Vis Comput
DOI 10.1007/s00371-013-0845-x

ORIGINAL ARTICLE

Visibility-driven progressive volume photon tracing

Charly Collin · Mickaël Ribardière · Adrien Gruson · Rémi Cozot · Sumanta Pattanaik · Kadi Bouatouch

Abstract In this paper, we present a novel approach to progressive photon-based volume rendering techniques. By making use of two Kd-trees (built in a preprocessing step) to store view beams (primary rays intersecting the medium) and visible points, our method allows to handle scenes with specular and refractive objects as well as homogeneous and heterogeneous participating media and does not require the storage of photon maps, which solves the memory management issue. These data structures are used to drive the photon shooting process by considering the photon visibility as an importance function (similarly to Hachisuka and Jensen in ACM Trans. Graph. 30(5):114:1–114:11, 2011) for scenes containing participating media. Finally, we demonstrate that our method can be easily combined with the most recent particle tracing approaches such as the one presented in Jarosz et al. (ACM Trans. Graph., vol. 30(6), 2011).

Keywords Rendering · Global illumination · Participating media · Markov chain Monte Carlo

C. Collin · S. Pattanaik
University of Central Florida, Orlando, USA

C. Collin
e-mail: charly.collin@bobbyblues.com

S. Pattanaik
e-mail: sumant@cs.ucf.edu

M. Ribardière (✉) · A. Gruson · R. Cozot · K. Bouatouch
IRISA, Université de Rennes 1, Rennes, France
e-mail: mickael.ribardiere@irisa.fr

A. Gruson
e-mail: adrien.gruson@irisa.fr

R. Cozot
e-mail: remi.cozot@irisa.fr

K. Bouatouch
e-mail: kadi.bouatouch@irisa.fr

Published online: 01 June 2013

1 Introduction

Rendering participating media (volume data) with multiple scattering in reasonable time still is a challenge. Indeed, computing light transport within such media is very demanding in terms of computing time and memory storage. Accurate and unbiased methods exist such as path tracing or other Monte Carlo based approaches. However, these methods get limited practically even for participative media of moderate complexity.

Fortunately, less precise but faster methods have been proposed in the last two decades, such as the photon mapping technique [8]. This latter has been first applied to surfaces then extended to participating media, and is called *Volume Photon Mapping* [9]. This method consists in emitting photons from light sources. These photons bounce off surfaces and interact with participating media in the scene. They get registered at each interaction with surfaces and participating media, creating records stored in a photon map. This photon map is used by a rendering step.

However, to compute the radiance at a visible point, an expensive final gathering or ray marching step is needed, depending on whether the point is on a surface or inside a volume. The rendering step starts only once all the photons have been emitted from the light sources. If the resulting image is not satisfactory, it is not possible to send additional photons to improve it. Consequently, to get a more accurate image a larger set of photons has to be emitted from scratch without exploiting all previous computations.

To cope with this problem, a method called *progressive photon mapping* [3] has been proposed. It allows to improve a resulting image by sending more and more additional photons. For each pass, a new set of photons is emitted and the resulting image is updated. Therefore, a preview is available after each iteration. This approach has since been extended

to volume rendering with techniques such as the one presented in [11].

The goal of this paper is to propose a novel approach to *progressive photon mapping* for scenes containing surface and volume objects. Our method allows to handle scenes with specular and refractive objects as well as homogeneous and heterogeneous participating media. It does not need to store photons in photon maps. During the shooting process, the visibility information allows knowing if a photon, hitting a surface or a volume, contributes to the final image or not. This visibility information is then used as a target sampling function to guide the photon shooting process so as to generate more contributive photon paths as in [2].

The main contributions of this paper are:

- the shot photons are not stored, they are discarded once they have contributed to visible points and beams, consequently the number of photons per pass is not limited;
- preprocessing is performed after every N passes, which allows removing aliasing artifacts efficiently;
- light paths are guided by visibility information for surfaces and participating media using a Metropolis strategy, which allows to handle complex lighting conditions such as environment maps, lighting through a double-glazed window, etc.

The next section provides information about the related works while Sect. 3 gives some technical background. An overview of our approach is given in Sect. 4, while details on our method are presented in Sect. 5. Finally, results are given in Sect. 6 before concluding.

2 Related work

The first method based on photon mapping and coping with participating media is *Volume Photon Mapping* [9]. It allows to simulate interactions between photons and participating media and to build a volume photon map. However, its costly rendering step consists in casting a ray from the camera through the volume and in using ray marching to gather the photons (stored in the photon map) close to the ray to compute their contributions. Volume photon mapping yields good results but it suffers from the same limitations as those of photon mapping: A huge number of photons is needed to render complex scenes. As a large number of photons requires a lot of memory, the rendering quality is limited by the available memory of the used computer. Moreover, ray marching, needed to retrieve nearby photons, is very expensive. The smaller the ray marching steps, the better the results, but also the longer the rendering. So *Volume Photon Mapping* has two limitations: high computation time and large memory requirement.

The *Beam Radiance Estimate* [7] method is one way to reduce the rendering time. In this method, each photon has an influence area, which is a disk of variable radius. An isolated photon is assigned a large radius while a smaller one is assigned to photons that are close to each other. To estimate the radiance of a ray cast from the camera, all the photons, within a distance from the ray that is smaller than their radius, are considered. This method speeds up the rendering process and also adapts to the density distribution within the volume and to the lighting conditions. When rendering scenes with sparse photon distribution, the *Beam Radiance Estimate* gives better results than the usual ray marching but remains costly when used for complex scenes. Indeed, the contribution of a ray needs the traversal of a hierarchy of spheres which takes time for a high number of photons.

Progressive Photon Mapping [3] (PPM) addresses the memory limitation problem. It is a progressive rendering method consisting in computing a noisy image that is improved after each pass. At each pass, only a limited number of photons are emitted then discarded when running the next pass. Therefore, only the photons emitted at the current pass are saved in memory. To each visible point is assigned a disk, and only the photons within this disk are used to compute the radiance at this point. The radius of a disk decreases after each pass to reduce the bias of this method. *A Probabilistic Approach* [11] (APA) of photon mapping is also a progressive method that handles scenes containing participating media. It allows to render many noisy images of a same scene that are merged to get a final image.

Another interesting approach, called the *Progressive Photon Beams* [6] (PPB), is close to ours but proceeds differently. It consists in shooting photon beams (instead of photons) from the light sources and storing them in a BVH data structure. For radiance estimation, the visible photon beams are decomposed into those that are handled using GPU through rasterization, and those handled by the CPU ray tracer.

Although PPB requires less memory than a classical photon-based method, its complexity increases with the number of light sources. Indeed, in case of many area light sources the budget of photon beams generated from one light source would be limited due to the memory constraint, which slows down the convergence to a good quality image. As it depends only on the image resolution (as will be explained later on), our approach overcomes this limitation. Note that PPB can be easily combined with our approach as shown in Sect. 6.

Another volume rendering method [13] relies on VPLs (Virtual Point Light) [10]. The method considers the light paths as VRLs (Virtual Ray Lights) rather than photon beams. Some improvements of this method have been proposed in [12]. This is an interesting method that proceeds differently from our approach.

Discussion While the above methods provide good results, they still suffer from a few issues or proceed differently from

our approach. Indeed, the more complex the scene, the bigger the memory budget for each pass (whether for photons or beams) and the higher the number of passes (iterations). Also, in [6, 11], a new hierarchy must be computed from scratch at each pass, hence adding a nonnegligible computational cost. Instead, our approach builds one beam hierarchy (beam Kd-tree) every N passes of the progressive process to avoid aliasing artifacts.

Moreover, storing view beams in a Kd-tree allows to extend to participating media the robust adaptive photon tracing approach based on Metropolis strategy and proposed by Hachisuka and Jensen [2]. These authors employ the photon path visibility as a target function to better sample the path space. They also use adaptive Markov Chain Monte Carlo to determine the best mutations strategies. In addition, their approach relies on the replica Exchange Monte Carlo to avoid the path sampling getting stuck at local peaks.

3 Background

The radiance at a point inside a participating medium is computed by solving the radiative transfer equation (RTE) [1]. Given a ray starting at a point x and going through the medium until a point x_s, the RTE provides the radiance at x in direction $\boldsymbol{\omega}$ as follows:

$$L(x,\boldsymbol{\omega}) = \int_0^s \mathrm{Tr}\big(x \leftrightarrow x'\big)\sigma_s\big(x'\big)L_i\big(x',\boldsymbol{\omega}\big)\,dx' + \mathrm{Tr}(x \leftrightarrow x_s)L(x_s,\boldsymbol{\omega}), \quad (1)$$

where s is the distance traversed by the ray within the volume, $L_i(x',\boldsymbol{\omega})$ the incoming radiance at x' and scattered in the direction $\boldsymbol{\omega}$. Other terms are defined in Table 1. For the sake of clarity, Eq. (1) does not take into account the radiance self-emitted by the medium, which is straightforward to compute.

The first term in Eq. (1) expresses the radiance L_m gathered along the ray through the medium. For a set of volume photons, L_m can be expressed as in *Beam Radiance Estimate* method [7]:

$$L_m \approx \frac{1}{N}\sum_{i=0}^{N} K_i(x_i,r_i)\mathrm{Tr}\big(x \leftrightarrow x_i'\big)\sigma_s\big(x_i'\big)\,p(x_i,\boldsymbol{\omega},\boldsymbol{\omega}_i)\Phi_i, \quad (2)$$

where K_i is a kernel function, x_i the photon position, and x_i' the projection of the photon position x_i on the view ray.

The second term is the radiance L_s of a visible point on the surface seen through the medium along the view ray. This radiance is computed using density estimation as in classical *Progressive Photon Mapping* [3] and is attenuated by the medium:

$$L_s \approx \frac{1}{N}\sum_{i=0}^{N} \frac{f_r(x_i,\boldsymbol{\omega},\boldsymbol{\omega}_i)(\mathbf{n}\cdot\boldsymbol{\omega}_i)\Phi_i}{\pi R_s^2} \times \mathrm{Tr}(x \leftrightarrow x_s) \quad (3)$$

The total radiance for a view ray is $L = L_m + L_s$.

Table 1 Definition of quantities used in this article

Symbol	Description
Φ_i	Flux of photon i
σ_a	Absorption coefficient
σ_s	Scattering coefficient
σ_t	Extinction coefficient
$\mathrm{Tr}(x \leftrightarrow x')$	Transmittance between the points x and x' $\mathrm{Tr}(x \leftrightarrow x') = e^{-\int_0^{\|x-x'\|}\sigma_t(x+t\boldsymbol{\omega})dt}$
$p(x,\boldsymbol{\omega},\boldsymbol{\omega}')$	Normalized phase function
$fr(x,\boldsymbol{\omega},\boldsymbol{\omega}')$	BRDF
L_s, L_m	Radiance at a surface visible point or in a medium beam respectively
L_s^t, L_m^t	Cumulative radiance at a visible point or a beam at the current iteration only
N_s, N_m	Number of photons within a disk or in a view beam since the beginning of the rendering
N_s^t, N_m^t	Number of photons within a disk or in a view beam in the current iteration only
R_s, R_m	Radius of a disk associated with a visible point or radius of a beam
α	User parameter to control the convergence rate

Algorithm 1 main()

```
for i = 1 to nbPass do
    if i%N==0 then
        Jitter-View-Rays(); // Jittering of view rays
        Preprocess(); // Done every N passes
    end if
    for j = 1 to nbPhoton do
        SendPhoton();
        UpdateCurrentRadiance();
    end for
    Render();
    UpdateImage();
    UpdateRadii();
end for
```

4 Overview

Our method is depicted by the global Algorithm 1.

When rendering surfaces, the progressive photon mapping method assigns a disk to every visible point lying on a surface. For volumes, we use beams rather than disks. To each ray of a ray path, cast through a pixel and traversing a medium, we assign a cylinder whose radius gets smaller after each pass. This cylinder is called a beam from now on. These beams get reflected/refracted if the associated ray hits a specular or refracting surface. We build a beam Kd-tree whose leaves are the resulting beams. When a ray hits a diffuse or a glossy surface, directly or indirectly through

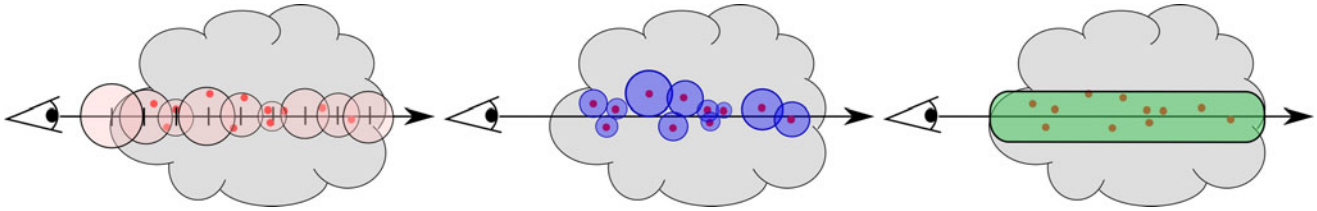


Fig. 1 Different ways to gather photons. Regular ray marching (**a**) may take into account several time a same photon whereas *Beam Radiance Estimate* (**b**) and our method (**c**) gathers all photons once and for all

specular reflection or refraction, the resulting hit point is registered and assigned a disk-shaped influence zone. The radius of this disk gets smaller after each pass. We build another Kd-tree, called view Kd-Tree, whose leaves are those hit points. We also precompute a 2D array storing ray paths traced from the viewpoint through each pixel, as explained later. To make our method computationally efficient, we decided to perform this preprocessing every N passes (function `Preprocess` detailed in Sect. 5.1). For every new N passes, we jitter (function `Jitter-View-Rays`) for each pixel its associated view ray then perform again the preprocessing. This allows to solve the aliasing artifacts with the help of oversampling.

At each pass, a set of photons are shot, using the `SendPhoton` function. Whenever one of those photons interacts with a volume (or a surface), the top-down traversal of the associated Kd-tree leads to beams (or visible points), which allows to compute the current radiance, due to the photon, of the hit points and the beams. Then the photon is discarded instead of stored in a photon map. Next, these current radiances are updated (`UpdateCurrentRadiance` function) to account for all the successive shot photons. Once all the photons have been shot, the image is computed using the `Render` function.

For more efficiency, every contributive light path is mutated using Metropolis, similarly to [2]. A light path is considered as contributive if one of its vertex contributes to visible points or beams. This increases the efficiency of the shooting process.

To speed up the traversal of the view Kd-tree, we use a same radius R_s for all disks. We also use a same radius R_m for all beams for the same reason. This slightly affects the efficiency of the method but the method keeps converging to an unbiased result as proved in [11]. Further details about these updates are given in Sect. 5.3.

Finally, at the end of each pass, the resulting image is updated using the `UpdateImage` function, and so are the radii (`UpdateRadii` function) of the disks and beams. These steps are described in Sects. 5.4 and 5.5.

In *Volume Photon Mapping*, once the photon map is built a ray is cast through each pixel and a costly ray marching (or a *Beam Radiance Estimate*) is used to determine the relevant photons used to compute the radiance associated with the ray. Rather, in our approach, as soon as a photon interacts with a medium it is assigned to one or more beams by going down the beam Kd-Tree, and its contribution to the scene is computed straightaway. This assignment operation spares the use of ray marching (Fig. 1). Note that, unlike other progressive volume photon mapping approaches, our method is not dependent on the number of photons shot at each pass, which allows to shoot as many photons as wanted before starting the first rendering operation.

On the one hand, the use of a beam Kd-tree makes the estimation of the radiance assigned to each beam faster. On the other hand, photon assignment requires the traversal of this hierarchy which takes time. Overall, our hierarchy-based approach is far faster than ray marching as shown in the results section.

5 Implementation details

In this section, only photons are shot from the light sources. We will show in Sect. 6 that our approach can also handle photon beams proposed in [6].

5.1 Preprocessing

Before starting any rendering, we need to initialize all the needed data structures. A view ray, traced from the viewpoint through a pixel (going through the participating media or not), is repeatedly reflected off specular surfaces and/or refracted. This results in a ray path. When a ray of a ray path crosses a medium, it is assigned a cylinder beam. Thus, a ray of a ray path is either a beam or a line segment the endpoints of which are intersection points with surfaces. This process results in a 2D array of ray paths. When computing the ray paths, the intersection points of the rays of a ray path with diffuse and glossy surfaces only, called visible points (points visible directly or indirectly from the viewpoint of the camera), are registered to compute a view Kd-tree as explained hereafter. If a ray of a ray path does not intersect any participating medium, it is assigned its endpoint (which is a hit

point), its current and cumulative radiances L_s and L_s^t. If a ray of the ray path intersects a medium, then it is assigned a beam generated after specular reflection and refraction (say a cylinder of a certain radius (Fig. 2)), the transmittance Tr of the beam, the endpoint (hitpoint) of the beam together with its current and cumulative radiances L_m and L_m^t.

Once the path tracing has been completed, we obtain a 2D array of ray paths, a set of beams and a set of visible points. These three data structures are computed in one single pass. We build a view Kd-tree from the visible points and a view-dependent beam Kd-tree from the beams. Each visible point is assigned a disk. A photon within this disk contributes to the associated visible point. A photon contributes to a beam if it lies in its associated cylinder. We store the view Kd-tree rather than a photon map, which makes our method independent of the number of light sources.

Recall that the radii associated with disks and beams decrease after each rendering pass. Note that the two Kd-trees do not depend on the values of these radii. The view Kd-tree is computed using a classical way, a disk being sorted according to its center.

The beam Kd-tree, however, is created following the method from Havran et al. [4], a beam being represented by a line segment supported by its associated ray (beam axis). The endpoints of this line segment are those of the beam (Fig. 3b). The first step of building the beam tree is to find the axis-aligned bounding box of all the beam segments, which represents the root of the tree. Then a splitting plane is chosen along the largest axis of that bounding box. All the beam segments on one side of that plane are assigned to one child, and the beams crossing this plane are assigned to both children. The splitting operation is recursively repeated until one of the stopping criteria is met (Fig. 3c):

- The number of beam segments in the current node is below a threshold (typically 32 in our implementation).
- The depth of the current node reaches a maximum value (between 20 and 30 for our scenes).

An internal node of the beam Kd-tree is a splitting plane while each leaf is a set of beams. Each beam segment is assigned a data structure, which contains a pointer to the associated beam computed in the preprocessing step. To compute the contribution of a photon to the beams, the beam Kd-tree is traversed top-down. For each node, if a photon is located on one side of the associated splitting plane and its orthogonal distance to this plane is larger a beam radius, then the associated subtree is traversed, otherwise we traverse the two subtrees. For each leaf node reached by the traversal, we compute the contribution of the photon to the beams corresponding to the beam segments within this leaf. A photon contributes to a beam of this leaf if its orthogonal distance to this beam is smaller than the beam radius.

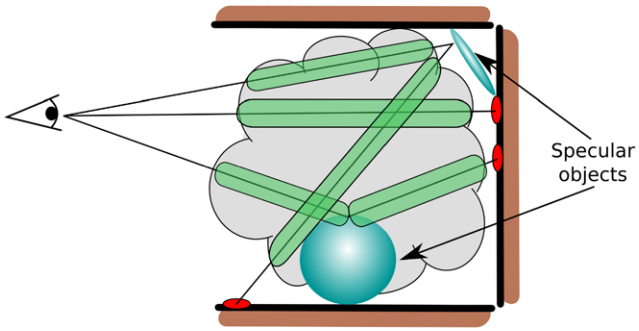


Fig. 2 Different possible view rays in a scene. The rays going through the medium have view beams (*in green*) associated, and visible points (*in red*) are created on diffuse surfaces

5.2 Visibility-driven photon shooting process

The photons are shot from the light sources. When a photon enters a participating medium, the beam Kd-tree is traversed top-down, and its contribution is brought to the leaf beams containing it. When it hits a surface, we compute its contribution to the visible points (nodes of the view Kd-tree whose disk contains the photon) by a top-down traversal of the view Kd-tree. In this way, as soon as a photon contributes to a visible point or a beam, it is discarded.

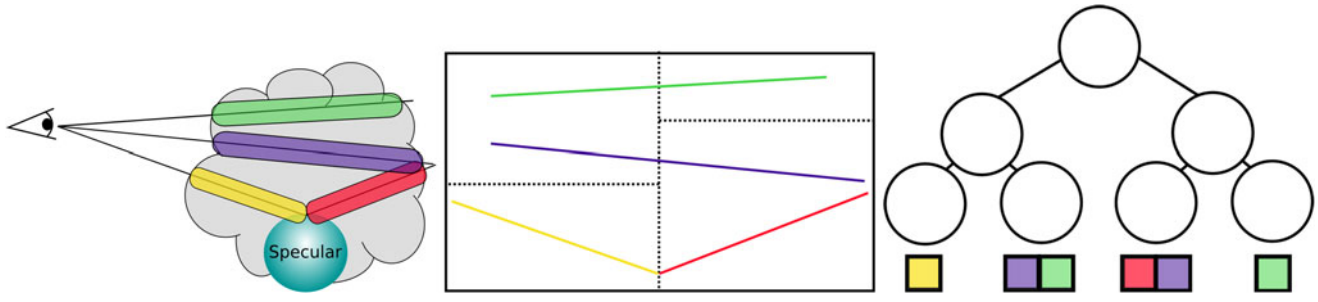


Fig. 3 Example of a view beam Kd tree building for a set of beams (**a**). Only the supporting rays are used in the tree (**b**). Some leaves (**c**) contains several beams, and beams crossing splitting planes are duplicated in the tree

To handle complex lighting, such as environment map lighting in highly occluded scenes (i.e., indoor scenes lit through windows), lighting through double glazed windows and lighting through participating media, we use a Metropolis-based approach [2] to guide the photon shooting process based on visibility information.

We use the same mutation strategy for surfaces and participating media, but with different parameters determined automatically according to the method presented in [2]. Unlike in [2], we use two different Markov chains; one for the surfaces and another for the participating media.

5.3 Radiance update

When a photon lies in a beam, it participates in the update of the cumulative radiance L_m^t of the beam using Eq. (2):

$$L_m^t += K(x_i, r_i)\mathrm{Tr}\big(x \leftrightarrow x_i'\big)\sigma_s\big(x_i'\big)\, p(x_i, \boldsymbol{\omega}, \boldsymbol{\omega}_i)\Phi_i \tag{4}$$

In case of heterogeneous medium, for a reason of efficiency each beam data structure stores a lookup table of the transmittance along the beam axis to quickly determine the transmittance $\mathrm{Tr}(x \leftrightarrow x')$.

When a photon lies in the disk associated with a visible point, the radiance of this latter is updated using Eq. (3):

$$L_s^t += \frac{fr(x_i, \boldsymbol{\omega}, \boldsymbol{\omega}_i)(\mathbf{n}\cdot\boldsymbol{\omega}_i)\Phi_i}{\pi R_s^2} \times \mathrm{Tr}(x \leftrightarrow x_s) \tag{5}$$

5.4 Image update

Once a sufficient number of photons have been shot for the current pass (iteration), the resulting image can be updated. This step consists in updating the radiances L_s of the visible points and the radiances L_m of the beams. It also updates the radius values.

The radiances are updated using the same local photon statistic as in *Progressive Photon Mapping* [3]:

$$L_s = \left(L_s + \frac{1}{N_s^t}L_s^t\right)\frac{N_s + N_s^t(1-\alpha)}{N_s} \tag{6}$$

This equation is valid for the update of beam radiance.

The radiance of a pixel brought by its associated ray path is

$$L = \mathrm{Tr}_{nbB}L_s + \sum_{i=1}^{nbB}\mathrm{Tr}_{i-1}L_{m,i}, \tag{7}$$

where $L_{m,i}$ is the radiance L_m of the ith beam of the ray path, Tr_i the transmittance associated with this beam ($\mathrm{Tr}_1 = 1$, the first beam starting from the viewpoint), nbB the number of successive beams of the ray path, and $\mathrm{Tr}_{nbB} = \prod_{i=1}^{nbB}\mathrm{Tr}_i$. Note that if $nbB = 0$ then $L = L_s$, which means that the ray path does not cross any participating medium.

5.5 Radius update

Finally, in order to achieve convergence, we need to update the radii as explained in [11]:

$$R_s = R_s\sqrt{\frac{n+\alpha}{n+1}}, \qquad R_m = R_m\sqrt[3]{\frac{n+\alpha}{n+1}}, \tag{8}$$

where n the number of the current iteration.

The contribution to the final image of one iteration is noisy. With only one iteration, we can get only a blurry and biased image, or a noisy and less biased one. Starting with large radii makes the first iterations blurry, but decreasing those radii after each iteration make the sharp details appear.

6 Results

In this section, we show some results obtained with our method using different shooting techniques (without and with metropolis optimization) and different kinds of particle such as photon or photon beams. In this way, we demonstrate that, although our method has been designed for photon tracing, it has been easily extended to photon beam tracing. We compare our method with PPB, which is the most related method, and APA which is another interesting approach, easy to implement since it is just a loop consisting in running a classical volume photon mapping. We use the following notations concerning different variants of our method:

PPT our approach where photons are traced from the light sources

PPBT our approach where photon beams are traced

PPT_metro our approach with photon tracing and Metropolis

PPBT_metro our approach with photon beam tracing and Metropolis

We have implemented our methods as well as APA and PPB using the Mitsuba renderer framework [5]. The different parameters used for each scene are given in Table 2 ($\alpha = 0.7$ for all the methods). The results have been obtained on a computer supplied with two 2.4 GHz Intel Xeon E5645 CPU (12 cores). Each method is executed in a multithreading context to use the 12 cores of the computer. For comparison, we have generated reference images using path tracing.

In the *dragon smokes* scene (Fig. 5), as the lighting conditions are quite simple the metropolis optimization is not really useful (except for the smoke which covers a small part of the scene). Indeed, the majority of the photon paths are visible so few of them need mutation. However, as shown in Fig. 4, our method (with different variants) performs as well as APA or PPB. In the two next scenes, *breakfast hall* and *kitchen* (Figs. 6 and 7), the lighting conditions are more

Table 2 Rendering parameters: each method is stopped at the same rendering time (*rightmost column*). *Volume Photons* = number of volume photons shot per pass in APA and PPT; *Photon Beams* = number of photons beams in PPB and PPBT

	# Polygons	Resolution	# Volume photons	# Photon beams	# Surface photons	Rendering time
Dragon smokes	100k	768×768	50k	5k	100k	1 h
Breakfast hall	1600k	1080×1920	100k	10k	100k	10 h
Kitchen	250k	720×1280	100k	10k	500k	10 h

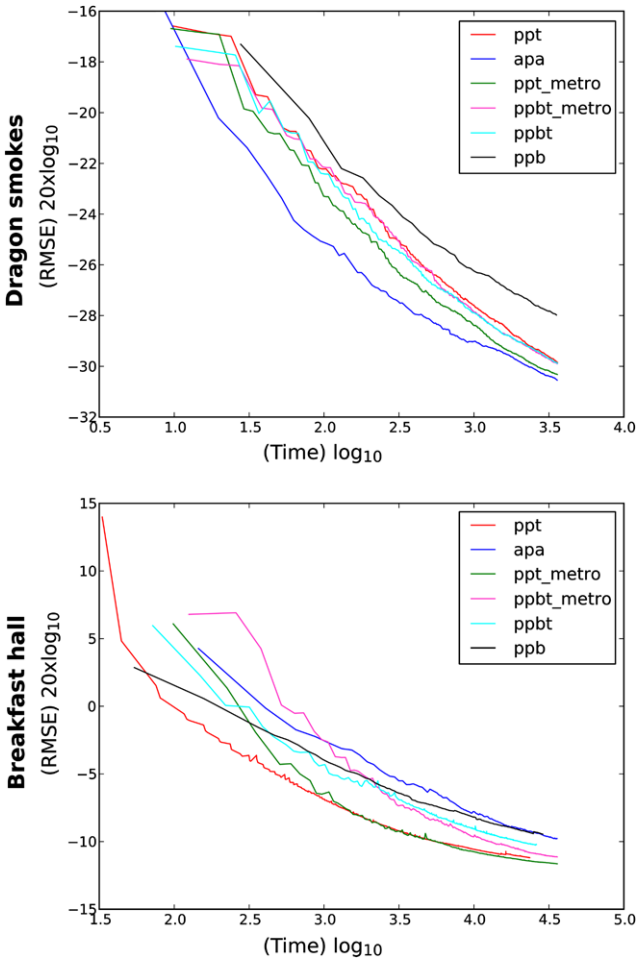


Fig. 4 Plots of the RMSE (between a reference image generated using path tracing and all the methods) as a function of time

complex. In the breakfast scene, light goes through the windows and the hall is filled with a homogeneous medium. The outside is represented by a horizontal large plane a small part of which is visible through the windows. This scene is challenging for Metropolis as well as for uniform sampling based methods. Indeed, for hitpoints lying outside, replica exchange builds a lot of useless paths that mutate outdoors while they are less contributive to the final image. This is why PPT without Metropolis performs as well as PPT_metro in the beginning of the progressive process (Fig. 4), but PPT_metro gets more and more efficient after a certain number of iterations. This can be explained as follows. First, after a certain number of iterations the burn-in period has elapsed. Second, the beam radius gets small, which makes difficult to find contributive paths. Thanks to mutations (enabling local exploration), Metropolis found contributive paths more efficiently.

As shown in this figure, our method PPT_metro converges faster than APA and PPB. The kitchen scene is lighted, through the double-glazed window, by a clear sky and the sunlight. The kitchen is filled with a heterogeneous medium. In Fig. 7, we show results obtained with different methods. We can see (close-up views) that Metropolis gives better visual results.

6.1 Discussion and future works

All the six presented methods suffer from the limitation of photon mapping. This limitation is a starting bias highly depending on the initial parameters (initial radii, number of photons, α, etc.)

Recall that our method guides the photon shooting process based on a Metropolis strategy that uses Adaptive Markov Chain Monte Carlo (AMCMC). Our experiments showed that AMCMC converges toward a small mutation size. This is why we initialize AMCMC with a mutation size smaller than the one proposed in [2] (0.1 gives good results). An interesting future work is to propose a better adaptive process to converge faster toward an optimal mutation size. Finally, using only visibility as an importance sampling function can fail in some cases. Indeed, visible photon paths may have a weak contribution to the final image (due to glossiness or transmittance in the volume). Finding a new importance sampling function, which takes into account these cases is a challenge.

7 Conclusion

We have proposed a global illumination method that handles scenes containing surface objects (diffuse, glossy, specular, and refractive) and participating media (homogeneous and heterogeneous). The method computes three data structures in a preprocessing step: view Kd-tree, view dependent beam Kd-tree and a 2D array of ray paths. No photon maps,

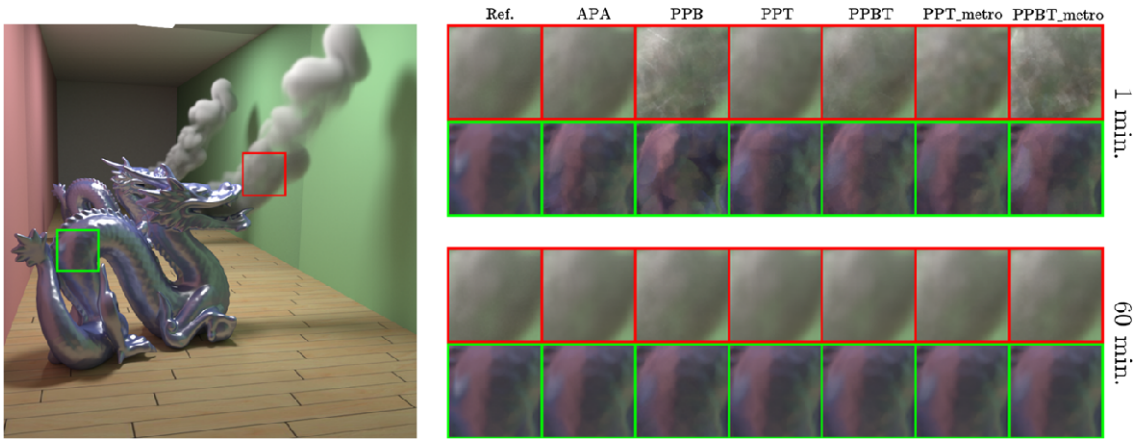


Fig. 5 Dragon smokes scene. Leftmost image is the reference image. Closup views for the different methods showing that all the methods converge to the reference solution

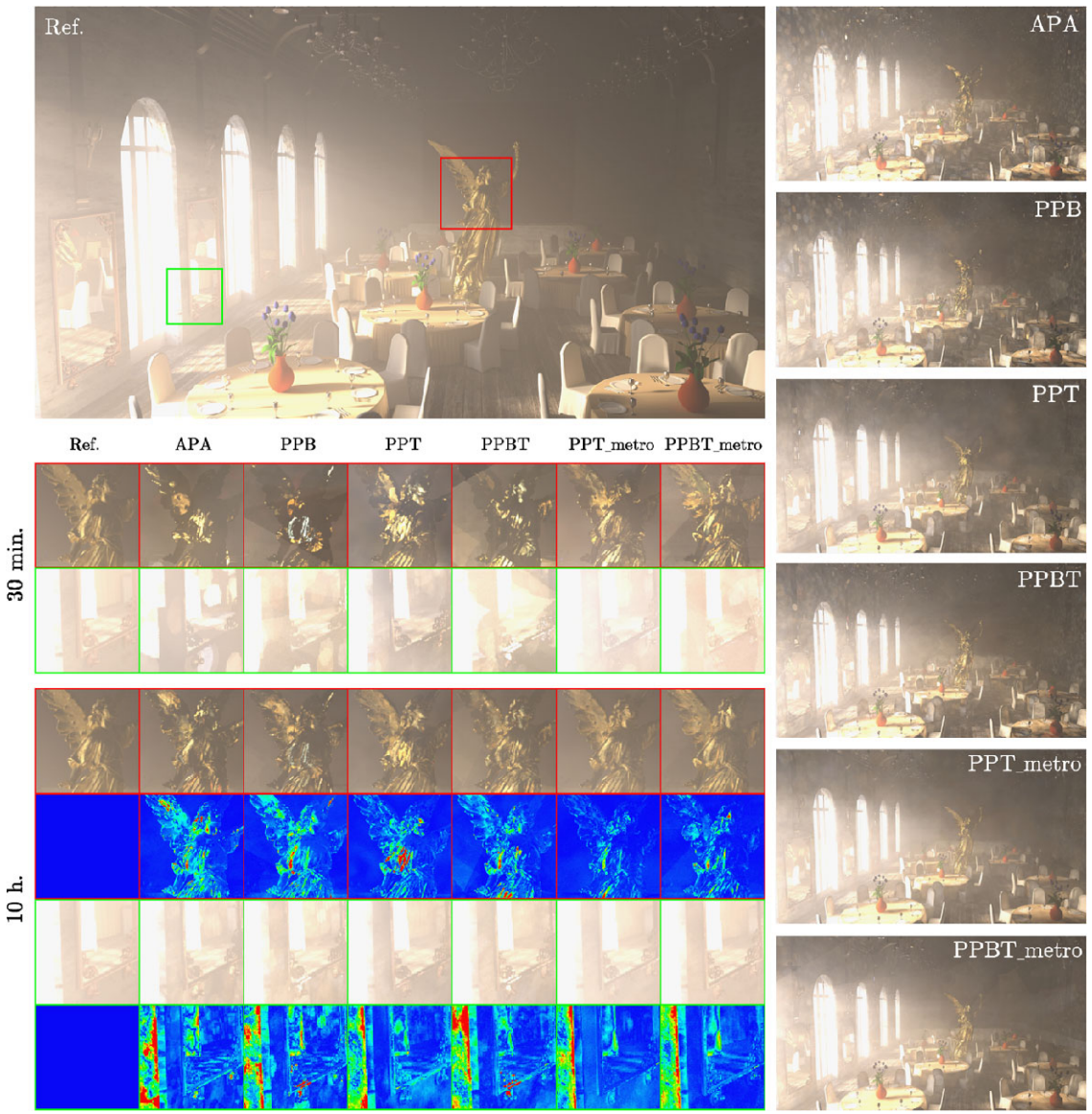


Fig. 6 Breakfast hall scene (courtesy of Greg Zaal). *Top left image*: reference image. *Rightmost column*: images obtained for the different methods after 10 h. Closup views for different methods and for 30 min and 1 h

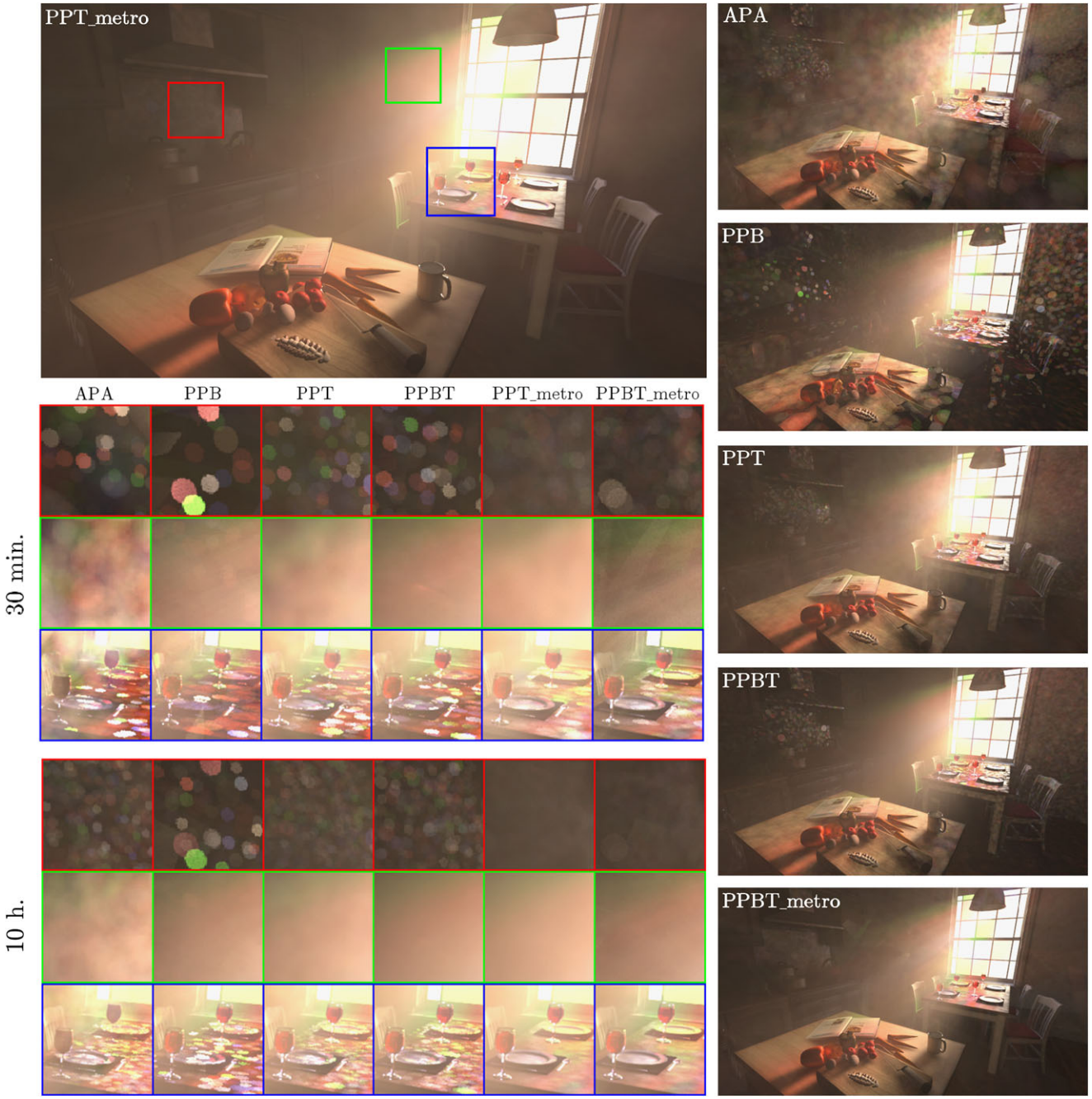


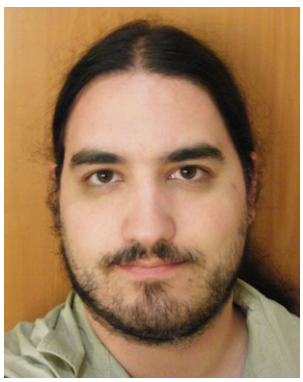
Fig. 7 Kitchen scene (courtesy of Jay-Artist). *Top left image*: result obtained with *PPT_metro*. *Rightmost column*: images obtained for the different methods after 10 h. Closup views for different methods and for 30 min and 1 h

nor volume photon maps are stored in memory. Moreover, the beams that are not visible from the viewpoint are not computed. Consequently, our method is not constrained by a beam budget due to memory limitation, which makes sending a large number of photons in one pass of the progressive process possible. To increase efficiency, our method uses Metropolis and visibility information to guide the photon shooting process. The results have shown that our approach converges faster to a same solution (image of a given RMSE) than APA [11] and the CPU version of PPB [6]. Even though our method has been implemented on the CPU, the obtained results demonstrate that our approach is fast. Another interesting feature of our method is that it could be easily transformed from progressive to interactive. More precisely, the user can set the number of photons to a very high value (this is possible since the photons are not stored), launches the program, interrupts it whenever he wants to display a resulting image, then restarts it to get a better image, and so on.

Acknowledgements Charly Collin has been supported in part by NSF grant IIS-1064427. Thanks go to the reviewers for their valuable reviews.

References

1. Chandrasekhar, S.: Radiative Transfer. Dover, New York (1960)
2. Hachisuka, T., Jensen, H.W.: Robust adaptive photon tracing using photon path visibility. ACM Trans. Graph. **30**(5), 114:1–114:11 (2011)
3. Hachisuka, T., Ogaki, S., Jensen, H.W.: Progressive photon mapping. ACM Trans. Graph. **27**, 130:1–130:8 (2008)
4. Havran, V., Bittner, J., Seidel, H.P.: Ray maps for global illumination. In: ACM SIGGRAPH 2004 Sketches, SIGGRAPH'04, p. 77. ACM, New York (2004)
5. Jakob, W.: Mitsuba renderer (2010). http://www.mitsuba-renderer.org
6. Jarosz, W., Nowrouzezahrai, D., Thomas, R., Sloan, P.P., Zwicker, M.: Progressive photon beams. In: Proceedings of ACM SIGGRAPH Asia. ACM Trans. Grap., vol. 30(6) (2011)
7. Jarosz, W., Zwicker, M., Jensen, H.W.: The beam radiance estimate for volumetric photon mapping. Comput. Graph. Forum **27**(2), 557–566 (2008) (Proceedings of Eurographics 2008)
8. Jensen, H.W.: Realistic Image Synthesis Using Photon Mapping. Peters, Natick (2001)
9. Jensen, H.W., Christensen, P.H.: Efficient simulation of light transport in sciences with participating media using photon maps. In: Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH'98, pp. 311–320. ACM, New York (1998)
10. Keller, A.: Instant radiosity. In: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH'97, pp. 49–56. ACM Press/Addison-Wesley, New York (1997)
11. Knaus, C., Zwicker, M.: Progressive photon mapping: a probabilistic approach. ACM Trans. Graph. **30**, 25:1–25:13 (2011)
12. Novák, J., Nowrouzezahrai, D., Dachsbacher, C., Jarosz, W.: Progressive virtual beam lights. In Proceedings of EGSR 2012. Comput. Graph. Forum, vol. 31(4) (2012)
13. Novák, J., Nowrouzezahrai, D., Dachsbacher, C., Jarosz, W.: Virtual ray lights for rendering scenes with participating media. In: Proceedings of ACM SIGGRAPH 2012. ACM Trans. Graph., vol. 31(4) (2012)



Charly Collin is a PhD student and research assistant in the Department of Electrical Engineering and Computer Science at University of Central Florida. His research interests are volume rendering, BRDF computation and rendering of polarization effects. He received his MS degree from the University of Rennes 1 (France) in 2011.

Mickaël Ribardière received his Master's degree in computer science in 2007 from the University of Limoges, and a PhD degree in 2010 from the University of Rennes 1. He is currently Research Scientist in the FRVSense team at IRISA (Institut de Recherche en Informatique et Systèmes Aléatoires). His research interests are global illumination, lighting simulation for complex environment and computer vision.

Adrien Gruson graduated from the Unitersity of Rennes 1, majoring in Digital Imagery. For his last year internship, he went to work for 6 months at UCF (University of Central Florida) under Sumanta Pattanaik's supervision on the topic of tone mapping and white balancing for walkthrough. After this internship, he received a research grant (MESR) to work on his PhD thesis: Interactive global illumination in large environments, and joined the research team FRVSense, under the supervision of Rémi Cozot and Kadi Bouatouch.

Rémi Cozot is an assistant professor at the University of Rennes 1 and a member of the FRVSense research team at IRISA. His research interests include enhancement of real-time rendering using human vision features and color and light adaptation. He is also involved in global illumination. Cozot received his PhD in computer science from the University of Rennes 1.

Sumanta Pattanaik is an associate professor of Computer Science in the Department of Electrical Engineering and Computer Science at University of Central Florida. His research interests include realistic image synthesis and realistic real-time rendering. Pattanaik has a PhD in Computer Science from Birla Institute of Technology and Science (BITS), Pilani. He is a member of IEEE and ACM. Contact him at sumant@cs.ucf.edu.

Kadi Bouatouch was an electronics and automatic systems engineer (ENSEM 1974). He was awarded a PhD in 1977 and a higher doctorate in computer science in the field of computer graphics in 1989. He is working on global illumination, lighting simulation for complex environments, GPU based rendering and computer vision. He is currently Professor at the University of Rennes 1 (France) and researcher at IRISA (Institut de Recherche en Informatique et Systèmes Aléatoires). He is member of Eurographics.