Rendu interactif d'image hyper spectrale par illumination globale pour la prédiction de la signature infrarouge d'aéronefs

Romain Hoarau

AIX-MARSEILLE UNIVERSITÉ
ONERA
ÉCOLE DOCTORALE EN MATHÉMATIQUES ET INFORMATIQUE DE MARSEILLE (ED184)
ONERA / DOTA / MVA
LIS UMR 7020 CNRS / AMU - G-Mod

Thèse présentée pour obtenir le grade universitaire de docteur

Discipline : Informatique

Romain HOARAU

Titre de la thèse : Rendu interactif d'image hyper spectrale par illumination globale pour la prédiction de la signature infrarouge d'aéronefs

Thesis: Interactive hyper spectral image rendering by global illumination for aircraft infrared signature prediction

Soutenue le 19/12/2019 devant le jury composé de :

Kadi BOUATOUCH	Université de Rennes / IRISA	Rapporteur
Stéphane MÉRILLOU	Université de Limoges / XLIM / ASALI	Rapporteur
Mathias PAULIN	Université Paul Sabatier / IRIT	Examinateur
Sidonie LEFEBVRE	ONERA / DOTA / MPSO	Examinatrice
Éric COIRO	ONERA / DOTA / MVA	Encadrant
Sébastien THON	LIS UMR 7020 CNRS / AMU - G-Mod	Co-directeur de thèse
Romain RAFFIN	LIS UMR 7020 CNRS / AMU - G-Mod	Directeur de thèse

Numéro national de thèse/suffixe local : 2019AIXM0479/029ED184

Résumé

Le dimensionnement de capteur est un enjeu majeur pour le domaine de la détection d'aéronefs dans les bandes infrarouge et visible. Dans cette optique, il est nécessaire de simuler ces capteurs via des modèles et un nombre conséquent d'images spectrales d'aéronefs dans divers scénarios. L'obtention de ces images via des campagnes aériennes de mesure est malheureusement onéreuse, chronophage et difficile. Une simulation rapide et robuste de ces données s'impose donc.

Afin de répondre aux besoins de précision, la complexité spectrale des scènes aériennes combinée à l'évolution technologique de la furtivité des aéronefs nous amène à utiliser des algorithmes d'illumination globale à haute dimension spectrale. Dans ces conditions, ces algorithmes posent d'importants problèmes de consommation mémoire et de temps de calcul. Le projet de recherche de cette thèse s'inscrit dans le cadre de ces problématiques.

Dans un premier temps, nous nous sommes focalisés sur l'algorithme du Path Tracing et la parallélisation GPU pour le rendu d'images spectrales. Nous avons d'abord analysé les problèmes de ce type de rendu sur GPU. Nous avons ensuite proposé le DPEPT (« Deferred Path Evaluation Path Tracing ») et un schéma de parallélisation spectral qui permettent de réduire significativement la consommation mémoire et les temps de calcul.

Dans un second temps, nous avons cherché à réduire la charge de calcul spectrale de la simulation en prenant soin de ne pas dégrader la précision. À cet égard, nous avons proposé de généraliser le rendu spectral stochastique d'image dans l'espace XYZ en rendu d'image spectrale stochastique. Cette nouvelle méthode permet de rendre directement et de manière plus précise et rapide les canaux d'un capteur en diminuant la dimension spectrale de la simulation, quand les données spectrales d'entrée sont complexes.

Pour conclure, les travaux de cette thèse permettent de simuler de manière précise des images multi, hyper et ultra spectrales. Le temps interactif peut être atteint dans notre cas (rendu d'image à faible résolution spatiale pour des scènes de complexité géométrique intermédiaire) en multi et hyper spectrale.

Mots clés : Illumination globale, Rendu spectral, Calcul sur GPU, Synthèse d'image infrarouge

Abstract

Sensor dimensioning is a major issue for the aircraft detection field in the infrared and the visible bands. In this vein, it is appropriate to simulate these sensors via models and a consequent set of spectral images in several scenarios. The acquisition of these images via an airborne measure campaign is unfortunately costly, time-consuming and difficult. A robust and fast simulation of these data is hence very appealing.

In order to answer the needs of accuracy, the spectral complexity of the airborne scenes combined with the technological evolution of the aircraft furtivity lead us to use global illumination method in high spectral dimension. In these circumstances, these methods raise serious issues in term of memory consumption and of computing time. Our research project focuses on these problematics.

In the first instance, we have focused on the Path Tracing method and its GPU parallelization for the spectral image rendering. We have investigated at first the issues of this kind of rendering on the GPU. Then we have proposed the DPEPT ("Deferred Path Evaluation Path Tracing") and an efficient spectral parallelization pattern which allows us to reduce significantly the memory consumption and the computing time.

In the second phase, we have investigated how to reduce the spectral computational load of the simulation in taking care not to deteriorate the accuracy. In that sense, we have proposed to generalize the stochastic spectral rendering of color (XYZ) image to the stochastic spectral image rendering. This new method renders directly the channels of a sensor which allows us to reduce the memory and the computing requirements by reducing the spectral computational load of the simulation when the spectral input data of the scene are complex.

To sum up, the works of this thesis allows us to simulate accurately multi, hyper and ultra spectral images. The interactive time can be achieved in our case (low spectral image spatial dimension and medium geometric complexity scenes) in multi and hyper spectral resolution.

Keywords: Global illumination, Spectral rendering, GPU computing, Infrared image synthesis

Remerciements

Je tiens tout d'abord à remercier M. Éric COIRO, M. Sébastien THON et M. Romain RAFFIN qui ont eu la patience et la disponibilité de m'encadrer tout au long de ma thèse.

Cette thèse n'aurait d'ailleurs pas été possible sans le soutien financier de la région PACA pour les trois premières années et de l'entreprise UVR pour la dernière année. Je tiens donc à leur témoigner toute ma gratitude.

J'adresse également mes remerciements à M. Philippe PORRAL, M. Kadi BOUATOUCH et M. Stéphane MÉRILLOU pour leurs relectures, leurs conseils et leurs corrections concernant le manuscrit.

J'associe également mes remerciements à toutes les personnes auprès desquelles j'ai pu passer des moments agréables durant ces trois ans de thèse ce qui inclut : l'équipe MVA du DOTA de l'ONERA, les personnes du centre de Salon-de-Provence de l'ONERA, et l'équipe G-Mod du LIS de l'Université d'Aix-Marseille.

Enfin, mes derniers remerciements vont à mes parents, ma sœur, mon frère et mes amis qui m'ont soutenu tout au long de cette épreuve.

Table des matières

Table des figures

Liste des tableaux

Liste des acronymes

BRDF
Bidirectional Reflectance Distribution Function. 11, 24, 42–44, 57, 58, 109

BTDF
Bidirectional Transmission Distribution Function. 11, 43, 44

CIE
Commission Internationale de l'Éclairage. 11, 19, 45, 47, 103, 109

MIS
Multi Importance Sampling. 11, 23, 26, 57, 58, 62, 64, 65, 108, 111, 130

GPU
Graphic Processing Unit. 11, 15, 19, 30, 31, 59, 66–73, 77, 79, 81, 82, 84, 86, 88, 90, 93, 98, 99, 101, 109, 111, 129–131, 141, 142, 144–148

GPGPU
General-Purpose computing on Graphics Processing Units. 11, 15, 68–70, 77, 140–142, 144, 146–148

DPEPT
Deferred Path Evaluation Path Tracing. 12, 26, 27, 92, 93, 97–99, 129, 130

PT
Path Tracing. 12, 65, 93, 97

SPP
Samples Per Pixel. 12, 112, 121

SWIR
Short-Wave InfraRed. 13, 101, 102, 114, 119–121, 125

MWIR
Mid-Wave InfraRed. 13, 39, 101, 102, 118, 120, 121, 126

CPU
Central Processing Unit. 19, 20, 66, 68, 70, 77, 79, 90, 111, 142, 146–148

SIMD
Single Instruction Multiple Data. 20, 68, 69, 73, 86, 88

BSDF
Bidirectional Scattering Distribution Function. 25, 44, 63–65

LWIR
Long-Wave InfraRed. 39, 62, 102, 121

IES
Illuminating Engineering Society. 41

BSSRDF
Bidirectional Scattering-Surface Reflectance Distribution Function. 44

BTF
Bidirectional Texture Function. 44

LTE
Light Transport Equation. 52

NEE
Next Event Estimation. 61, 63, 65

VLWIR
Very Long-Wave InfraRed. 62

CFD
Computational Fluid Dynamics. 68

SISD
Single Instruction Single Data. 68

MISD
Multiple Instructions Single Data. 68

MIMD
Multiple Instructions Multiple Data. 68

SIMT
Single Instruction Multiple Threads. 68, 69

VNIR
Visible and Near InfraRed. 101, 102

MAPE
Mean Absolute Percentage Error. 118

FPGA
Field-Programmable Gate Array. 141, 146–148

DSP
Digital Signal Processor. 146

Glossaire

sRGB
Espace de couleur RGB standard pour les écrans. 11, 46, 47

NTSC
Espace de couleur RGB de la NTSC (National Television System Committee). 11, 46, 47

noyau
Un noyau (ou kernel en anglais) est une fonction de point d'entrée pour lancer des calculs sur un GPU. 12, 69–72, 77–79, 81, 84, 90, 140–142, 144–148

work-group
Groupe de tâches d'une grille de calcul d'un noyau. 12, 69, 86–89

VRAM
Video Random-Access Memory: c'est la mémoire principale (mémoire globale) d'un GPU. 12, 66, 71, 93, 95, 96, 119, 130

RGB
Espace de couleur en Rouge Vert Bleu (Red Green Blue) affichable sur un écran. 12, 19, 20, 25, 45, 46, 49, 51, 71, 93, 103, 105, 118

LFC
Lampe Fluorescente Compacte (CFL en anglais). Les spectres d'émission de ce type de lumière sont généralement compliqués car ils sont composés de raies spectrales. 12, 13, 106, 118, 120–122, 151

CRIRA
Code de calcul du Rayonnement InfraRouge des Avions: code de simulation CPU d'image spectrale de l'ONERA dans l'infrarouge et le visible. 12, 79, 104, 107

XYZ
Espace de couleur de la CIE représentant l'ensemble des couleurs visibles par l'œil humain. 24, 25, 45, 46, 49, 88, 103, 108, 109, 129

LDR
Low Dynamic Range: terme utilisé pour désigner une image RGB dont la dynamique a été compressée pour être affichée sur un écran. 25

HDR
High Dynamic Range: terme utilisé pour désigner une image RGB dont la dynamique n'a pas été compressée pour être affichée sur un écran. 25, 93

LMS
Sensibilité photopique (vision diurne) des cônes L, M et S de l'œil humain. 45

RNG
Random Number Generator: Générateur de variable aléatoire μ. 59, 60, 62, 63, 65

work-item
Tâche (ou thread en anglais) exécutée par le GPU. 69

RGBA
RGB avec un canal alpha pour représenter la transparence. 71

MATISSE
Modélisation Avancée de la Terre pour l'Imagerie et la Simulation des Scènes et de leur Environnement: code de transfert radiatif de l'ONERA qui peut générer des sphères d'éclairement dans le visible et l'infrarouge. 93, 118

plume
Jet propulsif d'un avion. 101

SSE
Streaming SIMD Extensions: jeu d'instructions SIMD pour les CPU en architecture x86. 111

Nomenclature

Ω

Domaine d'intégration directionnel (hémisphère, sphère, angle solide, ...). 11, 37, 41, 42, 52, 61

$\mathcal{A}$

Domaine d'intégration surfacique (aire projetée par un angle solide, aire de l'ensemble des surfaces d'une scène, ...). 11, 37, 52, 61

ℓ

Longueur d'un segment (entre deux sommets par exemple). 11, 37, 62

L_o

Luminance énergétique (spectrale si paramétrée par une longueur d'onde) totale. 11, 52, 53, 60, 61, 63, 65

L_e

Luminance énergétique (spectrale si paramétrée par une longueur d'onde) émise par un objet. 11, 52–54, 60–63, 65, 85

L_r

Luminance énergétique (spectrale si paramétrée par une longueur d'onde) réfléchie par un objet. 11, 52, 53, 61, 64

L

Luminance énergétique (spectrale si paramétrée par une longueur d'onde). 12, 41, 42, 62, 80, 85

$\mathcal{T}$

Accumulation des atténuations de rebonds d'un chemin pour une longueur d'onde donnée. 12, 54, 59, 60, 62, 63, 65, 80, 85

λ

Longueur d'onde d'une onde monochromatique (échantillon spectral dans le cas de nos simulations). 12, 24, 25, 33, 35, 38, 41–43, 45, 49, 52, 54, 56, 57, 60–63, 65, 80, 85, 103, 108–111, 116

C_c

Consommation mémoire des variables communes des échantillons spectraux d'un chemin. 12, 80, 81, 90, 91

C_λ

Consommation mémoire d'un échantillon spectral. 12, 80–82

N_{ϖ}
Nombre de chemins calculés en parallèle. 12, 80–82, 86, 90, 91, 93

C_r
Consommation mémoire d'un réel (4 octets en simple précision et 8 octets en double précisions). 12, 82, 83

N_w
Nombre de pixels par ligne d'une image (largeur). 12, 82, 83

N_h
Nombre de pixels par colonne d'une image (hauteur). 12, 82, 83

C_v
Consommation mémoire d'un sommet du chemin. 12, 90, 91

μ
Variable aléatoire échantillonnée de manière uniforme ente 0 et 1. 20, 113

$\overrightarrow{\omega_i}$
Vecteur unitaire incident en un point d'une surface. Par convention le vecteur pointe vers l'extérieur de la surface. 22, 24, 25, 42–44, 52, 60–63, 65, 85

$\overrightarrow{\omega_o}$
Vecteur unitaire sortant en un point d'une surface. Par convention le vecteur pointe vers l'extérieur de la surface. 22, 24, 25, 42–44, 61, 62, 85

$\overrightarrow{n}$
Vecteur unitaire de la normale en un point d'une surface. 22, 52

i
Indice d'une colonne d'une image. 23, 25, 26, 28, 45, 46, 49, 54, 56, 57, 60, 63, 65, 103, 108–111, 113, 116

j
Indice d'une ligne d'une image. 23, 25, 26, 28, 45, 46, 49, 54, 56, 57, 60, 63, 65, 103, 108–111, 113, 116

$\overrightarrow{w}$
Direction. 24, 25, 41, 52

k
Indice d'un élément spectral (longueur d'onde ou canal) d'une image spectrale. 25, 27, 28, 45, 49, 54, 56, 57, 103, 109, 111, 113, 116

v
Structure de donnée d'un sommet d'un chemin de lumière. Un sommet n'est pas forcément qu'une position, il peut également contenir d'autre informations utiles ($\overrightarrow{\omega_i}$, $\overrightarrow{\omega_o}$, $\overrightarrow{n}$, ...) aux algorithmes d'illumination globale.. 25, 52, 54, 60–63, 65, 85

x

Position dans l'espace monde de la scène. 25, 26, 60, 62, 63, 65, 85

$\overline{\Omega}$

Ensemble des chemins passant par un pixel d'une image. 26, 54, 56, 103, 108–110, 116

t

Indice d'une technique d'échantillonnage dans le cadre du MIS. 26, 57, 58

f_{ij}

Fonction de mesure de Veach pour un pixel d'une image de coordonnées (i, j) et un chemin donné. La version spectrale prend également une longueur d'onde associée à ce chemin.. 27, 54, 56, 57, 108

c

Célérité d'une onde monochromatique Dans notre cas (onde électromagnétique), c'est la vitesse de la lumière dans le vide ($c \approx 3 \times 10^8\,\mathrm{m \cdot s^{-1}}$). 33, 38, 39

ν

Fréquence d'une onde monochromatique. 33

$\overline{\nu}$

Nombre d'onde d'une onde monochromatique. 35

T

Température en K. 38, 39, 41

L_P

Luminance énergétique (spectrale si paramétrée par une longueur d'onde) émise par un corps noir pour une température donnée. 38, 41

h

Constante de Planck: $6{,}626\,070\,15 \times 10^{-26}\,\mathrm{J \cdot s}$. 38, 39

k_B

Constante de Boltzmann: $1{,}380\,649 \times 10^{-17}\,\mathrm{J \cdot K^{-1}}$. 38, 39

M_P

Émittance énergétique émise par un corps noir pour une température donnée. 38, 41

σ

Constante de Stefan-Boltzmann: $5{,}670\,374 \times 10^{-2}\,\mathrm{W \cdot m^{-2} \cdot K^{-1}}$. 38

λ_w

Loi du déplacement de Wien qui permet de calculer le maximum de la loi de Planck pour une température donnée. 39

ε_h

Émissivité hémisphérique totale pour une température donnée. 41

Λ

Domaine d'intégration spectral. 41

$\theta_{\overrightarrow{w}}$

Angle d'élévation d'une direction $\overrightarrow{w}$ donnée. 41, 42, 52, 61

M

Émittance énergétique (spectrale si paramétrée par une longueur d'onde). Le symbole peut être paramétré par une température.. 41

ε_d

Émissivité directionnelle (spectrale si paramétrée par une longueur d'onde sinon totale) pour une température et une direction données. 41

τ_r

BRDF pour une longueur d'onde λ et les directions $\overrightarrow{\omega_o}$ et $\overrightarrow{\omega_i}$. 42

E

Éclairement énergétique (spectral si paramétré par une longueur d'onde) dans une direction donnée. 42

I^{XYZ}

Image en couleur XYZ. 45, 46, 108

K_m

Constante de normalisation ($K_m = 683{,}002\,\mathrm{lm} \cdot \mathrm{W}^{-1}$) qui représente l'efficacité lumineuse spectrale photopique maximale, atteinte par une longueur d'onde de 555 nm dans l'air. 45

N_Λ

Dimension spectrale d'une image spectrale. 45, 49, 82, 98, 103, 110, 111, 113, 114

$\overline{x}$

Valeur de la courbe X de l'espace de couleur XYZ pour une longueur d'onde donnée. 45, 108

$\overline{y}$

Valeur de la courbe Y de l'espace de couleur XYZ pour une longueur d'onde donnée. 45, 108

$\overline{z}$

Valeur de la courbe Z de l'espace de couleur XYZ pour une longueur d'onde donnée. 45, 108

I^λ

Image spectrale composée de longueurs d'onde. 45, 49, 54, 56, 57, 85, 103

$\overline{m}$

Courbe de réponse photopique (vision diurne) des cônes M de l'œil humain pour une longueur d'onde donnée. 45

$\overline{s}$

Courbe de réponse photopique (vision diurne) des cônes S de l'œil humain pour une longueur d'onde donnée. 45

$\overline{l}$

Courbe de réponse photopique (vision diurne) des cônes L de l' œil humain pour une longueur d'onde donnée. 45

I^{RGB}

Image en couleur RGB LDR ou HDR. 45, 46, 103

$M_{XYZ \to RGB}$

Matrice de passage pour transformer des couleurs XYZ en couleur RGB. 45

γ

Fonction de correction pour compenser la non-linéarité de la réponse des capteurs et des écrans. 46

I^{L}

Image intégrée en luminance énergétique. 49

$\overline{W_{ij}^{k}}$

Réponse complète moyenne d'un capteur pour un élément de coordonnées (i, j, k) d'une image spectrale. 49, 116

n

Indice d'un sommet d'un chemin de lumière. 52, 54, 61, 63, 65

δ

Fonction de calcul du point d'intersection le plus proche entre un rayon (composé d'une origine, d'une direction et optionnellement d'une longueur) donné dans une scène. Cette fonction renvoie un sommet v contenant une position x si le rayon touche un objet. Sinon, elle renvoie un sommet contenant une position nulle $x_{\varnothing}$. 52, 60–63, 65

τ

BSDF pour au moins une longueur d'onde λ et des directions $\overrightarrow{\omega_o}$ et $\overrightarrow{\omega_i}$. En fonction de la complexité du matériau, cette fonction peut nécessiter d'autre arguments en entrée. 52, 54, 60–63, 65, 85

$\widehat{\omega}$

Vecteur normalisé d'un vecteur $\overrightarrow{w}$ donné. 52, 54, 61

G

Terme géométrique. 52, 54, 61

$\mathcal{V}$

Fonction de test de visibilité entre deux sommets donnés. Renvoie 1 si le test réussit ou 0 autrement. 52

$\overline{\omega}$

Échantillon de l'espace $\overline{\Omega}$ (chemin). 54, 56–58, 85, 86, 103, 108–111, 113, 116

N_v

Nombre de sommets d'un chemin lumière. 54, 59, 90, 93

W_{ij}

Réponse partielle d'un capteur (sans sa courbe de réponse spectrale) pour un pixel de coordonnées (i, j). 54, 103

ρ

Densité de probabilité d'un échantillon. 56–58, 60, 62, 63, 65, 85, 110, 111, 113

N_s

Nombre d'échantillons calculés dans le cas d'une simulation avec la méthode de Monte-Carlo. 56, 57, 110, 111, 113

s

Indice d'un échantillon dans le cas d'une simulation avec la méthode de Monte-Carlo. 56–58, 110, 111, 113

Δ

Intervalle d'un espace d'intégration. 56

N_t

Nombre de techniques d'échantillonnage dans le cadre du MIS. 57, 58

N_{ts}

Nombre d'échantillons pour une technique d'échantillonnage t dans le cadre du MIS. 57, 58

w

Coefficient pour le MIS. Dans le cas des sommets du DPEPT, les coefficients contiennent également la densité de probabilité et le cosinus. 57, 58, 65, 85

ϱ_{rr}

Fonction de la roulette russe qui choisit stochastiquement si un chemin doit survivre ou pas. 59, 60, 63, 65

ρ_s

Probabilité de survie d'un chemin dans le cadre de la méthode de la roulette russe. 59

ϱ_c

Fonction d'échantillonnage de l'espace image d'un capteur. 60, 63, 65

ϑ

Fonction qui évalue la carte d'environnement pour un sommet contenant une position « vide » $x_\varnothing$ et une longueur d'onde donnés. 60, 63, 65, 85

ϱ_τ
Fonction d'échantillonnage d'un matériau. 60, 63, 65

L_d
Luminance énergétique (spectrale si paramétrée par une longueur d'onde) de la lumière directe. 61, 62, 64, 65, 85

L_i
Luminance énergétique (spectrale si paramétrée par une longueur d'onde) de la lumière indirecte. 61, 64, 65, 85

ϱ_{L_d}
Fonction qui échantillonne la lumière directe. 62, 63, 65

ϱ_l
Fonction d'échantillonnage des sources de lumières dans une scène. 62

C_A
Consommation mémoire d'un algorithme d'illumination globale. 80

$C_{\overline{\omega}}$
Consommation mémoire d'un chemin. 80, 82, 90

N_λ
Nombre de longueurs d'onde calculées par chemin. 80, 82, 93, 98

C_I
Consommation mémoire d'une image spectrale (composée de canaux ou de longueurs d'onde) que l'on veux simuler. 82

β
Groupe de longueurs d'onde. 85, 86

$\mathbf{V}$
Buffer des sommets des chemins sauvegardés. 85

$\mathcal{F}$
Fonction qui évalue la luminance spectrale énergétique d'un groupe de longueurs d'onde pour un chemin préconstruit dans la première étape du DPEPT. 85, 86

N_β
Nombre de groupes de longueurs d'onde à évaluer par chemin. 86

$\overline{f}$
f_{ij} sans la courbe de réponse du capteur pour une longueur d'onde et un chemin donnés. 103, 109–111, 113, 116

W^k
Réponse partielle d'un capteur (sans sa courbe de réponse spatiale) dans le canal k d'une image spectrale pour une longueur d'onde donnée. 103

I^c

Image spectrale composée de canaux. 109–111, 113, 116

${\lambda_{min}}^k$

Borne inférieure d'un canal k donnée. 109, 110, 113, 116

${\lambda_{max}}^k$

Borne supérieure d'un canal k donnée. 109, 110, 113, 116

W_{ij}^k

Réponse d'un capteur pour un élément de coordonnées (i, j, k) d'une image spectrale, une longueur d'onde et un chemin donnés. 109–111, 113, 116

φ^k

Fonction réciproque de la fonction de répartition associée à la densité de probabilité de l'échantillonnage des longueurs d'onde qui calcule à la volée l'échantillon spectral associé à une variable aléatoire uniforme et un canal k donnés. 113

Introduction

Contexte

La détection d'aéronefs dans les bandes infrarouge et visible est un domaine actif de recherche dans la communauté aérospatiale. L'un des principaux enjeux de ce domaine est de dimensionner les capteurs de détection. Dans cette optique, il faut pouvoir simuler ces capteurs via des modèles. Et pour cela, il faut également disposer d'un nombre conséquent d'images spectrales d'aéronefs dans divers scénarios. L'obtention de ces images via des campagnes aériennes de mesure est toutefois onéreuse, chronophage et difficile. Une simulation rapide et robuste de ces données s'impose donc.

Problématiques

Dans l'infrarouge, les méthodes d'illumination locales ([Whi79] par exemple) sont utilisées massivement pour simuler le transport de la lumière dans les cas où les contributeurs majoritaires à la signature sont visibles de manière directe par le capteur (tuyères à haute température, rayonnement du jet propulsif). Ces méthodes sont cependant inadéquates pour les nouvelles générations d'avions furtifs car des contributeurs importants au rayonnement sont en partie masqués par des entrées d'air ou des tuyères incurvées. Il faut alors modéliser les inter-réflexions de manière robuste à l'intérieur de ces conduits via des méthodes d'illumination globale. L'ONERA a commencé à explorer ce type de méthode [Coi12]. Ces méthodes ont cependant l'inconvénient d'augmenter significativement les temps de calculs.

De plus, les gaz de l'atmosphère (figure 1.1) et de la plume (jet propulsif) présentent des spectres d'émission et d'absorption complexes ce qui nécessitent une simulation spectrale extrêmement fine. Dans le meilleur des cas, la dimension spectrale de la simulation doit être au moins aussi élevée que la résolution spectrale des capteurs que l'on cherche à simuler (résolution multi et hyper spectrale). Afin de prendre en compte cette complexité spectrale, des méthodes d'illumination globale spectrales déterministes sont utilisées car elles permettent de conserver la sortie spectrale à la fin de la simulation. Ces méthodes sont cependant peu flexibles et gourmandes en mémoire et en temps de calculs.

Pour résumer, il existe des méthodes de rendu d'images spectrales en illumination globale dans l'infrarouge et dans le visible. Ces méthodes posent cependant des problèmes de consommation mémoire et de temps de calcul et plus spécifiquement dans notre cas ou les données spectrales d'entrée sont complexes et que l'on cherche de la précision.

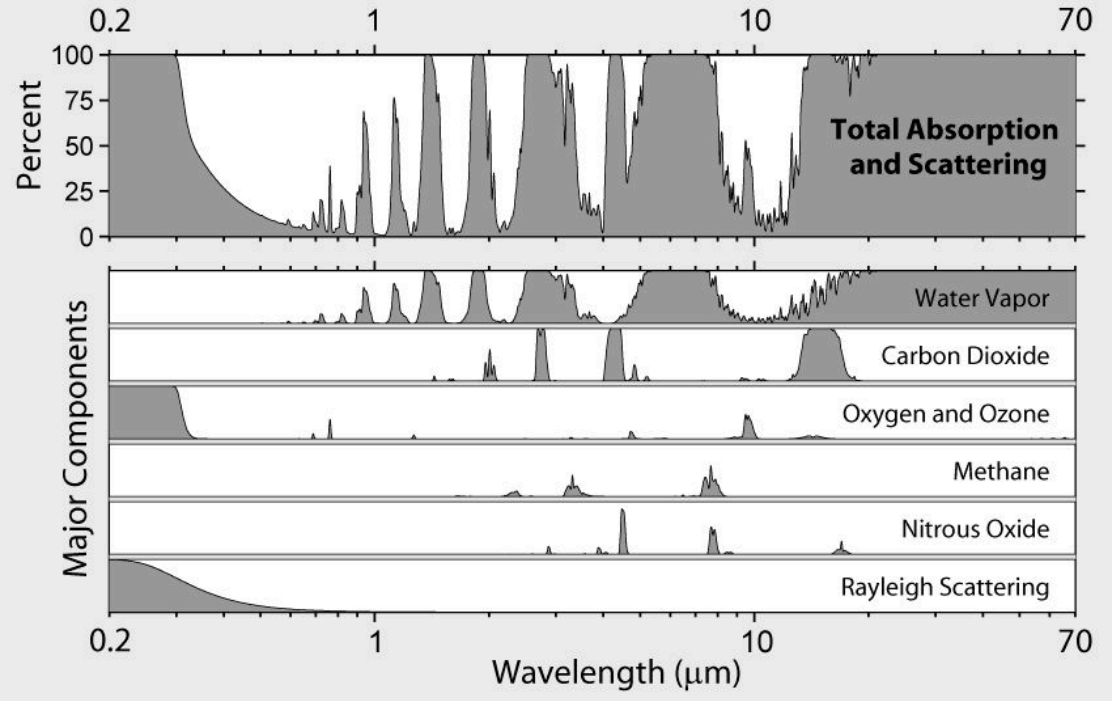


FIGURE 1.1. – Spectres d'absorption des gaz de l'atmosphère [Roh07].

Projet de recherche

Notre projet de recherche s'inscrit dans le cadre de ces problématiques de temps de calcul et de consommation mémoire. Une grande attention a également été portée à ne pas dégrader la précision. À ce titre, nous nous sommes focalisés sur les méthodes de Monte-Carlo pour leurs robustesses et plus spécifiquement sur l'algorithme du Path Tracing. Le choix de cette méthode a été motivé par la simplicité géométrique des scènes visées (scène aérienne prise par un capteur au sol ou aéroporté) et par l'efficacité effective des implémentations GPU de cette méthode.

En effet, nous avons décidé de nous orienter en priorité sur la parallélisation de cet algorithme sur GPU car c'est la solution la plus évidente pour réduire les problèmes des temps de calcul. Cette solution est toutefois contrariée par la capacité et la latence de la mémoire des GPUs qui restent assez limitées pour du rendu spectral en dépit des avancées technologiques. Dans un premier temps, nous avons donc cherché à cerner ces problèmes et adapter l'algorithme du Path Tracing pour le rendu d'image à haute résolution spectrale sur GPU.

En dépit d'un gain considérable de temps de calculs et de réduction de consommation mémoire, nous avons cependant continué à creuser pour chercher à améliorer nos résultats. À cet égard, nous avons dans un deuxième temps cherché à diminuer la charge de calcul en diminuant la dimension spectrale de la simulation tout en ne délaissant pas la précision.

Organisation du manuscrit

Suite à notre introduction, le manuscrit se découpe en trois grande parties :

1. **Notions** (chapitre 2) : les prérequis sur la parallélisation GPU et l'illumination globale dans le visible et dans l'infrarouge, sont décrits dans cette partie.

2. **Parallélisation GPU du rendu d'image à haute dimension spectrale** (chapitre 3) : le chapitre démarre par un état de l'art succinct sur les implémentations GPU du Path Tracing, les problèmes du Path Tracing à haute résolution spectrale sur GPU sont exposés. Une implémentation efficace en temps de calculs et en consommation mémoire est ensuite proposée et évaluée.

3. **Rendu stochastique d'image spectrale** (chapitre 4) : après un rappel exhaustif des problématiques du rendu d'image à haute résolution spectrale, un état de l'art sur le rendu spectral et d'image spectrale est réalisé dans un second temps. Une nouvelle méthode de rendu d'image spectrale stochastique qui réduit la dimension spectrale de la simulation est ensuite proposée et des cas d'utilisations sont discutés. Le chapitre se termine sur un benchmark de notre méthode.

2. Notions

2.1. L'optique

L'optique est le domaine qui étudie le comportement de la lumière.

2.1.1. La lumière

Par abus de langage, la lumière est associée au rayonnement électromagnétique visible par l' œil humain. En réalité, elle n'est pas restreinte à ce domaine spectral (figure 2.1). D'ailleurs, on utilise les termes lumière ultraviolet, lumière visible, lumière infrarouge pour éviter toute ambiguïté. Toutefois, on exclut généralement l'usage du terme « lumière » pour les rayons ionisants (X et Gamma), les micro-ondes et les ondes radio.

2.1.2. Les ondes électromagnétiques

Une onde électromagnétique est un modèle de représentation du rayonnement électromagnétique. Elle représente la variation macroscopique du champ électrique et du champ magnétique. Ces champs sont liés par les équations de Maxwell (père de l'électromagnétisme).

2.1.2.1. La longueur d'onde

Pour caractériser et différencier les ondes on utilise généralement la longueur d'onde qui est la période spatiale de l'oscillation d'une onde monochromatique dans un milieu homogène.

$$\lambda = \frac{c}{\nu} \tag{2.1}$$

En optique, la longueur d'onde d'une onde électromagnétique fait toujours référence à la longueur d'onde dont le milieu de propagation est le vide.

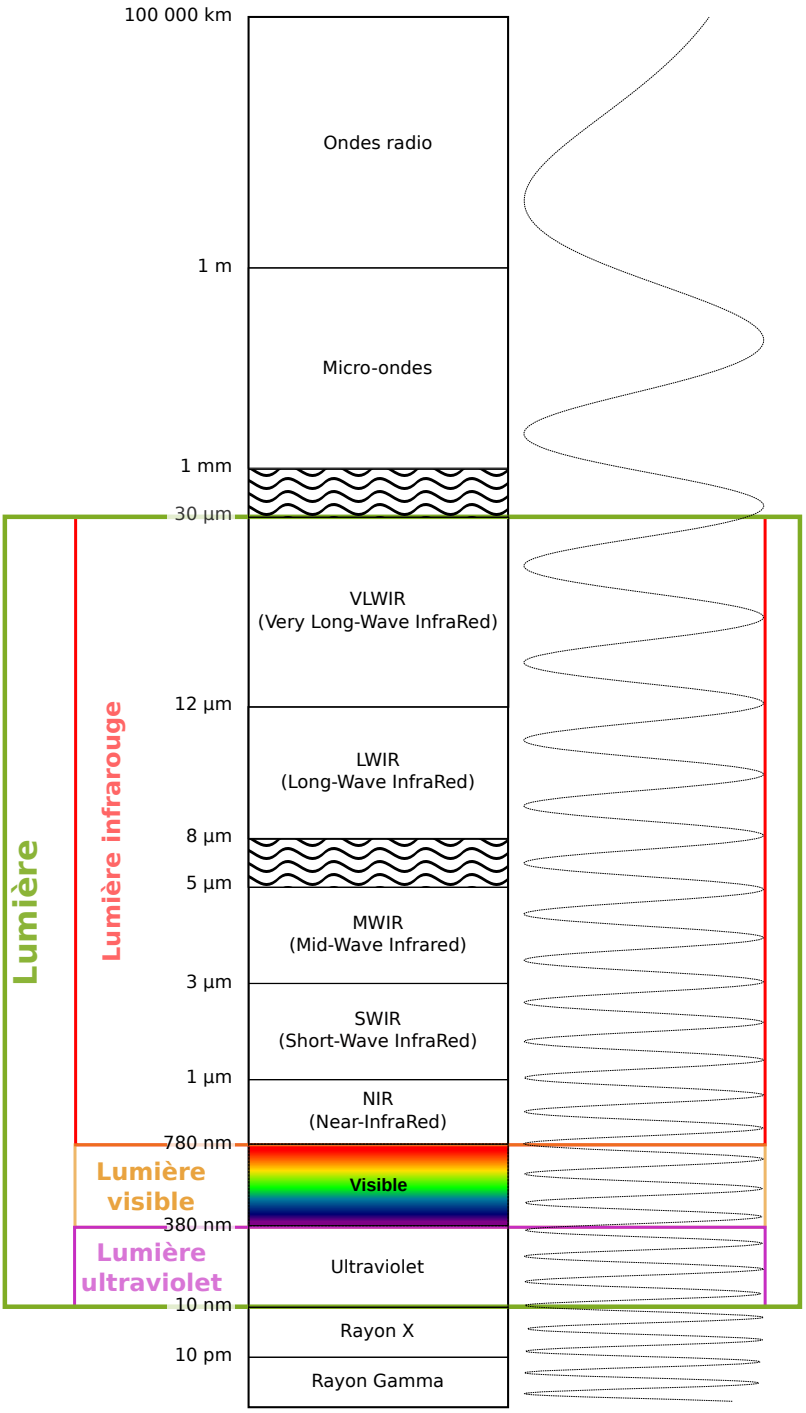


FIGURE 2.1. – Le spectre électromagnétique.

2.1.2.2. Le nombre d'onde

En spectroscopie, on préfère utiliser le nombre d'onde. Le nombre d'onde est la fréquence spatiale d'une l'onde électromagnétique (le nombre d'oscillation sur une unité de longueur) :

$$\overline{\nu} = \frac{1}{\lambda} \tag{2.2}$$

2.1.3. Les différents modèles de l'optique

Il existe différents modèles pour décrire le comportement de la lumière. Dans notre cas, les simulations se situent dans le modèle de l'optique géométrique.

2.1.3.1. L'optique géométrique

Introduite par le mathématicien et physicien arabe Alhazen sur la base d'observations simples, elle repose sur le modèle du rayon lumineux.

Ce modèle est une notion d'optique et un outil mathématique qui décrit le trajet de la lumière de manière simplifiée. Un rayon lumineux n'a pas d'existence physique réelle et représente le cas idéal où il serait possible de sélectionner un faisceau parallèle infiniment fin de lumière. L'optique géométrique se base sur les principes suivants :

— Le principe de Fermat : dans un milieu homogène et isotrope, les rayons lumineux se propagent de manière rectiligne en choisissant le trajet le plus optimal (temporellement parlant).

— Le principe du retour inverse : le trajet d'un rayon lumineux est identique dans les deux sens de propagation.

— Le principe d'indépendance des rayons lumineux : il n'y a pas d'interaction entre deux rayons lumineux, un rayon ne peut donc pas interférer ou dévier un autre rayon.

En pratique, cela se matérialise sous la forme de calculs d'intersection entre des droites (rayon lumineux) et des représentations géométriques (maillage de triangles, ...).

Ce modèle est valable uniquement lorsque le rayon lumineux se propage dans des milieux où les obstacles et composants optiques ont des dimensions très supérieures à la longueur d'onde.

De base, il ne prend pas en compte l'aspect ondulatoire de la lumière, il n'est donc pas apte à représenter des phénomènes liés à la polarisation, les interférences et la

diffraction. Des extensions sont cependant possibles pour prendre en compte certains aspects ondulatoires de la lumière comme la polarisation. Pour les interférences et la diffraction, on peut les simuler via l'optique ondulatoire pour construire des modèles macroscopiques qui eux sont utilisables en optique géométrique.

2.1.3.2. L'optique ondulatoire

L'optique ondulatoire (ou l'optique physique) utilise la notion d'onde électromagnétique pour décrire le comportement de la lumière. Elle permet d'expliquer les phénomènes affectant les ondes, comme les interférences et la diffraction.

2.1.3.3. L'optique quantique

L'optique quantique propose de considérer la lumière comme constituée de photons (particule de lumière) qui se comportent comme des corpuscules dans leur interaction avec la matière et comme des ondes pour leur propagation.

Le mouvement des photons se base sur la mécanique quantique qui est décrit à l'aide de probabilités de présence en un point donné.

Le point fort de ce modèle est de pouvoir décrire les phénomènes à l'échelle submicroscopique. L'optique ondulatoire fournit toutefois une description plus pertinente pour les grandes longueurs d'onde (ondes radio, ...). Les deux modèles d'optique se recouvrent quand le flux d'énergie est grand devant l'énergie des photons car on peut considérer que l'on a un flux quasi-continu de photons.

2.1.4. La radiométrie

Afin de simuler des images spectrales, on manipule des grandeurs radiométriques (tableau 2.1). La radiométrie est le domaine qui étudie la quantité d'énergie transportée et propagée par des ondes ou des particules.

Symbole	Nom	Unité
Q	Énergie électromagnétique	J
Φ	Flux énergétique	W
I	Intensité énergétique	$\mathrm{W} \cdot \mathrm{sr}^{-1}$
L	Luminance énergétique	$\mathrm{W} \cdot \mathrm{sr}^{-1} \cdot \mathrm{m}^{-2}$
E	Éclairement énergétique	$\mathrm{W} \cdot \mathrm{m}^{-2}$
M	Exitance énergétique	$\mathrm{W} \cdot \mathrm{m}^{-2}$
H	Exposition énergétique	$\mathrm{J} \cdot \mathrm{m}^{-2}$

TABLE 2.1. – Grandeurs radiométriques.

Dans le cas ou les grandeurs radiométriques sont données par longueur d'onde ou nombre d'onde, on ajoute l'adjectif « spectral(e) » à leur nom.

2.1.4.1. L'angle solide

L'angle solide est une notion capitale pour décrire certaines grandeurs radiométriques. Son unité SI est le stéradian (sr).

Par analogie avec un angle plan (défini dans un plan), l'angle solide est un angle défini dans un volume. On peut le calculer via cette formule :

$$\Omega = \frac{\mathcal{A}}{\ell^2} \tag{2.3}$$

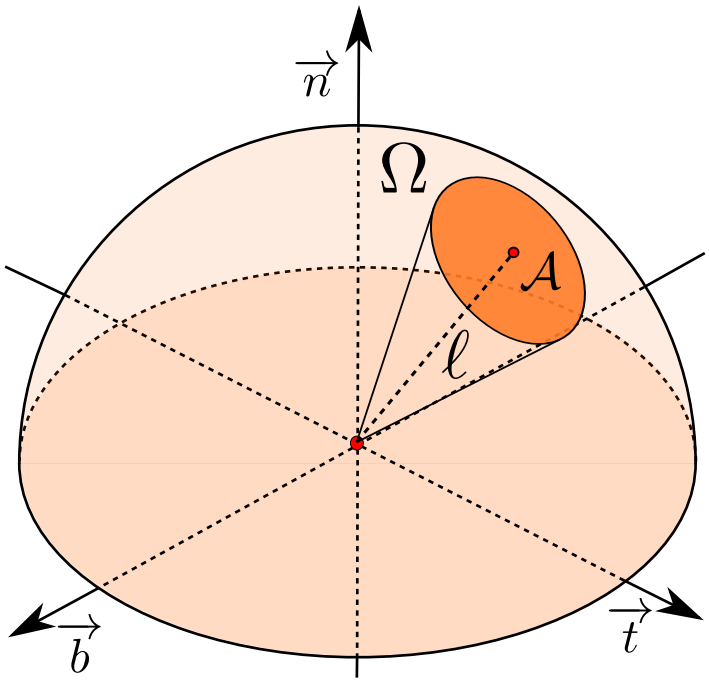


FIGURE 2.2. – Ω est l'angle solide que l'on cherche à calculer. $\mathcal{A}$ est l'aire d'une portion d'une sphère de rayon ℓ qu'intersecte le cône sur la figure.

2.1.5. La photométrie

La photométrie est l'équivalent de la radiométrie pour la vision humaine.

Pour différencier les grandeurs radiométriques et photométriques (tableau 2.1 et tableau 2.2), on ajoute généralement au nom des grandeurs l'adjectif « énergétique » pour la radiométrie et l'adjectif « lumineux(se) » pour la photométrie. Un indice « v » est également ajouté au symbole des grandeurs photométriques.

Symbole	Nom	Unité
Q_v	Quantité de lumière	$\mathrm{lm} \cdot \mathrm{s}^{-1}$
Φ_v	Flux lumineux	lm
I_v	Intensité lumineuse	cd
L_v	Luminance lumineuse	$\mathrm{cd} \cdot \mathrm{m}^{-2}$
E_v	Éclairement lumineux	lx
M_v	Exitance lumineuse	$\mathrm{lm} \cdot \mathrm{m}^{-2}$
H_v	Exposition	$\mathrm{lx} \cdot \mathrm{s}^{-1}$

TABLE 2.2. – Grandeurs photométriques.

Les grandeurs photométriques peuvent également être définies par longueur d'onde ou par nombre d'onde. Dans ce cas, on ajoute également l'adjectif « spectral(e) » au nom.

2.1.6. Modèles d'émission de la lumière

Des modèles d'émission de lumière sont indispensables pour nos simulations. C'est davantage nécessaire dans l'infrarouge où tout objet (à température ambiante ou à quelques centaines de °C) émet un rayonnement thermique perceptible par un capteur.

2.1.6.1. Corps noir

Le plus simple des modèles est le corps noir dont le nom est mentionné pour la première fois par Gustav Kirchhoff en 1862. Un corps noir est un objet idéal qui absorbe totalement la lumière qu'il reçoit. Cette absorption provoque une excitation thermique qui se traduit ensuite par une émission isotrope d'un rayonnement thermique appelé rayonnement du corps noir.

La loi de Planck (équation (2.4) et figure 2.3) établit une relation entre la luminance énergétique spectrale émise par un objet et sa température T à l'équilibre énergétique :

$$L_P(\lambda,\, T) = \frac{2hc^2}{\lambda^5} \times \frac{1}{e^{\frac{hc}{\lambda k_B T}} - 1} \tag{2.4}$$

La loi de Stefan-Boltzmann donne l'émittance totale d'un corps noir pour une température T donnée :

$$M_P(T) = \sigma T^4 \tag{2.5}$$

La loi du déplacement de Wien permet de déterminer le maximum de la loi de Planck (la longueur d'onde où la luminance spectrale est maximale) :

$$\lambda_w(T) = \frac{hc}{4.96511423174 \times k_B\, T} = \frac{2.89777291 \times 10^{-3}}{T} \tag{2.6}$$

Lorsque la température augmente (figure 2.3), ce maximum ce déplace vers les courtes longueurs d'onde (Visible, ...). D'ailleurs, le qualificatif de « noir » vient du fait que la lumière est entièrement absorbée par l'objet et qu'il faut atteindre des températures assez hautes ($\geq$ 700°C) dans le visible pour percevoir son rayonnement à l'œil nu.

Dans l'infrarouge, le rayonnement du corps noir est maximale dans le LWIR à température ambiante. Dans le MWIR, il faut atteindre quelques centaines de °C pour atteindre ce rayonnement maximale.

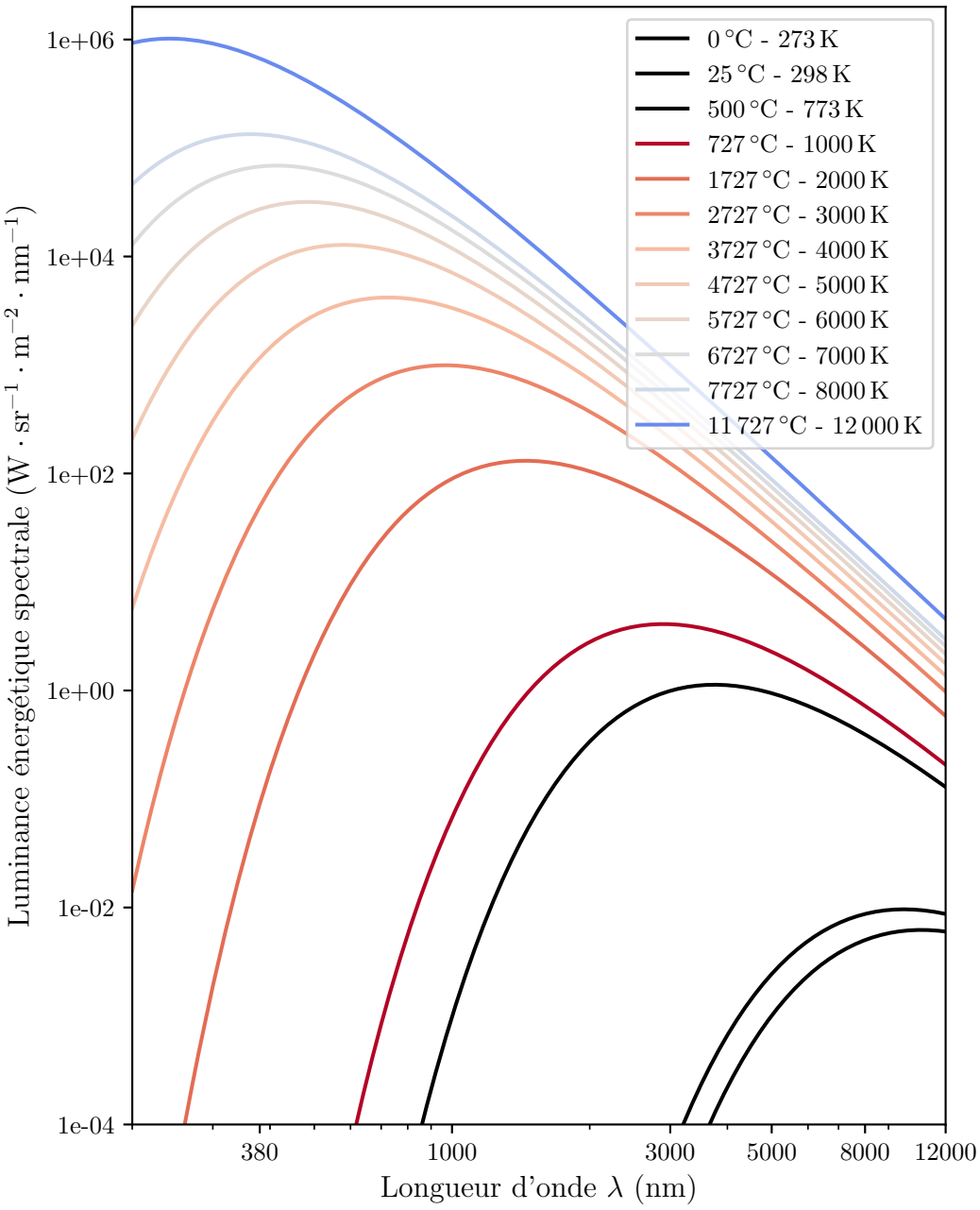


FIGURE 2.3. – Rayonnement de corps noir à différentes températures. Les abscisses et les ordonnées sont en échelle logarithmique pour faciliter la lecture. La couleur de la courbe est la couleur approximée du rayonnement dans le visible pour l'œil humain.

2.1.6.2. Corps gris

Un corps gris est un corps qui réfléchit une partie de la lumière reçue et donc qui n'absorbe pas toutes cette lumière incidente et cela indépendamment de la longueur d'onde.

Pour modéliser cela, on utilise un coefficient d'émissivité hémisphérique totale ε_h qui vient moduler la loi de Planck en fonction du matériau. Ce coefficient est calculé à partir du ratio entre la luminance énergétique spectrale $L(\lambda,\ \overrightarrow{w},\ T)$ du corps gris observé et la luminance énergétique spectrale $L_P(\lambda,\ T)$ émise par un corps noir placé dans les mêmes conditions de température T :

$$\varepsilon_h = \frac{\int_\Omega \int_\Lambda L(\lambda,\ \overrightarrow{w},\ T) |\cos\theta_{\overrightarrow{w}}|\ \mathrm{d}\lambda\ \mathrm{d}\overrightarrow{w}}{\int_\Omega \int_\Lambda L_P(\lambda,\ T) |\cos\theta_{\overrightarrow{w}}|\ \mathrm{d}\lambda\ \mathrm{d}\overrightarrow{w}} = \frac{M(T)}{M_P(T)} \tag{2.7}$$

2.1.6.3. Source lumineuse non idéale

Une source de lumière peut également présenter des variations directionnelles (matériau spéculaire, métal, ...) et spectrales qui ne peuvent pas être modélisées par un modèle de corps noir ou de corps gris.

Afin de représenter fidèlement l'émission de ce type de source de lumière, on calcule des coefficients d'émissivité directionnelle monochromatique. Comme pour l'émissivité hémisphérique totale, on va chercher à calculer le ratio entre la luminance énergétique spectrale $L(\lambda,\ \overrightarrow{w},\ T)$ émise par la source de lumière observée et la luminance énergétique spectrale $L_P(\lambda,\ T)$ émise par un corps noir placé dans les mêmes conditions de température T :

$$\varepsilon_d(\lambda,\ \overrightarrow{w},\ T) = \frac{L(\lambda,\ \overrightarrow{w},\ T)}{L_P(\lambda,\ T)} \tag{2.8}$$

Si le spectre d'émission observé n'est pas complexe on peut approximer ces coefficients d'émissivité directionnelle monochromatique en coefficients d'émissivité directionnelle totale :

$$\varepsilon_d(\overrightarrow{w},\ T) = \frac{\int_\Lambda L(\lambda,\ \overrightarrow{w},\ T)\ \mathrm{d}\lambda}{\int_\Lambda L_P(\lambda,\ T)\ \mathrm{d}\lambda} = \frac{L(\overrightarrow{w},\ T)}{L_P(T)} \tag{2.9}$$

Dans le visible, il est courant d'utiliser des profils IES pour modéliser fidèlement les variations directionnelles d'une source lumineuse. Ces diagrammes représentent la distribution d'intensité lumineuse dans l'hémisphère d'émission de celle-ci.

2.1.7. Modèles de diffusion de la lumière

Dans nos algorithmes, nous sommes amenés à calculer le trajet de chemin de la lumière. Afin de faire rebondir ces chemins sur des surfaces de la scène, nous avons besoin de modèles de matériaux qui décrivent comment ceux-ci diffusent la lumière. Dans cette sous section, nous allons introduire brièvement ces modèles de matériaux.

2.1.7.1. BRDF

Une Bidirectional Reflectance Distribution Function (BRDF) (voir l'illustration via la figure 2.4) est une famille de modèles qui représente la réflexion d'un matériau pour une longueur d'onde λ, une direction entrante $\overrightarrow{\omega_i}$ et une direction sortante $\overrightarrow{\omega_o}$ données. Son unité est le stéradian inverse (sr^{-1}). Elle est définie comme ceci :

$$\tau_r(\lambda,\ \overrightarrow{\omega_i},\ \overrightarrow{\omega_o}) = \frac{dL(\lambda,\ \overrightarrow{\omega_o})}{dE(\lambda,\ \overrightarrow{\omega_i})} = \frac{dL(\lambda,\ \overrightarrow{\omega_o})}{L(\lambda,\ \overrightarrow{\omega_i})|\cos\theta_{\overrightarrow{\omega_i}}|\ d\overrightarrow{\omega_i}} \tag{2.10}$$

Elle respecte les propriétés suivantes :

— Une BRDF est toujours positive ou nulle :

$$\tau_r \geq 0 \tag{2.11}$$

— La réciprocité : comme pour l'optique géométrique qui utilise ces modèles, on a :

$$\tau_r(\lambda,\ \overrightarrow{\omega_i},\ \overrightarrow{\omega_o}) = \tau_r(\lambda,\ \overrightarrow{\omega_o},\ \overrightarrow{\omega_i}) \tag{2.12}$$

— La conservation d'énergie : une BRDF ne peut pas renvoyer plus de lumière qu'elle n'en reçoit :

$$\int_\Omega \tau_r(\lambda,\ \overrightarrow{\omega_i},\ \overrightarrow{\omega_o})\ |\cos\theta_{\overrightarrow{\omega_i}}|\ d\overrightarrow{\omega_i} \leq 1 \tag{2.13}$$

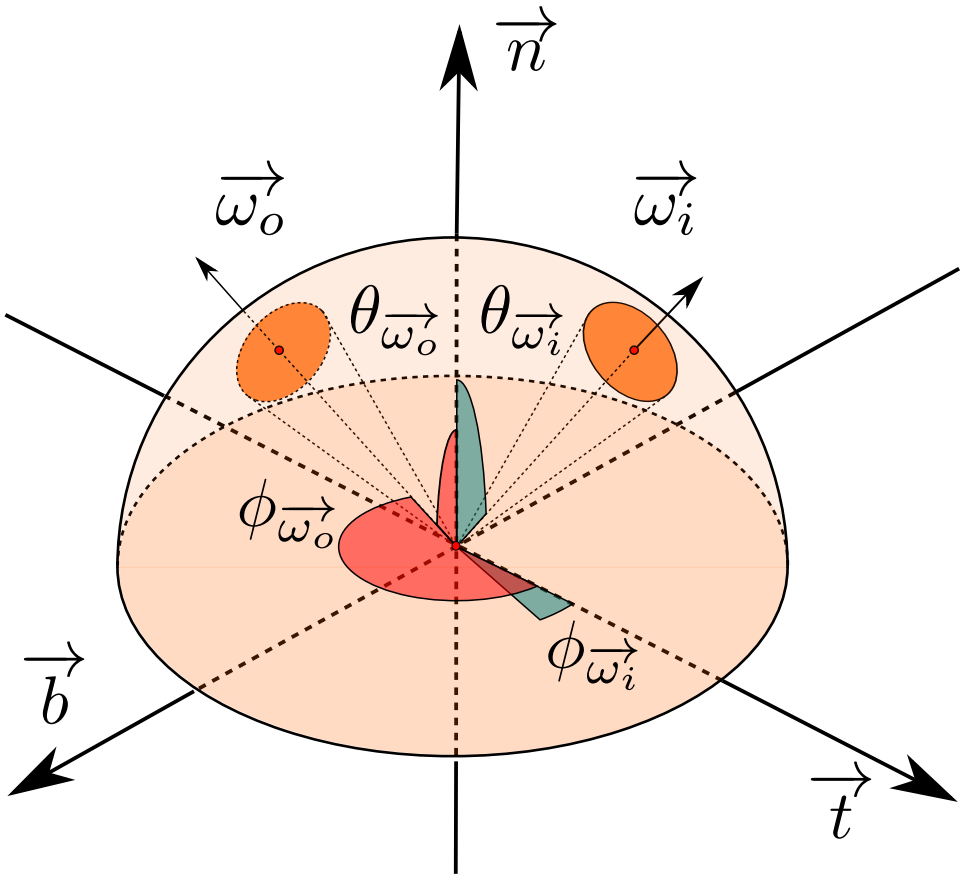


FIGURE 2.4. – Illustration des directions entrante et sortante d'une BRDF.

2.1.7.2. BTDF

Une Bidirectional Transmission Distribution Function (BTDF) (voir l'illustration de la figure 2.5) est exactement comme une BRDF, à l'exception près qu'elle représente la transmission d'un matériau pour une longueur d'onde λ, une direction entrante $\overrightarrow{\omega_i}$ et une direction sortante $\overrightarrow{\omega_o}$ donnée. Les directions $\overrightarrow{\omega_i}$ et $\overrightarrow{\omega_o}$ ne sont donc pas dans le même hémisphère (figure 2.5).

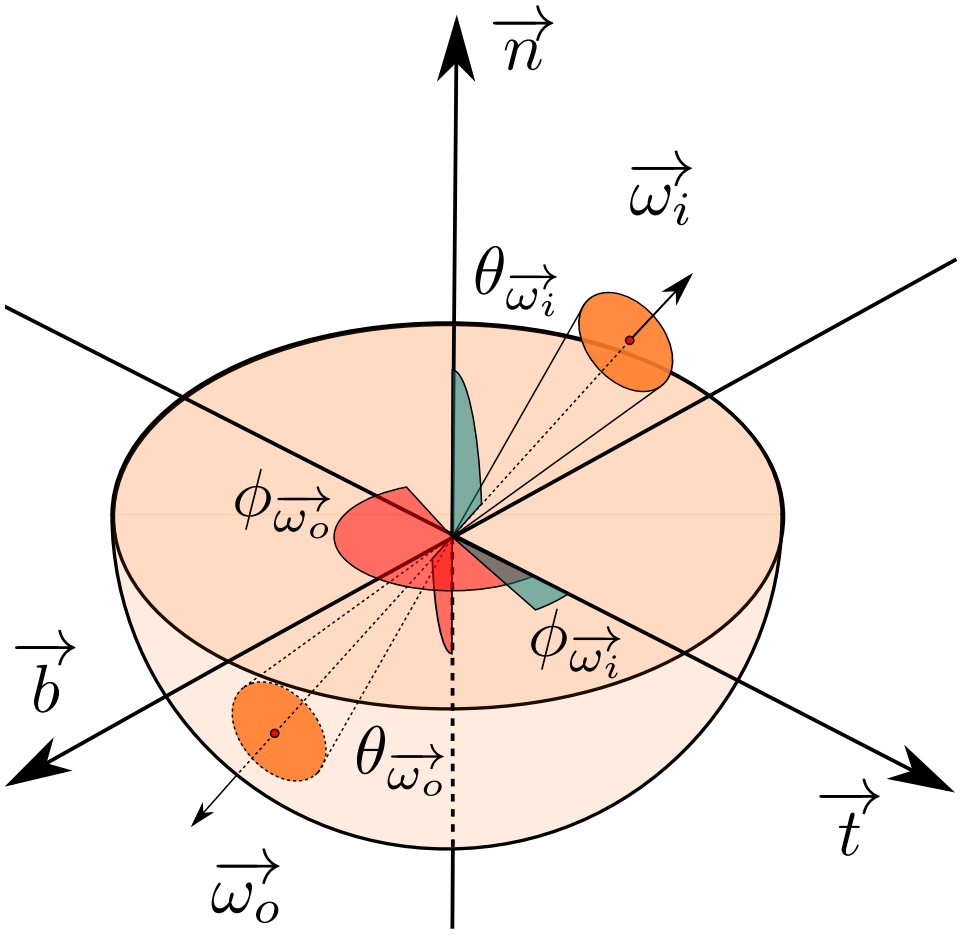


FIGURE 2.5. – Illustration des directions entrante et sortante d'une BTDF.

2.1.7.3. BSDF

Une Bidirectional Scattering Distribution Function (BSDF) est un terme générique pour désigner un modèle de matériau qui englobe une BRDF ou / et une BTDF. Parfois ce terme peut inclure d'autres modèles de matériaux plus compliqués comme par exemple :

— Une Bidirectional Scattering-Surface Reflectance Distribution Function (BSSRDF) : c'est une BRDF ou les points d'entrée de $\overrightarrow{\omega_i}$ et de sortie de $\overrightarrow{\omega_o}$ ne sont plus confondus.

— Une Bidirectional Texture Function (BTF) : une BRDF varie généralement spectralement et directionnellement (en fonction des directions $\overrightarrow{\omega_i}$, $\overrightarrow{\omega_o}$). Outre ces paramètres, une BTF est une BRDF qui varie en plus en fonction de la location du point d'intersection sur la surface.

2.1.8. De la radiométrie à la couleur

Un capteur optique (œil humain inclus) est sensible à la luminance énergétique spectrale. C'est cette dernière quantité que l'on va chercher à simuler dans le cas du rendu d'une image.

Les écrans ne peuvent qu'afficher des couleurs RGB. Leur dynamique et leur gamut (diagramme de chromaticité qui représente l'ensemble de couleurs qu'ils peuvent afficher) sont également limités.

L'image spectrale n'est donc pas directement visualisable, il faut la convertir en une image couleur affichable sur un écran.

2.1.8.1. Dans le visible

Dans le spectre visible, on va généralement chercher à calculer ce que peut voir l'œil humain. Dans ce cas, il faut par ordre chronologique (voir la figure 2.7 pour l'illustration) :

1. Convertir la sortie radiométrique de la simulation en XYZ (voir les courbes de la figure 7.2 dans l'annexe) :

$$I^{XYZ}(i,\ j) = K_m \sum_{k=1}^{N_\Lambda} \begin{bmatrix} \overline{x}(\lambda^k) \\ \overline{y}(\lambda^k) \\ \overline{z}(\lambda^k) \end{bmatrix} I^{\lambda}(i,\ j,\ k)\ (\lambda^{k+1} - \lambda^k) \tag{2.14}$$

   Le tristimulus XYZ est un espace de couleur introduit par la CIE en 1931. C'est l'espace pivot de tous les espaces couleurs et permet de convertir un stimuli de luminance énergétique spectrale en une couleur.

   Une correspondance peut être faite avec les courbes de réponse photopique (vision diurne) des cônes de l'œil humain (courbes LMS dans l'annexe : figure 7.1). $\overline{y}$ et $\overline{z}$ correspondent pratiquement et respectivement aux réponses normalisées des cônes M (courbe verte $\overline{m}$) et des cônes S (courbe bleu $\overline{s}$). La courbe $\overline{x}$ quant à elle, est une combinaison linéaire des réponses normalisées des cônes M et L (courbe rouge $\overline{l}$) de telle façon que $\overline{x}$ soit strictement positif.

2. Adapter la dynamique des valeurs photométriques : à ce point, les XYZs sont encore des valeurs physiques, il faut compresser leurs dynamiques entre 0 et 1 pour la prochaine étape.

3. Passer les valeurs photométriques en espace RGB : il faut des couleurs RGB pour visualiser une image. On doit donc changer l'espace colorimétrique XYZ en RGB via un changement de repère :

$$I^{RGB}(i,\ j) = M_{XYZ \to RGB} I^{XYZ}(i,\ j) \tag{2.15}$$

Il existe une multitude d'espaces RGB : AdobeRGB, sRGB, NTSC, ... En fonction des capacités de l'écran, ces espaces sont plus au moins supportés (figure 2.6).

L'espace sRGB est l'espace le moins exigeant (gamut plus restreint : voir figure 2.6) et le mieux supporté. Voici son équation pour transformer des XYZ :

$$I^{RGB}(i,\ j) = \begin{bmatrix} 3.239886 & -1.536869 & -0.498444 \\ -0.967675 & 1.872930 & 0.041488 \\ 0.056595 & -0.207515 & 1.075305 \end{bmatrix} I^{XYZ}(i,\ j) \tag{2.16}$$

Pour afficher correctement sur un écran, il faut également appliquer une correction gamma. Le sRGB nécessite une correction gamma γ particulière sur chaque élément du triplet :

$$\gamma(u) = \begin{cases} 12.92u & \text{si } u \leq 0.0031308 \\ 1.055u^{1/2.4} - 0.055 & \text{sinon} \end{cases} \tag{2.17}$$

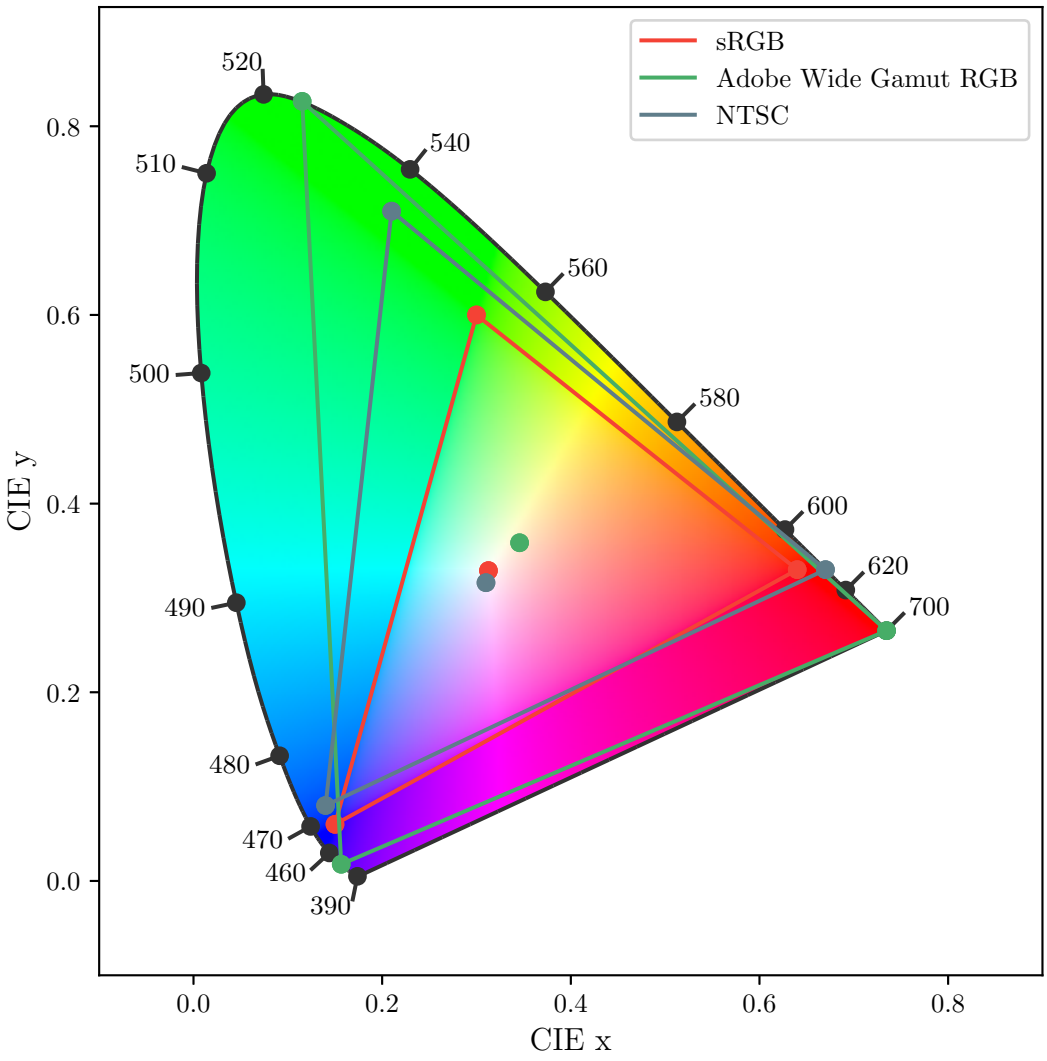


FIGURE 2.6. – Diagramme de chromaticité (gamut) CIE 1931 pour les espaces sRGB, Adobe Wide Gamut RGB et NTSC.

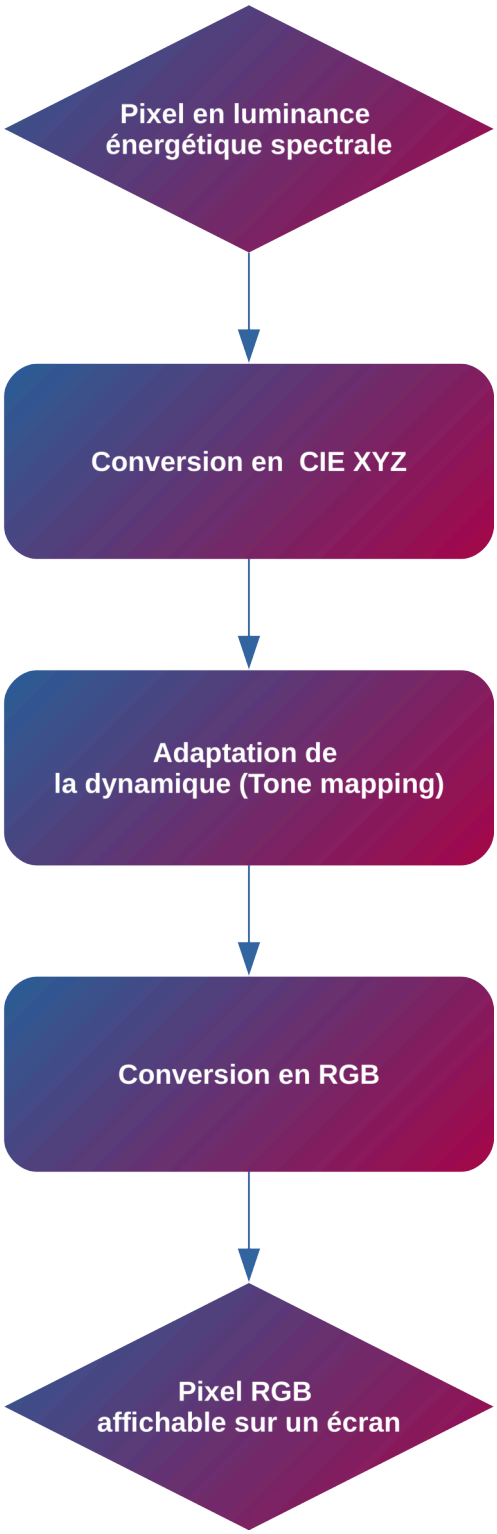


FIGURE 2.7. – Pipeline pour convertir une image spectrale en une image couleur affichable par un écran.

2.1.8.2. Dans les autres domaines spectraux

Pour les autres domaines spectraux (comme l'infrarouge dans notre cas), on va chercher à convertir l'image spectrale en image de niveau de luminance affichable par un écran (figure 2.8). Dans cette optique, il faut par ordre chronologique :

1. Intégrer l'image en luminance énergétique spectrale en image en luminance énergétique :

$$I^L(i,\ j) = \sum_{k=1}^{N_\Lambda} I^\lambda(i,\ j,\ k)\ (\lambda^{k+1} - \lambda^k) \tag{2.18}$$

   Si on veut simuler un capteur (et non un radiomètre parfait comme pour l'équation équation (2.18) ci-dessus), il faut prendre en compte la réponse (spectrale et spatiale) du capteur dans l'intégration :

$$I^L(i,\ j) = \sum_{k=1}^{N_\Lambda} I^\lambda(i,\ j,\ k)\ \overline{W_{ij}^k}\ (\lambda^{k+1} - \lambda^k) \tag{2.19}$$

   Au lieu d'intégrer l'ensemble de l'image spectrale, on peut également restreindre le domaine d'intégration spectral pour visualiser un canal en particulier du capteur.

2. Comme pour les valeurs XYZ du visible, il faut compresser la dynamique des valeurs fraîchement intégrées à l'étape 1 en luminance énergétique. Il faudra ensuite convertir l'image en couleur RGB. Pour cela il suffit de remplir le triplet avec la même valeur de luminance énergétique du pixel.

3. Et enfin et optionnellement, changer la table de correspondance des fausses couleur pour améliorer la visualisation.

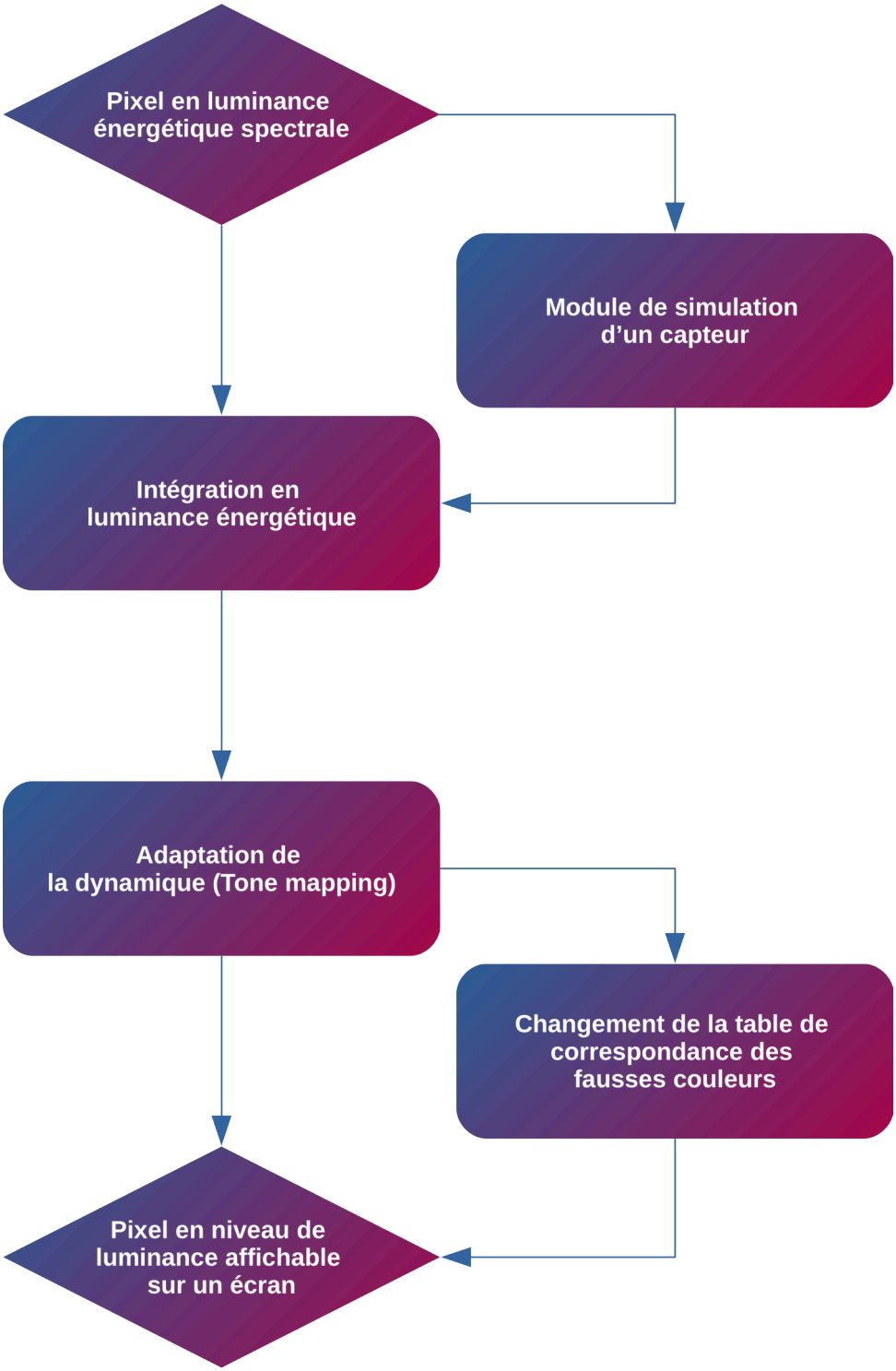


FIGURE 2.8. – Pipeline pour convertir une image spectrale en une image de niveau de luminance affichable par un écran.

2.1.9. De la couleur à la radiométrie

Malgré une récente démocratisation des capteurs multi et hyper spectraux, les données spectrales restent rares. Transformer une couleur ou une luminance en données spectrales est donc très tentant. Sans approximations et suppositions grossières, la transformation inverse n'est pas possible car l'étape d'intégration (1er étape) perd les précieuses informations spectrales.

Dans le visible, les travaux sur ce problème ont été initié par [Mac35]. La méthode la plus simple pour produire un spectre de réflectance plausible pour une couleur RGB donnée en entrée est proposée par [Smi99]. Il existe également des méthodes plus complexes et récentes qui produisent de meilleurs résultats : [Men+15], [OYH18], [MY19], [JH19].

Pour les autres domaines spectraux, les marges de manœuvre sont minces : mis à part prendre la courbe de réponse spectrale du capteur pour essayer de déduire un spectre plausible ou assumer que c'est un spectre constant qui a produit l'image en luminance énergétique.

Dans tous les cas, si on cherche de la précision (pour des analyses radiométriques ou photométriques par exemple), il faut bannir ce genre de donnée de nos scène. En effet, une couleur RGB peut être produite par une infinité de couple de spectres de réflectance et d'illuminant, cette notion se nome le métamérisme. Même si les méthodes récentes dans le visible produisent des spectres physiquement cohérents (qui conservent de l'énergie), elles ne permettent que de calculer un spectre possible dans les conditions d'illumination qui ont produit la couleur RGB. Ces méthodes ne permettent donc pas de reproduire avec fidélité la complexité spectrale des données. Elles ne devraient qu'être utilisées dans les cas ou la précision spectrale n'est pas importante ou pour faire des tests d'algorithmes si on n'a pas les données spectrales.

2.2. L'illumination globale

2.2.1. Équation de rendu

Afin de calculer l'illumination globale d'une scène, on doit simuler le transport de la lumière dans celle-ci. L'équation du rendu (nommée également LTE) formulée par [Kaj86] permet de décrire un bilan du rayonnement à l'équilibre (voir l'illustration de la section 2.2.1), voici la version spectrale pour les surfaces :

$$\begin{aligned} L_o(\lambda,\ v_{n-1},\ v_n) &= L_e(\lambda,\ v_{n-1},\ v_n) + \\ &\int_{\Omega_n} L_o(\lambda,\ v_n,\ \delta(v_n,\ \overrightarrow{\omega_i}))\ \tau(\lambda,\ \overrightarrow{\omega_i},\ \widehat{v_n v_{n-1}},\ \dots)\ |\cos\theta_{\overrightarrow{\omega_i}}|\ d\overrightarrow{\omega_i} \\ &= L_e(\lambda,\ v_{n-1},\ v_n) + L_r(\lambda,\ v_{n-1},\ v_n) \end{aligned} \tag{2.20}$$

Parfois on préfère la formulation par intégration surfacique de la scène à celle de l'intégration directionnelle (équation (2.20)) :

$$\begin{aligned} L_r(\lambda,\ v_{n-1},\ v_n) = L_e(\lambda,\ v_{n-1},\ v_n) + \\ \int_{\mathcal{A}-\{v_n\}} L_o(\lambda,\ v_n,\ v_{n+1})\ \tau(\lambda,\ \widehat{v_n v_{n+1}},\ \widehat{v_n v_{n-1}},\ \dots) \\ G(v_n,\ v_{n+1})\ dv_{n+1} \end{aligned} \tag{2.21}$$

C'est le terme géométrique $G(v_n,\ v_{n+1})$ qui permet de passer de la paramétrisation directionnelle à la paramétrisation surfacique de la scène :

$$G(v_n,\ v_{n+1}) = \frac{|(\widehat{v_n v_{n+1}} \cdot \overrightarrow{n}_{v_n})\ (\widehat{v_{n+1} v_n} \cdot \overrightarrow{n}_{v_{n+1}})|}{\|\overrightarrow{v_n v_{n+1}}\|^2}\ \mathcal{V}(v_n,\ v_{n+1}) \tag{2.22}$$

L'équation du rendu se base sur l'optique géométrique. Sans extension, les phénomènes lumineux suivants sont donc incompatibles :

- Polarisation,
- Interférence,
- Effet Doppler relativiste,
- Fluorescence,
- Phosphorescence,
- Optique non linéaire.

Elle est composée d'une intégrale à très grande dimension du second type de Fredholm et donc impossible à résoudre analytiquement en pratique. Par conséquent on la résout numériquement via deux classes de méthodes :

— **La méthode par éléments finis (radiosité) :** Ces algorithmes approximent l'équation pour ne prendre en compte que les interactions diffuses. Ils sont par conséquent inintéressants dans notre cas.

— **La méthode de Monte-Carlo (section 2.2.3) :** La méthode est robuste mais converge lentement.

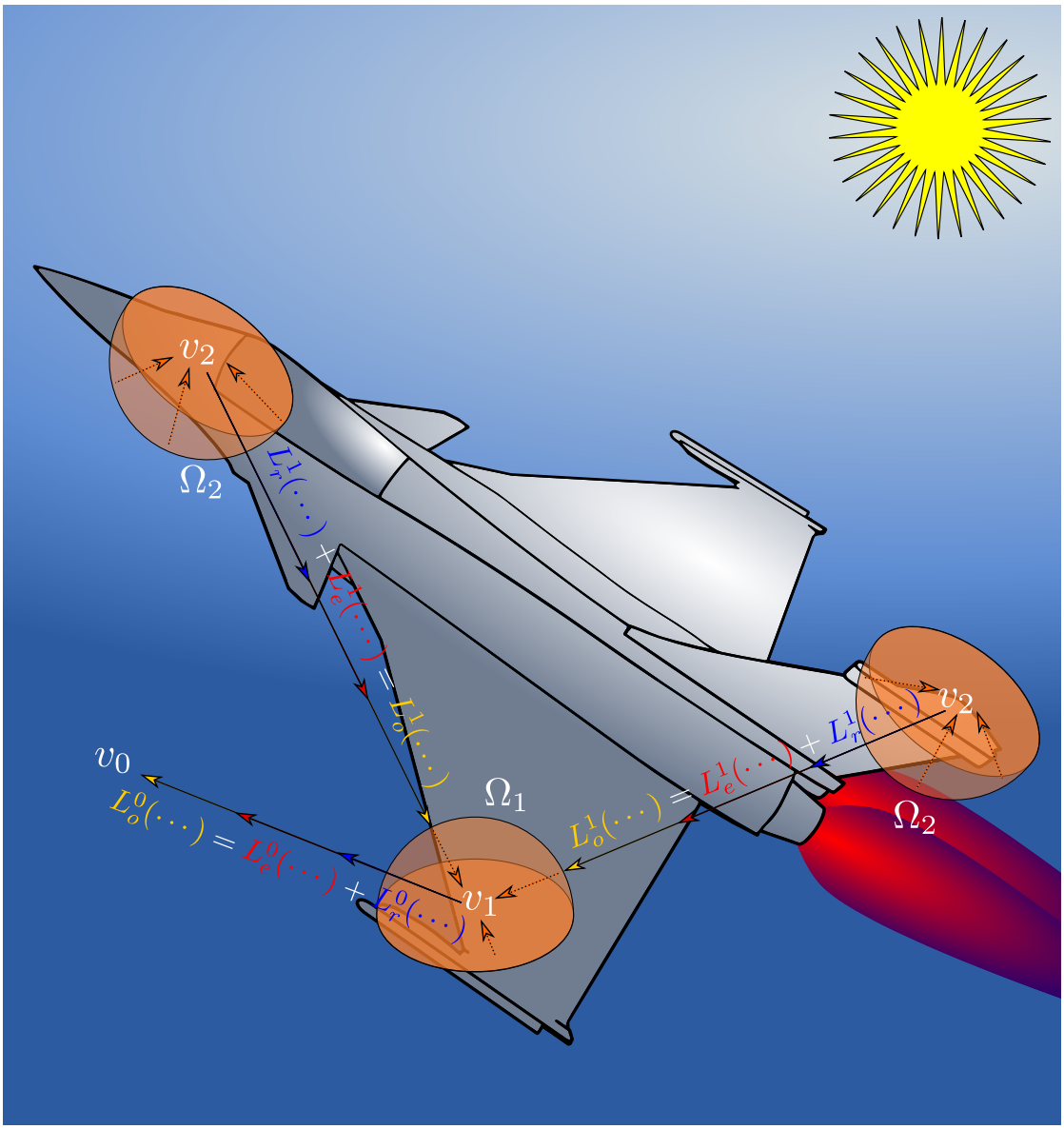


FIGURE 2.9. – Illustration d'un exemple de calcul de l'équation du rendu (équation (2.20)) pour deux récursions. Des exposants ont été rajoutés à L_o, L_e et L_r pour faciliter la visualisation de l'aspect récursif de l'équation.

2.2.2. Formulation par intégrale de chemin

L'équation du rendu (équation (2.20)) n'est pas adaptée à décrire le fonctionnement d'algorithme complexe. Dans sa thèse, [Vea98] a proposé (à partir de la paramétrisation surfacique de l'équation du rendu (équation (2.20))) la formulation par intégrale de chemin de lumière qui permet de représenter le calcul de la valeur d'un pixel par un algorithme d'illumination globale, voici une version spectrale de cette formule :

$$I^{\lambda}(i,\ j,\ k) = \int_{\overline{\Omega}_{ij}} f_{ij}(\lambda^k,\ \overline{\omega})\, d\overline{\omega} \tag{2.23}$$

Cette formulation est plus générique, flexible et concise que l'équation du rendu (équation (2.20)). L'idée est de considérer des chemins $\overline{\omega}$ composés de N_v sommets :

$$\overline{\omega} = v_0\, v_1 \ldots v_{N_v} \tag{2.24}$$

Par convention, on considère que le premier sommet v_0 est sur la lumière et que le dernier sommet v_{N_v} est sur le capteur. Afin d'évaluer la luminance spectrale, on définit une fonction de mesure $f_{ij}(\lambda^k,\ \overline{\omega})$ pour ce chemin $\overline{\omega}$ (voir l'illustration de la section 2.2.2) :

$$f_{ij}(\lambda^k,\ \overline{\omega}) = L_e(\lambda,\ v_0,\ v_1)\, \mathcal{T}(\lambda,\ \overline{\omega})\ W_{ij}(v_{N_v-1},\ v_{N_v}) \tag{2.25}$$

Dont l'accumulation des atténuations des rebonds $\mathcal{T}(\lambda,\ \overline{\omega})$ est égale à :

$$\mathcal{T}(\lambda,\ \overline{\omega}) = G(v_0,\ v_1) \prod_{n=1}^{N_v} \tau(\lambda,\ \widehat{v_n v_{n-1}},\ \widehat{v_n v_{n+1}},\ \ldots)\, G(v_n,\ v_{n+1}) \tag{2.26}$$

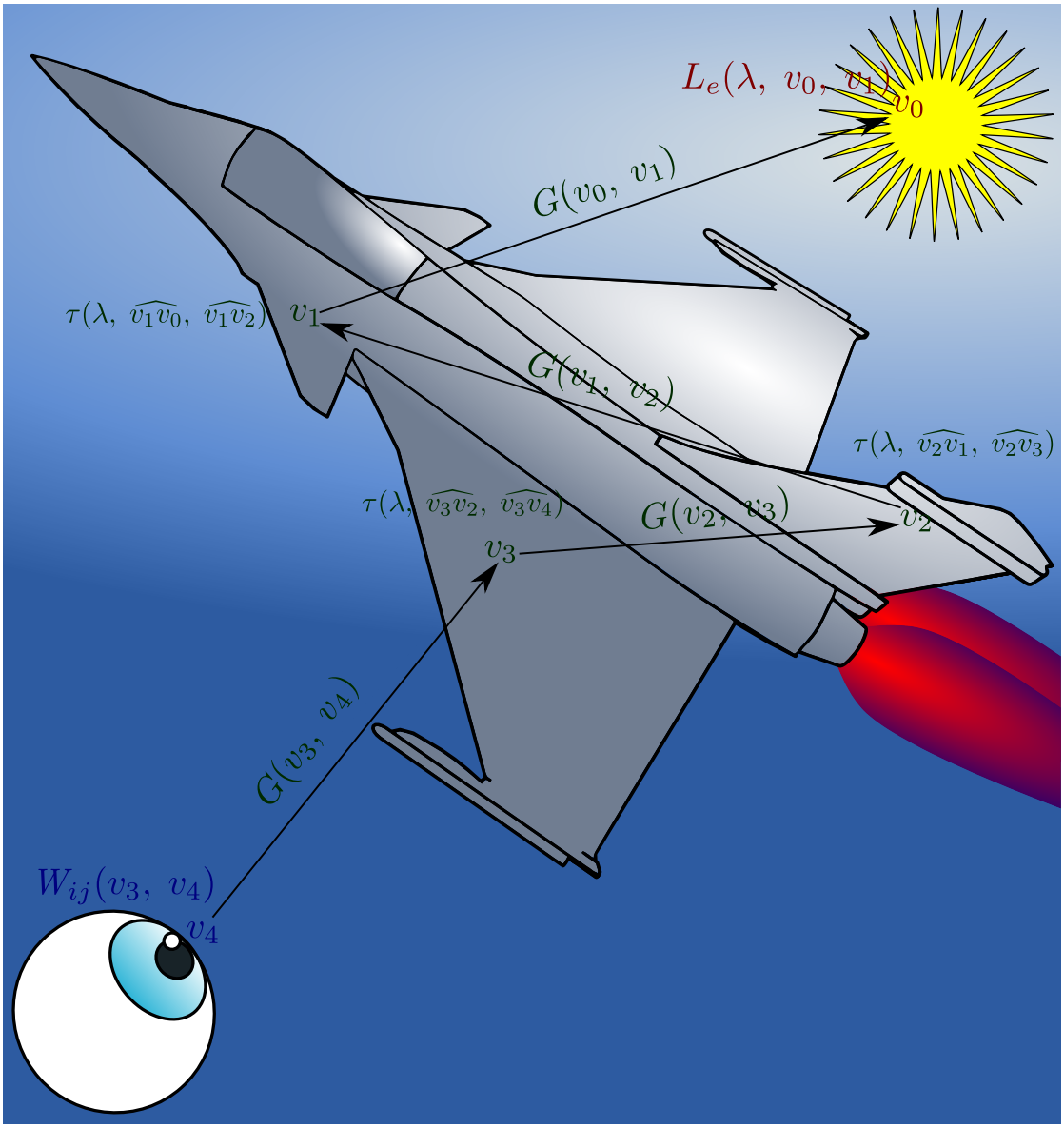


FIGURE 2.10. – Illustration d'un exemple de calcul de la fonction de mesure de Veach (équation (2.25)) pour un chemin de 5 sommets.

2.2.3. La méthode de Monte-Carlo

La méthode de Monte-Carlo est généralement utilisée pour calculer des intégrales à très hautes dimensions comme celles de l'équation (2.20) et de l'équation (2.23). C'est une méthode robuste et sa convergence est insensible à la dimension de l'intégrale.

Cette méthode se base sur l'espérance mathématique de la fonction cible pour calculer son intégrale. Considérons pour l'exemple le calcul de l'intégrale de l'équation (2.23). La méthode de Monte-Carlo va dans ce cas considérer que :

$$\begin{aligned} \langle\, I^{\lambda}(i,\ j,\ k)\,\rangle &= \mathbb{E}\left[\frac{f_{ij}(\lambda^k,\ \overline{\omega})}{\rho(\overline{\omega})}\right] \\ &= \int_{\overline{\Omega}_{ij}} \frac{f_{ij}(\lambda^k,\ \overline{\omega})}{\rho(\overline{\omega})}\ \rho(\overline{\omega})\ d\overline{\omega} \\ &\approx \frac{1}{N_s}\sum_{s=1}^{N_s} \frac{f_{ij}(\lambda^k,\ \overline{\omega}_s)}{\rho(\overline{\omega}_s)} \end{aligned} \tag{2.27}$$

Avec la loi des grands nombres, on obtient notre estimateur non biaisé et consistant :

$$\langle\, I^{\lambda}(i,\ j,\ k)\,\rangle = \lim_{N_s\to+\infty} \frac{1}{N_s}\sum_{s=1}^{N_s} \frac{f_{ij}(\lambda^k,\ \overline{\omega}_s)}{\rho(\overline{\omega}_s)} \tag{2.28}$$

Si l'échantillonnage est uniforme, on a la densité de probabilité qui est égale à :

$$\rho(\overline{\omega}_s) = \frac{1}{\Delta_{\overline{\Omega}_{ij}}} \tag{2.29}$$

L'équation (2.27) peut donc se simplifier :

$$\langle\, I^{\lambda}(i,\ j,\ k)\,\rangle = \lim_{N_s\to+\infty} \frac{\Delta_{\overline{\Omega}_{ij}}}{N_s}\sum_{s=1}^{N_s} f_{ij}(\lambda^k,\ \overline{\omega}_s) \tag{2.30}$$

La vitesse de convergence de la méthode est de l'ordre de $\frac{1}{\sqrt{N_s}}$ ce qui peut paraître peu en petite dimension mais salvateur en grande dimension comme dans notre cas.

2.2.4. Échantillonnage préférentiel

L'échantillonnage préférentiel est une technique de minimisation de variance qui permet d'accélérer la convergence des méthodes Monte-Carlo. L'idée est d'échantillonner de manière non uniforme avec un critère d'échantillonnage qui mime la tendance de la fonction à intégrer.

La densité de probabilité $\rho(\overline{\omega}_s\ )$ n'étant donc plus uniforme, l'équation (2.30) ne peut plus être utilisée. Il faut donc revenir à l'équation (2.28).

2.2.5. Multi Importance Sampling

Une technique d'échantillonnage n'est pas optimale dans toutes les circonstances. Par exemple, on peut voir qu'il n'est pas efficace d'échantillonner la BRDF (figure 2.11a) quand la lumière est petite et que la surface est rugueuse et vice versa pour l'échantillonnage des lumières (figure 2.11b).

Dans ses travaux [VG95], Eric Veach a cherché à combiner différentes stratégies d'échantillonnages de manière optimale (figure 2.11c) dans le cadre du calcul de l'équation (2.23) via du Monte-Carlo. Pour cela il a introduit un modèle de multi échantillonnage préférentiel (MIS) en pondérant les différentes techniques :

$$\langle\, I^{\lambda}(i,\ j,\ k)\,\rangle = \lim_{N_s \to +\infty} \sum_{t=1}^{N_t} \frac{1}{N_{ts}} \sum_{s=1}^{N_{ts}} w_t(\overline{\omega}_s)\ \frac{f_{ij}(\lambda^k,\ \overline{\omega}_s)}{\rho_t(\overline{\omega}_s)} \tag{2.31}$$

Le nouveau estimateur est non biaisé et consistant si les conditions suivantes sont remplies :

$$\begin{cases} \sum\limits_{t=1}^{N_t} w_t(\overline{\omega}) = 1 & \text{si } f_{ij}(\lambda^k,\ \overline{\omega}) \neq 0 \\ w_t(\overline{\omega}) = 0 & \text{si } \rho_t(\overline{\omega}) = 0 \end{cases} \tag{2.32}$$

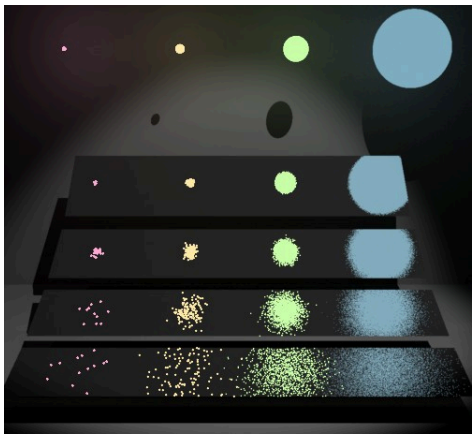
Les variances des techniques d'échantillonnages sont additives entre elles, il faut donc calculer les poids w_t de manière à les minimiser. Veach proposa des heuristiques pour calculer ces poids dont voici les plus connues et optimales (d'après [VG95]) :

— l'heuristique de la balance :

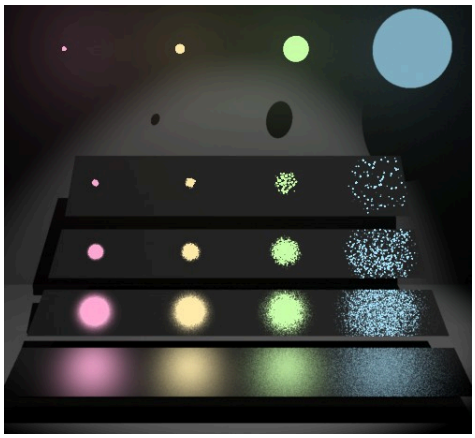
$$w_t(\overline{\omega}) = \frac{N_{ts}\ \rho_t(\overline{\omega})}{\sum\limits_{t'=1}^{N_t} N_{t's}\ \rho_{t'}(\overline{\omega})} \tag{2.33}$$

— l'heuristique de la puissance (généralement $\alpha = 2$) :

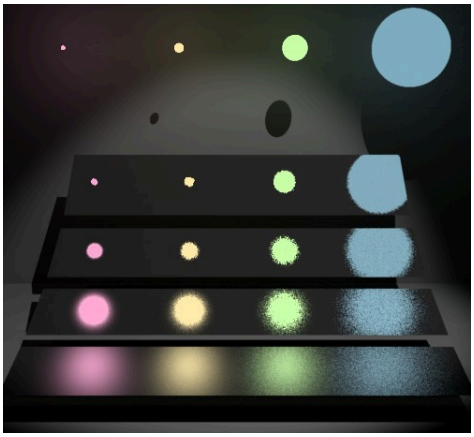
$$w_t(\overline{\omega}) = \frac{(N_{ts}\ \rho_t(\overline{\omega}))^\alpha}{\sum_{t'=1}^{N_t} (N_{t's}\ \rho_{t'}(\overline{\omega}))^\alpha} \tag{2.34}$$



(a) Échantillonnage avec les BRDFs.



(b) Échantillonnage avec les lumières.



(c) Combinaison des techniques d'échantillonnages avec du MIS et l'heuristique de la puissance (équation (2.34)).

FIGURE 2.11. – Intérêt du MIS (les images proviennent de [VG95]).

2.2.6. Longueurs des chemins de lumière

Un chemin de lumière peut rebondir indéfiniment sur des objets d'une scène. Sa longueur peut donc poser des problèmes de calcul et de consommation mémoire. Plusieurs stratégies (non exclusives entre elles) peuvent être utilisées pour diminuer la taille des chemins :

— Fixer un nombre de rebonds maximum N_v : cette stratégie entraîne un biais si le nombre de rebonds est trop faible pour la scène. Elle peut également entraîner des calculs inutiles si le nombre de rebonds est trop important et que la scène est trop simple.

— Fixer un seuil d'arrêt pour l'accumulation de l'atténuation $\mathcal{T}$: cette technique entraîne également un biais important et difficilement maîtrisable. Elle est donc très peu utilisée.

— Utiliser la roulette russe (voir algorithme 1) : cette méthode permet de réduire la profondeur des chemins sans biaiser le résultat. À chaque rebond, l'idée est de stochastiquement choisir à partir d'une probabilité (ligne 2 de l'algorithme 1) si on continue le chemin ou pas. Si le chemin « survit » à la roulette russe, son atténuation $\mathcal{T}$ doit être divisée par la probabilité de survie de l'échantillon (ligne 5 de l'algorithme 1) pour ne pas biaiser le résultat.

Algorithm 1: Méthode de la roulette russe.

```
1 Function ϱ_rr(ρ_s, RNG, 𝒯) :
2     killPath ← RNG() > ρ_s
3
4     if killPath is false then
5         𝒯 ← 𝒯 / ρ_s
6     end
7
8     return killPath
```

2.2.7. Path Tracing

Afin de résoudre l'équation (2.20), [Kaj86] introduisit au même moment la méthode du « Path Tracing » qui est de type Monte-Carlo. C'est une méthode très simple qui consiste à tracer des chemins lumineux de la caméra vers les sources de lumière. Elle est souvent utilisée comme référence pour comparer les résultats des autres méthodes. De plus, sa parallélisation GPU est relativement simple et efficace par rapport aux autres algorithmes d'illumination globale plus complexes.

2.2.7.1. Le Path Tracing naïf

Dans sa version originale (algorithme 2), les chemins sont construits incrémentalement en rebondissant sur les objets de la scène en fonction de la nature de leurs matériaux.

Algorithm 2: Version itérative et spectrale du Path Tracing naïf.

```
1  Function sampleAndEvalPath(λ, i, j, RNG) :
2      L_o, T ← 0, 1  // Initialisation.
3      v ← ϱ_e(i, j, RNG)  // Génération d'un sommet primaire sur le capteur.
4
       // Construction et évaluation itératif du chemin de lumière.
5      while ... do
           // Calcul d'intersection du prochain rebond.
6          v ← δ(v.x, v.ω_i)
7
           // Ajout de la contribution de la carte d'environement si
           // le rayon ne touche rien.
8          if v.x = x_∅ then
9              L_o ← L_o + T × ϑ(λ, v)
10             break
11         end
12
           // Ajout de l'émission propre.
13         L_o ← L_o + T × L_e[v](λ, v)
14
           // Échantillonnage d'une nouvelle direction.
15         v.ω_i, τ, ρ ← ϱ_τ[v](λ, v, RNG)
16
           // Accumulation de l'atténuation de l'échantillon.
17         T ← T × τ/ρ
18
           // Roulette russe: la probabilité de survie est
           // ici proportionelle à l'atténuation du chemin.
19         if ϱ_rr(T, RNG, T) = true then
20             break
21         end
22     end
23
24     return L_o
```

Afin que que L_o ne soit pas nulle et donc utile, on a besoin que le chemin touche une surface dont l'émission n'est pas nulle (ligne 13 de l'algorithme 2) ou qu'une carte d'éclairement soit disponible dans la scène (ligne 9 de l'algorithme 2).

Or dans cette version du Path Tracing, on construit les chemins en interrogeant les matériaux (ligne 15 de l'algorithme 2) qui ne prennent pas en compte la visibilité des lumières dans le choix de la nouvelle direction à chaque rebond.

Dans le cas où la scène contient des lumières cachées ou très petites, on a très peu de chance qu'elles soient touchées par des chemins de lumière. Cela ralentit considérablement la convergence de l'algorithme.

2.2.7.2. Le Path Tracing explicite

Afin de pallier à ce défaut, cette méthode va explicitement chercher à connecter directement chaque sommet d'un chemin aux sources de lumières (en les échantillonnant, voir l'algorithme 3) au lieu d'attendre que celui-ci tombe dessus par hasard en échantillonnant les matériaux. Cette partie du Path Tracing est souvent nommée Next Event Estimation (NEE).

Si la lumière est directement connectable, on parle alors de lumière directe à l'opposé de la lumière indirecte qui est issue de plusieurs rebonds. L'essence de cette méthode est donc de séparer l'estimation de L_r dans l'équation (2.20) en deux estimateurs pour la lumière directe L_d et la lumière indirecte L_i :

$$
\begin{aligned}
L_d(\lambda,\ v_{n-1},\ v_n) &= \int_{\mathcal{A}_{L_d}} L_e(\lambda,\ v_n,\ v_{n+1})\ \tau(\lambda,\ \widehat{v_n v_{n+1}},\ \widehat{v_n v_{n-1}},\ \dots) \\
&\qquad G(v_n,\ v_{n+1})\ dv_{n+1} \\
L_i(\lambda,\ v_{n-1},\ v_n) &= \int_{\Omega_n} L_r(\lambda,\ v_n,\ \delta(v_n,\ \overrightarrow{\omega_i}))\ \tau(\lambda,\ \overrightarrow{\omega_i},\ \widehat{v_n v_{n-1}},\ \dots)\ |\cos\theta_{\overrightarrow{\omega_i}}|\ d\overrightarrow{\omega_i} \\
L_r(\lambda,\ v_{n-1},\ v_n) &= L_d(\lambda,\ v_{n-1},\ v_n) + L_i(\lambda,\ v_{n-1},\ v_n)
\end{aligned}
\tag{2.35}
$$

On a donc maintenant L_o qui est égale à :

$$
L_o(\lambda,\ v_{n-1},\ v_n) = L_e(\lambda,\ v_{n-1},\ v_n) + L_d(\lambda,\ v_{n-1},\ v_n) + L_i(\lambda,\ v_{n-1},\ v_n) \tag{2.36}
$$

Pour fonctionner, il faut également veiller à respecter les règles suivantes :

— Les matériaux en fonction de Dirac (matériau spéculaire à surface parfaitement lisse) ne sont pas échantillonnables via une source de lumière (ligne 20 de l'algorithme 4) : c'est à dire que pour une direction incidente $\overrightarrow{\omega_i}$, il y a qu'une seul et unique direction sortante $\overrightarrow{\omega_o}$ possible. La probabilité que le $\overrightarrow{\omega_o}$ unique

(associé à un $\overrightarrow{\omega_i}$) du matériau coïncide avec la direction échantillonnée par la lumière est quasiment nulle et donc inutile à calculer.

[KD13] et [Bou+13] ont proposé des solutions à ce problème en introduisant des techniques de régularisation. Ces méthodes régularisent la distribution de Dirac en introduisant du biais, qui se résorbe au cours des itérations.

— Il faut prendre en compte L_e (ligne 16 de l'algorithme 4) et la carte d'environnement (ligne 10 de l'algorithme 4) uniquement au premier rebond et pour les rayons provenant de matériau en fonction de Dirac. Sinon la lumière directe sera prise en compte plusieurs fois.

Dans l'infrarouge, [Coi12] utilise une variante de cette technique. L'idée de l'auteur est de restreindre L_d au soleil car c'est un contributeur important dans notre cas quand on se rapproche du visible.

Toutefois, dans ces bandes spectrales, toutes les surfaces de la scène émettent de la lumière, il serait donc intéressant d'étendre L_d à toutes les lumières de la scène. Cela permettrait d'optimiser le rendu des scènes de nuit ou dans les bandes LWIR et VLWIR où le soleil n'est pas le contributeur principal de la scène. Pour cela, nous avons préféré d'utiliser la version MIS de cette méthode (section 2.2.7.3).

Algorithm 3: Fonction d'échantillonnage de la lumière directe L_d.

1 **Function** $\varrho_{L_d}(\lambda,\ v,\ RNG)$ **:**
2 $\quad L_d \leftarrow 0$
3 $\quad \overrightarrow{\omega_i},\ \ell,\ L,\ \rho \leftarrow \varrho_l(\lambda,\ v,\ RNG)$
4
`// Test si le rayon d'ombrage n'est pas occulté.`
5 $\quad$ **if** $\delta(v.x,\ \overrightarrow{\omega_i},\ \ell).x = x_\varnothing$ **then**
6 $\quad\quad L_d \leftarrow \mathcal{T} \times \frac{\tau[v](\lambda,\ \overrightarrow{\omega_i},\ v.\overrightarrow{\omega_o},\ \dots)}{\tau[v].\rho(\lambda,\ \overrightarrow{\omega_i},\ v.\overrightarrow{\omega_o},\ \dots)} \times \frac{L}{\rho}$
7 $\quad$ **end**
8
9 $\quad$ **return** L_d

Algorithm 4: Version itérative et spectrale du Path Tracing explicite.

```
1  Function sampleAndEvalPath(λ, i, j, RNG) :
2      L_o, 𝒯, isDirac ← 0, 1, false  // Initialisation.
3      v ← ϱ_c(i, j, RNG)  // Génération d'un sommet primaire sur le capteur.
4
       // Construction et évaluation itératif du chemin de lumière.
5      while ... do
           // Calcul d'intersection du prochain rebond.
6          v ← δ(v.x, v.ω⃗_i)
7
           // Ajout de la contribution de la carte d'environement.
8          if v.x = x_∅ then
9              if v.n = 0 or isDirac = true then
10                 L_o ← L_o + 𝒯 × ϑ(λ, v)
11             end
12             break
13         end
14
           // Ajout de l'émission propre.
15         if v.n = 0 or isDirac = true then
16             L_o ← L_o + 𝒯 × L_e[v](λ, v)
17         end
18
           // Échantillonnage de la lumière directe (NEE)
19         isDirac ← v.τ = DiracBSDF
20         if isDirac = false then
21             L_o ← L_o + ϱ_{L_d}(λ, v, RNG)
22         end
23
           // Échantillonnage de la BSDF.
24         v.ω⃗_i, τ, ρ ← ϱ_τ[v](λ, v, RNG)
25         𝒯 ← 𝒯 × τ/ρ
26
           // Roulette russe.
27         if ϱ_rr(𝒯, RNG, 𝒯) = true then
28             break
29         end
30     end
31
32     return L_o
```

2.2.7.3. Le Path Tracing explicite avec du MIS

L'objectif de cette méthode est également de connecter directement et explicitement chaque sommet du chemin aux sources de lumières en les échantillonnant.

La théorie par derrière est cependant différente. En effet, au lieu de séparer la lumière directe L_d et la lumière indirecte L_i de L_r en deux estimateurs indépendants, cette méthode va considérer L_d (échantillonnage via les sources de lumières) et L_i (échantillonnage via les BSDFs) comme des techniques d'échantillonnages différentes afin de les combiner optimalement via du MIS (équation (2.31)).

Cette méthode est plus performante et plus flexible que la précédente, c'est celle que l'on utilisera tout au long des chapitres.

Comme pour le Path Tracing explicite, il faut veiller à respecter les règles suivantes :

— Il faut prend en compte les poids du MIS pour la technique d'échantillonnage des lumières (ligne 25 de l'algorithme 5) et pour celle des BSDFs (ligne 20 de l'algorithme 5 et ligne 12 de l'algorithme 5)

— Il ne faut cependant pas ajouter les poids du MIS aux échantillons de la technique des BSDFs (ligne 18 de l'algorithme 5 et ligne 10 de l'algorithme 5) au premier rebond ou quand le matériau précédent était un matériau en fonction de Dirac car on ne peut pas évaluer les poids dans ce cas.

— Les matériaux en fonction de Dirac ne sont pas échantillonnables via une source de lumière (ligne 24 de l'algorithme 5).

Algorithm 5: Version itérative et spectrale de la version MIS du PT explicite.

```
1  Function sampleAndEvalPath(λ, i, j, RNG) :
2      L_o, 𝒯, isDirac ← 0, 1, false  // Initialisation.
3      v ← ϱ_c(i, j, RNG)  // Génération d'un sommet primaire sur le capteur.
4
5      while ... do
6          v ← δ(v.x, v.ω⃗_i)  // Calcul d'intersection du prochain rebond.
7
           // Ajout de la contribution de la carte d'environement.
8          if v.x = x_∅ then
9              if v.n = 0 or isDirac = true then
10                 L_o ← L_o + 𝒯 × ϑ(λ, v)
11             else
12                 L_o ← L_o + 𝒯 × w_{L_i}(v) × ϑ(λ, v)
13             end
14             break
15         end
16
           // Ajout de l'émission propre.
17         if v.n = 0 or isDirac = true then
18             L_o ← L_o + 𝒯 × L_e[v](λ, v)
19         else
20             L_o ← L_o + 𝒯 × w_{L_i}(v) × L_e[v](λ, v)
21         end
22
           // Échantillonnage de la lumière directe (NEE)
23         isDirac ← v.τ = DiracBSDF
24         if isDirac = false then
25             L_o ← L_o + w_{L_d}(v) × ϱ_{L_d}(λ, v, RNG)
26         end
27
           // Échantillonnage de la BSDF.
28         v.ω⃗_i, τ, ρ ← ϱ_τ[v](λ, v, RNG)
29         𝒯 ← 𝒯 × τ/ρ
30
           // Roulette russe.
31         if ϱ_rr(𝒯, RNG, 𝒯) = true then
32             break
33         end
34     end
35
36     return L_o
```

2.3. Pré-requis en parallélisation GPU

2.3.1. Présentation et historique

Un Graphic Processing Unit (GPU) ou un processeur graphique en français est un coprocesseur massivement parallèle situé généralement sur une carte graphique mais il peut également être hébergé par un CPU ou plus rarement sur une carte mère (premier ordinateur).

On parle de GPU dédié ou discret quand il se trouve sur une carte graphique. Dans ce cas le GPU est branché à un slot PCI Express et possède sa propre mémoire (VRAM). Autrement quand le GPU est hébergé par processeur on parle de GPU intégré et il partage sa mémoire avec le CPU.

2.3.1.1. Le pipeline programmable

Historiquement, ils étaient spécialisés dans les tâches de calcul d'affichage (rastérisation). À partir de 2001, les principaux constructeurs (Nvidia et ATI) ont commencé à flexibiliser leurs cartes graphiques pour les jeux vidéos en rendant programmables certaines parties du pipeline de rastérisation (figure 2.12). Ces parties se nomment des « shaders » (« nuanceur » en français). Cette notion a été introduite pour la première fois en 1988 par Pixar pour ses films dans sa spécification d'interface de RenderMan (« RenderMan Interface Specification, Version 3.0 »). À l'époque il n'y avait que ces deux types de shader :

— Vertex shader : qui prend en entrée un sommet d'un triangle avec un ensemble d'attributs (normales, coordonnées textures, ...) et renvoie un sommet modifié.

— Fragment shader : qui prend en entrée un fragment issu de la rastérisation et peut renvoyer un ensemble de couleurs ou de valeurs de profondeur.

Les pipelines les plus récents ajoutent maintenant d'autre types de shader : Tesselation, Geometry et Compute shader

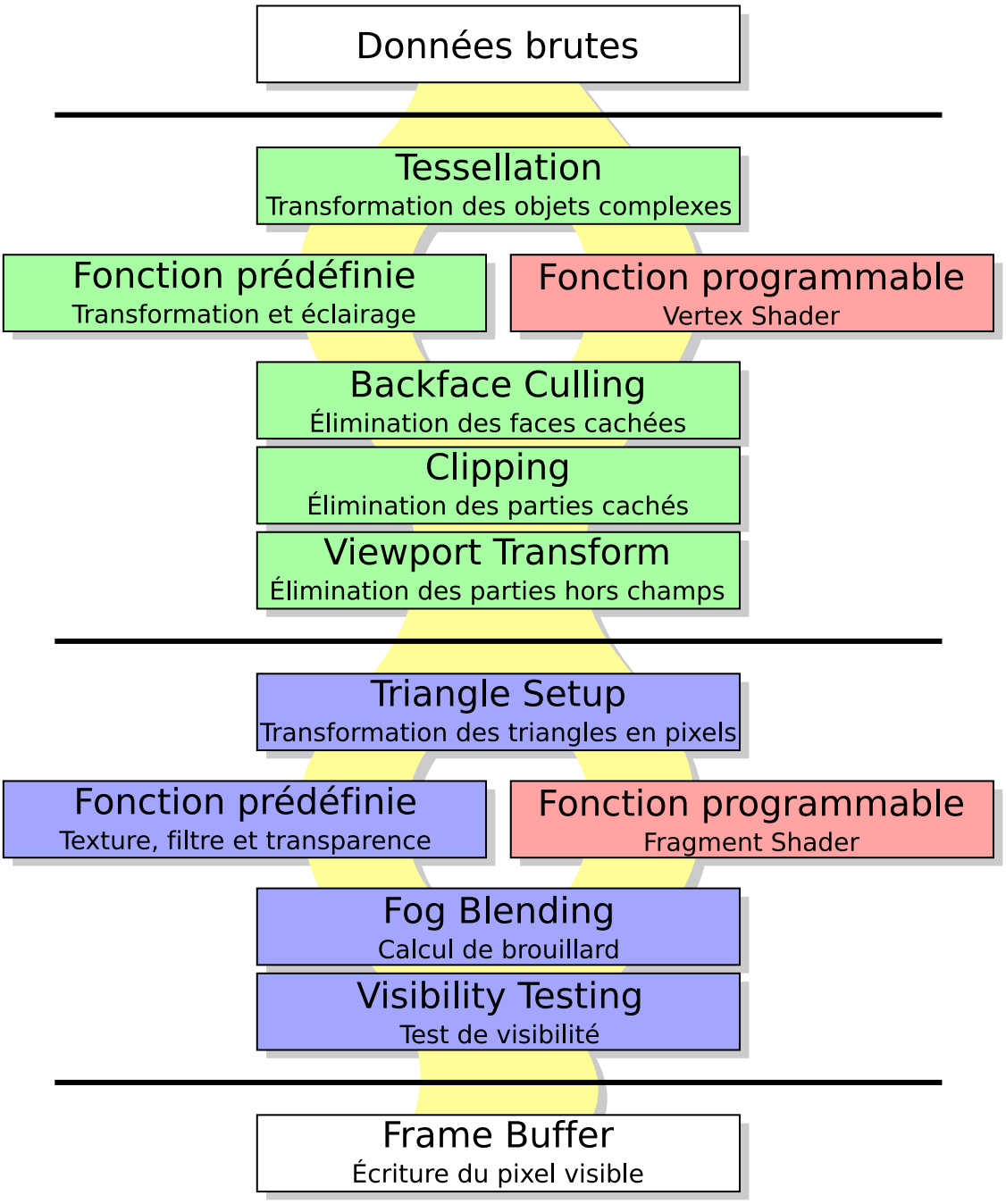


FIGURE 2.12. – Premier pipeline programmable des GPUs pour la rastérisation [Bay]. Les parties en rouge (vertex et fragment shader) sont programmables.

2.3.1.2. Le GPGPU

Ces nouvelles évolutions en termes de flexibilité, de puissance de calcul massivement parallèle et un rapport performance/prix avantageux ont amené les développeurs à détourner la fonction initiale des GPU pour effectuer d'autres types de calculs parallèles via des shaders. C'est ce que l'on a nommé le General-Purpose computing on Graphics Processing Units (GPGPU). Afin de répondre à ces nouveaux besoins, les constructeurs de cartes graphiques ont par la suite amélioré et développé des technologies propres et optimisées pour le GPGPU. Aujourd'hui, leurs usages ne se cantonnent plus uniquement à cette tâche initiale de calcul d'affichage, comme par exemple :

— Décodage / encodage vidéo.
— Simulation CFD.
— Rendu en lancer de rayon.
— Apprentissage pour de l'intelligence artificielle (« Deep learning », ...).

2.3.2. Architecture et fonctionnement

2.3.2.1. Paradigme de parallélisation

Il existe plusieurs paradigmes de parallélisation. La taxonomie de Flynn proposée par l'américain Michael J. Flynn en 1966 permet de classer les architectures d'ordinateur selon le type d'organisation des flux de données et d'instructions :

— Single Instruction Single Data (SISD) : Cette architecture purement séquentielle fonctionne avec une instruction par donnée. Elle fut utilisée par les premiers ordinateurs. Cette classe correspond à l'architecture de Von Neumann.

— Single Instruction Multiple Data (SIMD) : Elle fonctionne en parallélisant la même instruction sur plusieurs données différentes. C'est le paradigme utilisé par les processeurs vectoriels.

— Multiple Instructions Single Data (MISD) : Plusieurs instructions traitent en parallèle exactement la même donnée. Il existe peu de cas d'usage de cette classe mis à part quelques domaines comme le filtrage numérique ou encore la vérification de redondance dans les systèmes critiques.

— Multiple Instructions Multiple Data (MIMD) : C'est l'architecture des CPUs et donc celle qui est la plus utilisée et la plus flexible. Elle permet de traiter en parallèle différentes instructions avec différentes données.

Les 1er GPU utilisaient le paradigme SIMD. Les GPUs modernes utilisent maintenant le paradigme Single Instruction Multiple Threads (SIMT) qui consiste à rajouter un niveau de parallélisme supplémentaire au paradigme SIMD. L'idée est de flexibiliser sont utilisation, en automatisant la gestion des branches (divergence

d'instruction) et en ordonnançant l'exécution de plusieurs tâches sur plusieurs unités SIMD.

2.3.2.2. Modèle d'exécution

La programmation GPGPU se résume à lancer des noyaux (« kernel » en anglais) sur un ou des GPUs depuis le code hôte.

Les noyaux sont les points d'entrée des calculs sur GPU et peuvent ingérer des paramètres et des buffers (tableau continu de donnée) en entrée pour calculer et modifier des paramètres ou des buffers en sortie. Les noyaux sont ce que l'on appelle le code « device ». En opposition au code hôte qui est le code qui est exécuté par le programme principal.

Le nombre de tâches (« thread » en CUDA ou « work-item » en OpenCL) des noyaux est généralement défini sur une grille (1D, 2D ou en 3D : voir figure 2.13) par le développeur.

Cette grille de calcul est elle-même divisée par des groupes de tâches appelés work-group (ou « block » en CUDA). Un groupe de tâches a la possibilité de se synchroniser (barrière mémoire) ou encore de partager de la mémoire locale (ou « shared memory » en CUDA).

En lançant un noyau, le GPU va essayer d'exécuter un maximum de groupes de tâches simultanément. Ce nombre de tâches dépend des limites et des ressources matérielles disponibles, c'est-à-dire :
— Du nombre de registres alloués par tâche.
— De la taille de la mémoire locale allouée par groupe de tâches.
— De la taille des groupes de tâches.
— De la taille et du nombre d'unité SIMD.

De par son architecture en SIMT, un GPU peut mettre en pause ou reprendre l'exécution de un ou de plusieurs groupe de tâches pour optimiser le taux d'occupation du GPU (c'est ce qu'on appelle l'« occupancy » en anglais). Augmenter ce taux d'occupation peut être une stratégie d'optimisation pour cacher la latence des accès mémoires qui sont relativement longs par rapport à la puissance de calcul d'un GPU.

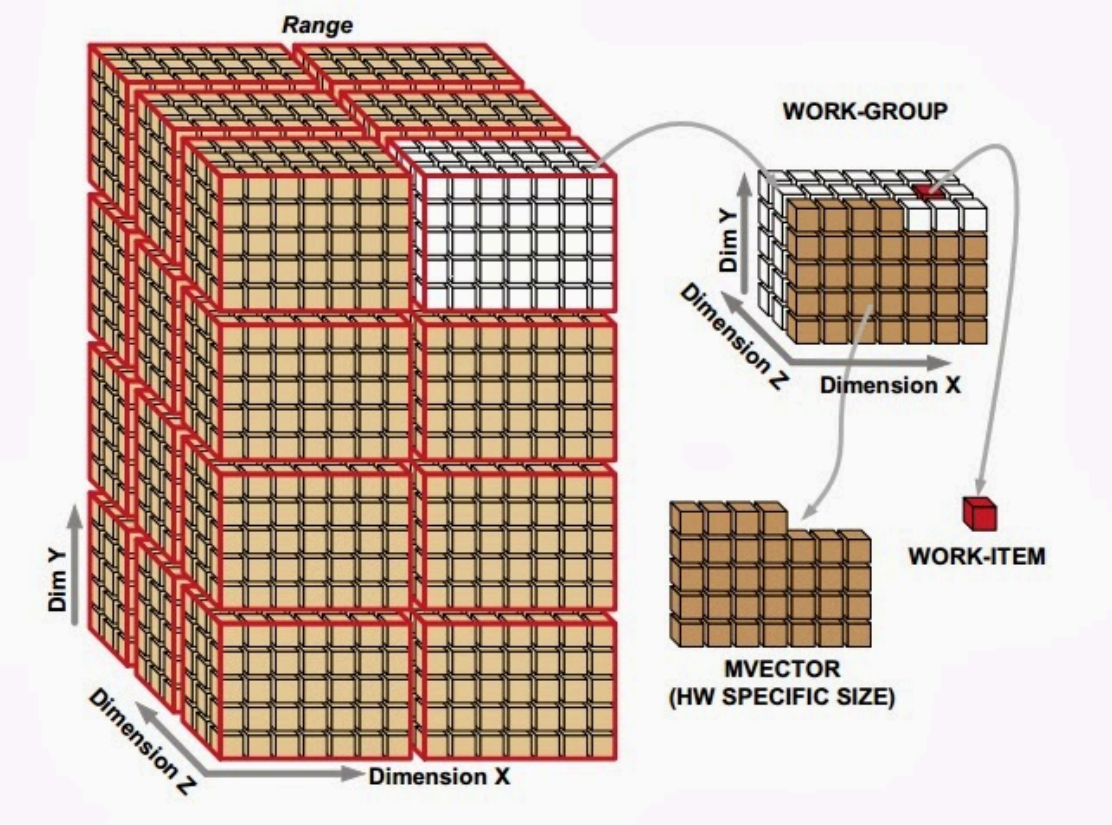


FIGURE 2.13. – Modèle d'exécution utilisé généralement par les APIs GPGPU (source : AMD OpenCL User Guide 2015).

2.3.2.3. Les différents types de mémoire

Dans la majorité des cas, le calcul sur GPU est limité par les accès mémoires. Afin d'atténuer cette problématique, les GPUs disposent de différents types de mémoire (tableau 2.3), les voici par ordre de rapidité :

1. La mémoire privée : Elle regroupe les registres attribués à un thread. Comme son nom l'indique, elle est privée au thread. C'est également et de loin la mémoire la plus rapide et la plus précieuse. En effet, malgré le nombre élevé de registres dont dispose un GPU (par rapport un CPU), il se doit de les partager entre des milliers de threads. Par conséquent, un GPU ne pourra pas lancer beaucoup de threads en parallèle si les noyaux consomment trop de registres.

   Plus grave encore, s'il n'y a pas assez de registres pour lancer un minimum de threads, le compilateur va allouer automatiquement des zones de mémoire plus lentes pour compenser le manque de registres. Ce phénomène diminue dramatiquement les performances et se nomme le « register spilling ».

Malheureusement, l'utilisateur n'a généralement pas directement la main sur l'allocation de la mémoire privée pour chaque thread (car le travail est délégué au compilateur) : le seul moyen d'influencer est de découper les noyaux de manière astucieuse afin de les minimiser. La durée de vie de cette mémoire n'excède pas celle du thread.

2. La mémoire locale : La mémoire locale est partagée par un groupe de thread. Afin d'atteindre le maximum de performance, il faut éviter les « bank conflicts » qui sérialisent les accès mémoires. Les « banks » sont des subdivisions de la mémoire locale. La taille de ces banks dépend du matériel (en général 32 bit). Une exception existe quand tous les threads accèdent exactement à la même bank en lecture. Dans ce cas, la mémoire est « broadcastée » à tous ces threads sans sérialisation.

   Pour augmenter la taille de la mémoire locale à allouer par groupe de tâches, il faut agrandir la taille des groupes au bout d'un moment (dépend du matériel). Mais là encore, il faut choisir avec parcimonie la taille du groupe et la taille de la mémoire locale à attribuer pour chaque groupe car le GPU devra lancer moins de tâches en parallèle pour satisfaire un besoin excessif en mémoire locale, celle-ci n'est pas illimitée. La durée de vie de la mémoire locale se restreint à la durée vie du groupe de tâches.

3. La mémoire constante : Elle fait partie de la mémoire globale et est destinée aux données en lecture seule. Comme pour la mémoire globale et les textures, ses accès utilisent des caches. La durée de vie est lié à la durée vie du buffer de mémoire constante.

4. La mémoire texture : C'est un héritage des API graphiques. Elle fait partie de la mémoire globale mais elle est optimisée pour l'échantillonnage de « texel » (pixel de texture) d'images RGB et RGBA : les interpolations sont accélérées matériellement et les images sont stockées en « Z-curve » (Courbe de Lebesgue ou en code de Morton) pour maximaliser la localité des caches. La durée de vie est dépendante de la durée vie de la texture allouée sur l'hôte.

5. La mémoire globale : Appelée aussi VRAM est la mémoire principale et la plus lente du GPU. Toutes les données de la mémoire hôte transitent par cette mémoire avant de migrer vers les mémoires plus rapides (locale et privée). Pour atteindre le pic de bande passante, il faut que les tâches y accèdent de manière coalescente (accès de manière ordonnée et alignée à des blocs de mémoire continue) afin de maximaliser l'utilisation des caches de mémoire.

	Mémoire privée	Mémoire locale	Mémoire globale
Bande passante	> 20 To $\cdot$ s^{-1}	2 - 15 To $\cdot$ s^{-1}	50 - 900 Go $\cdot$ s^{-1}
Capacité mémoire	63 - 255 registres de 32 bit par thread	32 - 96 ko par groupe de thread	1 - 48 Go
Visibilité	Thread	Groupe de thread	Ensemble des noyaux et hôte
Durée de vie	Thread	Groupe de thread	Buffer
Notes	Le nombre de threads concurrents décroit très rapidement avec le nombre de registres utilisés par thread.	Accès mémoire optimale en évitant les « banks conflicts ». Le nombre de threads concurrents décroient avec la quantité de mémoire utilisée par les groupes de tâches.	Pic de bande passante avec des accès mémoire coalescents.

TABLE 2.3. – Récapitulatif des caractéristiques des différents types principaux de mémoire sur les GPUs contemporains (2019).

3. Parallélisation GPU du rendu d'image à haute dimension spectrale

3.1. Introduction

La simulation d'image à haute dimension spectrale par un algorithme d'illumination globale (ou locale) impose une consommation mémoire et des temps de calcul prohibitifs.

Mettre à profit la puissance de calcul parallèle des GPUs permet de réduire significativement le temps de calcul. La capacité et la bande passante de leur mémoire vive sont cependant limitées. De plus, afin d'exploiter au mieux leurs ressources, les algorithmes doivent être adaptés pour leurs paradigmes de parallélisation SIMD.

Dans ce chapitre, ces problèmes seront étudiés en détail et des solutions seront proposées. L'attention sera portée sur le Path Tracing (section 2.2.7) car c'est l'algorithme d'illumination globale qui convient le plus à notre besoin (un aéronef dans l'atmosphère est une scène extérieure qui nécessite peu de rebonds). Le diagnostic et les solutions sont toutefois transposables à n'importe quel autre algorithme d'illumination locale ou globale.

3.2. Travaux précédents

3.2.1. Le Path Tracing sur GPU

3.2.1.1. L'approche multi-passes de Purcell

L'apparition du pipeline graphique programmable a permis à [Pur+02] de mettre au point la première implémentation GPU du Path Tracing. L'algorithme a dû cependant être séparé en plusieurs passes (figure 3.1) car les GPUs et ce pipeline n'étaient pas encore suffisamment flexibles à l'époque. Les passes étaient exécutées via des fragment shaders et communiquaient entre elles via des buffers (textures) intermédiaires.

En raison du nombre de rebonds différents des chemins tracés, cette approche présente un certain nombre de problèmes, dont les principaux sont les suivants :

— Des charges de calcul inefficaces. La charge de calcul (nombre de chemins traités) diminue au fur et à mesure des rebonds et donc le nombre de threads actifs.
— Des divergences d'exécution importantes. En plus de leur longueur différente, les chemins tracés n'utilisent pas tous les mêmes matériaux et lumières à chaque rebond.

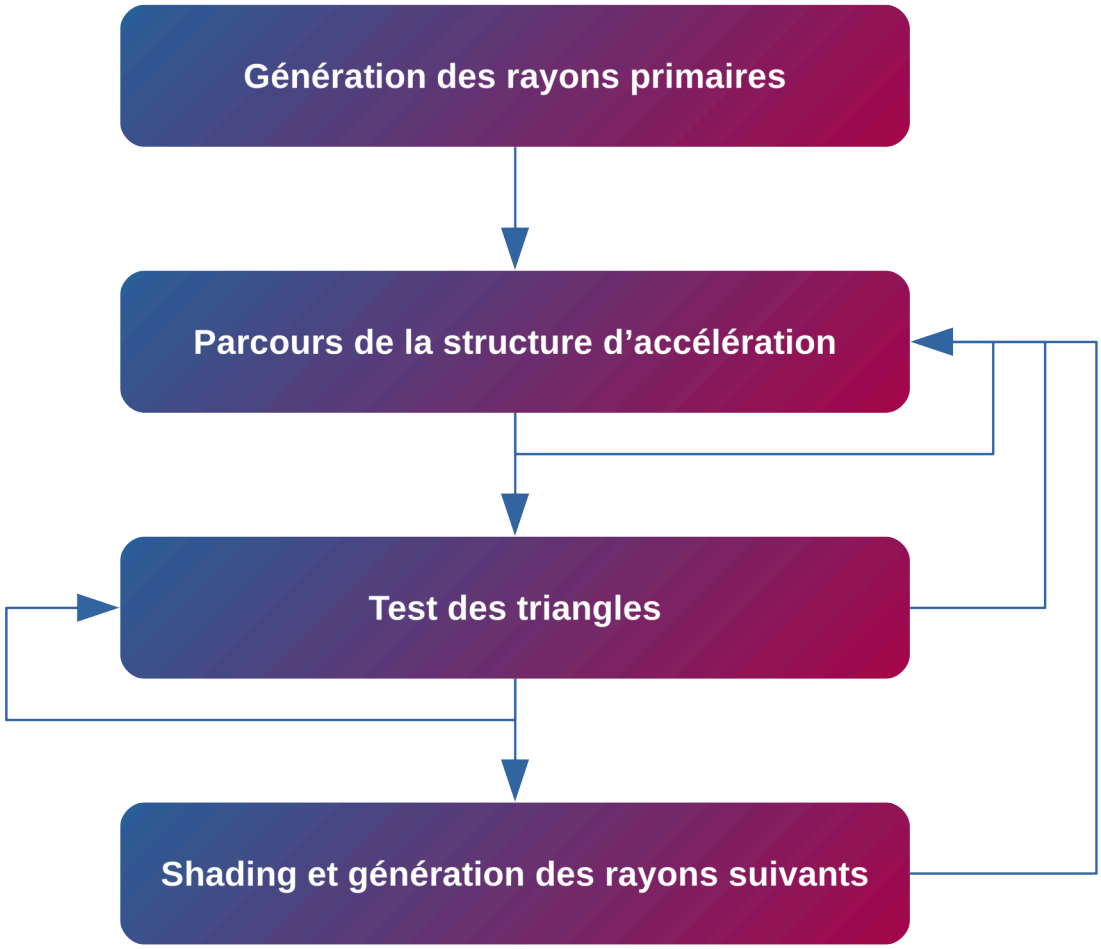


FIGURE 3.1. – Pipeline du Path Tracing de [Pur+02].

3.2.1.2. Le Regenerative Path Tracing

Afin de résoudre le problème des charges de calcul inefficaces, [NHD10] introduisirent le Regenerative Path Tracing. L'idée est d'utiliser un pool de tâches persistants plus petit que l'image désirée où chaque tâche est associée à un chemin d'un pixel. Quand les chemins se terminent, de nouveaux chemins sont attribués aux threads inactifs. Par exemple, sur la figure 3.2, on peut voir que les threads inactifs des chemins 3, 5, 7 et 8 se « régénèrent » en passant aux chemins 9, 10, 11 et 12 suivants.

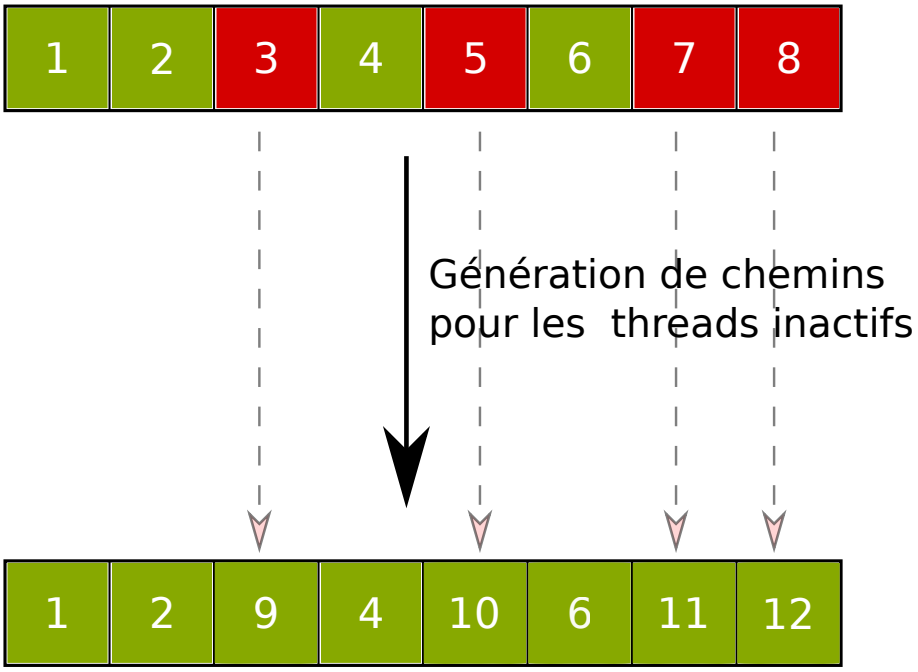


FIGURE 3.2. – Principe du Regenerative Path Tracing de [NHD10] : chaque case correspond à un thread avec un numéro de chemin. La couleur représente l'état d'activité du thread (vert si le chemin n'est pas terminé et rouge sinon).

Cette approche a cependant l'inconvénient d'aggraver le problème de divergences d'exécution car après plusieurs itérations les chemins du pool de tâches sont très incohérents entre eux. C'est-à-dire que les chemins traités par ce pool de tâches empruntent des branches de code très différentes des unes des autres (calcul d'intersection, matériaux, ...) .

3.2.1.3. Le Streaming Path Tracing

Afin de mitiger le problème de divergence, [Ant11] proposa le Streaming Path Tracing. L'idée est d'abord de compacter les chemins actifs pour les regrouper entre eux dans le pool de thread et de ensuite régénérer les threads inactifs pour leurs attribuer de nouveaux chemins à la suite (figure 3.3). Cette étape de compactage permet de regrouper les nouveaux chemins et d'améliorer la cohérence d'exécution lors du premier rebond pour les tests d'intersection.

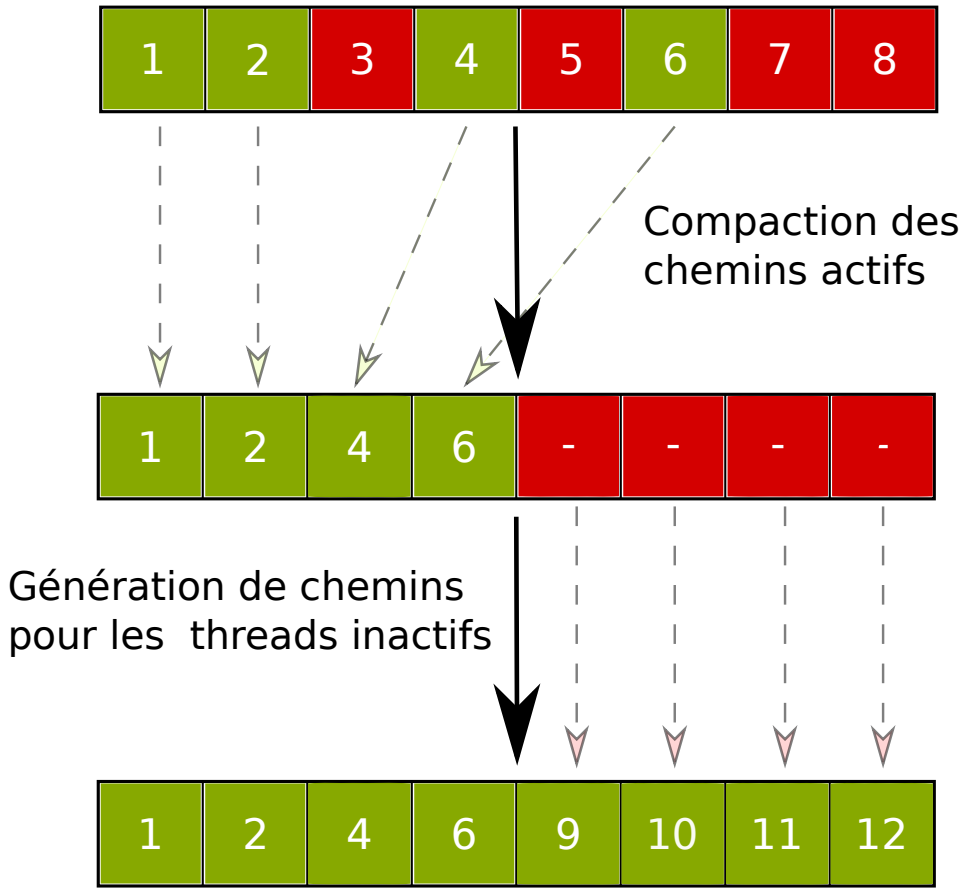


FIGURE 3.3. – Principe du Streaming Path Tracing [Ant11] : chaque case correspond à un thread avec un numéro de chemin. La couleur représente l'état d'activité du thread (vert si le chemin n'est pas terminé et rouge sinon).

3.2.1.4. Le Wavefront Path Tracing

Plus tard, [LKA13] soulignèrent le problème de pression des registres des approches mega noyau. Une approche mega noyau consiste à implémenter l'intégralité d'un algorithme dans un seul noyau. Le noyau est donc très gros et requiert beaucoup de mémoire privée (registre) pour pouvoir s'exécuter.

Cet approche a vu le jour grâce à l'émergence des technologies GPGPU et était très populaire à l'époque car elle nécessitait peu d'adaptations pour passer un code CPU en GPU.

Le problème de cette approche est qu'elle consomme beaucoup de registres notamment quand on a affaire à des scènes avec des matériaux et/ou des lumières complexes. Contrairement aux CPUs, les GPUs doivent partager leurs registres entre des milliers de threads. Une consommation excessive de ces registres provoque des baisses de performance car elle réduit le nombre possible de threads exécutés simultanément par l'ordonnanceur du GPU.

S'il n'y a pas assez de registre, le GPU va utiliser des mémoires plus lentes telles que la mémoire locale ou pire la mémoire globale pour combler ce manque. Ce phénomène se nomme le « register spilling » et dégrade considérablement les performances.

Afin de répondre à ces problèmes, les auteurs proposèrent le Wavefront Path Tracing qui consiste à séparer l'algorithme dans des noyaux différents (figure 3.4). Pour réduire le « register spilling » et améliorer la cohérence d'exécution, un noyau est dédié à chaque matériau. Une étape « Logique » (figure 3.4) permet de router les chemins en fonction de leur état vers les prochaines étapes de l'algorithme.

Cette approche présente des gains de performance pour des scènes contenant des matériaux complexes. Cependant, le routage des chemins vers les bons noyaux et leurs sur-coûts de lancement (hôte, communication entre l'hôte et GPU) peuvent être pénalisants dans le cas de scènes contenant des matériaux très simples (noyau de matériau très court et simple à calculer).

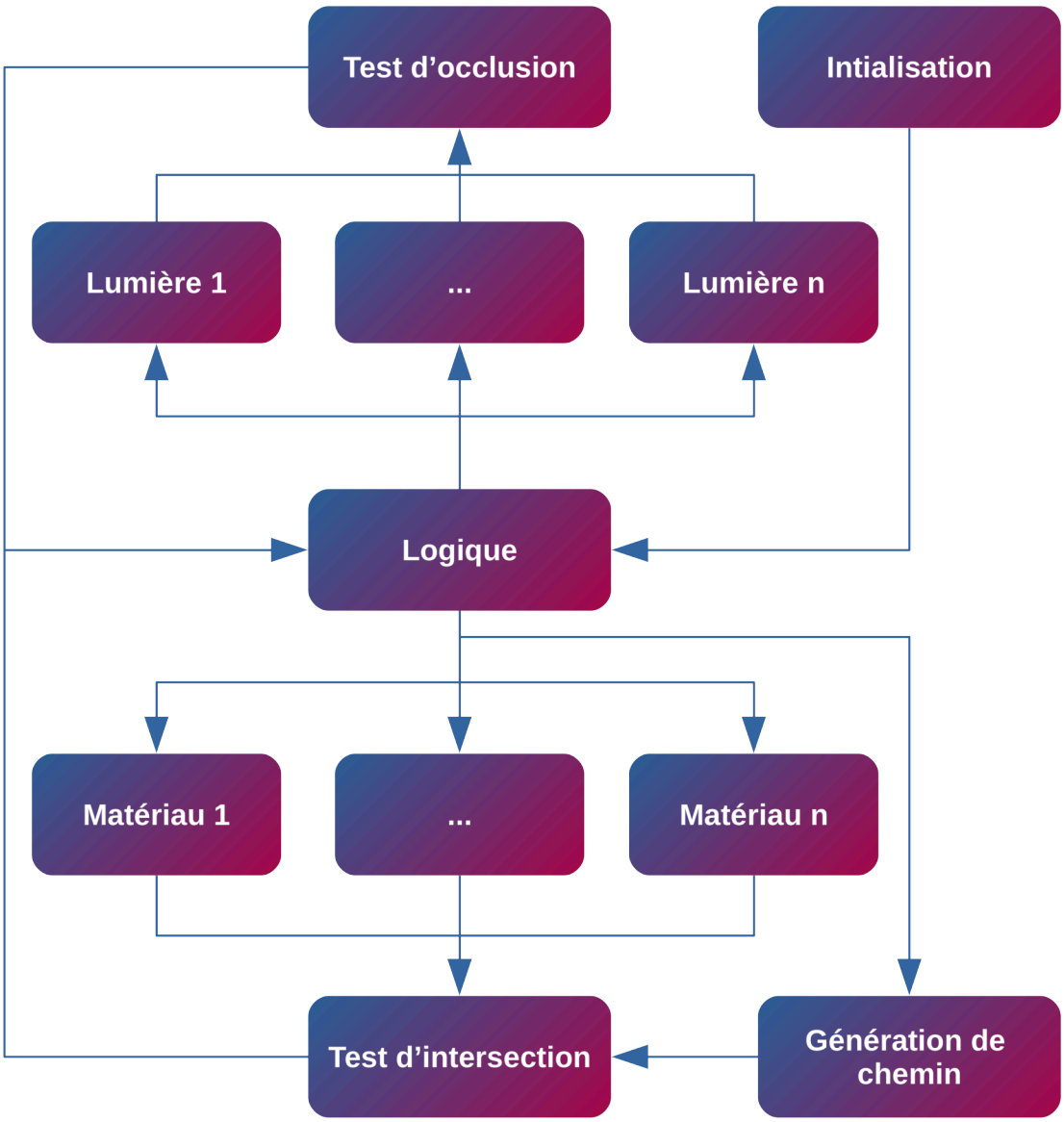


FIGURE 3.4. – Les différents noyaux du Wavefront Path Tracing [LKA13]

3.2.1.5. Discussion sur la meilleure implémentation

[Dav+14] firent un état exhaustif des implémentations GPU du Path Tracing. Ils proposèrent de nouvelles approches et comparèrent les différentes implémentations. Il ressortit de l'étude que les méthodes les plus performantes sont le Streaming Path Tracing (section 3.2.1.3) en multi noyaux et le Regenerative Path Tracing en mega noyau (section 3.2.1.2) sur les GPUs Nvidia. La première méthode a l'avantage d'être portable au niveau des performances entre les différentes génération et constructeurs de carte graphique. Quant au Wavefront Path Tracing (section 3.2.1.4), les auteurs n'ont pas obtenu des résultats probants car leurs scènes de benchmark contenaient des matériaux trop simple.

3.2.2. Rendu d'image spectrale sur GPU

Le rendu d'image spectrale n'est pas quelque chose de commun en informatique graphique. Sur CPU, certains codes académiques proposent de conserver l'image spectrale pour des validations radiométriques et colorimétriques. La dimension spectrale de l'image est cependant généralement assez limitée (80 longueurs d'onde au grand maximum). Sur GPU, de par les contraintes de capacité mémoire, il n'existe pas de moteur de rendu spectral qui propose de restituer une image spectrale à la fin de la simulation.

En optique, l'ONERA a développé un démonstrateur GPU de CRIRA avec OptiX qui peut simuler des images contenant jusqu'à 54 longueurs d'onde.

3.3. Problèmes du rendu d'image spectrale en illumination globale sur GPU

3.3.1. Consommation mémoire des algorithmes d'illumination globale

Le principe de ces algorithmes est de faire rebondir des chemins de lumière dans la scène. Au cours de chaque rebond, un certains nombre de variables sont accumulées. Les chemins de lumière étant construits incrémentalement, il faut donc maintenir en « vie » ces variables tout au long du processus.

Dans le cas du rendu spectral, il faut accumuler et donc stocker un exemplaire de ces variables (figure 3.5) pour chaque échantillon spectral de chaque chemin tracé. Un compromis doit être trouvé entre consommation mémoire et performance : un faible nombre d'échantillons spectraux associés à chaque chemin utilise peu de mémoire mais nécessite de tracer plus de chemin pour avoir le même résultat et donc déplacer le problème vers des temps de calcul d'intersection importants.

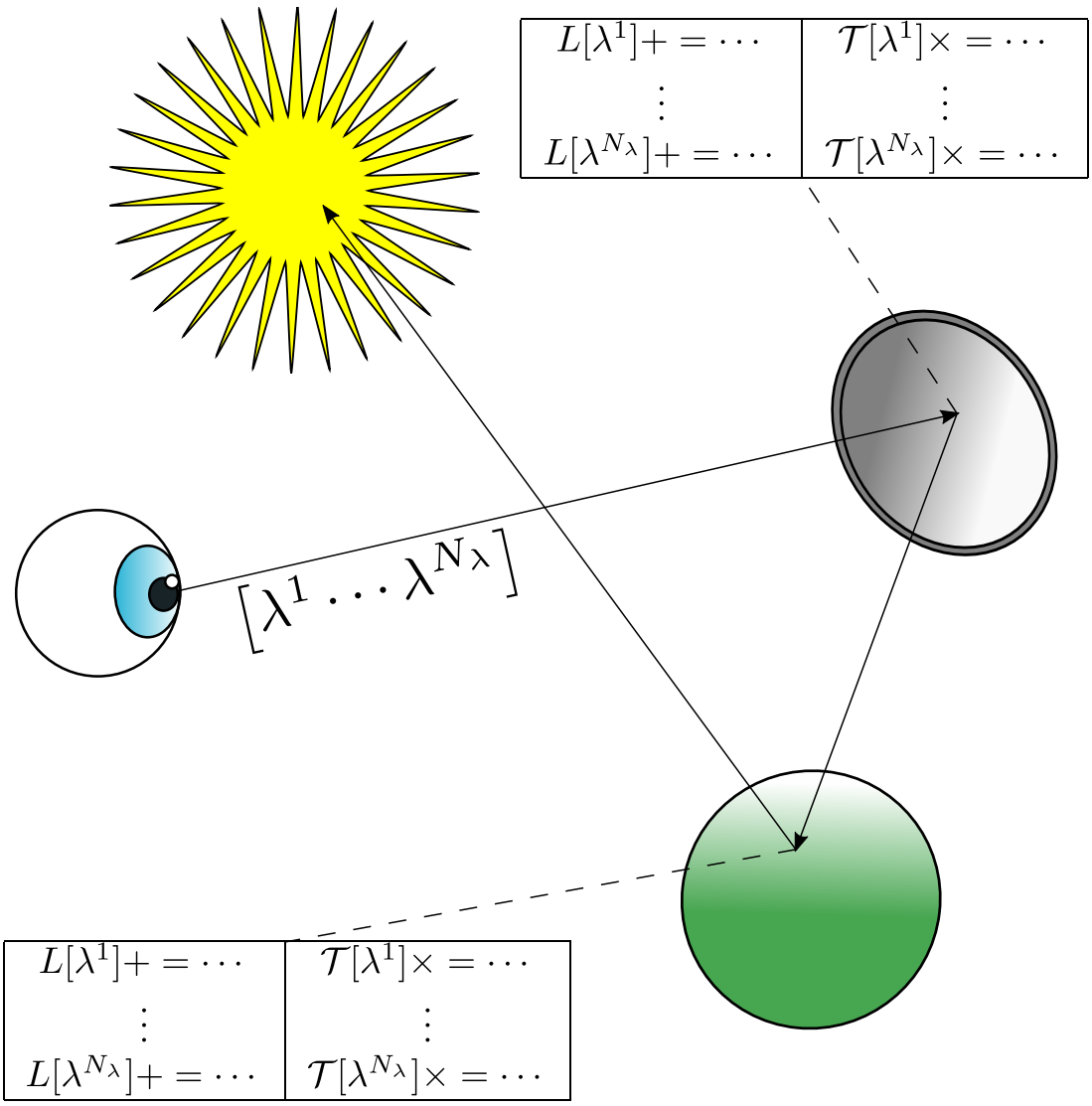


FIGURE 3.5. – À chaque rebond d'un chemin, le Path Tracing doit accumuler au minimum la luminance spectrale L et l'atténuation des rebonds $\mathcal{T}$ de l'ensemble des échantillons spectraux $\left[\lambda^1 \cdots \lambda^{N_\lambda}\right]$.

La consommation mémoire d'un algorithme peut être formulée sous cette forme :

$$C_A = N_{\overline{\omega}} \times C_{\overline{\omega}} \tag{3.1}$$

En prenant en compte les éléments ci-dessus, la consommation mémoire d'un chemin pour le rendu spectral peut être traduite par cette équation :

$$C_{\overline{\omega}} = N_\lambda \times C_\lambda + C_c \tag{3.2}$$

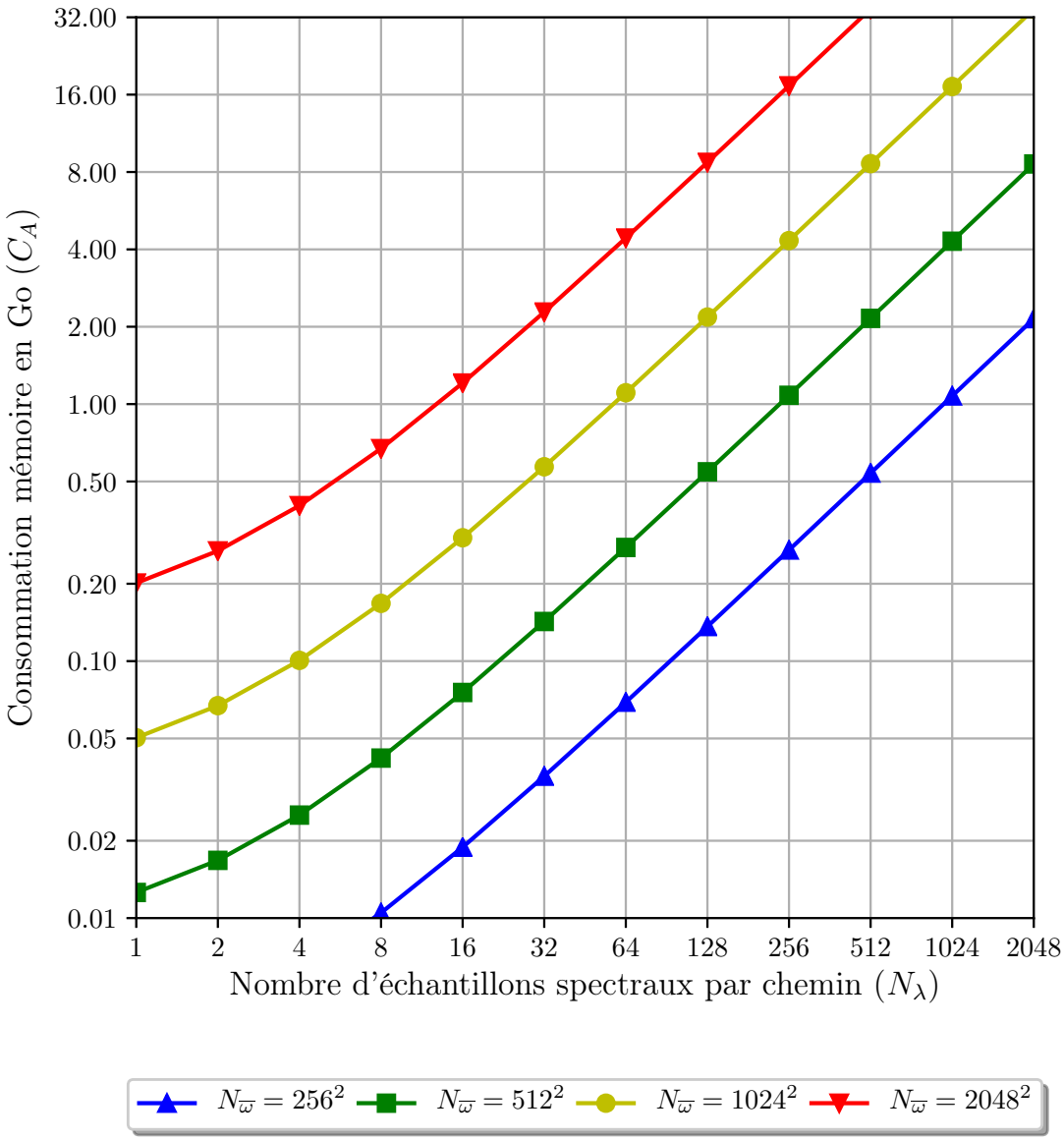


FIGURE 3.6. – Consommation mémoire du Path Tracing ($C_c = 32$ octets et $C_\lambda = 16$ octets) pour différent N_ϖ.

Dans nos expérimentations, nous avons observé, que si $N_\varpi < 1024^2$ chemins, les performances baissent significativement. Au delà, les performances s'améliorent sensiblement. Cela est dû au fait qu'un GPU a besoin d'une certaine charge de calcul pour amortir les latences mémoires (accès intra-noyau et transfert entre l'hôte et le GPU) et le surcoût des lancements des noyaux.

Or en étudiant la consommation mémoire du Path Tracing (figure 3.6) pour $N_\varpi = 1024^2$ chemins, on constate que la consommation mémoire pour un C_λ élevé va poser problème à la grande majorité des cartes graphiques (par exemple, il faut

environ 8,6 Go de mémoire pour une dimension spectrale de $C_\lambda = 512$ juste pour faire fonctionner l'algorithme).

Par conséquent, l'emploi de ces algorithmes sur GPU pour un rendu à haute dimension spectrale nécessite de réduire le niveau de parallélisme ($N_{\overline{\omega}}$) et donc les performances.

3.3.2. La latence mémoire

Faute de place, l'état d'un chemin (de taille $C_{\overline{\omega}}$) ne peut être stocké dans les mémoires rapides d'un GPU (mémoire privée ou locale, voir la section 2.3.2.3).

La mémoire globale du GPU est la seule option disponible si N_λ et $N_{\overline{\omega}}$ ne sont pas trop grands (voir la figure 3.6). À chaque rebond, il faudra donc mettre à jour ces états ce qui aura pour effet de réduire les performances puisque c'est la mémoire la plus lente du GPU (10 à 100 fois plus lente que la mémoire locale ou privée d'après la tableau 2.3).

3.3.3. La consommation mémoire de la sortie spectrale

Une image spectrale est une image à trois dimensions. Sa consommation mémoire peut se traduire par cette équation :

$$C_I = N_w \times N_h \times N_\Lambda \times C_r \tag{3.3}$$

Lorsque l'on augmente une ou plusieurs de ces dimensions, on peut voir sur la figure 3.7 que la consommation mémoire augmente très rapidement.

En résolution multi spectrale ($3 \leq N_\Lambda \leq 10$), l'image est très légère, le problème ne se pose donc pas. Par contre, en hyper spectral ($100 \leq N_\Lambda \leq 300$) et d'autant plus en ultra spectral ($N_\Lambda \geq 1000$), la consommation mémoire de l'image est problématique quand la résolution spatiale est élevée. Par exemple, une image avec une résolution spatiale full HD (1920×1080 pixels) avec $N_\Lambda = 1000$ consommera environs 8,3 Go de mémoire. Toutefois, dans notre cas, la résolution spatiale des capteurs de détection d'aéronef dans l'infrarouge est en général assez faible (au mieux en 512×512 pixels en l'état actuel). Avec cette limite de résolution spatiale, la consommation mémoire n'excédera pas 1 Go en simple précision en résolution hyper spectrale. Le stockage d'une telle image est donc possible pour des GPUs « grand public » (entre 4 et 8 Go). Cependant pour une simulation d'une telle image, il faut également qu'il reste de la place pour la scène et l'algorithme de rendu (voir la figure 3.6).

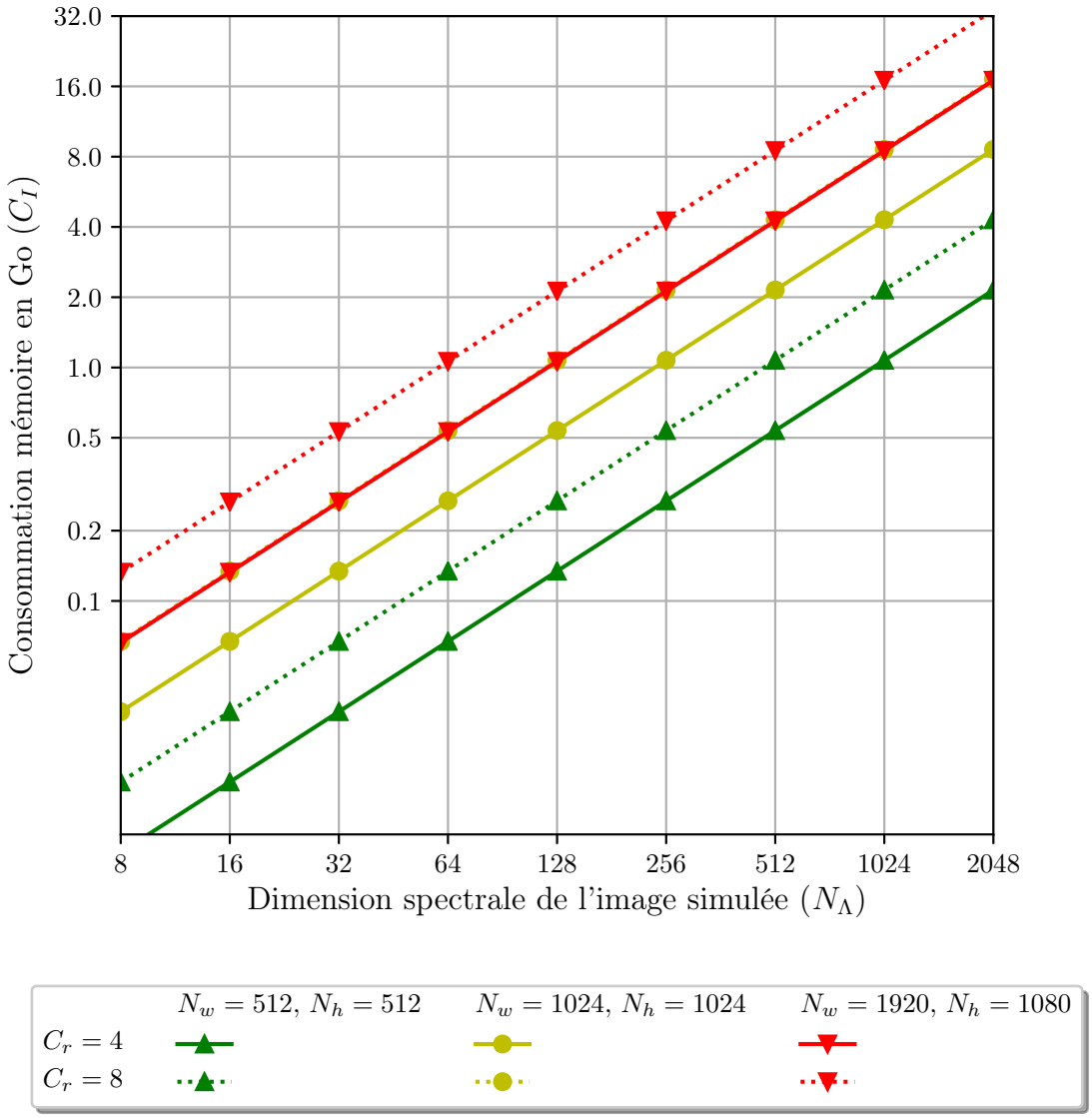


FIGURE 3.7. – Consommation mémoire d'une image spectrale en simple ($C_r = 4$) et double ($C_r = 8$) précisions pour différentes résolutions spatiales (N_w et N_h).

3.4. Deferred Path Evaluation Path Tracing

3.4.1. Algorithme

Afin de surmonter les problèmes de consommation de l'algorithme de la section 3.3.1 et de latence mémoire de la section 3.3.2, nous proposons de dissocier la génération des chemins de leurs évaluations spectrales.

Le problème des algorithmes d'illumination globale est qu'ils construisent incrémentalement les chemins. Il faut donc évaluer tous les échantillons spectraux en même temps, puisque les informations des rebonds précédents ne sont pas conservées. L'idée de l'algorithme proposé est donc de :

1. Construire entièrement les chemins et sauvegarder le strict nécessaire à chaque rebond via une implémentation GPU du Streaming Path tracing en multi noyaux (voir la section 3.2.1.3).
2. Évaluer les échantillons spectraux de chaque chemin par groupe de longueurs d'onde (voir la algorithme 6).

Étant donné que l'on conserve entièrement les chemins, il est possible d'évaluer les échantillons spectraux de chaque chemin par groupe de longueurs d'onde. Cela permet de contrôler la consommation mémoire des échantillons spectraux.

De cette manière, les états des échantillons spectraux peuvent être stockés dans les mémoires rapides des GPUs. La mémoire privée étant limitée et précieuse, il est donc préférable d'utiliser la mémoire locale.

La partie critique de l'algorithme est l'étape d'évaluation des chemins (algorithme 6). Nous avons donc cherché à étudier un schéma de parallélisation efficace.

Algorithm 6: Évaluation d'un groupe de longueurs d'onde β pour un chemin $\overline{\omega}$.

Mémoire locale: L, $\mathcal{T}$
Mémoire globale: $\mathbf{V}$, I^{λ}, ϑ, L_e, L_d, τ

Function $\mathcal{F}(\beta, \overline{\omega})$ **:**
  // Initialisation pour l'évaluation des longueurs d'onde de β.
  foreach $\lambda \in \beta$ **do**
    $L[\lambda] \leftarrow 0$
    $\mathcal{T}[\lambda] \leftarrow 1$
  end

  // Évaluation du groupe de longueur d'onde.
  foreach $v \in \mathbf{V}[\overline{\omega}]$ **do**
    if $v.x = x_{\varnothing}$ **then**
      foreach $\lambda \in \beta$ **do**
        $L[\lambda] \leftarrow L[\lambda] + \mathcal{T}[\lambda] \times v.w_{L_i} \times \vartheta(\lambda, v)$
      end
    else
      foreach $\lambda \in \beta$ **do**
        $L[\lambda] \leftarrow L[\lambda] + \mathcal{T}[\lambda] \times v.w_{L_i} \times L_e[v](\lambda, v)$
        $L[\lambda] \leftarrow L[\lambda] + \mathcal{T}[\lambda] \times v.w_{L_d} \times L_d[v](\lambda, v)$
        $\mathcal{T}[\lambda] \leftarrow \mathcal{T}[\lambda] \times \frac{\tau[v](\lambda, v.\overrightarrow{\omega_i}, v.\overrightarrow{\omega_o}, \dots)}{v.\rho_{\tau}}$
      end
    end
  end

  foreach $\lambda \in \beta$ **do**
    // Raffinement de l'image spectrale I^{λ} avec les nouveaux échantillons spectraux calculés
  end
return

3.4.2. Schémas de parallélisation de l'étape d'évaluation des chemins

3.4.2.1. Schéma de parallélisation naïf avec un chemin par thread

Le schéma de parallélisation le plus simple de cette étape (figure 3.8) pour un work-group est de laisser chaque thread traiter indépendamment son propre chemin.

Une couleur différente a été attribuée à chaque colonne (thread). Elle indique les instructions nécessaires pour évaluer un chemin. Il y a plusieurs appels de $\mathcal{F}(\cdots, \cdots)$ pour chaque chemin car les longueurs d'onde sont évaluées par groupe pour contrôler la consommation mémoire.

Par exemple, $\mathcal{F}(\beta_{N_{\overline{\omega}}}^{N_\beta}, \overline{\omega}_{N_{\overline{\omega}}})$ est l'évaluation du chemin $\overline{\omega}$ d'indice $N_{\overline{\omega}}$ pour son groupe $\beta_{N_{\overline{\omega}}}$ de longueurs d'onde d'indice N_β. Les couleurs utilisées permettent de visualiser plus facilement l'ensemble des instructions nécessaires pour évaluer un chemin en entier.

Les performances de cette approche sont mauvaises quand la dimension spectrale de la simulation est élevée car les accès mémoire de l'image spectrale depuis la mémoire globale ne sont pas coalescents.

De plus, la différence de matériaux à chaque rebond entre chaque chemin entraîne une faible cohérence du code (ce qui est très mauvais pour les unités SIMD des GPUs) et un faible taux d'utilisation des caches mémoire.

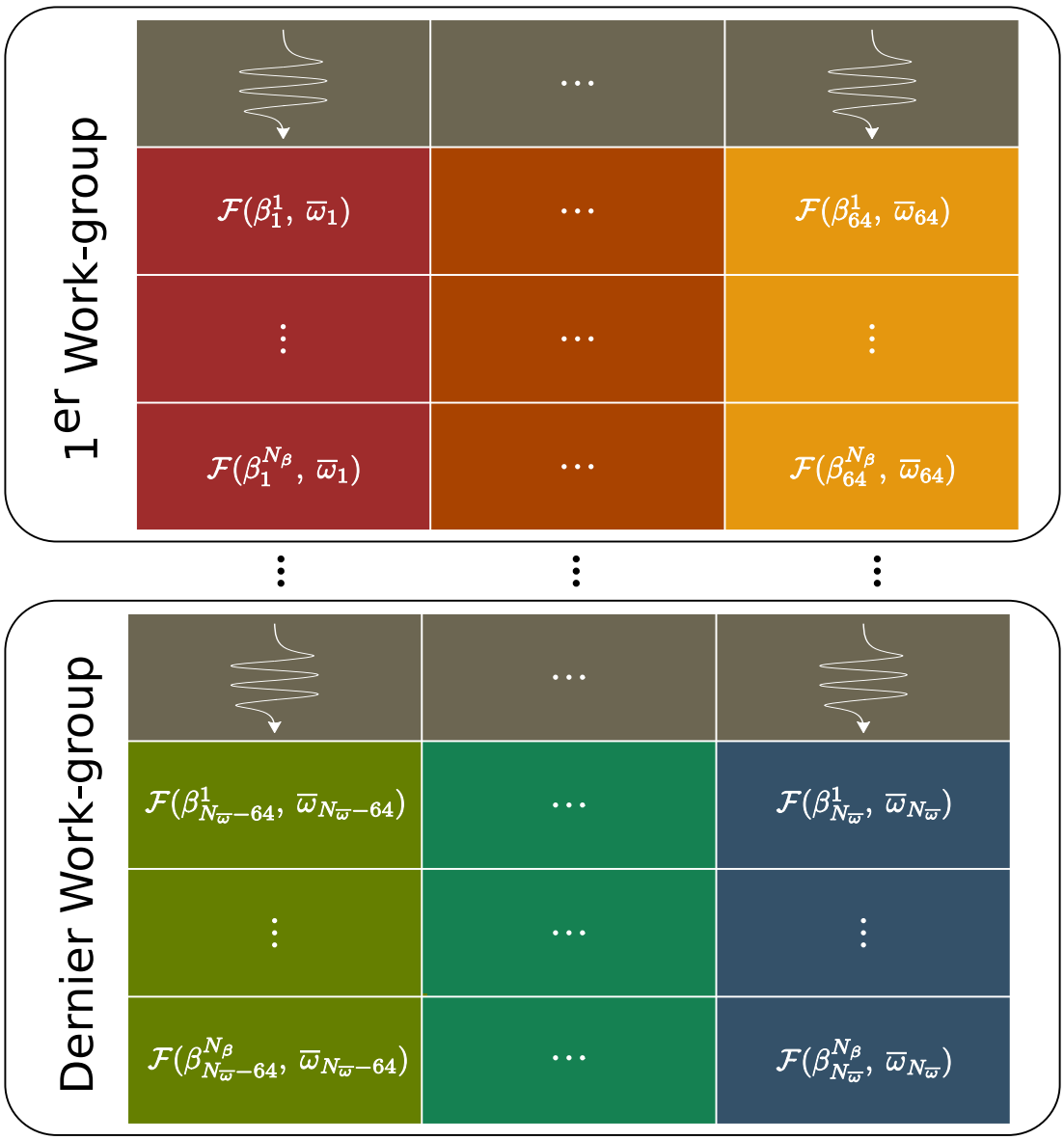


FIGURE 3.8. – Schéma de parallélisation avec un chemin par thread pour des work-groups de 64 threads.

3.4.2.2. Schéma de parallélisation spectral

Pour résoudre ces problèmes, nous proposons un schéma de parallélisation spectral (figure 3.9) de l'étape d'évaluation des chemins pour les groupes de longueurs d'onde. Au lieu de laisser un thread évaluer son chemin indépendamment, le groupe de threads va se synchroniser pour évaluer un même chemin.

Chaque thread de ce groupe va donc évaluer un groupe de longueurs d'onde (différent) du même chemin en même temps. C'est pour cela qu'il y a une seule couleur par ligne sur la figure 3.9.

Ce schéma permet donc d'améliorer la cohérence du code exécuté en parallèle par le groupe de thread. D'autre part, les accès mémoire à l'image spectrale peuvent être coalescents et les caches mémoires sont réutilisés plus souvent. Cela permet donc d'augmenter significativement les performances quand la dimension spectrale de la simulation est élevée.

Toutefois, si il n'y a pas assez de groupe de longueurs d'onde à paralléliser, certain threads vont être inactifs. Dans ce cas, le premier schéma de parallélisation (figure 3.8) sera préférable.

Nous n'avons pas exploré pour le moment un schéma hybride entre les deux approches car cela complexifie l'accumulation (opération de réduction) des XYZ (pour afficher une image à l'utilisateur) dans le work-group pour le traitement des cas particuliers (nombre impair de longueurs d'onde par exemple).

Sur les GPUs AMD, en dessous de 16 longueurs d'onde, il est préférable d'utiliser le premier schéma (figure 3.8). C'est sans doute dû à la largeur des unités SIMD du matériel qui est généralement de 16 chez ce constructeur.

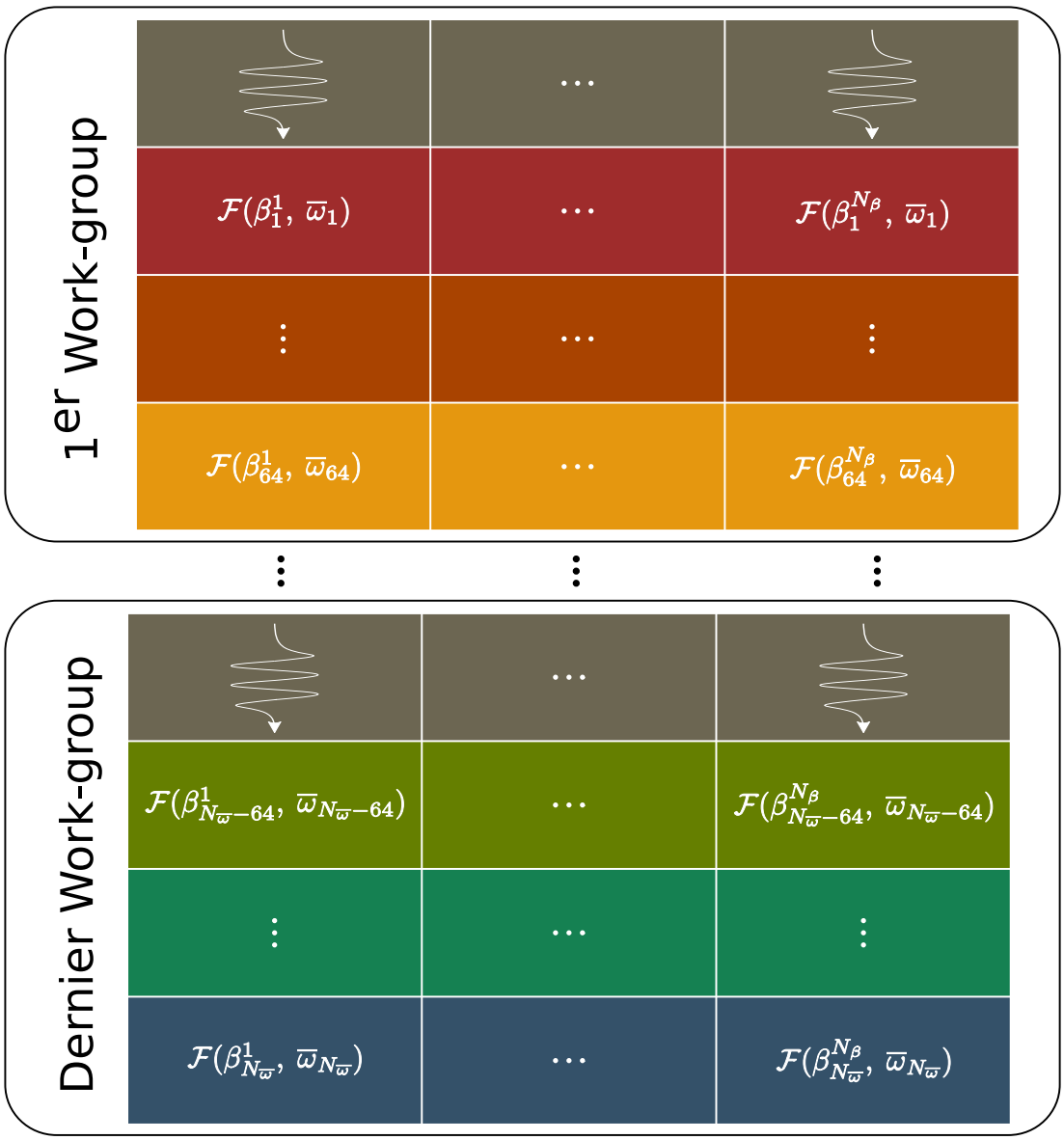


FIGURE 3.9. – Schéma de parallélisation spectral pour des work-groups de 64 threads.

3.4.3. Consommation mémoire

Avec cette méthode, la consommation mémoire d'un chemin est dépendante de son nombre de rebonds maximal N_v, et peut s'exprimer sous cette forme :

$$C_{\overline{\omega}} = N_v \times C_v + C_c \tag{3.4}$$

Généralement, les GPUs ne supportent pas l'allocation dynamique de mémoire intra-noyau. C'est à dire que l'allocation de la mémoire (buffers, ...) se fait sur l'hôte en amont des lancements des noyaux. Et quand la fonctionnalité est disponible, elle entraîne des dégradations importantes de performance. Il faut donc fixer à l'avance un nombre maximal et global de sommets à allouer pour tous les chemins depuis le code de l'hôte (code CPU).

Pour l'instant, le poids de nos sommets a pu être contenu dans 64 octets (voir le liste 1 pour les détails). Avec la complexification du code de simulation (prise en compte des textures, volume, rayon différentiel, ...) dans le futur, il est raisonnable de tabler sur un poids maximal de 128 octets pour un sommet.

Si on regarde la figure 3.10 pour $N_{\overline{\omega}} = 1024^2$ chemins avec $N_v = 16$ rebonds, l'algorithme va consommer environ 1,1 Go de mémoire pour $C_c = 64$ octets et environ 2,2 Go pour $C_c = 128$ octets quel que soit le nombre d'échantillons spectraux par chemin.

Les scènes aériennes d'aéronef ne nécessitent généralement pas beaucoup de rebonds (entre 4 et 16), la consommation mémoire (figure 3.10) reste donc raisonnable par rapport aux algorithmes traditionnels (figure 3.6) dans tous les cas. Et même si on a besoin de plus de rebonds, la consommation mémoire reste inférieure à celle des approches précédentes quand le nombre de longueurs d'onde par chemin augmente.

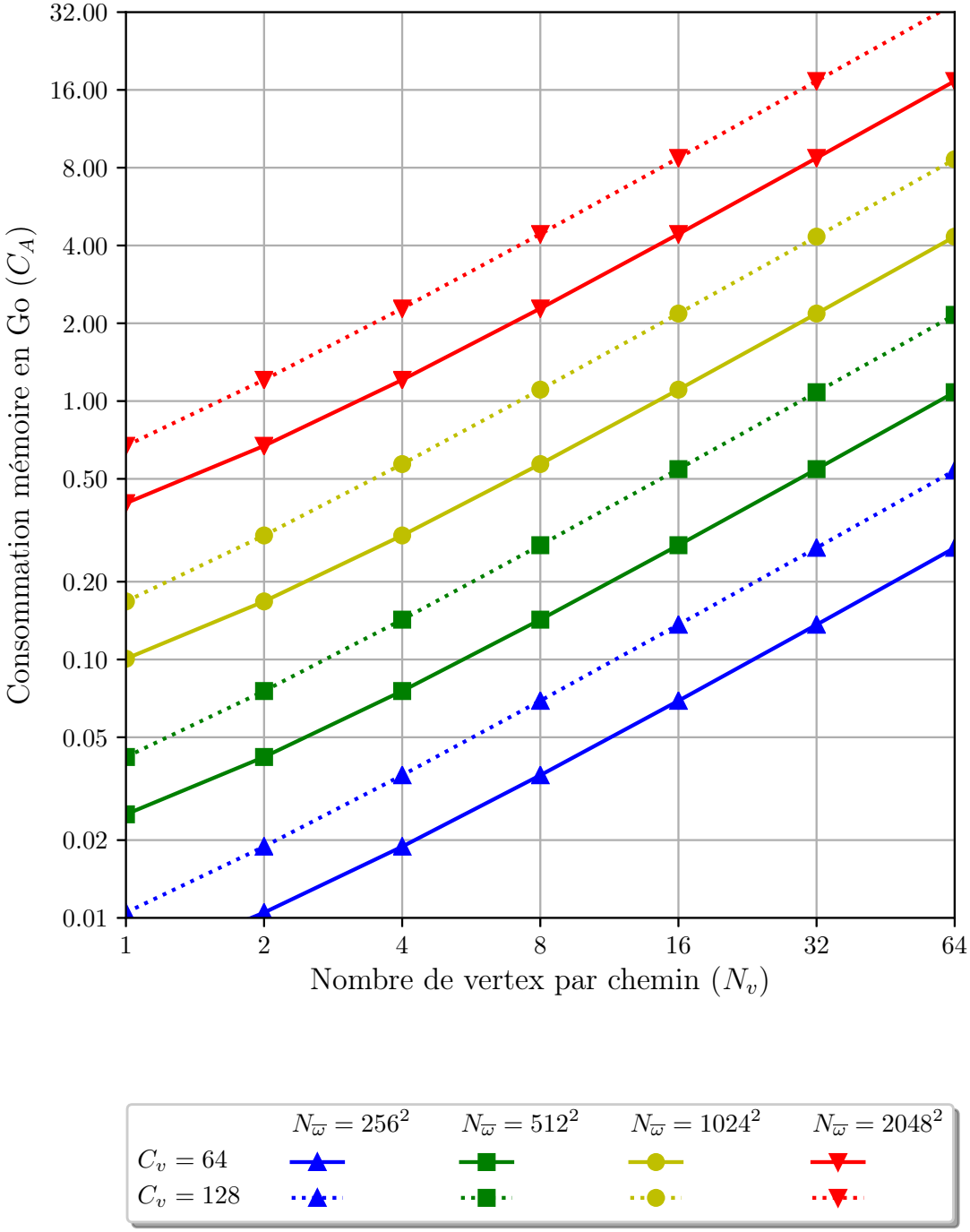


FIGURE 3.10. – Consommation mémoire du Deferred Path Evaluation Path Tracing avec $C_c = 32$ octets pour différents C_v et N_ϖ.

```
typedef std::array<float, 4> Float4;

////////////////////////////////////////////////////////////////////////
// Sommet: 64 octets dont 4 octets de padding
////////////////////////////////////////////////////////////////////////
struct Vertex {
    // - Flags (occlusion, ...): 4 octets.
    // - Direction (azimut, élévation) incidente
    //   (espace BSDF): 8 octets.
    // - Poids Li (PDF, coefficient MIS et cosinus): 4 octets.
    Float4 flags_wi_LeWeight;

    // - Indice de la lumière échantillonnée: 4 octets.
    // - Direction (azimut, élévation) vers la lumière
    //   (espace BSDF) : 8 octets.
    // - padding: 4 octets
    Float4 lightId_woToLight_padding;

    // - Direction (azimut, élévation) vers la lumière
    //   (espace monde): 8 octets.
    // - Distance: 4 octets.
    // - Poids Ld (PDF, coefficient MIS et cosinus): 4 octets.
    Float4 lightSample;

    // - Indice matériau: 4 octets.
    // - Direction (azimut, élévation) de wo
    //   (espace BSDF): 8 octets.
    // - PDF (et cosinus): 4 octets.
    Float4 bsdfSample;
};

////////////////////////////////////////////////////////////////////////
// PathState: 32 octets + ses Nv sommets.
////////////////////////////////////////////////////////////////////////
struct PathState {
    // - État d'activité du chemin: 4 octets.
    // - Longueur du chemin: 4 octets.
    // - Indice du pixel: 4 octets.
    // - Indice du rayon: 4 octets.
    Float4 idle_depth_pixelIndice_rayIndice;

    // - PRNG: 4 octets.
    // - Échantillon spectral (voir la section 4.3.2.3): 4 octets.
    // - Dernière PDF pour les coefficients du MIS: 4 octets.
    // - Indice de la longueur d'onde sélectionnée
    // lors d'une dispersion (voir la section 4.3.2.2): 4 octets.
    Float4 rngState_spectralSample_lastPDF_wavelengthId;
};
```

Listing 1 – Structure de donnée pour le DPEPT : les données sont paquetées par vecteur de 16 octets pour optimiser les accès mémoire.

3.5. Benchmark

Pour comparer le DPEPT et le PT, un benchmark (figure 3.11 et figure 3.13) consistant à mesurer les performances brutes (nombre de chemins par seconde) et l'interactivité (nombre d'images par seconde) a été effectué. Les mesures ont été prises pour différents $N_{\overline{\omega}}$ et N_λ. Le benchmark a été réalisé avec les scènes suivantes :

— Cornell box : une scène simple d'intérieur avec un nombre de rebonds maximal fixé à 5.

— Conference : une scène d'intérieur de difficulté moyenne avec un nombre de rebonds maximal fixé à 8.

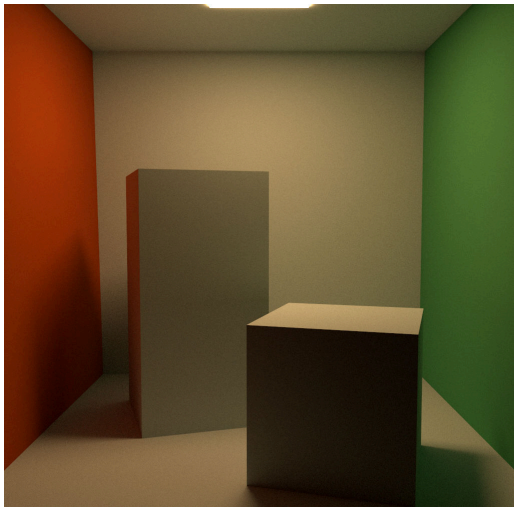
— Aircraft : une scène aérienne typiquement utilisée pour dimensionner des capteurs de détection. La scène contient une carte d'environnement spectrale générée par MATISSE ([Sim+06]). Le nombre de rebonds maximal a été fixé à 5.

— Statues : une scène extérieure un peu plus complexe malgré les apparences car elle contient un objet réfringent. Elle contient également une carte d'environnement spectrale générée à partir d'une image RGB HDR. Le nombre de rebonds maximal a été fixé à 16.

Mis à part le modèle de Suzanne de Blender (scène : Statues) et l'avion de ONERA (scène : Aircraft), les autres géométries proviennent du dépôt de [McG17].

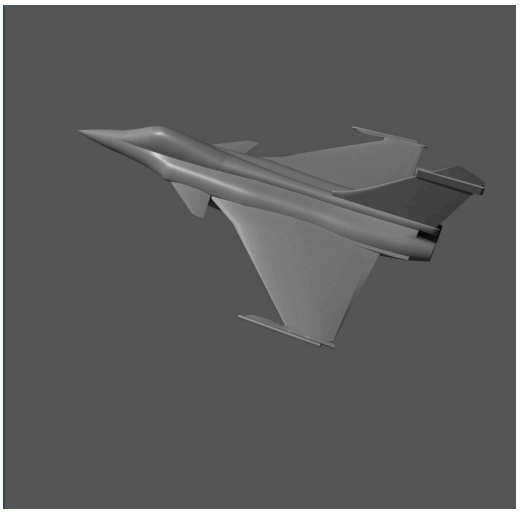
Avec $N_{\overline{\omega}} = 1024^2$ et $N_v = 16$ sur une Fury X (GPU AMD avec 4 Go de VRAM) et en considérant que la scène consommera au maximum 500 Mo (ce qui est assez peu), une image en résolution spatiale full HD (1920×1080 pixels) limiterait N_λ à 288 pour le DPEPT (figure 3.10) et 138 pour le Path Tracing (figure 3.6). De plus, la résolution spatiale des capteurs de détection dans l'infrarouge est plus faible que dans le visible. Par conséquent, nous avons décidé de fixer la résolution spatiale de l'image à 512×512 pixels pour le benchmark.

Cornell Box.



Conference.

Aircraft.



Statues.

FIGURE 3.11. – Scènes du benchmark.

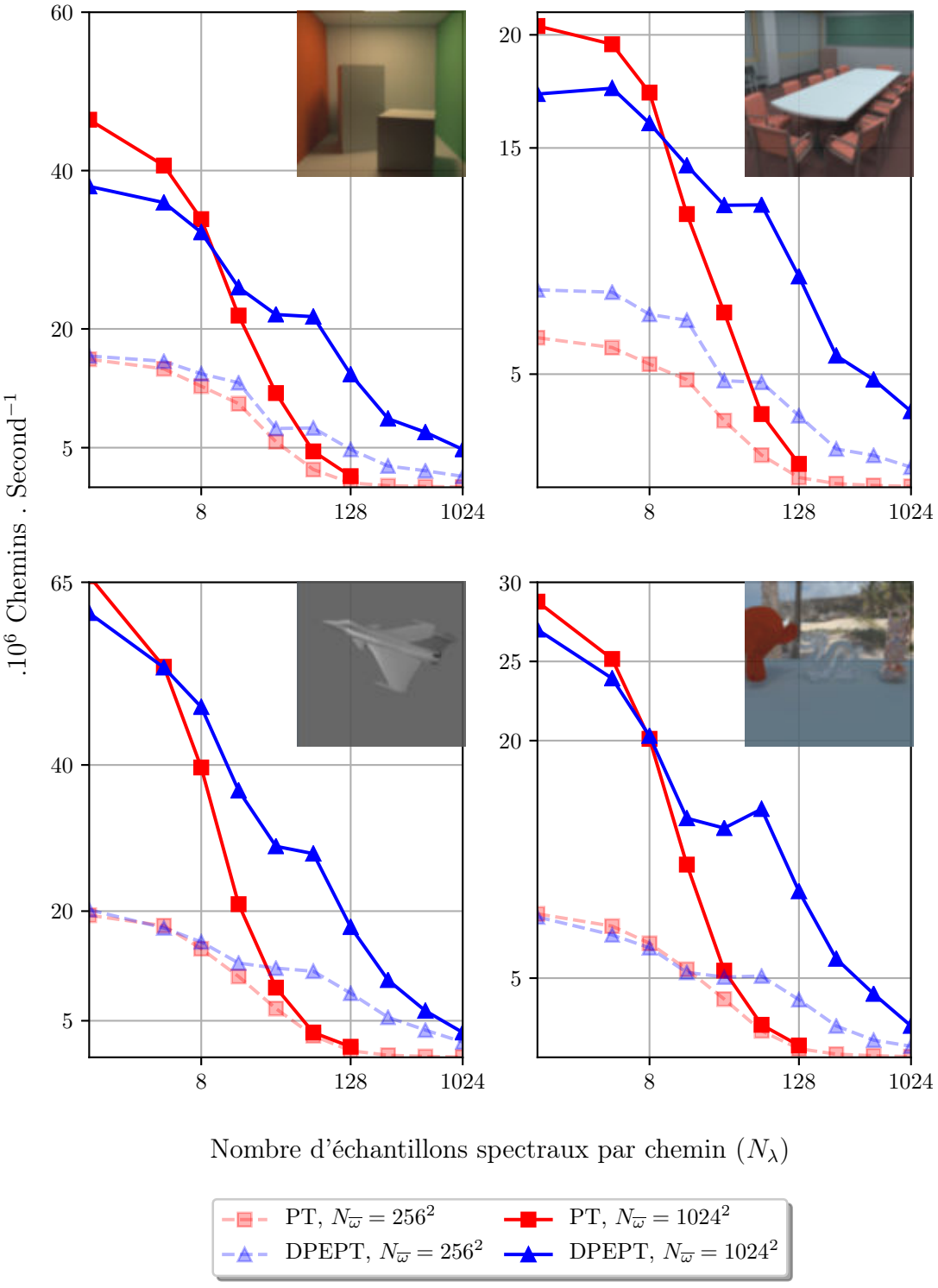


FIGURE 3.12. – Performance brute pour les scènes de la figure 3.11 avec une AMD Fury X (VRAM : 4 Go). L'échelle des abscisses est en logarithme base 2.

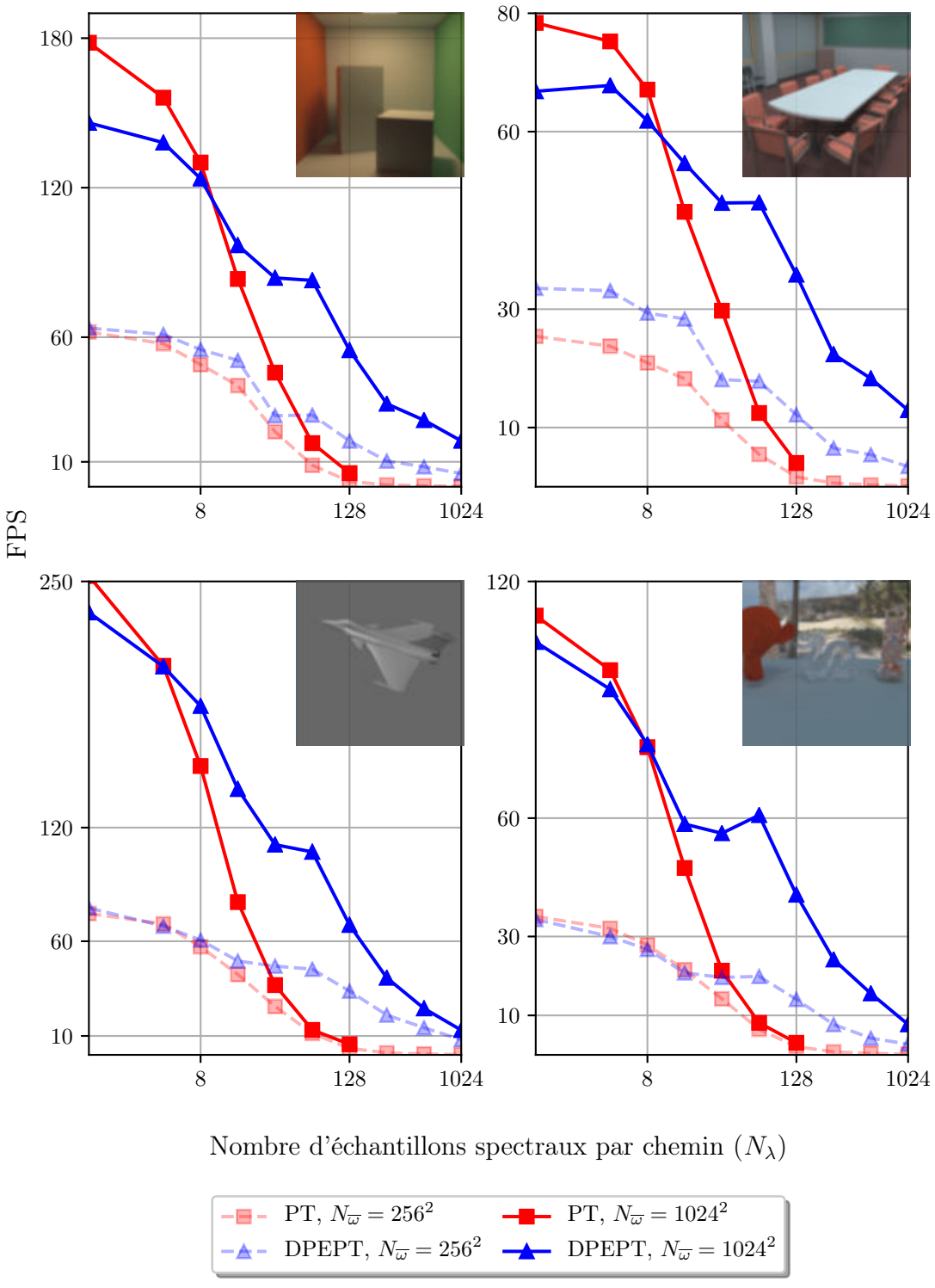


FIGURE 3.13. – Interactivité pour les scènes de la figure 3.11 avec une AMD Fury X (VRAM : 4 Go). L'échelle des abscisses est en logarithme base 2.

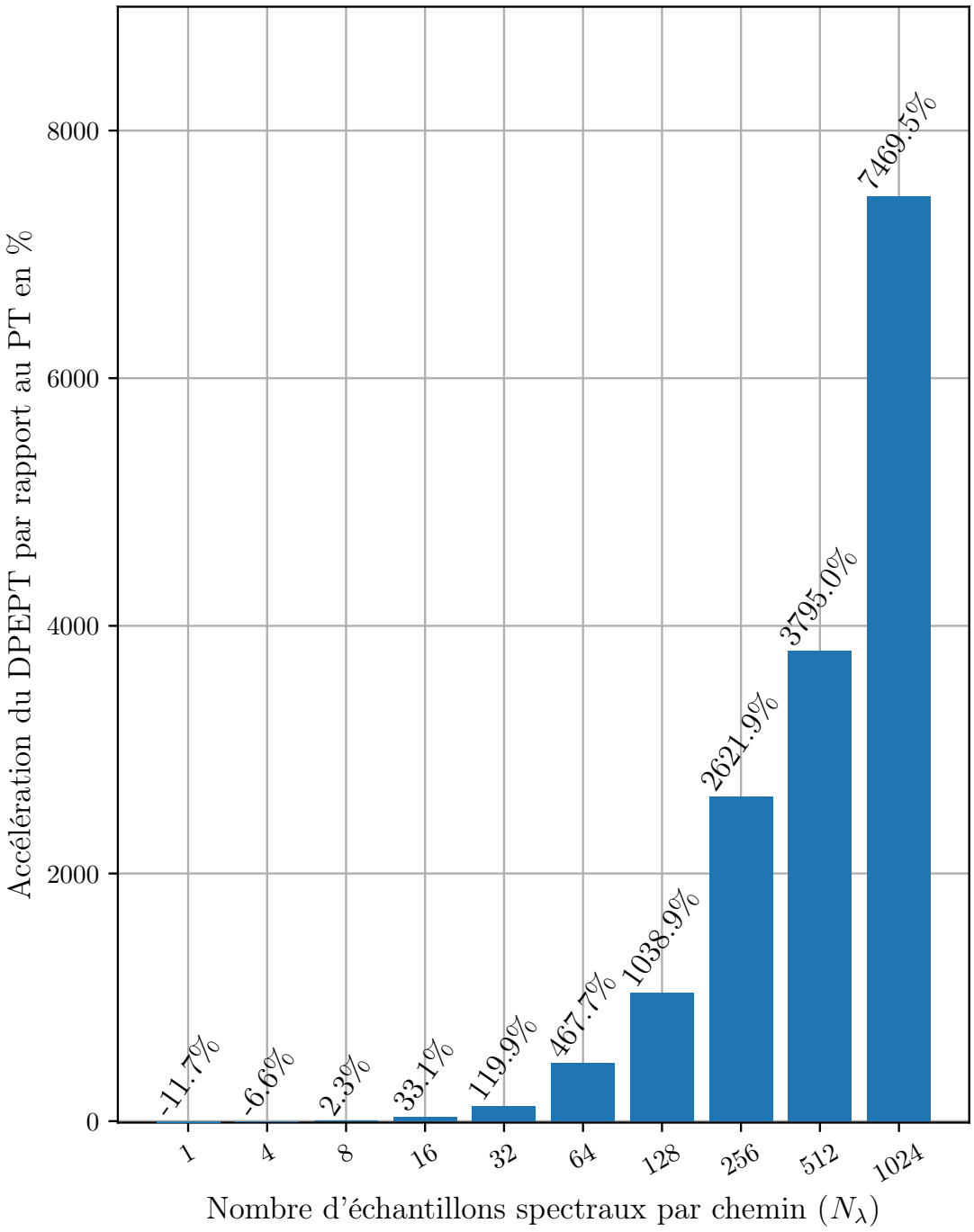


FIGURE 3.14. – Performance relative moyenne du DPEPT par rapport au PT pour le benchmark (figure 3.12).

Les figure 3.12 et figure 3.14 montrent que les performances de notre méthode sont supérieures au Path Tracing dès que le nombre $N_\lambda > 8$, avec des gains considérables en hyper spectral.

Contrairement au Path Tracing, la consommation mémoire du DPEPT n'est pas dépendante de N_λ. Par conséquent on peut simuler un plus grand nombre de chemins en parallèle. Par exemple, si on veut faire le rendu d'une image avec une dimension spectrale de 512, le Path Tracing ne pourra pas tracer 1024^2 chemins en même temps, alors que notre méthode peut facilement atteindre ce nombre de chemins et sera ainsi environ 38 fois plus rapide en moyenne (figure 3.14) que le Path Tracing.

Les résultats du benchmark nous montrent que notre méthode permet un rendu d'image hyper spectrale (N_Λ entre 100 et 300 longueurs d'onde) interactif (entre 10 et 20 images par seconde) pour des scènes de complexité géométrique intermédiaire. L'algorithme proposé est donc particulièrement adapté pour le rendu d'image spectrale à haute résolution spectrale. La limite n'est donc plus au niveau de la consommation mémoire de l'algorithme mais au niveau de la taille de l'image spectrale et du temps de calcul qui peut être encore assez long malgré nos contributions si la dimension spectrale ou spatiale de l'image est élevée.

3.6. Conclusion

Sur GPU, outre le problème de temps de calcul qui est mitigé (par la puissance de calcul des GPU), le rendu d'image à grande dimension spectrale pose trois problèmes majeurs :

— la consommation mémoire de l'algorithme de rendu.
— la latence mémoire de la mémoire globale utilisée pour stocker et mettre à jour les états des échantillons spectraux des chemins.
— l'empreinte mémoire de l'image rendu.

Pour répondre aux problématiques de consommation mémoire du Path Tracing et de latence mémoire, le Deferred Path Evaluation Path Tracing (DPEPT) est proposé dans la section 3.4.1. L'algorithme consiste à découpler la phase de construction du chemin de son évaluation. La consommation n'est ainsi plus dépendante de la résolution spectrale de la simulation mais du nombre maximal de rebonds des chemins.

Reporter l'évaluation des échantillons spectraux des chemins permet d'utiliser la mémoire locale au lieu de la mémoire globale car on peut contrôler leurs consommations mémoire en les évaluant par groupe de longueur d'onde. L'utilisation de la mémoire locale permet ainsi de diminuer la latence mémoire.

Afin d'améliorer la méthode, la parallélisation de la seconde étape a été étudiée et un schéma de parallélisation spectral (figure 3.9) est proposé pour améliorer les performances des simulations à haute dimension spectrale.

Le DPEPT couplé au schéma de parallélisation spectral surpasse de loin le Path Tracing usuel (figure 3.5) quand la dimension spectrale de la simulation augmente. Les contributions exposées dans ce chapitre permettent le rendu d'image multi, hyper et voire même ultra spectral. Le temps interactif en hyper spectral peut d'ailleurs être atteint dans notre cas (rendu d'image à faible résolution spatiale pour des scènes de complexité géométrique intermédiaire). Le problème d'empreinte mémoire de l'image spectrale n'est toutefois pas encore résolu ce qui peut encore limiter les dimensions de l'image si il n'y a pas assez de place dans la mémoire globale du GPU.

4. Rendu stochastique d'image spectrale

4.1. Introduction

La méthode de rendu d'image spectrale actuellement utilisée par une chaîne de dimensionnement de capteur impose des simulations à très haute dimension spectrale. En effet, pour être précis, la dimension spectrale de la simulation doit correspondre à la résolution spectrale des données. Si ce n'est pas le cas, on court le risque de ne pas prendre en compte dans le calcul des longueurs d'onde importantes de la scène.

Dans notre cas, les scènes aériennes d'aéronef sont des scènes triviales géométriquement mais complexes spectralement. Les spectres d'émission et d'absorption des gaz contenus dans la plume mais aussi dans l'atmosphère (voir figure 1.1) sont effectivement très complexes et nécessitent une résolution spectrale très fine de l'ordre de $1\,\mathrm{cm}^{-1}$ pour être correctement représentés. Dans les bandes VNIR et SWIR, la résolution en terme de nombre d'onde augmente très rapidement, la résolution en longueur d'onde est donc généralement préférée. Par exemple, pour atteindre $1\,\mathrm{cm}^{-1}$ dans la bande MWIR (voir tableau 4.1), il faut simuler 1334 longueurs d'onde par pixel de l'image.

Ces contraintes imposées par le besoin de précision peuvent en dépit de nos travaux sur la parallélisation du rendu d'image spectrale sur GPU (chapitre 3), poser des problèmes de temps de calculs et de consommation mémoire.

Afin de répondre à ces problèmes de dimension spectrale élevée, une nouvelle méthode de rendu d'image spectrale et des chaînes de dimensionnements de capteur sont proposées dans ce chapitre.

Bande Spectrale			VNIR 0,38 - 1 µm	SWIR 1 - 2,5 µm	MWIR 3 - 5 µm	LWIR 8 - 12 µm
Résolution	**Nombre d'onde**	$100\,cm^{-1}$	164	60	14	5
		$50\,cm^{-1}$	327	120	27	9
		$25\,cm^{-1}$	653	240	54	17
		$10\,cm^{-1}$	1632	600	134	42
		$5\,cm^{-1}$	3264	1200	267	84
		$1\,cm^{-1}$	16 316	6000	1334	417
	Longueur d'onde	100 nm	7	15	20	40
		50 nm	13	30	40	80
		25 nm	25	60	80	160
		10 nm	62	150	200	400
		5 nm	124	300	400	800
		1 nm	620	1500	2000	4000

TABLE 4.1. – Dimension spectrale d'une simulation dans les différentes bandes spectrales visées à différentes résolutions.

4.2. État d'art

Pour simuler correctement le transport de la lumière, le rendu spectral est nécessaire. Il y a actuellement deux grandes familles de rendu spectral :
— les méthodes de rendu spectrales déterministes.
— les méthodes de rendu spectrales stochastiques.

4.2.1. Les méthodes de rendu spectrales déterministes

Le principe de ces méthodes est de simuler une image de luminances spectrales de dimension N_Λ d'un ensemble de longueur d'onde $\{\lambda^1, \ldots, \lambda^{N_\Lambda}\}$ fixées à l'avance. On peut décrire leurs fonctionnements avec cette équation :

$$I^\lambda(i,j,k) = \int_{\overline{\Omega}_{ij}} W_{ij}(\overline{\omega})\ \overline{f}(\lambda^k,\ \overline{\omega})\ \mathrm{d}\overline{\omega} \tag{4.1}$$

Un des problèmes majeur de ces méthodes est l'aliasing spectral. Ce biais s'exacerbe quand la dimension spectrale de la simulation est faible et que des longueurs d'onde importantes de la scène ne sont pas prises en compte dans le calcul.

Afin d'éviter de perdre trop d'information spectrale dans le cas où la précision radiométrique est recherchée, il est usuel de simuler un très grand nombre de longueur d'onde prédéfinies espacées à intervalle régulier pour couvrir la bande spectrale visée. Cette approche amène toutefois la simulation à une consommation mémoire et des temps de calculs excessifs.

4.2.1.1. Les méthodes de rendu spectrales déterministes en informatique graphique

En informatique graphique, dans la plupart des cas, la sortie d'une méthode de rendu spectrale déterministe est convertie en couleur en l'intégrant avec les courbes de réponses spectrales de l'œil humain (courbe CIE XYZ). Pour des raisons de temps calculs et comme la luminance spectrale d'un pixel $I^\lambda(i,j,k)$ est une fonction tabulée en une dimension, on calcule généralement cette intégration via une somme de Riemann (équation (2.14)).

Dans certain cas, il peut y avoir des besoins pour la validation colorimétrique via la simulation d'un appareil photo. Il faut donc appliquer directement les courbes de réponses spectrales d'un appareil photo pour simuler sa couleur RGB en sortie :

$$I^{RGB}(i,\ j) = \sum_{k=1}^{N_\Lambda} \begin{bmatrix} W^R(\lambda^k) \\ W^G(\lambda^k) \\ W^B(\lambda^k) \end{bmatrix} I^\lambda(i,\ j,\ k)\ (\lambda^{k+1} - \lambda^k) \tag{4.2}$$

Dans tous les cas, le nombre de longueurs d'onde simulées peut influencer grandement sur la précision de la simulation et de l'intégration en couleur. D'ailleurs,

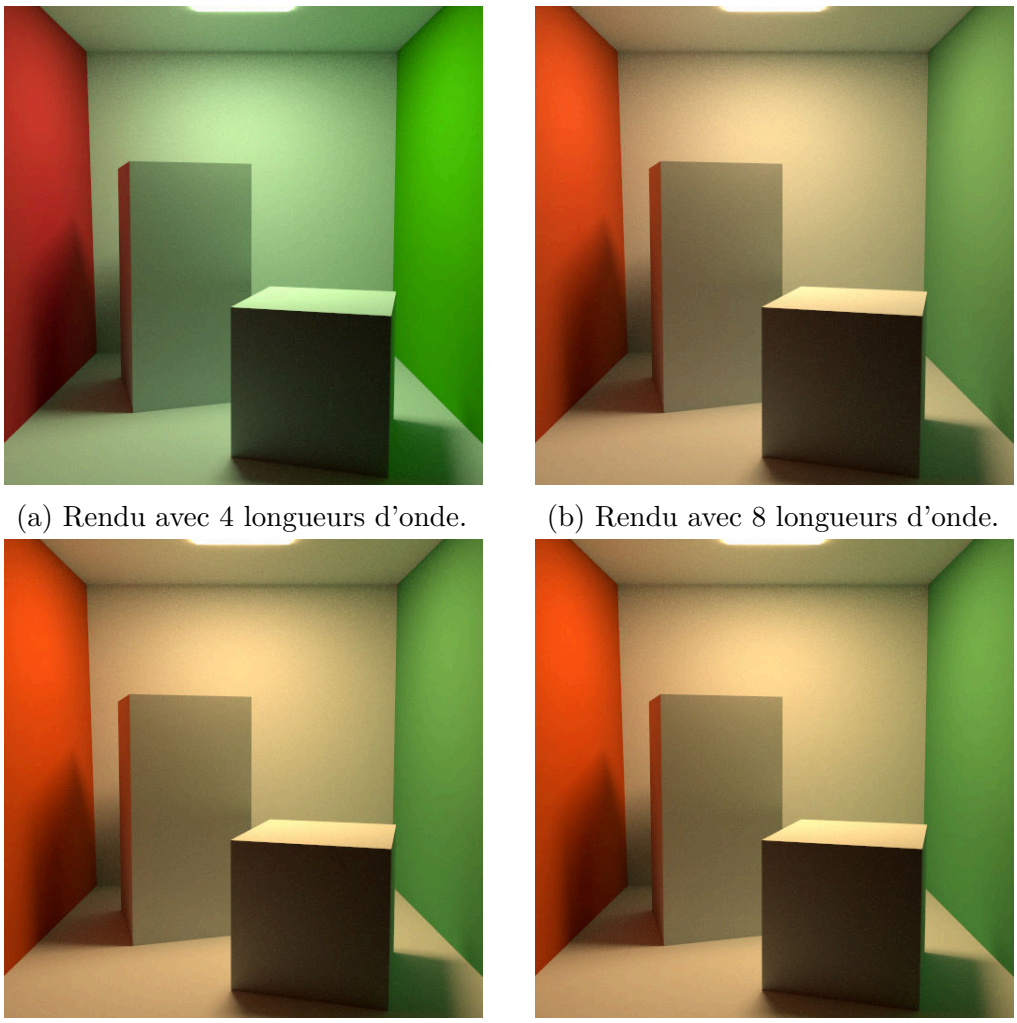
cette imprécision peut être visible à l'œil nu comme sur la figure 4.1 en dessous de 32 longueurs d'ondes.

Pour éviter ce problème, il est conseillé dans la littérature de simuler au minimum 32 longueurs d'onde espacées à intervalle régulier dans le spectre visible. Simuler un tel nombre de longueurs d'onde peut poser déjà pas mal de problèmes en terme de temps de calculs et de consommation mémoire. Et dans certains cas pathologiques c'est très loin d'être suffisant comme par exemple les scènes contenant des lampes à spectres à raie comme sur la figure 4.2 où en dessous de 200 longueurs d'ondes, les différences sont visible à l'œil nu.

Pour mitiger ce problème, [ZCB98], [WTP00] et [CW05] proposèrent de choisir précautionneusement un ensemble de longueur d'onde optimale en fonction de chaque scène simulée. Trouver un ensemble de longueur d'onde optimale n'est toutefois pas une tâche facile. De plus, même avec un ensemble optimal, cela ne résout pas le problème des simulations sur de larges bandes (dans l'infrarouge par exemple) pour des scènes complexes spectralement. Dans ces cas, un très grand nombre de longueur d'onde sera toujours requis.

4.2.1.2. En optique

En dépit de ces problèmes de biais spectral, de consommation mémoire et de temps calculs, ces méthodes sont toujours utilisées en optique car ce sont les seules qui permettent de conserver la sortie spectrale à la fin de la simulation. D'ailleurs, l'ONERA utilise ces méthodes pour dimensionner et étudier des capteurs via CRIRA (voir son pipeline sur la figure 4.3).



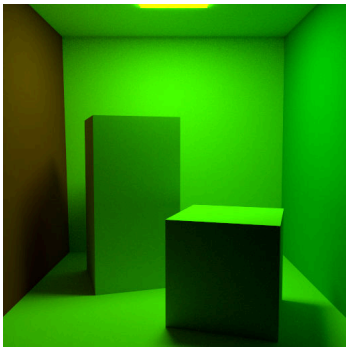
(a) Rendu avec 4 longueurs d'onde.

(b) Rendu avec 8 longueurs d'onde.

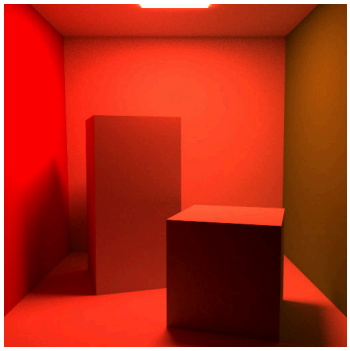
(c) Rendu avec 32 longueurs d'onde.

(d) Référence (400 longueurs d'onde).

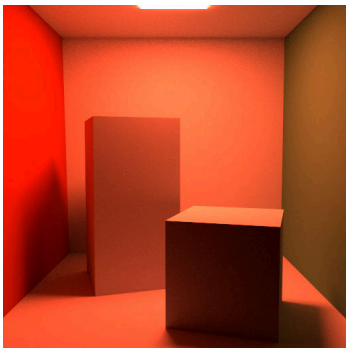
FIGURE 4.1. – Erreur induite par les méthodes de rendu spectrale déterministe pour une scène spectralement triviale (données obtenues avec des conversions de couleur RGB via la méthode de [Smi99]).



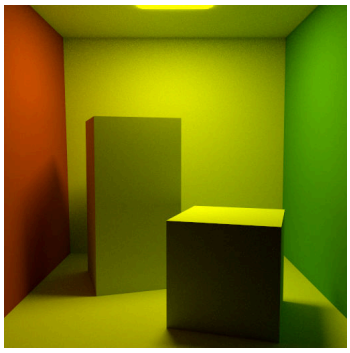
(a) 4 longueurs d'onde.



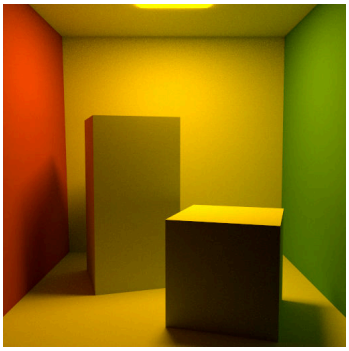
(b) 8 longueurs d'onde.



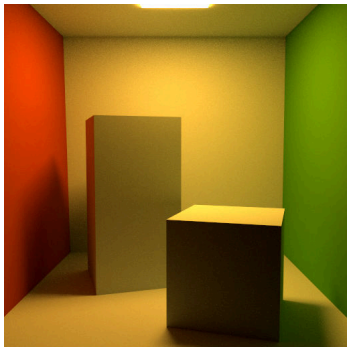
(c) 16 longueurs d'onde.



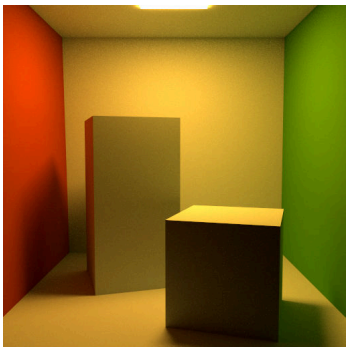
(d) 32 longueurs d'onde.



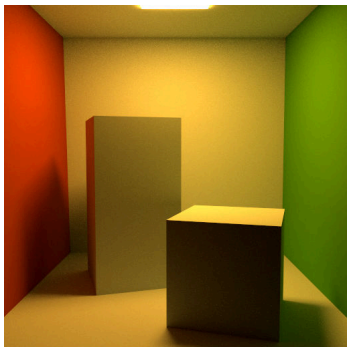
(e) 80 longueurs d'onde.



(f) 200 longueurs d'onde.



(g) 400 longueurs d'onde.



(h) 800 longueurs d'onde (Référence).

FIGURE 4.2. – Erreur induite par les méthodes de rendu spectrale déterministe pour une scène spectralement compliquée (contient une LFC).

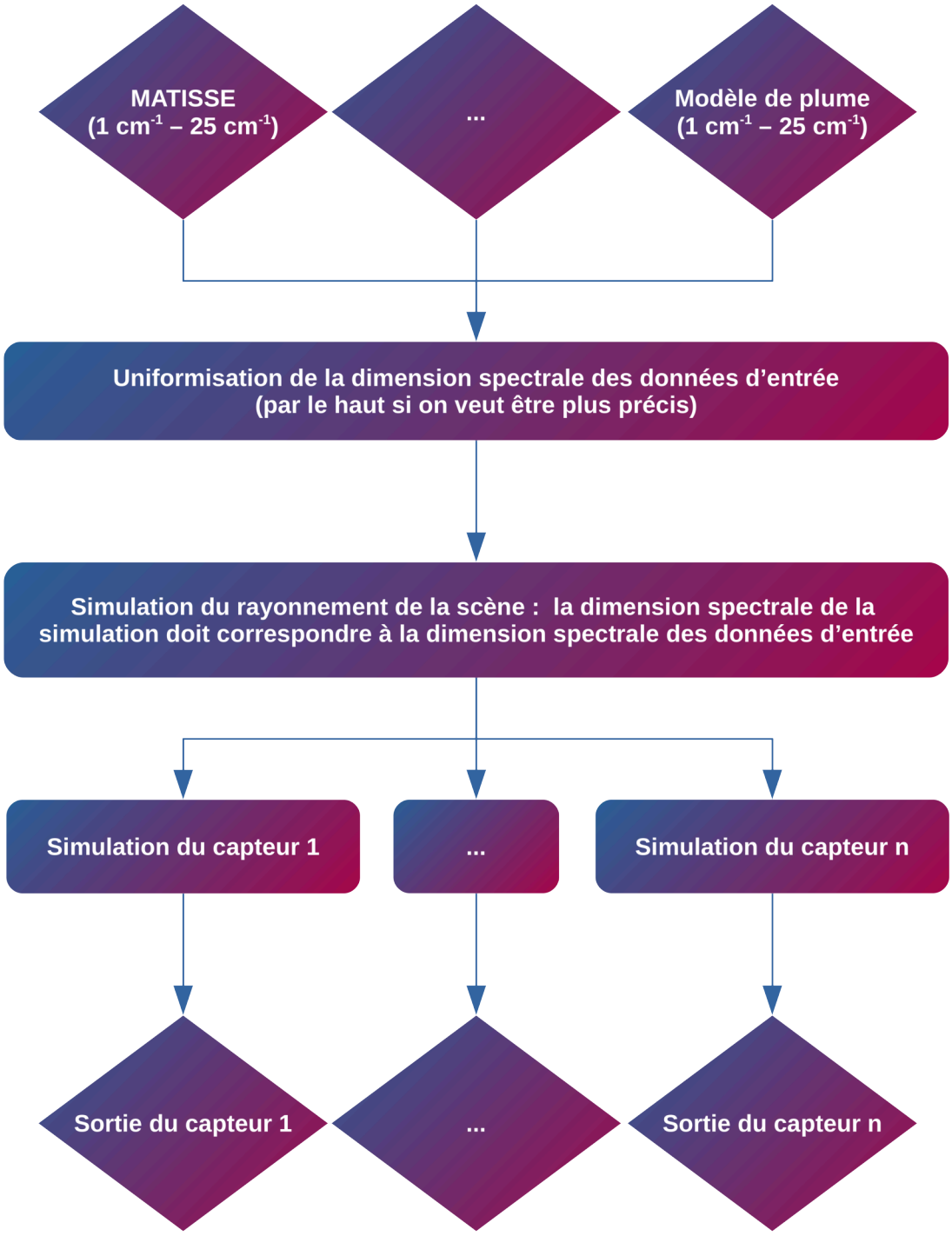


FIGURE 4.3. – Pipeline de dimensionnement de capteurs avec CRIRA.

4.2.2. Les méthodes de rendu spectrales stochastiques

Afin de contourner les contraintes ci-dessus, la communauté de l'informatique graphique a préconisé une solution stochastique au problème. Au lieu de choisir un ensemble de longueurs d'onde prédéfinies, une dimension spectrale est ajoutée à l'équation intégrale par chemin de Veach (équation (2.23)) pour calculer directement le tristimulus XYZ. Ajouter une dimension à cette intégrale ne pose aucun problème car elle est calculée avec la méthode de Monte-Carlo dont la vitesse de convergence n'est pas dépendante du nombre de dimensions de l'intégrale. Ces méthodes peuvent être décrites par cette équation :

$$I^{XYZ}(i,\ j) = \int_{\overline{\Omega}_{ij}} \int_{380\,\mathrm{nm}}^{780\,\mathrm{nm}} \begin{bmatrix} \overline{x}(\lambda) \\ \overline{y}(\lambda) \\ \overline{z}(\lambda) \end{bmatrix} f_{ij}(\lambda,\ \overline{\omega})\ \mathrm{d}\lambda\ \mathrm{d}\overline{\omega} \tag{4.3}$$

La méthode la plus intuitive de cette famille est de simuler une longueur d'onde par chemin de lumière. Cette approche est toutefois inefficiente car un chemin de lumière est dans la plupart des cas valide pour n'importe quelle longueur d'onde.

Pour accélérer la convergence, [EM99] proposèrent donc de réutiliser un chemin de lumière pour un petit groupe de longueur d'onde. Ce groupe de longueur d'onde était échantillonné préférentiellement en prenant en compte les spectres d'émission des lumières de la scène. Dans le cas des réfractions où le chemin est valide pour uniquement une longueur d'onde, les auteurs proposèrent trois stratégies :

- la dégradation « Degradation » : une longueur d'onde est choisie pour continuer le reste du chemin. Les autres longueurs d'ondes sont dégradées (leurs propagation est stoppée), ce qui a pour effet d'augmenter la variance.
- la séparation « Splitting » : le chemin est littéralement séparé pour chaque longueur d'onde. Cette stratégie n'est pas facile à mettre en œuvre. Elle entraîne également des temps de calculs important.
- le report « Deferral » : un chemin est généré pour une seul longueur d'onde. Si il n'y a pas de réfraction, le chemin est réutilisé plus tard pour calculer les autres longueurs d'onde.

[RBA09] améliorèrent la technique de [EM99] en introduisant un échantillonnage spectral pour la génération de chemin. Afin de réduire la variance, les auteurs proposèrent d'utiliser du MIS ([VG95]) pour prendre en compte les différentes densités de probabilité des longueurs d'onde portées par un chemin.

[Wil+14] clarifièrent l'utilisation de plusieurs longueurs d'onde par chemin et de la méthode du MIS. Ils proposèrent également une version plus simple et optimisée. Le groupe de longueur d'onde était généré par une seule longueur d'onde échantillonnée (nommée « hero wavelength » par les auteurs) en appliquant un offset régulier pour couvrir l'ensemble du spectre visible.

Ces approches ont l'avantage d'être non biaisées spectralement et de nécessiter très peu de longueurs d'onde par chemin (au maximum 8) pour fonctionner de manière optimale. Cela allège considérablement les temps de calcul et la consommation mémoire.

Le problème majeur de ces méthodes pour la communauté de l'optique est qu'elles ne conservent pas la sortie spectrale car elles calculent directement le tristimulus XYZ.

4.3. La méthode de rendu proposée

Afin de répondre au problème de biais des méthodes de rendu spectral déterministe et de dimension spectrale de la simulation, une méthode de rendu spectral stochastique des canaux d'un capteur est proposée. Un canal ou une bande spectrale n'est pas une longueur d'onde mais l'intégration de la luminance spectrale modulé par la courbe de réponse du capteur sur un intervalle donné. Nous proposons donc de formuler la définition d'un canal k d'un pixel de coordonnée $(i,\ j)$ d'une image spectrale I^c sous cette forme :

$$I^c(i,\ j,\ k) = \int_{\overline{\Omega}_{ij}} \int_{\lambda_{min^k}}^{\lambda_{max^k}} W_{ij}^k(\lambda,\ \overline{\omega})\ \overline{f}(\lambda,\ \overline{\omega})\ \mathrm{d}\lambda\ \mathrm{d}\overline{\omega} \tag{4.4}$$

Si on regarde bien cette formule, on peut voir que c'est la généralisation de l'équation (4.3). Le triplet XYZ calculé par cette équation peut ainsi être considéré comme des canaux d'un capteur (œil humain) intégrés par ses courbes de réponse (courbes CIE XYZ) sur le même intervalle spectral (spectre visible).

4.3.1. Les entrées

Les entrées de la simulation sont des données spectrales (réflectance, indice complexe de réfraction, BRDF, carte d'environnement, ...). Quand on échantillonne une longueur d'onde et que l'on doit évaluer la valeur associée, on effectue une interpolation linéaire entre les bornes encadrantes de l'échantillon pour la donnée concernée.

Intuitivement, on pourrait stocker les données sous la forme de fonctions tabulées, mais la recherche des valeurs encadrantes nécessite au mieux une recherche binaire. La recherche binaire en parallèle n'est toutefois pas adaptée aux GPUs car les accès à la mémoire globale sont totalement imprévisibles et donc mauvais pour les performances.

Par conséquent, les données spectrales sont stockées avec un espacement régulier ce qui permet de trouver rapidement les bornes d'un échantillon spectral. La

résolution de chaque donnée spectrale est entièrement indépendante ce qui permet d'avoir une très bonne flexibilité et précision si besoin.

4.3.2. La simulation

Afin d'obtenir une image de dimension spectrale N_Λ, il faut simuler un N_Λ nombre de canaux en discrétisant le domaine spectral de l'équation du transport de la lumière. On peut alors formuler le rendu d'une telle image sous cette forme :

$$I^c(i,\ j) = \begin{bmatrix} \int_{\overline{\Omega}_{ij}} \int_{\lambda_{min}{}^1}^{\lambda_{max}{}^1} W_{ij}^1(\lambda^1,\ \overline{\omega}^1)\ \overline{f}(\lambda^1,\ \overline{\omega}^1)\ \mathrm{d}\lambda^1\ \mathrm{d}\overline{\omega}^1 \\ \vdots \\ \int_{\overline{\Omega}_{ij}} \int_{\lambda_{min}{}^{N_\Lambda}}^{\lambda_{max}{}^{N_\Lambda}} W_{ij}^{N_\Lambda}(\lambda^{N_\Lambda},\ \overline{\omega}^{N_\Lambda})\ \overline{f}(\lambda^{N_\Lambda},\ \overline{\omega}^{N_\Lambda})\ \mathrm{d}\lambda^{N_\Lambda}\ \mathrm{d}\overline{\omega}^{N_\Lambda} \end{bmatrix} \tag{4.5}$$

Cette approche est toutefois inefficace quand la dimension spectrale est élevée, car un chemin lumineux est valide pour l'ensemble des échantillons spectraux dans la majorité des cas. Mutualiser un chemin pour le maximum d'échantillons spectraux (1 échantillon au minimum par canal calculé) est donc la voie à suivre si on veux diminuer le nombre de calcul d'intersection inutile :

$$I^c(i,\ j) = \int_{\overline{\Omega}_{ij}} \begin{bmatrix} \int_{\lambda_{min}{}^1}^{\lambda_{max}{}^1} W_{ij}^1(\lambda^1,\ \overline{\omega})\ \overline{f}(\lambda^1,\ \overline{\omega})\ \mathrm{d}\lambda^1 \\ \vdots \\ \int_{\lambda_{min}{}^{N_\Lambda}}^{\lambda_{max}{}^{N_\Lambda}} W_{ij}^{N_\Lambda}(\lambda^{N_\Lambda},\ \overline{\omega})\ \overline{f}(\lambda^{N_\Lambda},\ \overline{\omega})\ \mathrm{d}\lambda^{N_\Lambda} \end{bmatrix} \mathrm{d}\overline{\omega} \tag{4.6}$$

Comme pour toutes les méthodes d'illumination globale robustes, l'équation se résout via la méthode Monte-Carlo :

$$\langle\ I^c(i,\ j)\ \rangle \approx \frac{1}{N_s} \sum_{s=1}^{N_s} \begin{bmatrix} \frac{W_{ij}^1(\lambda_s^1, \overline{\omega}_s)\ \overline{f}(\lambda_s^1, \overline{\omega}_s)}{\rho(\lambda_s^1, \overline{\omega}_s)} \\ \vdots \\ \frac{W_{ij}^{N_\Lambda}(\lambda_s^{N_\Lambda}, \overline{\omega}_s)\ \overline{f}(\lambda_s^{N_\Lambda}, \overline{\omega}_s)}{\rho(\lambda_s^{N_\Lambda}, \overline{\omega}_s)} \end{bmatrix} \tag{4.7}$$

4.3.2.1. Extension des chemins lumineux

Durant la simulation, les chemins sont construits en échantillonnant de nouvelles directions à chaque rebond. Baser le critère d'échantillonnage préférentiel de ces nouvelles directions sur les données spectrales de la scène est intuitivement la stratégie gagnante. D'ailleurs [Wil+14] proposèrent une méthode d'échantillonnage préférentielle spectrale en MIS qui améliora le temps de convergence du rendu spectral. Dans ce cas, la densité de probabilité ρ d'un couple d'échantillon λ_s^k et $\overline{\omega}_s$ doit s'exprimer sous cette forme :

$$\rho(\lambda_s^k,\ \overline{\omega}_s) = \rho(\lambda_s^k)\ \rho(\overline{\omega}_s|\lambda_s^k) \tag{4.8}$$

Toutefois, cette méthode peut s'avérer coûteuse en mémoire et difficile à mettre en place sur GPU. De plus, la méthode MIS de [Wil+14] nécessite de calculer les densités de probabilité de chaque échantillon spectral d'un chemin et de les accumuler pour calculer leur poids. Comme l'ont montré les auteurs, cela n'engendre que très peu de surcoût de temps de calcul sur CPU grâce aux instructions SSE pour 4 ou 8 longueurs d'onde par chemin. Ces opérations ne sont cependant pas triviales sur GPU, notamment dans notre cas où l'on a besoin de beaucoup plus de longueurs d'onde par chemin.

Dans cette étude, nous avons donc décidé de nous limiter à un l'échantillonnage préférentiel non spectralement dépendant pour le moment. Afin d'échantillonner préférentiellement de nouvelles directions, nous nous basons donc sur des valeurs intégrées ou moyennes des données spectrales de la scène. Dans cette configuration, la densité de probabilité ρ d'un couple d'échantillons λ_s^k et $\overline{\omega}_s$ peut être simplifiée :

$$\rho(\lambda_s^k,\ \overline{\omega}_s) = \rho(\lambda_s^k)\ \rho(\overline{\omega}_s) \tag{4.9}$$

L'équation (4.7) peut donc également se simplifier :

$$\langle\ I^c(i,\ j)\ \rangle \approx \frac{1}{N_s}\sum_{s=1}^{N_s}\frac{1}{\rho(\overline{\omega}_s)}\begin{bmatrix} \frac{W_{ij}^1(\lambda_s^1,\overline{\omega}_s)\ \overline{f}(\lambda_s^1,\overline{\omega}_s)}{\rho(\lambda_s^1)} \\ \vdots \\ \frac{W_{ij}^{N_\Lambda}(\lambda_s^{N_\Lambda},\overline{\omega}_s)\ \overline{f}(\lambda_s^{N_\Lambda},\overline{\omega}_s)}{\rho(\lambda_s^{N_\Lambda})} \end{bmatrix} \tag{4.10}$$

4.3.2.2. Gestions des phénomènes dispersifs

Les échantillons spectraux ne peuvent réutiliser le même chemin quand celui-ci se disperse. Dans ce cas, nous utilisons la stratégie de dégradation de [EM99]. Cette stratégie consiste à continuer la propagation du chemin avec une seule longueur d'onde. Cependant, il faudra également veiller à la pondérer correctement durant la dégradation. Par exemple, si un échantillonnage uniforme est utilisé pour choisir celle-ci, il faudra multiplier son atténuation par le nombre initial de longueurs d'onde du chemin. Cette stratégie augmente la variance (visualisable via un bruit coloré caractéristique dans le visible sur la figure 4.4).

(a) Avec 16 chemins par pixel.

(b) Avec 1024 chemins par pixel.

FIGURE 4.4. – Réfraction avec la stratégie de dégradation avec un échantillonnage uniforme pour des chemins de 64 longueurs d'onde à différent SPP.

4.3.2.3. Réduction de la consommation mémoire

Quand la dimension spectrale de l'image est élevée, le stockage d'un nombre important d'échantillons spectraux par chemin peut s'avérer coûteux en consommation mémoire. [Wil+14] proposèrent de générer à la volée les échantillons spectraux en appliquant un offset régulier à une longueur d'onde échantillonnée et stockée. Cela permet donc de stocker uniquement une longueur d'onde par chemin et donc de réduire la consommation mémoire. Cependant dans notre cas, cette méthode est inadaptée car les canaux ne sont pas forcément adjacents et leurs tailles ne sont pas nécessairement uniformes.

Afin d'éviter de stocker un échantillon spectral par canal, nous proposons de générer et de stocker une seul et unique variable aléatoire μ de loi uniforme sur $[0\,;1]$ pour chaque chemin. Cette variable est ensuite réutilisée par la fonction réciproque φ^k de la fonction de répartition de la densité de probabilité pour calculer à la volée la longueur d'onde (associée à cette variable aléatoire) de chaque canal d'indice k. Dans les faits, l'équation (4.7) se traduit maintenant par :

$$\langle\, I^c(i,\ j)\,\rangle \approx \frac{1}{N_s}\sum_{s=1}^{N_s}\begin{bmatrix} \frac{W^1_{ij}(\varphi^1(\mu_s),\overline{\omega}_s)\,\overline{f}(\varphi^1(\mu_s),\overline{\omega}_s)}{\rho(\varphi^1(\mu_s),\overline{\omega}_s)} \\ \vdots \\ \frac{W^{N_\Lambda}_{ij}(\varphi^{N_\Lambda}(\mu_s),\overline{\omega}_s)\,\overline{f}(\varphi^{N_\Lambda}(\mu_s),\overline{\omega}_s)}{\rho(\varphi^{N_\Lambda}(\mu_s),\overline{\omega}_s)} \end{bmatrix} \tag{4.11}$$

Si l'échantillonnage des directions de $\overline{\omega}_s$ n'est pas spectralement dépendant comme dans notre cas, on obtient cette équation :

$$\langle\, I^c(i,\ j)\,\rangle \approx \frac{1}{N_s}\sum_{s=1}^{N_s}\frac{1}{\rho(\overline{\omega}_s)}\begin{bmatrix} \frac{W^1_{ij}(\varphi^1(\mu_s),\overline{\omega}_s)\,\overline{f}(\varphi^1(\mu_s),\overline{\omega}_s)}{\rho(\varphi^1(\mu_s))} \\ \vdots \\ \frac{W^{N_\Lambda}_{ij}(\varphi^{N_\Lambda}(\mu_s),\overline{\omega}_s)\,\overline{f}(\varphi^{N_\Lambda}(\mu_s),\overline{\omega}_s)}{\rho(\varphi^{N_\Lambda}(\mu_s))} \end{bmatrix} \tag{4.12}$$

L'avantage de notre méthode s'estompe quand la fonction φ^k est très gourmande en temps de calculs, car il faut à chaque fois recalculer les longueurs de chaque canal pour éviter de les stocker. Pour l'instant, nous utilisons un échantillonnage uniforme des canaux, la fonction réciproque φ^k et la densité de probabilité associée sont donc très simples à calculer :

$$\rho(\varphi^k(\mu_s)) = \frac{1}{{\lambda_{max}}^k - {\lambda_{min}}^k} \tag{4.13}$$

$$\varphi^k(\mu) = (1-\mu)\times {\lambda_{min}}^k + \mu \times {\lambda_{max}}^k \tag{4.14}$$

4.3.3. Les résultats de la simulation

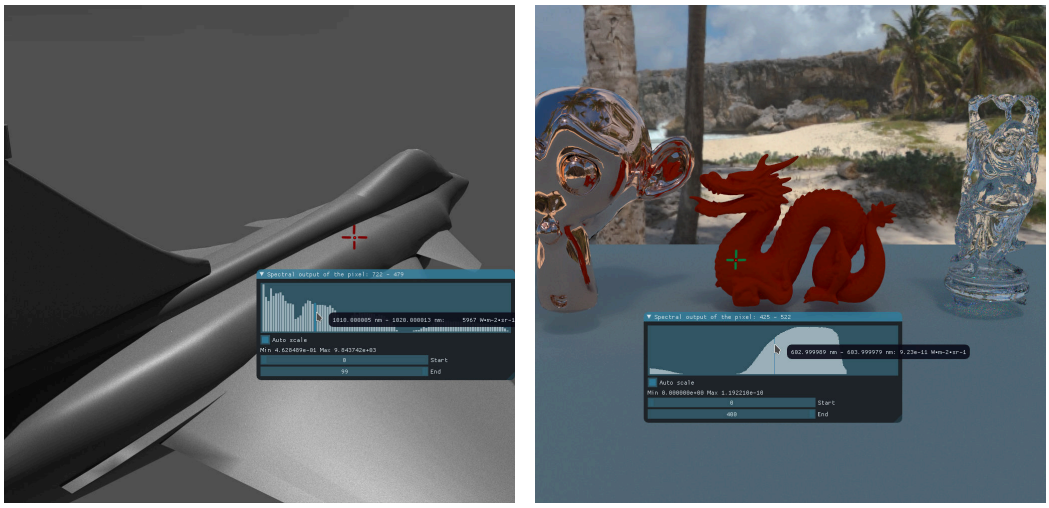
La sortie spectrale de la simulation doit être stockée dans une structure de donnée adéquate. Une structure de donnée robuste et efficace est actuellement un problème non résolu. L'approche la plus intuitive est de stocker la sortie dans une image spectrale composée de N_Λ longueur d'ondes ou canaux d'un capteur. Cette approche est toutefois inefficace au niveau consommation mémoire (image en 3D) et calcul (chaque composante spectrale de l'image doit être calculée).

En fonction des besoins de l'utilisateur, deux types de simulations sont envisageables :

— la simulation directe des canaux d'un capteur.

— la simulation en deux temps des canaux de plusieurs capteurs .

Dans tous les cas, rien n'empêche de mixer les différentes approches et de visualiser la sortie spectrale et l'image affichable en même temps à tout moment de la simulation.



(a) Luminance énergétique d'un pixel composé de 100 canaux dans le SWIR.

(b) Luminance énergétique d'un pixel composé de 400 canaux dans le spectre visible.

FIGURE 4.5. – Visualisation de la sortie spectrale d'un pixel (point rouge sur les deux images) et de l'image affichable durant la simulation.

4.3.3.1. La simulation directe des canaux d'un capteur

Dans cette configuration (figure 4.6), la luminance spectrale est intégrée avec le module capteur en utilisant l'équation (4.4). Ce mode de simulation est préconisé pour étudier le comportement d'un capteur de manière précise et rapide. En effet, l'approche stochastique de la méthode garantit que la sortie n'est pas biaisée spectralement quelle que soit la dimension spectrale de la simulation. Cette dimension sera donc égale au nombre de canaux du capteur que l'on cherche à simuler. Le nombre de canaux est généralement beaucoup plus faible (≈ 200 pour les capteurs hyper spectraux) que le nombre de longueurs d'onde nécessaires pour simuler correctement le rayonnement de la scène avec l'ancienne méthode (tableau 4.1), d'où la diminution des temps de calcul et de consommation mémoire. Si on veut étudier plusieurs capteurs, la simulation devra être toutefois relancée entièrement pour chaque capteur.

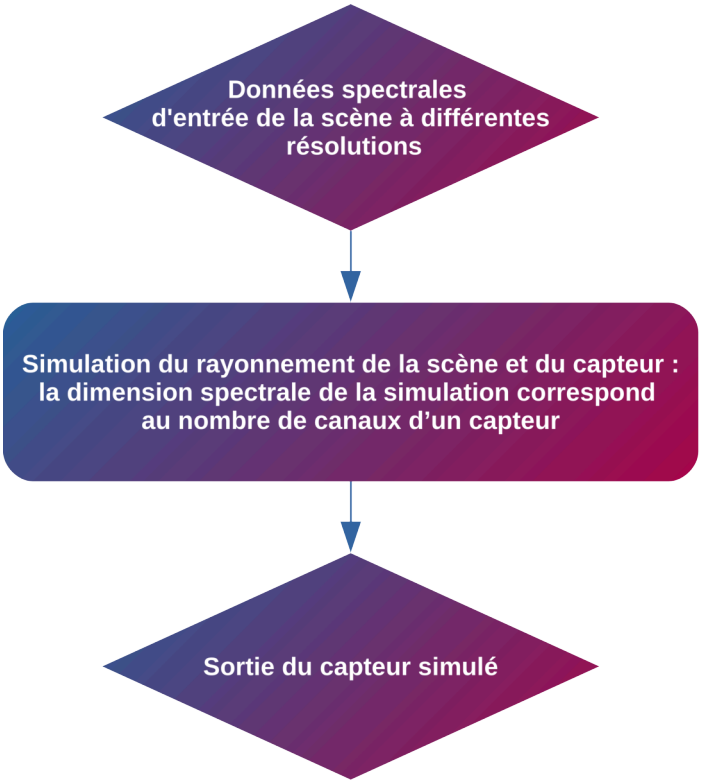


FIGURE 4.6. – Simulation avec le module capteur intégré. La dimension de la simulation n'est pas liée à la résolution spectrale des données en entrée mais au nombre de canaux du capteur simulé.

4.3.3.2. La simulation en deux temps des canaux de plusieurs capteurs

Dans ce cas d'usage, la luminance spectrale n'est pas intégrée avec le module capteur lors de la simulation mais après si besoin (figure 4.7). Dans ce cas, il faudra considérer le capteur comme un radiomètre parfait (c'est à dire que la réponse du capteur W_{ij}^{k} est constant à 1) et donc calculer la luminance énergétique d'un canal via cette formule :

$$I^{c}(i,\ j,\ k) = \int_{\overline{\Omega}_{ij}} \int_{\lambda_{min^{k}}}^{\lambda_{max^{k}}} \overline{f}(\lambda,\ \overline{\omega})\ \mathrm{d}\lambda\ \mathrm{d}\overline{\omega} \tag{4.15}$$

On peut également obtenir la luminance spectrale moyenne exacte du canal en divisant sa luminance (équation (4.15)) par son intervalle. La méthode déterministe aura cependant l'avantage de fournir la valeur exacte de luminance spectrale d'une longueur d'onde bien précise.

$$I^{c}(i,\ j,\ k) = \frac{1}{\lambda_{max^{k}} - \lambda_{min^{k}}} \int_{\overline{\Omega}_{ij}} \int_{\lambda_{min^{k}}}^{\lambda_{max^{k}}} \overline{f}(\lambda,\ \overline{\omega})\ \mathrm{d}\lambda\ \mathrm{d}\overline{\omega} \tag{4.16}$$

Cette méthode stochastique peut être utilisée dans le cas où l'utilisateur a besoin d'évaluer un grand nombre de capteurs. En effet, étant donné que la sortie spectrale n'est pas intégrée avec une réponse capteur (équation (4.15) et équation (4.16)), elle peut être réutilisée pour simuler plusieurs capteurs sans qu'il y ait le besoin de tout recalculer. Dans ce cas, il faudra considérer constante la courbe de réponse dans l'intervalle des canaux et dans les pixels en utilisant cette équation :

$$I^{c}(i,\ j,\ k) = \overline{W_{ij}^{k}} \int_{\overline{\Omega}_{ij}} \int_{\lambda_{min^{k}}}^{\lambda_{max^{k}}} \overline{f}(\lambda,\ \overline{\omega})\ \mathrm{d}\lambda\ \mathrm{d}\overline{\omega} \tag{4.17}$$

Si ce n'est pas le cas, il faudra augmenter la dimension spectrale de la simulation pour considérer comme constante les courbes de réponse des capteurs dans les subdivisions de leurs canaux.

Cette approche est également utile lorsque l'on souhaite évaluer spectralement la signature de la scène en amont d'un capteur pour identifier les bandes d'intérêt.

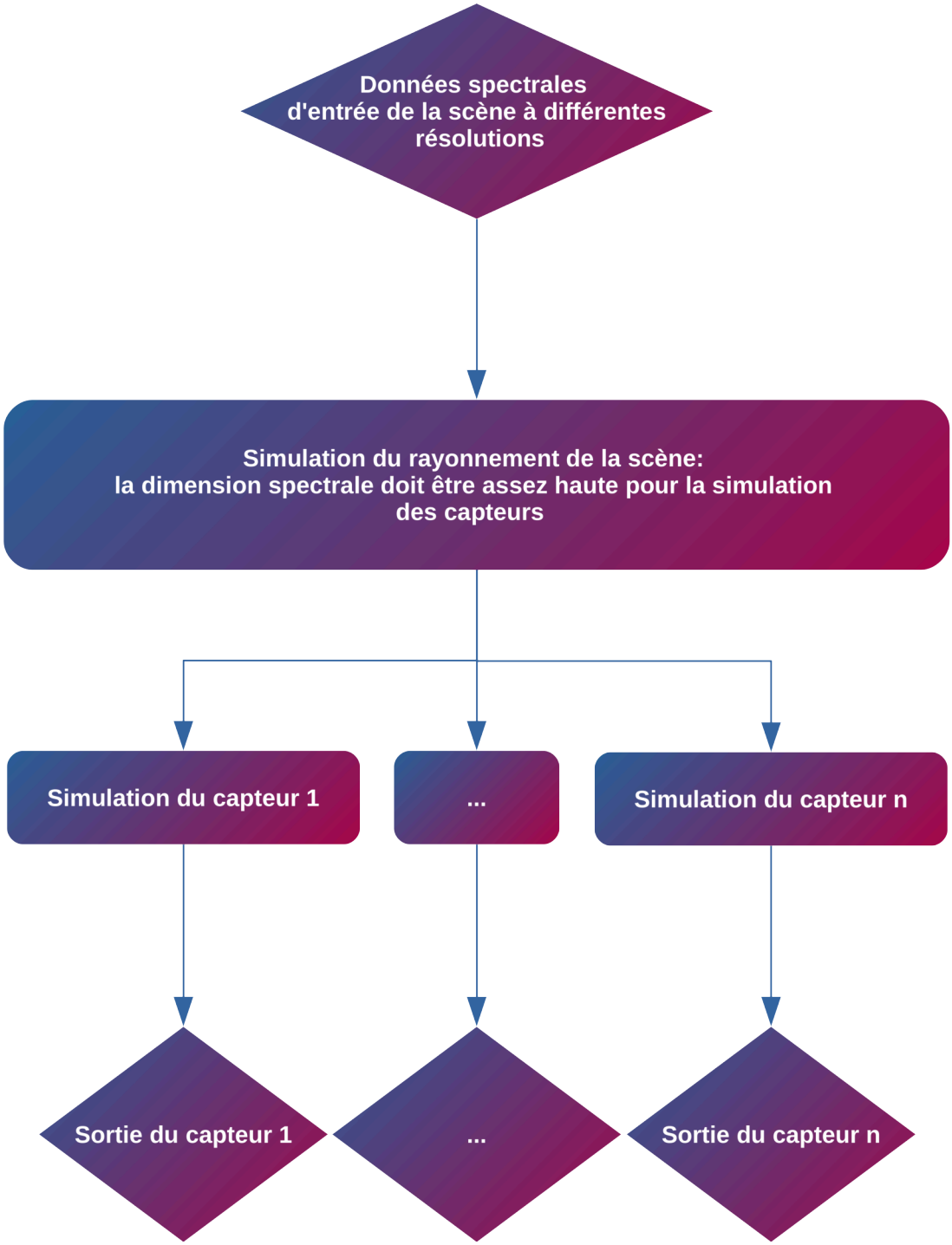


FIGURE 4.7. – Simulation en deux temps de plusieurs capteurs.

4.4. Benchmark

Afin d'illustrer l'avantage d'utiliser notre méthode, un benchmark de convergence des différentes méthodes pour différentes scènes à différentes résolutions spectrales a été réalisé.

Pour évaluer la convergence, une référence de chaque scène a été calculée très finement (au minimum à la dimension de la donnée spectrale d'entrée la plus fine) via la méthode antérieure (i.e. déterministe à pas spectral fixe). Pour chaque résolution spectrale de chaque scène du benchmark, le Mean Absolute Percentage Error (MAPE) a été calculé par rapport à cette référence au fil des itérations (toutes les 40 itérations). Afin de comparer chaque canal de l'image simulée, l'image de référence en luminance énergétique spectrale est intégrée en luminance énergique.

Voici les scènes (figure 4.8) utilisées pour le benchmark :

— Cornell Box (LFC) : les murs, le sol, le plafond et les parallélépipèdes sont des matériaux lambertiens. Leur réflectance a été calculé à partir de leur couleur RGB via la méthode de [Smi99]. Le spectre d'émission de la lampe a été remplacé par celui d'une lampe fluorescente compacte (voir le spectre d'émission via la figure 7.3). Ce spectre est donné à 0.5 nm de résolution spectrale. On peut donc considérer la scène comme spectralement complexe.

— Statues : les matériaux diffus sont des lambertiens. Leur réflectance et le spectre d'émission de chaque pixel de la carte d'environnement ont été calculés à partir de leur couleur RGB via la méthode de Smits. Un matériau réfringent (BK7) est utilisé pour le dragon de Stanford. Le verre utilisé n'est cependant pas très dispersif (voir la figure 7.4).

— SDuct : cette scène a été utilisée pour évaluer la précision des différents codes de simulation de signature infrarouge du groupe OTAN AVT-232. Elle représente une tuyère d'avion. Les mesures de matériaux ont été effectuées par le [FOI]. Les matériaux du fond de la tuyère et de la carte d'environnement sont des corps noirs. Le matériau du reste de la tuyère est pratiquement constant spectralement. Nous pouvons donc considérer cette scène comme très simple.

— Aircraft (MWIR) : une scène aérienne typiquement utilisée pour dimensionner des capteurs de détection. La scène contient une carte d'environnement spectrale générée par MATISSE ([Sim+06]) à 1 cm^{-1} dans le MWIR. Pour l'avion, le matériau du SDuct a été utilisé. La scène de ce benchmark est considérée comme simple car faute de temps et de problème de confidentialité, nous n'avons malheureusement pas pu incorporer le modèle de jet, la transmission de l'atmosphère et le vrai matériau de la cellule de l'avion. Ces éléments auraient significativement complexifiés la scène au niveau spectral.

— Aircraft (SWIR) : c'est la même scène que la précédente à l'exception près que la carte d'environnement n'est pas la même et que de la simulation se passe dans le SWIR. La carte d'environnement spectrale est générée par MATISSE à $1\,\mathrm{cm}^{-1}$ dans le SWIR. La scène est un peu plus compliquée que la précédente car le spectre d'émission du soleil est plus chahuté dans cette bande spectrale.

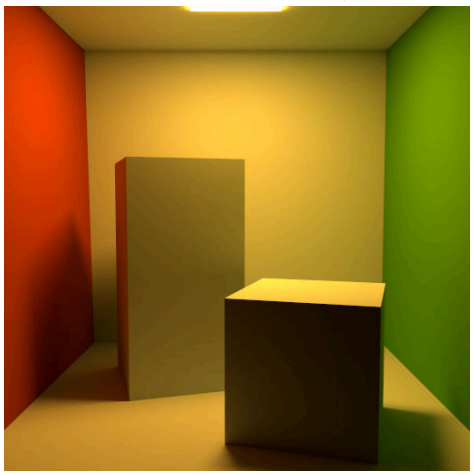
Comme pour le benchmark du précédent chapitre, la résolution spatiale est fixée à 512×512 car la mémoire de la carte graphique (AMD Fury avec 3,5 Go de VRAM) limite la résolution spectrale des images simulées. Les paramètres du benchmark de chaque scène sont récapitulés et détaillés dans le tableau 4.2.

Concernant les résultats en terme de précision, l'avantage de notre méthode n'est pas clairement visible pour les scènes simples (figure 4.11 et figure 4.13) bien qu'on peut voir un léger avantage en terme de vitesse de convergence.

Mais quand les scènes (figure 4.10 et figure 4.12) se complexifient, les résultats montrent que notre méthode est plus précise et plus rapide (quand la dimension spectrale est faible) que la méthode antérieure (i.e. déterministe à pas spectral fixe).

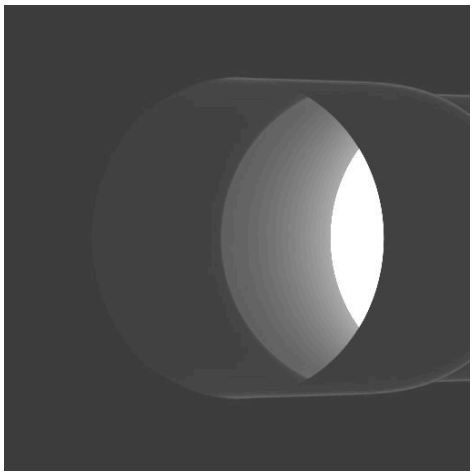
Quant aux cas vraiment difficiles (figure 4.9), l'intérêt est réel : à 20 000 chemins par pixel, l'ancienne méthode réduit au mieux le MAPE à 7% dans le meilleur des cas à 1 nm avec une courbe de convergence quasiment stabilisée. Alors que la méthode proposée réduit l'erreur à moins de 2% avec une bonne tendance de courbe et ce, quelle que soit la dimension spectrale de la simulation.

Cornell Box (LFC).



Statues.

SDuct.



Aircraft (SWIR).

Aircraft (MWIR).

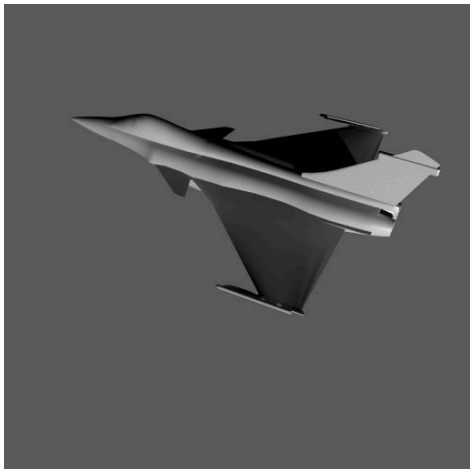


FIGURE 4.8. – Scènes du benchmark.

Scène		Cornell Box (LFC)	Statues	SDuct (LWIR)	Aircraft (SWIR)	Aircraft (MWIR)
Domaine spectral		400 - 800 nm	400 - 800 nm	855 - 1315 cm^{-1}	1 - 1,8 µm	2000 - 3333,3 cm^{-1}
Complexité spectrale		★★★★☆	★★★☆☆	★☆☆☆☆	★★☆☆☆	★☆☆☆☆
Résolution spatiale		512 × 512	512 × 512	512 × 512	512 × 512	512 × 512
Référence	Résolution spectrale	0,5 nm	1 nm	1 cm^{-1}	1 nm	1 cm^{-1}
	Nombre de longueurs d'onde	800	400	460	800	1334
	SPP	80 000	80 000	80 000	80 000	80 000
	Temps de calculs	$\approx 5h:42m$	$\approx 14h:47m$	$\approx 3h:26m$	$\approx 4h:28m$	$\approx 7h:14m$
Benchmark	Tailles des canaux	100 nm 50 nm 25 nm 10 nm 5 nm 2 nm 1 nm	100 nm 50 nm 25 nm 10 nm 5 nm 2 nm 1 nm	100 cm^{-1} 50 cm^{-1} 25 cm^{-1} 10 cm^{-1} 5 cm^{-1} 2 cm^{-1} 1 cm^{-1}	100 nm 50 nm 25 nm 10 nm 5 nm 2 nm 1 nm	100 cm^{-1} 50 cm^{-1} 25 cm^{-1} 10 cm^{-1} 5 cm^{-1} 2 cm^{-1} 1 cm^{-1}
	SPP	20 000	20 000	20 000	20 000	20 000

TABLE 4.2. – Récapitulatif des paramètres du benchmark.

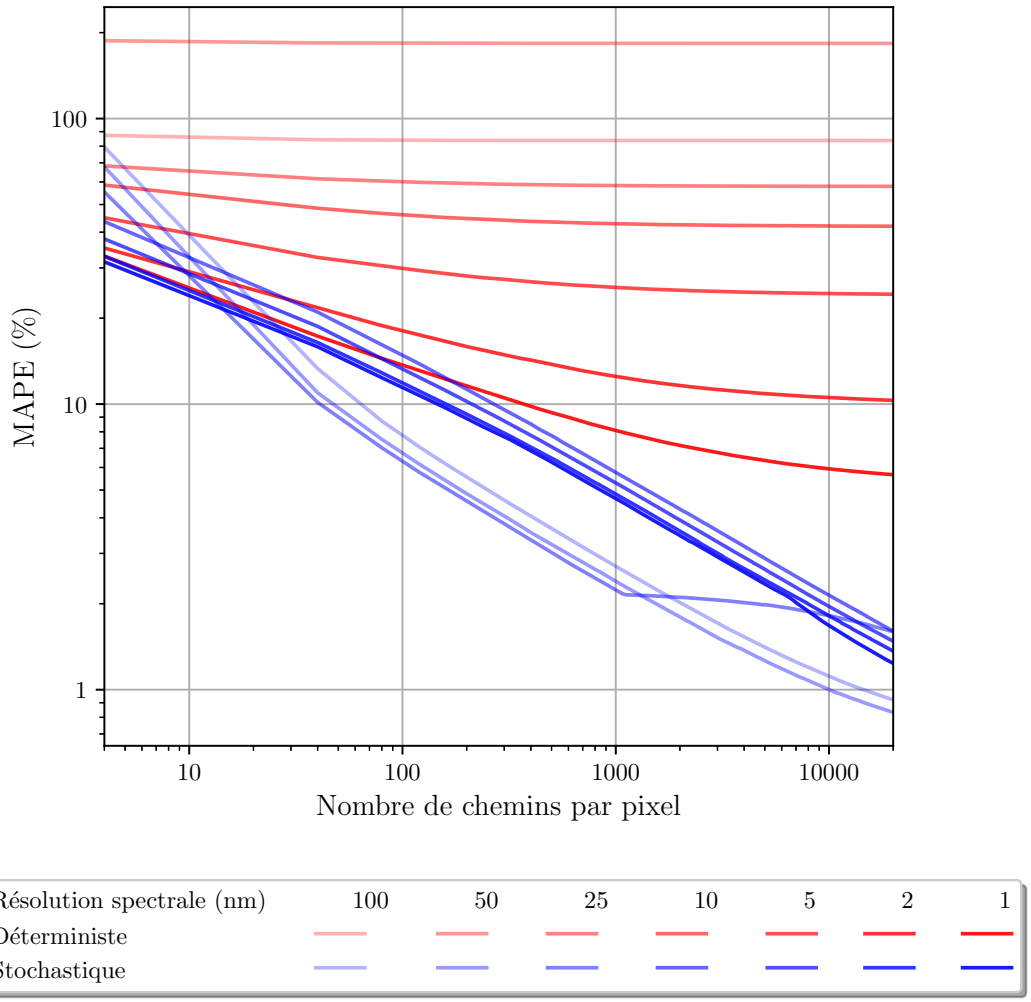


FIGURE 4.9. – Convergence moyenne (entre les canaux) à différentes résolutions spectrales pour la scène CornellBox (LFC). Les axes sont en échelle logarithmique.

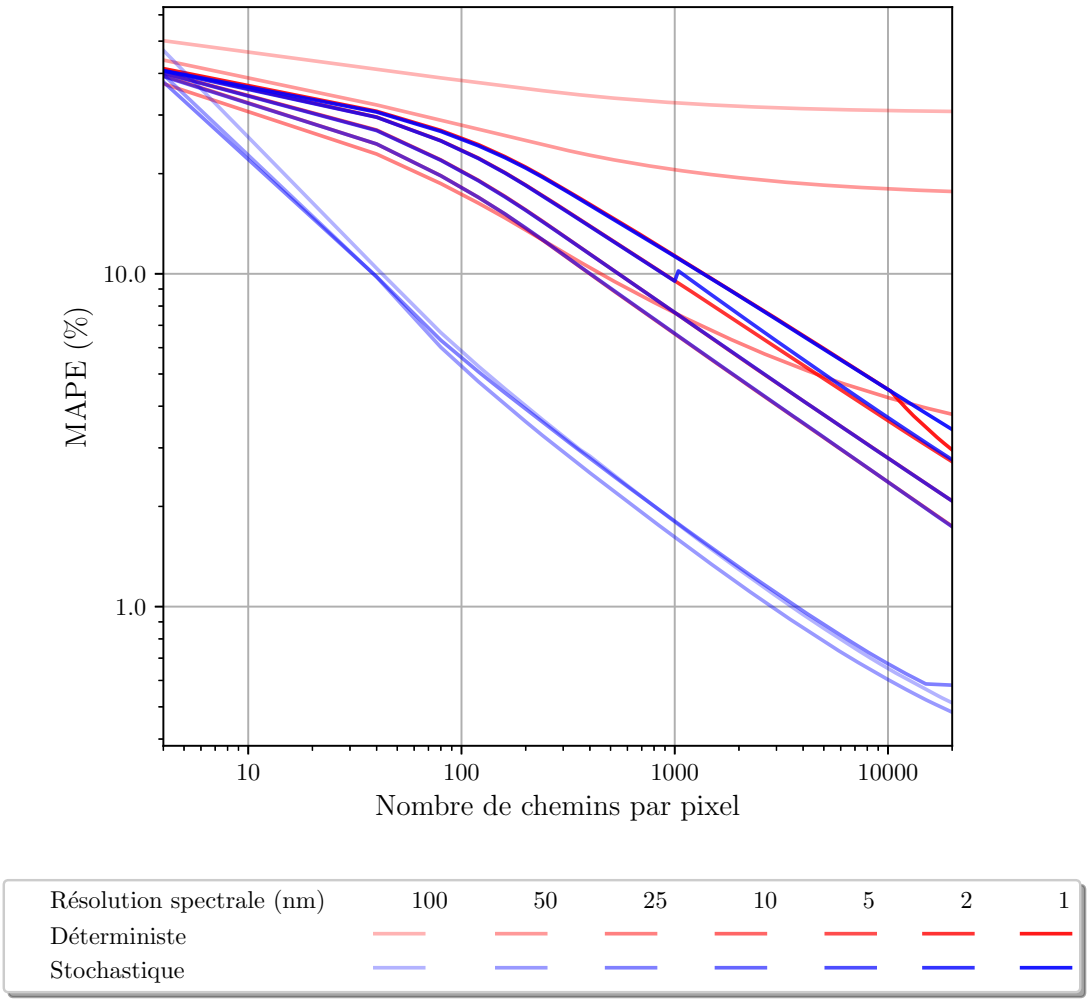


FIGURE 4.10. – Convergence moyenne (entre les canaux) à différentes résolutions spectrales pour la scène Statues. Les axes sont en échelle logarithmique.

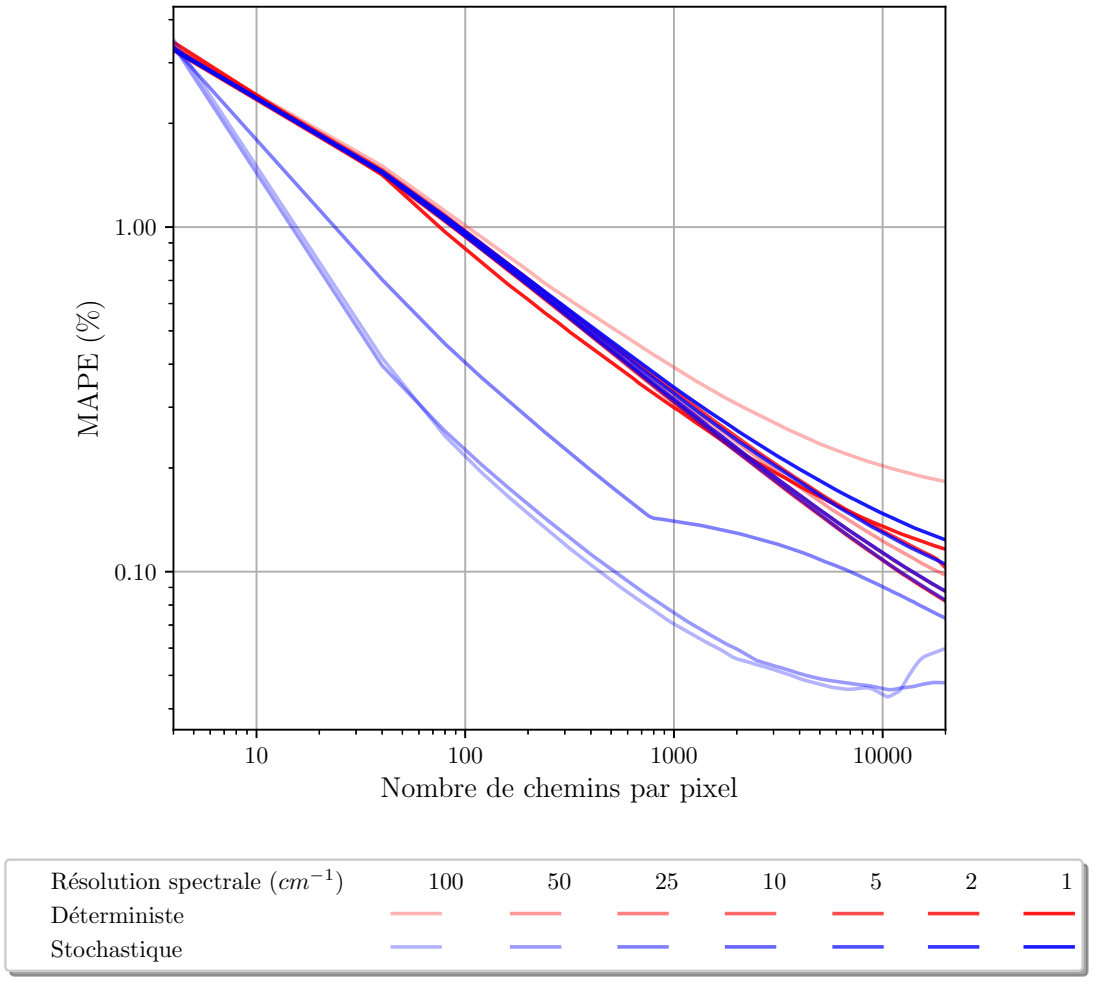


FIGURE 4.11. – Convergence moyenne (entre les canaux) à différentes résolutions spectrales pour la scène SDuct. Les axes sont en échelle logarithmique.

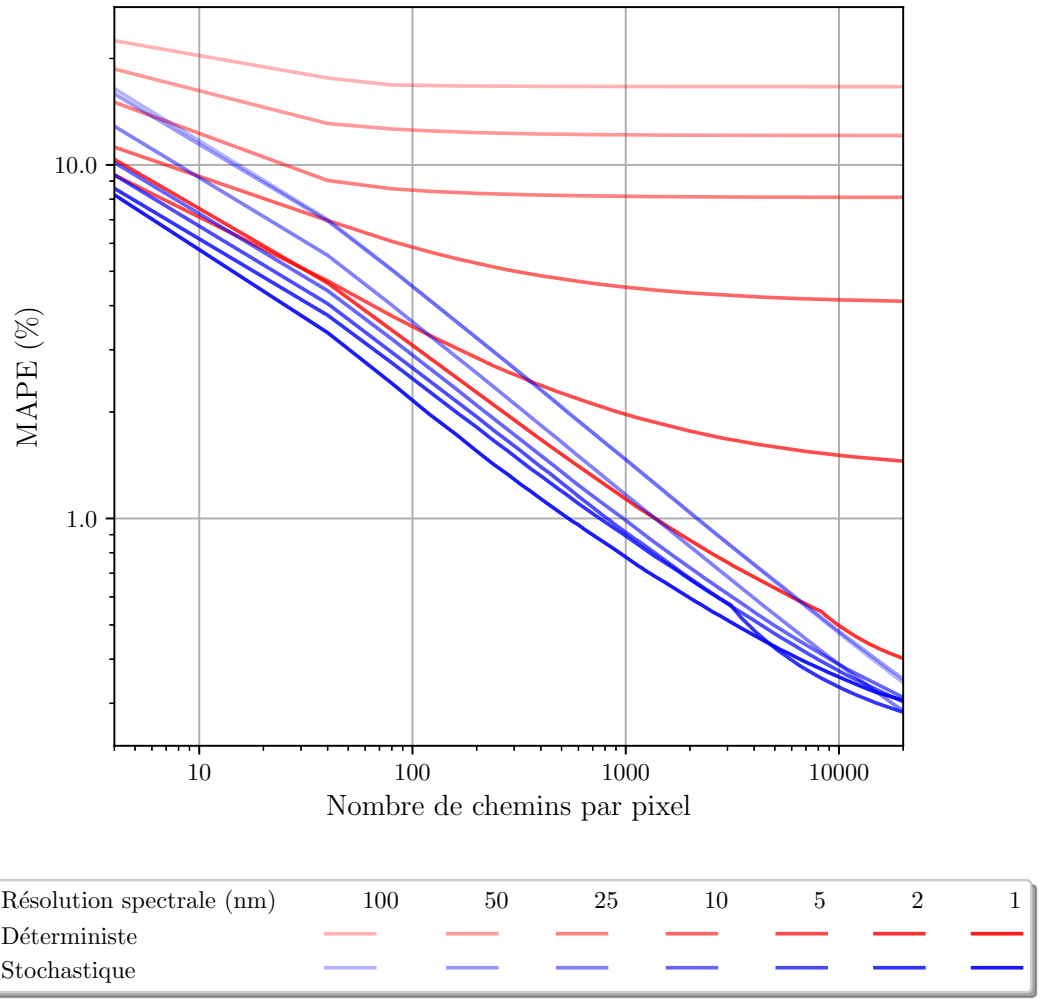


FIGURE 4.12. – Convergence moyenne (entre les canaux) à différentes résolutions spectrales pour la scène Aircraft (SWIR). Les axes sont en échelle logarithmique.

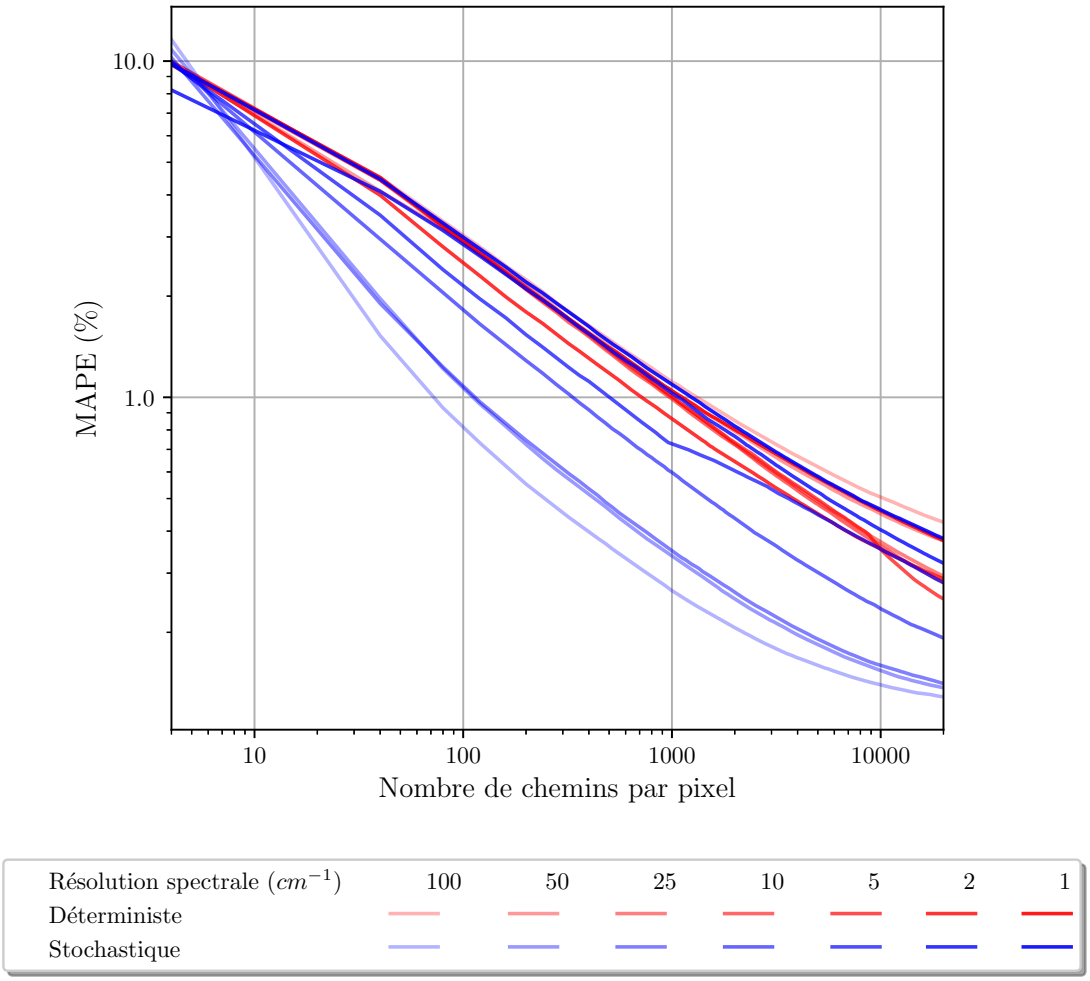


FIGURE 4.13. – Convergence moyenne (entre les canaux) à différentes résolutions spectrales pour la scène Aircraft (MWIR). Les axes sont en échelle logarithmique.

4.5. Conclusion

L'emploi de méthodes de rendu déterministe pour générer une image spectrale pose un certain nombre de problèmes :

— l'imprécision : en fixant un ensemble de longueurs d'onde, elles introduisent un biais spectral (aliasing).

— le temps de calculs : afin de mitiger ce biais spectral, on cherche généralement à faire correspondre les longueurs d'onde de la simulation avec celles des données d'entrée. Cela a pour conséquence d'augmenter inutilement la dimension spectrale de la simulation et donc du temps de calcul.

— la consommation mémoire : l'augmentation de la dimension spectrale de la simulation entraîne une surconsommation de mémoire.

Dans ce chapitre, une méthode de rendu d'image spectrale stochastique est proposée. La dimension spectrale de cette méthode n'est pas fixée par la dimension des données d'entrées mais par les besoins de l'utilisateur (nombre de canaux de l'image à simuler).

De par sa nature stochastique, cette méthode n'introduit pas de biais : quelque soit la finesse spectrale des données d'entrées, elle garantie de converger vers la bonne solution. Il en ressort deux utilisations principales :

— La simulation directe d'un capteur : c'est le mode d'utilisation principal préconisé où l'utilisateur simule en même temps le rayonnement de la scène et le capteur. L'intérêt ici est de faire un rendu précis en diminuant la dimension spectrale de la simulation.

— La simulation en deux temps des canaux pour un ensemble de capteurs : ce mode calcule dans un premier temps la luminance énergétique spectrale moyenne pour une discrétisation choisie par l'utilisateur. Cette image en luminance énergétique spectrale est ensuite utilisée pour simuler plusieurs capteurs. L'avantage est de factoriser la partie chronophage de la simulation. Par contre on introduit un biais en considérant que les courbes de réponse des capteurs sont constantes dans les pixels et dans les canaux de l'image.

Conclusion

Au travers de ce projet de recherche, nous nous sommes efforcés de répondre aux problèmes de temps de calcul et de consommation mémoire du rendu d'images spectrales en illumination globale à haute dimension spectrale.

Dans un premier temps, nous nous sommes focalisés sur la parallélisation GPU de l'algorithme du Path Tracing pour le rendu d'images spectrales. Nous avons d'abord analysé les problèmes de ce type de rendu sur GPU. Nous avons ensuite proposé le Deferred Path Evaluation Path Tracing (DPEPT) dans la section 3.4.1 et son schéma de parallélisation spectral dans la figure 3.9 qui permettent de réduire significativement la consommation mémoire et les temps de calcul.

Dans un second temps, nous avons cherché à réduire la charge de calcul spectrale de la simulation sans dégrader la précision. Dans cette optique, nous avons proposé de généraliser le rendu spectral stochastique d'image XYZ en rendu d'image spectrale stochastique. Cette nouvelle méthode permet de rendre directement et de manière plus précise et rapide les canaux d'un capteur en diminuant la dimension spectrale de la simulation quand les données spectrales d'entrée sont complexes.

Pour conclure, nos travaux permettent de simuler de manière précise des images multi, hyper et ultra spectrales. Le temps interactif peut être atteint dans notre cas (rendu d'image à faible résolution spatiale pour des scènes de complexité moyenne) en multi et hyper spectrale.

Autocritiques

Nous pensons toutefois qu'il y a encore une grande marge d'amélioration sur le sujet. Voici les principaux manques dans nos travaux :

— **l'hypothèse que chaque chemin doit porter au moins une longueur d'onde par élément spectrale de l'image** : Afin de diminuer les temps de calcul, nous avons assumé que pour chaque chemin, il fallait calculer au minimum un échantillon spectral pour chaque élément (canaux ou longueur d'onde) d'un pixel spectral. Cette hypothèse diminue le nombre d'intersections à calculer mais mène à des problèmes de consommation mémoire excessive que nous avons résolus avec le DPEPT. Il se pourrait cependant qu'il existe un meilleur compromis entre la charge de calcul spectrale (nombre de longueurs d'onde à évaluer par chemin) et la charge de calcul des intersections (nombre d'intersections à calculer pour une itération d'un pixel spectral).

Dans le cas d'une faible charge de calcul spectral, le Wavefront Path Tracing pourrait s'avérer être une meilleure approche que le DPEPT, notamment si des matériaux complexes sont utilisés dans la scène.

— **la complexité algorithmique en nombre de chemins par pixel** : Afin de répondre au besoin de l'étude, nous avons délaissé la complexité en terme de nombre de chemins par pixel (échantillonnage spectral préférentiel, ...) au profit de la complexité en terme de temps et de consommation mémoire (performance brute, consommation mémoire, parallélisation GPU, ...).

— **la consommation mémoire de l'image spectrale en sortie** : nous n'avons pas eu le temps de résoudre ce problème. En effet, bien que l'on ai pu simuler une image de résolution $512 \times 512 \times 1024$ en réduisant la consommation des algorithmes avec une AMD Fury X et ses modestes 4 Go de mémoire, la VRAM du GPU reste le facteur limitant de la résolution spatiale ou spectrale de l'image de sortie.

Approfondissements et perspectives

Les travaux qui pourraient être engagé au terme de cette thèse sont les suivantes :

— **Améliorer la complexité algorithmique en temps et en consommation mémoire** : comme il est dit ci-dessus, nous avons assumé que pour chaque chemin, il fallait calculer au minimum un échantillon spectral pour chaque élément d'un pixel spectral. Nous somme actuellement en train d'explorer une méthode de rendu stochastique d'image spectrale par régression de noyaux qui réduit la charge de calcul spectral en décorrélant la dimension spectrale de la simulation (nombre de longueurs d'onde à évaluer par chemin) et la résolution spectrale de l'image en sortie. Les premiers résultats sont très prometteurs, nous devons toutefois encore améliorer la méthode et la tester avec l'implémentation GPU du Wavefront Path Tracing.

— **Réduire la complexité algorithmique en nombre de chemins par pixel** : nous avons pour le moment délaissé cet axe de recherche, voici une liste non exhaustive des travaux possibles :
  — Adapter et intégrer le MIS spectral de [Wil+14] pour le rendu stochastique d'image spectrale.
  — Ajouter un échantillonnage spectral préférentiel pour le choix du cluster de longueurs d'onde de chaque chemin. Le critère d'échantillonnage s'inspirera de l'analyse en amont des données spectrales de la scène des méthodes de rendu d'image spectral déterministe ([ZCB98], [WTP00] et [CW05]).

— **Résoudre le problème de consommation mémoire de l'image spectral en sortie** : la VRAM du GPU reste le facteur limitant de la résolution

spatiale ou spectrale de l'image en sortie. À cet égard, nous avons imaginé plusieurs solutions :

— Utiliser une technique Out-of-Core de Multi Buffering pour raffiner l'image spectrale en faisant chevaucher alternativement les calculs et les transferts mémoire de deux ou plusieurs petits buffers.

— Paralléliser le rendu d'une image spectrale sur plusieurs GPU ou l'image spectrale sera découpée spatialement. Pour réduire la latence, uniquement l'image couleur sera reconstituée à chaque itération pour l'affichage. L'image spectrale sera quant à elle fusionnée uniquement sur demande de l'utilisateur.

— Une approche plus ambitieuse serait de développer une structure de donnée compacte (ondelette, ...) pour l'image spectrale en sortie. Cette structure de donnée devra également être efficace et adaptée pour les GPUs car il faudra raffiner l'image à chaque itération.

Production scientifique

Article de journal :

— Romain Hoarau, Eric Coiro, Sebastien Thon et al. « Interactive Hyper Spectral Image Rendering on GPU ». En cours de publication. N.D.

Article de conférence :

— Romain Hoarau, Eric Coiro, Sebastien Thon et al. « Channel based spectral image rendering of aircraft for optronic sensor dimensioning ». In : *OPTRO 2018*. Paris, France, fév. 2018

— Romain Hoarau, Eric Coiro, Romain Raffin et al. « Interactive Hyper Spectral Image Rendering on GPU ». In : *International Conference on Computer Graphics Theory and Applications*. Funchal, Portugal : SCITEPRESS - Science and Technology Publications, jan. 2018. doi : 10.5220/0006549800710080. url : https://hal.archives-ouvertes.fr/hal-01785872

— Romain Hoarau, Eric Coiro, Sébastien Thon et al. « Channel-based Spectral Image Rendering on GPU ». In : *IGRV*. Prix du meilleur papier AFIG/EGFR. Rennes, France, oct. 2017. url : https://hal.archives-ouvertes.fr/hal-01785890

Poster :

— Romain Hoarau, Eric Coiro, Sébastien Thon et al. *Interactive Hyper Spectral Image Rendering on GPU*. High-Performance Graphics. Poster. Juil. 2017. url : https://hal.archives-ouvertes.fr/hal-01785882

Bibliographie

[Ant11] Dietger Van Antwerpen. « Improving SIMD Efficiency for Parallel Monte Carlo Light Transport on the GPU ». In : *Hpg 2011* X (2011), p. 10. doi : 10.1145/2018323.2018330 (cf. p. 76).

[Bay] Bayo. https://commons.wikimedia.org/wiki/File:Position_des_shaders_dans_le_pipeline.svg?uselang=fr (cf. p. 67).

[Bou+13] Guillaume Bouchard, Jean-Claude Iehl, Victor Ostromoukhov et al. « Improving Robustness of Monte-Carlo Global Illumination with Directional Regularization ». In : *SIGGRAPH Asia 2013 Technical Briefs*. SA '13. Hong Kong, Hong Kong : ACM, 2013, 22 :1-22 :4. isbn : 978-1-4503-2629-2. doi : 10.1145/2542355.2542383. url : http://doi.acm.org/10.1145/2542355.2542383 (cf. p. 62).

[CW05] Jin R. Chern et Chung Ming Wang. « A novel progressive refinement algorithm for full spectral rendering ». In : *Real-Time Imaging* 11.2005 (2005), p. 117-127. issn : 10772014. doi : 10.1016/j.rti.2005.01.004 (cf. p. 104, 130).

[Coi12] E Coiro. « Global Illumination Technique for Aircraft Infrared Signature Calculations ». In : *Journal of Aircraft* 50.1 (2012), p. 103-113. issn : 0021-8669. doi : 10.2514/1.C031787. url : http://dx.doi.org/10.2514/1.C031787 (cf. p. 29, 62).

[Dav+14] Tomáš Davidovič, Jaroslav Křivánek, Miloš Hašan et al. « Progressive Light Transport Simulation on the GPU ». In : *ACM Transactions on Graphics* 33.3 (2014), p. 1-19. issn : 07300301. doi : 10.1145/2602144. arXiv : 1006.4903. url : http://dl.acm.org/citation.cfm?doid=2631978.2602144 (cf. p. 79).

[EM99] Glenn F. Evans et Micheal D. McCool. « Stratified wavelength clusters for efficient spectral Monte Carlo rendering ». In : *Proceedings of the 1999 conference on Graphics interface '99*. 1999, p. 42-49. isbn : 1-55860-632-7. url : http://dl.acm.org/citation.cfm?id=351631.351648 (cf. p. 108, 112).

[FOI] FOI. https://www.foi.se/en.html (cf. p. 118).

[JH19] Wenzel Jakob et Johannes Hanika. « A Low-Dimensional Function Space for Efficient Spectral Upsampling ». In : *Computer Graphics Forum* 38.2 (2019), p. 147-155. DOI : 10.1111/cgf.13626. eprint : https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13626. URL : https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13626 (cf. p. 51).

[Kaj86] James T Kajiya. « The Rendering Equation ». In : *SIGGRAPH Comput. Graph.* 20.4 (1986), p. 143-150. ISSN : 0097-8930. DOI : 10.1145/15886.15902. URL : http://doi.acm.org/10.1145/15886.15902 (cf. p. 52, 59).

[KD13] Anton S. Kaplanyan et Carsten Dachsbacher. « Path Space Regularization for Holistic and Robust Light Transport ». In : *Computer Graphics Forum* 32.2pt1 (2013), p. 63-72. DOI : 10.1111/cgf.12026. eprint : https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12026. URL : https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12026 (cf. p. 62).

[LKA13] Samuli Laine, Tero Karras et Timo Aila. « Megakernels Considered Harmful : Wavefront Path Tracing on GPUs ». In : *mediatech.aalto.fi* typically 32 (2013). URL : https://mediatech.aalto.fi/%7B~%7Dtimo/publications/laine2013hpg%7B%5C_%7Dpaper.pdf (cf. p. 77, 78).

[Mac35] David L. MacAdam. « The Theory of the Maximum Visual Efficiency of Colored Materials ». In : *J. Opt. Soc. Am.* 25.8 (août 1935), p. 249-252. DOI : 10.1364/JOSA.25.000249. URL : http://www.osapublishing.org/abstract.cfm?URI=josa-25-8-249 (cf. p. 51).

[MY19] Ian Mallett et Cem Yuksel. « Spectral Primary Decomposition for Rendering with sRGB Reflectance ». In : *Eurographics Symposium on Rendering - DL-only and Industry Track*. Sous la dir. de Tamy Boubekeur et Pradeep Sen. The Eurographics Association, 2019. ISBN : 978-3-03868-095-6. DOI : 10.2312/sr.20191216 (cf. p. 51).

[McG17] Morgan McGuire. *Computer Graphics Archive*. https://casual-effects.com/data. Juil. 2017 (cf. p. 93).

[Men+15] Johannes Meng, Florian Simon, Johannes Hanika et al. « Physically Meaningful Rendering using Tristimulus Colours ». In : *Computer Graphics Forum* 34.4 (2015), p. 31-40. DOI : 10.1111/cgf.12676. eprint : https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12676. URL : https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12676 (cf. p. 51).

[NHD10] Jan Novák, Vlastimil Havran et Carsten Dachsbacher. « Path Regeneration for Interactive Path Tracing ». In : *Eurographics 2010* (2010), p. 1-4 (cf. p. 75).

[OYH18] H. Otsu, M. Yamamoto et T. Hachisuka. « Reproducing Spectral Reflectances From Tristimulus Colours ». In : *Computer Graphics Forum* 37.6 (2018), p. 370-381. DOI : 10.1111/cgf.13332. eprint : https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13332. URL : https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13332 (cf. p. 51).

[Pol19] Mikhail N. Polyanskiy. *Refractive index database*. https://refractiveindex.info. Accessed on 2019-06-21. 2019 (cf. p. 152).

[Pur+02] Timothy J Purcell, Ian Buck, William R Mark et al. « Ray tracing on programmable graphics hardware ». In : *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2002* 21 (2002), p. 703-712. ISSN : 07300301. DOI : 10.1145/566654.566640 (cf. p. 73, 74).

[RBA09] Michal Radziszewski, Krzysztof Boryczko et Witold Alda. « An improved technique for full spectral rendering ». In : *Journal of WSCG* 17.1-3 (2009), p. 9-16. ISSN : 0014-2956. URL : http://otik.uk.zcu.cz:80/handle/11025/1278 (cf. p. 108).

[Rob14] Johanne Roby. *Spectre d'une lampe LFC de 20W et de 2700K*. http://galileo.graphycs.cegepsherbrooke.qc.ca/app/fr/lamps/2479. 2014 (cf. p. 151).

[Roh07] Robert A. Rohde. *Atmospheric Transmission*. https://en.wikipedia.org/wiki/File:Atmospheric_Transmission.png. 2007 (cf. p. 30).

[Sim+06] Pierre Simoneau, Karine Caillault, S Fauqueux et al. « MATISSE : Version 1.4 and future developments ». In : 6364 (jan. 2006) (cf. p. 93, 118).

[Smi99] Brian Smits. « An RGB-to-spectrum Conversion for Reflectances ». In : *J. Graph. Tools* 4.4 (déc. 1999), p. 11-22. ISSN : 1086-7651. DOI : 10.1080/10867651.1999.10487511. URL : http://dx.doi.org/10.1080/10867651.1999.10487511 (cf. p. 51, 105, 118).

[Vea98] Eric Veach. « Robust Monte Carlo Methods for Light Transport Simulation ». AAI9837162. Thèse de doct. Stanford, CA, USA, 1998. ISBN : 0-591-90780-1. URL : https://graphics.stanford.edu/papers/veach_thesis/thesis-bw.pdf (cf. p. 54).

[VG95] Eric Veach et Leonidas J. Guibas. « Optimally combining sampling techniques for Monte Carlo rendering ». In : *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques - SIGGRAPH '95* (1995), p. 419-428. ISSN : 00978930. DOI : 10.1145/218380.218498. URL : http://portal.acm.org/citation.cfm?doid=218380.218498 (cf. p. 57, 58, 108).

[Whi79] Turner Whitted. « An improved illumination model for shaded display ». In : *ACM SIGGRAPH Computer Graphics* 13.2 (1979), p. 14. ISSN : 00978930. DOI : 10.1145/965103.807419. URL : http://portal.acm.org/citation.cfm?doid=965103.807419 (cf. p. 29).

[Wil+14] Alexander Wilkie, Sehara Nawaz, Marc Droske et al. « Hero Wavelength Spectral Sampling ». In : *Computer Graphics Forum* 33.4 (2014), p. 123-131. ISSN : 01677055. DOI : 10.1111/cgf.12419. URL : http://doi.wiley.com/10.1111/cgf.12419 (cf. p. 108, 111, 113, 130).

[WTP00] Alexander Wilkie, Robert Tobler et Werner Purgathofer. « Raytracing of Dispersion Effects in Transparent Materials. » In : (2000) (cf. p. 104, 130).

[ZCB98] E Zeghers, S Carré et Kadi Bouatouch. « Error-bound wavelength selection for spectral rendering ». In : *The Visual Computer* 13.9 (jan. 1998), p. 424-434. ISSN : 1432-2315. DOI : 10.1007/s003710050115. URL : https://doi.org/10.1007/s003710050115 (cf. p. 104, 130).

ANNEXES

A. Choix entres les différentes technologies GPGPU

Il existe plusieurs technologies GPGPU. Dans cette section, nous avons tenté de les classer par modèle de programmation pour les comparer (tableau 7.1, tableau 7.2 et tableau 7.3) et choisir une technologie pour nos travaux de thèse. Voici les catégories retenues :

— Le modèle de programmation des sources séparées.
— Le modèle de programmation basé sur des directives.
— Le modèle de programmation des sources partagées.

Voici nos critères de comparaison :

— **La performance intra-noyau** : une note sur 5 qui prend en compte la performance et les possibilités d'optimisation de l'intérieur d'un noyau (instruction primitive de workgroup, opération atomique, mémoire locale, ...) offertes à l'utilisateur.
— **La performance extra-noyau** : une note sur 5 qui prend en compte la performance et les possibilités d'optimisation à l'extérieur des noyaux (transfert mémoire, ordonnancement et latence d'exécutions des tâches, ...) offertes à l'utilisateur.
— **La prise en main et la productivité** : une note sur 5 qui évalue la difficulté d'apprentissage et la productivité (verbosité, ...) de la technologie.
— **La maturité des implémentations** : une note sur 5 qui prend en compte la maturité et la qualité moyenne des implémentations de la technologie.
— **Le langage des noyaux** : un langage très différent du code de l'hôte est une difficulté supplémentaire à gérer.
— **Le centre de décision** : un consortium assure une certaine pérennité de la technologie en l'érigeant en standard mais ralentit ses évolutions. Alors qu'une seule entreprise avancera à marche forcée mais aura la possibilité de prendre en otage ses utilisateurs en leurs imposant ses choix.
— **La disponibilité du code source des implémentations** : des sources ouvertes permettent aux utilisateurs de reprendre le contrôle au cas où le centre de décision abandonnerait la technologie ou imposerait des choix qui vont à l'encontre de l'intérêt des utilisateurs.
— **La portabilité logicielle** : les technologies multiplateformes sont à privilégier pour éviter de n'être enfermé que sur un seul système d'exploitation ou de maintenir des bases de code différentes pour chaque plateforme.
— **La portabilité matérielle** : là encore il est préférable de choisir les technologies qui nous ne limiteront pas à un seul type de matériel ou pire à un seul constructeur.

A.1. Le modèle de programmation en source séparée

Le modèle de programmation en source séparée est le plus primitif des modèles car il provient à l'origine de la programmation GPGPU par des Vertex et Fragment shader. D'ailleurs les API graphiques OpenGL, DirectX et récemment Vulkan et Metal ont développé leurs propres shaders spécialisés pour le GPGPU.

OpenCL est un autre poids lourd de ce modèle de programmation et est une API entièrement dédié aux GPGPU voire au « Heterogeneous Computing » (calcul sur différents types d'accélérateur : GPU, FPGA, ...).

Comme pour les autres shaders, les sources sont séparées du code de l'hôte et compilées à l'exécution du programme. Le langage des noyaux est souvent assez différent du langage de l'hôte. Voici les avantages et inconvénients de ce modèle de programmation :

+ La compilation à l'exécution permet une certaine flexibilité que n'offre pas le C ou le C++ pour générer de manière avancée du code en manipulant les chaîne de caractères des sources (la méta-programmation en C++ ne permet pas de tout faire et obscurcit la compréhension du code).
+ Les implémentations des API graphiques sont matures et très bien supportées car leurs développements sont soutenus par l'industrie du divertissement (jeux vidéo, ...). Quant à la qualité et la maturité des implémentations d'OpenCL, c'est assez disparate et elles dépendent beaucoup de l'implémentation et des pilotes des constructeurs de matériel. Malgré tout, OpenCL reste quand même assez bien supporté.
-/+ Mis à part OpenCL, les performances intra-noyau sont généralement médiocres car les shaders n'exposent pas l'ensemble de la palette d'outils d'optimisations et que leurs implémentations sont peu optimisées pour le GPGPU actuellement. Cela est toutefois en train de changer, car les shaders introduisent de plus en plus de fonctionnalités bas niveau pour augmenter leurs performances intra-noyau étant donnée que les besoins en calculs de rendu de l'industrie du divertissement se complexifient de plus en plus (lancer de rayon, illumination globale, ...). D'ailleurs, Khronos Group prévoit de fusionner à terme OpenCL et Vulkan. Sinon les performance extra-noyau sont correctes.
- Besoins de gérer manuellement la protection de la propriété intellectuelle (cryptage des sources, obscurcissements des sources, ...). Des initiatives pour générer du code intermédiaire (SPIR-V) sont cependant prévues mais mal supportées pour l'instant.
- Prise en main difficile et productivité minimale car :
  — Les APIs sont très verbeuses même pour faire des choses très simples.
  — Les sources des noyaux sont dans un langage différent de celui de l'hôte et séparées du reste du projet.
  — Ces langages sont assez primitifs et ne gèrent pas par exemple la programmation orientée objet ou encore des structures de données contenant des

pointeurs. Cela complique donc l'implémentation d'algorithmes complexes et le développement de gros projets ou la modularité est nécessaire.

A.2. Le modèle de programmation basée sur des directives

Ce modèle de programmation propose à l'utilisateur d'insérer des directives au niveau des parties à paralléliser du code : « !$ » pour Fortran et « #pragma » en C/C++ (liste 2, liste 3) . À la compilation, le compilateur va se charger de paralléliser automatiquement au mieux ces parties du code en s'aidant de ces directives. Le concept a été proposé pour la 1er fois en 1997 via l'API OpenMP (« Open Multi-Processing ») pour le langage Fortran afin d'accélérer les calculs parallèles sur des processeurs multicœurs (CPUs). La version C/C++ sera publiée l'année suivante en octobre 1998.

Avec l'avènement du GPGPU, Cray, CAPS, Nvidia et PGI ont sorti en 2011 la spécification d'OpenACC. OpenACC est une autre API qui utilise le modèle de programmation basé sur des directives. Elle a cependant la particularité de se focaliser sur les GPUs.

Suite au développement d'OpenACC, OpenMP a étendu sa spécification dans sa version 4.0 pour prendre en compte les accélérateurs (ce qui inclut les GPUs). Les deux organisations souhaitent maintenant discuter d'une hypothétique fusion des deux API.

Voici les avantages et inconvénients de ce modèle de programmation sur GPU :

+ La prise en main et la productivité qui permet aux développeurs d'éviter les APIs bas niveaux et verbeuses.
+ Le bon support des compilateurs pour OpenMP. Pour OpenACC, il y a un bon support pour les compilateurs commerciaux (PGI) mais sinon il n'y a pas grand chose mis à part GCC.
- Le manque d'options d'optimisation manuelles qui nuit aux performances intra et extra noyau car trop de responsabilités sont transférées au compilateur qui ne peut pas faire de miracle dans toutes les situations.

```
    // Input (a, b) and Output (c) vectors
    int n = 1e8;
    size_t sizeInBytes = n * sizeof(double);
    double* a = (double*)(malloc(sizeInBytes));
    double* b = (double*)(malloc(sizeInBytes));
    double* c = (double*)(malloc(sizeInBytes));

    // Initialize content of input vectors
    for(int i = 0; i < n; i++) {
        a[i] = sin(i) * sin(i);
        b[i] = cos(i) * cos(i);
    }

    // Sum component wise and save result into vector c
    #pragma omp target teams \
        map(to:a[0:n], b[0:n]) map(from:c[0:n])
    #pragma omp distribute parallel for
    for(int i = 0; i < n; i++)
        c[i] = a[i] + b[i];

```

Listing 2 – Addition de vecteurs avec OpenMP.

```
    // Input (a, b) and Output (c) vectors
    int n = 1e8;
    size_t sizeInBytes = n * sizeof(double);
    double* a = (double*)(malloc(sizeInBytes));
    double* b = (double*)(malloc(sizeInBytes));
    double* c = (double*)(malloc(sizeInBytes));

    // Initialize content of input vectors
    for(int i = 0; i < n; i++) {
        a[i] = sin(i) * sin(i);
        b[i] = cos(i) * cos(i);
    }

    // Sum component wise and save result into vector c
    #pragma acc kernels copyin(a[0:n],b[0:n]), copyout(c[0:n])
    for(int i = 0; i < n; i++)
        c[i] = a[i] + b[i];

```

Listing 3 – Addition de vecteurs avec OpenACC.

A.3. Le modèle de programmation en source partagée

Le modèle de programmation en source partagée est un modèle de programmation GPGPU qui consiste à intégrer le code des noyaux dans le code hôte. Le modèle précédent pourrait être considéré comme un modèle en source partagée également, mais de par sa particularité (utilisation de directive de préprocesseur), il n'est généralement pas classé ensemble.

La technologie CUDA (initialement l'acronyme de « Compute Unified Device Architecture ») a introduit ce modèle à la sortie de son 1er SDK en février 2007. CUDA a su s'imposer dans ce domaine de par simplicité, son intégration avec le C/C++ et les options d'optimisation offertes à l'utilisateur ce qui n'est pas sans poser de problème. En effet, CUDA n'est pas un standard ouvert (les décisions sont prises uniquement par Nvidia) et fonctionne exclusivement sur les cartes propriétaires. Ce qui assure à Nvidia (au grand dam des utilisateurs) un monopole et un contrôle quasi exclusif du marché du HPC (« High Performance Computing » : super-calculateurs composés de GPU) et du monde académique. Google a proposé son compilateur CUDA ouvert en 2016. Ce compilateur (GPUCC) compile plus rapidement que NVCC (compilateur de NVIDIA) mais il n'est pas aussi complet et optimisé que NVCC. Son évolution n'est également pas à l'abri d'une action en justice si le projet nuit aux intérêts de Nvidia.

Afin de casser ce monopole de manière pérenne, AMD a développé HIP (« Heterogeneous Compute Interface »). HIP est une API open source qui vient interfacer CUDA et HCC (« Heterogeneous Compute Compiler » : une technologie open source en source partagée pour les GPUs AMD). Afin d'éviter les problèmes juridiques et d'attirer les développeurs CUDA, l'API d'HIP a la particularité de ressembler énormément à CUDA sans pour autant la copier. AMD a également proposer l'outil « hipify » qui permet d'automatiser une très grande partie du portage des sources CUDA. Bien que HIP soit ouvert, il n'est pas un standard et est encore assez jeune (par exemple, Windows n'est pas supporté pour l'instant).

Entre temps et depuis mars 2014, Khronos Group a décidé de développer SYCL. SYCL est un standard ouvert qui permet de programmer en OpenCL en sources partagées. SYCL conserve les mêmes performances intra-noyau qu'OpenCL. Il permet également de simplifier la programmation GPGPU en gérant automatiquement les transferts de mémoire et l'ordonnancement des tâches de manière consistante et optimale. Ce dernier avantage peut dans certain cas (scénario « out-of-core » ou multi GPU) pénaliser les performances extra-noyau. Conscient de ce problème, Khronos Group réfléchit à étendre la spécification pour pouvoir redonner, si besoin, du contrôle sur l'optimisation des transferts mémoires et de l'ordonnancement des tâches. Les seuls défauts de SYCL sont donc pour l'instant la jeunesse des implémentations et le manque d'option pour optimiser les performances extra-noyau. Malgré ses défauts, nous avons tout de même décidé de nous baser sur cette

technologie dans les travaux de cette thèse, car nous croyons à la pérennité qu'offre un standard ouvert.

Voici les avantages et inconvénient de ce modèle de programmation sur GPU :

+ La prise en main est généralement plus facile que le modèles à source séparée notamment avec SYCL et HCC qui réduisent au minimum la verbosité.
+ Les très bonnes performances intra et extra noyau (à l'exception de SYCL ou les performances extra-noyau peuvent être médiocres dans certain cas pour l'instant).
-/+ Le quasi monopole de CUDA qui nuit à la portabilité matérielle des projets. CUDA a une implémentation et des outils de très bonnes qualités. Les API concurrentes émergent mais elles ne sont pas encore suffisamment matures pour l'instant.

Nom	Compute Shader (Vulkan et OpenGL ≥4.3)	DirectCompute (DirectX ≥10)	Metal	OpenCL
Performance intra-noyau	★★½☆☆	★★½☆☆	★★★☆☆	★★★★☆
Performance extra-noyau	★★★★☆	★★★★☆	★★★★☆	★★★★☆
Prise en main et productivité	★☆☆☆☆	★☆☆☆☆	★☆☆☆☆	★☆☆☆☆
Maturité des implémentations	★★★★½	★★★★★	★★★★★	★★★★☆
Langage des noyaux	GLSL	HLSL	C++14 limité et non standard	C99 limité et non standard
Centre de décision	Khronos Group (consortium)	Microsoft	Apple	Khronos Group (consortium)
Code source des implémentations	Généralement fermé sauf sous Linux pour Intel et AMD	Fermé	Fermé	Généralement fermé sauf sous Linux pour Intel et AMD
Portabilité logicielle	Multiplateforme (OpenGL est déprécié sur iOS et MacOS)	Windows ≥ 7	iOS et MacOS	Multiplateforme (déprécié sur iOS et MacOS)
Portabilité matérielle	GPU	GPU	GPU	CPU, DSP, FPGA, GPU, ...

TABLE 7.1. – Comparatif des technologies GPGPU utilisant le modèle de programmation en source séparée.

Nom	OpenMP ($\geq$ 4.0)	OpenACC
Performance intra-noyau	★☆☆☆☆	★½☆☆☆
Performance extra-noyau	★☆☆☆☆	★½☆☆☆
Prise en main et productivité	★★★★½	★★★★☆
Maturité des implémentations	★★★★★	★★★★☆
Langage des noyaux	C, C++ et Fortran	C, C++ et Fortran
Centre de décision	OpenMP Architecture Review Board (consortium)	L'organisation OpenACC (consortium)
Code source des implémentations	Ouvert pour Clang et GCC	Ouvert pour Clang et GCC
Portabilité logicielle	Multiplateforme	Multiplateforme
Portabilité matérielle	CPU, FPGA, GPU, ...	CPU et GPU

TABLE 7.2. – Comparatif des technologies GPGPU utilisant le modèle de programmation basé sur des directives.

Nom	CUDA	HIP	HCC (déprécié depuis juin 2019)	SYCL
Performance intra-noyau	★★★★★	★★★★½	★★★★½	★★★★☆
Performance extra-noyau	★★★★★	★★★★★	★★★★★	★★★☆☆
Prise en main et productivité	★★½☆☆	★★½☆☆	★★★★☆	★★★★☆
Maturité des implémentations	★★★★★	★★★☆☆	★★★☆☆	★★★☆☆
Langage des noyaux	C++14 limité et non standard	C++14 limité et non standard	C++17 limité et non standard	C++17 limité
Centre de décision	Nvidia	AMD	AMD	Khronos Group (consortium)
Code source des implémentations	Fermé	Ouvert	Ouvert	Ouvert sauf pour ComputeCpp
Portabilité logicielle	Windows, Linux et MacOS	Linux	Linux	Potentiellement multiplateforme (pas d'implémentation MacOS et iOS pour le moment)
Portabilité matérielle	GPU (Nvidia)	CPU, GPU (AMD et NVIDIA)	GPU (AMD)	CPU, FPGA, GPU, ...

TABLE 7.3. – Comparatif des technologies GPGPU utilisant le modèle de programmation en sources partagées.

B. Courbes pour la photométrie.

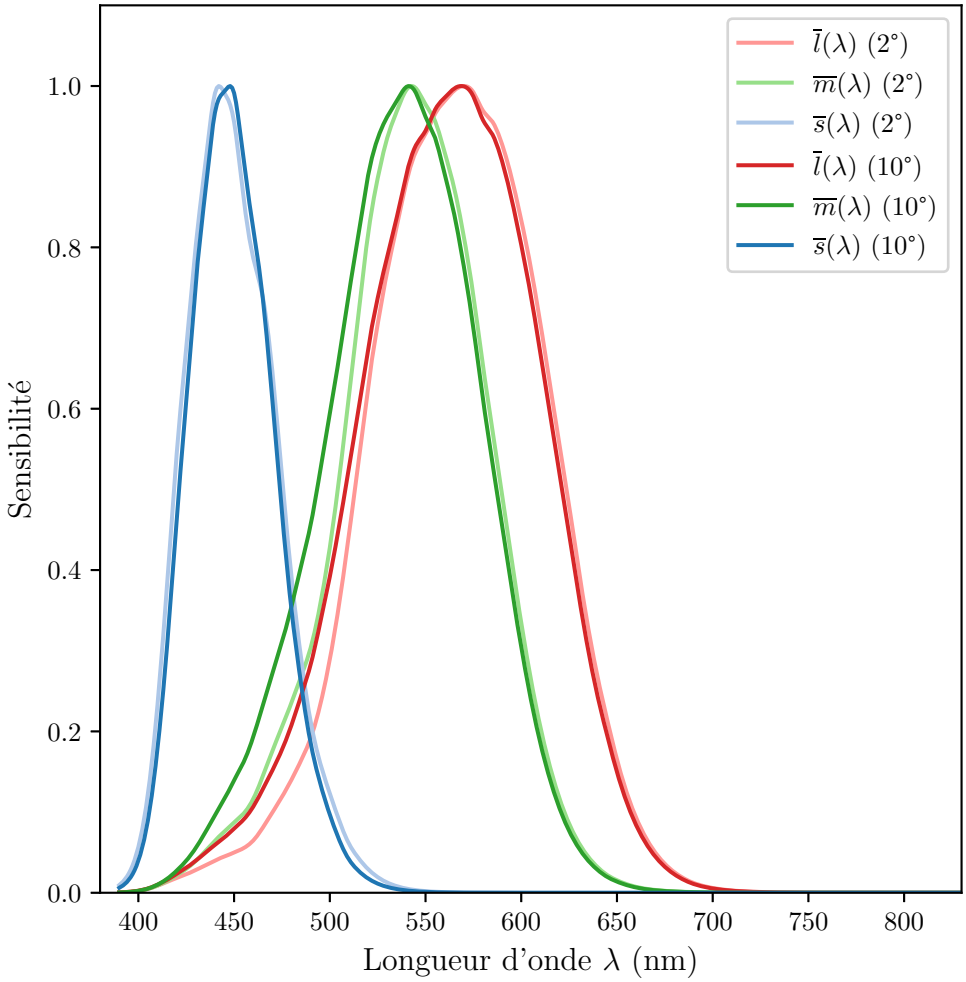


FIGURE 7.1. – Courbe de réponse des cônes de l'œil humain (vision photopique en 2° et 10°).

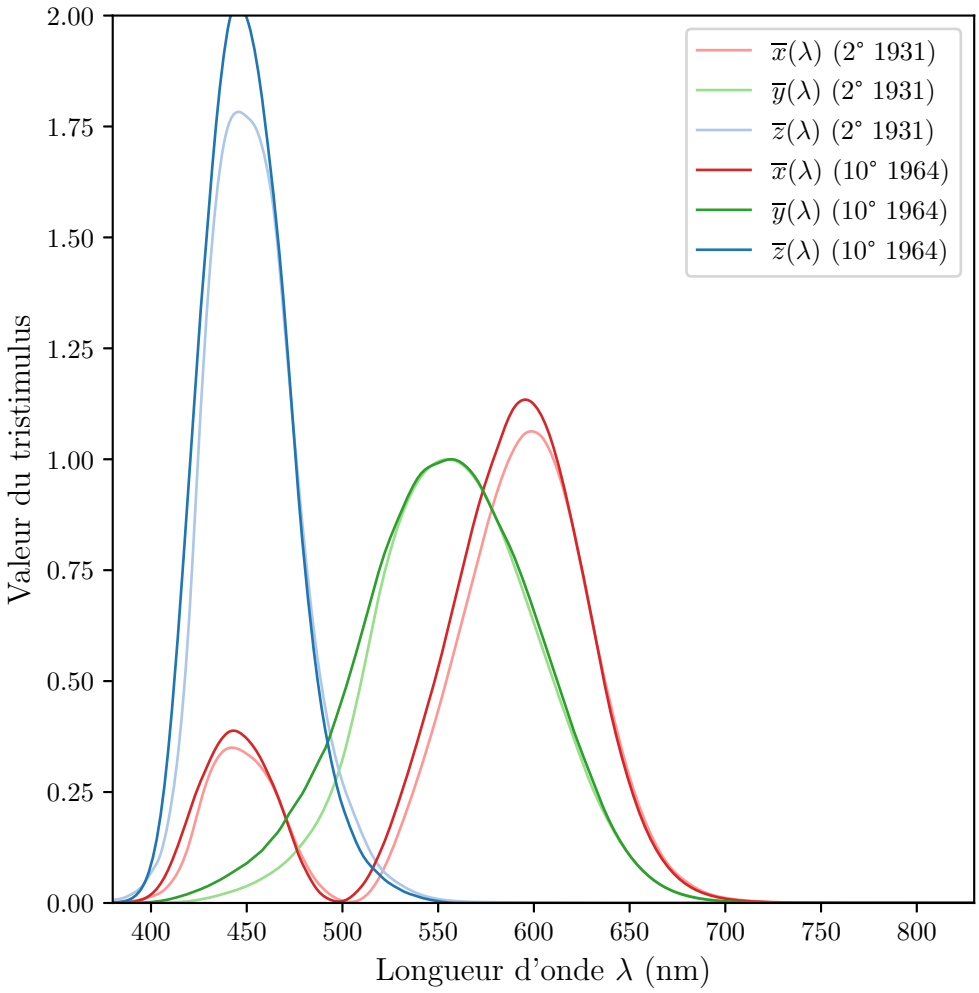


FIGURE 7.2. – Courbes CIE XYZ 1931 (2°) et 1964 (10°).

C. Détails des données spectrales de scène.

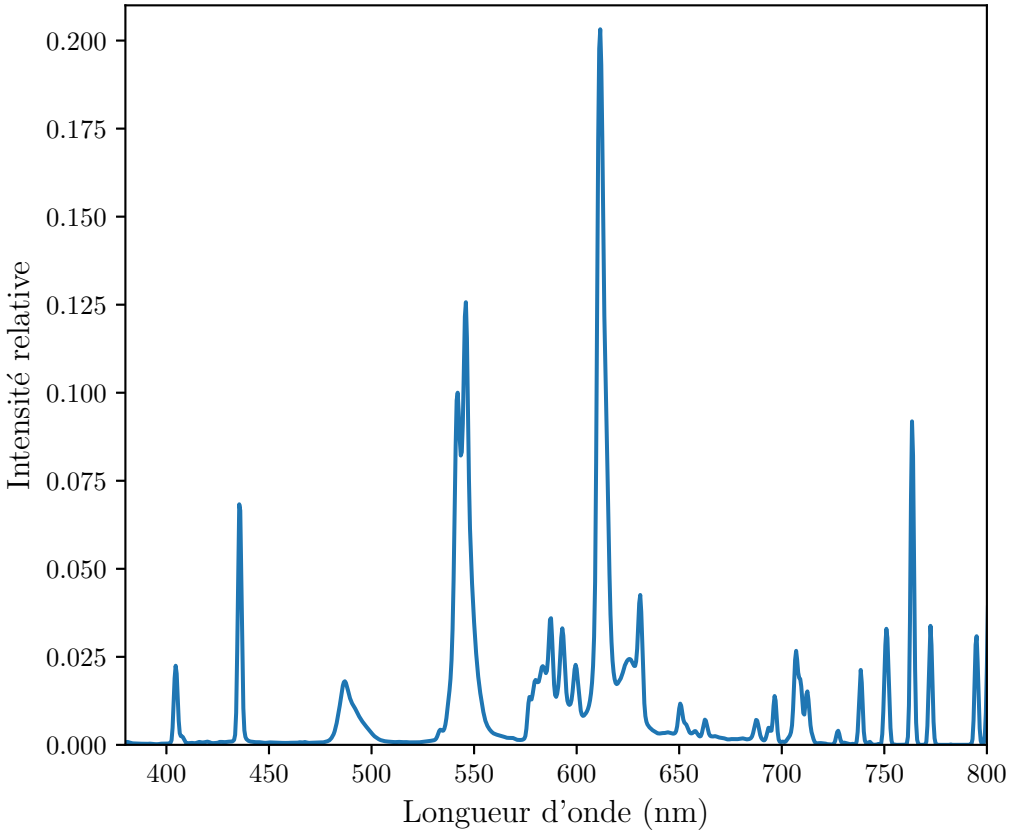


FIGURE 7.3. – Spectre d'émission d'une LFC de 20 W en 2700 K ([Rob14]). Les données sont fournies avec un échantillonnage régulier de 0,5 nm pour pouvoir représenter correctement les raies.

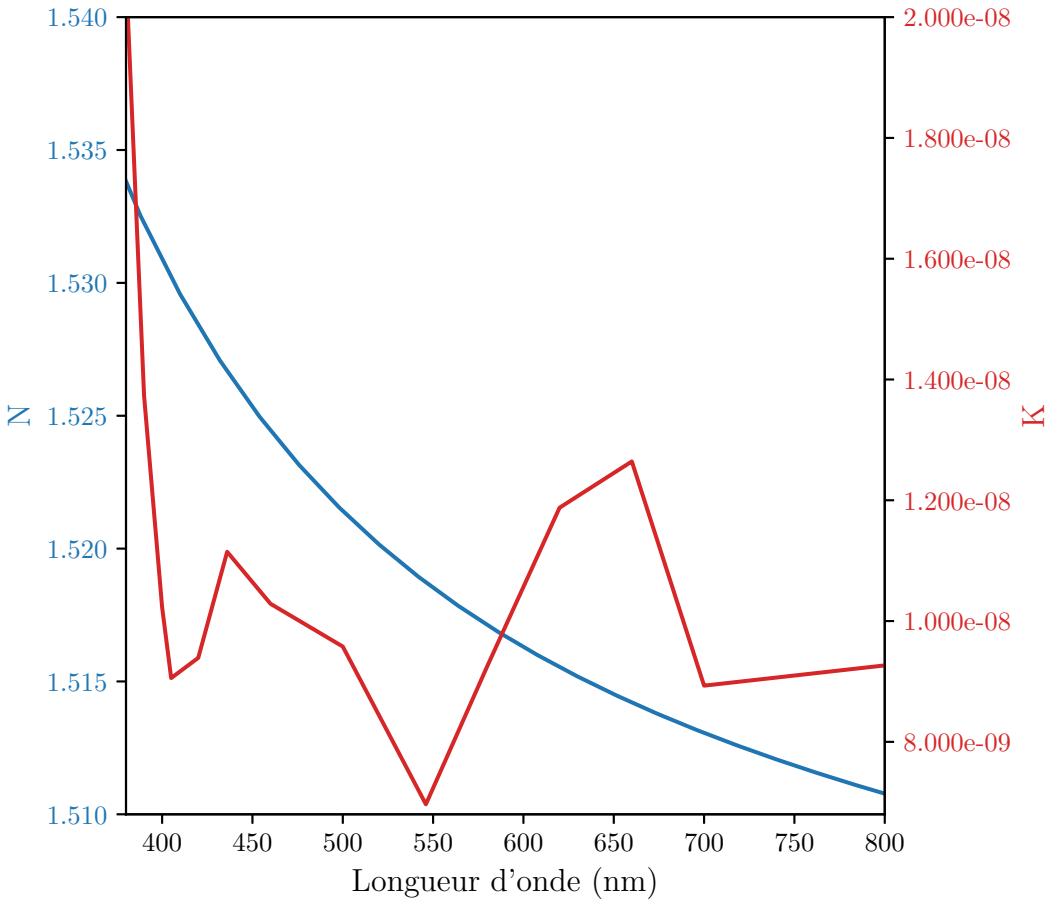


FIGURE 7.4. – Indices de réfraction complexe du BK7 ([Pol19]). Le « delta spectral » de N ($\approx 2{,}5 \times 10^{-2}$) et de K ($\approx 1{,}3 \times 10^{-8}$) sont très faibles dans le visible. Le verre n'est donc pas très dispersif.

Rendu interactif d'image hyper spectrale par illumination globale pour la prédiction de la signature infrarouge d'aéronefs

Le dimensionnement de capteur est un enjeu majeur pour le domaine de la détection d'aéronefs dans les bandes infrarouge et visible. Dans cette optique, il est nécessaire de simuler ces capteurs via des modèles et un nombre conséquent d'images spectrales d'aéronefs dans divers scénarios. L'obtention de ces images via des campagnes aériennes de mesure est malheureusement onéreuse, chronophage et difficile. Une simulation rapide et robuste de ces données s'impose donc. Afin de répondre aux besoins de précision, la complexité spectrale des scènes aériennes combinée à l'évolution technologique de la furtivité des aéronefs nous amène à utiliser des algorithmes d'illumination globale à haute dimension spectrale. Dans ces conditions, ces algorithmes posent d'importants problèmes de consommation mémoire et de temps de calcul. Le projet de recherche de cette thèse s'inscrit dans le cadre de ces problématiques.
Dans un premier temps, nous nous sommes focalisés sur l'algorithme du Path Tracing et la parallélisation GPU pour le rendu d'images spectrales. Nous avons d'abord analysé les problèmes de ce type de rendu sur GPU. Nous avons ensuite proposé le DPEPT (« Deferred Path Evaluation Path Tracing ») et un schéma de parallélisation spectral qui permettent de réduire significativement la consommation mémoire et les temps de calcul.
Dans un second temps, nous avons cherché à réduire la charge de calcul spectrale de la simulation en prenant soin de ne pas dégrader la précision. À cet égard, nous avons proposé de généraliser le rendu spectral stochastique d'image dans l'espace XYZ en rendu d'image spectrale stochastique. Cette nouvelle méthode permet de rendre directement et de manière plus précise et rapide les canaux d'un capteur en diminuant la dimension spectrale de la simulation, quand les données spectrales d'entrée sont complexes.
Pour conclure, les travaux de cette thèse permettent de simuler de manière précise des images multi, hyper et ultra spectrales. Le temps interactif peut être atteint dans notre cas (rendu d'image à faible résolution spatiale pour des scènes de complexité géométrique intermédiaire) en multi et hyper spectrale.

Mots-clés : ILLUMINATION GLOBALE ; RENDU SPECTRAL ; CALCUL GPU ; SYNTHESE IMAGE INFRAROUGE

Interactive hyper spectral image rendering by global illumination for aircraft infrared signature prediction

Sensor dimensioning is a major issue for the aircraft detection field in the infrared and the visible bands. In this vein, it is appropriate to simulate these sensors via models and a consequent set of spectral images in several scenarios. The acquisition of these images via an airborne measure campaign is unfortunately costly, time-consuming and difficult. A robust and fast simulation of these data is hence very appealing. In order to answer the needs of accuracy, the spectral complexity of the airborne scenes combined with the technological evolution of the aircraft furtivity lead us to use global illumination method in high spectral dimension. In these circumstances, these methods raise serious issues in term of memory consumption and of computing time. Our research project focuses on these problematics.
In the first instance, we have focused on the Path Tracing method and its GPU parallelization for the spectral image rendering. We have investigated at first the issues of this kind of rendering on the GPU. Then we have proposed the DPEPT ("Deferred Path Evaluation Path Tracing") and an efficient spectral parallelization pattern which allows us to reduce significantly the memory consumption and the computing time.
In the second phase, we have investigated how to reduce the spectral computational load of the simulation in taking care not to deteriorate the accuracy. In that sense, we have proposed to generalize the stochastic spectral rendering of color (XYZ) image to the stochastic spectral image rendering. This new method renders directly the channels of a sensor which allows us to reduce the memory and the computing requirements by reducing the spectral computational load of the simulation when the spectral input data of the scene are complex.
To sum up, the works of this thesis allows us to simulate accurately multi, hyper and ultra spectral images. The interactive time can be achieved in our case (low spectral image spatial dimension and medium geometric complexity scenes) in multi and hyper spectral resolution.

Keywords : GLOBAL ILLUMINATION ; SPECTRAL RENDERING ; GPU COMPUTING ; INFRARED IMAGE SYNTHESIS

ONERA
THE FRENCH AEROSPACE LAB

BP 80100 - Chemin de la Hunière FR-91123 PALAISEAU CEDEX - Tél. : +33 1 80 38 60 60 - Fax : +33 1 80 38 65 10

www.onera.fr