrashekhar. *Radiative Transfer*. Dover Publications, 1953.

[Cha98] T.R. Chandrupatla. An efficient quadratic fit–sectioning algorithm for minimization without derivatives. *Computer Methods in Applied Mechanics and Engineering*, 152(1-2):211 – 217, 1998.

[CJAMJ05] P. Clarberg, W. Jarosz, T. Akenine-Möller, and H.W. Jensen. Wavelet importance sampling: Efficiently evaluating products of complex functions. In *SIGGRAPH '05*, pages 1166 – 1175, 2005.

[CNS+11] C. Crassin, F. Neyret, M. Sainz, S. Green, and E. Eisemann. Interactive indirect illumination using voxel cone tracing. In *PG '11*, 2011.

[Cra11] C. Crassin. *GigaVoxels: A voxel-based rendering pipeline for efficient exploration of large and detailed scenes*. PhD. thesis, Université de Grenoble, 2011.

[CTE05] D. Cline, J. Talbot, and P. Egbert. Energy redistribution path-tracing. In *SIGGRAPH '05*, pages 1186–1195, 2005.

[DBB02] P. Dutre, K. Bala, and P. Bekaert. *Advanced Global Illumination*. A. K. Peters, Ltd., Natick, MA, USA, 2002.

[Deh77] P. Deheuvels. Estimation non paramétrique de la densité par histogrammes généralisés (in french). *Revue de Statistique Appliquée*, 25:5–42, 1977.

[DF95] D. Dasgupta and S. Forrest. Novelty detection in time series data using ideas from immunology. In *International Conference on Intelligent Systems*, 1995.

[DGMR05] R. Douc, A. Guillin, J.-M. Marin, and C.P. Robert. Minimum variance importance sampling via Population Monte Carlo. Research Report RR-5699, INRIA, 2005.

[DHK08] H. Dammertz, J. Hanika, and A. Keller. Shallow bounding volume hierarchies for fast SIMD ray tracing of incoherent rays. In *EGSR '08*, 2008.

[DLW93] P. Dutré, E.P. Lafortune, and Y. Willems. Monte Carlo light tracing with direct computation of pixel intensities. In *Compugraphics '93*, pages 128–137, 1993.

[DR07] P.J. Davis and P. Rabinowitz. *Methods of Numerical Integration: Second Edition*. Dover Publications, 2007.

[DSHL10] H. Dammertz, D. Sewtz, J. Hanika, and H. Lensch. Edge-avoiding a-trous wavelet transform for fast global illumination filtering. In *HPG 2010*, pages 67–75, 2010.

[Dut94] P. Dutré. *Mathematical frameworks and Monte Carlo Algorithms for global illumination in computer graphics*. PhD. thesis, Katholieke Universiteit Leuven, 1994.

[DWB+06] M. Donikian, B. Walter, K. Bala, S. Fernandez, and D.P. Greenberg. Accurate direct illumination using iterative adaptive sampling. *IEEE Transactions on Visualization and Computer Graphics*, 12(3):353–364, 2006.

[DWR10] C. DeCoro, T. Weyrich, and S. Rusinkiewicz. Density-based outlier rejection in Monte Carlo rendering. In *PG '10*, 2010.

[Ebe97] D.S. Ebert. Volumetric modeling with implicit functions: a cloud is born. In *SIGGRAPH '97*, page 147, 1997.

[EMP+02] D.S. Ebert, F.K. Musgrave, D. Peachey, K. Perlin, and S. Worley. *Texturing and Modeling: A Procedural Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2002.

[Epa69] V.A. Epanechnikov. Non-parametric estimation of a multivariate probability density. *Theory of Probability and its Applications*, 14(1):153–158, 1969.

[Fan06] S. Fan. *Sequential monte carlo methods for physically based rendering*. PhD thesis, University of Wisconsin at Madison, Madison, WI, USA, 2006. AAI3234657.

[FCH+06] S. Fan, S. Chenney, B. Hu, K-W. Tsui, and Y-C. Lai. Optimizing control variate estimators for rendering. 25(3):351–358, 2006.

[FCL05] S. Fan, S. Chenney, and Y-C. Lai. Metropolis photon sampling with optional user guidance. In *EGSR '05*, pages 127–138, 2005.

[Gen87] A. Genz. *Numerical integration – recent developments*, chapter A package for testing multiple integration subroutines, pages 337–340. Reidel, 1987.

[GG07] G. Guennebaud and M. Gross. Algebraic point set surfaces. In *SIGGRAPH '07*, 2007.

[Gon11] P. Gonnet. A Review of Error Estimation in Adaptive Quadrature. `http://arxiv.org/abs/1003.4629`, 2011.

[GPP+10] O. Gourmel, A. Pajot, M. Paulin, L. Barthe, and P. Poulin. Fitted bvh for fast raytracing of metaballs. In *Eurographics '10*, 2010.

[HA04] V. Hodge and J. Austin. A survey of outlier detection methodologies. *Artif. Intell. Rev.*, 22(2):85–126, 2004.

[Hah05] T. Hahn. Cuba - a library for multidimensional numerical integration. *Computer Physics Communications*, 168(2):78–95, 2005.

[HJ09] T. Hachisuka and H.W. Jensen. Stochastic progressive photon mapping. In *SIGGRAPH Asia '09 papers*, pages 1–8, 2009.

[HOJ08] T. Hachisuka, S. Ogaki, and H.W. Jensen. Progressive photon mapping. *ACM Trans. Graph.*, 27(5):1–8, 2008.

[HP02] H. Hey and W. Purgathofer. Importance sampling with hemispherical particle footprints. In *SCCG '02*, pages 107–114, 2002.

[Jen95] H.W. Jensen. Importance driven path tracing using the photon map. In *EGWR '05*, pages 326–335, 1995.

[Jen96] H.W. Jensen. Global illumination using photon maps. In *EGWR '96*, pages 21–30, 1996.

[JMLH01] H.W. Jensen, S.R. Marschner, M. Levoy, and P. Hanrahan. A practical model for subsurface light transport. In *SIGGRAPH '01*, pages 511–518, August 2001.

[JNSJ11] W. Jarosz, D. Nowrouzezahrai, I. Sadeghi, and H.W. Jensen. A comprehensive theory of volumetric radiance estimation using photon points and beams. *ACM Trans. Graph.*, 30:5:1–5:19, 2011.

[KA91] D. Kirk and J. Arvo. Unbiased sampling techniques for image synthesis. In *SIGGRAPH '91*, pages 153–156, 1991.

[Kaj86] J.T. Kajiya. The rendering equation. In *SIGGRAPH '86*, pages 143–150, 1986.

[Kel96] A. Keller. Quasi-monte carlo methods in computer graphics. *Zeitschrift fur Angewandte Mathematik und Mechanik*, 76(3):109–112, 1996.

[KH01] A. Keller and W. Heidrich. Interleaved sampling. In *EGWR'01*, pages 269–276, 2001.

[Kno06] A. Knoll. A survey of octree volume rendering methods. In *1st IRTG Workshop*, 2006.

[Kno07] A. Knoll. A survey of implicit surface rendering methods, and a proposal for a common sampling framework. In *2nd IRTG Workshop*, 2007.

[Knu98] D.E. Knuth. *The art of computer programming : semi-numerical algorithms*, volume 2, chapter 3, page 232. Addison-Wesley, 1998.

[KSKAC02] C. Kelemen, L. Szirmay-Kalos, G. Antal, and F. Csonka. A simple and robust mutation strategy for the Metropolis light transport algorithm. In *Eurographics '02*, pages 531–540, 2002.

[KW86] M.H. Kalos and P.A. Whitlock. *Monte Carlo Methods: Basics*. J. Wiley & Sons, 1986.

[KW06] A. Keller and C. Wächter. Instant ray tracing: The bounding interval hierarchy. In *EGSR '06*, pages 139–149, 2006.

[KZ11] C. Knaus and M. Zwicker. Progressive photon mapping: A probabilistic approach. *ACM Trans. Graph.*, 30:25:1–25:13, May 2011.

[LC87] W.E. Lorensen and H.E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *SIGGRAPH '87*, pages 163–169, 1987.

[Lem09] C. Lemieux. *Monte Carlo and Quasi-Monte Carlo Sampling*. Springer Series in Statistics, 2009.

[Lep78] G.P. Lepage. A new algorithm for adaptive multidimensional integration. *Journal of Computational Physics*, 27:192–203, 1978.

[Lev88] M. Levoy. Display of Surfaces from Volume Data. *IEEE CGA*, 8(3):29–37, May 1988.

[LFCD07] Y-C. Lai, S. Fan, S. Chenney, and C. Dyer. Photorealistic image rendering with population Monte Carlo energy redistribution. In *EGSR '07*, pages 287–296, 2007.

[LRR04] J. Lawrence, S. Rusinkiewicz, and R. Ramamoorthi. Efficient BRDF importance sampling using a factored representation. In *SIGGRAPH '04*, pages 496–505, 2004.

[Lux10] LuxRender. Luxrays. http://www.luxrender.net/wiki/index.php?title=LuxRays, 2010.

[LW93] E.P. Lafortune and Y.D. Willems. Bi-directional path tracing. In *Compugraphics '93*, pages 145–153, 1993.

[LW94] E.P. Lafortune and Y.D. Willems. Using the modified phong reflectance model for physically based rendering. Technical Report CW197, Katholieke Universiteit Leuven, 1994.

[MBC79] M.D. McKay, R.J. Beckman, and W.J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):pp. 239–245, 1979.

[McC99] M.D. McCool. Anisotropic diffusion for monte-carlo noise reduction. *ACM Trans. Graph.*, 18(2):171–194, 1999.

[MRR+53] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A. H. Teller, and E. Teller. Equations of state calculations by fast computing machines. *Journal of Chemical Physics 21*, 1953.

[MWB+96] J. Menon, B. Wyvill, C. Bajaj, J. Bloomenthal, B. Guo, J. Hart, and G. Wyvill. Implicit surfaces for geometric modeling and computer graphics, 1996.

[ODJ04] V. Ostromoukhov, C. Donohue, and P.M. Jodoin. Fast hierarchical importance sampling with blue noise properties. In *SIGGRAPH '04*, pages 488–495, 2004.

[OGG09] C. Oztireli, G. Guennebaud, and M. Gross. Feature preserving point set surfaces based on non-linear kernel regression. In *Eurographics '09*, pages 493–501, 2009.

[Owe92] A.B. Owen. A central limit theorem for latin hypercube sampling. *Journal of the Royal Statistical Society. Series B (Methodological)*, 54(2):pp. 541–551, 1992.

[Par07] S. Paris. A gentle introduction to bilateral filtering and its applications. In *SIGGRAPH '07 courses*, SIGGRAPH '07, 2007.

[PBP11a] A. Pajot, L. Barthe, and M. Paulin. Effective despeckling of hdr images. In *SIGGRAPH Asia '11, Technical Sketches*, 2011.

[PBP11b] A. Pajot, L. Barthe, and M. Paulin. Robust adaptive sampling for monte-carlo-based rendering. In *SIGGRAPH Asia '11, Technical Posters*, 2011.

[PBP11c] A. Pajot, L. Barthe, and M. Paulin. Sample-space bright-spot removal using density estimation. In *GI '11*, 2011.

[PBPP11a] A. Pajot, L. Barthe, M. Paulin, and P. Poulin. Combinatorial bidirectional path-tracing for efficient hybrid cpu/gpu rendering. In *Eurographics '11*, 2011.

[PBPP11b] A. Pajot, L. Barthe, M. Paulin, and P. Poulin. Representativity for robust and adaptive multiple importance sampling. *IEEE TVCG*, 17(8):1108 –1121, aug. 2011.

[PH04] M. Pharr and G. Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., 2004.

[Pha] M. Pharr. Extended photon map implementation. http://www.pbrt.org/plugins/exphotonmap.pdf.

[PSS99] A.J. Preetham, P. Shirley, and B. Smits. A practical analytic model for daylight. In *SIGGRAPH '99*, pages 91–100, 1999.

[Pur87] W. Purgathofer. A statistical method for adaptive stochastic sampling. *Computers & Graphics*, 11(2):157 – 162, 1987.

[PWP08] V. Pegoraro, I. Wald, and S.G. Parker. Sequential monte carlo adaptation in low-anisotropy participating media. In *EGSR '08*, 2008.

[RCL+08] F. Rousselle, P. Clarberg, L. Leblanc, V. Ostromoukhov, and P. Poulin. Efficient product sampling using hierarchical thresholding. In *CGI '08*, pages 465–474, 2008.

[RRS00] S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. *SIGMOD Rec.*, 29(2):427–438, 2000.

[RW94] H.E. Rushmeier and G.J. Ward. Energy preserving non-linear filters. In *SIGGRAPH '94*, pages 131–138, 1994.

[RWPD05] E. Reinhard, G. Ward, S. Pattanaik, and P. Debevec. *High Dynamic Range Imaging: Acquisition, Display, and Image-Based Lighting (The Morgan Kaufmann Series in Computer Graphics)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[Seg08] B. Segovia. Radius-CUDA raytracing kernel. http://bouliiii.blogspot.com/2008/08/real-time-ray-tracing-with-cuda-100.html, 2008.

[SFD09] M. Stich, H. Friedrich, and A. Dietrich. Spatial splits in bounding volume hierarchies. In *HPG '09*, HPG '09, pages 7–13, 2009.

[Sil86] B.W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, 1986.

[SIP07] B. Segovia, J.C. Iehl, and B. Péroche. Coherent metropolis light transport with multiple-try mutations. Technical Report RR-LIRIS-2007-015, LIRIS UMR 5205 CNRS/INSA de Lyon/Université Claude Bernard Lyon 1/Université Lumière Lyon 2/Ecole Centrale de Lyon, April 2007.

[SJ09] B. Spencer and M.W. Jones. Into the blue: Better caustics through photon relaxation. In *Eurographics '09*, pages 319–328, 2009.

[SKTM11] L. Szirmay-Kalos, B. Tóth, and M. Magdics. Free path sampling in high resolution inhomogeneous participating media. *CGF*, 30:85–97, 2011.

[Tal05] J.F. Talbot. Importance resampling for global illumination. Master's thesis, Brigham Young University, 2005.

[TJ97] R. Tamstorf and H.W. Jensen. Adaptive sampling and bias estimation in path tracing. In *EGWR '97*, pages 285–295, 1997.

[TM98] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *1998 IEEE International Conference on Computer Vision*, pages 839–846, 1998.

[TM00] W.W. Tsang and G. Marsaglia. The ziggurat method for generating random variables. *Journal of Statistical Software*, (i08), 2000.

[Tur93] B.A. Turlach. Bandwidth selection in kernel density estimation: A review. In *CORE and Institut de Statistique*, pages 23–493, 1993.

[Vea97] E. Veach. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD. thesis, Stanford University, 1997.

[VG94] E. Veach and L.J. Guibas. Bidirectional estimators for light transport. In *EGWR '94*, pages 147–162, 1994.

[VG95] E. Veach and L.J. Guibas. Optimally combining sampling techniques for monte carlo rendering. In *SIGGRAPH '95*, pages 419–428, 1995.

[VG97] E. Veach and L.J. Guibas. Metropolis light transport. In *SIGGRAPH '97*, pages 65–76, 1997.

[WA09] R. Wang and O. Akerlund. Bidirectional importance sampling for unstructured illumination. *Eurographics '09*, pages 269–278, 2009.

[Wal07] I. Wald. On fast construction of SAH based bounding volume hierarchies. In *2007 Eurographics/IEEE Symposium on Interactive Ray Tracing*, 2007.

[Wei96] J. Weickert. Anisotropic diffusion in image processing, 1996.

[Wet94] D. Wettschereck. *A study of distance-based machine learning algorithms*. PhD thesis, Oregon State University, 1994. Adviser-Dietterich, Thomas G.

[Wil03] H.S. Wilf. *Algorithms and Complexity*. AK Peters, 2003.

[WMHT07] B. Walter, S.R. Marschner, L. Hongsong, and K.E. Torrance. Microfacet models for refraction through rough surfaces. In *EGSR '07*, pages 195–206, 2007.

[WMW86] G. Wyvill, C. McPheeters, and B. Wyvill. Data structure for soft objects. *The Visual Computer*, 2(4):227–234, August 1986.

[WS00] G. Wyszecki and W.S. Stiles. *Color Science: Concepts and Methods, Quantitative Data and Formulae*. Wiley-Interscience, 2 edition, August 2000.

[WW07] A. Weidlich and A. Wilkie. Arbitrarily layered micro-facet surfaces. In *GRAPHITE '07*, pages 171–178, 2007.

[WW11] A. Wilkie and A. Weidlich. Physically plausible model for light emission from glowing solid objects. In *EGSR '11*, 2011.

[XP05] R. Xu and S.N. Pattanaik. A novel monte carlo noise reduction operator. *IEEE CGA*, 25(2):31–35, 2005.

[XSXZ07] Q. Xu, M. Sbert, L. Xing, and J. Zhang. A novel adaptive sampling by tsallis entropy. In *CGIV '07*, pages 5–10, 2007.

[YIC+10] Y. Yue, K. Iwasaki, B-Y. Chen, Y. Dobashi, and T. Nishita. Unbiased, adaptive stochastic sampling for rendering inhomogeneous participating media. In *SIGGRAPH ASIA '10*, pages 177:1–177:8, 2010.

Personal bibliography

[1] O. Gourmel, A. Pajot, M. Paulin, L. Barthe, and P. Poulin. Fitted bvh for fast raytracing of metaballs. In *Eurographics '10*, 2010.

[2] A. Pajot, L. Barthe, and M. Paulin. Effective despeckling of hdr images. In *SIGGRAPH Asia '11, Technical Sketches*, 2011.

[3] A. Pajot, L. Barthe, and M. Paulin. Robust adaptive sampling for monte-carlo-based rendering. In *SIGGRAPH Asia '11, Technical Posters*, 2011.

[4] A. Pajot, L. Barthe, and M. Paulin. Sample-space bright-spot removal using density estimation. In *GI '11*, 2011.

[5] A. Pajot, L. Barthe, M. Paulin, and P. Poulin. Combinatorial bidirectional path-tracing for efficient hybrid cpu/gpu rendering. In *Eurographics '11*, 2011.

[6] A. Pajot, L. Barthe, M. Paulin, and P. Poulin. Representativity for robust and adaptive multiple importance sampling. *IEEE TVCG*, 17(8):1108 –1121, aug. 2011.

Résumé

Le rendu fondé sur la physique est utilisé pour le design, l'illustration ou l'animation par ordinateur. Ce type de rendu produit des images photo-réalistes en résolvant les équations qui décrivent le transport de la lumière dans une scène.

Bien que ces équations soient connues depuis longtemps, et qu'un grand nombre d'algorithmes aient été développés pour les résoudre, il n'en existe pas qui puisse gérer de manière efficace toutes les scènes possibles. Plutôt qu'essayer de développer un nouvel algorithme de simulation d'éclairage, nous proposons d'améliorer la robustesse de la plupart des méthodes utilisées à ce jour et/ou qui sont amenées à être développées dans les années à venir.

Nous faisons cela en commençant par identifier les sources de non-robustesse dans un moteur de rendu basé sur la physique, puis en développant des méthodes permettant de minimiser leur impact. Le résultat de ce travail est un ensemble de méthodes utilisant différents outils mathématiques et algorithmiques, chacune de ces méthodes visant à améliorer une partie spécifique d'un moteur de rendu. Nous examinons aussi comment les architectures matérielles actuelles peuvent être utilisées à leur maximum afin d'obtenir des algorithmes plus rapides, sans ajouter d'approximations. Bien que les contributions présentées dans cette thèse aient vocation à être combinées, chacune d'entre elles peut être utilisée seule : elles sont techniquement indépendantes les unes des autres.

Mots-clés: Rendu basé sur la physique, intégration numérique, méthodes stochastiques, robustesse, calcul haute-performance

Abstract

Physically-based rendering is used for design, illustration or computer animation. It consists in producing photorealistic images by solving the equations which describe how light travels in a scene.

Although these equations have been known for a long time and many algorithms for light simulation have been developed, no algorithm exists to solve them efficiently for any scene. Instead of trying to develop a new algorithm devoted to light simulation, we propose to enhance the robustness of most methods used nowadays and/or which can be developed in the years to come.

We do this by first identifying the sources of non-robustness in a physically-based rendering engine, and then addressing them by specific algorithms. The result is a set of methods based on different mathematical or algorithmic methods, each aiming at improving a different part of a rendering engine. We also investigate how the current hardware architectures can be used at their maximum to produce more efficient algorithms, without adding approximations. Although the contributions presented in this dissertation are meant to be combined, each of them can be used in a standalone way: they have been designed to be internally independent of each other.

Keywords: Physically-based rendering, numerical integration, stochastic methods, robustness, high-performance computing