Reconstruction sur GPU de Metropolis Instant Radiosity

Anthony Pajot, Loïc Barthe, Mathias Paulin

Revue Électronique Francophone d'Informatique Graphique, Volume 1, Numéro 1, pp. 1–10, 2007

Reconstruction sur GPU de Metropolis Instant Radiosity

Anthony Pajot, Loïc Barthe, Mathias Paulin

Université de Toulouse ; IRIT
UPS, 118 route de Narbonne, 31062 Toulouse Cedex 9

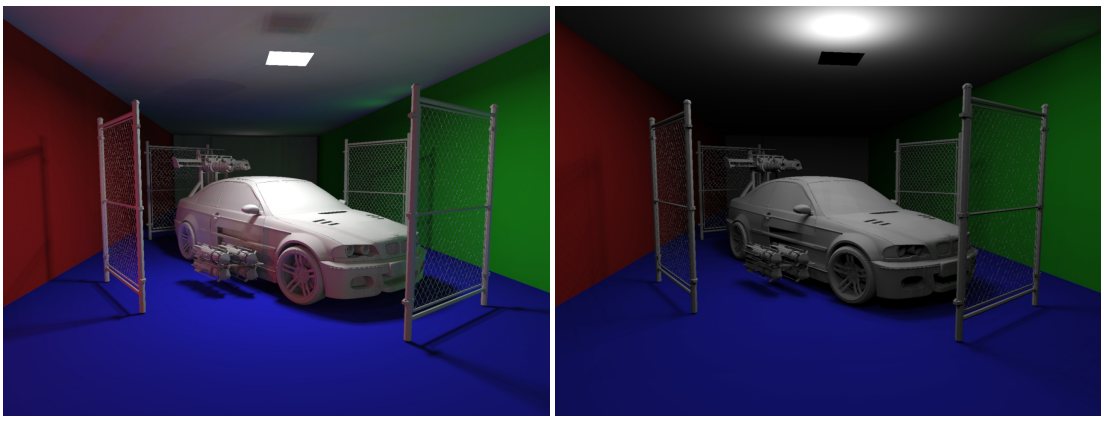


Figure 1: *Résultats obtenus avec un éclairage principalement direct (à gauche) et presque entièrement indirect (à droite). L'éclairage est calculé avec l'algorithme de Metropolis Instant Radiosity, la reconstruction finale étant faite avec notre algorithme de reconstruction sur carte graphique. Images originales en 1600x1200 pixels avec 256 sources ponctuelles virtuelles, scène comportant 1.4 millions de triangles. Temps de rendu total : 1 minute 17 dont 17 secondes de reconstruction pour l'image de gauche, et 1 minute 30 dont 30 secondes de reconstruction pour l'image de droite.*

Résumé

Nous présentons une reconstruction efficace de l'éclairage global sur carte graphique pour les algorithmes utilisant un grand nombre de sources virtuelles ponctuelles. Nous nous basons sur l'algorithme de Radiosité instantanée avec échantillonnage de Metropolis pour disposer de manière robuste les sources dans la scène, et effectuons la reconstruction sur GPU avec CUDA. Cette reconstruction nécessite énormément de calculs répétitifs, à savoir un test de visibilité et une évaluation de BSDF. La nature très parallèle de ces calculs a conduit notre choix de le faire sur carte graphique avec CUDA. Notre algorithme de reconstruction peut atteindre cent quarante millions de calculs "élémentaires" par seconde sur des scènes de taille normale (comportant plusieurs centaines de milliers de triangles).

Mots clé : Éclairage global, méthodes de Monte-Carlo, échantillonnage de Metropolis, sources ponctuelles virtuelles, lancer de rayon sur carte graphique

1. Introduction

Calculer une image d'une scène 3D de manière physiquement réaliste en temps interactif voire temps réel est un des objectifs principaux de la recherche actuelle dans le domaine du rendu photoréaliste. Pour cela, il faut calculer toutes les interactions lumineuses présentes dans la scène. Les algorithmes réalisant cela sont nommés algorithmes d'éclairage global. Parvenir à cet objectif permettrait des applications très variées, allant de la conception par ordinateur aux jeux vidéos en passant par l'architecture.

Dans ce cadre, de nombreux algorithmes ont été présentés, prenant en compte tout ou partie des effets. Dans cet article, nous présentons une implémentation sur GPU de la phase de reconstruction de l'un d'entre eux, l'algorithme dit de "Radiosité instantanée avec échantillonnage de Metropo-

lis" [**?**] (*Metropolis Instant Radiosity* en anglais, et appelé *MIR* dans la suite de ce document). Cette partie est la plus coûteuse de l'algorithme. Le MIR se base sur l'algorithme de *radiosité instantanée* [**?**]. Il fait partie de la famille des algorithmes utilisant des sources ponctuelles virtuelles pour simuler l'éclairage indirect dans des scènes *diffuses* ("source ponctuelle virtuelle" sera abrégé en *VPL* - pour *Virtual Point Light* - par la suite). Les VPLs ne sont valides que dans le cadre de scènes diffuses, car leur émission est supposée lambertienne. Cet algorithme n'est donc pas un algorithme général : les scènes comportant des réflexions ou des réfractions ne sont pas gérées, mais il est très efficace pour les scènes diffuses.

Le problème de l'éclairage global est d'abord présenté, ainsi que les méthodes numériques utilisées et un rapide état de l'art des méthodes existantes. Le MIR tel que présenté dans [?] est détaillé, avec les bases théoriques et les différentes phases de l'algorithme, l'accent étant mis sur la phase de reconstruction, cible de ce document. Plusieurs algorithmes constituant notre contribution sont présentés, les résultats des différents tests effectués étant ensuite présentés et discutés dans la dernière section de cet article.

2. Contexte

2.1. Le problème

La synthèse d'une image revient à calculer pour chaque pixel l'énergie provenant des sources de lumière. Cette énergie peut arriver soit directement (la caméra voit la source de lumière, par exemple si un néon est dans le champ de vision), soit après n rebonds à la surface des objets. Une hypothèse commune est de considérer que le milieu ambiant est vide, *ie.* qu'il n'y a pas de milieu participant. L'équation qui donne l'énergie émise par un point $\mathbf{x}$ situé sur la surface d'un objet dans une direction ω_o est l'équation de rendu [**?**] :

$$L_o(\mathbf{x},\omega_o) = L_e(\mathbf{x},\omega_o) + \int_{\Omega^2} f_s(\mathbf{x}, w_i \leftrightarrow w_o) L_i(\mathbf{x},\omega_i) d\sigma^{\perp}_{\mathbf{x}}(\omega_i) \quad (1)$$

avec

- $L_o(\mathbf{x},\omega_o)$: la radiance sortante depuis $\mathbf{x}$ dans la direction ω_o
- $L_e(\mathbf{x},\omega_o)$: la radiance émise par la surface elle-même depuis $\mathbf{x}$ dans la direction ω_o (cette valeur est non nulle uniquement pour des surfaces représentant des sources lumineuses, comme des néons par exemple)
- $f_s(\mathbf{x}, w_i \leftrightarrow w_o)$: la valeur de la fonction de diffusion bidirectionnelle au point $\mathbf{x}$ pour les directions ω_i et ω_o
- $L_i(\mathbf{x},\omega_i)$: la radiance reçue en $\mathbf{x}$ depuis la direction ω_i
- $d\sigma^{\perp}_{\mathbf{x}}(\omega_i)$: la mesure par rapport à l'angle solide projeté au point $\mathbf{x}$ de la direction ω_i. En notant $\sigma()$ la mesure sur les angles solides, $d\sigma^{\perp}_{\mathbf{x}}(\omega_i) = cos(\theta) d\sigma(\omega_i)$.

Chaque pixel peut être considéré comme un capteur. La réponse d'un capteur est fonction de l'incidence qu'il recoit. La fonction $W_e^{(j)}(\mathbf{x_0},\omega)$ permet de refléter cela : elle donne la fraction de l'énergie recue selon la direction ω quand celle-ci part de $\mathbf{x_0}$, situé sur la lentille de la caméra pour le pixel j. L'*équation de mesure* est alors le calcul de l'énergie recue par un pixel :

$$I_j = \int_{\mathscr{M}_l} \int_{\Omega^2} W_e^{(j)}(\mathbf{x_0},\omega) L_i(\mathbf{x_0},\omega) d\sigma^{\perp}_{\mathbf{x_0}}(\omega) d\mathscr{A}(\mathbf{x_0}) \quad (2)$$

avec $\mathscr{M}_l$ l'ensemble des points de la lentille de la caméra et Ω^2 l'ensemble des directions de la sphère unité. La fonction $W_e^{(j)}(\mathbf{x_0},\omega)$ est souvent nulle pour la plupart des directions. Un exemple de fonction basique est la fonction qui renvoie 1 si la demi-droite $\mathbf{x_0} + \alpha\omega$ passe par le pixel j, 0 sinon. Avec une telle fonction, une valeur de $L_i(\mathbf{x_0},\omega)$ influencera au plus un pixel (ça peut être 0 si la demi-droite $\mathbf{x_0} + \alpha\omega$ ne passe par aucun pixel). Il est à noter qu'en général, cette fonction est la même pour tous les pixels, la seule différence étant que la zone non-nulle est centrée sur le pixel j.

Avec l'hypothèse de milieu ambiant vide, et en notant $tr(\mathbf{x},\omega)$ la fonction de tracé de rayon qui retourne le point vu par $\mathbf{x}$ dans la direction ω, il apparaît que $L_i(\mathbf{x},\omega) = L_o(tr(\mathbf{x},\omega),-\omega)$. L'équation de mesure 2 peut alors se calculer numériquement en estimant $L_o(tr(\mathbf{x_0},\omega),-\omega)$.

2.2. Méthode de Monte-Carlo

Les méthodes déterministes d'intégration numérique demandent des connaissances analytiques sur la fonction à intégrer que l'on ne peut obtenir de manière générale en rendu, il est donc impossible de les utiliser dans ce cadre. La méthode utilisée est alors la méthode de Monte-Carlo.

À la base, cette méthode permet d'estimer l'espérance de toute fonction f définie sur un domaine $\mathscr{D}$. Il suffit de pouvoir calculer sa valeur en tout point, sans autre besoin de propriété analytique. Formellement, l'estimateur de Monte-Carlo est le suivant :

$$F_N(f(X)) = \frac{1}{N} \sum_{i=1}^{N} f(x_i) \quad (3)$$

où les x_i sont des réalisations indépendantes d'une même variable aléatoire X, son domaine de définition devant au moins couvrir celui où f est non nulle pour être sans biais. Il faut donc pouvoir générer des éléments selon la densité de probabilité de X, notée p_X.

En remplaçant la variable aléatoire $f(X)$ par

$f(X)/p_X(X)$, on obtient :

$$E\left[F_N\left(\frac{f(X)}{p_X(X)}\right)\right] = \frac{1}{N}\sum_{i=1}^{N}\int_{\mathscr{D}}\frac{f(x_i)}{p_X(x_i)}p_X(x_i)dx_i = \int_{\mathscr{D}} f(x)dx$$

La méthode de Monte-Carlo permet donc d'estimer de manière non-biaisée des intégrales sur des domaines totalement abstraits, tant qu'il est possible de générer des échantillons suivant une certaine densité de probabilité couvrant tout le domaine sur lequel la fonction f est non-nulle. Par contre, cette méthode a un écart-type en $1/\sqrt{N}$, ce qui signifie que pour diviser par deux l'incertitude sur la valeur réelle d'une estimation, il faut multiplier par 4 le nombre d'échantillons (et donc globalement le temps de calcul).

Une technique, appelée échantillonnage par importance, permet de réduire cette variance sans recourir à l'explosion du nombre d'échantillons. Elle consiste à faire l'estimation en utilisant une variable aléatoire dont la densité est proche de l'intégrande. En effet, l'estimateur $F_N(f(X)/p_X(X))$ est juste la moyenne d'une variable aléatoire Y définie comme étant $f(X)/p_X(X)$. Intuitivement, plus une variable aléatoire est proche de la constance, plus sa moyenne est facile à estimer avec peu d'échantillons. Dans le cas parfait où elle est constante, un seul échantillon suffit pour avoir la réponse exacte. Donc, si $f(X)/p_X(X)$ est constante (*ie.* $p_X(X) \propto f(X)$), la variance de l'estimateur est nulle. Cela dit, il est en pratique impossible d'échantillonner exactement selon une densité proportionnelle à $f(X)$ (il faudrait connaître son intégrale pour cela). Mais il est parfois possible de se rapprocher de cet objectif en utilisant des approximations, ou en ne prenant en compte qu'une partie de l'intégrande (en décomposant $f(x)$, par exemple en un produit).

2.3. État de l'art

Une implémentation naïve de l'évaluation de l'équation 2 par la méthode de Monte-Carlo amènerait au calcul récursif de l'équation de rendu (1), créant ainsi un "arbre" d'intégrales à évaluer pour chaque pixel, chaque niveau de l'arbre correspondant à un rebond en plus de la lumière dans la scène avant d'atteindre la caméra. Ceci est bien sûr très inefficace, et différentes familles de méthodes sont apparues pour estimer efficacement l'équation 2 soit dans toute sa généralité, soit en ne prenant en compte que certains éléments. Parmi ces familles, il y a les méthodes qui calculent l'équation 2 en le tranformant en un problème d'intégration pur (Suivi de chemin [?], suivi bidirectionnel de chemins [**?**] [**?**]), et les méthodes qui précalculent certaines données pour ensuite faire la reconstruction en utilisant ces données (carte de photons [**?**], radiosité [**?**], cache d'irradiance [**?**], transferts de radiance précalculés [**?**]). Tous les algorithmes cités comme exemples de cette dernière catégorie nécessitent des précalculs relativement lourds, qui les rendent peu pratiques dans un cadre dynamique, et ne gèrent pas forcément tous les effets lumineux possibles. Les algorithmes basés sur les VPLs font partie de cette catégorie, mais nécessitent des calculs beaucoup moins lourds.

3. Radiosité instantanée

L'algorithme de base utilisant les VPLs s'appelle la radiosité instantanée. Dans cet algorithme, les VPLs sont générées de la manière suivante : pour chaque source de lumière, un certain nombre N de couples (point, direction) sont créés, chaque point étant sur la source de lumière. Pour chaque couple $(\mathbf{x},\omega)$, une VPL est placée au point $\mathbf{y} = tr(\mathbf{x},\omega)$, et calibrée de manière à représenter l'éclairage indirect de niveau 1 (1 seul rebond). Ensuite, pour un certain nombre $|\rho N|$ de points ($\rho < 1$), de nouvelles directions sont générées, et des VPLs "de niveau 2" sont placées et calibrées. Le processus continue, en générant pour chaque niveau n $|\rho^{n-1}N|$ VPLs. ρ est dépendant des matériaux présents dans la scène, il représente la réflectance moyenne des matériaux (facile à calculer ou approcher car la scène est diffuse).

Une fois les VPLs déposées, la reconstruction est faite en calculant l'éclairage apporté par celles-ci à la caméra par éclairage direct. Ceci revient à approcher l'équation 2 par :

$$I_j \simeq \int_{\mathscr{M}_l}\int_{\Omega^2}\left[W_e^{(j)}(\mathbf{x},\omega)\sum_{s=1}^{n}C_s(tr(\mathbf{x},\omega),-\omega)\right]d\sigma^{\perp}_{\mathbf{x}}(\omega)d\mathscr{A}(\mathbf{x}) \tag{4}$$

où $C_s(tr(\mathbf{x},\omega),-\omega)$ est l'estimation de la radiance recue depuis la VPL s au point $\mathbf{y} = tr(\mathbf{x},\omega)$ et réemise dans la direction ω_o, à savoir :

$$C_s(\mathbf{y},\omega) = \frac{f_s(\mathbf{y},\omega_i^{(s)} \leftrightarrow -\omega)L_i(\mathbf{y},\omega_i^{(s)})}{p_{\sigma^{\perp}_{\mathbf{y}}}(\omega_i^{(s)})} \tag{5}$$

avec $\omega_i^{(s)}$ la direction unitaire allant de $\mathbf{y}$ à la VPL s. $\mathbf{x}$ étant sur la lentille, si la lentille est ponctuelle, $\mathbf{y}$ est le point vu par la caméra dans la direction ω, et $C_s(\mathbf{y},\omega)$ est la radiance arrivant à la caméra selon la direction ω, après être partie de la VPL s et avoir subi une diffusion en $\mathbf{y}$.

La méthode de la radiosité instantanée présente plusieurs défauts majeurs :

- Plus le nombre de rebonds est important, moins le nombre de VPLs associées est grand. Si on a une scène où le nombre minimum de rebonds pour atteindre la caméra est grand, le rendu sera de très mauvaise qualité.
- Le placement des VPLs ne prend pas du tout en compte le point de vue. Ainsi, même si une source de lumière est dans une pièce fermée située très loin de la caméra, des VPLs inutiles pour l'image à rendre seront générées à partir de celle-ci. On peut trouver des travaux

cherchant à minimiser cet effet, tel que [**?**], mais les approches proposées restent perfectibles.
– Même si une VPL est issue d'une source de lumière contribuant par éclairage indirect à l'image, rien ne dit qu'elle influencera l'image. En effet, rien n'indique que cette source sera placée à un endroit où elle éclairera directement des points vus par la caméra.
– Cet algorithme est biaisé, il y aura toujours une erreur systématique même avec un très grand nombre de VPLs.

Tous ces défauts font que les images calculées avec cette méthode exhibent beaucoup de variance, qui se manifeste sous la forme d'ombres franches (issues des VPLs) très visibles (au lieu d'ombres qui semblent douces), de zones beaucoup plus brillantes qu'elles ne devraient l'être, *etc.*.

Une idée pour résoudre les problèmes de variance de la radiosité instantanée est d'essayer de placer des VPLs uniquement à des endroits où elles éclaireront directement des points vus par la caméra, et uniquement en fonction de leur importance pour l'image, *ie.* indépendamment du nombre de rebonds qu'il y a entre la source de lumière et la VPL. C'est ce qui est fait par l'algorithme du MIR, en se basant sur l'échantillonnage de Metropolis.

4. Metropolis Instant Radiosity

4.1. Échantillonage de Metropolis

L'échantillonnage de Metropolis [**?**] permet de génerer des échantillons selon une densité de probabilité π_∞ arbitraire, appelée distribution *stationnaire*. La seule condition sur cette densité de probabilité est qu'elle soit évaluable en tout point de son domaine de définition. Cet algorithme peut même être utilisé avec une distribution stationnaire non normalisée γ_∞, les échantillons étant alors distribués selon la densité de probabilité proportionnelle à γ_∞ :

$$\pi_\infty(x) = \frac{\gamma_\infty}{\int_{\mathscr{D}} \gamma_\infty(x)dx}$$

À partir d'un échantillon x_0 créé en utilisant une méthode d'échantillonnage quelconque sur l'espace de définition de π_∞, l'algorithme crée une chaîne de Markov d'échantillons, où l'échantillon x_i est obtenu par modification (mutation) de l'échantillon x_{i-1}. Il faut bien noter que la densité à partir de laquelle x_0 a été généré n'est pas π_∞, mais une autre densité p_{X_0}. À l'étape $t+1$, un échantillon tentative y est généré à partir de x_t avec une certaine probabilité $T(x_t \to y)$, et est accepté avec une probabilité $a(x_t \to y)$ définie par :

$$a(x_t \to y) = \min\left(1, \frac{T(y \to x_t)\gamma_\infty(y)}{T(x_t \to y)\gamma_\infty(x)}\right) \qquad (6)$$

Cette expression de $a(x_t \to y)$ permet de maximiser la vitesse de convergence pour aller de p_{X_0} vers π_∞.

Si l'échantillon tentative y est accepté, $x_{t+1} = y$, sinon $x_{t+1} = x_t$.

L'algorithme 1 reprend tout ceci.

Algorithme 1 Échantillonnage de Metropolis

$x_0 =$ échantillon_source()
pour i = 0 à N **faire**
  $y =$ muter (x_i)
  $a = a(x_i \to y)$
  si aléatoire() $< a$ **alors**
    $x_{i+1} = y$
  sinon
    $x_{i+1} = x_i$
  fin si
fin pour

En rendu, cette méthode a tout d'abord été utilisée dans le cadre d'algorithmes considérant le problème du rendu comme un problème d'intégration pur sur un espace abstrait Ω, l'espace des chemins. Un chemin $\bar{x}$ est un ensemble de points $\mathbf{x_0}, \mathbf{x_1}, ..., \mathbf{x_k}$ visibles entre eux, $\mathbf{x_0}$ étant sur une source de lumière et $\mathbf{x_k}$ sur la lentille de la caméra. Un chemin pouvant avoir un nombre arbitraire de points, cet espace est de dimension infinie. Il est possible de définir une fonction f qui donne la radiance arrivant à la lentille par le chemin et le problème du calcul d'un pixel est alors l'évaluation d'une seule intégrale :

$$I_j = \int_\Omega W_e^{(j)}(\mathbf{x_k} \to \mathbf{x_{k-1}}) f(\bar{x}) d\mu(x) \qquad (7)$$

Une image se résume donc à l'estimation de plusieurs milliers ou millions d'intégrales fortement liées, et dont les intégrandes sont toutes dominées par la même fonction f. En utilisant l'échantillonnage de Metropolis pour générer des échantillons selon une densité proportionnelle à f, on aboutit à l'algorithme du transport lumineux avec échantillonnage de Metropolis [**?**].

Cet algorithme utilise des mutations de chemins, qui consistent à modifier le chemin de certaines manières (supprimer un segment et le remplacer par un autre, ...), de façon à pouvoir calculer les probabilités conditionnelles $T(\bar{x} \to \bar{y})$. Parmi les différentes mutations envisageables, une mutation particulière permet d'explorer tout l'espace en faisant de grands changements aux chemins, la mutation bidirectionnelle, les autres étant plus locales et destinées à prendre en compte efficacement des effets tels que les caustiques ou les réflexions/refractions proches du spéculaire. Une autre mutation sur un espace plus abstrait permet de tout prendre en compte d'un seul coup [**?**].

Il se trouve qu'en utilisant la même méthode d'échantillonnage, il est possible de placer des VPLs ayant des propriétés très intéressantes.

4.2. Principe du MIR

Le MIR [?] consiste à placer des VPLs uniquement à des endroits où elles éclairent directement des points vus par la caméra, et uniquement en fonction de leur importance pour l'image, *ie.* indépendamment du nombre de rebonds qu'il y a entre la source lumière et la VPL. L'échantillonnage de Metropolis est utilisé à cet effet : les chemins les plus importants sont parcourus en priorité, et si on place une VPL au 3ème point en partant de la caméra sur chaque chemin, elle participera forcément à l'éclairage direct de la partie de la scène vue par la caméra. Ceci est illustré par la figure 2, le point $\mathbf{x_v}$ étant le point où la VPL sera placée. Afin de conserver un temps de rendu faible, peu de VPLs sont générées (entre 100 et 1000), et il est important de prendre en compte tout l'éclairage. Pour cela, il faut utiliser une mutation large telle que la mutation bidirectionnelle, les effets locaux n'étant pas présents dans une scène diffuse. De plus, afin d'améliorer l'exploration de l'espace des chemins, une variante de l'algorithme 1 est utilisée, dite à essais multiples [**?**] : elle consiste à générer plusieurs candidats à chaque étape et choisir de manière probabiliste et non biaisée le meilleur candidat.

Contrairement à la radiosité instantanée qui propage l'énergie des sources de lumière *via* des VPLs, ici elles vont véritablement remplacer les sources de lumière originales, ce qui veut dire que la reconstruction ne prendra plus du tout en compte les sources de lumière originales pour le calcul de l'éclairage des surfaces non émettrices. De manière plus générale, les VPLs représenteront des chemins (les chemins liant la position de la VPL à la source de lumière), et non plus des points émettant de la lumière. Ainsi, plutôt que de noter une VPL par sa position $\mathbf{x_{v_i}}$, deux notions sont à distinguer :

- la VPL *géométrique*, qui n'est que le point et notée $\mathbf{x_v}$,
- la VPL *de chemin*, qui prend en compte le chemin jusqu'à la source de lumière. On la dénotera par sa position *et* le chemin $\overline{x_s}$ jusqu'à la source de lumière, ce qui donnera donc la notation $\{\mathbf{x_v}, \overline{x_s}\}$

Ceci donne un découpage du chemin en trois parties, illustré par la figure 2 :

1. les deux points partant de la caméra, que l'on notera $\mathbf{x_0}$ et $\mathbf{x_1}$ (ce qui est *l'inverse* par rapport aux indices normalement utilisés pour les chemins jusqu'à présent, mais qui est plus pratique pour la suite),
2. le point où se situe la VPL, $\mathbf{x_v}$,
3. le chemin jusqu'à la source de lumière $\{\mathbf{x_v}, \overline{x_s}\}$

La méthode d'échantillonnage présentée dans l'algorithme 2 (par mutation de chemins) permet d'avoir plusieurs VPLs de chemin pour une même VPL géométrique : si la mutation ne change pas le point $\mathbf{x_v}$ (elle change soit $(\mathbf{x_0}, \mathbf{x_1})$ soit $\overline{x_s}$), une deuxième VPL sera posée exactement au même endroit. Cette propriété d'*agglomération* de VPLs de chemin est fortement exploitée dans l'algorithme, et motive la distinction VPL géométrique/VPL de chemin. Elle influence aussi la conception des structures de données, une VPL étant représentée par la position de la VPL géométrique $\mathbf{x_v}$, la normale au point $\mathbf{x_v}$ pour gérer le fait que les VPLs sont hémisphériques, et un ensemble des VPLs de chemin dont le dernier point est $\mathbf{x_v}$ (*ie.* de la forme $\{\mathbf{x_v}, \overline{x_s}\}$).

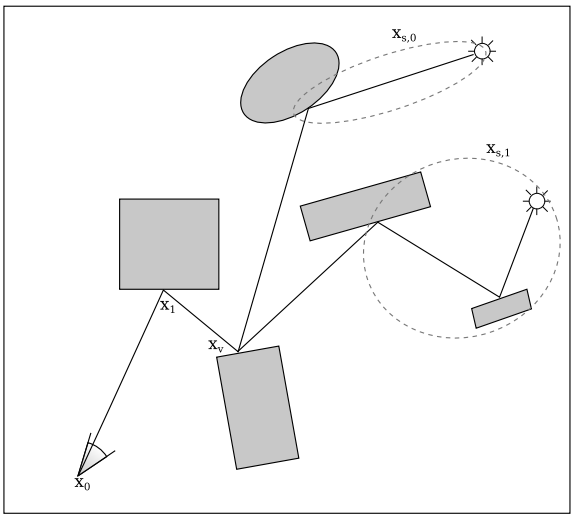


Figure 2: *La séparation d'un chemin en trois parties. Ici, il y a une VPL géométrique, et deux VPLs virtuelles de chemin. On voit bien l'aggrégation des VPLs de chemin partageant le même point* $\mathbf{x_v}$.

Algorithme 2 Construction des sources virtuelles par échantillonnage de Metropolis

$vpls = \{\}$
$nb\_vpls = 0$
$\overline{x}$ = chemin_source()
$vpl = \mathbf{x_v}$
ajouter_source_de_chemin ($vpl, \{\mathbf{x_v}, \overline{x_s}\}$)
ajouter ($vpls, vpl$)
tant que $nb\_vpls < N$ **faire**
  $\overline{x}$ = échantillonner($\overline{x}$) *// renvoie une mutation de* $\overline{x}$ *si acceptée,* $\overline{x}$ *sinon*
  si $\mathbf{x_v}$ a changé **alors**
    $vpl = \mathbf{x_v}$ *// nouvelle VPL* géométrique
    ajouter_source_de_chemin ($vpl, \{\mathbf{x_v}, \overline{x_s}\}$)
    ajouter ($vpls, vpl$)
    $nb\_vpls = nb\_vpls + 1$
  sinon
    ajouter_source_de_chemin ($vpl, \{\mathbf{x_v}, \overline{x_s}\}$) *// nouvelle VPL* de chemin
  fin si
fin tant que

On voit à la ligne 8 de l'algorithme 2 qu'une VPL peut être dupliquée, si la mutation est rejetée. Ceci est la manière qu'a l'échantillonnage de Metropolis de maintenir ou atteindre la distribution stationnaire, en renforcant les chemins les plus intéressants, au prix d'une corrélation. C'est ici que l'on voit l'importance d'une bonne mutation : plus la mutation génèrera des chemins ayant une forte probabilité d'acceptation, moins la corrélation sera présente.

L'échantillonnage de Metropolis est un processus fortement séquentiel, une parallélisation suffisante pour une implémentation GPU nécessitant de tester beaucoup de chemins candidats à chaque étape, et/ou d'avoir beaucoup d'échantillonneurs utilisant des chemins sources différents (chemins sources qu'il aura fallu générer avant). Malgré cela, cette phase de MIR prend un temps négligeable comparé au temps nécessaire à la reconstruction de l'image.

4.3. Propriétés des VPLs

Premièrement, la probabilité d'avoir généré une VPL de chemin (donc la probabilité d'avoir généré les points constituant $\{\mathbf{x_v}, \overline{x_s}\}$), est proportionnelle au flux qu'elle apporte à la caméra. La probabilité d'une VPL de chemin $\{\mathbf{x_v}, \overline{x_s}\}$ est une probabilité *marginale* de celle d'avoir générée un chemin, car c'est la probabilité d'avoir généré *tous les chemins* pouvant s'écrire $\overline{x} = (\mathbf{x_0}, \mathbf{x_1}, \mathbf{x_v}, \overline{x_s})$ avec $\mathbf{x_0} \in \mathscr{M}_l$ et $\mathbf{x_1} \in \mathscr{M}$, $\mathscr{M}$ étant l'ensemble des surfaces de la scène. Plus formellement, on peut voir $\overline{x} = (\mathbf{x_0}, \mathbf{x_1}, \mathbf{x_v}, \overline{x_s})$ comme une variable aléatoire à $k+1$ dimensions (k étant la longueur du chemin), et $(\mathbf{x_v}, \overline{x_s})$ comme une variable aléatoire marginale à $k-1$ dimension, sa probabilité étant alors donnée par :

$$p(\{\mathbf{x_v}, \overline{x_s}\}) = \int_{\mathscr{M}_l} \int_{\mathscr{M}} p((\mathbf{x_0}, \mathbf{x_1}, \mathbf{x_v}, \overline{x_s})) d\mathscr{A}(\mathbf{x_1}) d\mathscr{A}(\mathbf{x_0}) \tag{8}$$

L'échantillonnage de Metropolis génère des chemins ayant une densité de probabilité proportionnelle à $f(\overline{x})$. En notant $c = 1/\int_\Omega f(\overline{x}) d\mu(\overline{x})$ la constante de normalisation, on a $p(\overline{x}) = c.f(\overline{x})$, et donc :

$$\begin{aligned} p(\{\mathbf{x_v}, \overline{x_s}\}) &= \int_{\mathscr{M}_l} \int_{\mathscr{M}} c.f((\mathbf{x_0}, \mathbf{x_1}, \mathbf{x_v}, \overline{x_s})) d\mathscr{A}(\mathbf{x_1}) d\mathscr{A}(\mathbf{x_0}) \\ &= c. \underbrace{\int_{\mathscr{M}_l} \int_{\mathscr{M}} f((\mathbf{x_0}, \mathbf{x_1}, \mathbf{x_v}, \overline{x_s})) d\mathscr{A}(\mathbf{x_1}) d\mathscr{A}(\mathbf{x_0})}_{\text{flux fourni par la VPL} \{\mathbf{x_v}, \overline{x_s}\} \text{ à l'image}} \\ &= c.P_{sv}(\{\mathbf{x_v}, \overline{x_s}\}) \end{aligned} \tag{9}$$

en notant $P_{sv}(\{\mathbf{x_v}, \overline{x_s}\})$ le flux fourni à l'image par la VPL de chemin $\{\mathbf{x_v}, \overline{x_s}\}$.

De plus, le flux (ici, valeur scalaire représentant par exemple la luminance) arrivant sur l'image *via* une estimation de Monte-Carlo où les échantillons sont donnés par les n VPLs de chemin s'exprime par :

$$\begin{aligned} P_c &= \int_\Omega f(\overline{x}) d\mu(\overline{x}) \\ &\simeq \frac{1}{n} \sum_{i=1}^{n} \frac{\int_{\mathscr{M}_l} \int_{\mathscr{M}} f((\mathbf{x_0}, \mathbf{x_1}, \mathbf{x_{v_i}}, \overline{x_{s_i}})) d\mathscr{A}(\mathbf{x_1}) d\mathscr{A}(\mathbf{x_0})}{p(\{\mathbf{x_{v_i}}, \overline{x_{s_i}}\})} \\ &\simeq \frac{1}{n} \sum_{i=1}^{n} \underbrace{\frac{P_{sv}(\{\mathbf{x_{v_i}}, \overline{x_{s_i}}\})}{p(\{\mathbf{x_{v_i}}, \overline{x_{s_i}}\})}}_{\text{contribution au flux total de la VPL } i} \\ &\simeq \frac{1}{n} \sum_{i=1}^{n} \frac{1}{c} \\ &= \frac{1}{n} \sum_{i=1}^{n} P_c = \sum_{i=1}^{n} \frac{P_c}{n} \end{aligned} \tag{10}$$

ce qui veut dire que toutes les VPLs de chemin amènent le même flux à l'image finale (égal à P_c/n). Autrement dit, il n'y a pas de VPL de chemin "dominante", d'où moins de variance. De plus, le flux reçu par l'image depuis la source ne dépend pas du chemin précis. Le flux reçu par la caméra à partir d'une VPL géométrique ayant k VPLs de chemin est donc égal à $k.P_c/n$. Lorsque le spectre d'émission de chaque VPL est déterminé, il est normalisé de manière à assurer cette propriété [?].

5. Reconstruction de l'image

La reconstruction de l'image finale consiste à rassembler pour un ensemble de points d'échantillonnage du plan image (que l'on nommera *luxel* - pour "élément de luminance spectrale") les informations de luminance qui serviront ensuite à calculer la valeur de chaque pixel par filtrage *via* un noyau de reconstruction (Sync dans notre cas), évaluant ainsi la valeur de chaque pixel de manière non biaisée. Ces points sont répartis uniformément sur l'écran, il peut y en avoir plusieurs par pixels si du suréchantillonnage est souhaité.

Dans notre implémentation, la reconstruction de l'image se fait en 2 phases, correspondant à une approche d'un rendu différé : la première phase rassemble des informations sur les points 3D vus par les luxels (position, normale, BSDF, *etc.*) dans un buffer de géométrie ; la seconde calculant la radiance de chaque luxel à partir des VPLs et de ces informations.

Cette reconstruction est la partie la plus coûteuse du MIR, car elle nécessite énormément de tests de visibilités (autant que de VPLs pour chaque luxel). L'algorithme 3 décrit les principales étapes de cette phase, avec les précisions suivantes :

- <*chargement données luxel*> charge les données du buffer de géométrie pour le luxel.
- <*ajouter l_e*> ajoute le terme source l_e à la radiance du luxel courant.
- <*calcul ombrage*> calcule le terme d'ombrage $f_s(\mathbf{x}, \omega_o \leftrightarrow \omega_i)/p_\sigma(\omega_i)$. Renvoie 0 si $p_\sigma(\omega_i) = 0$, ce qui peut arriver dans le cas de matériaux partiellement spéculaires.

Algorithme 3 Algorithme de reconstruction

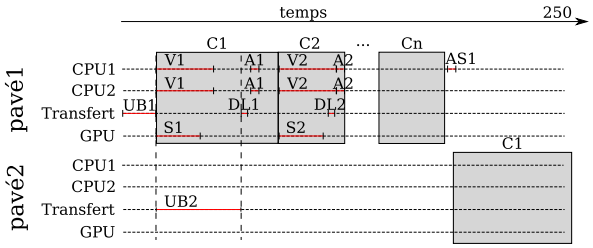
```
pour tout luxel l faire
  si intersection non vide alors
    <chargement données luxel>
    <ajouter l_e>
    pour tout vpl v faire
      si v est potentiellement visible (culling) alors
        <calcul ombrage>
        si ombrage non nul alors
          si v est visible depuis le point vu par l alors
            ajouter la contribution de v à l
          fin si
        fin si
      fin si
    fin pour
  fin si
fin pour
```

Afin d'évaluer l'apport d'une parallélisation GPU/CPU, quatre adaptations de l'algorithme 3 ont été implantées : la première, servant de validation, reprend l'algorithme 3 entièrement sur CPU dans une version multi-threadée. Les 3 autres adaptations utilisent divers niveaux de parallélisme CPU/GPU. Ces algorithmes sont décrits dans la suite de cet article. L'ombrage est évalué en utilisant le modèle de Cook-Torrance [**?**], et aucune technique d'échantillonage entrelacé telle que [**?**] n'est utilisée, celles-ci apportant du biais.

5.1. Premier algorithme : évaluation des BSDF sur GPU

Le premier algorithme, appelé "OS" (pour "Ombrage Seulement") dans la suite, effectue les tests de visibilité et l'accumulation des résultats sur CPU. L'évaluation des BSDF est réalisée en parallèle pour plusieurs luxels sur le GPU. L'évaluation des BSDF pour une paire de direction (ω_o, ω_i) consiste juste à charger les données de la BSDF pour le luxel courant, évaluer le modèle de Cook-Torrance et stocker le résultat. Pour limiter la quantité de mémoire utilisée sur le GPU, l'image est découpée en pavés rectangulaires (par exemple des pavés de 300×200 pixels), chaque pavé étant calculé l'un après l'autre. Pour calculer un pavé, il faut donc :

1. transférer sur GPU les paramètres des BSDF (en même temps que le pavé précédent se calcule),
2. calculer sur CPU les résultats des tests de visibilité entre chaque luxel et chaque VPL,
3. en même temps, évaluer sur GPU les BSDF pour chaque luxel et pour chaque VPL,
4. transférer ces valeurs vers la mémoire centrale pour que le CPU puisse les combiner avec la visibilité,
5. accumuler la radiance obtenue pour obtenir la valeur finale du luxel.



UBi = upload BSDF du pavé i, transfert CPU -> GPU
Vj = calcul de la visibilité pour la source j
Sj = calcul de la BSDF pour la source j
Aj = accumulation pour la source j
DLj = téléchargement des résultats pour la source j, transfert GPU -> CPU
Cj = calcul pour la source j
ASi = ajout à l'image des valeurs calculées pour le pavé i

Figure 3: *Exécution de l'algorithme OS sur une machine ayant 2 CPU et un GPU supportant les transferts mémoire en parallèle du calcul (type GTX280), illustré pour les 3 premiers pavés. Chaque C_i constitue une grille de threads. Les barres ne sont pas vraiment à l'échelle, mais le parallélisme indiqué est celui mis en place. On voit que DL1 doit attendre qu'UB2 soit fini, ce qui implique un ralentissement du traitement de la première VPL. Ceci n'aura plus lieu avec la prochaine génération de carte graphique, des transferts dans les 2 sens pouvant être faits en même temps.*

Les ressources limitées du GPU et les besoins importants en mémoire de cet algorithme nécessitent des transferts CPU/GPU fréquents des valeurs de BSDF calculées, : les VPLs sont ainsi traitées les unes après les autres. Le déroulement temporel de cet algorithme est présenté sur la figure 3.

Les performances des calculs parallèles sur GPU sont étroitement liées aux nombres d'accès à la mémoire globale du GPU. Afin de minimiser ce nombre d'accès, les données doivent être organisées de manière cohérentes par rapport aux *threads* d'exécution. Les accès mémoire sont alors *coalescents* : les *threads* exécutés sur des unités voisines doivent avoir des données voisines. Pour cela, les pavés constituant l'image sont découpés en blocs rectangulaires (qui donneront les blocs de *threads* de CUDA), et les données des BSDF sont organisées en "structure de tableaux", indexés en 2D par des coordonnées (x, y) du luxel dans le pavé. De plus (figure 4), chaque tableau est organisé en interne en utilisant une structure de blocs, qui ont la taille des blocs de *threads* CUDA. Ceci assure donc l'agrégation des accès en mémoire sur le GPU. De manière générale, tout tableau associant une valeur à un luxel devant être utilisé sur le GPU est organisé de cette manière.

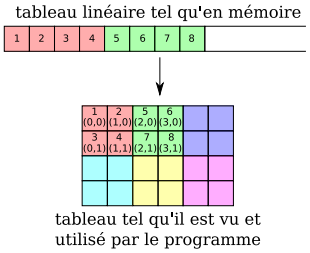
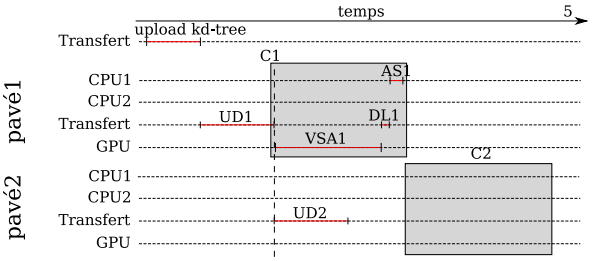


Figure 4: *Adressage utilisé pour avoir des accès coalescents*



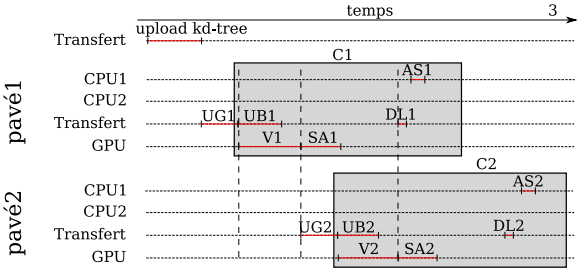
UD*i* = upload des données du pavé *i*, transfert CPU -> GPU
VSA*i* = visibilité, BSDF et accumulation pour le pavé *i*
AS*i* = ajouts des valeurs calculées pour le pavé *i* à l'image
DL*i* = téléchargement des résultats pour le pavé *i*,
transfert GPU -> CPU
C*i* = calcul pour le pavé *i*

Figure 5: *Exécution de l'algorithme RC sur la même configuration que celle de la figure 3.*

5.2. Visibilité et BSDF sur GPU

L'approche parallèle de l'algorithme OS entraîne un fort déséquilibre de charge entre CPU et GPU, le calcul de la visibilité sur CPU étant beaucoup plus coûteux que l'évaluation des BSDF sur GPU. Le développement récent d'un lancer de rayons performant, basé sur un *kd-tree*, implanté sur le GPU en CUDA et rendu disponible librement par son auteur, Benjamin Segovia, a permis d'envisager une intégration efficace du calcul des visibilités sur le GPU [**?**]. L'algorithme RC, dont le déroulement temporel est présenté sur la figure 5, tire parti de cette intégration pour effectuer tout l'algorithme de reconstruction sur le GPU, permettant ainsi de supprimer les transferts de résultats intermédiaires entre le GPU et le CPU, l'organisation en pavé restant la même afin de contrôler l'utilisation des ressources mémoire.

Il est possible d'exploiter plus efficacement le parallélisme entre les transferts de données du CPU vers le GPU, et les calculs sur ces données, en découpant le traitement en 2 étapes successives travaillant sur des données indépendantes. Le calcul des valeurs des luxels se fait alors en séparant le calcul des visibilités d'une part, et l'évaluation des BSDF d'autre part. Le calcul des visibilités nécessite des données formant un sous-ensemble des données nécessaires à l'évaluation des BSDF. Il est donc possible de masquer



UG*i* = upload des données géométriques du pavé *i*,
transfert CPU -> GPU
UB*i* = upload des données propres à la BSDF du pavé *i*,
transfert CPU -> GPU
V*i* = calcul de la visibilité pour le pavé *i*
SA*i* = calcul de la BSDF et accumulation pour le pavé *i*
AS*i* = ajouts des valeurs calculées pour le pavé *i* à l'image
DL*i* = téléchargement des résultats pour le pavé *i*,
transfert GPU -> CPU
C*i* = calcul du pavé *i*

Figure 6: *Exécution de l'algorithme RCVP sur la même configuration que celle de la figure 3.*

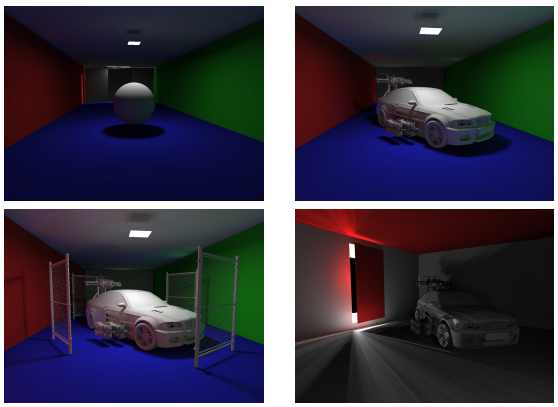


Figure 7: *Scènes de test utilisées : la première, "sphère", contient 2000 polygones, la deuxième, "voiture", 480.000, la troisième, "voiture+grillage", 1.4 millions de polygones. La dernière "garage", a la même complexité que "voiture", mais exhibe beaucoup plus de cohérence pour les tests de visibilité.*

le chargement des données de BSDF par le calcul des visibilités. Une conséquence extrêmement intéressante de ce découpage est de rendre partiellement gratuite la partie la plus coûteuse de l'algorithme. Le déroulement temporel de cet algorithme, nommé "RCVP" (pour "reconstruction complète avec visibilité en parallèle") est présenté sur la figure 6.

6. Résultats

Les mesures ont été effectuées sur 3 scènes différentes (les 3 premières de la figure 7), nommées "sphère", "voiture"

CPU	OS	RC	RCVP
3m20s	4m07s	5s	3s

Figure 8: *temps de reconstruction sur la scène "sphère" par les 4 algorithmes de reconstruction.*

et "voiture+grillage". Elles sont de complexité géométrique croissante mais partagent toutes un éclairage similaire : une source de lumière principale au dessus de l'objet d'intêret (orientée soit vers le plafond soit vers le sol), et une source de lumière située derrière un plan pour exhiber un éclairage indirect provenant par un petit passage, comme le cas classique de la porte entr'ouverte donnant sur une autre pièce. Si la source de lumière principale est orientée vers le bas, l'éclairage est essentiellement direct, et si elle est orienté vers le haut, il est essentiellement indirect.

La résolution des images finales est 1600x1200 avec un luxel par pixel. Sauf mention explicite, les mesures sont réalisées avec 256 VPLs et les pavés font 256×256 pixels. La machine de test est un core2duo à 3GHz équipé d'une carte NVidia GTX280 programmée en CUDA 2. Nous nous intéressons ici uniquement au temps pris par la reconstruction, les autres parties du MIR étant communes.

Le temps de reconstruction de chaque algorithme a d'abord été testé sur la scène la plus simple, "sphère". Les temps de calcul sont indiqués dans la figure 8. On remarque que l'algorithme OS est plus lent que la version de base exécutée uniquement sur CPU, ce qui s'explique par le volume des transferts mémoire à effectuer et le coût des calculs de visibilité effectués sur CPU par rapport aux calculs d'ombrage effectués sur GPU. Le coût rhédibitoire des transferts des résultats intermédiaires par VPL se retrouverait aussi de manière symétrique dans un algorithme où les BSDFs seraient évaluées sur CPU pendant que le GPU calculerait la visibilité, nous n'avons donc pas implémenté cette variante.

Pour voir l'impact du recouvrement des transferts mémoire par du calcul (disponibles sur les cartes NVidia à base de GTX2XX ou supérieures), les 3 scènes ont été rendues avec les algorithmes RC et RCVP, avec les deux configurations d'éclairages. La figure 9 présente les résultats. L'impact du recouvrement apparaît clairement (des gains de l'ordre de 25%), indépendamment de la configuration de l'éclairage et de la complexité de la scène sauf pour la plus simple où ce sont les transferts mémoires qui dominent. La configuration directe fait apparaître de fortes différences de performances dans le cas de la troisième scène. En effet, les VPLs étant le plus probablement placées sur les murs – éclairés directement, donc apportant plus d'énergie à l'image –, un grand nombre des rayons utilisés pour le calcul de visibilité traversent majoritairement les régions contenant les grillages. Ceux-ci étant constitués de géométrie haute fréquence, la cohérence inter-rayons s'en trouve considérablement réduite.

		sphère	voiture	voiture+grillage
RC	direct	5s	17s	44s
	indirect	5s	17s	23s
RCVP	direct	3s	12s	31s
	indirect	5s	12s	17s

Figure 9: *temps de reconstruction sur les 3 scènes pour les algorithmes RC et RCVP, en configuration d'éclairage direct et indirect.*

	simple	voiture/direct	garage
contributions (M/s)	196	45	140

Figure 10: *Performances de pic sur 3 scènes, dont les 2 dernières de même poids mais avec des rayons de visibilité aux cohérences très différentes. Les chiffres donnent le nombre de contributions calculées, en millions par seconde.*

Afin de tester la puissance brute du moteur de lancer de rayon sur GPU utilisé, le nombre de VPLs a été augmenté. En effet, les transferts mémoire n'augmentent pas de manière sensible avec le nombre de sources virtuelles, tandis que le nombre de calculs de contributions augmente linéairement (une contribution étant un test de visibilité et une évaluation de BSDF). Une progression du temps de calcul linéaire par rapport au nombre de VPLs indique une utilisation maximale des ressources de calcul, et donc un calcul fiable de la performance en pic, où les transferts mémoires sont presque entièrement cachés par le calcul (seuls les premiers et derniers transferts ne sont pas cachés). Ce comportement est atteint pour plus de 512 VPLs sur les scènes de tests. Ceci a aussi permis de juger de l'impact de la cohérence de la traversée de la structure d'accélération, en utilisant la scène "garage", de même poids en géométrie que la scène "voiture", mais où la majorité des rayons suivent des parcours similaires. En effet, une proportion non négligeable de sources sont dans une partie de la pièce attenante à la pièce principale, cette zone contribuant énormément à la puissance recue à l'image, et la porte bloque énormément de rayons. Les résultats sont présentés dans la figure 10.

Le découpage de l'image en pavé induit un compromis entre la taille des transferts non recouverts et le nombre de calculs par pavé. Des pavés trop petits impliquent trop peu de calculs pour cacher efficacement la latence des accès des noyaux à la mémoire globale. Des pavés trop grands impliquent des chargements non recouverts très coûteux du premier pavé et des résultats du dernier. Le coût des transferts des pavés intermédiaires ne rentre pas en compte, le nombre de calculs augmentant linéairement avec la taille des pavés. La figure 11 indique les temps de reconstruction pour la scène "voiture" en utilisant des pavés de tailles croissantes, pour 256 VPLs.

100×100	100×256	256×256	512×256	512×512
15s	14s	12s	15s	15s

Figure 11: *Influence de la taille des pavés sur les temps de reconstruction de la scène "voiture" en configuration d'éclairage direct.*

7. Conclusion

Cet article présente différentes adaptations d'un algorithme générique réalisant la phase de reconstruction des algorithmes utilisant des VPLs pour représenter l'éclairage global. Ces adaptations tirent parti de la puissance des cartes graphiques actuelles à différents niveaux. L'algorithme le plus rapide (RCVP) utilise le recouvrement des chargements du CPU vers le GPU par du calcul pour rendre gratuite la partie la plus coûteuse de la reconstruction, constituée de tests de visibilité. Nous montrons qu'il est possible de calculer plus de 140 millions de contributions par secondes sur des scènes de plusieurs centaines de milliers de triangles exhibant de la cohérence inter-rayons sur une GTX280, et 40 millions pour des scènes moins favorables.

Le MIR permet un placement robuste des VPLs dans les scènes diffuses pour des configurations d'éclairage très différentes. Cependant ce placement n'est pas optimal lorsque la scène exhibe un éclairage indirect haute fréquence, comme dans la scène "garage" ou la scène "voiture+grillage". Dans le premier cas, l'éclairage du sol se trouvant face à l'ouverture de la porte devrait varier continûment, mais il exhibe de multiples bordures d'ombres franches. Dans le second, l'ombre indirecte des grillages apparaît franche alors qu'elle devrait être douce. De plus, cet algorithme ne permet pas de prendre en compte les scènes fortement spéculaires. Des recherches dans ce sens permettraient d'obtenir une solution extrêmement robuste pour le calcul rapide de l'éclairage global. Ainsi, le premier problème peut être traité en échantillonnant les VPLs non seulement en fonction de leur puissance, mais aussi en fonction du contenu fréquentiel de la scène : des objets comportant de hautes fréquences devraient être éclairés par plus de VPLs que des objets de basse fréquence. Une solution au second problème est de mélanger le MIR avec un autre algorithme extrêmement efficace pour les scènes spéculaires, par exemple une version modifiée de l'algorithme du transport lumineux avec échantillonnage de Metropolis.