utc
Université de Technologie
Compiègne

# Les méthodes de Monte-Carlo appliquées aux problèmes urbains

*Proposition d'un algorithme de calcul de la radiation solaire reçue par une surface en contexte urbain, avec prise en compte des conditions météorologiques et des réflexions multiples*

Rapport d'UV TX

Thibaut Dubernet, GSU04
Printemps 2010

Enseignants suiveurs:
Benoit Beckers
Eduard Antaluca

**Résumé**

Le présent rapport constitue le rapport de l'UV TX «les méthodes de Monte Carlo appliquées aux problèmes urbains», réalisée au semestre de printemps 2010 au sein du département GSU de l'Université de Technologie de Compiègne.

Un algorithme d'estimation de la radiation solaire reçue par une surface en contexte urbain y est proposé. Cet algorithme repose sur un tir aléatoire de rayons ; il prend en compte, par l'utilisation de modèles développés par divers auteurs, les réflexions multiples ainsi que les conditions météorologiques. Une implémentation effective en langage Octave/MATLAB est par ailleurs proposée.

Cette implémentation est pensée pour être utilisable dans un cadre de conception. Pour cette raison, peu de paramètres d'entrée sont nécessités ; sa réalisation a par ailleurs été guidée par la volonté de conserver un temps de calcul raisonnable.

*The sciences do not try to explain, they hardly even try to interpret, they mainly make models. By a model is meant a mathematical construct which, with the addition of certain verbal interpretations, describes observed phenomena. The justification of such a mathematical construct is solely and precisely that it is expected to work.*

**J. Von Neumann, Methods in physical Sciences, 1955**

# Table des matières

# Introduction

Le rayonnement solaire fourni la majorité de l'énergie terrestre et est essentiel à la vie. Trop de soleil peut cependant être désagréable, voire nuisible.

Dans toutes les parties du globe, l'architecture s'adapte au rayonnement local : des maisons andalouses blanches à petites fenêtres aux constructions «bioclimatiques», en passant par les grandes verrières du XIX$^{e}$ siècle,la recherche d'un optimum solaire prend la forme de la recherche d'un équilibre entre les deux aspects fondamentaux de la lumière — l'aspect visuel et l'aspect énergétique. Aucune solution générale n'existe : de grandes surfaces vitrées permettront l'entrée de la lumière, mais seront responsable de surchauffe en climat chaud, et de déperdition de chaleur en climat plus froid.

En contexte urbain, la géométrie complexe de l'environnement complique encore la prédiction : les effets de masque ou de de réflexion peuvent infirmer totalement une estimation trop grossière, pourtant nécessaire dans cette recherche d'optimisation du système bâti.

La science physique, par ailleurs, élabore des modèles de plus en plus fins des phénomènes radiatifs. Il est loin, le temps de l'optique géométrique de Descartes ! Toute les productions scientifiques (et en particulier dans le champ physique) consistent principalement en *modèles*, la plupart du temps mathématisés, du monde. Comme l'a affirmé Von Neumann, ces modèle n'expliquent pas le monde, mais le décrivent, le prédisent. Divers modèles peuvent exister pour un même phénomène, plus ou moins fins, reposant sur des hypothèses différentes.

Dans ce rapport, nous proposons un algorithme d'estimation du rayonnement solaire reçu par une surface. Nous nous basons pour cela sur divers modèles rencontrés dans la littérature scientifique, que nous passons rapidement en revue dans le chapitre 1. Cet algorithme, présenté au chapitre 2, est basé sur une intégration Monte Carlo par lancer de rayons. Cette méthode permet la prise en compte des phénomènes de réflexion et de masques, et est à ce titre particulièrement adapté dans le contexte urbain. L'utilisation d'un modèle de ciel permet la prise en compte des conditions atmosphériques.

Une implémentation directement utilisable de cette algorithme est ensuite proposée, en langage Octave/MATLAB[1]. L'approche retenue pour rendre cette implémentation efficace en termes de temps de calcul est présentée au chapitre 3. L'intégralité du code source est par ailleurs proposée en annexe B.

[1] MATLAB est un logiciel de calcul numérique édité par la société the MathWorks ; GNU Octave est un logiciel libre et gratuit publié sous LICENCE GNU GPL, offrant une excellente compatibilité avec MATLAB

Chapitre 1

# Radiation solaire et modélisation des phénomènes radiatifs

Les méthodes de Monte-Carlo et la technique de lancer de rayons sont des méthodes utilisées dans de nombreux domaines. De nombreuses autres approches sont par ailleurs utilisées pour la prédiction de la radiation solaire reçue en contexte urbain. Le présent chapitre propose un bref aperçu des données rencontrées dans la littérature sur ces sujets.

## 1.1 Les méthodes de Monte-Carlo : concept et méthodes de réduction de la variance

Les méthodes de Monte-Carlo sont une catégorie de méthodes de simulation reposant sur l'utilisation de valeurs aléatoires. Grossièrement, le sujet simulé peut être de deux types [Madras 02] :

1. système *par nature* aléatoire[1] ;
2. système *déterministe*, auquel on donne artificiellement un caractère aléatoire pour les besoins de la simulation.

Ces méthodes peuvent servir à la résolution d'un grand nombre de problèmes, et les algorithmes peuvent prendre un grand nombre de formes. Les algorithmes génétiques —qui consistent en la génération aléatoire de solutions, améliorées par choix et combinaison— et la méthode du "recuit simulé" —méthode de recherche d'optima locaux utilisant des "sauts" aléatoires—sont par exemple classés dans les méthodes de Monte-Carlo.

Une autre forme de méthodes de Monte-Carlo est l'interprétation d'une intégrale comme l'espérance d'une variable aléatoire. C'est cette forme de méthodes de Monte-Carlo qui sera utilisée dans ce projet, et c'est de cette forme qu'il sera maintenant question.

### 1.1.1 Intégration Monte-Carlo : principes généraux

L'intégration Monte-Carlo est une méthode numérique d'approximation d'intégrales, généralement utilisée pour des intégrales de dimension élevée [Madras 02]. Plutôt de de construire une interpolation intégrable, comme le proposent d'autres méthodes [Utc 07], l'intégrale est considérée comme l'espérance d'une variable aléatoire.

Le principe est le suivant : soit $I$ l'intégrale à approximer, d'expression

$$I = \int_S k(x)\,dx,\ S \subseteq \mathbb{R}^d \tag{1.1}$$

On observe sans peine que, pour toute application $f$ à valeur dans $\mathbb{R}$, on a :

$$(\forall x \in S,\ f(x) \neq 0) \Rightarrow I = \int_S \frac{k(x)}{f(x)} f(x)\,dx \tag{1.2}$$

[1] à cette catégorie s'appliquent les «Méthodes de Monte-Carlo par Chaînes de Markov», qui sont des méthodes permettant la simulation de "Chaînes de Markov", systèmes dont les états futurs ne peuvent pas êtres prédits grâce à la connaissance des états passés et présent.

Le lien avec les probabilités est alors immédiat, et l'on a :

$$(\forall x \in S,\ f(x) \in \mathbb{R}_+) \Rightarrow I = \mathbb{E}\left(\frac{k(X)}{f(X)}\right) = \mathbb{E}\left(h(X)\right) \tag{1.3}$$

Avec $X$ une variable aléatoire à valeurs dans $S$, de loi de densité de probabilité $f$. Or la moyenne arithmétique d'une expérience aléatoire est un estimateur sans biais convergent de l'espérance mathématique [Denoeux 09]. Un estimateur de l'intégrale recherchée est alors

$$\hat{I}_n = \frac{1}{n}\sum_{i=1}^{n} h(x_i) \tag{1.4}$$

Avec $x_i,\ i = 1, \ldots, n$, $n$ réalisations de $X$.
La variance de l'estimateur est alors :

$$\mathrm{var}(\hat{I}_n) = \frac{\mathrm{var}(h(X))}{n} \propto \frac{1}{n} \tag{1.5}$$

Cette relation entre le nombre de tirages et la variance est importante : puisque les variations de l'erreur suivent celles de l'écart type, qui est la racine carrée de la variance, il faut quadrupler le nombre de tirages pour diviser par deux l'erreur [Uni 03, Madras 02].

Pour cette raison, on préfère souvent d'autres méthodes pour les intégrales continues de dimension faible, pour lesquelles des méthodes d'approximation moins coûteuses existent [Madras 02, Uni 03].

### 1.1.2 Remarques sur la génération aléatoire

L'intégration Monte-Carlo repose donc sur la réalisation, généralement informatisée, du tirage d'une variable aléatoire de loi de probabilité connue. Les problèmes suivants se posent alors :

- on cherche à réaliser une expérience aléatoire sur une machine informatique, par nature déterministe ;
- Selon le tirage, l'estimation calculée peu varier ; notamment, certains tirages «atypiques», de même probabilité que les autres, peuvent amener à une estimation médiocre [2] ;

La génération, dite génération pseudo-aléatoire, se fait en général par le biais d'une suite chaotique prenant en entrée une valeur définie par le système (fréquence CPU par exemple), permettant d'obtenir une famille d'éléments de $[0, 1]$, ensuite modifiée pour obéir à la distribution souhaitée.

Une autre manière de considérer le problème, menant à des algorithmes dits «quasi-Monte-Carlo » [Uni 03], est la suivante : puisque l'on sait facilement construire une suite régulière ayant la même probabilité que n'importe quelle autre dans le cas uniforme, pourquoi ne pas effectuer le calcul avec une suite de départ régulière ?

Ces méthodes vont alors utiliser un maillage régulier, ou, du moins, totalement déterministe —au sens ou le même maillage sera toujours généré—, sur lequel les valeurs de la fonction à intégrer vont être calculées. Afin d'éviter certains *artefacts* numériques, les maillages peu réguliers, ou «randomisés», sont préférés [Uni 03].

[2] Prenons pour exemple un cas concret : le lancer à répétition d'un dé, supposé parfaitement équilibré. Sous l'hypothèse de l'indépendance des réalisations, la suite 111111111..., *i.e.* correspondant à la sortie d'un 1 à chaque lancer, a strictement *la même* probabilité que n'importe laquelle des autres suites aléatoires générables. Il est cependant évident qu'avec une telle suite, l'estimation ne sera pas améliorée par rapport à un seul lancer. Une réponse à ce paradoxe a été proposée par Kolmogorov, avec sa définition de la «complexité algorithmique» d'une suite [Bienvenu 10].

### 1.1.3 Améliorer la précision : méthodes de réduction de la variance

Comme vu en 1.1.1, la variance d'une estimation par méthode de Monte-Carlo «brute» est peu satisfaisante. Diverses astuces, appelées méthodes de réduction de la variance, ont étés développées pour permettre de diminuer la variance du résultat. Nous présentons ici deux de ces méthodes, largement utilisées [Madras 02, Uni 03] : l'échantillonage préférentiel et la stratification.

La méthode d'échantillonage préférentiel est fréquemment utilisée en rendu d'images [Uni 03, Szirmay-Kalos 00, Schlick 94]. Cette méthode consiste à choisir une loi de densité de probabilité non triviale[3] pour la variable aléatoire. Ce choix est cependant délicat : la variance peut être fortement diminuée par le choix d'une loi de densité de probabilité adaptée, mais peut aussi être fortement augmentée par un mauvais choix ! Il est par ailleurs difficile de quantifier l'amélioration possible de la variance *a priori* [Madras 02].

Une autre méthode, appelée stratification, s'inspire de la pratique courante en statistiques descriptives. Prenons, pour exemple, le cas d'un institut statistique souhaitant étudier une certaine caractéristique de la population nationale. Cette étude sera basée sur la récolte d'un nombre limité d'enquêtes, dont les réponses devront être synthétisées en un indicateur le plus représentatif possible.

La pratique courante dans ce cas se base sur l'hypothèse que les pratiques des individus sont fortement liées à la catégorie socioprofessionnelle à laquelle ils appartiennent. Les réponses sont associées à la mention de cette catégorie socioprofessionnelle, et un poids est affecté à chaque réponse, de manière à obtenir une distribution des individus par CSP fictivement identique dans l'échantillon et dans la population.

La transposition aux méthodes de Monte-Carlo se fait au niveau de la génération pseudo-aléatoire. Plutôt que de générer des valeurs directement sur l'ensemble du domaine, le domaine est séparé en sous-domaines, sur lesquels sont générées des valeurs. On montre qu'avec cette méthode, la variance du résultat est toujours inférieure ou égale à la la méthode «brute». L'ampleur de l'amélioration sera cependant variable selon la méthode de stratification utilisée.

Une première utilisation de la stratification est appelée allocation proportionnelle : le nombre de valeurs générées par sous-domaine est directement lié à la loi de densité de probabilité. Une autre méthode, plus efficace, consiste à générer plus de valeurs dans les sous-domaines où l'on sait l'information plus complexe. Cette deuxième méthode suppose cependant un connaissance préalable de la distribution spatiale de l'information.

Une forme extrême d'allocation proportionnelle, nommée jittered sampling, est rencontrée en graphisme pour générer une répartition uniforme [Uni 03]. Dans cette approche, le domaine est subdivisé en autant de sous-domaines que l'on veut générer de points.

[3] c'est à dire, dans la plupart des cas, non uniforme

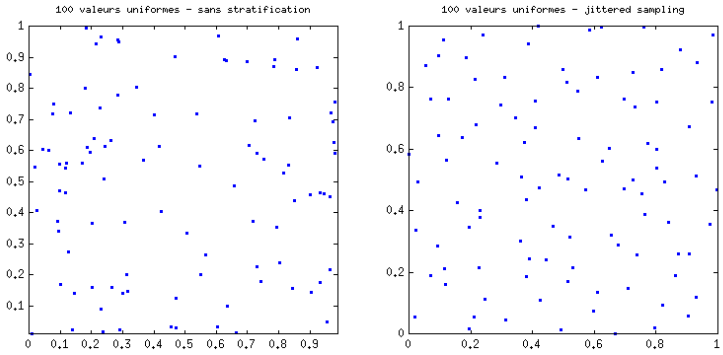



FIG. 1.1 – jittered sampling : génération de 100 points sur le carré unité

## 1.2 La simulation par lancer de rayons : domaines d'application et principes algorithmiques

La simulation par lancer aléatoire de rayons est utilisée dans de nombreux domaines, de la climatologie à l'acoustique, en passant par le rendu d'images. Suivant le but de l'algorithme, de nombreuses approches peuvent être adoptées. Nous passons en revue ci-après diverses approches rencontrées dans la littérature.

### 1.2.1 Lancer aléatoire de rayons : exemples en climatologie

Dans le domaine de l'étude de la radiation lumineuse, deux domaines utilisent abondamment la simulation par lancer de rayons aléatoire :

- l'étude du transfert radiatif, c'est à dire de la manière dont les rayonnements de dissipent au cours de leur traversée de l'atmosphère. La simulation de ce phénomène peut permettre de corriger les données de rayonnement infrarouge mesurées par satellite, ou encore de mieux comprendre certains phénomènes climatiques.
- le rendu photoréaliste d'images. Dans cette approche, un visuel est reconstitué à partir d'une scène 3D, par le biais d'un lancer de rayons se voulant le plus proche possible de la réalité physique.

Un exemple de simulation dans le domaine de la mesure à distance peut être trouvé dans [Doyle 98]. Le but du modèle de Doyle est l'estimation des propriétés et de la nature des particules en suspension dans l'atmosphère sur la base de la mesure de la couleur des océans par satellite. Ce qui nous intéresse ici est la manière d'arriver à un résultat exploitable : un lancer de rayons est effectué, à travers une atmosphère modélisée. Le chemin suivi par les rayons est régi par l'équation du transfert radiatif (RTE, pour Radiative Transfer Equation). Cette loi physique décrit la radiance d'un point de l'espace en fonction du flux incident en ce point. En considérant l'équation du transfert radiatif comme une loi de densité de probabilité portant sur la direction de diffraction des rayons incidents, un algorithme par méthode de Monte-Carlo est développé, permettant l'estimation de la radiance mesurée en un point de l'atmosphère en fonction des propriétés de l'atmosphère.

Plutôt que de considérer chaque lancer comme le lancer d'un photon, un poids est affecté au rayon : ce poids, initialement égal à un, est décrémenté au cours de la simulation. Cette approche permet de diminuer le nombre de rayons "perdus" par absorption totale : le poids peut être interprété comme la proportion de photons arrivant effectivement à destination en suivant ce chemin.

Un rayon est suivi jusqu'à la chute du poids sous une valeur seuil, l'atteinte d'un capteur ou des limites du domaine.

Un autre modèle utilisé en climatologie nous parait intéressant ici, du fait de l'inclusion de nuages en trois dimensions [Chen 06]. Les nuages modélisés sont constitués d'un assemblage complexe de cellules cubiques, aux propriétés optiques (absorption et émissivité) fixées. Le parcours des rayons est ensuite calculé conformément à l'équation du transfert radiatif, avec une approche semblable à celle du modèle précédent.

### 1.2.2 Le rendu d'images par lancer de rayons : approche et méthodes

On désigne par rendu d'images toutes les méthodes visant à transformer des données géométriques abstraites en représentation visuelle. Ces méthodes se basent sur des approximations, plus ou moins rigoureuses, des phénomènes physiques liés à la propagation lumineuse. Nous présentons ici le principe général de beaucoup de ces approches, puis nous détaillons certaines de ces méthodes, fondées du point de vue physique.

#### Le rendu d'images comme solution d'une équation

Une manière très classique de présenter formellement le rendu d'images est de le considérer comme la résolution d'une équation, dite équation de rendu. Cette équation fut introduite en 1986 par James T. Kajiya [Kajiya 86]. Elle a ensuite été largement reprise [Keller 01, Szirmay-Kalos 00, Uni 03].

Il s'agit d'une équation permettant la génération d'une image composée de pixels, chaque pixel correspondant à un angle solide autour d'un direction. L'équation de rendu formalise l'estimation de la radiance de chaque pixel ; c'est à dire de l'intensité lumineuse émise par la surface du domaine visible «tombant» dans l'angle solide correspondant.

La force de l'équation de rendu est de proposer une formulation physiquement basée de cette radiance, en fonction des propriétés de toute la scène. Une hypothèse fondamentale permet de rendre la formulation finale éminemment simple et efficace : il s'agit de l'hypothèse de considérer le milieu de propagation (l'air dans l'immense majorité des cas) comme non participatif , c'est a dire n'absorbant ni ne courbant le trajet des radiations le traversant.

La radiance incidente au point $x$ selon la direction $\omega$ correspond alors à la radiance de la surface observée selon cette direction :

$$L_{in}(x,\omega) = L(h(x,\omega), -\omega) \tag{1.6}$$

Où $h : \mathbb{R}^3 \times \Omega \rightarrow \mathbb{R}^3$ est une «fonction de visibilité», c'est à dire une application associant le premier point appartenant à une surface observé depuis un point donné selon une direction $\omega$ donnée, appartenant à l'ensemble du domaine visible $\Omega$.

Or, la radiance émise par une surface peut être décomposée en radiance directement émise et radiance réfléchie par la surface. La principale difficulté est d'exprimer cette dernière quantité.

Pour ce faire, définissons les notations suivantes :

| | |
|---|---|
| $\Phi_r(x, \omega, d\omega)$ | est le flux réfléchi en $x$ dans l'angle solide élémentaire $d\omega$ autour de la direction $\omega$ $(W)$ |
| $\Phi_{in}(x, \omega', d\omega')$ | est le flux incident en $x$ dans l'angle solide élémentaire $d\omega'$ autour de la direction $\omega'$ $(W)$ |
| $w(x, \omega', \omega)$ | est la probabilité qu'un photon incident en $x$ avec la direction $\omega'$ en reparte avec la direction $\omega$ |
| $L_r(x, \omega)$ | est la radiance émise après réflexion en $x$ selon la direction $\omega$ |
| $L_{in}(x, \omega')$ | est la radiance reçue en $x$ selon la direction $\omega'$ |
| $dA$ | est la surface infinitésimale supportant $x$ |
| $\theta$ (resp. $\theta'$) | est l'angle entre la direction $\omega$ (resp. $\omega'$) et la normale à la surface supportant $x$ |

Ces définitions nous permettent d'exprimer le flux réfléchi comme la somme des flux incidents, pondérés par leur probabilité d'être réfléchis vers la direction d'intérêt selon l'angle solide $d\omega$ :

$$\Phi_r(x, \omega, d\omega) = \int_{\Omega'} \Phi_{in}(x, \omega', d\omega') \left(w(x, \omega', \omega) d\omega\right) \tag{1.7}$$

Par définition, la radiance étant la puissance dirigée dans un angle solide donné traversant une surface donnée, on a :

$$\begin{aligned} \Phi_r(x, \omega, d\omega) &= L_r(x, \omega) dA \cos\theta d\omega \\ \Phi_{in}(x, \omega', d\omega') &= L_{in}(x, \omega') dA \cos\theta' d\omega' \end{aligned} \tag{1.8}$$

La combinaison de (1.7) et (1.8) nous permet d'obtenir :

$$L_r(x, \omega) dA \cos\theta d\omega = \int_{\Omega'} L_{in}(x, \omega') dA \cos\theta' d\omega' \left(w(x, \omega', \omega) d\omega\right) \tag{1.9}$$

et finalement

$$L_r(x, \omega) = \int_{\Omega'} L_{in}(x, \omega') \frac{w(x, \omega', \omega)}{\cos\theta} \cos\theta' d\omega' \tag{1.10}$$

En posant $L(x, \omega) = L_e(x, \omega) + L_r(x, \omega)$, et en remarquant que la radiance $L_{in}(x, \omega')$ reçue en $x$ selon $\omega'$ correspond à la radiance émise selon $-\omega'$ au point observé $h(x, \omega')$, on obtient une expression récursive de la radiance :

$$L(x, \omega) = L_e(x, \omega) + \int_{\Omega'} L(h(x, \omega'), -\omega') \frac{w(x, \omega', \omega)}{\cos\theta} \cos\theta' d\omega' \tag{1.11}$$

On pose en général $f_r(x, \omega', \omega) = \frac{w(x, \omega', \omega)}{\cos\theta}$, et on appelle cette quantité Bidirectionnal Reflection Density Function (BRDF). L'estimation de cette fonction est un point crucial de la résolution de l'équation de rendu.

### Paramétrisations des propriétés optiques des surfaces

De nombreuses BRDF ont été proposées dans la littérature. Ces fonctions se basent sur des paramétrisations des surfaces, permettant la prise en compte d'une infinité d'états intermédiaires entre la réflexion spéculaire et la réflexion lambertienne [Sylvain 05]. Les nombreuses fonctions proposées par les chercheurs d'horizons divers sont grossièrement classables en trois catégories :

1. les modèles dérivés des lois de la physique, aux paramétrisations complexes ;

2. les modèles reposant sur une paramétrisation simple, grâce à des simplifications fortes. Ces modèles restent cependant cohérents avec les lois physiques fondamentales[4] ;
3. les modèles empiriques, violant une ou plusieurs des lois physiques fondamentales. Le principal modèle de ce type est le modèle de B.T. Phong, qui a été le premier modèle proposé en rendu d'image pour les surfaces quasi-spéculaires [Szirmay-Kalos 00]. Du fait de leur non-plausibilité physique, ces modèles ne seront pas plus détaillés ici.

La majorité de ces modèles sont basés sur les lois physiques, couplées à une simplification des surfaces : la surface est considérée comme un ensemble de facettes aléatoirement orientées, ou supportant des cavités cylindriques de profondeur infinie [Lafortune , Ozturk 08, Meister 98, He 91]. Partant de tels modèles, il est possible, moyennant simplification, d'élaborer des paramétrisation intuitives [Schlick 94].

Certains auteurs développent plutôt des méthodes permettant la construction empirique de BRDF pour divers matériaux sur la base de mesures [Atkinson 08].

### Les méthodes de Monte-Carlo dans la solution de l'équation de rendu

Nous avons vu que le problème de l'estimation de la radiation reçue en un point était formalisée sous la forme d'une équation intégrale (voir paragraphe 1.2.2). La valeur de cette intégrale dépend de la géométrie de la scène, des propriétés des surfaces et, bien sûr, de la position et de l'intensité des sources lumineuses. Diverses méthodes ont été proposées pour résoudre cette équation [Szirmay-Kalos 00, Kajiya 86] : par éléments finis, décomposition en séries de Neumann...

Nous détaillons ici brièvement certaines des méthodes utilisant un lancer de rayons, en nous efforçant de faire ressortir les différences d'approches qui existent au sein de ces méthodes.

Les algorithmes se décomposent en :

1. les algorithmes pour lesquels le point de tir est la source lumineuse (ponctuelle)
2. les algorithmes pour lesquels le point de tir est le récepteur. On distingue :
   - les algorithmes de *ray casting* : un tir sans réflexion est effectué depuis le récepteur, et l'illumination par les sources (ponctuelles) des points atteints est testée par le tir d'un rayon vers ces sources. Une amélioration de ces algorithmes est constituée par les algorithmes de *visibility ray tracing* : les réflection et réfractions idéales sont considérées.
   - Les algorithmes de tir distribué, prenant en compte les BRDF non triviales.

La prise en compte des réflexions peut se faire de diverses manières :
- en affectant un poids à chaque chemin tracé, correspondant à la proportion de la radiation lumineuse arrivant effectivement au récepteur depuis la source par le chemin tracé ;
- en considérant des photons uniques, avec un mécanisme de «roulette russe» : la BRDF est considérée comme une probabilité d'absorption de la particule. L'arrêt ou la continuation du tir sont décidés par un tirage pseudo-aléatoire.

[4] ces lois incluent principalement la loi de conservation de l'énergie et le principe de symétrie de Helmholtz (postulant la réflexivité de la BRDF)

### 1.2.3 Problèmes algorithmiques

Le lancer de rayons implique, on l'a vu, le test de l'intersection avec les différentes surfaces constituant la scène. Un algorithme de test de l'intersection avec un triangle, proposé par Tomas Möller et Ben Trumbore, est détaillé [Möller 97]. Cet algorithme peut bien sûr ensuite servir pour d'autres surfaces polygonales, par décomposition en triangles [Lagae 05].

Le test des intersections avec tous les triangles d'une scène est cependant une opération très coûteuse : divers codages permettent de réduire cette complexité. L'un de ces codages est expliqué ici. Il s'agit du référencement par octree, que nous utilisons dans notre implémentation (voir 3.1.2).

#### Tests d'intersection

Un algorithme de test d'intersection, proposé par Tomas Möller et Ben Trumbore, repose sur l'utilisation des coordonnées barycentriques d'un point dans un triangles. Avec $m = (x_m, y_m, z_m)$ un point dans le triangle de sommets $s_i = (x_i, , y_i, z_i),\ i \in 1, 2, 3$, ces coordonnées $(u, v)$ s'expriment :

$$m = (1 - u - v)s_1 + us_2 + vs_3 \tag{1.12}$$

où

$$\begin{cases} u \geq 0 \\ v \geq 0 \\ u + v \leq 1 \end{cases} \tag{1.13}$$

Avec $O$ l'origine du rayon, et $\omega$ sa direction, la distance $t$ à l'intersection, si elle existe, obéit à :

$$O + \omega t = (1 - u - v)s_1 + us_2 + vs_3 \tag{1.14}$$

Ce qui donne le système linéaire suivant (avec tous les vecteurs $\omega$, $s_i$ et $O$ exprimés en colonne) :

$$\begin{pmatrix} -\omega & (s_2 - s_1) & (s_3 - s_1) \end{pmatrix} \begin{pmatrix} t \\ u \\ v \end{pmatrix} = O - s_1 \tag{1.15}$$

Qui permet d'obtenir, si l'intersection existe, sa distance à l'origine.

#### Subdivision de l'espace

Une représentation standard des surfaces 3D en machine se fait sous la forme de triangulations, dans laquelle toute forme est approximée par un ensemble de triangles. Nous avons vu comment calculer l'intersection d'un rayon avec un triangle ; la réalisation d'un tel test avec tous les triangles d'une scène est cependant extrêmement coûteux. Nous remarquons notamment que la présence d'une surface courbe fait exploser le nombre de triangles (voir fig. 1.2).

Une manière de diminuer le nombre de calculs d'intersection à effectuer est de subdiviser l'espace, pour ne faire les calculs que sur les triangles contenus dans le sous-espace courant.

Une manière très commode de stocker une subdivision de l'espace est de la représenter comme un arbre : chaque nœud de l'arbre correspond à une partie du domaine étudié ; la fonction père correspond à la relation d'inclusion, et l'ensemble des sous-domaines

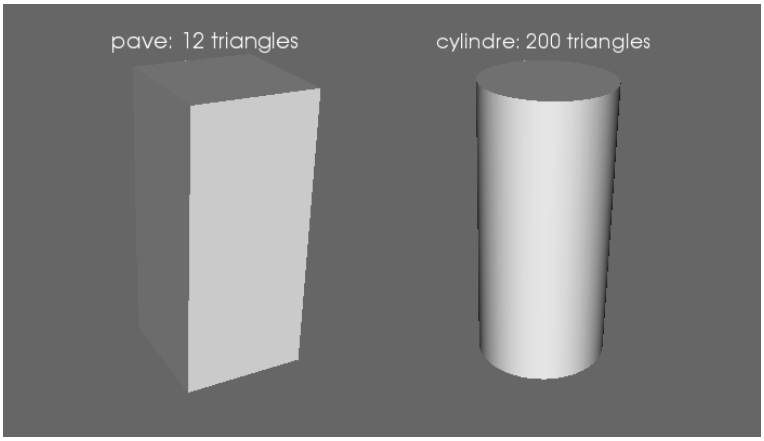



FIG. 1.2 – Comparaison du nombre de triangles composant la représentation d'un pavé et celle d'un cylindre

correspondant aux fils d'un nœud forme une partition du sous-domaine correspondant au nœud père[5].

Une forme d'arbre très utilisée est celle d'octree : tout nœud non terminal possède exactement huit fils [Chen 88, Aronov 05, Brönnimann 06]. Un tel codage permet une navigation facile, la recherche de voisins se faisant par parcours d'arbre [Samet 89].

Remarquons que ce type particulier d'arbre peut être utilisé à d'autres fins : création d'un maillage hexaédrique [Maréchal 04] ou référencement des surfaces dans une géométrie codée par des voxels[6] [Juan-Arinyo 95] par exemple.

Dans le cas qui nous intéresse, le principe de construction est le suivant [Chen 88, Aronov 05] :

- Un cube englobant toute la scène est défini, et associé à la racine de l'octree ;
- ce cube est divisé en huits cubes fils, nommés octants, à leur tour subdivisés ;
- la subdivision s'arrête lorsqu'un critère est atteint : nombre de subdivisions successives, nombre d'éléments contenus dans les octants terminaux[7]...

Un des principaux intérêts de cette méthode de construction est qu'elle permet l'adaptation locale de la résolution à la complexité locale de l'information (voir fig. 1.3).

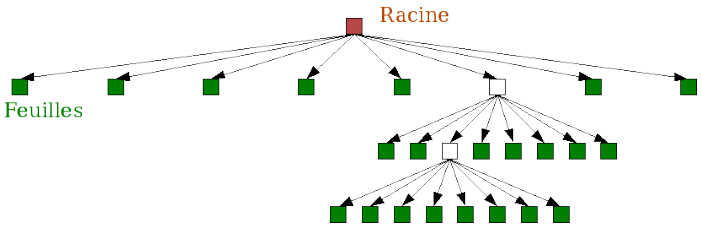



FIG. 1.3 – exemple de représentation d'un octree avec adaptation locale de la résolution

[5] ces notions sont détaillés de manière moins théorique ci-après, avec la présentation de la subdivision en octree

[6] équivalents 3D des pixels 2D bien connus.

[7] dans la suite de ce rapport, ces octants seront systématiquement désignés comme *feuilles*.

Une méthodologie de recherche de feuille voisine, sachant par quel face/arrête/coin sort le rayon, peut alors être élaborée sous la forme d'un parcours d'arbre [Samet 89].

## 1.3 Méthodes de calcul de la radiation solaire reçue en contexte urbain

Diverses approches peuvent être utilisées pour quantifier la radiation reçue en contexte urbain.

Une méthode proposée par Daren Robinson repose sur l'utilisation de modèles de ciel cumulés (correspondant à la moyenne sur l'année) [Robinson 04]. Conformément aux standards de la CIE, le modèle de ciel est discrétisé en 145 patchs (voir fig. 1.4). Dans le «modèle de radiosité simplifié» proposé par Daren Robinson, l'obstruction des patchs est ensuite calculée. La radiation reçue par la surface de calcul est alors calculée.

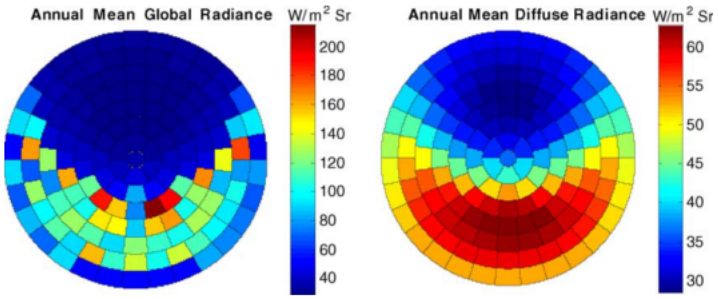



FIG. 1.4 – Exemple de modèle de ciel cumulé

De nombreux modèles de ciel ont étés proposés[8]. Leur méthode de construction peut grandement varier. Schématiquement, nous pouvons distinguer :

1. Les modèles empiriques, basés sur la synthèse statistique de mesures au sol ;
2. les modèles théoriques, dérivés des lois physiques, par le biais d'une paramétrisation de l'atmosphère [Kocifaj 09] ;
3. les modèles mixtes, dont la construction est principalement basée sur des mesures mais se veut adaptable à divers contextes [Igawa 04].

[8] pour le seul ciel clair, Viorel Badescu en compare 52 [Badescu 10] !

Chapitre 2

# Vers un algorithme d'estimation du rayonnement solaire : délimitation du problème et modélisation adoptée

## 2.1 Estimation de la radiation solaire en contexte urbain : délimitation du problème et formalisation

Le but de ce projet est de concevoir un algorithme d'estimation du rayonnement solaire, à la fois précis et utilisable dans le cadre d'études en contexte urbain. Les ambitions générales de cet algorithme sont les suivantes :

- permettre l'estimation des deux grandeurs fondamentales de la lumière : irradiance (aspect énergétique) et illuminance (aspect visuel) ;
- permettre la prise en compte de réflexions multiples, en prenant en compte les caractéristiques optiques des matériaux ;
- permettre le calcul sur diverses périodes, sous diverses conditions météorologiques.

Ces ambitions doivent cependant s'insérer dans la finalité du programme réalisé : être utilisable dans un *cadre de conception*. Dans ce but, il est fondamental de suivre les règles suivantes :

- les paramétrisations adoptées pour les propriétés de surfaces et du ciel se doivent d'être *simples* et *intuitives*, afin de permettre une utilisation du programme face au manque de données empiriques ;
- le temps de calcul doit rester raisonnable.

Pour ces raisons, seuls les phénomènes de réflexion sur des surfaces solides seront considérés dans la variation du trajet lumineux : les phénomènes de réfraction, réflexion ou diffusion, dans l'atmosphère notamment, ne sont pas pris en compte. De tels phénomènes sont cependant responsable de ce qu'il est convenu d'appeler la «radiation diffuse », c'est à dire la portion de radiation solaire parvenant au sol depuis une direction différente de celle du disque solaire.

Afin de prendre en compte cette radiation, la source lumineuse considérée n'est pas le soleil, mais la voûte céleste dans son ensemble : l'estimation de la composante diffuse de la radiation solaire est ainsi laissée à un modèle de ciel. Ceci permet par ailleurs de se reposer sur le modèle de ciel pour d'autres aspects :

- la sélection du domaine spectral sur lequel est effectué le calcul — un calcul par sous-domaine spectral devra se faire par la donnée de divers modèles de ciel ;
- la prise en compte de la nébulosité ;
- la sélection de la quantité estimée — luminance, radiance . . .

Les surfaces, enfin, sont considérées comme non émettrices de radiations : leur comportement se résume à absorber une partie de la radiation et à en réfléchir le reste.

La section suivante formalise cette approche, permettant de déboucher sur un algorithme de résolution immédiatement implémentable.

## 2.2 Vers un algorithme de résolution : principes du calcul et hypothèses

### 2.2.1 Du problème général à la formulation intégrale

Comme vu en 1.2.2, une approche classique en rendu d'images est la considération du problème d'estimation de la lumière reçue comme solution d'une équation : tous les algorithmes de rendu d'images peuvent être formalisés comme autant de méthodes de résolution de cette équation.

Une approche similaire est ici adoptée.

On cherche à estimer, en divers points d'un maillage, le flux radiatif incident. Comme vu en 1.2.2, ce problème revient à estimer la valeur d'une expression intégrale de dimension non connue. La quantité estimée aux points du maillage est l'irradiance[1], soit la somme des radiances du domaine visible :

$$I(x) = \int_{\Omega} L(h(x,\omega), -\omega) d\omega \tag{2.1}$$

où :

| | |
|---|---|
| $I(x)$ | est l'irradiance reçue au point $x$ ($\mathrm{W.m^{-2}}$) |
| $\Omega$ | est le domaine angulaire visible depuis $x$ |
| $h : \mathbb{R}^3 \times \Omega \to \mathbb{R}^3$ | est la «fonction de visibilité», c'est à dire une application associant le premier point appartenant à une surface observé depuis un point donné selon une direction donnée |
| $L(x,\omega)$ | est la radiance au point $x$ selon la direction $\omega$ ($\mathrm{W.m^{-2}.sr^{-1}}$) |

Pour chaque direction $\omega$, deux cas peuvent alors se présenter : soit le point visible appartient au ciel, soit il appartient à une autre surface. Dans le cas où le point appartient au ciel, la radiance est donnée par le modèle de ciel. Sinon, reprenant les expressions rencontrées en graphisme (voir l'équation 1.11 p.8), la radiance s'exprime comme une fonction de la radiation reçue. En ne considérant que des surfaces non émettrices, nous simplifions l'expression 1.11 en :

$$L(x,\omega) = \int_{\Omega'} L(h(x,\omega'), -\omega') f_r(x,\omega,\omega') \cos\theta' d\omega' \tag{2.2}$$

où $\theta'$ est l'angle entre la direction $\omega'$ et la normale à la surface.

#### Choix de la BRDF

L'expression choisie pour $f_r$ est celle développée par Christophe Schlick [Schlick 94]. Les raisons de ce choix sont les suivantes :

- il s'agit d'une expression obéissant aux principes physiques fondamentaux (loi de conservation de l'énergie, principe de symétrie de Helmholtz) ;
- les paramètres utilisés pour caractériser les surfaces sont peu nombreux (trois seulement) et intuitifs ;

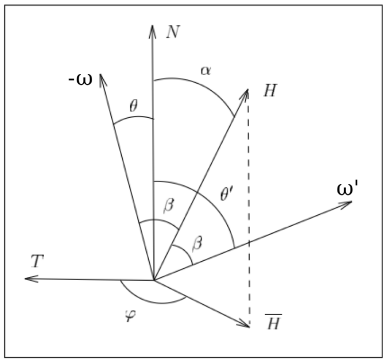
[1] remarquons que tout ce qui va être dit ici sous les termes de radiance et irradiance est transposable dans le domaine visuel (luminance et illuminance), le passage de la radiance à la luminance consistant à l'affectation d'un poids variable aux différentes longueurs d'ondes [Szirmay-Kalos 00, Igawa 04]. Les phénomènes en jeu sont donc les mêmes, seule la donnée d'entrée diffère.

– la même expression permet de varier continument de la réflexion spéculaire à la réflexion lambertienne.

À dire vrai, Christophe Schlick propose deux expressions : l'une pour des matériaux dits «simples», l'autre pour des matériaux dits «doubles». Cette deuxième expression sert pour les matériaux hétérogènes ; dans le cadre de notre formalisation, nous nous contenterons cependant de l'expression pour les matériaux simples. Cela permet de ne pas multiplier les paramètres à entrer. Cette approche nous parait tout à fait pertinente, au vu des bons résultats obtenus par certains auteurs en contexte urbain, en ne considérant que la réflexion lambertienne [Kastendeuch 09].

Les paramètres servant à décrire les surfaces sont les trois réels compris entre 0 et 1 suivants :

1. $C_\lambda$, le facteur de réflexion pour la longueur d'onde $\lambda$ ;
2. $r$, le facteur de rugosité (variant de 0 pour une réflexion spéculaire à 1 pour une réflexion lambertienne), quantifiant la dispersion selon l'écart zénithal entre direction incidente et direction réfléchie ;
3. $p$, le facteur d'isotropie (variant de 0 pour une surface parfaitement anisotropique à 1 pour une surface parfaitement isotropique), quantifiant la dispersion selon l'écart azimutal entre direction incidente et direction réfléchie.




ω: direction incidente
ω': direction de réflection

N: normale à la surface
T: tangente à la surface

H: normale à la "facette" de réflexion
H̄: composante de H orthogonale à N

FIG. 2.1 – notation des différentes quantités géométriques pour la BRDF

L'expression proposée par Christophe Schlick prend la forme suivante :

$$f_r(x,\omega,\omega') = \underbrace{\left(C_\lambda + (1-C_\lambda)(1-\cos\beta)^5\right)}_{S_\lambda(\omega,\omega')} \underbrace{\frac{1}{4\pi\cos\theta\cos\theta'} Z(\omega,\omega')A(\omega,\omega')}_{D(\omega,\omega')} \tag{2.3}$$

où $S_\lambda(\omega,\omega')$ est le «facteur spectral», et $D(\omega,\omega')$ est un «facteur de direction», décomposé en dépendance à l'angle zénithal :

$$Z(\omega,\omega') = \frac{r}{(1 + r\cos^2\alpha - \cos^2\alpha)^2} \tag{2.4}$$

et dépendance à l'azimuth :

$$A(\omega,\omega') = \sqrt{\frac{p}{p^2 - p^2\cos^2\varphi + \cos^2\varphi}} \tag{2.5}$$

Remarquons que l'expression retenue pour $S_\lambda(\omega,\omega')$, qui est une approximation de la loi de Fresnel, ne permet pas la prise en compte de matériaux absorbant totalement la radiation : ce terme sera non nul même pour $C_\lambda = 0$.

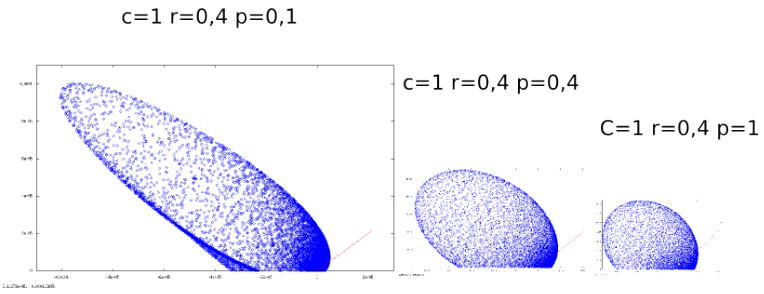



FIG. 2.2 – évolution de la répartition des rayons avec la variation du facteur d'isotropie

### 2.2.2 Principes de résolution de la formulation intégrale

Le problème de l'estimation a été mis sous une forme intégrale bien définie, qu'il nous faut maintenant résoudre. La méthode adoptée consiste en un tir récursif de rayons depuis la surface de calcul. Le principe général, en chaque point du maillage de calcul, est le suivant :

1. des rayons sont lancés selon une distribution uniforme, depuis la surface de calcul, en testant l'intersection avec une surface du contexte 3D ;
2. si une surface est rencontrée, un nouveau tir est effectué depuis le point d'intersection. Les rayons sont affecté d'un poids, déterminé grâce à la BRDF ;
3. si aucune intersection n'est rencontrée, le point de la voûte céleste pointé est retourné ;
4. les orientations, les coefficients associés, ainsi que les points d'intersection sont stockés ;
5. les orientations et leurs coefficients sont passées à un modèle de radiance du ciel, puis sommées pour obtenir l'irradiance due à la radiation diffuse ;
6. un tir est fictivement effectué pour déterminer l'irradiance due à la radiation directe (voir 2.2.2, p.19) ;
7. les irradiances sont intégrées sur la surface, permettant d'obtenir le flux moyen reçu.

Le détail de la méthode est présenté ci-après, et sa validité est prouvée.

#### estimation de la radiance par lancer de rayons

Nous avons formalisé notre problème sous la forme de deux relations intégrales : l'équation (2.1), définissant l'irradiance en un point du maillage en fonction de la radiance du domaine visible ; l'équation (2.2), définissant la radiance réfléchie par une surface dans une direction donnée, en fonction de la radiance du domaine visible.

Nous avons par ailleurs vu en 1.1.1 qu'une intégrale est facilement interprétable en termes d'espérance mathématique d'une variable aléatoire. Cette espérance est alors estimable par la moyenne des réalisations d'une variable aléatoire.

Ici, l'application de cette méthode nous donne alors un estimateur pour chacune des deux relations intégrales : l'équation (2.1)

$$I(x) = \int_{\Omega} L(h(x,\omega), -\omega) d\omega \tag{2.6}$$

devient

$$I(x) = \mathbb{E}\left(\frac{L\left(h\left(x, W_0\right), -W_0\right)}{d_0\left(W_0\right)}\right) \tag{2.7}$$

avec $W_0$ une variable aléatoire de loi de densité de probabilité $d_0$. Cette quantité est estimée par :

$$\hat{I}(x) = \sum_{i=1}^{n_0} \left( \frac{1}{n_0 \times d_0\left(w_{0,i}\right)} \hat{L}\left(h\left(x, w_{0,i}\right), -w_{0,i}\right) \right) \tag{2.8}$$

Avec $\hat{L}$ l'estimateur de la radiance dérivé de l'équation (2.2)

$$L(x,\omega) = \int_{\Omega'} L(h(x,\omega'), -\omega') f_r(x,\omega,\omega') \cos\theta' d\omega' \tag{2.9}$$

qui devient

$$L(x,\omega) = \mathbb{E}\left[L\left(h\left(x, W_j\right), -W_j\right) f_r\left(x,\omega, W_j\right) \cos\theta(W_j)\right] \tag{2.10}$$

et est donc estimée par

$$\hat{L}(x,\omega) = \sum_{k=1}^{n_j} \left( \frac{f_r\left(x,\omega,w_{j,k}\right)\cos\theta(w_{j,k})}{n_j \times d_j(w_{j,k})} L\left(h\left(x, w_{j,k}\right), -w_{j,k}\right) \right) \tag{2.11}$$

Ces relations exhibent autant de lois de densité de probabilité et de cardinaux générés que l'on effectue de tirs. Dans la pratique, nous fixons cependant le même nombre de rayons et la même loi de densité de probabilité pour tous les tirs.

**génération pseudo-aléatoire : densité et stratification**

Dans notre problème, les valeurs à générées sont des orientations dans l'espace, représentées par des vecteurs unitaires.

Un choix naturel dans l'estimation d'une intégrale est de choisir une variable aléatoire de distribution uniforme. Afin de réduire la variance du résultat, des valeurs sont générés uniformément sur la demi-sphère unité par jittered sampling.

La génération des points se fait de la manière suivante :

soient $\gamma$ et $\zeta$ deux vecteurs de $m^2$ valeurs sur $[0,1]$, de loi de densité de probabilité uniforme. En décomposant le carré unité en $m^2$ sous-domaines carrés de côté $h = \frac{1}{m}$, une valeur par sous-domaine est obtenue par (voir fig 2.3) :

$$\begin{cases} s_k = [((k-1)\%m) + \gamma_k]h & \forall k = 1, \ldots, m^2 \\ t_k = [((k-1)/m) + \zeta]h & \forall k = 1, \ldots, m^2 \end{cases} \tag{2.12}$$

Où «$a/b$» représente la division entière de $a$ par $b$ et «$a\%b$» représente le reste de cette division.

Une paramétrisation conservant les aires, proposée par Jim Arvo [Uni 03], est ensuite utilisée pour obtenir une répartition sur la demi-sphère unité :

$$\omega_k = \begin{pmatrix} \sqrt{s_k(2-s_k)}\cos(2\pi t_k) \\ \sqrt{s_k(2-s_k)}\sin(2\pi t_k) \\ 1 - s \end{pmatrix} \forall k = 1, \ldots, m^2 \tag{2.13}$$

L'ensemble d'orientations ainsi généré est ensuite orienté selon la normale à la surface de tir.

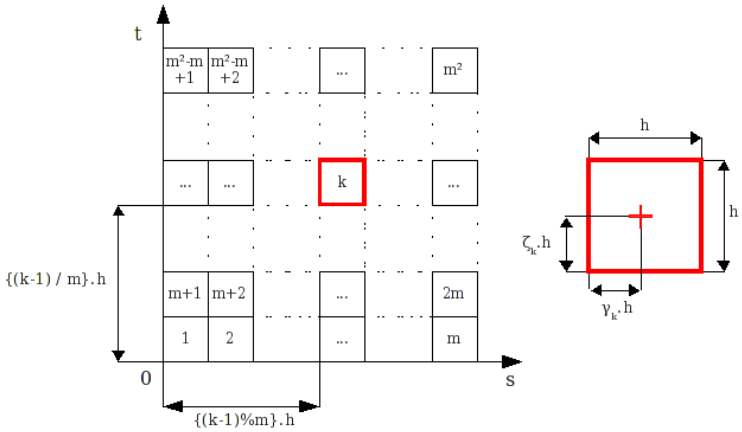



FIG. 2.3 – génération de points par jittered sampling sur le carré unité

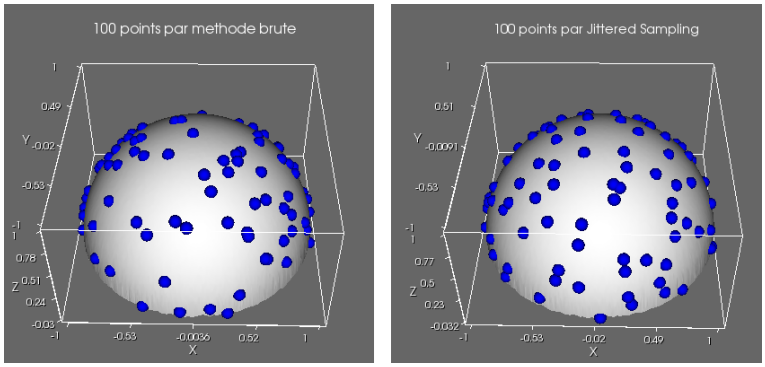



FIG. 2.4 – Exemple de génération pseudo-aléatoire sur la demi-sphère unité.

**algorithme de tir**

L'estimation de l'irradiance en chaque point se fait en trois étapes. D'abord, un lancer de rayons est effectué selon l'algorithme suivant, appelé «lancer de rayons distribué» [Szirmay-Kalos 00, L. Cook 84] :

1. un ensemble d'orientations, affectées du coefficient défini par l'équation (2.8), est généré. La première orientation est choisie ;
2. l'intersection de la demi-droite générée par l'orientation depuis le point de tir avec le contexte 3D est testée ;
3. si aucune intersection n'est trouvée, l'orientation, son coefficient ainsi que le point de tir correspondant sont stockés comme points de la voûte céleste ;
4. sinon, le point d'intersection, l'orientation incidente et le coefficient associé sont stockés. Si le nombre maximal de réflexions fixé n'est pas atteint, un nouvel ensemble d'orientations est généré, affectées du produit du coefficient initial et du coefficient défini par l'équation (2.11). La première des orientations est choisie, et l'on retourne à l'étape 2 ;
5. on s'arrête lorsque toutes les orientations ont été testées.

Les orientations sont ensuite passées au modèle de ciel, qui retourne les radiances correspondant à la *radiation diffuse* en ces points. D'après les relations (2.8) et (2.11), l'irradiance due à la radiation diffuse au point est la somme des radiances célestes, pondérées par les coefficients obtenus.

Il reste encore à estimer l'irradiance due à la radiation directe. Les orientations obtenues précédemment ne permettent cependant pas d'estimer correctement cette radiation, du fait de la faible probabilité qu'une orientation aléatoire pointe le disque solaire. Cette part de la radiation est pourtant prédominante —par ciel clair notamment—, et sa prise en compte est donc essentielle.

La solution adoptée consiste en un «tir fictif» depuis les différents points de réflexion identifiés. En voici le principe, appliqué à tous les points d'intersection ainsi qu'aux nœuds du maillage :

1. le pointage du disque solaire par une ou plusieurs des orientations originaires du point est testé. Si l'ensemble de telles orientations est non vide, il est retourné avec les coefficients associés ;
2. sinon, l'intersection entre la droite pointant le soleil et le contexte 3D est effectué. Si aucune intersection n'est trouvée, l'orientation solaire, pondérée par le produit du coefficient au point et du coefficient défini par l'équation (2.11), est retournée.

Par cette procédure, un nouveau lancer de rayons est fictivement effectué, avec le même nombre de tirs que lors du premier. La méthode adoptée permet cependant de s'assurer, sans modifier en rien la loi de densité de probabilité, que la radiation directe est prise en compte en tous les points irradiés.

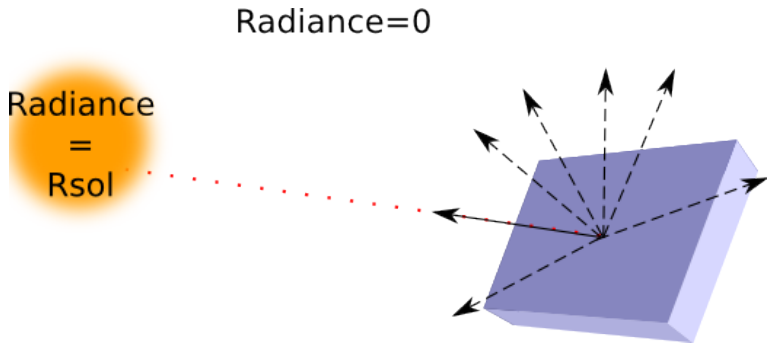



FIG. 2.5 – Tir fictif : seules les orientations pointant une radiance non nulle sont calculées

### Calcul de la radiance céleste : choix du modèle de ciel

Il vient d'être dit que l'estimation de la radiation s'appuyait sur un modèle de ciel. Il a été vu en 1.3 que de tels modèles sont nombreux. Il a été choisi ici d'adopter le All Sky Model de Igawa *et al.*, pour plusieurs raisons :

- il s'agit d'un modèle prévu pour être utilisé dans n'importe quelle partie du globe ;
- il s'agit d'un modèle analytique, qui permet donc d'avoir immédiatement des radiances ponctuelles, ce que demande la méthodologie définie ;
- il s'agit, comme son nom l'indique, d'un modèle «tout type de ciel», permettant de varier continûment d'un ciel couvert à un ciel dégagé ;
- la paramétrisation des types de ciel est simple, et peut être déterminée à partir de mesure de la radiation directe et de la radiation diffuse.

Cette paramétrisation des types de ciel s'appuie sur deux paramètres : l'«indice de ciel clair » (*Clear Sky Index*) $K_c$ et l'«indice de nébulosité » (*Cloudless Index*) $Cle$. L'indice de ciel clair est défini comme le rapport de l'irradiance globale à une irradiance globale «standard» ; l'indice de nébulosité est défini à partir du rapport entre irradiance

diffuse et directe :

$$K_c = \frac{Eeg}{Seeg}$$
$$Cle = \frac{1 - Ce}{1 - Ce_s} \tag{2.14}$$

où :

| | |
|---|---|
| $Eeg$ | est l'irradiance globale |
| $Seeg$ | est l'irradiance globale standard, correspondant à l'irradiance globale théorique maximale |
| $Ce = \frac{Eed}{Eeg}$ | est la «proportion de nuages » (*Cloud Ration*), définie comme le rapport de l'irradiance diffuse $Eed$ à l'irradiance globale $Eeg$ |
| $Ce_s$ | est une proportion de nuages standard. |

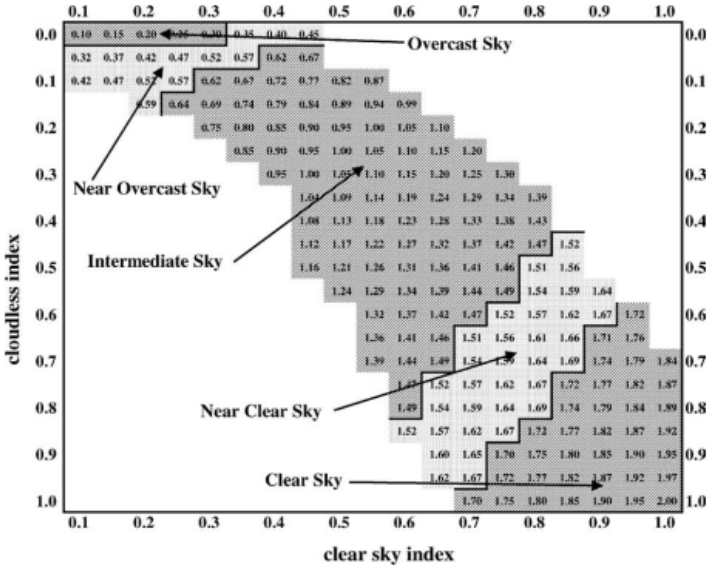



FIG. 2.6 – Relation entre les paramètres du All Sky Model et le type de ciel

Comme le montre la figure 2.6, ces deux paramètres permettent de se déplacer continument dans l'espace des types de ciel. Si les paramètres sont immédiatement fixés (et non déterminés à partir de mesures d'irradiance), leur formulation permet de plus de retrouver la valeur de l'irradiance directe : la radiance du disque solaire peut alors être calculée pour la phase de «tir fictif».

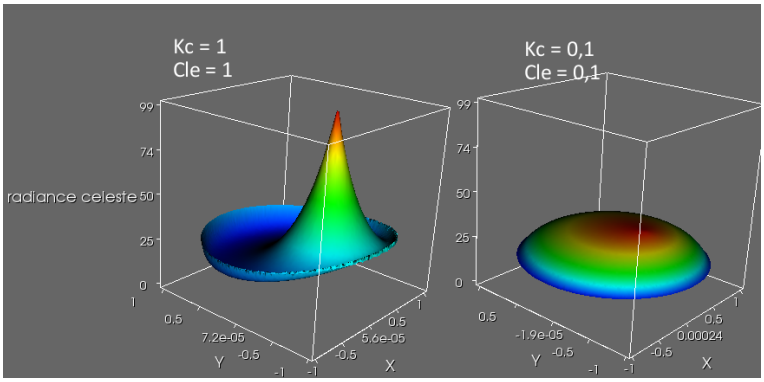



FIG. 2.7 – représentation de la radiance diffuse calculée par le All Sky Model au 21 juin à midi, par 45 ° de latitude Nord, pour deux valeurs des paramètres

La position solaire selon la date, la latitude et l'heure peut être calculée de diverses manières. Celle qui a été retenue est celle proposée par Mindjid Maïzia dans son cours TU01 [Maïzia 07].

### Intégration sur la surface de calcul

Après obtention des irradiances aux points du maillage, une intégration des valeurs obtenues est effectuée. La méthodologie est la suivante :

1. une intégration est effectuée sur chaque triangle du maillage, par le biais d'une interpolation bilinéaire ;
2. les valeurs par triangle sont sommées pour obtenir une valeur globale.

Ces deux étapes se font en fait simultanément, grâce à un changement de variables *ad hoc* [Quarteroni 00].

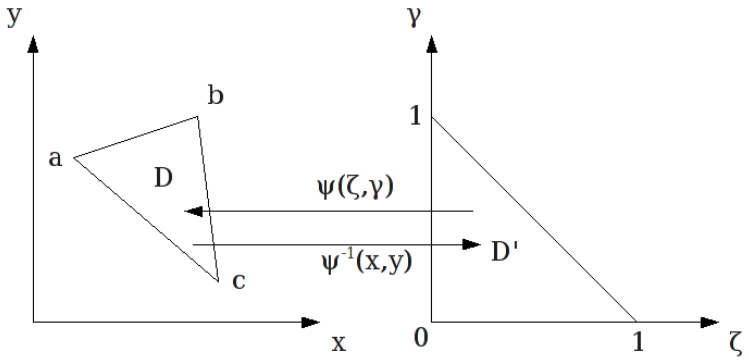



FIG. 2.8 – Changement de variables pour l'intégration

Soient D le domaine triangulaire initial, D' le triangle rectangle de côté 1, et $\psi$ : $D' \rightarrow D$ la transformation bijective permettant le passage de $D'$ à $D$, d'expression :

$$\psi \begin{pmatrix} \zeta \\ \gamma \end{pmatrix} = \underbrace{\begin{pmatrix} x_b - x_a & x_c - x_a \\ y_b - y_a & y_c - y_a \end{pmatrix}}_{T_\psi} \begin{pmatrix} \zeta \\ \gamma \end{pmatrix} + \begin{pmatrix} x_a \\ y_a \end{pmatrix} \tag{2.15}$$

Avec $x$, $y$ les coordonnées dans le plan défini par le triangle et $\zeta$, $\gamma$ les nouvelles coordonnées, dites coordonnées barycentrique, et $a$, $b$, $c$ les sommets du triangle.

Soit par ailleurs $L_{bil}$ une approximation bilinéaire de la radiance sur $D$, construite de la manière suivante :

$$L_{bil} = \begin{pmatrix} N_a(x,y) & N_c(x,y) & N_c(x,y) \end{pmatrix} \begin{pmatrix} L_a \\ L_b \\ L_c \end{pmatrix} \tag{2.16}$$

où :

| | |
|---|---|
| $N_a(x,y)$ (resp. $N_b$, $N_c$) | est une fonction bilinéaire valant 1 en $a$ (resp. $b$, $c$) et 0 aux autres sommets du triangle. |
| $L_a$ (resp. $L_b$, $L_c$) | est la radiance au point $a$ (resp. $b$, $c$) |

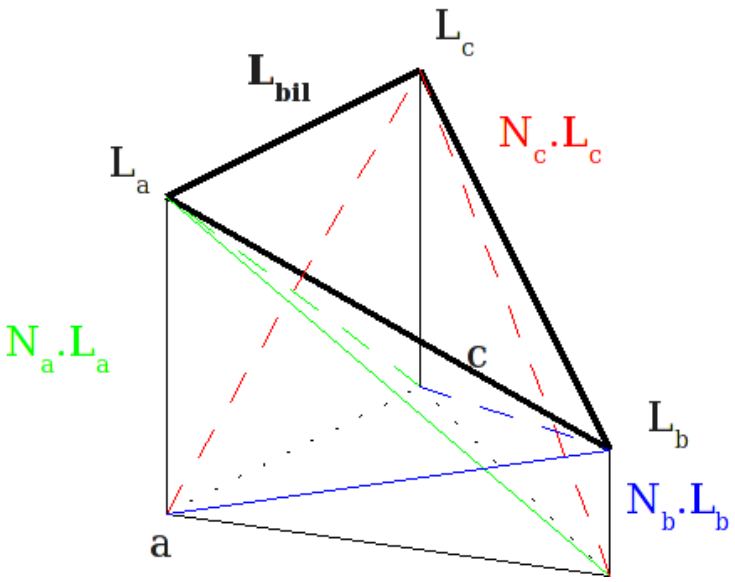



FIG. 2.9 – Construction de la fonction d'interpolation

Le flux reçu par le triangle est alors approché par :

$$\begin{aligned} \Phi &= \int_D L_{bil} dxdy \\ &= \int_D \begin{pmatrix} N_a(x,y) & N_c(x,y) & N_c(x,y) \end{pmatrix} dxdy \begin{pmatrix} L_a \\ L_b \\ L_c \end{pmatrix} \end{aligned} \tag{2.17}$$

Posons maintenant $\alpha_i = \int_D N_i(x,y)dxdy \; \forall i \in \{a,b,c\}$, et calculons ces quantités par un changement de variables grâce à $\psi$. Nous remarquons que le jacobien de la transformation correspond à la matrice $T_\psi$ identifiée en (2.15). Ceci nous permet d'écrire :

$$\alpha_i = \int_{D'} N_i(\psi(\zeta,\gamma)) \left|\det(T_\psi)\right| d\zeta d\gamma \tag{2.18}$$

En notant $N'_i(\zeta,\gamma) = N_i(\psi(\zeta,\gamma)$ et $D_\psi = \det(T_\psi)$, on a :

$$\alpha_i = D_\psi \int_0^1 \int_0^{1-\gamma} N'_i(\zeta,\gamma) d\zeta d\gamma \tag{2.19}$$

On construit les $N'_i$ de la manière suivante :

$$\begin{cases} N'_a(\zeta,\gamma) = 1 - (\zeta + \gamma) \\ N'_b(\zeta,\gamma) = \zeta \\ N'_c(\zeta,\gamma) = \gamma \end{cases} \tag{2.20}$$

Le calcul de (2.19) pour $a$, $b$ et $c$ nous donne :

$$\begin{pmatrix}\alpha_a & \alpha_b & \alpha_c\end{pmatrix} = D_\psi \begin{pmatrix}\frac{1}{6} & \frac{1}{6} & \frac{1}{6}\end{pmatrix} \tag{2.21}$$

Ceci nous permet d'écrire :

$$\Phi = D_\psi \begin{pmatrix}\frac{1}{6} & \frac{1}{6} & \frac{1}{6}\end{pmatrix} \begin{pmatrix}L_a \\ L_b \\ L_c\end{pmatrix} \tag{2.22}$$

Cette formule est valable pour *tout* triangle du maillage : seules les valeurs de $D_\psi$ et des $L_i$ changent. Cette propriété va maintenant nous servir à construire une formule permettant l'obtention directe du flux sur la totalité du maillage.

Pour ce faire, indexons par $j = 1, \ldots, n$ les points du maillage, et par $k = 1, \ldots, m$ les triangles du maillage. On note $\Phi_k$ le flux sur le triangle $k$, $D_{\psi,k}$ le déterminant du jacobien de la transformation permettant le passage de $D'$ au triangle $k$, et $\delta_{j,k}$ le symbole défini par :

$$\delta_{j,k} = \begin{cases} 1 & \text{si le point } j \text{ appartient au triangle } k \\ 0 & \text{sinon} \end{cases} \tag{2.23}$$

En observant que l'on a

$$\Phi_k = \frac{1}{6}\begin{pmatrix}\delta_{1,k}D_{\psi,k} & \cdots & \delta_{j,k}D_{\psi,k} & \cdots & \delta_{n,k}D_{\psi,k}\end{pmatrix} \begin{pmatrix}L_1 \\ \vdots \\ L_n\end{pmatrix} \tag{2.24}$$

ceci nous permet d'exprimer le flux total $\Phi_{tot}$ comme :

$$\begin{aligned}\Phi_{tot} &= \sum_{k=1}^{m} \Phi_k \\ &= \frac{1}{6}\begin{pmatrix}\sum_{k=1}^{m}\delta_{1,k}D_{\psi,k} & \cdots & \sum_{k=1}^{m}\delta_{j,k}D_{\psi,k} & \cdots & \sum_{k=1}^{m}\delta_{n,k}D_{\psi,k}\end{pmatrix} \begin{pmatrix}L_1 \\ \vdots \\ L_n\end{pmatrix}\end{aligned} \tag{2.25}$$

**récapitulatif et remarques générales sur la méthode**

Dans ce chapitre, une méthode de calcul de la radiation solaire reçue par une surface a été développée, utilisant un lancer de rayons. Les deux étapes fondamentales que sont le lancer de rayons et le calcul de la radiance observée y sont clairement séparées. Cette séparation possède des avantages non négligeables par rapport à une méthode où elle n'existerait pas. Elle permet notamment :

- de faire le calcul en de nombreuses dates, pour de nombreuses conditions météo, à partir d'un seul lancer de rayons ;
- de comparer les résultats obtenus avec divers modèles de ciel.

La radiation solaire se sépare en radiation directe et radiation diffuse, la première part étant prépondérante. Cette partie de la radiation provient cependant d'une portion extrêmement réduite de la voûte céleste, qu'un rayon tiré depuis la surface n'a que de faibles chances d'atteindre. Afin d'éviter que ce fait n'engendre une trop grande erreur de calcul, l'influence de la radiation directe et de la radiation diffuse sont calculées séparément :

- l'influence de la radiation diffuse est calculée directement, en passant les points de la voûte céleste visés à un modèle de ciel diffus.
- l'influence de la radiation directe est calculée par une procédure que nous avons désignée par «tir fictif», bien moins coûteuse que le lancer de rayons initial, grâce à l'utilisation des données de ce lancer. Il s'agit essentiellement d'une correction des données du lancer, de telle manière à ce que la probabilité de viser le soleil lorsqu'il est visible soit de 1.

Une limitation de cette méthode vient cependant que seules des surfaces planes peuvent être utilisées comme récepteur. Cette limitation provient de la méthodologie d'intégration retenue. En effet, le tir selon une distribution hémisphérique suppose que le point de tir n'est pas angulaire. Hors, dans le cas d'une surface non plane, les sommets des triangles correspondent précisément à de tels points. Ce problème peut être évité par l'utilisation d'une autre méthodologie d'intégration, couplée à un tir depuis des points internes aux triangles. Toutes les fonctions de l'implémentation proposée (exceptée la fonction d'import du maillage et la fonction d'intégration) sont compatibles avec un tel tir.

Chapitre 3

# Implémentation de la solution : structure du programme et algorithme détaillé

La méthode présentée au précédent chapitre a été implémentée sous la forme d'un programme Octave/Matlab. Le passage du modèle théorique à l'implémentation effective a apporté de nouveaux problèmes algorithmiques, sur les points de l'import des données géométriques et de la réduction de la complexité des calculs. Ce chapitre consiste donc en la présentation de deux éléments : l'approche adoptée pour résoudre ces deux problèmes, d'abord ; l'application de cette approche au cœur du programme, ensuite. L'intégralité du code source est proposée en annexe du présent rapport, ainsi que sur le cd compagnon. Une fiche résumant l'utilisation du programme est par ailleurs proposée en annexe A.

## 3.1 Import et codage des données géométriques : présentation des problèmes algorithmiques et de leurs solutions

Deux problèmes sont relatifs à l'implémentation effective :

1. l'import des données géométriques, à partir d'un logiciel de cao ;
2. la robustesse du programme face à une géométrie complexe.

### 3.1.1 Méthodologie d'import des données géométriques

L'algorithme de calcul élaboré dans le chapitre 2 nécessite la connaissance des données géométriques suivantes :

- la géométrie des surfaces physiques, avec les caractéristiques optiques associées ;
- la géométrie du maillage de calcul.

Le format de dialogue choisi entre le programme de cao et le programme proposé est le format stl. Ce format existe en deux versions, ascii et binaire, et consiste en une liste de triangles définis par leurs sommets et leur normale.

La fonction stlimport, développée par Luigi Giaccari, est utilisée pour l'import de tels fichiers dans Octave/Matlab.

#### Import du contexte urbain

La méthodologie retenue pour l'import est la suivante :

- le nord doit être aligné sur l'axe «des $y$» des fichiers ;
- un fichier stl par matériau doit être généré depuis le logiciel de cao ;
- les fichiers sont passés en argument à la fonction d'import, avec les caractéristiques associées (voir 2.2.1).

Les données sont obtenues sous la forme d'une liste de points (apparaissant un nombre de fois égal au nombre de triangles auxquels ils appartiennent), d'une liste de normales, d'une matrice des caractéristiques et d'un index référençant les sommets de chaque triangle.

#### Import du maillage

Le maillage est considéré comme défini en STL par l'utilisateur : aucune fonction générale n'est proposée pour raffiner ou modifier un maillage.

La méthodologie retenue pour l'import est la suivante :

- le nord doit être aligné sur l'axe «des $y$» des fichiers ;
- l'orientation du maillage est définie par l'utilisateur, selon l'axe de son choix ;
- le maillage est traité comme une pure surface de calcul, sans caractéristiques physiques : si il correspond à une surface physique, cette surface doit être passée à la fonction d'import décrite précédemment.

Les données sont obtenues sous la forme d'une liste de points uniques (l'unicité étant essentielle pour n'effectuer qu'un lancer par point), d'une liste de normales, d'un index référençant les sommets de chaque triangle, ainsi que d'une liste des feuilles de l'octree (voir 3.1.2) auxquelles appartiennent les points.

### 3.1.2 Robustesse face à une géométrie complexe

La méthode de lancer de rayons implique le test des intersections avec les divers triangles constituant le maillage. Comme vu en 1.2.3 dans le cas d'un ensemble non structuré de triangles, la seule solution connue pour déterminer la première intersection est de tester l'intersection avec tous les triangles, ce qui peut faire exploser le temps de calcul à l'apparition de la moindre surface courbe, ou dans le cas d'un modèle de grande emprise.

Une subdivision de l'espace sous forme d'octree est retenue (voir 1.2.3), la racine correspondant au plus petit cube englobant la scène et ayant ses faces orientées selon les axes. La structure de donnée retenue se compose de :

- une matrice $n \times 8$ recensant les successeurs des $n$ nœuds (0 correspondant à l'absence de successeur) ;
- un vecteur à $n$ éléments, recensant les successeurs des $n$ nœuds ;
- un tableau recensant les limites de chaque octant selon chaque axe ;
- un tableau de vecteurs listant chacun les triangles contenus dans les octants[1].

## 3.2 La géométrie au cœur du programme

La structure générale du programme est ici présentée. Le lecteur intéressé par le détail de l'algorithme trouvera l'intégralité du code source en annexe B.

Le programme se décompose basiquement en trois étapes (voir fig. 3.1) :

1. import de la géométrie et construction de l'octree ;
2. lancer de rayons ;
3. calcul énergétique.

[1] notons ici qu'une optimisation pourrait être effectuée : les triangles de tous les octants sont retenus, bien que cette information ne soit utile que pour les feuilles

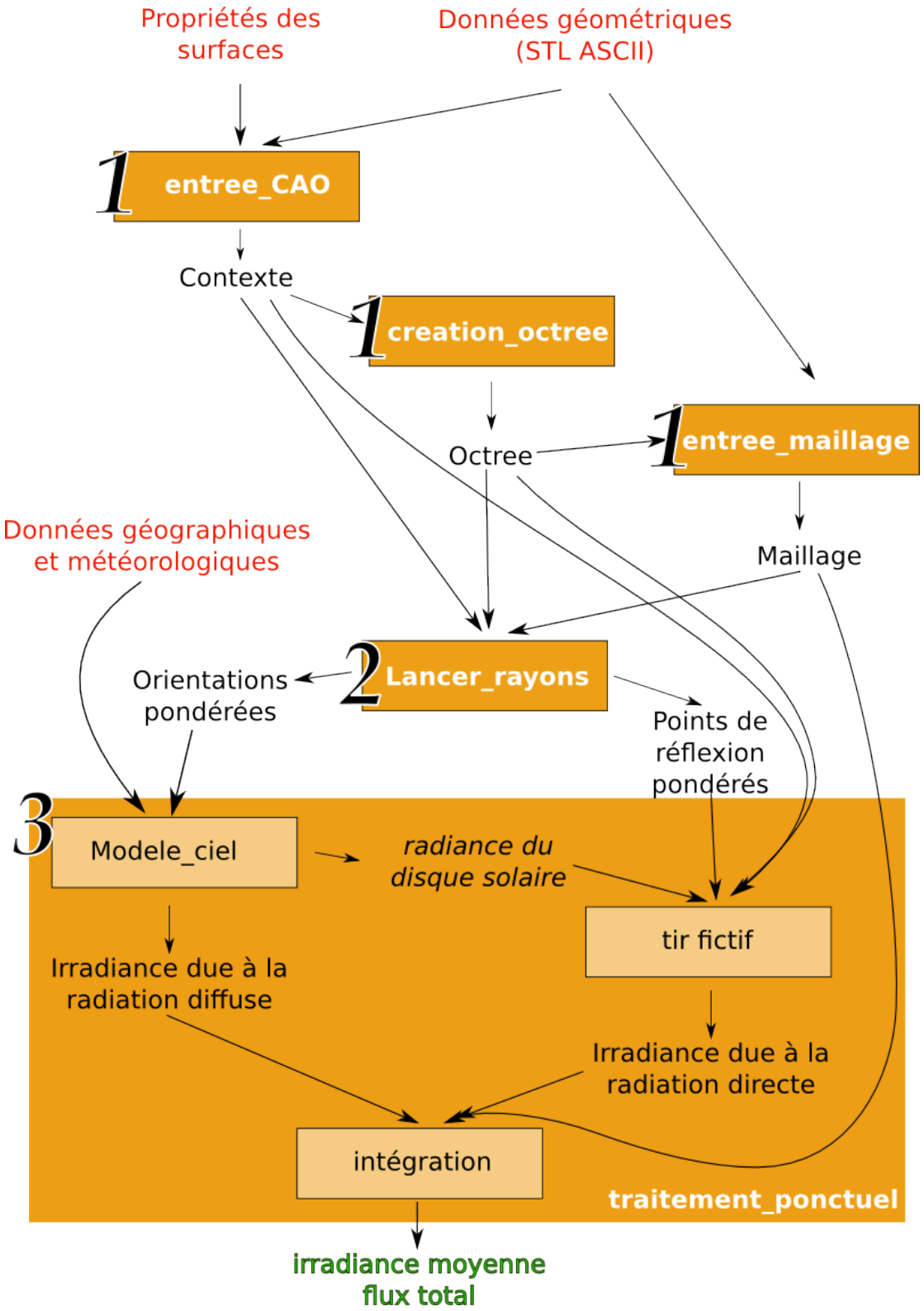



FIG. 3.1 – Représentation schématique de la structure du programme

### 3.2.1 Import de la géométrie

Après import des fichiers STL par le biais de la fonction STLimport, la construction de l'octree se fait de la manière suivante :

1. Un cube englobant la scène, dont les faces sont parallèles aux axes, est créé ;

2. Tant que le critère d'arrêt de la subdivision[2] n'est pas atteint, l'octant courant est subdivisé en huit, et les fils sont placés dans une pile des nœuds à examiner ;
3. l'appartenance des triangles de l'octant père est testée pour chacun des octants fils ;
4. l'examen continue jusqu'à ce que la pile soit vide.

L'appartenance des triangles aux octants se fait via la fonction `halfspace_imp_triangle_int_3d` de la bibliothèque Geometry[3] de l'université d'état de Floride. Le code source de cette opération est proposé en annexe B.1.3.

### 3.2.2 Lancer de rayons

Le lancer de rayons consiste en un tir récursif de rayons depuis chaque point du maillage. La détermination des intersections se fait grâce à un parcours d'octree, selon le principe suivant :

1. Les intersections sont testées dans l'octant courant, grâce à l'algorithme de Tomas Möller et Ben Trumbore ;
2. si aucune intersection n'est trouvée, le prochain octant traversé par le rayon est calculé, et la recherche est continuée dans cet octant.

La procédure de recherche du prochain octant s'inspire de celle proposée dans [Samet 89] :

1. le point et la(ou les) faces de sortie sont calculés ;
2. un parcours d'octree permet de trouver l'octant dans lequel le rayon entre.

Quelques précisions sont nécessaires quant au calcul du point de sortie. Tout d'abord, un même point peut correspondre à plusieurs faces (arrête ou coin par exemple) : toutes ces faces sont nécessaires pour un parcours d'arbre efficace, dont l'algorithme est présenté ci-après. Un autre point important est qu'il peut arriver que le point d'entrée corresponde au point de sortie[4] : un test d'intersection ne donne alors rien, et une recherche spécifique des faces est alors nécessaire. Le détail de la procédure est présenté en annexe B.2.8.

La connaissance des faces permet le parcours d'arbre : on recherche le premier prédécesseur contenant strictement les faces de sortie, puis l'on redescend jusqu'à la feuille dans laquelle entre le rayon. Cette recherche est facilitée par le stockage de la position de la face de sortie : si l'on sort par la face située au maximum (resp. minimum) selon un axe, on entrera par la face située au minimum (resp. maximum) selon cet axe. La construction éventuelle des faces de sortie lorsque le point de sortie correspond au point d'entrée se fait de telle manière à ce que cette approche reste valable.

## 3.3 Quelques résultats

Afin d'avoir une idée des temps de calcul et du type de résultat obtenu, quelques utilisations du programme ont été réalisées sur un contexte simpliste (voir fig. 3.2). C'est ce contexte qui a été pris comme exemple pour le guide proposé en annexe A.

Les calculs présentés en annexe A ont étés effectués, avec un maillage de 4 points et avec un maillage de 25 points. Les mêmes calculs ont étés effectués avec un nombre limite de 2 réflexions au lieu d'une, et avec deux profondeurs d'octree différentes. Comme dans l'exemple du guide, 100 tirs par point sont effectués.

[2] par défaut, ce critère est fixé à trois subdivisions ou trois triangles par octant ; la possibilité est cependant laissée à l'utilisateur de définir manuellement ces paramètres.

[3] http ://people.sc.fsu.edu/ burkardt/m_src/geometry/geometry.html

[4] par exemple lorsque le point du maillage est situé à un coin de la racine de l'octree, ce qui peut arriver dans des géométries simples

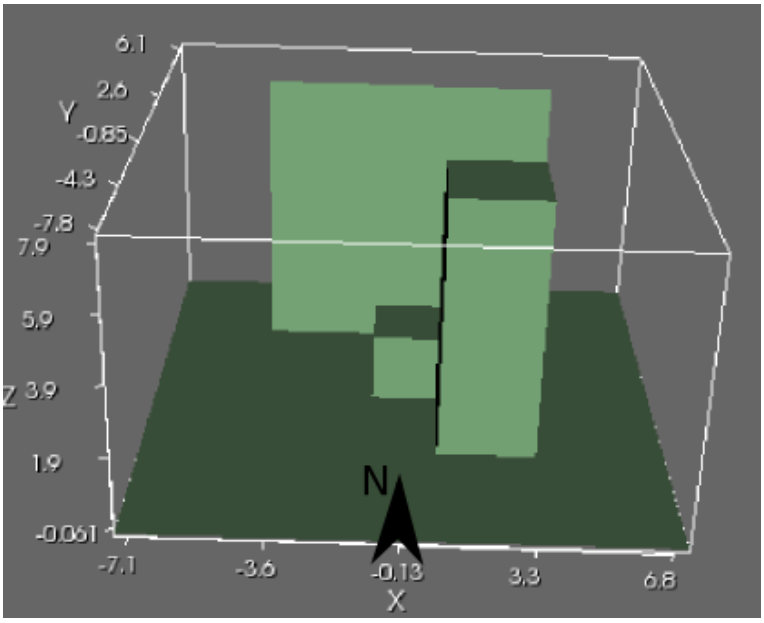



FIG. 3.2 – Modèle simple utilisé pour la phase de test

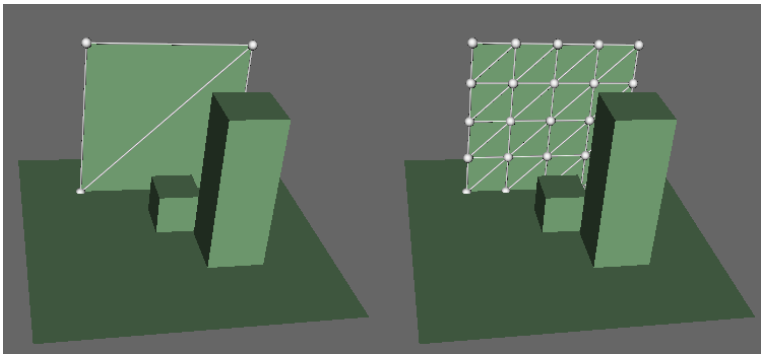


FIG. 3.3 – maillage de 4 points et maillage de 25 points

Les résultats sont présentés ci-après, pour un ciel clair au 21 juin à 12h00, par 49 ° de latitude Nord (Compiègne).

| Nombre de noeuds du maillage | profondeur d'octree | nombre de réflexions | durée du lancer de rayons ($s$) | durée du tir fictif ($s$) | Puissance totale reçue ($W$) | Flux moyen reçu ($W.m^{-2}$) |
|---|---|---|---|---|---|---|
| 4 | 3 | 1 | 11.41 | 0.29 | 385430 | 3043.5 |
| | | 2 | 13.36 | 0.61 | 385780 | 3046.3 |
| | 5 | 1 | 30.71 | 1.01 | 386150 | 3049.2 |
| | | 2 | 38.32 | 1.29 | 387980 | 3063.6 |
| 25 | 3 | 1 | 981.48 | 22.52 | 458850 | 3623.2 |
| | | 2 | 3944.5 | 214.97 | 449640 | 3550.5 |
| | 5 | 1 | 797.34 | 28.83 | 412830 | 3259.8 |
| | | 2 | 2035 | 125.25 | 415170 | 3278.3 |

Le nombre de calculs est une fonction exponentielle du nombre maximum de réflexions : au plus, $N^m$ rayons par point sont tirés, avec $N$ le nombre de rayons par tir et $m$ le nombre de réflexions. Ainsi, si le temps de calcul est extrêmement raisonnable pour

une profondeur d'une réflexion (de l'ordre d'un quart d'heure avec 25 points), il devient déjà très élevé dans le cas de deux réflexions [5] : dans le cas d'un octree peu profond avec 25 points de calcul, le temps de calcul est quadruplé avec le passage à deux réflexions.

Le temps du tir fictif est, comme prévu, parfaitement raisonnable, ne dépassant pas 4 minutes pour un tir ayant duré plus d'une heure.

Le nombre de nœuds du maillage influe évidemment sur la précision du résultat, en plus d'influer sur la compréhension de sa spatialité : l'écart entre les valeurs de puissance pour les deux maillages varie de 7% à 19%. L'influence du nombre de réflexions est bien moindre : pour la puissance, le plus grand écart constaté entre la valeur avec 1 et celle avec 2 réflexions est de 2% (pour 25 points et 3 subdivisions d'octree).

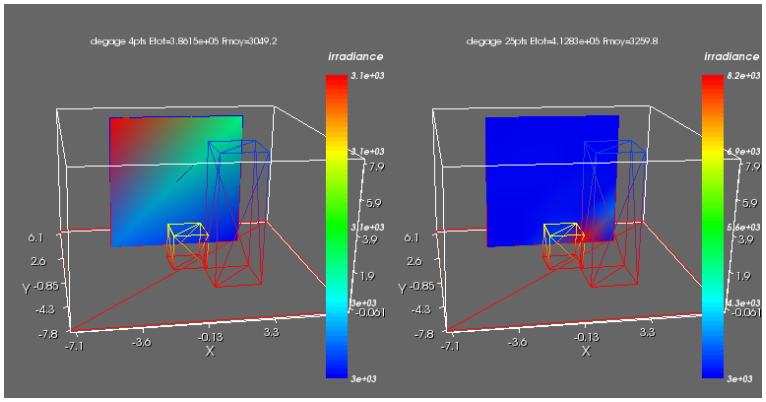



FIG. 3.4 – Représentation des résultats pour un maillage de 4 points et un maillage de 25 points

Remarquons enfin que sur une géométrie simple comme celle-ci, l'octree semble apporter plus d'erreurs de calcul que d'améliorations du temps de calcul : pour 25 points, le temps de calcul est bien diminué par un octree plus profond, mais la différence entre les valeurs obtenues atteint 10% ; pour 4 points, la variation dans le résultat n'est que de l'ordre de 1%, mais le temps de calcul est empiré...

En effet, chaque changement d'octant revient à 6 calculs d'intersection (voir annexe B.2.8). Notre scène simpliste ne contient que 28 triangles : le parcours d'un octree trop fin revient au test de plus d'intersections que si l'on testait systématiquement l'intersection avec tous les triangles. Gageons cependant qu'un octree fin améliore grandement le temps de calcul dans le cas d'une géométrie complexe.

Une étude plus approfondie serait intéressante. L'étude de la dépendance du résultat du calcul aux divers paramètres permettrait en effet d'établir des réglages optimaux du point de vue du temps de calcul, en fonction de la précision souhaitée. Les critères d'optimalité pour l'octree demandent notamment à être fixés. Une étude comparative avec des résultats de mesure serait aussi nécessaire.

[5] notons qu'un essai a été effectué sur un ordinateur portable avec une profondeur de 5 réflexions : après plus de 10 heures de calcul, l'ordinateur tournait encore...

# Conclusion

Dans ce rapport, un algorithme de calcul de la radiation solaire reçue par une surface en contexte urbain, avec prise en compte des réflexions multiples et des conditions météorologiques, a été conçu. Une implémentation a par ailleurs été réalisée, et est fournie sur le CD compagnon.

L'algorithme proposé prend en compte les caractéristiques des surfaces à travers trois paramètres : réflexivité, rugosité, anisotropie. Bien que ces paramètres soient intuitifs, une détermination expérimentale de leurs valeurs pour divers matériaux de construction pourrait permettre une utilisation plus éclairée du programme.

Afin d'améliorer le temps de calcul en cas de géométrie complexe, une subdivision de l'espace par octree a été adoptée. Un rapide test a cependant révélé que dans des cas simples, cette subdivision pouvait au contraire faire empirer le temps de calcul.

Le temps de calcul de l'implémentation proposée s'est révélé très variable suivant les réglages et la précision demandée. Une étude approfondie des relations entre les divers réglages possibles et la précision des résultats, bien que non proposée ici, nous semble fondamentale pour permettre une utilisation appropriée du programme. Une telle étude pourrait déboucher sur l'établissement de réglages type pour diverses utilisations, en termes de finesse du maillage, nombre de réflexions et profondeur d'octree.

# Annexes

Annexe A

# Guide de l'utilisateur

La présente annexe présente succinctement le mode d'utilisation du programme. Précisons que toutes les fonctions sont documentées : `help <nom de la fonction>` permettra d'obtenir des informations sur les entrées et sorties des diverses fonctions. Le but de ce court guide n'est donc pas d'être exhaustif, mais plutôt de décrire l'enchaînement des fonctions pour arriver au résultat chiffré souhaité.

Les différents répertoires contenant les fonctions sont supposés chargés dans Octave/Matlab (fonction `addpath`).

Supposons que le contexte de calcul soit composé des éléments suivants, avec leurs caractéristiques :

| nom du fichier | réflexivité $c$ | rugosité $r$ | anisotropie $p$ |
|---|---|---|---|
| surface_calcul.stl | $0,2$ | $0,5$ | $0,2$ |
| sol.stl | $0$ | $0$ | $0$ |
| cube.stl | $0,5$ | $0,5$ | $0,5$ |
| tour.stl | $0,8$ | $0,2$ | $0,2$ |

La première action à effectuer est d'importer les données géométriques, par la commande suivante :

```
[sommets, normales, indices, caracteristiques]=entree_cao('surface_calcul.stl', [0.2 0.5 0.2], 'sol.stl', [0 0 0], 'cube.stl', [0.5 0.5 0.5], 'tour.stl', [0.8 0.2 0.2]);
```

Les données obtenues peuvent ensuite être passées à la fonction de création de l'octree :

```
[successeurs, predecesseurs, triangles_octants, octants]=creation_octree(sommets, indices, 5, 2);
```

Où les deux derniers arguments, optionnels, signifient que l'on fixe la condition d'arrêt de la subdivision à 2 triangles par octant et à 5 subdivisions maximum.

Le maillage de calcul va alors pouvoir être inséré dans la scène :

```
[pts_maill, ind_maill, norm_maill, noeuds_maill] = entree_maillage('surface_calcul.stl', [0 -1 0], octants, successeurs);
```

Le deuxième argument signifiant que le tir s'effectuera dans le sens des $y$ négatifs.

La géométrie est maintenant parfaitement définie, et le tir des rayons peut être effectué :

```
[orientations, coefficients, chemin, temps] = lancer_rayons(pts_maill, norm_maill, noeuds_maill, successeurs, predecesseurs, octants, triangles_octants, sommets, normales, indices, caracteristiques, 100, 1);
```

Nous effectuons ici 100 tirs par point du maillage. Le dernier argument, optionnel, signifie que l'on ne prend en compte qu'un niveau de réflexion (en l'absence d'argument, la valeur par défaut est de 2).

Le temps de calcul est retourné dans la variable `temps`, à des fins d'information.

Après exécution de la commande précédente, il est possible d'effectuer le calcul de la radiation reçue en divers instants, par diverses conditions météorologiques. Par exemple, voici le calcul effectué à Compiègne au 21 juin à midi, par temps dégagé puis par temps couvert :

```
1 [Etot_deg, Fmoy_deg, irrad_deg, irr_dir_deg, irr_diff_deg, tempsdir_deg] =
      traitement_ponctuel(coefficients, orientations, 49, 172, 12, 1, 1, 2,
      ind_maill, pts_maill, norm_maill, sommets, normales, indices,
      caracteristiques, chemin, successeurs, predecesseurs, octants,
      triangles_octants);
2
3 [Etot_couv, Fmoy_couv, irrad_couv, irr_dir_couv, irr_diff_couv,
      tempsdir_couv] = traitement_ponctuel(coefficients, orientations, 49,
      172, 12, 0.1, 0.1, 2, ind_maill, pts_maill, norm_maill, sommets,
      normales, indices, caracteristiques, chemin, successeurs, predecesseurs
      , octants, triangles_octants);
```

Annexe B

# code source

## B.1 fonctions d'import

### B.1.1 entree_CAO

```
function [sommets, normales, indices, caracteristiques]=entree_CAO(varargin)
%
%entree_CAO
%
%fonction gerant l'import des donnees geometriques
%
%synopsis:
% [sommets, normales, indices, caracteristiques]=entree_CAO(fichier1, caract1,...,fichierN,caractN)
%
%ou:
% sommets: matrice des coordonnees des sommets des triangles, avec repetition
% normales: normales des triangles passes en entree
% indices: index des sommets de chaque triangle
%          interet:
%          -fonction "trisurf" en a besoin
%          -toutes les autres fonctions sont en adequation avec un codage sans repetition, plus compact
% caracteristiques: vecteur recensant les caracteristiques par triangle (peut etre facilement etendu a d'autres caracteristiques)
%
%entrees
% fichiers stl et vecteur des caracteristiques correspondant, alternativement
% la surface d'interet DOIT etre passee si il s'agit d'une surface physique
%
% REMARQUE SUR L'ENTREE DE FICHIER STL:
% les fichiers STL sont importes par la fonction STL_Import de Luigi Giaccari
% Les fichiers STL binaires exportes depuis certains logiciels peuvent poser des problemes
% La conversion de tels fichiers en fichiers STL ASCII (par l'utilitaire libre Admesh par exemple) resoud le probleme
%

%test de la validite des entrees
if rem(nargin,2)~=0
   error('nombre d''arguments impair!')
end

%initialisation des sorties
 sommets = [];
 normales = [];
 indices  = [];
 caracteristiques =[];

 for i=1:2:(nargin-1)
```

```
41    %import des triangles
42    fichier_import = varargin{i};
43    [sommets_import,normales_import]=STL_Import(fichier_import,2);
44    sommets = [sommets;sommets_import];
45    normales = [normales;normales_import];
46
47    %import des caracteristiques des triangles
48    [n,m] = size(sommets_import);
49    caract_import = ones((n/3),1)*varargin{i+1};
50    caracteristiques = [caracteristiques;caract_import];
51   end
52
53   %construction de l'index
54   [n,m] = size(sommets);
55   for i=1:3:n
56    indices = [indices; i i+1 i+2];
57   end
58
59   end
```

### B.1.2 creation_octree

```
1  function [successeurs,predecesseur,triangles_octants,octants]=
       creation_octree(sommets,indices,varargin)
2  %
3  %creation_octree
4  %
5  %fonction qui contruit l'octree sur la base de donnees geometriques
6  %
7  %function [successeurs,predecesseur,triangles_octants,octants]=
       creation_octree(sommets,indices,(profondeur_max,N_triangles))
8  %
9  %entree:
10 %sommets: coordonnees de chaque sommet
11 %index: pour chaque triangle, indice des trois lignes contenant les trois
       sommets du triangle
12 %profondeur_max et N_triangles: conditions d'arret
13 %
14 %sorties:
15 %successeurs: matrice a 8 colonnes, recensant les successeurs de chaque
       noeud. une ligne de 0 caracterise une feuille
16 %predecesseur: vecteur recensant pour chaque noeud son predecesseur
17 %triangles_octant: tableau de vecteurs recensant les triangles inclus dans
       chaque octant
18 %octants: matrice 3D: octant(i,coordonnee,:) contient le min et le max de l
       'octant correspondant au noeud i selon la coordonnee donnee
19 %
20
21 %idee:
22 %on subdivise recursivement les octants jusqu'a la condition d'arret (
       profondeur ou nombre de triangles)
23 %pour ce faire, on conserve en memoire dans une pile le chemin parcouru
       jusqu'au pave courrant, ainsi que la profondeur associee.
24 %on opere "en profondeur d'abord"
25
26 %parametres par defaut
27 if (length(varargin) == 0)
28   profondeur_max= 3;
29   N_triangles=3;
30   elseif (length(varargin) == 2)
31     profondeur_max= varargin{1};
32     N_triangles=varargin{2};
```

```
33      else
34        error('nombre d''arguments inadapte');
35  end
36
37  %initialisation des min de la racine
38  octants=zeros(1,3,2);
39  for i=1:3
40    octants(1,i,1)=min(sommets(:,i));
41  end
42
43  %determination de la longueur du cote de la racine
44  longueur=0;
45  for i=1:3
46    longueur=max(longueur,(max(sommets(:,i)))-octants(1,i,1));
47  end
48
49  %remplissage des max de la racine
50  for i=1:3
51    octants(1,i,2)=octants(1,i,1)+longueur+eps;
52  end
53
54  %tous les triangles appartiennent a la racine
55  [n,m]=size(indices);
56  triangles_octants{1}=[1:n]';
57
58  %ici debute la construction de l'octree
59  %initialisation des variables servant a la construction
60  pointeur=1;
61  pile(1)=1;       %pile permetant le marquage du chemin emprunte pour
        atteindre le noeud courrant
62  profondeur(1)=1;    %pile listant la profondeur de chaque noeud de pile
63  noeud=1;       %indice du dernier noeud ajoute a l'arbre
64  unite=ones(8,1);    %pour l'affectation des predecesseurs
65  nullite=zeros(8,1);   %pour le marquage des feuilles
66
67
68  while (pointeur>0)
69    %choix de l'octant au dessus de la pile
70    pere = pile(pointeur);
71    profondeur_pere = profondeur(pointeur);
72    pointeur = pointeur-1;
73
74    %subdivision dans l'arbre
75    successeurs(pere,:)=[noeud+1:noeud+8];
76    predecesseur(noeud+1:noeud+8,1) = pere*unite;
77
78    %definition de la partition
79    %idee: on defini le premier fils, et on construit les autres par une
          transformation lineaire.
80    octants(noeud+1:noeud+8,:,:)=subdivision_geom(octants(pere,:,:));
81
82    %affectation des triangles aux octants
83    triangles_octants(noeud+1:noeud+8)=subdivision(octants(pere,:,:),
          triangles_octants{pere},indices,sommets);
84
85    %placement des octants dans la pile des noeuds a subdiviser si le nombre
          de triangle>N_triangles et si profondeur<max
86    %marquage comme feuille sinon
87    for i=1:8
88      noeud=noeud+1;
89      if ((length(triangles_octants{noeud})>N_triangles)&&(profondeur_pere<
            profondeur_max))
90        pointeur=pointeur+1;
```

```
91          pile(pointeur)=noeud;
92          profondeur(pointeur)=profondeur_pere+1;
93        else
94          successeurs(noeud,:) = 0;
95        end
96      end
97    end
```

### B.1.3 subdivision

```
1  function [triangles_octant]=subdivision(pere , triangles_pere, indices,
       sommets)
2  % subdivision
3  %
4  % reference les triangles inclus dans chaque fils d'un octant pere
5  %
6  % [triangles_octant]=subdivision(pere , triangles_pere, indices, sommets)
7  %
8  % entrees:
9  %   pere: matrice 1x3x2 avec limites de l'octant pere sur chaque axe
10 %   triangles_pere: vecteur contenant les indices des triangles contenus
       par l'octant pere
11 %   indices, sommets: donnees geometriques sur les triangles
12 %
13 % sortie:
14 %   triangles_octant: tableau de vecteurs recensant les triangles contenus
       par chaque octant fils
15 %
16
17 %initialisation de la sortie
18 triangles_octant{1:8}=[];
19
20 %initialisation de la variable de travail triangles_despace:
       triangles_despace(i,j) est l'index des triangles du jeme1/2 espace de
       coord i
21 triangles_despace{1:3,1:2} = [];
22
23   %pour chaque coordonnee, on teste l'appartenance de chaque triangle aux
         deux demis-espaces
24         for i=1:3
25   %definition de l'equation du plan (ax+by+cz+d<=0)
26                 plan=zeros(1,4) ;
27                 plan(1,i)=1;
28                 plan(1,4)=(pere(1,i,1)+pere(1,i,2))/2;  %defini le plan
                       median selon la coordonnee i
29
30                 %on reference les triangles contenus dans chaque demi-
                       espace, intersections simples (un seul angle en contact
                       ) exclues
31                 for j=1:length(triangles_pere)
32                       indices_sommets=indices(triangles_pere(j),:);
                                 %indices des sommets du triangle j
33                       triangle=[sommets(indices_sommets(1),:);sommets(
                             indices_sommets(2),:);sommets(indices_sommets
                             (3),:)]; %matrice 3x3 avec sommets
34                       intersections_1=intersection_gauche(plan,triangle);
35                       intersections_2=intersection_droite(plan,triangle);
36                       if intersections_1>1
37                               triangles_despace{i,1}=[triangles_despace{i
                                     ,1};triangles_pere(j)];
38                       end
39                       if intersections_2>1
```

```
40                                  triangles_despace{i,2}=[triangles_despace{i
                                        ,2};triangles_pere(j)];
41                              end
42                      end
43              end
44
45              %on reference les triangles de chaque octant
46              %on compte selon x croissant, puis y croissant, puis z croissant
47              compteur=1;
48              for z=1:2
49                      for y=1:2
50                              for x=1:2
51                                  triangles_octant{compteur}=intersect(
                                        triangles_despace{1,x},intersect(
                                        triangles_despace{2,y},
                                        triangles_despace{3,z}));
52                                  compteur=compteur+1;
53                              end
54                      end
55              end
56 end
57
58
59 function [intersections]=intersection_gauche(plan,triangle)
60 %intersections est le nombre d'angles du triangles contenus dans le demi-
       espace defini par:
61 %plan(1)*x + plan(2)*y +plan(3)*z + plan(4) >= 0
62 a=plan(1);
63 b=plan(2);
64 c=plan(3);
65 d=plan(4);
66  [intersections, pint] = halfspace_imp_triangle_int_3d (a,b, c, -d,
       triangle );
67 end
68
69 function [intersections]=intersection_droite(plan,triangle)
70 %intersections est le nombre d'angles du triangles contenus dans le demi-
       espace defini par:
71 %plan(1)*x + plan(2)*y +plan(3)*z + plan(4) <= 0
72 a=plan(1);
73 b=plan(2);
74 c=plan(3);
75 d=plan(4);
76  [intersections, pint ] = halfspace_imp_triangle_int_3d (-a,-b, -c, d,
       triangle );
77 end
```

### B.1.4 subdivision_geom

```
1 function fils = subdivision_geom(pere)
2 %
3 %subdivision_geom
4 %
5 %fils = subdivision_geom(pere)
6 %
7 %fonction qui produit les octants fils a partir de l'octant pere, au format
       defini globalement
8 %entree:
9 % pere: octant a subdiviser, matrice 3D de taille 1x3x2 (pour la
      compatibilite avec les autres fonctions)
10 %sortie:
11 % fils: octants fils, dans une matrice 3D de taille 8x3x2
```

```
12
13 %initialisation et definition des grandeurs
14 base=zeros(3,2);
15 fils=zeros(8,3,2);
16
17 unite=ones(8,1);
18
19 %matrice definissant l'ecart de chaque octant
20 %a l'origine selon chaque axe
21 %par exemple:
22 % premier octant colle a l'origine
23 % deuxieme decolle de l'origine selon x
24 % etc.
25 T=[0 0 0;
26 1 0 0;
27 0 1 0;
28 1 1 0;
29 0 0 1;
30 1 0 1;
31 0 1 1;
32 1 1 1];
33
34 l = (pere(1,1,2) - pere(1,1,1))/2;
35
36 for i=1:3
37   base(i,1) = pere(1,i,1);
38   base(i,2) = pere(1,i,1) + l;
39 end
40
41 %construction des fils
42 for i=1:2
43   fils(:,:,i)=unite*(base(:,i))' + T*l;
44 end
45 end
```

### B.1.5 entree_maillage

```
1 function [points, indices, normales, noeuds] = entree_maillage(fichier,
      orientation, octants, successeurs)
2 %
3 % entree_maillage
4 %
5 % import d'un maillage de calcul au format stl
6 %
7 % [points, indices, normales, noeuds] = entree_maillage(fichier,orientation
      , octants, successeurs)
8 %
9 % entrees:
10 %   fichier: nom du fichier stl a importer. tous les triangles doivent
       avoir la MEME orientation
11 %   orientation: vecteur a trois composantes, dont l'une au moins est egale
       a +/-1: cela defini le sens de tir selon cet axe.
12 %   octants, successeurs: donnees de l'octree
13 %
14 % sorties:
15 %   points: matrice a trois colonnes listant les points du maillage
16 %   indices: indices des sommets des triangles
17 %   normales: normales a la surface.
18 %   noeuds: vecteur listant les indices des octants auquels appartiennent
       les points du maillage
19 %
20 % cette fonction est basee sur la fonction d'import STL de Luigi Giaccari
```

```
21
22 %import de la surface, sous la forme de points uniques
23 [points,indices,normales]=STL_Import(fichier,1);
24
25 %%On commence par l'orientation de la surface
26 %verification de la planearite
27 verif_normales = unique(normales,'rows');
28 diff_normales = (size(verif_normales,1));
29
30 if (diff_normales ~= 1)&&((diff_normales ~= 2)||(verif_normales(1,:)~= -
       verif_normales(2,:)))
31   error('surface non planaire!');
32 end
33
34 %recherche de la premiere coordonnee pour laquelle un sens est indique
35 for i=1:3
36   coord = i;
37   sens = orientation(coord);
38   if (abs(sens) == 1)
39     break
40   end
41 end
42
43 if abs(sens)~=1
44   error('format d''orientation non reconnu');
45 end
46
47 %orientation de la normale
48 normales = (ones(size(points,1),1) * (normales(1,coord)/abs(normales(1,
       coord)))*sens)*normales(1,:);
49
50 %%on fini par la recherche des octants contenant les points
51 Npoints = size(points,1);
52 noeuds=ones(Npoints,1);
53
54 for i=1:Npoints
55   noeuds(i) = recherche(points(i,:), octants, successeurs);
56 end
57 end
58
59 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
60 function [noeud] = recherche(point, octants, successeurs)
61 %fonction interne permettant la recherche de l'octant contenant un point
62
63 noeud = 1;
64
65 while (successeurs(noeud,1) ~= 0)
66   for i=1:8
67     noeud_test = successeurs(noeud,i);
68     if (point >= octants(noeud_test, [1:3]', 1))&&(point <= octants(
         noeud_test, [1:3]', 2))
69       noeud = noeud_test;
70       break
71     end
72   end
73 end
74
75 end
```

## B.2 Lancer de rayons

### B.2.1 lancer_rayons

```
function [orientations, coefficients, chemin, temps] = lancer_rayons(
    points_calcul, normales_calcul, noeuds_calcul, successeurs,
    predecesseurs, octants, triangles_octants, sommets, normales, indices,
    caracteristiques, N, varargin)
% lancer_rayons
%
% effectue un lancer de rayons monte carlo et retourne des orientations
    ponderees
%
% [orientations, coefficients, chemin, temps] = lancer_rayons(points_calcul
    , normales_calcul, noeuds_calcul, successeurs, predecesseurs, octants,
    triangles_octants, sommets, normales, indices, caracteristiques, N, (
    profondeur_max))
%
% sorties:
%   orientations, coefficients: tableaux de vecteurs recensant les points
    de la voute celeste et les coefficients associes
% chemin: tableau de structures contenant des vecteurs recensant les points
     de reflection, ainsi que les
%   orientations incidentes, triangles, coefficients, noeuds et nombre de
    tirs associes
%
%   temps: temps de calcul
%
% entrees:
%   points_calcul, noeuds_calcul, normales_calcul: donnees du maillage
%   successeurs, predecesseurs, octants, triangles_octants: donnees de l'
    octree
%   sommets, normales, indices, caracteristiques: donnees geometriques
%   N: nombre de lancers de rayons par point
% profondeur max: nombre de reflections maximales prises en compte (
    optionnel, 2 par defaut)

if ~isempty(varargin)
  profondeur_max = varargin{1};
else
  profondeur_max = 2;
end

temps = cputime;
%tir depuis chaque point
for i=1:size(points_calcul,1)
  fprintf('\n debut du tir au point %i \n',i)

  [orientations_tir, M] = generation_aleatoire(N);
  [orientations_tir] = orientation_vecteurs_aleatoires(orientations_tir,
      normales_calcul(i,:));

  coefficients_tir = ((pi^2)/M) * ones(M,1);

  [orientations{i}, coefficients{i}, chemin{i}] = tir(...
    successeurs, predecesseurs, octants, triangles_octants,...
    sommets, normales, indices, caracteristiques,...
    points_calcul(i,:), 1, noeuds_calcul(i), orientations_tir,
        coefficients_tir, M, profondeur_max, 0);

  %ajout du point de calcul au chemin retenu
  chemin{i}.points = [points_calcul(i,:); chemin{i}.points];
  chemin{i}.coefs = [(pi^2)/M; chemin{i}.coefs];
```

```
46    chemin{i}.triangles = [0; chemin{i}.triangles];
47    chemin{i}.noeuds = [noeuds_calcul(i); chemin{i}.noeuds];
48    chemin{i}.ntirs = [M; chemin{i}.ntirs];
49    chemin{i}.orientations = [0 0 0; chemin{i}.orientations];
50
51    fprintf('%i points de reflexion\n%i orientations celestes\n', max(chemin{
          i}.pointsfinaux)-1,length(coefficients{i}))
52  end
53
54  temps = cputime - temps;
55
56  end
```

### B.2.2 tir

```
1  function [orientations_finales, coefficients_finaux, chemin, i_tir_courrant
       ] = tir(successeurs, predecesseurs, octants, triangles_octants, sommets
       , normales, indices, caracteristiques, point_tir, i_tir, noeud_tir,
       orientations_tir, coefficients_tir, M, profondeur_max,
       profondeur_actuelle)
2  %
3  % tir
4  %
5  % renvoie un lot d'orientations ponderees, correspondant aux points de la
       voue celeste illuminant un unique point
6  %
7  %[orientations_finales, coefficients_finaux, chemin, i_tir_courrant] = tir(
       successeurs, predecesseurs, octants, triangles_octants, sommets,
       normales, indices, caracteristiques, point_tir, i_tir, noeud_tir,
       orientations_tir, coefficients_tir, M, profondeur_max,
       profondeur_actuelle)
8  %
9  % sorties:
10 %   orientations_finales: matrice a trois colonnes, listant les
       orientations sous forme de vecteurs unitaires
11 %   coefficients_finaux: vecteur colonne recensant le coefficient affectant
        chaque orientation
12 % chemin: tableau de structures contenant des vecteurs recensant les points
        de reflection, ainsi que les
13 %   orientations incidentes, triangles, coefficients, noeuds et nombre de
       tirs associes
14 % i_tir_courrant: indice du dernier point de tir dans l'arbre des chemins
15 %
16 % entrees:
17 %   successeurs, predecesseurs, octants, triangles_octants: donnees de l'
       octree
18 %   sommets, normales, indices, caracteristiques: donnees geometriques
19 %   point_tir: coordonnees du point de tir (vecteur ligne)
20 %   i_tir: indice du point de tir dans l'arbre des chemins
21 %   noeud_tir: indice de l'octant contenant le point de tir
22 %   orientations_tir: matrice a trois colonnes, recensant les orientations
       de tir sous forme de vecteurs unitaires
23 %   coefficients_tir: coefficients affectant chacune des directions
24 % M: nombre de valeurs samplees (peut etre different de la longueur de
       coefficients_tir, qui peut etre reduit a la liste des coefficients non
       nuls
25 % profondeur_max, profondeur_actuelle: entiers permettant de tester la
       condition d'arret
26 %
27
28 orientations_finales = [];
29 coefficients_finaux = [];
```

```
chemin.points = [];
chemin.orientations = [];
chemin.triangles = [];
chemin.coefs = [];
chemin.noeuds = [];
chemin.ntirs = [];
chemin.pointsfinaux = [];

%pointe la position du point de reflection courrant dans l'arbre
%permet de lier les orientations finales aux points depuis
%lesquelles elles ont etees visees
i_tir_courrant = i_tir;

for i=1:length(coefficients_tir)
  %fprintf('profondeur %d tir %d\n',profondeur_actuelle,i)

  %verification de la norme de l'orientation
  switch norm(orientations_tir(i,:))
          case 1
                orientation = orientations_tir(i,:);
          otherwise
                orientation = orientations_tir(i,:) / norm(orientations_tir
                    (i,:));
      end
      coefficient = coefficients_tir(i);

  %recherche de la prochaine intersection, si elle existe
  [intersect_bool, triangle_intersect, point_intersect, noeud_intersect] =
      prochaine_intersection(...
    successeurs, predecesseurs, triangles_octants, octants,...
    sommets, indices, point_tir, noeud_tir, orientation);

  %recursion si une intersection est trouvee
  if (intersect_bool == 1)&&(profondeur_actuelle < profondeur_max)
    %calcul des coefficients de reflexion
    [orientations_refl, coefficients_refl, ntirs_refl] = reflection(...
      normales(triangle_intersect,:), caracteristiques(triangle_intersect
          ,:), orientation, M);

    %stockage du point de reflexion
    chemin.points = [chemin.points; point_intersect];
    i_tir_courrant = i_tir_courrant + 1;
    chemin.orientations = [chemin.orientations; orientation];
    chemin.triangles = [chemin.triangles; triangle_intersect];
    chemin.coefs = [chemin.coefs; coefficient];
    chemin.noeuds = [chemin.noeuds; noeud_intersect];
    chemin.ntirs = [chemin.ntirs; ntirs_refl];

    %recursion
    if ~isempty(coefficients_refl)
      [orientations_rec, coefficients_rec, cheminrec, i_tir_courrant] = tir
          (...
        successeurs, predecesseurs, octants, triangles_octants,...
        sommets, normales, indices, caracteristiques,...
        point_intersect, i_tir_courrant, noeud_intersect,...
        orientations_refl, (coefficients_refl*coefficient),...
        M, profondeur_max, profondeur_actuelle + 1);

      %stockage des resultats de la recursion
      chemin.points = [chemin.points; cheminrec.points];
      chemin.orientations = [chemin.orientations; cheminrec.orientations];
      chemin.triangles = [chemin.triangles; cheminrec.triangles];
      chemin.coefs = [chemin.coefs; cheminrec.coefs];
```

```
89          chemin.noeuds = [chemin.noeuds; cheminrec.noeuds];
90          chemin.ntirs = [chemin.ntirs; cheminrec.ntirs];
91
92          orientations_finales = [orientations_finales; orientations_rec];
93          chemin.pointsfinaux = [chemin.pointsfinaux; cheminrec.pointsfinaux];
94          coefficients_finaux = [coefficients_finaux; coefficients_rec];
95        end
96      else
97        orientations_finales = [orientations_finales; orientation];
98        chemin.pointsfinaux = [chemin.pointsfinaux; i_tir];
99        coefficients_finaux = [coefficients_finaux; coefficient];
100     end
101 end
102
103 end
```

### B.2.3 generation_aleatoire

```
1  function [orientations_aleatoires, M] = generation_aleatoire(N)
2
3  %generation_aleatoire
4  %
5  %genere un ensemble de valeurs sur la demi-sphere unite, selon une
       repartition uniforme, par "jittered sampling"
6  %
7  %[orientations_aleatoires, M] = generation_aleatoire(N)
8  %
9  %entree:
10 % N: nombre (approximatif) de vecteurs a generer
11 %
12 %sortie:
13 % orientations_aleatoires matrice a trois colonnes, avec des vecteurs
       unitaires en ligne
14 % M: nombre reel de valeurs aleatoires generees
15
16 %determination du nombre de valeurs de l'angle theta
17 m = fix(sqrt(N));
18
19 %determination du nombre reel de valeurs tirees
20 M= m^2;
21
22 %tirage des valeurs sur le carre unite
23 [gama] = rand(M,1);
24 [zeta] = rand(M,1);
25
26 %On re-repartit les valeurs sur le carre unite, en le pavant par M carres
27 %construction des matrices utiles
28 k=[1:M]';
29 s = (mod((k-1),m) + gama)*1/m;
30
31 t = (fix((k-1) / (m)) + zeta)*1/m;
32
33 %production des vecteurs aleatoires
34 racine = sqrt( prod([s (2*ones(M,1)- s)],2));
35 angle = 2*pi*t;
36 orientations_aleatoires = [prod([racine cos(angle)],2) prod([racine sin(
       angle)],2) 1-s];
37
38 end
```

### B.2.4 orientation_vecteurs_aleatoires

```
function [orientations_globales, ex, ey, ez] =
    orientation_vecteurs_aleatoires(orientations_locales, normale)
%
%permet de passer des vecteurs aleatoires generiques a des vecteurs
    orientes selon la normale a la surface
%
%[orientations_globales, ex, ey, ez] = orientation_vecteurs_aleatoires(
    orientations_locales, normale)
%
%entree: orientations_locales, matrice a 3 colonnes enumerant les vecteurs
     dans le repere local
%   normale: vecteur ligne a trois composantes, normale a la surface
%
%sortie:
% orientations_globales, matrice a 3 colonnes enumerant les vecteurs dans
    le repere global
% ex, ey, ez: vecteurs de la base locale, en ligne (utiles pour la BRDF)

%initialisation de la base de l'axe "des z" de la base locale
ez = (1/norm(normale))*normale';

%initialisation de la base de l'axe "des x" de la base locale
ex = zeros(3,1);

switch ez(2)
  case 0
    ex(2) = 0;
  otherwise
    ex(2) = -( ex(1) * ez(1) ) / ez(2);
end

ex(1) = sqrt(1 - ez(2));

%initialisation de la base de l'axe "des y" de la base locale
ey = cross(ez,ex);

%changement de base
orientations_globales  = orientations_locales * [ex ey ez];

end
```

### B.2.5 prochaine_intersection

```
function [intersect_bool, triangle, point, noeud] = prochaine_intersection(
    successeurs, predecesseurs, triangles_octants, octants, sommets,
    indices, point_actuel, noeud_actuel, orientation)

%prochaine_intersection
%
%[intersect_bool, triangle, point, noeud] = prochaine_intersection(
    successeurs, predecesseurs, triangles_octants, octants, sommets,
    indices, point_actuel, noeud_actuel, orientation)
%
%determine la prochaine intersection avec un triangle si elle existe
%
%entrees:
% successeurs, predecesseurs, triangles_octants, octants: donnees relatives
     a l'octree
% sommets, indices: donnees relatives a la geometrie
```

```
% point_actuel: point a partir duquel on effectue la recherche
% noeud_actuel: indice de l'onctant contenant ce point
% orientation: orientation de la recherche (vecteur unitaire)
%
%sorties:
% intersect_bool: 1 si une intersection est trouvee avant sortie du domaine, 0 sinon
% triangle: indice du triangle concerne par l'intersection (vide si aucun triangle rencontre)
% point: point d'intersection
% noeud: indice de l'octant contenant le triangle intersecte
%

%initialisation
noeud = noeud_actuel;

%test de l'octant courrant
[intersect_bool, triangle, point] = intersection_triangles(triangles_octants{noeud}, sommets, indices, orientation, point_actuel);

%poursuite de la recherche dans d'autres octants, si aucune intersection dans l'octant
if (intersect_bool == 0)||(~((point >= octants(noeud, [1:3]', 1))&&(point <= octants(noeud, [1:3]', 2))))
  intersect_bool = 0;

  %calcul de l'octant suivant
  [point_sortie, face] = sortie_octant(point_actuel, orientation, octants(noeud,:,:));
  [noeud] = prochain_octant(successeurs, predecesseurs, octants, noeud, point_sortie, face);

  %test de l'octant suivant
  if noeud ~= 0
    [intersect_bool, triangle, point, noeud] =...
      prochaine_intersection(successeurs, predecesseurs, triangles_octants, octants, sommets, indices,...
      point_sortie, noeud, orientation);
  end
end

end
```

### B.2.6 intersection_triangles

```
function [intersect_bool,triangle,point] = intersection_triangles(triangles,sommets,indices,orientation,point_tir,varargin)
%
%intersection_triangles
%
%[intersect_bool,triangle,point] = intersection_triangles(triangles,sommets,indices,orientation,point_tir (,espilon))
%teste l'intersection entre une droite et des triangles, et renvoie la premiere intersection rencontree
%
%entrees:
% triangles: vecteurs contenant les indices des triangles a tester
% sommets, indices: donnees geometriques telles que produites par entree_CAO
% orientation: vecteur ligne unitaire donnant la direction de tir
% point_tir: vecteur ligne coordonnees du point de tir
%   epsilon: distance minimale a l'intersection. optionel.
```

```
% /!\  cette fonction etant concue pour etre utilisee au coeur du programme, aucune verification n'est effectuee quant au format des donnees d'entree
%
%sorties:
% intersect_bool: 1 si au moins une intersection, 0 sinon
% triangle: indice du premier triangle rencontre
% point: vecteur ligne correspondant au point d'intersection avec le triangle

%distance minimale a l'intersection (pour eviter intersection avec la surface de tir)
if isempty(varargin)
    epsilon = 0.0000000001;
else
    epsilon = varargin{1};
end


%initialisation de la sortie
intersect_bool = 0;
triangle = [];
point = [];

%calcul des distances aux intersections
distances = distances_intersections(triangles,sommets,indices,orientation,point_tir);

%recherche de l'intersection la plus proche
dmin = Inf;
for i=1:length(distances)
  if (distances(i) < dmin)&&(distances(i) >= epsilon)
    dmin = distances(i);
    intersect_bool = 1;
    triangle = i;
  end
end

if dmin < Inf
  point = point_tir + dmin*orientation;
end

end
```

### B.2.7 distances_intersections

```
function [distances] = distances_intersections(triangles,sommets,indices,orientation,point)
%distances_intersections
%[distances] = distances_intersections(triangles,sommets,indices,orientation,point)
%
%revoie les distances entre un point et l'intersection aux triangles, selon une direction donnee
%fonction basee sur l'algorithme de test d'intersections de Moller
%
%entrees:
% triangles: vecteur contenant les numeros des triangles a tester
% sommets, indices: donnees geometriques telles que produites par entree_CAO
% orientation: vecteur ligne unitaire donnant la direction de tir
% point: vecteur ligne coordonnees du point de tir
```

```
% /!\  cette fonction etant concue pour etre utilisee au coeur du programme, aucune verification n'est effectuee quant au format des donnees d'entree
%
%sortie:
% distances: vecteur contenant les distances a l'intersection. Une distance infinie correspond a l'absence d'intersection

epsilon = 5*eps;
direction = (-1)*orientation';
point_tir = point';
n = length(triangles);
distances = Inf(n,1);

for i=1:n
  indice_courrant = indices( triangles(i),:);

  sommets_courrants =...
        [sommets(indice_courrant(1),:) ;
        sommets(indice_courrant(2),:);
        sommets(indice_courrant(3),:)]';

  A = [direction  (sommets_courrants(:,2) - sommets_courrants(:,1)) (sommets_courrants(:,3) - sommets_courrants(:,1))];

  if abs(det(A)) > 0
    tuv = inv(A)*(point_tir - sommets_courrants(:,1));
    if (tuv(2) >= 0)&&(tuv(3) >= 0)&&((tuv(2) + tuv(3))<=1 + epsilon)
      distances(i) = tuv(1);
    end
  end
end

end
```

### B.2.8 sortie_octant

```
function [point, face] = sortie_octant(point_entree, orientation, octant)

%sortie_octant
%
%[point, face] = sortie_octant(point_entree, orientation,octant)
%
%renvoie des informations sur le point de sortie d'un octant
%
%entrees:
% point_entree: vecteur ligne correspondant a la position actuelle du
%   point de calcul
% orientation: vecteur ligne unitaire correspondant a la direction de
%   recherche
% octant: matrice 1x3x2, extraite de la matrice des octants
%
%sorties:
% point: coordonnees en ligne du point de sortie
% face: matrice de vecteurs a deux elements: [coordonnee extremum]
%   si deux lignes sont presentes, le point de sortie est sur une arrete
%   si trois lignes sont presentes, le point de sortie est un coin

epsilon = 0.005;
unite4 = ones(4,1);
coord = [1:3];
face = [];
```

```
26
27 %on itere sur les faces, testant l'intersection avec la face
       potentiellement visee
28 for i=1:3
29   if abs(orientation(i)) > 0
30     %determination des proprietes de l'orientation: vise-t-on la face max
           ou min?
31     if orientation(i) > 0
32         face_test = 2;
33           else
34         face_test = 1;
35     end
36     %determination des autres axes
37     autres_axes = coord;
38     autres_axes(i) = [];
39
40     %creation de la matrice des sommets des triangles
41     sommets = zeros(4,3);
42     sommets(:,i) = unite4*octant(1, i, face_test);
43     sommets(:,autres_axes(1)) =...
44       [octant(1, autres_axes(1), 1);
45       octant(1, autres_axes(1), 2);
46       octant(1, autres_axes(1), 1);
47       octant(1, autres_axes(1), 2)];
48
49     sommets(:,autres_axes(2)) =...
50       [octant(1, autres_axes(2), 1);
51        octant(1, autres_axes(2), 2);
52        octant(1, autres_axes(2), 2);
53        octant(1, autres_axes(2), 1)];
54
55     %construction des donnees utiles a la fonction d'intersection
56     indices = [1 2 3; 1 2 4];
57     triangles = [1 2];
58
59     %test de l'intersection
60     [intersect_bool, triangle, point_intersect] = intersection_triangles(
           triangles,sommets,indices,orientation,point_entree);
61
62     if intersect_bool == 1
63       face = [face; i face_test];
64       point = point_intersect;
65         end
66   end
67 end
68
69 % si on arrive ici sans avoir affecte face, c'est que le point de sortie
       doit etre le point d'entree
70 % on veut cependant la face/arrete/coin adajacente a l'octant voisin:
71 % le non parcours dans l'octant peut donner lieu a des orientations
       etranges.
72 % on ne prend que les faces effectivement adjacentes au voisin
73 if isempty(face)
74   for i=1:3
75         for j=1:2
76             if ((orientation(i)>0)&&(j==2)&&(abs(point_entree(i) - octant
                   (1,i,j))<=epsilon))||...
77           ((orientation(i)<0)&&(j==1)&&(abs(point_entree(i) - octant(1,i,j)
                  )<=epsilon))
78                 point = point_entree;
79                 face = [face; i j];
80                 break
81             end
```

```
        end
  end
end

end
```

### B.2.9 prochain_octant

```
function [noeud] = prochain_octant(successeurs, predecesseurs, octants,
    octant_quitte, point_sortie, face_sortie)
%prochain_octant
%[noeud] = prochain_octant(octants, predecesseurs,successeurs,octant_quitte
    ,point_sortie,face_sortie)
%
%determine le prochain octant
%
%sortie:
% noeud: indice de l'octant penetre. nul si sortie du domaine
%
%entree:  octants, predecesseurs,successeurs : codage de l'octree tel que
    defini par la fonction creation octree
%   octant_quitte: indice de l'octant quitte
%   point_sortie: vecteur tridimensionnel
%   face_sortie: matrice de vecteurs a deux composantes, de la forme [
    coordonnee, ext], avec coordonnee = 1, 2 ou 3 et ext= 1 (min) ou 2 (max
    )
%

%initialisation de la plus grande valeur consideree comme nulle
epsilon = 0.001;
coord = face_sortie(:,1);
extr_s = face_sortie(:,2);
vois_type = length(coord);
unite = ones(vois_type,1);

%matrice d'indexation de la sortie:
sortie = zeros(3,2);
for i=1:vois_type
  sortie(coord(i),extr_s(i)) = 1;
end
sortie = logical(sortie);

%test de la sortie du domaine: on teste l'appartenance des diverses faces
    adjacentes au voisin a la limite
if ~(abs( octants(octant_quitte,sortie) - octants(1,sortie) ) > epsilon)
  noeud = 0;
  return
end

%initialisation valeurs utiles
extr_e = mod(extr_s, 2) + unite;
entree = zeros(3,2);
for i=1:vois_type
  entree(coord(i),extr_e(i)) = 1;
end
entree = logical(entree);


coord_autres = 1:3;
coord_autres(coord) = [];

%remontee jusqu'au premier octant contenant strictement le point de sortie:
```

```
49 noeud = octant_quitte;
50
51 %tant que l'une des faces de sortie fait encore partie des limites de l'
       octant courrant, on remonte
52 while ~(abs( octants(noeud,sortie) - octants(octant_quitte,sortie) ) >
       epsilon)
53   noeud = predecesseurs(noeud);
54 end
55
56 %descente jusqu'a la feuille correspondante
57 while successeurs(noeud,1) ~= 0
58   for i=1:8
59     if  (abs( octants(successeurs(noeud,i), entree) - octants(octant_quitte
           , sortie) ) < epsilon)...
60       &&(...
61         (isempty(coord_autres))||...
62         (...
63           (octants(successeurs(noeud,i),coord_autres,1) < point_sortie(
               coord_autres))...
64           &&(octants(successeurs(noeud,i),coord_autres,2) > point_sortie(
               coord_autres))...
65         )...
66       )
67
68         noeud = successeurs(noeud,i);
69         break;
70     end
71   end
72 end
73
74 end
```

### B.2.10 reflection

```
1  function [nouvelles_orientations, coefficients, M] = reflection(normale,
       caracteristiques, orientation_incidente, N, varargin)
2
3  % brdf_schlick
4  %
5  %[nouvelles_orientations, coefficients, M] = reflection(normale,
       caracteristiques, orientation_incidente, N)
6  %
7  % OU
8  %
9  %[nouvelles_orientations, coefficients, M] = reflection(normale,
       caracteristiques, orientation_incidente, N, orientation_reflechie)
10 %
11 % produit des orientations ponderees, en utilisant la BRDF de Christophe
       Schlick
12 %
13 % entrees:
14 %   normale: normale a la surface de reflection
15 %   caracteristiques: vecteur des caracteristiques de la surface: [albedo
       rugosite isotropie]
16 %   orientation_incidente: vecteur unitaire en ligne oriente vers la
       surface
17 %   N: nombre de valeurs a sampler
18 %     OU
19 %   N: nombre de valeurs tirees au point
20 %   orientation_reflechie: orientation pour laquelle le coefficient est
       voulu (vecteur unitaire oriente vers le ciel)
21 %
```

```
22 % sorties:
23 %   nouvelles_orientations: matrice a trois colonnes, listant les
       oriantations en vecteurs unitaires
24 %   coefficients: vecteur colonne, listant les coefficients non nuls
       relatifs a chaque orientation
25 % M: nombre de valeurs effectivement samplees, en comptant les valeurs
26 %   a coefficient nul (non listees dans les deux vecteurs precedents)
27
28 %definition de la plus petite valeur consideree comme nulle
29 epsilon = 0.000001;
30
31 %orientation correcte de la normale: le produit scalaire de l'orientation
       incidente et de la normale doit etre negatif (pointage "vers l'
       exterieur")
32 %le diviseur est non nul dans le cas d'une intersection
33 normale_orientee = -(dot(orientation_incidente,normale) / abs(dot(
       orientation_incidente,normale)))*normale;
34
35 c = caracteristiques(1);
36 r = caracteristiques(2);
37 p = caracteristiques(3);
38
39 %generation des nouvelles orientations
40 if isempty(varargin)
41   [orientations, M] = generation_aleatoire(N);
42   [orientations, ex, ey, ez] = orientation_vecteurs_aleatoires(orientations
         , normale_orientee);
43 else
44   orientations = varargin{1};
45   ez = normale_orientee';
46   ey = [0; dot(ez,[0 0 1]); dot(ez,[0 (-1) 0])];
47   M = N;
48 end
49
50 %application de la BRDF aux orientations
51 coefficients = [];
52 nouvelles_orientations = [];
53
54 V = -orientation_incidente';
55 v = dot(V,ez);
56 compteur = 1;
57
58 for i=1:size(orientations,1)
59   H = (V + (orientations(i,:))') / norm(V + (orientations(i,:))');
60   Hb = H - (dot(H,ez)*ez);
61   t = dot(H,ez);
62   u = dot(H,V);
63   vp = dot(orientations(i,:),ez);
64   w = dot(ey,Hb);
65
66   %calcul du coefficient
67   %pose un probleme pour prendre en compte le cas speculaire parfait
68   f_r =   (c + (1 - c)*((1 - u)^5)) *...
69     (1/(4*pi*v*vp)) * (r / (1 + r*t*t - t*t)^2) *...
70     sqrt(p / (p^2 - (p^2)*(w^2) + w^2));
71
72   %calcul du coefficient monte-carlo
73   %on ne prend que les orientations de coef
74   %strictement positif
75   coef = (vp * pi^2 * f_r) / (M);
76   if abs(coef) > epsilon
77     coefficients(compteur,1) = coef;
78     nouvelles_orientations(compteur,:) = orientations(i,:);
```

```
    compteur = compteur + 1;
  end
end

end
```

## B.3 Calcul énergétique

### B.3.1 traitement_ponctuel

```
function [Etot, Fmoy, irrad, irr_dir, irr_diff, tempsdir] =
    traitement_ponctuel(coefficients, orientations, l, j, h, rad_globale,
    rad_diffuse, option, ind_maill, somm_maill, norm_maill, sommets,
    normales, indices, caracteristiques, chemin, successeurs, predecesseurs
    , octants, triangles_octants)
%
% traitement_ponctuel
%
% fonction qui calcule l'energie totale et le flux energetique moyen sur la
     surface, en un instant unique
%
%[Etot, Fmoy, irrad] = traitement_ponctuel(coefficients, orientations, l, j
    , h, rad_globale, rad_diffuse, option, ind_maill, somm_maill,
    norm_maill, sommets, normales, indices, caracteristiques, chemin,
    successeurs, predecesseurs, octants, triangles_octants)
%
%
% entrees:
%   orientations, coefficients: tableaux de vecteurs, resultats du lancer
    de rayons
%   latitude: latitude en degres
%   jour: jour de l'annee, compte depuis le 1er janvier (1 janvier: jour =
    1; 31 decembre: jour = 365)
%   heure: heure de la journee, en heures (par exemple, 12h30: heure=12,5)
%   rad_globale, rad_diffuse: donnees sur le type de ciel. depend de option
    :
%     option=1: rad_globale et rad_diffuse sont les irradiances globale et
    diffuse en W/m^2, respectivement
%     option=2: rad_globale et rad_diffuse sont le "clear sky index" et le
    "cloudless index", respectivement (se referer au tableau liant ces
    valeurs au type de ciel
%   ind_maill, somm_maill, norm_maill: indices, sommets et normale du
    maillage
%   indices, sommets: donnees du contexte, facultatif (a des fins de
    representation seulement)
%
% sorties:
%   Etot: energie totale recue par la surface, en W
%   Fmoy: flux moyen sur la surface, en W/m^2
%   irradiances aux points, en W/m^2

%altitude de l'horizon
epsilon = 0;

%initialisation des donnees solaires:
[alt_sol, azimuth_sol] = position_solaire(l, j, h);

%initialisation du nombre de points
N = length(coefficients);

irr_dir = zeros(N,1);
```

```
irr_diff = zeros(N,1);
irrad = zeros(N,1);

%calcul des coefficients pour la radiation directe
[coefficients_dir, tempsdir] = radiation_directe(successeurs, predecesseurs, octants, triangles_octants, sommets, normales, indices, caracteristiques, chemin, orientations, alt_sol, azimuth_sol);

%calcul de l'irradiance diffuse en chaque point
for i=1:N

  alt_points = asin(orientations{i}(:,3));

  %le modele d'Igawa et al. ne traite pas de maniere plausible les points d'altitude negative ou nulle
  %(points la plupart du temps sous l'horizon)
  %on elimine donc de tels points avant passage au modele
  %ceci est pertinent pour la plupart des cas en milieu urbain
  choix = logical(alt_points>epsilon);
  alt_points = alt_points(choix);
  coefficients_points = coefficients{i}(choix);

  azimuth_points = acos( -orientations{i}(choix,2) ./ cos(alt_points) );

  %calcul des radiances celestes
  [radiances, radiance_dir] = modele_ciel(alt_sol, azimuth_sol, alt_points, azimuth_points, rad_globale, rad_diffuse, option);

  %calcul de l'irradiance au point
  irr_dir(i) = sum((coefficients_dir{i} .* radiance_dir));
  irr_diff(i) = sum((coefficients_points .* radiances));
end
%l'erreur de calcul peut donner, du fait des nombreux acos et asin,
%des valeurs d'irradiance complexes de partie imaginaire negligeable.
%on les elimine ici
choix = (imag(irr_dir) ~= 0);
if ~isempty(irr_dir(choix))
  fprintf('\n')
  fprintf('suppression de la partie imaginaire %g\n',imag(irr_dir(choix)));
  fprintf('\n')
  irr_dir = real(irr_dir);
  clear choix;
end

choix = (imag(irr_diff) ~= 0);
if ~isempty(irr_diff(choix))
  fprintf('\n')
  fprintf('suppression de la partie imaginaire %g\n',imag(irr_diff(choix)));
  fprintf('\n')
  irr_diff = real(irr_diff);
  clear choix;
end


%calcul de l'irradiance totale
irrad = irr_dir + irr_diff;

%integration
Etot = integration(irrad, ind_maill, somm_maill, norm_maill(1,:));
Fmoy = Etot / integration(ones(N,1), ind_maill, somm_maill, norm_maill(1,:));

```

```
93 %representation
94 hold on
95 if ~isempty(varargin)
96   indices = varargin{1};
97   sommets = varargin{2};
98
99   trimesh(indices, sommets(:,1), sommets(:,2), sommets(:,3), 'FaceAlpha',
       0);
100 end
101
102 trisurf(ind_maill, somm_maill(:,1), somm_maill(:,2), somm_maill(:,3), irrad
    );
103
104 shading('interp');
105 colormap(hot);
106 colorbar;
107 title('irradiance sur la surface, en W/m^2');
108 hold off
109
110 end
```

### B.3.2 position_solaire

```
1  function [alt, azimuth] = position_solaire(l, j, h)
2  % position_solaire
3  %
4  % retourne la position solaire sur la base de donnees spatio-temporelles
5  %
6  % [alt, azimuth] = position_solaire(latitude, jour, heure)
7  %
8  % entrees:
9  %   latitude: latitude en degres
10 %   jour: jour de l'annee, compte depuis le 1er janvier (1 janvier: jour =
       1; 31 decembre: jour = 365)
11 %   heure: heure de la journee, en heures (par exemple, 12h30: heure=12,5)
12 %
13 % sorties:
14 %   alt: hauteur solaire en radians
15 %   azimut: angle par rapport au sud, en radians
16
17
18 %calcul des donnees de base: declinaison et deviation, en degres
19 decl = 23.45*sin( ((j + 284)/365)*2*pi);
20 dev = 15*(12 - h);
21
22 %calcul des sorties, en radians
23 alt = asin(...
24   sind(l)*sind(decl) + cosd(decl)*cosd(dev)*cosd(l)...
25   );
26
27 if cos(alt) ~= 0
28   azimuth = asin(...
29     ( cosd(decl)*sind(dev) ) / cos(alt)...
30     );
31 else
32   %definition d'une valeur quelcquonque de l'azimuth si le soleil est au
       zenith
33   azimuth = 0;
34 end
35 end
```

### B.3.3 modele_ciel

```
function [radiances, radiance_dir] = modele_ciel(alt_soleil, azimuth_soleil, alt_points, azimuth_points, rad_globale, rad_diffuse, option)
% modele_ciel
%
% fonction qui calcule les radiances en chaque point, pour une position solaire donnee, par le modele de Igawa et al.
%
% [radiances, radiance_dir] = modele_ciel(alt_soleil, azimuth_soleil, alt_points, azimuth_points, rad_globale, rad_diffuse, option)
%
% entrees:
%   alt_soleil, azimuth_soleil: altitude et azimuth solaires, en radians
%   alt_points, azimuth_points: vecteurs recensant les coordonnees des points, en radians
%   rad_globale, rad_diffuse: donnees sur le type de ciel. depend de option :
%     option=1: rad_globale et rad_diffuse sont les irradiances globale et diffuse en W/m^2, respectivement
%     option=2: rad_globale et rad_diffuse sont le "clear sky index" et le "cloudless index", respectivement
% sorties:
%   radiances: radiances *celestes* aux points vises
%   radiance dir: radiance du disque solaire (ratio de l'irradiance directe par l'angle solide apparent du soleil)

%rayon angulaire apparent du soleil (rad)
Rsol = 4.66E-3;

%initialisation: constante solaire - energie incidente au sommet de l'atmosphere:
Eeo = 1367;

%initialisation: matrice de calcul des coefficients pour la radiance au zenith
C = zeros(6,7,5);

C(:,:,1) =...
  [0.4015 -0.0564 -0.1460 1.0626 -2.8711 2.7842 -0.9167;
  0.0044 -0.0316 -0.0829 0.3772 0.0930 -0.8960 0.5302;
  0.0090 0.4062 -3.7111 11.6859 -16.8828 11.3017 -2.8773;
  -0.0200 -0.4699 4.3546 -12.3903 15.9467 -8.4637 1.3678;
  0.0177 0.3300 -1.9825 1.8522 4.1516 -8.4370 3.8397;
  -0.0068 -0.0817 0.2833 1.1362 -4.8599 5.9466 -2.3236];

C(:,:,2) =...
  [-0.0509 0.2509 1.6549 -15.2722 31.5218 -26.2167 7.8350;
  -0.0767 -0.8417 14.3852 -48.3263 64.8618 -38.0729 8.0616;
  -0.0842 -0.8060 -3.9888 54.5553 -137.6762 131.5008 -44.0612;
  0.2446 0.3678 9.7897 -100.4209 254.2300 -245.8816 83.3628;
  -0.2173 -0.7237 -18.4959 150.9138 -344.9908 313.3116 -101.3222;
  0.0888 0.1990 7.8043 -61.5252 140.6794 -128.3895 41.7667];

C(:,:,3) =...
  [0.0863 -1.2699 -11.1184 61.4569 -109.9849 85.2745 -24.2309;
  0.3435 2.8287 -41.5866 112.7302 -112.2037 38.0638 0.0588;
  0.3359 3.9616 20.6861 -220.7694 496.1507 -442.3932 142.5802;
  -1.0970 -2.6865 -44.2429 428.3022 -971.1740 872.1058 -283.3008;
  0.8603 3.9671 86.0313 -617.6864 1266.6167 -1067.6426 329.5323;
  -0.3572 -1.1026 -37.2452 258.3140 -530.7575 450.5763 -140.1650];

C(:,:,4) =...
```

```
  [-0.3754  1.0003  17.6094  -79.6806  133.1647  -98.6168  26.7019;
  -0.5077  -4.1306  47.1414  -99.9641  57.7891  18.6492  -19.1604;
  -0.3234  -5.5006  -47.5262  345.6662  -701.8434  597.8211  -189.0674;
  1.7185  6.1345  93.5278  -681.3328  1398.0054  -1197.9089  382.3517;
  -1.0320  -7.2571  -149.2886  899.2027  -1698.4940  1369.3651  -414.3978;
  0.4479  2.2956  63.7066  -377.9488  719.1565  -585.7678  178.9761];

C(:,:,5) =...
  [0.1944  -0.1519  -8.0731  32.8606  -52.2028  36.8154  -9.3016;
  0.2274  1.9924  -17.6413  27.4527  2.5328  -28.3222  13.7469;
  -0.0077  2.0469  28.9888  -171.1919  327.9505  -273.5933  86.4222;
  -0.5003  -2.7390  -57.2731  343.9805  -665.1001  558.7982  -178.8391;
  0.1200  2.9337  80.2519  -422.7656  758.4764  -599.2154  181.5249;
  -0.0673  -0.9299  -34.0204  178.5947  -323.9300  259.0233  -79.2551];


%masse optique relative de l'air
m = 1/(...
  cos((pi/2) - alt_soleil) +...
  (0.50572 * (96.07995 - (360/pi)*(pi/2 - alt_soleil))^(-1.6364))...
  );

%irradiance globale standard
Seeg = 0.84 * (Eeo / m) * exp(-0.0675 * m);

%standard cloud ratio
Ces = 0.01299 + 0.07698*m - 0.003857*(m^2) + 0.0001054*(m^3) -
    0.000001031*(m^4);

%type de ciel
switch option
  case 1
    Kc = rad_globale / Seeg;
    Cle = (1 - (rad_diffuse / rad_globale)) /...
       (1 - Ces);
    Eed = rad_diffuse;

    %radiance du disque solaire
    radiance_dir = (rad_globale - rad_diffuse) / (2*pi*Rsol);
  case 2
    Kc = rad_globale;
    Cle = rad_diffuse;
    Eed = (1 - (Cle*(1 - Ces))) * Kc * Seeg;

    %radiance du disque solaire
    radiance_dir = ((Kc*Seeg) - Eed) / (2*pi*Rsol);
end

Si = Kc + sqrt(Cle);

%coefficients des fonctions
a = (4.5 / (1 + 0.15*exp(3.4*Si))) - 1.04;
b = (-1 / (1 + 0.17*exp(1.3*Si))) - 0.05;
c = 1.77*((1.22*Si)^3.56)*(exp(0.2*Si))*((2.1 - Si)^0.8);
d = -3.05 / (1 + 10.6*exp(-3.4*Si));
e = 0.48 / (1 + 245*exp(-4.13*Si));

%radiance au zenith
radiance_zenith = 0;
B = zeros(7,5);
A = zeros(1,5);
for k=1:5
  for j=1:7
```

```
114     for i=1:6
115       B(j,k) = B(j,k) + C(i,j,k)*(alt_soleil^(i-1));
116     end
117     A(k) = A(k) + (B(j,k)*(Cle^((j-1)/2)));
118   end
119   radiance_zenith = radiance_zenith + (A(k)*(Kc^(k-1)));
120 end
121 radiance_zenith = radiance_zenith * Eed;
122
123 %calcul de la radiance diffuse en chaque point
124 radiances = radiancecal(alt_soleil, azimuth_soleil, radiance_zenith, a, b,
        c, d, e, alt_points, azimuth_points);
125
126 end
127
128 %%%%%%%%%%%%%%%%%%%%%%%%%%
129 function [radiance] = radiancecal(alt_soleil, azimuth_soleil,
        radiance_zenith, a, b, c, d, e, alt_point, azimut_point)
130 % fonction calculant la radiance
131
132 %pre-calcul
133 zeta = acos(...
134   (sin(alt_soleil) .* sin(alt_point)) +...
135   (cos(alt_soleil) .* cos(alt_point) .* cos(azimuth_soleil - azimut_point))
          ...
136   );
137
138 %radiance diffuse
139 radiance = radiance_zenith .* (...
140   (phi(alt_point, a, b) .* f(zeta, c, d, e)) ./...
141   (phi((pi/2) , a, b) .* f(((pi/2) - alt_soleil), c, d, e))...
142   );
143
144 end
145
146 %%%%%%%%%%%%%%%%%%%%%%%%%%
147 function [val] = phi(x,a,b)
148   val = 1 + (a*exp(b ./ sin(x)));
149 end
150
151 %%%%%%%%%%%%%%%%%%%%%%%%%%
152 function [val] = f(x, c, d, e)
153   val = 1 + (c*(exp(d*x) - exp(d*pi/2))) + (e*((cos(x)).^2));
154 end
```

### B.3.4 radiation_directe

```
1 function [coefficients_dir, temps] = radiation_directe(successeurs,
      predecesseurs, octants, triangles_octants, sommets, normales, indices,
      caracteristiques, chemin, orientations_finales, alt_soleil,
      azimuth_soleil)
2 %
3 % effectue un lancer de rayons depuis ls points de reflexion pour le calcul
       du rayonnement provenant du soleil
4 %
5 %
6 %[coefficients_dir, temps] = radiation_directe(successeurs, predecesseurs,
      octants, triangles_octants, sommets, normales, indices,
      caracteristiques, chemin, orientations_finales, coefficients_finaux,
      alt_soleil, azimuth_soleil)
7 %
8 % sorties:
```

```
%   coefficients_dir: tableau de vecteurs recensant les coefficients
    associes a la direction solaire
%   temps: temps de calcul
%
% entrees:
%   successeurs, predecesseurs, octants, triangles_octants: donnees de l'
    octree
%   sommets, normales, indices, caracteristiques: donnees geometriques
%   p_refl, t_refl, c_refl, n_refl: donnees des points de reflection
%   alt_soleil, azimuth_soleil: position solaire

%rayon angulaire apparent du soleil (rad)
Rsol = 4.66E-3;

temps = cputime;

orientation_solaire = [(-cos(alt_soleil)*sin(azimuth_soleil)) (-cos(
    alt_soleil)*cos(azimuth_soleil)) sin(alt_soleil)];

for i=1:length(chemin)
  fprintf('calcul de la radiation directe au point %i\n',i)
  coefficients_dir{i} = [];
  n_orientations_pt = length(chemin{i}.pointsfinaux);
  comptage = [1:n_orientations_pt]';
  unite = ones(n_orientations_pt,1);

  for j=1:size(chemin{i}.points,1)
    %test de la visibilite du soleil
    [intersect_bool] = prochaine_intersection(...
      successeurs, predecesseurs, triangles_octants, octants, sommets,
          indices,...
      chemin{i}.points(j,:), chemin{i}.noeuds(j), orientation_solaire);

    %si le soleil est visible, on calcule le coefficient associe
    if (intersect_bool == 0)
      %on calcule le coefficient de reflexion seulement si le point
      %n'appartient pas a la surface de calcul
      if j >1
        [nouvelles_orientations, coefficients_dir{i}(j)] = reflection(...
          normales(chemin{i}.triangles(j),:), caracteristiques(chemin{i}.
              triangles(j),:),...
          chemin{i}.orientations(j,:), chemin{i}.ntirs(j),
              orientation_solaire);

        %on prend en compte les coefficients negatifs
        %(non retournes par la fonction de reflection)
        %pour eviter plus loin l'utilisation de valeurs
        %inexistantes dans les calculs
        if isempty(nouvelles_orientations)
          coefficients_dir{i}(j)=0;
        end

        clear nouvelles_orientations;
      else
        coefficients_dir{i}(j) = 1;
      end

      coefficients_dir{i}(j) = coefficients_dir{i}(j)*chemin{i}.coefs(j);

      %test du nombre de tirs initialement effectuee dans le disque solaire
      %
      %on choisi les orientations dont le point correspond au point
      %courant
```

```
66        choix = logical(...
67          (chemin{i}.points(chemin{i}.pointsfinaux,:) ==...
68          unite*chemin{i}.points(j,:))...
69          );
70        choix = prod(choix, 2);
71        choix = comptage(logical(choix));
72
73        %calcul de l'ecart "en cosinus"
74        ecart = 1 .- (orientations_finales{i}(choix,:) * orientation_solaire
              ');
75
76        %calcul de l'ecart en radians
77        ecart = acos(ecart);
78
79        n_disque = sum(logical(abs(ecart) <= Rsol));
80        if n_disque > 1
81          coefficients_dir{i}(j) = coefficients_dir{i}(j)*n_disque;
82        end
83
84      end
85
86    end
87  end
88
89  temps = cputime - temps;
90
91  end
```

### B.3.5 integration

```
1  function [Etot] = integration(irradiances, indices, sommets, normale)
2  % integration
3  %
4  % integre l'irradiance sur la surface de calcul pour connaitre l'energie
       totale
5  %
6  % [Etot] = integration(irradiances, indices, sommets, normale)
7  %
8  % entrees:
9  %   irradiances: vecteur listant les irradiances en chaque point
10 %   indices, sommets, normale: donnees du maillage
11 %
12 % sortie:
13 %   Etot: energie totale recue par la surface
14
15 if size(sommets,1) ~= length(irradiances)
16   error('les irradiances ne sont pas affectees identiquement aux sommets');
17 end
18
19 N = size(indices,1);
20 unit3 = ones(3,1);
21
22 coefs = 0*irradiances;
23
24 %passage aux coordonnees locales
25 [sommets_local, ex, ey, ez] = orientation_vecteurs_aleatoires(sommets,
       normale);
26 sommets_local = sommets_local(:, 1:2);
27
28 %assemblage
29 for i=1:N
30   %determination des sommets du triangle:
```

```
31    a = indices(i,1);
32    b = indices(i,2);
33    c = indices(i,3);
34
35    det_jac = abs(...
36      (sommets_local(b,1) - sommets_local(a,1))*...
37      (sommets_local(c,2) - sommets_local(a,2))-...
38      (sommets_local(b,2) - sommets_local(a,2))*...
39      (sommets_local(c,1) - sommets_local(a,1))...
40      );
41
42    coefs([a b c]) = coefs([a b c]) + unit3*(det_jac/6);
43  end
44
45  Etot = dot(coefs,irradiances);
46
47  end
```

# Bibliographie


[Aronov 05] Boris Aronov, Hervé Brönnimann, Allen Y. Chang & Yi-Jen Chiang. *Cost-driven octree construction schemes : an experimental study.* Computational Geometry, vol. 31, no. 1-2, pages 127 – 148, 2005. Special Issue on the 19th Annual Symposium on Computational Geometry - SoCG 2003.

[Atkinson 08] Gary A. Atkinson & Edwin R. Hancock. *Two-dimensional BRDF estimation from polarisation.* Computer Vision and Image Understanding, vol. 111, 2008.

[Badescu 10] Viorel Badescu. *Testing 52 models of clear sky solar irradiance computation under the climate of Romania.* In Solar Energy at Urban Scale, Compiègne, 2010. Université de Technologie de Compiègne.

[Bienvenu 10] Laurent Bienvenu & Mathieu Hoyrup. *Une brève introduction à la théorie effective de l'aléatoire.* Gazette de la Société Mathématique de France, vol. 123, pages 35 – 47, 2010.

[Brönnimann 06] Hervé Brönnimann & Marc Glisse. *Octrees with near optimal cost for ray-shooting.* Computational Geometry, vol. 34, no. 3, pages 182 – 194, 2006.

[Chen 88] Homer H. Chen & Thomas S. Huang. *A survey of construction and manipulation of octrees.* Computer Vision, Graphics, and Image Processing, vol. 43, no. 3, pages 409 – 431, 1988.

[Chen 06] Y. Chen & K.N. Liou. *A Monte Carlo method for 3D thermal infrared radiative transfer.* Journal of Quantitative Spectroscopy and Radiative Transfer, vol. 101, no. 1, pages 166 – 178, 2006.

[Denoeux 09] Thierry Denoeux & Gérard Govaert. SY02, cours de statistiques. Université de Technologie de Compiègne, printemps 2009.

[Doyle 98] J. P. Doyle & H. Rief. *Photon transport in three-dimensional structures treated by random walk techniques : Monte Carlo benchmark of ocean colour simulations.* Mathematics and Computers in Simulation, vol. 47, no. 2-5, pages 215 – 241, 1998.

[He 91] Xiao D. He, François X. Torrance Kenneth E.and Sillion & Donald P. Greenberg. *A Comprehensive Physical Model for Light Reflection.* Computer Graphics, vol. 25, no. 4, 1991.

[Igawa 04] Norio Igawa, Yasuko Koga, Tomoko Matsuzawa & Hiroshi Nakamura. *Models of sky radiance distribution and sky luminance distribution.* Solar Energy, vol. 77, no. 2, pages 137 – 157, 2004.

[Juan-Arinyo 95] Robert Juan-Arinyo & Jaume Solé. *Constructing face octrees from voxel-based volume representations.* Computer-Aided Design, vol. 27, no. 10, pages 783 – 791, 1995.

[Kajiya 86] James T. Kajiya. *The rendering equation.* In Computer Graphics, pages 143–150, 1986.

[Kastendeuch 09] P.P. Kastendeuch & G. Najjar. *Simulation and validation of radiative transfers in urbanised areas.* Solar Energy, vol. 83, no. 3, pages 333 – 341, 2009.

[Keller 01] Alexander Keller. *Hierarchical Monte Carlo image synthesis.* Mathematics and Computers in Simulation, vol. 55, no. 1-3, pages 79 – 92, 2001.

[Kocifaj 09] M. Kocifaj. *Sky luminance/radiance model with multiple scattering effect.* Solar Energy, vol. 83, no. 10, pages 1914 – 1922, 2009.

[L. Cook 84] Robert L. Cook, Thomas Porter & Loren Carpenter. *Distributed ray tracing.* Computer Graphics, vol. 18, no. 3, july 1984.

[Lafortune ] Eric P. F. Lafortune, Sing-Choong Foo, Kenneth E. Torrance & Donald P. Greenberg. *Non-Linear Approximation of Reflectance Functions.* Program of Computer Graphics.

[Lagae 05] Ares Lagae & Philip Dutré. *An Efficient Ray-Quadrilateral Intersection Test.* Journal of Graphics Tools, vol. 10, no. 4, pages 23–32, October 2005.

[Madras 02] Neal Madras. Lectures on monte carlo methods, volume 16 of *Fields Institute Monographs.* American Mathematical Society, Providence, RI, 2002.

[Maréchal 04] Loïc Maréchal. *Maillage hexaédrique par octree : avancées et limites.* INRIA, 2004.

[Maïzia 07] Mindjid Maïzia & Olivier Jung. TU01, confort thermique dans les espaces fermés et ouverts urbains. 2007.

[Meister 98] Gerhard Meister, Rafael Wiemker, René Monno, Hartwig Spitzer & Alan Strahler. *Investigation on the Torrance-Sparrow Specular BRDF Model.* In Investigation on the Torrance-Sparrow Specular BRDF Model. IEEE, 1998.

[Möller 97] Tomas Möller & Ben Trumbore. *Fast, minimum storage ray-triangle intersection.* Journal of Graphics Tools, vol. 2, no. 1, pages 21–28, 1997.

[Ozturk 08] Aydin Ozturk, Murat Kurt, Ahmet Bilgili & Cengiz Gungor. *Linear approximation of Bidirectional Reflectance Distribution Functions.* Computers & Graphics, vol. 32, no. 2, pages 149 – 158, 2008.

[Quarteroni 00] Alfio Quarteroni, Riccardo Sacco & Fausto Saleri. numerical mathematics, volume 37 of *Texts in applied mathematics.* Springer, 2000.

[Robinson 04] Darren Robinson & Andrew Stone. *Solar radiation modelling in the urban context.* Solar Energy, vol. 77, no. 3, pages 295 – 309, 2004.

[Samet 89] Hanan Samet. *Neighbor finding in images represented by octrees.* Computer Vision, Graphics, and Image Processing, vol. 46, no. 3, pages 367 – 386, 1989.

[Schlick 94] Christophe Schlick. *An inexpensive brdf model for physically-based rendering.* In Computer Graphics Forum, volume 13, pages 233–246, 1994.

[Sylvain 05] Michel Sylvain. *Diffuse reflection by rough surfaces : an introduction.* Comptes Rendus Physique, vol. 6, no. 6, pages 663 – 674, 2005. Interaction of electromagnetic fields with the environment.

[Szirmay-Kalos 00] László Szirmay-Kalos. Monte-carlo methods in global illumination. Institute of Computer Graphics, 2000.

[Utc 07] Utc. MT09, analyse numérique élémentaire. Université de Technologie de Compiègne, juin 2007.

[Uni 03] University of California. Monte carlo ray tracing, San Diego, juillet 2003. Siggraph.

# Glossaire

**A**

**allocation proportionnelle** méthode de réduction de la variance par stratification consistant à générer sur chaque sous domaine un nombre de valeurs proportionnel au rapport de la mesure du sous-domaine à celle du domaine. 5, 69

**B**

**BRDF** Bidirectionnal Reflection Density Function, fonction décrivant la proportion du flux lumineux réfléchi dans une direction selon la direction du rayon incident. 8, 9, 16, 70

**E**

**échantillonage préférentiel** classe de méthodes de réduction de la variance reposant sur la combinaison de diverses lois de densité de probabilité. 5

**équation de rendu** en rendu d'images, équation intégrale décrivant la radiance observée en un point à partir de la géométrie de la scène et des caractéristiques des surfaces. 7, 8

**F**

**flux** *i.e.* flux radiatif : puissance des radiations reçues, en $W$. 8, 16, 22, 23

**G**

**génération pseudo-aléatoire** Nom donnée à la génération déterministe de suites mimant une distribution aléatoire. 4, 5, 9

**I**

**illuminance** équivalent de l'irradiance dans le domaine visuel, en $lm.m^{-2}$. 13, 14

**irradiance** Puissance lumineuse reçue par unité de surface, en $W.m^{-2}$. 13, 14, 16, 18–21, 69

**J**

**jittered sampling** méthode de réduction de la variance par stratification avec allocation proportionnelle, consistant à discrétiser un domaine en autant de sous-domaines que l'on génère de points. 5, 17

**L**

**loi de conservation de l'énergie** loi physique stipulant que l'énergie sortant d'un système est la somme des énergies rejetées et émises, moins l'énergie absorbée. 9, 14

**loi de Fresnel** Loi décrivant le comportement d'une onde électromagnétique à l'interface entre deux milieux de caractéristiques différentes. 15

**luminance** équivalent de la radiance dans le domaine visuel : radiance spectrale pondérée par la sensibilité de l'œil humain aux diverses longueurs d'ondes, en $lm.m^{-2}.sr^{-1}$. 13, 14

## M

**modèle de ciel** modèle décrivant la répartition de la radiance sur la voûte céleste. 1, 12, 19, 24

**méthode de Monte-Carlo** classe de méthodes de simulation mathématique reposant sur la génération de valeurs aléatoires. 1, 3, 5, 6, 70

**méthode de réduction de la variance** classe de méthodes permettant de réduire la variance du résultat obtenu par méthode de Monte-Carlo. 5, 69, 70

## O

**octant** nom donné au sous-domaine cubique correspondant à un nœud dans un octree. 11, 26, 28, 30, 35, 70

**octree** arbre dans lequel chaque nœud non terminal a exactement 8 fils. Ce type d'arbre est utilisé pour organiser une subdivision d'un domaine cubique, dans laquelle chaque domaine cubique est subdivisé en huit sous-domaines, nommés octants. 10, 11, 26–28, 30, 31, 35, 70

## P

**principe de symétrie de Helmholtz** loi physique spécifiant que la direction incidente et la direction réfléchie peuvent être inversées dans la BRDF. 9, 14

## Q

**quasi-Monte-Carlo** qualificatif donné aux méthodes reprenant le raisonnement des méthodes de Monte-Carlo, tout en utilisant des valeurs fixées *a priori*. 4

## R

**radiance** Puissance lumineuse émise par une surface selon une direction, en $W.m^{-2}.sr^{-1}$. 7, 8, 13, 14, 16–20, 22, 69, 70

**radiation diffuse** part de la radiation solaire provenant de la voûte céleste, du fait de phénomènes optiques divers dans l'atmosphère. 13, 16, 18, 19, 23, 24

**radiation directe** part de la radiation solaire provenant du disque solaire. 16, 19, 23, 24

**réflexion lambertienne** réflexion de la radiation uniforme dans toutes les directions (réflexion «diffuse», surface «rugueuse»). 8, 15

**réflexion spéculaire** réflexion de la radiation pour laquelle une bijection existe entre la direction incidente et la direction réfléchie («miroir parfait», «surface de Fresnel»). 8, 9, 15

## S

**stratification** classe de méthodes de réduction de la variance reposant sur la partition du domaine en divers sous-domaines, sur lesquels des valeurs aléatoires sont générées. 5, 69