

Carte de photons progressive dans des scènes contenant des milieux participatifs

Charly Collin

▶ **To cite this version:**

Charly Collin. Carte de photons progressive dans des scènes contenant des milieux participatifs. Synthèse d'image et réalité virtuelle [cs.GR]. 2011. dumas-00636156

HAL Id: dumas-00636156

https://dumas.ccsd.cnrs.fr/dumas-00636156

Submitted on 26 Oct 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.





Carte de photons progressive dans des scènes contenant des milieux participatifs

Rapport de stage
Master 2 Recherche en Informatique, 2010-2011

Auteur : Charly COLLIN (charly.collin@etudiant.univ-rennes1.fr)

Encadrants : Kadi BOUATOUCH (kadi@irisa.fr)
Rémi COZOT (cozot@irisa.fr)

Équipe : Bunraku

Résumé

Le rendu de milieux participatifs, tels la fumée ou le feu, reste aujourd'hui très gourmand en mémoire et en temps de calcul. Cela pose un problème pour le rendu de scènes complexes avec les méthodes existantes, notamment celles utilisant les techniques de cartes de photon. Le but de ce document est de proposer une nouvelle méthode de rendu progressif. Celle-ci utilise des cartes de photons et est basée sur des faisceaux traversant les milieux participatifs. Dans un premier temps, cette méthode permet seulement de limiter la mémoire nécessaire au rendu. Dans un second temps cependant, nous présenterons une hiérarchie de ces faisceaux permettant d'accélérer sensiblement les calculs.

Table des matières

1 Introduction **4**

2 État de l'art **6**
2.1 Milieux participatifs 6
2.2 Équation de transfert radiatif et résolution 8
2.2.1 Méthodes à base de caches de luminance 9
2.2.2 Carte de photons volumiques 12
2.3 Carte de photons progressive 16

3 Présentation du stage **18**
3.1 Choix de départ 18

4 Implémentation de la carte de photons volumiques **20**
4.1 Carte de photons de PBRT 20
4.2 Extension au volume 21
4.3 Estimation de luminance de faisceaux 24
4.4 Résultats 24

5 Carte de photons volumiques progressive **26**
5.1 Implémentation 26
5.2 Résultats 28

6 Structure accélératrice **32**
6.1 Hiérarchie de faisceaux 32
6.2 faisceaux englobants 34

7 Conclusion **36**

Bibliographie **37**

1 Introduction

La synthèse d'image en informatique est maintenant, et depuis quelques années, monnaie courante, que ce soit dans les films, jeux vidéos, mais aussi pour des simulations, par exemple en architecture. Si pour les premiers cas, l'objectif est simplement d'avoir «l'air vrai», dans le cadre des simulations, il est nécessaire d'avoir un rendu réaliste en accord avec les lois de la physique. C'est pourquoi toute une partie de la synthèse d'image consiste à reprendre le travail de physiciens et de l'adapter à l'outil informatique, par exemple en discrétisant des intégrales ou en approchant des formules. Cependant, au delà de ces adaptations, il est d'autres contraintes qui sont le temps de calcul et la capacité mémoire d'une machine. Même si les processeurs sont de plus en plus rapides et la mémoire de moins en moins chère, les calculs demandés sont aussi de plus en plus complexes. Il est donc nécessaire de les accélérer.

Comprendre le cheminement de la lumière depuis une lampe jusqu'à notre œil et discrétiser une scène sous forme de triangles n'est pas suffisant pour avoir un rendu réaliste en temps raisonnable. Il faut approcher ce cheminement en trouvant le meilleur compromis entre ressources nécessaires et qualité de rendu. De plus certains types de milieu sont encore aujourd'hui trop gourmands en mémoire et temps de calcul.

Ainsi, si l'on est capable de synthétiser un immeuble ou une pièce de maison de manière réaliste, nous avons en revanche beaucoup plus de mal à synthétiser une image avec un feu de cheminée ou un filet d'eau en un temps raisonnable. En effet, il existe aujourd'hui plusieurs méthodes s'intéressant à ces milieux, appelés milieux participatifs, mais toutes restent relativement coûteuses. Il y a encore tout un travail à faire à ce niveau et c'est la problématique de ce document.

Dans celui-ci nous allons tout d'abord dresser un état de l'art des différentes méthodes de rendu volumique. Le chapitre suivant portera sur les objectifs du stage, ainsi que l'environnement de travail de ce dernier. Nous présenterons ensuite l'implémentation d'une première méthode de rendu volumique, le *Volume Photon Mapping*, puis une première amélioration, le *Progressive Volume Photon Mapping*. Nous étudierons enfin une seconde amélioration, encore théorique pour le moment.

Enfin nous verrons les premiers résultats et pourrons conclure sur le travail déjà effectué.

2 État de l'art

Cette première partie a pour objectif de faire un état de l'art sur les travaux existants quant au rendu de milieux participatifs. Pour cela, nous allons d'abord définir un milieu participatif et nous intéresser aux lois physiques le régissant. Puis nous nous intéresserons aux différentes méthodes de rendu volumique existantes basées sur deux techniques similaires : le cache de luminance et la carte de photons. Enfin, nous étudierons une amélioration apportée à une méthode de rendu non volumique en nous demandant s'il ne serait pas possible de l'adapter justement à un rendu volumique.

2.1 Milieux participatifs

Un milieu participatif peut être défini comme un milieu dans lequel la lumière se propage et interagit avec les particules. Par exemple, l'eau, le feu ou la fumée sont des milieux participatifs. Les principales interactions possibles sont l'émission de la lumière, l'absorption de celle-ci et la diffusion (Figure 1).

- Dans le cas de l'**émission**, le milieu se comporte comme source lumineuse dans la scène et ajoute de l'énergie lumineuse à la scène.

- Dans le cas de l'**absorption**, le milieu va au contraire faire diminuer l'énergie du flux lumineux.

- Enfin, dans le cas de la **diffusion**, le milieu va pouvoir influer sur la direction de propagation de la lumière, tout en ajoutant de l'énergie au flux lumineux (diffusion incidente) ou en retirant de l'énergie (diffusion sortante).

Un des avantages des milieux participatifs est que, quel que soit le milieu considéré, ces interactions sont toujours modélisées par la fonction de phase et la équation de transfert radiatif (ou ETR).

La fonction de phase permet de décrire la distribution selon laquelle la lumière est diffusée dans le milieu. Ainsi, pour un point x et deux directions $\vec{\omega}$ et $\vec{\omega}'$, la fonction de phase $p(x, \vec{\omega}, \vec{\omega}')$ donne la probabilité qu'un photon incident arrivant en

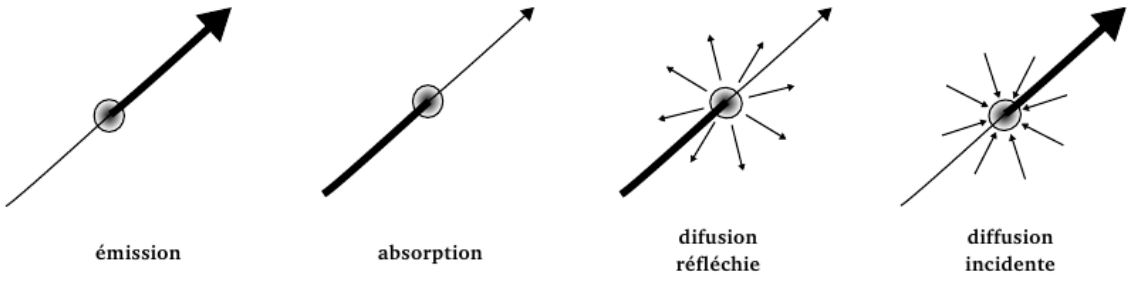


FIGURE 1 – Principales interactions des milieux participatifs avec la lumière

x selon la direction $\vec{\omega}$ soit diffusé selon la direction $\vec{\omega}'$. Cependant, il est courant que seul l'angle θ entre le deux directions intervienne dans le calcul de cette fonction de phase. On s'intéresse alors à $p(\theta)$.

Plusieurs fonctions de phase sont couramment utilisées, notamment la fonction de Henyey-Greenstein :

$$p(\theta) = \frac{1 - g^2}{4\pi(1 + g^2 - 2g\cos(\theta))^{1,5}}$$

où $g \in]-1, 1[$.

Il est possible d'utiliser une combinaison linéaire de différentes fonctions de Henyey-Greenstein afin d'obtenir des fonctions de phase complexes.
Mais souvent, il n'est pas nécessaire d'être aussi précis et puisque les formes obtenues avec la fonction précédente sont proches d'ellipsoïdes, il est possible d'utiliser une fonction de phase simplifiée, écrite par Schlick :

$$p(\theta) = \frac{1 - k^2}{4\pi(1 + k\cos(\theta))^2}$$

où $k \in]-1, 1[$.

Cette expression permet par ailleurs de choisir facilement l'angle θ via la formule suivante :

$$\cos(\theta) = \frac{2\xi + k - 1}{2k\xi - k + 1}$$

où ξ est une variable aléatoire tirée uniformément dans $[0, 1]$.

L'équation de transfert radiatif, quant à elle, pour un point x donné et une direction $\vec{\omega}$, exprime le changement de luminance L.

Nous avons vu deux interactions pouvant diminuer la valeur de la luminance. Dans le cas de l'absorption, ce changement est exprimé par [Jen09] :

$$(\vec{\omega} \cdot \nabla)L(x, \vec{\omega}) = -\sigma_a(x)L(x, \vec{\omega})$$

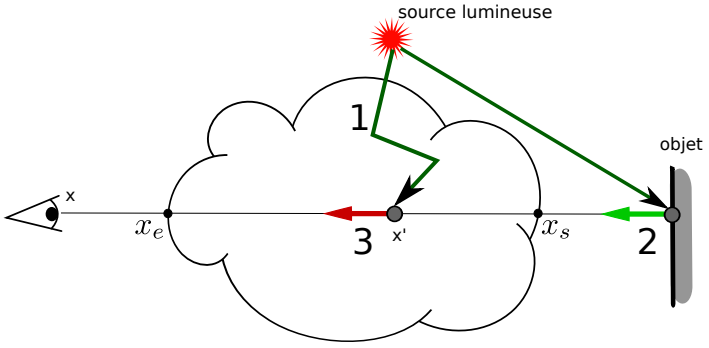


FIGURE 2 – Pour un point du milieu x', la luminance provient de la lumière issue des sources lumineuses traversant le volume (1), de la lumière réfléchie sur les surfaces (2) et de la lumière émise par le milieu (3).

De même, dans le cas de diffusion sortante :

$$(\vec{\omega} \cdot \nabla)L(x, \vec{\omega}) = -\sigma_s(x)L(x, \vec{\omega})$$

Soit, en combinant les deux formules :

$$(\vec{\omega} \cdot \nabla)L(x, \vec{\omega}) = -\sigma_t(x)L(x, \vec{\omega})$$

où $\sigma_t = \sigma_s + \sigma_a$.

Dans le cas d'apport d'énergie, le changement de luminance dû à l'émission est donné par :

$$(\vec{\omega} \cdot \nabla)L(x, \vec{\omega}) = \sigma_a(x)L_e(x, \vec{\omega})$$

où L_e est la luminance émise par le milieu. Quant à la diffusion incidente, nous obtenons :

$$(\vec{\omega} \cdot \nabla)L(x, \vec{\omega}) = \sigma_s(x) \int_{\Omega_{4\pi}} p(x, \vec{\omega'}, \vec{\omega})L_i(x, \vec{\omega})d\vec{\omega'}$$

où la luminance incidente, L_i, est intégrée sur toutes les directions d'une sphère, avec $p(x, \vec{\omega'}, \vec{\omega})$ la fonction de phase du milieu.

En combinant tous ces changements, nous obtenons le changement total de luminance par unité de distance :

$$(\vec{\omega} \cdot \nabla)L(x, \vec{\omega}) = \sigma_a(x)L_e(x, \vec{\omega}) - \sigma_t(x)L_e(x, \vec{\omega}) + \sigma_s(x) \int_{\Omega_{4\pi}} p(x, \vec{\omega'}, \vec{\omega})L_i(x, \vec{\omega})d\vec{\omega'}$$

Enfin, nous pouvons intégrer cette équation le long d'un segment de taille s pour obtenir la forme intégrée de l'ETR [Jen09] :

$$\begin{aligned} L(x, \vec{\omega}) &= \int_0^s e^{-\tau(x,x')}\sigma_a(x, x')L_e(x')dx' \\ &+ \int_0^S e^{-\tau(x,x')\sigma_s(x')} \int_{\Omega_{4\pi}} p(x', \vec{\omega'}, \vec{\omega})L_i(x', \vec{\omega'})d\vec{\omega'}dx' \\ &+ e^{-\tau(x,x+s\vec{\omega})}L(x + s\vec{\omega}, \vec{\omega}) \end{aligned}$$

Où $-\tau(x, x + s\vec{\omega})$ est la transmittance du milieu.

Parmi les méthodes de résolution de cette équation, nous trouvons des méthodes à base de cache de luminance, ainsi qu'une méthode probabiliste appelée *Volume Photon Mapping*. Ce sont ces méthodes que nous allons voir par la suite.

2.2 Équation de transfert radiatif et résolution

Nous verrons ici plusieurs méthodes de résolution de l'équation de transfert radiatif (ou ETR). Tout d'abord nous présenterons deux méthodes basées sur des caches de luminance, puis une méthode probabiliste nommée *Volume Photon Mapping* (Carte de Photons Volumiques).

2.2.1 Méthodes à base de caches de luminance

Sur une surface, l'éclairage direct peut varier énormément d'un pixel à l'autre à cause des lampes ou des ombres. L'éclairage indirect, quant à lui, varie progressivement. Ainsi, il devient possible d'estimer l'éclairage indirect en un point en se basant sur l'éclairage indirect des points voisins. La méthode de cache de luminance [Rib10] est basée sur cette observation. Elle permet alors d'éviter de calculer l'éclairage indirect en un point s'il a déjà été calculé dans un voisinage.

Pour ce faire, il faut utiliser des enregistrements pour plusieurs points de la scène, chacun de ces enregistrement contenant entre autres sa position, ainsi que les luminances incidentes en ce point. De plus, ces enregistrement ont une zone d'influence représentée par une sphère dont le rayon est une constante a. En dehors de cette zone, l'apport de l'enregistrement est négligeable.

La figure 3 représente une scène contenant déjà des enregistrements (carrés bleus) dont les limites des zones d'influences sont représentées par des cercles bleus. Pour calculer l'éclairage en un point visible p de la scène, on utilise les sources lumineuses pour le direct et les enregistrements dont le point p intersecte la zone d'influence.

La méthode de cache de luminance peut se traduire par l'algorithme suivant :

Algorithm 1 Calcul des luminances

```
1:  for all Points Pi visibles do
2:      calculer eclairage-direct
3:      if Pi est dans la zone d'influence d'enregistrements Rj then
4:          eclairage-indirect = combinaison des Rj
5:      else
6:          calculer eclairage-indirect
7:          ajouter enregistrement en Pi
8:      end if
9:      luminance en Pi = eclairage-direct + eclairage-indirect
10: end for
```

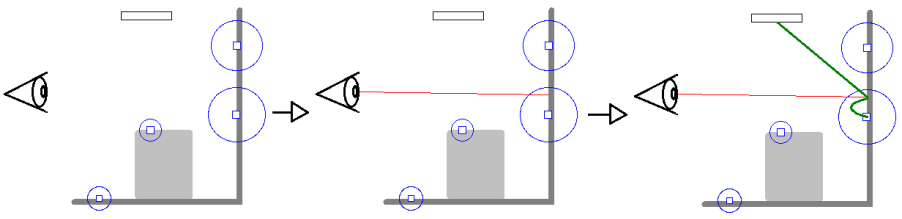


FIGURE 3 – Ici la scène contient déjà des enregistrements. On lance un rayon depuis la caméra pour obtenir un point visible. En ce point on calcule l'éclairage direct depuis la source lumineuse. Pour l'éclairage indirect, on utilise l'enregistrement à portée.

Le but est de construire le moins d'enregistrement possible. Le cache d'éclairement ne doit donc être rempli qu'en cas de nécessité. C'est pourquoi, lorsque l'on s'intéresse à un point visible Pi, on vérifie d'abord qu'il n'est pas dans la zone d'influence d'un ou plusieurs enregistrements. Si et seulement si c'est le cas, il est nécessaire de calculer précisément l'éclairage indirect en Pi pour créer un nouvel enregistrement. Nous sommes alors sûr de ne calculer que les enregistrements nécessaires

Plusieurs méthodes se basent sur ces caches de luminances. Nous allons voir ici une méthode d'estimation de luminance de faisceaux, nommée *Beam Radiance Estimate*, puis un cache d'éclairement pour les milieux participatifs appelé *Radiance Caching for Participating Media.*

Estimation de luminance de faisceau

Si la méthode de cache de luminance permet d'estimer la luminance pour un point sur une surface, rien n'existe cependant pour un point appartenant à un milieu. L'utilisation de la méthode *Beam Radiance Estimate* [JZJ08] permet de palier cela.

Cette méthode repose sur un envoi de faisceaux à travers le milieu qui vont collecter les enregistrements dont ils traversent la zone d'influence. Ici, un faisceau correspond à la portion d'un rayon traversant un milieu (Figures 4 et 5). Pour la phase de rendu, la luminance est alors calculée en intégrant la contribution des zones d'influence d'enregistrements interceptant le faisceau :

$$\frac{1}{N}\sum_{i=1}^{N} Ki(x,\vec{w},x_i)\tau(x,x_i')\rho(x_i,\vec{w_i},\vec{w_i'}\alpha_i),$$

avec N le nombre d'enregistrements collectés, K_i une fonction noyau permettant de calculer la contribution de l'enregistrement i et α_i le poids associé à l'enregistrement i.

Cette méthode permet le rendu de milieux participatifs dans une scène.

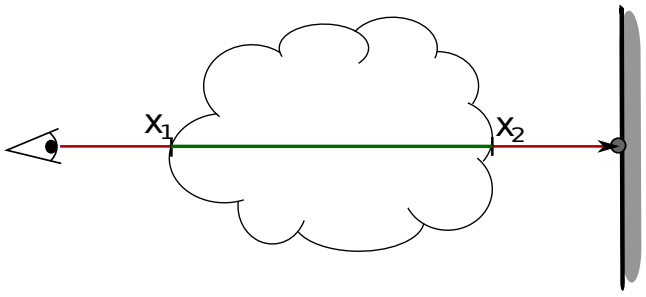


FIGURE 4 – Pour ce rayon, lancé depuis la caméra, il existe un faisceau correspondant à la portion de faisceau entre x_1 et x_2, points d'entrée et de sortie du milieu.

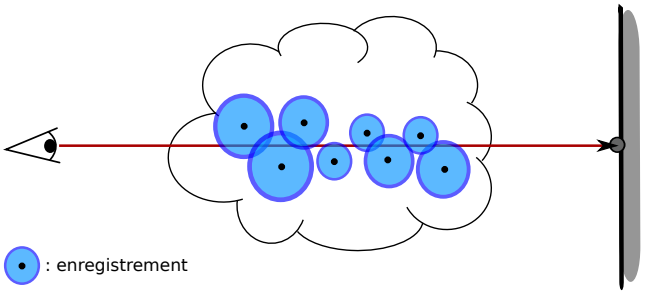


FIGURE 5 – Ici les disques bleus représentent les zones d'influences des enregistrements et les points noirs leurs positions. Pour calculer la luminance du milieu nous ne considérons que les enregistrements intersectant le faisceau.

Cache d'éclairement pour milieux participatifs

Dans cette méthode [JDZJ08], c'est le cache de luminance qui est étendu à des milieux participatifs. En plus de stocker des enregistrements pour des surfaces, il faut donc en stocker en tout point du milieu participatif. Ici, la partie la plus coûteuse est le calcul de la luminance diffuse incidente en un point p du milieu. Pour cela, depuis p, on envoie des rayons dont la direction est choisie aléatoirement, uniformément sur une sphère. Ces rayons subiront les diffusions du milieu jusqu'à s'arrêter ou rencontrer une surface.

La luminance au point d'arrêt des rayons est utilisée pour estimer la luminance indirecte en p. Cependant les directions des rayons étant choisies aléatoirement, il est possible de ne pas atteindre les sources lumineuses. Pour calculer les luminances directes, on envoie alors un rayon directement jusqu'à chacune des sources (Figure 6).

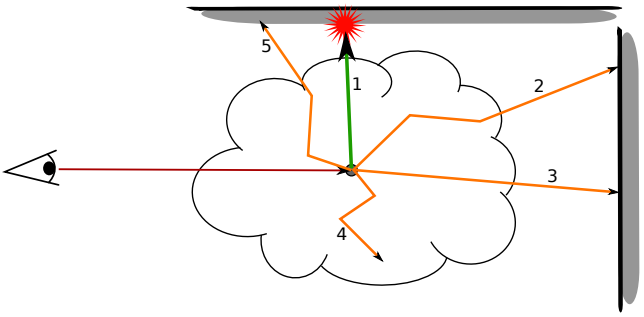


FIGURE 6 – Pour estimer la luminance en un point du volume, on calcule l'éclairage direct en envoyant un rayon vers la source (1) et l'indirect en envoyant des rayons dans des directions aléatoires. Certains seront diffusés jusqu'à ressortir du volume (2, 3 et 5) alors que d'autres s'arrêtent dans le volume (4).

Pour cette méthode il est donc nécessaire de stocker 4 caches de luminance pour chacun de nos enregistrements :

- cache des surfaces ;
- cache de la luminance due à la diffusion simple venant des sources lumineuses ;
- cache de la luminance due à la diffusion simple venant des surfaces ;
- cache de la luminance due aux diffusions multiples.

De même que précédemment, pour calculer la luminance en un point, on regarde si les caches les plus proches sont utilisables (donc si nous sommes dans une zone d'influence). Si c'est le cas il est possible de la calculer par interpolation. Sinon, il faut ajouter un nouvel enregistrement. De plus, il a été remarqué que l'intensité lumineuse diminue de façon exponentielle dans les milieux participatifs. C'est donc une interpolation exponentielle qui est utilisée dans le cas de cette méthode.

2.2.2 Carte de photons volumiques

Le *Volume Photon Mapping* [Jen09] est une extension aux milieux participatifs d'une autre méthode appelée *Photon Mapping* [Jen09]. Celle-ci est une méthode s'exécutant en deux étapes. Tout d'abord des photons sont émis dans une scène 3D depuis chacune des sources de lumière, tout en enregistrant les informations sur les intersections entre les photons et la scène, ce qui donne la carte de photons (ou *photon map*). Puis cette carte de photons est utilisée pour accélérer un rendu à base de lancer de rayons classique.

Sur la figure 7 nous pouvons voir les deux étapes du *Photon Mapping*. Le premier schéma représente le lancer de photons avec les trajets en bleu et les enregistrements représentés par des cercles. Une fois ces enregistrements créés, des rayons sont lancés depuis la caméra jusqu'à intersecter la scène. Les photons enregistrés à portée de ces points sont utilisés pour calculer la luminance des surfaces.

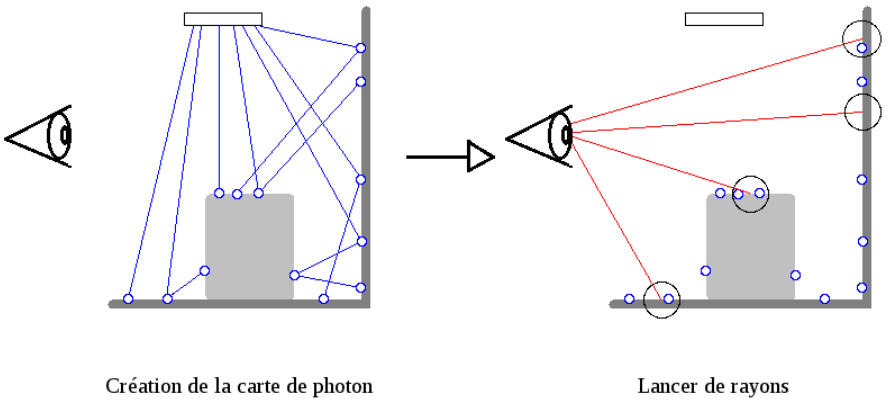


FIGURE 7 – Les deux phases du *Photon Mapping*

Pour la phase de lancer de photons, un nombre important de photons sont émis à partir de chaque source lumineuse de la scène. Chaque photon contient une fraction du flux émis par la source.

L'algorithme pour émettre les photons est le suivant :

Algorithm 2 Emettre_photons()

1: $nb_photons_lances = 0$
2: **while** cartes de photons non pleines **do**
3: // On tire une direction (θ, ϕ) depuis la source lumineuse
4: $\theta = random() \cdot \pi - \frac{\pi}{2}$
5: $\phi = random() \cdot 2 \cdot \pi$
6: $d = vecteur_Direction(\theta, \phi)$
7: $p =$ position de la source lumineuse
8: $lancer_Photon(p, d)$
9: $nb_photons_lances$ ++
10: **while** photon intersecte la scène en P **do**
11: $effectuer_enregistrement(photon, P)$
12: // On utilise une roulette russe pour gérer les reflexions du photon
13: // jusqu'à arrêt de celui-ci
14: $roulette_Russe()$ // détaillé en algorithme 3
15: **end while**
16: **end while**

La fonction `lancerPhoton` fonctionne de la même manière qu'un lancer de rayon classique. Cependant, là où l'utilisateur choisit arbitrairement si un rayon est réfléchi ou non lorsqu'il rencontre un objet, le photon, lui, sera réfléchi ou absorbé aléatoirement, selon le type de matériau de l'objet. La méthode utilisée pour cela est la roulette russe. La réflectivité du matériau étant comprise entre 0 et 1. Dans ce cas, la roulette russe revient à tirer un nombre aléatoire uniformément dans ce même intervalle. Puis, si le nombre obtenu est inférieur à la réflectivité du matériau, le photon est réfléchi, sinon il est absorbé.

Algorithm 3 Roulette_russe(photon p, materiau m)

1: $c = m$.reflectivite
2: $r = random()$
3: **if** $r < c$ **then**
4: reflechir(p)
5: **else**
6: absorber(p)
7: **end if**

Dans le cas d'une réflexion, c'est la BRDF (pour *Bidirectional Reflectance Distribution Function*) du matériau qui détermine la direction de celle-ci. La BRDF est une fonction de distribution permettant de connaître la probabilité qu'un rayon de lumière arrivant d'une direction ω_i sur une surface soit réfléchi selon une direction ω_o. Il est possible d'étendre la méthode de roulette russe pour prendre en compte la réflexion diffuse ou spéculaire, en considérant deux tests au lieu d'un (Algorithme 4).

Algorithm 4 Roulette_russe(photon p, materiau m)

1: $c_1 = m$.reflectiviteDiffuse
2: $c_2 = m$.reflectiviteSpeculaire
3: $r = random()$
4: **if** $r < c_1$ **then**
5: reflexion_diffuse(p)
6: **else if** $r < (c_1 + c_2)$ **then**
7: reflexion_speculaire(p)
8: **else**
9: absorber(p)
10: **end if**

Quand un de ces photons intersecte un des éléments de la scène, on stocke cette intersection dans la carte de photons. Celle-ci est stockée sous la forme d'un arbre kd-tree équilibré dont chaque noeud contient la position de l'intersection, l'énergie contenue par le photon et la direction incidente du photon. Au delà d'être pratique à utiliser en mémoire, le kd-tree équilibré permet une recherche efficace des photons dans la scène. Cependant, en pratique deux cartes de photons sont utilisées : une servant à stocker uniquement les caustiques et une carte de photons globale.

Pour la phase de rendu, il faut lancer un rayon à travers chaque pixel de l'image et récupérer le point p d'intersection entre ce rayon et la scène. Pour chacun de ces points p, il faut localiser les photons les plus proches, grâce au kd-tree. Enfin, l'éclairage en p est calculé à l'aide d'une estimation de densité (Algorithme 5).

Pour le *Volume Photon Mapping*, ces deux étapes sont réutilisées. La seule différence est qu'une troisième carte de photons est ajoutée afin de stocker les photons interagissant avec le milieu participatif.

Algorithm 5 Estimation(point p, direction w, int n)

localiser les n photons les plus proches
r = distance au n^{ieme} photon le plus proche $resultat = 0$
for all photon p_i **do**
    $d = p_i$.direction
    $\alpha = p_i$.energie
    $resultat+ =$ fonction_de_repartition(x, w, d) $\cdot\alpha$
end for
$resultat = resultat \cdot \frac{1}{2\pi} \cdot r^{-2}$
return $resultat$

Lors de la phase de lancer de photons, si l'un d'eux entre dans le milieu, il faut tirer aléatoirement une distance au bout de laquelle il interagira avec ce dernier. Cette distance est donnée par :

$$d = \frac{-log\xi}{\kappa},$$

avec ξ tiré aléatoirement sur $]0, 1]$ et κ le coefficient d'extinction.

Lorsque le photon interagit avec le milieu, la méthode de roulette russe est utilisée pour savoir s'il est absorbé ou diffusé. Si le photon est diffusé, comme dans le cas du *Photon Mapping*, la nouvelle direction est tirée aléatoirement selon la fonction de phase du milieu (Figure 8).

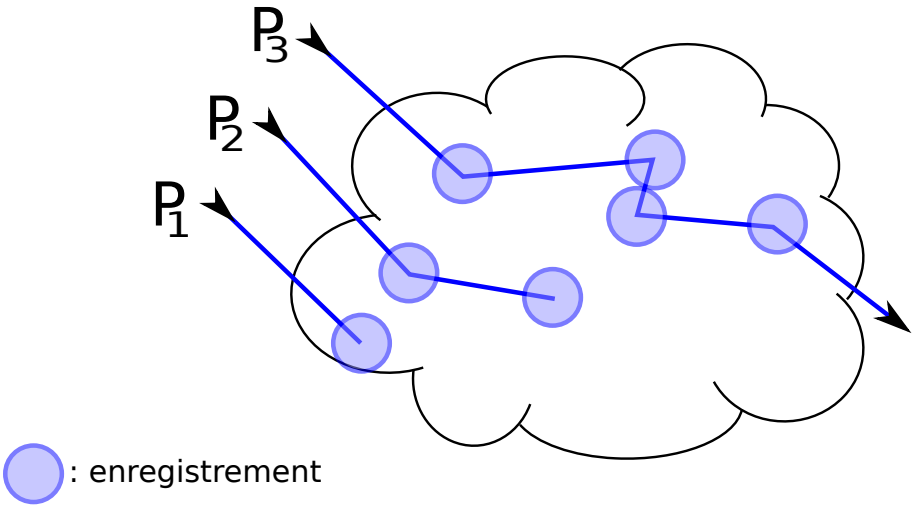


FIGURE 8 – Schéma du trajet de trois photons p_1, p_2 et p_3 dans un milieu participatif. Une fois dans le milieu, p_1 est absorbé dès la première interaction, p_2 est diffusé une fois et p_3 est diffusé successivement jusqu'à ressortir du milieu. À chaque interaction, un enregistrement est créé dans le milieu.

2.3 Carte de photons progressive

Nous avons vu dans la partie précédente qu'il existe plusieurs méthodes de résolution de l'équation de transfert radiatif. Cependant, toutes ces méthodes sont très coûteuses à la fois en temps de calcul et en mémoire, puisqu'il faut calculer puis stocker les nombreux enregistrements ou photons nécessaires à un résultat satisfaisant.

Nous avons entre autre vu la méthode du *Volume Photon Mapping*, qui s'inspire de la méthode de *Photon Mapping*. Dans le cas de scènes complexes, pour ces deux méthodes, il faudra générer un nombre énorme de photons afin d'obtenir des résultats corrects. Bien souvent il faudra même générer plus de photons que les mémoires d'ordinateurs actuels le permettent. Or il existe une amélioration permettant de sensiblement alléger le *Photon Mapping*. Une idée pourrait donc être d'étendre cette amélioration, appelée *Progressive Photon Mapping* [HOJ08], aux milieux participatifs.

Le but de cette méthode est d'éviter d'avoir à stocker la totalité de la carte de photons en mémoire. Pour cela, les traditionnelles phases de lancer de photons et lancer de rayons sont inversées. De plus, la phase de lancer de photons pourra être répétée afin d'améliorer la qualité du rendu.

La phase de lancer de rayons est similaire à celle effectuée précédemment, sans porter d'attention aux photons, puisqu'ils n'ont pas encore été lancés. De plus, tous les rayons sont réfléchis jusqu'à rencontrer une surface spéculaire. Il est cependant possible d'utiliser une méthode de type roulette russe pour arrêter le rayon plus tôt. Une fois les rayons lancés, il faut stocker les informations sur chacun des points p_i correspondants aux intersections ou réflexions.

Pour chacun de ces points, les données à stocker sont, entre autres :

- la position x ;
- la normale en x ;
- la direction du rayon $\vec{\omega}$;
- le rayon de la zone d'influence du point ;
- le flux reçu τ.

Vient ensuite la phase de lancer de photons. L'idée ici est de ne stocker que les photons influant sur les points stockés lors du lancer de rayons. Pour cela les photons sont lancés de manière classique, mais ne sont gardés que ceux se trouvant dans la zone d'influence des points p_i. Ensuite ces photons servent à mettre à jour le flux reçu τ, puis à raffiner l'estimation de la luminance en ces points. Après, les photons ne sont plus utiles et il n'est plus nécessaire de les garder en mémoire. Cette étape de lancer de photons est répétée jusqu'à ce que la qualité de l'image soit suffisante (Figures 9 et 10).

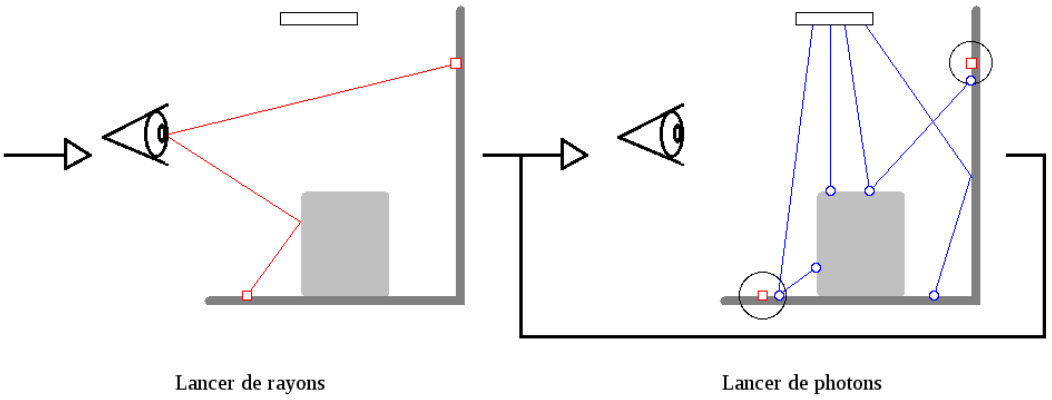


FIGURE 9 – Le *Progressive Photon Mapping* est constitué d'une passe de lancer de rayons, suivie de plusieurs passes de lancer de photons

Le choix du rayon de la zone d'influence des points peut cependant poser un problème. La valeur de la luminance en p_i étant la moyenne des luminances de la zone d'influence, un fort rayon de cette dernière provoquera une image floue. Mais un rayon trop faible ne prendra pas assez de photons en compte pour obtenir une image correcte. Pour résoudre cela, une méthode consiste à commencer par un grand rayon et le mettre à jour à chaque passe de lancer de photons de manière à le faire diminuer progressivement. Le rayon diminuant, la méthode se concentre sur les photons les plus proches du point concerné et l'image résultat converge vers une image détaillée.

Ainsi, l'algorithme du *Progressive Photon Mapping* est le suivant :

Algorithm 6 Progressive_Photon_Mapping()

1: $\{P_i\} = lancer_rayons()$
2: **while** qualite de l'image insuffisante **do**
3: $lancer_photons()$
4: **for all** points d'intersection P_i **do**
5: mettre a jour le flux reçu par P_i
6: mettre a jour le rayon de la zone d'influence de P_i
7: **end for**
8: $effacer_photons()$
9: **end while**

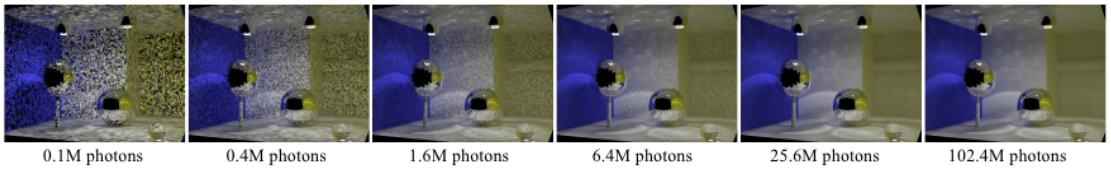


FIGURE 10 – Résultats au fur et à mesure des itérations

3 Présentation du stage

Nous avons vu ce qu'était un milieu participatif et l'équation de transfert radiatif permettant de modéliser les interactions entre ces milieux et la lumière. Nous avons aussi vu plusieurs méthodes actuellement utilisées pour les rendus de ce type de milieu, soit en utilisant des caches de luminance, comme le *Beam Radiance Estimate* ou le *Radiance Caching for Participating Media*, soit en utilisant des cartes de photons avec le *Volume Photon Mapping*.

Cependant, dans le cas de scènes trop complexes, une quantité importante d'enregistrements ou de photon devra être stockée afin d'obtenir une image correcte. Ces méthodes vont donc être très gourmandes à la fois en mémoire, les mémoires actuelles pouvant être insuffisantes pour certaines scènes, mais aussi en temps de calcul puisque les recherches d'enregistrements ou de photons parmis les cartes vont être longues si les cartes sont grandes.

Cependant, certains chercheurs ont proposé une amélioration de la méthode de rendu non volumique, le *Photon Mapping*, qui est à l'origine du *Volume Photon Mapping*. Cette amélioration permet non seulement au rendu d'être très peu gourmand en mémoire, mais aussi d'accélérer sensiblement le processus de rendu. Cette amélioration, nommée *Progressive Photon Mapping*, peut alors aussi être intéressante dans le cadre de rendu volumique.

L'objectif du stage est alors d'appliquer le cheminement parcouru du *Photon Mapping* jusqu'au *Volume Photon Mapping* mais cette fois en partant du *Progressive Photon Mapping*. Tout cela afin d'obtenir des rendus volumiques en des temps et coûts mémoire raisonnables. Cette méthode serait alors un *Progressive Volume Photon Mapping*

3.1 Choix de départ

Afin de pouvoir évaluer une nouvelle méthode, il nous faudra pouvoir la comparer avec les méthodes existantes. Pour cela nous avons choisi de commencer avec un moteur de rendu existant, intégrant déjà plusieurs de ces méthodes, auquel nous pourrions greffer la nôtre.

Notre choix s'est porté sur PBRT (version2 du 28 janvier 2011) (pour *Physically Based Rendering Tool*) , un moteur de rendu développé par Matt Phar et Greg Humphreys [PH04]. Nous l'avons choisi pour plusieurs raisons. Tout d'abord il fonctionne avec un système de modules. Il nous semblait alors facile d'y ajouter une nouvelle méthode de rendu. De plus, il possède déjà une technique de *Photon Mapping* que nous pourrons étendre à du *Volume Photon Mapping* dans un premier temps (la méthode n'étant pas disponible) puis en *Progressive Volume Photon Mapping*. PBRT possède aussi plusieurs structures accélératrices, comme des Kd-Trees pour stocker nos photons, et propose une parallélisation facile des algorithmes. Enfin nous l'avons aussi choisi parce qu'il est déjà utilisé dans le milieu scientifique.

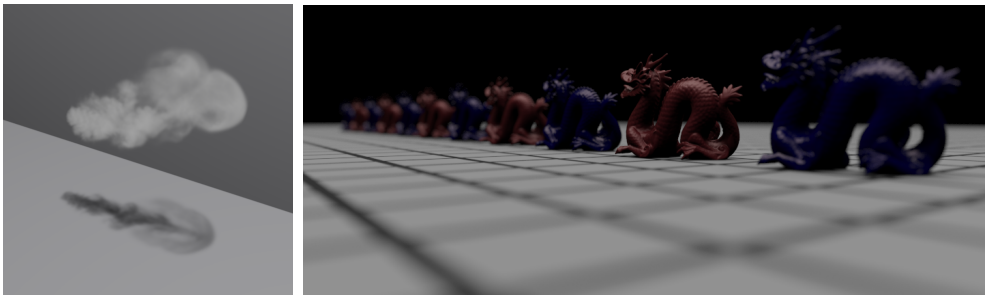


FIGURE 11 – Exemples de rendus avec PBRT

4 Implémentation de la carte de photons volumiques

Notre première tâche a été d'étendre aux milieux participatifs la méthode de *Photon Mapping* intégrée dans PBRT. Pour cela nous présenterons le fonctionnement de PBRT en général et de son lancer de photons en particulier. Avoir une implémentation du *Volume Photon Mapping* nous servira par la suite de méthode de comparaison, une fois notre méthode de *Prdiogressive Volume Photon Mapping* créée.

4.1 Carte de photons de PBRT

Pour pouvoir effectuer un rendu avec PBRT, il faut spécifier deux méthodes de rendu : une pour les surfaces et une pour les volumes (dans PBRT ces milieux sont simplement appelés volumes, nous pourrons utiliser cette appellation dans la suite du document). Pour chaque pixel de l'écran, la méthode de rendu surfacique nous donne la luminance provenant de la portion de surface visible Li. La méthode de rendu volumique, quant à elle, donne à la fois la luminance provenant du volume visible Lvi et la transmittance associée Tr. PBRT calcule ensuite la couleur des pixels en faisant $Li \times Tr + Lvi$. La transmittance représente l'atténuation de la lumière traversant le volume. Un volume très dense aura une transmittance quasi nulle et un volume léger aura une transmittance proche de 1.

Le *Photon Mapping* fait partie des méthodes de rendu surfacique. Comme les autres méthodes de rendu de PBRT, elle fonctionne en deux temps :

1. Une phase de pré-traitement consistant au remplissage des cartes de photons.
2. Une phase de rendu où, pour un rayon partant de la caméra et traversant un pixel, l'on calculera la luminance surfacique associée.

L'algorithme du pré-traitement est très simple et permet de remplir deux cartes de photons : une pour les photons indirects (issus de réflexions par exemple) et une pour les caustiques. Les photons directement issus des sources lumineuses ne sont pas utiles car PBRT recalcule l'éclairage direct au moment du rendu. De plus, afin de visualiser les ombres dues à un éventuel volume, lorsqu'un photon traverse un volume, seule la lumière qu'il transporte est atténuée.

L'algorithme de lancer de photon est alors le suivant :

Algorithm 7 Remplir_carte_photons()

```
1: while les cartes ne sont pas pleines do
2:    Lancer un photon depuis la source lumineuse
3:    while le photon intersecte les surfaces do
4:       if le photon traverse le milieu then
5:          attenuer la lumiere transportee par le photon
6:       end if
7:       stocker le photon dans les cartes de photons correspondantes
8:       Calculer le photon reflechi ou s'arreter (roulette russe)
9:    end while
10: end while
```

Ces cartes sont alors stockées sous forme de Kd-Trees, afin d'accélérer la recherche de photons durant la phase de rendu. Lorsqu'un photon est stocké, on stocke en fait trois valeurs :

- p la position du photon lors de l'enregistrement ;
- α l'apport lumineux du photon ;
- ω_i la direction incident du photon.

La phase de rendu s'effectue du point de vue de la caméra. Depuis celle-ci, PBRT lance un rayon à travers chacun des pixels. Ensuite, pour chacun des points d'intersection P_i, PBRT récupère les photons les plus proches. La luminance surfacique sera enfin estimée au niveau des P_i en utilisant ces photons.

4.2 Extension au volume

Notre but ici est d'étendre la méthode précédente au rendu de milieux participatifs. Cependant, le *Volume Photon Mapping* est une méthode permettant à la fois de visualiser les surfaces et les volumes. PBRT appelant les méthodes de rendu surfaciques et volumiques séparément, nous avons ajouté une troisième méthode globale pouvant remplacer les deux autres.

La première étape ensuite est de changer le comportement des photons durant la phase de lancer. Comme nous avons vu dans la première partie, un photon traversant un milieu peut interagir avec celui-ci. Quand c'est le cas, soit le photon est absorbé, soit il est diffusé. De plus nous devons gérer une carte de photons supplémentaire, la carte de photons volumiques.

La méthode de lancer devient alors :

Algorithm 8 Remplir_carte_photons()

```
1: while les cartes ne sont pas pleines do
2:    Lancer un photon depuis la source lumineuse
3:    while le photon intersecte les surfaces do
4:       if le photon traverse et interagit avec le milieu then
5:          stocker photon dans la carte volumique
6:          if absorption (roulette russe) then
7:             arreter le photon courant
8:          else
9:             mettre a jour la direction du photon
10:         end if
11:      end if
12:      stocker le photon dans les cartes de photons correspondantes
13:      Calculer le photon reflechi ou s'arreter (roulette russe)
14:   end while
15: end while
```

Il n'est cependant plus nécessaire d'atténuer la lumière transportée par le photon. En effet, les photons en traversant le volume vont être absorbés ou déviés. Les ombres vont toujours se former sous le volume, mais cette fois à cause d'une plus faible concentration de photon.

Un des points délicat est la gestion des interactions dans le volume. Nous avons implémenté dans les classes de volumes de PBRT une fonction permettant de savoir si un évènement se produit sur un trajet donné. Dans ce cas, ces fonctions calculent la position de l'interaction, son type (absorption ou diffusion) et dans le cas d'une diffusion, la nouvelle direction du photon.

Il y a deux types de volumes existants dans PBRT : les volumes homogènes et les volumes hétérogènes représentés sous forme de grilles régulières. Implémenter une telle fonction est facile dans le cas du volume homogène où les caractéristiques du milieu sont les mêmes partout, la fonction est une variante de l'algorithme 4 (Roulette russe). En cas de diffusion, il est ensuite possible de connaître la nouvelle direction en utilisant les fonctions de phases déjà présentes dans pbrt.

Cependant pour le cas des volumes hétérogènes, chaque cellule de la grille régulière possède ses caractéristiques propres. Il faut alors les traiter comme des volumes séparés. Ainsi, lorsque qu'un photon entre dans une première cellule, nous calculons la distance qu'il parcourt avant qu'un évènement ne se produise. Si parcourir cette distance nous fait changer de cellule, nous redémarrons de zéro à l'entrée de la nouvelle cellule. Sinon, nous traitons l'évènement comme dans le cas d'un volume homogène.

Pour la phase de rendu, le *Volume Photon Mapping* est basé sur le *Photon Mapping* qui permet déjà de calculer la luminance surfacique associée à un pixel. Nous avons alors seulement à implémenter une fonction pour la luminance volumique et une autre pour la transmittance. Nous avons utilisé pour ces deux fonctions une méthode à base de ray marching.

Pour calculer la luminance volumique, nous allons parcourir le rayon caméra pixel depuis le point de sortie du volume jusqu'au point d'entrée, avec un pas λ constant (choisi par l'utilisateur). À chaque pas sur le rayon nous cherchons les photons proches (Figure 12). Au n^{ieme} pas sur le rayon, la luminance volumique Lvi_n est calculée ainsi :

$$Lvi_n = \frac{1}{n} \sum_{i=1}^{|photons|} \alpha_i \cdot \frac{3}{4\pi \cdot maxd} \cdot p(x_i, \vec{\omega}, \vec{\omega_i}) \cdot \lambda$$

avec α_i l'apport lumineux du i^{ieme} photon (estimation de densité), $\vec{\omega_i}$ sa direction incidente, x_i sa position, $maxd$ la distance au photon le plus éloigné et p la fonction de phase du milieu.

Pour calculer la transmittance, nous parcourons le volume de la même manière et mesurons les transmittances Tr_i à chaque arrêt. Nous considérons ensuite que la transmittance est constante entre chaque arrêt. La transmittance totale est donc la somme des transmittances multipliée par la longueur d'un pas :

$$Transmittance = \sum_{i=1}^{n} Tr_i \cdot stepsize$$

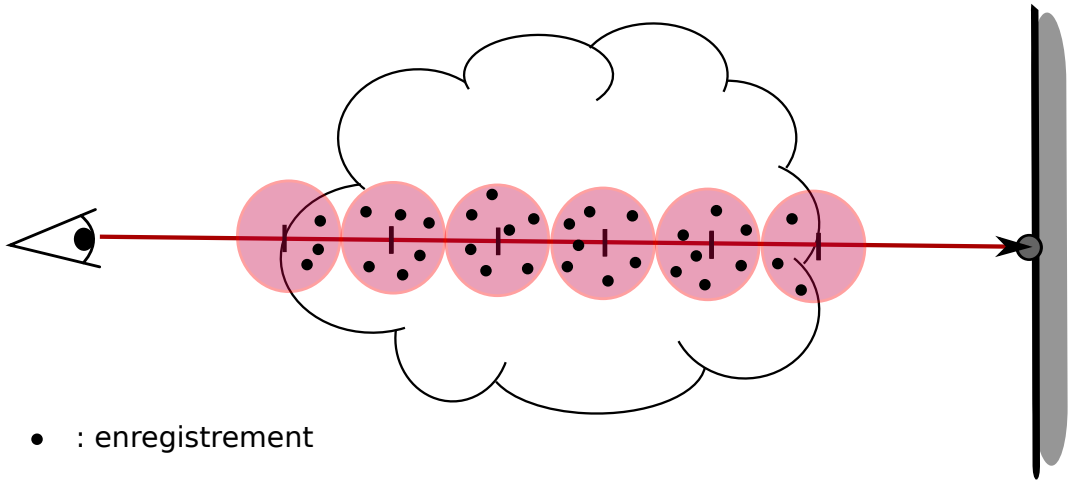


FIGURE 12 – Pour le rendu volumique, nous avançons à pas constant sur le faisceau. À chaque pas nous estimons la luminance avec les photons suffisamment proches de la position courante.

4.3 Estimation de luminance de faisceaux

Une fois le *Volume Photon Mapping* implémenté, l'équipe de recherche a implémenté la méthode de *Beam Radiance Estimate*. PBRT utilisant des rayons lancés depuis la caméra pour le rendu, il est facile de calculer les faisceaux associés. Ensuite, il a fallu rajouter à la structure des photons un rayon d'influence. Enfin, pour estimer la luminance volumique en un point du faisceau, au lieu de considérer les n photons les plus proches, il faut considérer les photons dont le faisceau intersecte la zone d'influence.

4.4 Résultats

Les deux méthodes ainsi implémentées nous seront surtout utiles par la suite, pour valider les résultats du *Progressive Volume Photon Mapping* et comparer les ressources utilisées. Cependant nous pouvons déjà comparer ces méthodes de rendu entre elles, ainsi qu'avec le rendu volumique de PBRT présent de base.

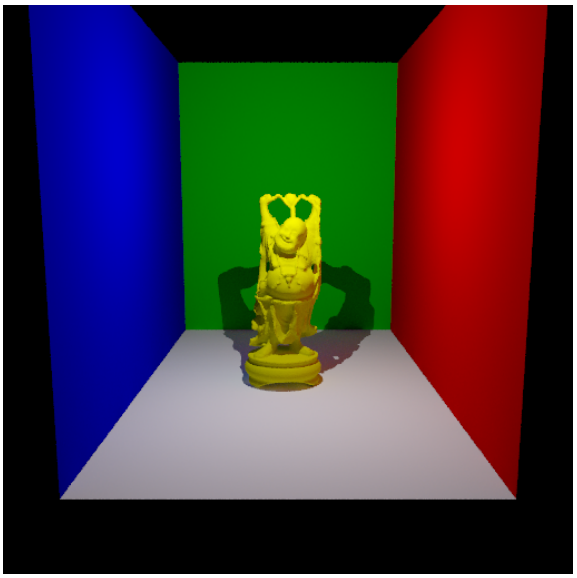
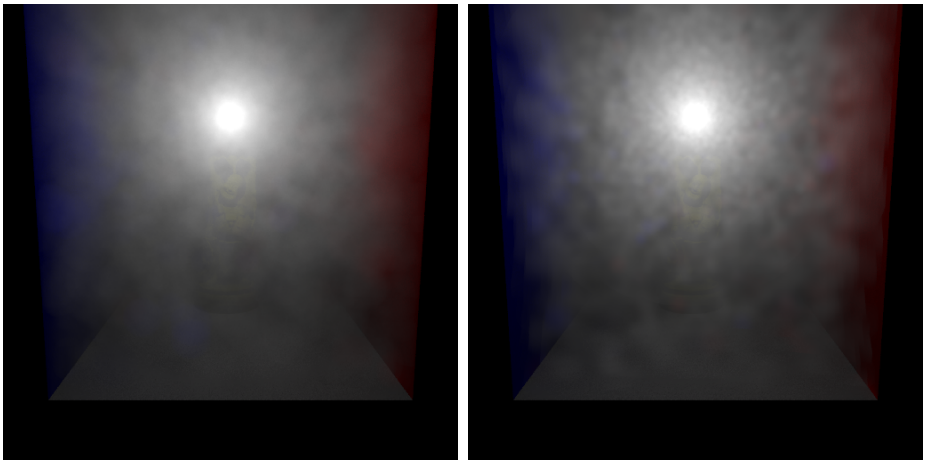


FIGURE 13 – Rendu de la scène d'exemple sans fumée avec la méthode de *Photon Mapping*. Temps de rendu : 2m7s.

Nous avons effectué ces tests sur une scène simple composée de 3 murs et une figurine de Bouddha posée au sol. La scène est éclairée par une lumière ponctuelle située au dessus de la tête du Bouddha. Enfin la scène est totalement plongée dans une fumée homogène. La scène mesure 80 unités de longueur en largeur, profondeur et hauteur.

Le rendu avec la méthode de *Photon Mapping* est très rapide, ne prenant que deux minutes (Figure 13). Cependant ce rendu ne prend pas en compte le volume et nous voyons la scène vide de fumée.

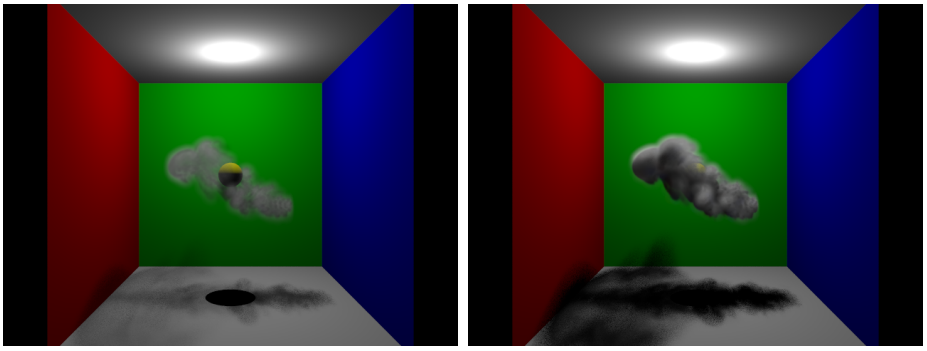


(a) Temps de rendu : 12m47s (b) Temps de rendu : 17m19s

FIGURE 14 – La même scène enfumée, rendue avec les techniques de *Volume Photon Mapping* à gauche et *Beam Radiance Estimate* à droite.

Les rendus suivants ont été effectués avec le *Volume Photon Mapping* et le *Beam Radiance Estimate* (Figure 14). Pour ces rendus nous avons utilisé 400000 enregistrements photons volumiques, avec l'utilisation de 100 enregistrements à chaque étape du raymarching. De plus, le pas d'avancé sur le rayon était de 2 unités de longueur. L'effet de moutonnement que l'on peut observer vient du faible nombre de photons utilisés.

Nous avons aussi vérifié le fonctionnement de la méthode avec des fumées hétérogènes dans une scène similaire où nous avons remplacé le Bouddha par une sphère au milieu de la fumée (Figure 15).



(a) Fumée peu dense (b) Fumée très dense

FIGURE 15 – Exemples de rendu de fumées hétérogènes avec le *Volume Photon Mapping*.

5 Carte de photons volumiques progressive

Maintenant que nous avons une méthode de référence, le *Volume Photon Mapping*, nous pouvons créer la version progressive. Le but de cette méthode est d'accélérer le rendu et surtout de limiter l'occupation mémoire.

5.1 Implémentation

Pour cela nous allons inverser le processus de rendu, comme cela a été fait pour le *Progressive Photon Mapping*. Tout d'abord, nous nous plaçons au niveau de la caméra. Depuis celle-ci, et à travers chaque pixel de l'image, nous allons lancer un rayon. Cela nous permet de connaître les surfaces ainsi que les zones de volume visibles. Une fois celle-ci connues, nous envoyons des photons dans la scène afin de mettre à jour progressivement les luminances associées. De cette manière, les photons sont utilisés aussitôt après avoir été lancés et il n'est pas nécessaire de les garder en mémoire.

Dans le cadre du *Progressive Photon Mapping*, les parties visibles de la scène sont représentées par des disques sur les surfaces, positionnés là où les rayons intersectent la scène (Figure 16). Ensuite les photons entrant en contact avec la surface à l'intérieur du disque sont utilisés pour mettre à jour la luminance surfacique associée au rayon. Le rayon de ces disques diminue au fur et à mesure du rendu afin d'augmenter progressivement la précision de l'image en ne considérant que les photons les plus proches du point d'intersection.

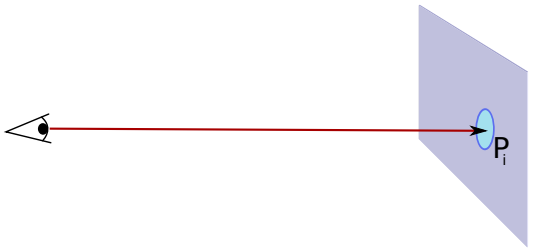


FIGURE 16 – Un rayon est lancé depuis la caméra et intersecte la scène en P_i.Dans le cadre du *Progressive Photon Mapping*, la partie visible est le disque bleu entourant P_i

Il n'existe cependant pas d'équivalent à ces disques surfaciques pour le volume. Afin d'adapter l'algorithme aux milieux participatifs,nous nous sommes inspirés de la méthode *Beam Radiance Estimate* et avons opté pour l'utilisation de faisceaux. Ainsi lorsque qu'un rayon traverse le volume, nous lui associons un cylindre correspondant à la portion de rayon à l'intérieur du volume (Figure 17). Afin de conserver l'analogie, nous diminuerons le rayon du cylindre à chaque passe de lancer de photons. Lors du lancer de photons, s'il y a un évènement à l'intérieur d'un des cylindres

ainsi créés, nous mettrons alors à jour la luminance volumique du rayon associé.

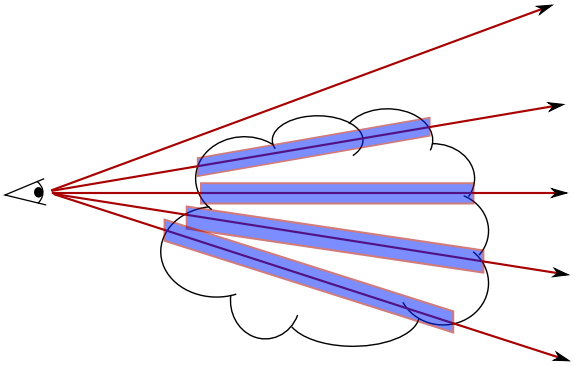


FIGURE 17 – Sur les cinq rayons lancés depuis la caméra, quatre intersectent le milieu. On leur associe quatre faisceaux, représentés par des zones bleues ici.

Comme les autres méthodes, le *Progressive Volume Photon Mapping* possède alors une phase de pré-traitement, puis une phase de rendu. À la différence près que nous bouclons sur la phase de rendu jusqu'à ce que la qualité de l'image soit satisfaisante.

Durant la phase de pré-traitement, nous calculons tout ce qui n'aura pas a être mis à jour au cours des différents rendus : les points visibles sur les surfaces et les faisceaux dans les volumes.

La structure de faisceau utilisée est la suivante :

- Cylindre
- Transmittance à la sortie du faisceau : Tr
- Luminance volumique Lvi (nulle avant la première phase de rendu)

Nous pourrions dès lors calculer une première image, la couleur d'un pixel étant obtenue comme précédemment : $Li \times Tr + Lvi$ (avec L_i la luminance surfacique et Lv_i la luminance volumique). Cependant le volume ne sera pas encore visible.

Pour la phase de rendu ensuite (Algorithme 9), nous générons un nombre limité de photons depuis les sources lumineuses. Ces photons sont envoyés à travers la scène, et à chaque interaction avec le volume, si ce dernier a lieu à l'intérieur d'un ou plusieurs cylindres, nous associons un enregistrement du photon aux faisceaux correspondants (Figure 18). Une fois la phase de lancer terminée, nous savons quel photon a impacté quel faisceau. Nous pouvons alors mettre à jour la luminance volumique des faisceaux.

Algorithm 9 Phase_de_rendu_volumique()

```
1: for i = 1 à n do
2:    lancer photon ph depuis la source lumineuse
3:    while intersection avec la scène do
4:       if interaction avec le volume en p then
5:          chercher les faisceaux incluant p
6:          ajouter ph à ces faisceaux
7:          diffuser ou absorber le photon (roulette russe)
8:       else if intersection avec une surface then
9:          if reflexion (roulette russe) then
10:            ph = reflexion(ph)
11:         else
12:            // absorption
13:            break
14:         end if
15:      end if
16:   end while
17: end for
18: for all faisceaux do
19:    mettre a jour Lv_i
20: end for
21: effacer les enregistrements
```

Pour cela nous utilisons la formule suivante, pour des photons $(x_i, \alpha_i, \vec{\omega}_i)$:

$$Lvi = \frac{1}{|photons|} \sum_{photons} (\kappa \cdot Tr \cdot \sigma_s(x_i) \cdot p(x_i, \vec{\omega}, \vec{\omega}_i) \cdot \alpha_i)$$

avec κ une fonction noyau, Tr la transmittance, σ_s le coefficient de diffusion du milieu, p la fonction de phase du milieu, et $\vec{\omega}$ la direction du faisceau.

La fonction noyau permet de pondérer l'apport du photon en fonction de sa distance au centre du faisceau :

$$\kappa(r, x, d) = r^{-2} \cdot 3\pi^{-1}(1 - x^2) \cdot \frac{d}{r}$$

avec r le rayon du faisceau, x la position du photon et d la distance entre le photon et le centre du cylindre.

5.2 Résultats

L'avantage ici est que nous ne lançons qu'une quantité limitée de photons à la fois, nous n'occupons alors qu'une quantité limitée de mémoire, qui restera constante quel que soit le nombre d'itérations effectuées.

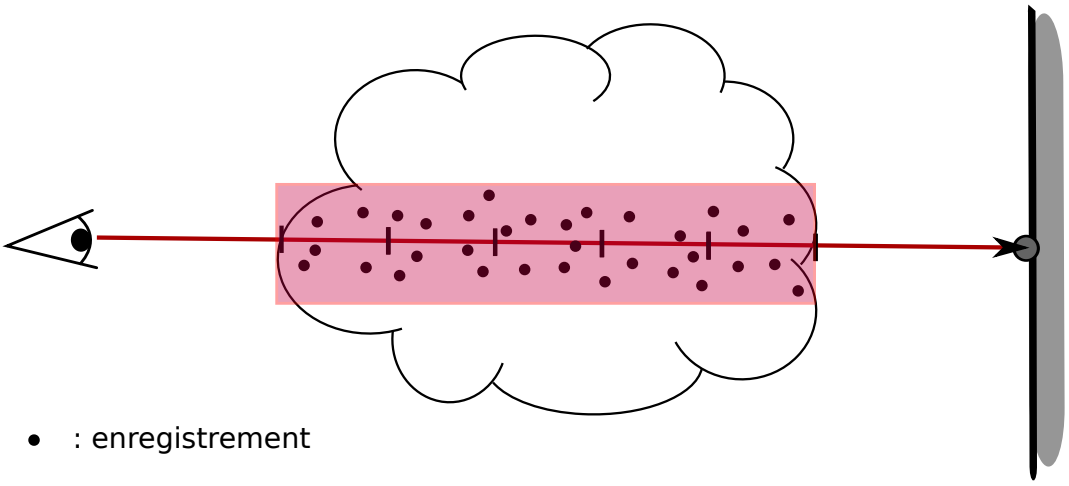
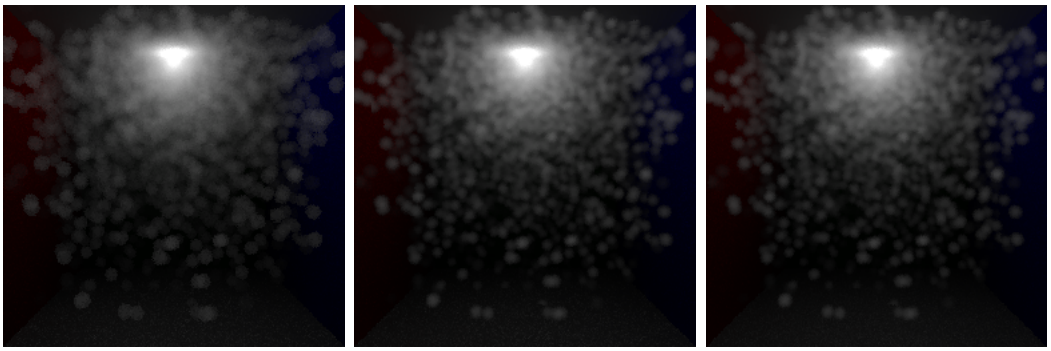


FIGURE 18 – Seuls les enregistrements de photons effectués à portée du faisceau sont conservés pour effectuer le rendu.

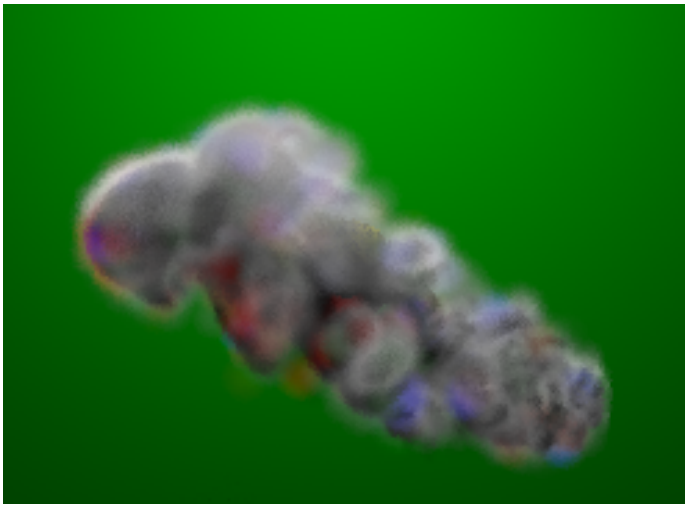


(a) *Volume Photon Mapping* (b) *Beam Radiance Estimate* (c) *Progressive Volume Photon Mapping*

FIGURE 19 – Une scène avec une carte de photons précalculée rendue avec trois méthodes différentes.

Afin de valider la méthode, nous avons utilisé une scène consistant en une boîte vide avec une lampe ponctuelle au plafond. Nous y avons généré une carte de 10000 photons que nous avons sauvegardée dans un fichier. Ce faible nombre de photons permet de simuler une unique passe de rendu du *Progressive Volume Photon Mapping*. Ensuite, nous avons utilisé cette carte de photons à l'aide des trois méthodes implémentées : le *Volume Photon Mapping*, le *Beam Radiance Estimate* et enfin la méthode courante (Figure 19).

Nous pouvons remarquer que l'image obtenue avec notre méthode et celle obtenue avec le *Beam Radiance Estimate* sont identiques. Notre méthode semble alors fonctionner correctement. Cependant, les temps de rendu ne sont pas du tout comparables. Pour cet exemple, le *Beam Radiance Estimate* ne prend que 23 secondes alors que notre méthode a besoin de 8 minutes pour rendre la même image.



(a) *Volume Photon Mapping*

FIGURE 20 – Utilisation du *Progressive Volume Photon Mapping* sur une fumée hétérogène.

Nous pouvons aussi l'utiliser sur la même fumée homogène que précédemment, en générant encore une fois 10000 photons (Figure 20). Les taches rouges et bleues proviennent des photons se réfléchissant sur les murs rouges et bleus de la scène. Comme nous utilisons beaucoup moins de photons que dans le cas du *Volume Photon Mapping*, les photons réfléchis ont beaucoup plus de poids sur le résultat et sont donc plus visibles.

6 Structure accélératrice

Nous avons donc un *Progressive Volume Photon Mapping* fonctionnel. Celui-ci nous permet d'augmenter considérablement le nombre de photons utilisés pour générer une image. Cependant pour savoir quels faisceaux un photon va impacter, nous devons tester son appartenance à autant de cylindres qu'il y a de faisceaux. Cela engendre des temps de calculs immenses alors qu'un de nos objectifs était de créer une méthode de rendu plus rapide que le *Volume Photon Mapping* classique. Il nous faut donc trouver un moyen d'accélérer cette recherche de faisceau.

6.1 Hiérarchie de faisceaux

Pour cela nous avons décidé d'organiser nos faisceaux sous forme de hiérarchie. L'idéal serait de pouvoir regrouper les faisceaux proches spacialement. Nous pourrions ensuite associer à chacun de ces groupes un faisceau englobant. Ainsi lorsque l'on cherche les faisceaux impactés par un photon, nous pourrons tester l'appartenance du photon au faisceaux englobants, et en cas de succès seulement, aller tester l'appartenance aux faisceaux du groupe.

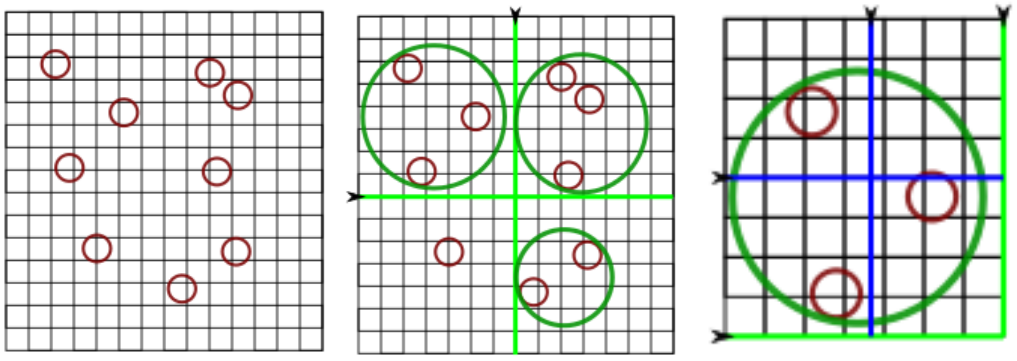
Il nous faut cependant trouver un critère pour regrouper nos faisceaux, car on ne peut pas définir la position spatiale d'un faisceau comme on la définit pour un point. Nous savons par contre que les rayons porteurs de nos faisceaux ont tous la même origine, la caméra. Ainsi, nous savons que deux rayons de directions proches vont créer des faisceaux proches spacialement. Enfin nous savons que deux rayons de directions proches vont intersecter le plan image en deux points proches eux aussi. Nous avons donc décidé de regrouper nos rayons selon la position de leurs intersections avec le plan image.

Avec ce critère de regroupement, nous allons pouvoir construire un arbre quad-tree hiérarchisant nos faisceaux. Pour créer ce quad-tree, nous séparons d'abord le plan image en quatre sous plans égaux. Les faisceaux appartiennent au sous plan où se trouve leur point d'intersection (Figure 21). Nous obtenons quatre groupes de faisceaux. Ensuite pour chacun de ces groupes, nous calculons un faisceau englobant. Enfin nous répétons le procédé pour chacun des sous plans jusqu'à obtenir des portions de plan image ne contenant qu'un faisceau.

L'algorithme de prétraitement du *Progressive Volume Photon Mapping* peut alors s'écrire ainsi :

Algorithm 10 Phase_de_pretraitement()

```
1: for all pixels du plan image do
2:    if volume visible depuis le pixel then
3:       etiqueter le pixel
4:    end if
5: end for
6: organiser le plan image sous forme d'arbre quad-tree
7: // les feuilles de l'arbre ne contiennent qu'un seul pixel etiquete
8: for all feuilles de l'arbre do
9:    creer_faisceau()
10: end for
11: for all noeuds de l'arbre do
12:    creer_faisceau_englobant()
13: end for
```



(a) Intersection entre les faisceaux et le plan image (b) Premier découpage avec faisceaux englobants (c) Second découpage de l'un des sous arbres

FIGURE 21 – Intersection entre les faisceaux et le plan image

Nous pouvons voir un exemple de découpage sur la Figure 21. Sur la première image, nous pouvons voir la grille de pixel et le contour de l'intersection avec les faisceaux englobants en rouge. Sur la seconde, nous pouvons voir les découpages du plan image en 4 parties, avec pour trois d'entre elles les faisceaux englobants en vert. Aucun faisceau englobant n'a été calculé pour la partie située au sud ouest, puisqu'elle ne contient qu'un seul faisceau. Enfin, sur la dernière image nous pouvons voir le second et dernier découpage de la zone nord ouest, chaque sous zone contenant au maximum un faisceau, il ne sera pas nécessaire de découper davantage.

Au final, la racine de l'arbre représente le plan image entier. Chaque noeud représente une portion du plan image, et leurs quatre fils représentent les subdivisions de ces portions. De plus, il existe un faisceau englobant associé à chacun des noeuds. Ce faisceau englobe tous les faisceaux des fils du noeud associé. Enfin, chacune des

feuilles une portion du plan image est aussi petite qu'un pixel, ce qui revient à représenter un faisceau utile.

6.2 faisceaux englobants

Pour créer un faisceau englobant au niveau d'un noeud de l'arbre, nous considérons uniquement les faisceaux associés aux fils directs (au maximum quatre). La direction du faisceau est alors simplement la moyenne des faisceaux fils.

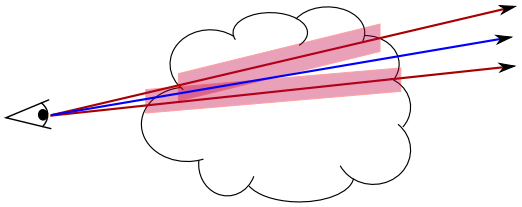


FIGURE 22 – En rouge les directions des faisceaux, en bleu la direction du faisceau englobant. Cette direction est la moyenne des directions des faisceaux.

Pour calculer le rayon, nous devons partir du fait que tous les faisceaux ont pour origine la caméra. Nous pouvons alors en déduire que le point d'un faisceau f_i le plus éloigné du faisceau f_g qui l'englobe est le point où f_i quitte le volume. Connaissant alors cette plus grande distance séparant un faisceau fils du faisceau englobant, nous pouvons alors calculer un rayon englobant tous les fils. Ce rayon est égal à la distance maximale entre les points de sortie du volume des faisceaux fils et le faisceau englobant, auquel on ajoute le rayon maximum des faisceaux fils. Ce noeud englobera aussi récursivement toutes les feuilles du sous arbre dont il est la racine.

Le faisceau englobant présente cependant deux défauts. Le premier est que dans le cas de faisceaux très longs, nous aurons des faisceaux englobants au diamètre énorme. Il sera alors courant d'avoir des photons à l'intérieur d'un faisceau englobant sans qu'il soit à l'intérieur d'un faisceau englobé. La méthode n'est donc pas optimale.

Le second défaut se trouve à l'extrémité du faisceau. De par sa construction, il existe une zone de la scène où les photons seront ignorés. Cependant, comme nous diminuons le diamètre du faisceau au fur et à mesure des itérations, cette zone diminue au cours du temps. Nous devrions donc converger vers une image correcte.

Cette structure accélératrice est implémentée et est en cours de mise au point. Il nous reste encore des tests à effectuer pour générer des résultats probants.

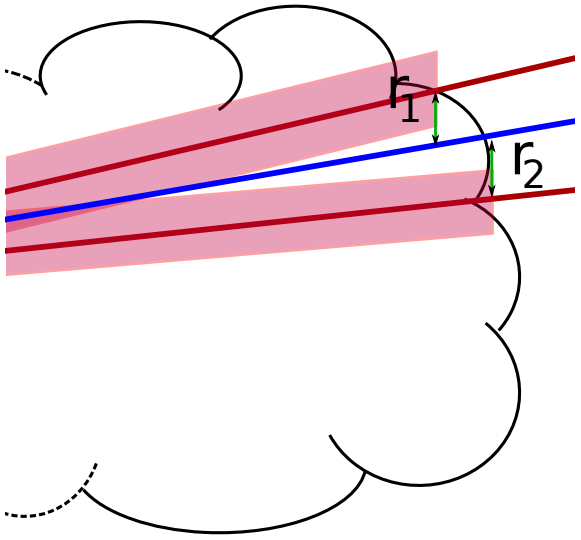


FIGURE 23 – Pour calculer le rayon du faisceau englobant de la figure 22 nous considérons les distances r_1 et r_2 des faisceaux englobés au faisceau englobant. Seul le maximum des deux sera utilisé.

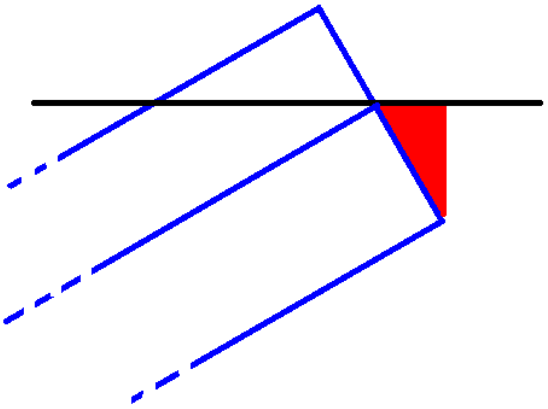


FIGURE 24 – Erreur engendrée par les faisceaux. En noir la surface et en bleu l'extrémité du faisceau. Les photons arrivant dans la zone rouge seront ignorés à tort.

7 Conclusion

Le but de notre travail était de mettre au point une nouvelle méthode de rendu de milieux participatifs. Cette méthode devait notamment améliorer la technique du *Volume Photon Mapping* sur deux points : diminuer le temps calcul et l'occupation mémoire du rendu.

Pour cela nous sommes partis d'un moteur de rendu existant (PBRT) où nous avons implémenté plusieurs méthodes de rendu de milieux participatifs :

- le *Volume Photon Mapping* ;
- le *Beam Radiance Estimate*.

L'implémentation de ces méthodes nous a aussi amené à implémenter une méthode de ray marching.

Ensuite, nous avons adapté une méthode de rendu surfacique, le *Progressive Photon Mapping* aux milieux participatifs. Il nous a fallu repenser le *Volume Photon Mapping*, afin d'adapter les points visibles sur les surfaces en faisceaux visibles dans les volumes. Cela nous a permis de proposer une méthode, le *Progressive Volume Photon Mapping*, assurant une utilisation constante et limitée de la mémoire. Elle est basée sur un système de faisceaux mis à jour au fur et à mesure du rendu par les photons traversant la scène. Nous pouvons alors utiliser un nombre théoriquement illimité de photons pour effectuer le rendu d'une scène, cela grâce à une utilisation quasi immédiate des photons lancés dans la scène. Dans notre méthode, il n'est alors plus nécessaire de conserver d'énormes cartes de photons en mémoire, d'où un gain important sur l'occupation mémoire.

Cependant cela ne remplit que la moitié des objectifs, le rendu restant très long. Comme la partie la plus longue du rendu est la recherche de faisceaux modifiés par un photon, nous avons eu l'idée d'accelérer le rendu en organisant nos faisceaux sous forme de hiérarchie. Dans ce but, nous avons trouvé un critère permettant de regrouper nos faisceaux spatialement et de les organiser sous forme d'arbre de recherche. Cette hiérarchie permettra d'accelérer sensiblement la recherche de faisceau, mais n'est pas encore entièrement fonctionnelle. Pour poursuivre le travail, il nous faudra donc résoudre ce problème.

Enfin, une fois la mise au point terminée, nous pourrons réfléchir à un portage sur GPU afin de paralléliser l'envoi de photons et les différentes phases de rendu et donc d'accélérer sensiblement la méthode.

Références

[HOJ08] Toshiya Hachisuka, Shinji Ogaki, and Henrik Wann Jensen. Progressive photon mapping. *ACM Trans. Graph.*, 27 :130 :1–130 :8, December 2008.

[JDZJ08] Wojciech Jarosz, Craig Donner, Matthias Zwicker, and Henrik Wann Jensen. Radiance caching for participating media. *ACM Trans. Graph.*, 27 :7 :1–7 :11, March 2008.

[Jen09] Henrik Wann Jensen. *Realistic Image Synthesis Using Photon Mapping*. A. K. Peters, Ltd., Natick, MA, USA, 2009.

[JZJ08] Wojciech Jarosz, Matthias Zwicker, and Henrik Wann Jensen. The Beam Radiance Estimate for Volumetric Photon Mapping. *Computer Graphics Forum (Proc. Eurographics EG'08)*, (2) :557–566, 4 2008.

[PH04] Matt Phar and Greg Humphreys. Physically based rendering from theory to implementation. *Morgan Kaufmann*, December 2004.

[Rib10] Mickael Ribardiere. Simulateur pour l'étude de la visibilité dans les environnements enfumés. December 2010.