# Double Hierarchies for Directional Importance Sampling in Monte Carlo Rendering

Norbert Bus Tamy Boubekeur
LTCI, Télécom ParisTech
Paris-Saclay University, France

## Abstract

We describe a novel representation of the light field tailored to improve importance sampling for Monte Carlo rendering. The domain of the light field i.e., the product space of spatial positions and directions is hierarchically subdivided into subsets on which local models characterize the light transport. The data structure, that is based on *double trees*, approximates the exact light field and enables very efficient queries for importance sampling and easy tracing of photons in the scene. The framework is simple yet flexible, enabling the usage of any type of local model for representing the light field, provided it can be efficiently importance sampled. The method also supports progressive refinement with an arbitrary number of photons. We provide a reference open source implementation.

## 1. Introduction

The efficient solution of the rendering equation for photo-realistic image generation from a 3D scene model is a long-standing problem in computer graphics, requiring efficient algorithms that are able to produce good approximation in the limited amount of time dictated by the application scenario. Some of the most successful methods for rendering photo-realistic images are based on Monte Carlo integration e.g., *bidirectional path tracing* [Veach and Guibas 1995a]. In order to improve the efficiency of these methods, it is necessary to sample the function that is integrated according to a distribution that is as similar as possible to the function itself.

An ideal distribution would be proportional to the integrand. Several methods have been proposed, all of which construct different distributions that attempt to approximate the integrand with efficient estimators. The simplest strategy samples the BRDF or direct light sources, using *multiple importance sampling* [Veach and Guibas 1995b], providing an optimal mechanism to combine these estimators. Jensen [1995] and recently Vorba et al. [2014] proposed importance sampling distribution models

 ISSN 2331-7418

based on the incoming light, while accounting for the indirect illumination. The latter is considered the current state-of-the-art method, but it models distributions using Gaussian mixture models (or GMM) that require lengthy preprocessing.

Based on the same framework, we propose a novel representation of the light field with the key property that it enables efficient importance sampling for Monte Carlo methods without lengthy preprocessing. Moreover, our approach is very simple to implement and enables progressive refinement, i.e., training with an arbitrary number of photons. At a high level, we model the light field—the function mapping radiance values to points in the product space of spatial positions and directions—as a collection of simple functions (or *local models*). More precisely, we cluster the product space and, for each cluster, create a local model that approximates the light field locally, generating them by tracing photons in a preprocessing phase prior to rendering. This representation removes redundant information by assigning distributions to larger areas and improves the approximation by subdividing the space into smaller subspaces, therefore enabling models to capture local phenomena more precisely.

## 2. Previous Work

### Importance Sampling for Incoming Light

A recent approach for representing incoming light in the importance distribution was proposed by Vorba et al. [2014]. This method starts by creating a dense sampling of local distributions by associating a hemispherical distribution with points in the scene. Then, during sampling, the closest associated distribution is queried with respect to the rendered point and used to sample the incoming light. The underlying distribution model uses GMMs, but this representation can be replaced by any other model that can be efficiently sampled, such as histograms [Jensen 1995] or sets of cones [Hey and Purgathofer 2002], for instance. As this method only models the incoming light, it must eventually be combined with BRDF sampling techniques using multiple importance sampling. Alternatively, Hua and Lowe [2015] aim at importance sampling of incoming light using Metropolis sampling on a dense set of virtual point lights [Keller 1997] (or VPLs) in the scene. Although numerous methods have been proposed to represent the incoming light from environment maps and BRDFs [Clarberg et al. 2005; Jarosz et al. 2009], such strategies have limited applicability in our case as they cannot capture indirect lighting.

### Representing the Light Field

Ren et al. [2013] achieve real-time global illumination using a similar light-field domain subdivision strategy and training neural-network models that approximate the light field in each subspace. Our method shares this idea of training local models, but

it additionally enables efficient sample generation by exploiting *double tree hierarchies* [Bus et al. 2015] to compute illumination with VPLs. Our proposed technique develops this data structure further to enable efficient sampling and training with photons.

## 3. Method

Let us first define the notation used in this paper. We seek to solve the rendering equation [Kajiya 1986] (Equation (1)) for a point $x \in \mathbb{R}^3$ and direction $\omega \in \Omega_x$ where $n_x$ is the surface normal and $\Omega_x$ is the hemisphere of directions at $x$, $f$ is the BRDF function, and $L$ is the luminance (with $L_e$ being the emitted radiance at $x$):

$$L(x,\omega) = L_e(x,\omega) + \int_{\Omega_x} f(x,\omega_i,\omega) L(x,\omega_i) \cos(n_x \cdot \omega_i) d\omega_i. \tag{1}$$

Let $C \subset \mathbb{R}^3$ be a dense subset of points on the surfaces of a scene. The method of [Vorba et al. 2014] creates a local representation of the incoming light at each of the points in $C$. Assuming that $D$ is the space of the hemispherical distributions, this can be written as a mapping $m : C \to D$. Typically, $D$ is restricted to a special family of distributions, e.g., histograms (piece-wise constant) or Gaussian mixture models (GMM). In the latter case, $m(p) = GMM(\lambda(p))$ where $p \in C$ and $\lambda$ denotes the parameters of the Gaussian mixture model. At rendering time, for an arbitrary point $x \in \mathbb{R}^3$, the algorithm finds the point $p \in C$ closest to $x$ and takes its distribution $m(p)$ to be used for importance sampling, resulting in a piecewise constant reconstruction of $m(\cdot)$ over $\mathbb{R}^3$.

### Intuition

In the Monte Carlo setting, $m(\cdot)$ is used as an estimator of the integral: $L(x,\omega) = \frac{1}{n}\sum_{i=1}^{n} f(x,\omega_i,\omega)L(x,\omega_i)\cos(n_x \cdot \omega_i)/m(x,\omega_i)$, where $m(x,\omega_i) = m(x)(\omega_i)$ for simplicity. This simple notation reveals that what we truly seek is a function $g : \mathbb{R}^3 \times S^2 \to \mathbb{R}$, such that given any $x \in \mathbb{R}^3$, $g(x,\cdot)$ can be easily sampled and the marginal integral, $G(x) = \int_{\Omega_x} g(x,\omega)d\omega$, is known, so that $g(x,\cdot)$ can be normalized to obtain a distribution. This means that $m(x)(\omega) = g(x,\omega)/G(x)$ and, therefore, $g$ can be used as an estimator. Note that $g(\cdot,\cdot)$ is ideally the representation of the light field in such a way that, for a given $x$, we can easily obtain a distribution that can be sampled.

### Overview

In this paper, we propose a construction that approximates the light field and possesses this desired property. The high-level idea is very simple: we subdivide the space into subspaces and, for each of the subspaces, we create a local model that approximates the light field on this restricted domain. The underlying model is arbitrary,

e.g., a simple constant function or a Gaussian mixture model. The only restriction on it is that it enables efficient sampling and marginal integral calculation. This condition prohibits, for example, the usage of neural networks, as we are not aware of efficient sampling methods for neural networks. The only difficulty is how to actually obtain a hemispherical distribution—or more precisely sample it—for any point in the scene as one would have to assemble several local models for each point to have one proper distribution for the hemisphere. Our key idea is to address this issue by structuring these local models over a *double hierarchy* of the position-normal product space in a way that sampling hemispherical distributions becomes efficient. Similarly to Vorba et al. [2014], each of these local models are created from multiple batches of photons in a preprocessing phase. Additionally, our method enables training with a theoretically infinite number of photons.

In the remainder of this paper, we describe the details of the data structure and the algorithms for sampling and creating/training the local models.

### 3.1. Data Structure

We first introduce the data structures that support our algorithm. Consider a hierarchical subdivision of both $\mathbb{R}^3$ and $S^2$ to a given depth, where each subspace is divided into $c$ smaller subspaces. Let us denote these hierarchical structures by $\mathcal{R} = \{R_i^j \subseteq \mathbb{R}^3\}$ and $\mathcal{Q} = \{Q_i^j \subseteq S^2\}$, where the indices are as follows: $R_i^j$ is the $j$th subspace on the $i$th level ($0 \leq j < c^i$). In practice, these structures might be any hierarchical clustering data structures, e.g., kd-trees or octrees. Each subspace (hereafter called node) stores references to the parent and children nodes. For our implementation, we have opted for simplicity; therefore, we used simple octrees that are built using the Cartesian coordinates for $\mathcal{R}$ and the polar coordinates for $\mathcal{Q}$. In the case of the latter, we have a degenerate octree, i.e., a quadtree, but for the sake of simplicity, we will not distinguish between them. Furthermore, each node in $\mathcal{R}$

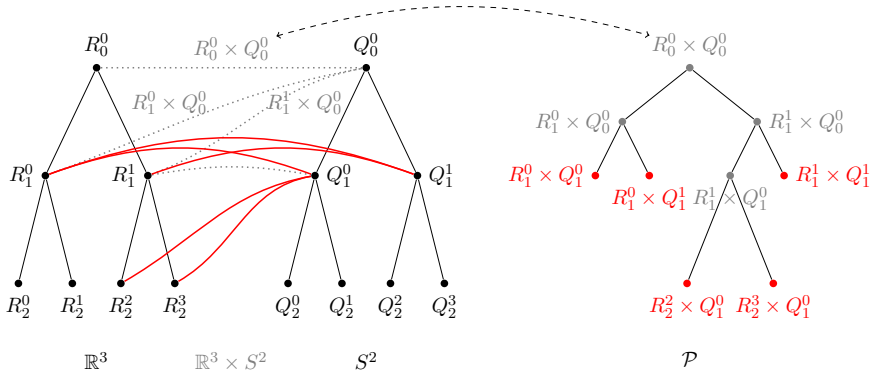


**Figure 1**. Double hierarchy for the product space (left). The dotted gray lines constitute nodes of the product graph $\mathcal{P}$, and the solid red links constitute its leaf nodes.

stores an initially empty list of product-space nodes, `pnodeList` (see below for their definition) and a simple number, `val` (see Section 3.2 for its utility).

Let us denote a hierarchical subdivision of the product space $\mathbb{R}^3 \times S^2$ as $\mathcal{P}$ and restrict the subspaces to have the form $R_i^j \times Q_k^l$ with their children being formed by either subdividing $R_i^j$ or $Q_k^l$ into its $c$ children. Hence, each product-space subspace can be represented by a link between two nodes in $\mathcal{R}$ and $\mathcal{Q}$. Together with the children/parent relationships, $\mathcal{P}$ is stored as a tree structure with the same arity as $\mathcal{R}$ and $\mathcal{Q}$. (See Figure 1.) In addition to this information, each node will also have two pointers, `rnode` and `qnode`, to the associated tree nodes, $R_i^j$ and $Q_k^l$. Moreover the leaf nodes, i.e., the finest subspaces of the hierarchical subdivision are contained in the `pnodeList` field of $R_i^j$. Note that these leaves are disjoint, and that their union covers the whole product space.

A node in $\mathcal{P}$ represents a spatial subspace coupled with a subset of directions. We associate each of the leaf nodes with a local model, $g_{R_i^j \times Q_k^l}$, representing the sought function, $g(\cdot,\cdot)$, a function on this subspace, i.e., $g_{R_i^j \times Q_k^l}(x,\omega_i) = g(x,\omega_i)|_{R_i^j \times Q_k^l}$. For simplicity, we will refer to this model as the `node.model`. Since our local models need to provide quick access to their marginal integral, we restrict these local models to be constant along the spatial dimension (this is needed for instantly accessing the marginal integrals; see Section 3.2). One could relax this condition, provided that one has constant time access to the marginal integrals of the local model, but we have not investigated if this is feasible.

### 3.2. Primitives

In this section we describe the primitives associated to our data structure.

*Construction.* Depending on the exact realization, it is straightforward to build $\mathcal{R}$ and $\mathcal{Q}$. To build $\mathcal{P}$, one simply recursively subdivides product-space clusters starting with $P_0^0 = R_0^0 \times Q_0^0$. To decide whether the spatial or directional node is to be subdivided at a given step, we simply take the one with smaller depth in its tree. The stopping criterion is discussed in the next paragraph. See Algorithm 1 for the pseudocode.

*Refinement.* It is clear that each of these structures can be adaptively refined by simply adding new leaf nodes. After each batch of photons, we refine the trees according to the following rule: Each leaf node in $\mathcal{P}$ is subdivided as long as its local model's marginal integral is not smaller than $\varepsilon$ times the average of the sum of all marginal integrals along the path between $R_0^0$ and its spatial node's leafs. We set $\varepsilon$ to 0.1 in our implementation. Local models are simply moved to the new leaves, and if the node in $\mathcal{Q}$ is subdivided, we also divide the models with the arity of $\mathcal{Q}$. This ensures that none of the local models represent a significant portion of $\int g(x,\omega)d\omega$ with respect

**Algorithm 1** Construction of $\mathcal{P}$.

**Require:** $\mathcal{R}$ a hierarchical space subdivision structure of $\mathbb{R}^3$
**Require:** $\mathcal{Q}$ a hierarchical space subdivision structure of $S^2$
**Require:** S a stack of pairs

```
1:  function BUILDP
2:      proot ← (R.root, Q.root)
3:      S ← ∅
4:      S.push(proot)
5:      while S ≠ ∅ do
6:          p ← S.top()
7:          S.pop ()
8:          p.rnode.pnodeList.remove(p)
9:          m ← μ / 2
10:         if depth(p.rnode) > m and depth(p.qnode) > m then
11:             p.rnode.pnodeList.append(p)
12:         else
13:             if depth(p.rnode) < depth(p.qnode) then
14:                 for all i ∈ children(p.rnode) do
15:                     u ← (i, p.qnode)
16:                     i.pnodeList.append(u)
17:                     S.push(u)
18:                     p.children.append(u)
19:             else
20:                 for all i ∈ children(p.qnode) do
21:                     v ← (p.rnode, i)
22:                     p.rnode.pnodeList.append(v)
23:                     S.push(v)
24:                     p.children.append(v)
25:     return proot
```

to any point in $\mathbb{R}^3$. This follows from the fact that, for any spatial leaf node, the links along the path to the root disjointly cover $S^2$—a direct consequence of the recursive construction algorithm. Moreover, we limit the maximum depth of $\mathcal{R}$ and $\mathcal{Q}$ to $\mu$ which is set to 9 in our implementation. See Algorithm 2 for the pseudocode.

*Training.* Since $\mathcal{P}$ has a tree structure, we can easily descend to a leaf node and update the local model using the photons generated by the system for training. When all photons are processed, we calculate the marginal integral of $g_{R_i^j \times Q_k^l}(\cdot, \cdot)$ for each leaf node $R_i^j \times Q_k^l$ in $\mathcal{P}$ and accumulate it in the `val` field of $R_i^j$. We omit the pseudocode for this step for brevity and refer the reader to the accompanying source

**Algorithm 2** Refinement of $\mathcal{P}$.

**Require:** $\mathcal{R}$ with each node's `val` set to the average sum of marginal integrals
**Require:** S a stack of pairs

```
1:  function REFINE
2:      S ← ∅
3:      S.push(P.root)
4:      while S ≠ ∅ do
5:          p ← S.top()
6:          S.pop ()
7:          if children (p) ≠ ∅ then
8:              for all i ∈ children(p) do
9:                  S.push(i)
10:         else
11:             if p.model.marginal > ε · p.rnode.val then
12:                 if depth(p.rnode) < depth(p.qnode) then
13:                     if depth(p.rnode) < μ then
14:                         for all i∈ children(p.rnode) do
15:                             u ← (i, p.qnode)
16:                             u.model ← p.model
17:                             i.pnodeList.append(u)
18:                             S.push(u)
19:                             p.children.append(u)
20:                 else
21:                     if depth(p.qnode) < μ then
22:                         for all i ∈ children(p.qnode) do
23:                             v ← (p.rnode, i)
24:                             v.model ← p.model / c
25:                             p.rnode.pnodelist.append(v)
26:                             S.push(v)
27:                             p.children.append(v)
```

code which can be found at `http://www.jcgt.org/published/0006/03/02/doublehierarchies_source.zip`.

*Sample generation.* For a given position $x$, we descend to the leaf of $\mathcal{R}$ containing $x$ and sum up the marginal integrals of the product-space nodes stored along the path during this process. Clearly, this sum is the marginal integral on the whole $S^2$ for $x$. Using this value as the normalization constant, we can simply pick one product-space cluster (from the links along the path of the descent) and sample its local model. Note that this gives a sample on $S^2$ instead of the hemisphere $\Omega_x$ defined by the

surface normal at $x$. To solve this problem, one could use rejection sampling. In practice, however, we symmetrized the distribution, i.e., given a sampled direction, $\omega$, we also evaluate $g(x, -\omega)/G(x)$ and return the average of the two probabilities and the direction on the proper side of the surface. This preserves the constant time sampling at the cost of losing precision (see Algorithm 3).

**Algorithm 3** Sample generation.

**function** SAMPLE(x)
    sum_marginals←0
    p ← $\mathcal{R}$.root
    **while** children(p) $\neq \emptyset$ **do**
        p ← child of p containing x
        sum_marginals += i.marginal
    $(\omega, prob_\omega)$ ← pick a random product node proportionally to sum_marginals along the path to $x$ and sample the local model
    $prob_{-\omega}$ ← find the leaf in $\mathcal{P}$ containing (x, -$\omega$) and calculate its probability
    **return** the direction pointing away from the surface and the probability $\frac{1}{\text{sum\_marginals}} \cdot (prob_\omega + prob_{-\omega})$

Given these primitives, our method works as follows: (i) prior to rendering, we *build* shallow hierarchies and *train* the local models with the photons in each batch, while after each batch we *refine* the structures; (ii) during rendering, we simply *sample* the data structure.

## 4. Experiments

We implemented our algorithm in Mitsuba [Jakob 2010], as an extension of the publicly available source code of Vorba et al. [2014] to enable easy comparison with their method. As a consequence, our algorithm also uses the same batch-based training implemented there, i.e., interleaved training with importons and photons and GMM sampling of the environment map. All of our experiments were carried out on an Intel Xeon E5-1660 v3 processor (8 cores @ 3 GHz) with 32 GB DDR4 2133 MHz memory. Our source code is publicly available.

We used several scenes to give examples of difficult scenarios (see Figure 2). In particular, the POOL scene contains a significant amount of caustics, and it is lit by an environment map. The DOOR scene contains an area light source which is not directly visible from the majority of the scene, and it also contains some caustics from this light source. The SPONZA and DINING scenes are included to compare the algorithms' performance in more classical scenarios.

Table 1 gives detailed error, memory, and timing results for the tested scenes and methods. The reference has been created by the Gaussian distribution-based algo-

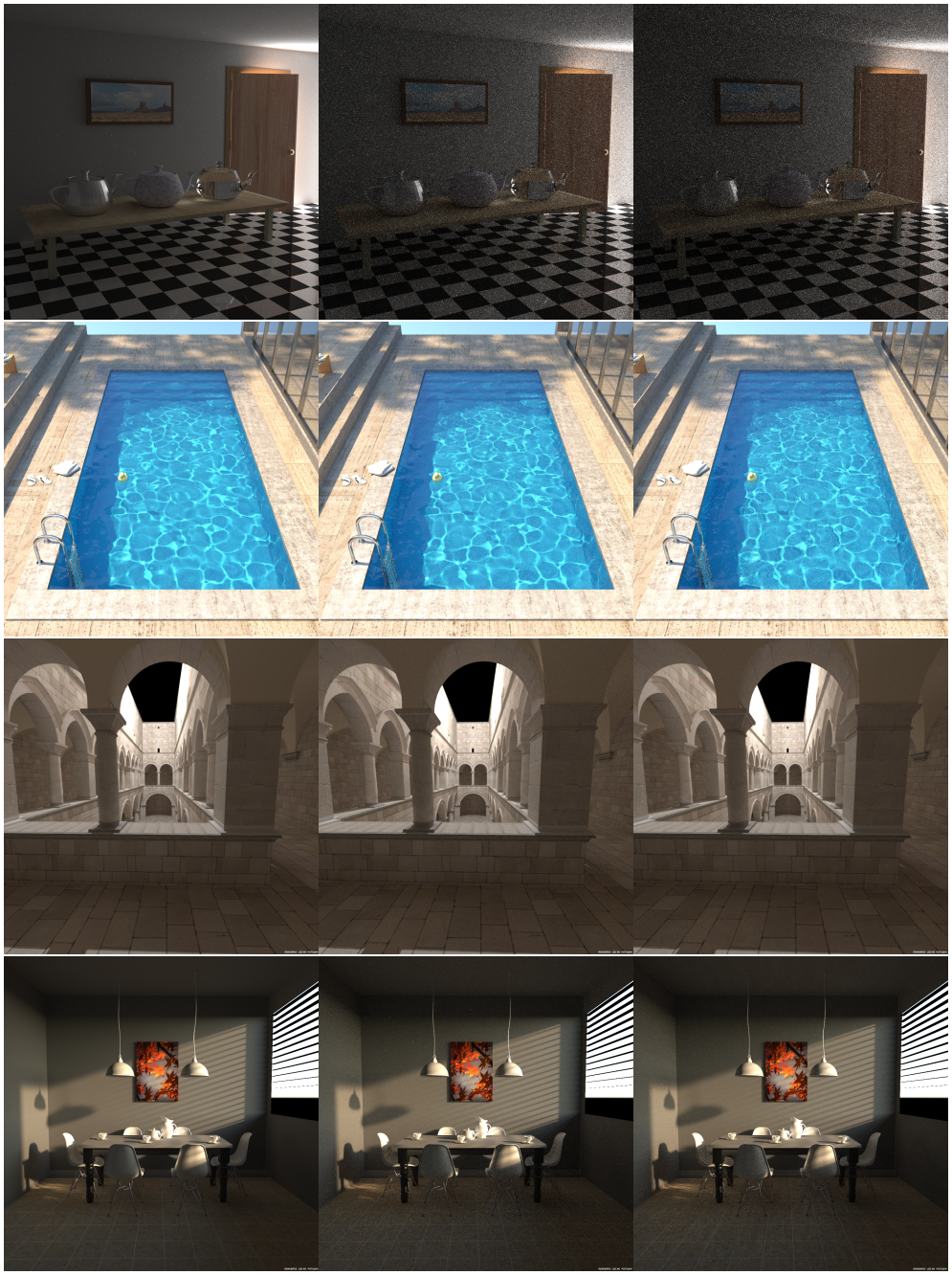

**Figure 2.** Images rendered with the different algorithms. From left to right: reference, Gaussian, double hierarchy. Top to bottom: DOOR, POOL, SPONZA and DINING scenes.

rithm of Vorba et al. [2014] using 65K samples per pixel, while the test images have 256 samples per pixel. We compare our method denoted as Double with this method, denoted as Gauss. All images have $1024 \times 1024$ resolution, and the maximum depth

| Scene | Method | L1 | RMSE | SSIM | Memory | Preproc. | Render | Total(s) |
|---|---|---|---|---|---|---|---|---|
| POOL | Gauss | 0.213967 | 1.537650 | 0.663241 | 897.57 | 444.00 | 354.00 | 805.00 |
| | Double | 0.257429 | 1.132270 | 0.611492 | 1456.75 | 282.00 | 282.00 | 570.00 |
| DOOR | Gauss | 0.000226 | 0.002333 | 0.997712 | 938.51 | 816.00 | 528.00 | 1341.00 |
| | Double | 0.000255 | 0.001881 | 0.997360 | 2299.05 | 336.00 | 510.00 | 845.00 |
| SPONZA | Gauss | 0.001202 | 0.002221 | 0.998713 | 785.91 | 600.00 | 660.00 | 1263.00 |
| | Double | 0.000950 | 0.001855 | 0.999544 | 866.84 | 120.00 | 582.00 | 704.00 |
| DINING | Gauss | 0.006490 | 0.060370 | 0.899290 | 740.75 | 468.00 | 630.00 | 1096.00 |
| | Double | 0.005835 | 0.029639 | 0.918666 | 1080.86 | 210.00 | 648.00 | 861.00 |

**Table 1**. Numerical results for images with 256 samples per pixel comparing the Gaussian-mixtures approach by Vorba et al. (Gauss) with our method (Double).

of each light path is 10. The preprocessing includes training the data structures with 30 batches of 300K photons. We observe that, even with the very simple constant model, our method performs comparably to the Gauss method in terms of quality for a similar memory usage (except in the DOOR scene). We believe that the high memory usage and somewhat lower quality in the DOOR and SPONZA scenes is a consequence of our refinement algorithm being suboptimal, as it only takes into account photons but not importons. One could augment our method similar to the solution in [Simon et al. 2015]. Moreover, the subdivision criteria and the symmetrization of distributions we used is not efficient for spatial subspaces where there is a large difference between actually visible light and the incoming photons (e.g., two sides of the door being very differently lit in the DOOR scene). This could also explain the high memory usage as it refines the trees to an unnecessary depth. Most importantly, in terms of performance, our approach achieves a significant speed-up in pre-processing time, with an additional speed-up in the rendering time. Additionally, we note that our approach handles a sparse training set of photons easily even if our final piecewise constant distributions (histograms) have hundreds of bins as they are trained on big areas containing photons. However, high-frequency features might be less precisely represented due to our coarse-to-fine construction strategy. Finally, we believe that the simplicity of our method makes it a valuable alternative to methods based on Gaussian mixtures.

## 5. Discussion and Future Work

Our method provides a fast, very robust, easy to implement, and flexible framework for importance sampling in Monte Carlo rendering that could be useful in many other sampling problems in rendering. In particular, our double-hierarchy structure enables the usage of arbitrary distribution models and performs well even with the simplest constant models. As future work, it would be interesting to study how one could jointly sample this illumination model and the BRDFs, as well as to explore more ex-

pressive local models and more adaptive refinement strategies that take into account, for example, surface normals for photons or the difference between the children light fields of a product node.

### Acknowledgements

This work is partially supported by the French National Research Agengy (ANR) under grant ANR 16-LCV2-0009-01 ALLEGORI and by BPI France, under grant PAPAYA. We would also like to thank the authors of [Vorba et al. 2014; Lehtinen et al. 2013], Benedikt Bitterli (for the scenes created by Jay-Artist, Wig42), and Marko Dabrovic for making the scenes used in this paper available for research.

## References

BUS, N., MUSTAFA, N. H., AND BIRI, V. 2015. IlluminationCut. *Computer Graphics Forum (Proceedings of Eurographics 2015) 34*, 2, 561–573. 27

CLARBERG, P., JAROSZ, W., AKENINE-MÖLLER, T., AND JENSEN, H. W. 2005. Wavelet importance sampling: Efficiently evaluating products of complex functions. *ACM Trans. Graph. 24*, 3 (July), 1166–1175. URL: http://doi.acm.org/10.1145/1073204.1073328, doi:10.1145/1073204.1073328. 26

HEY, H., AND PURGATHOFER, W. 2002. Importance sampling with hemispherical particle footprints. In *Proceedings of the 18th Spring Conference on Computer Graphics*, ACM, New York, NY, USA, SCCG '02, 107–114. URL: http://doi.acm.org/10.1145/584458.584476, doi:10.1145/584458.584476. 26

HUA, B.-S., AND LOW, K.-L. 2015. Guided path tracing using clustered virtual point lights. In *SIGGRAPH Asia 2015 Posters*, ACM, New York, NY, USA, SA '15, 43:1–43:1. URL: http://doi.acm.org/10.1145/2820926.2820955, doi:10.1145/2820926.2820955. 26

JAKOB, W., 2010. Mitsuba renderer. http://www.mitsuba-renderer.org. 32

JAROSZ, W., CARR, N. A., AND JENSEN, H. W. 2009. Importance sampling spherical harmonics. *Computer Graphics Forum (Proceedings of Eurographics) 28*, 2 (Apr.), 577–586. doi:10.1111/j.1467-8659.2009.01398.x. 26

JENSEN, H. W. 1995. Importance driven path tracing using the photon map. In *Rendering Techniques '95: Proceedings of the Eurographics Workshop in Dublin, Ireland, June 12–14, 1995*, P. M. Hanrahan and W. Purgathofer, Eds. Springer Vienna, Vienna, 326–335. doi:10.1007/978-3-7091-9430-0_31. 25, 26

KAJIYA, J. T. 1986. The rendering equation. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '86, 143–150. 27

KELLER, A. 1997. Instant radiosity. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, SIGGRAPH '97, 49–56. doi:10.1145/258734.258769. 26

LEHTINEN, J., KARRAS, T., LAINE, S., AITTALA, M., DURAND, F., AND AILA, T. 2013. Gradient-domain Metropolis light transport. *ACM Trans. Graph. 32*, 4 (July), 95:1–95:12. URL: http://doi.acm.org/10.1145/2461912.2461943, doi:10.1145/2461912.2461943. 35

REN, P., WANG, J., GONG, M., LIN, S., TONG, X., AND GUO, B. 2013. Global illumination with radiance regression functions. *ACM Trans. Graph. 32*, 4 (July), 130:1–130:12. 26

SIMON, F., HANIKA, J., AND DACHSBACHER, C. 2015. Rich-VPLs for improving the versatility of many-light methods. *Comput. Graph. Forum 34*, 2 (May), 575–584. doi:10.1111/cgf.12585. 34

VEACH, E., AND GUIBAS, L. 1995. Bidirectional estimators for light transport. In *Photorealistic Rendering Techniques*, G. Sakas, S. Müller, and P. Shirley, Eds., Focus on Computer Graphics. Springer Berlin Heidelberg, 145–167. 25

VEACH, E., AND GUIBAS, L. J. 1995. Optimally combining sampling techniques for Monte Carlo rendering. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '95, 419–428. 25

VORBA, J., KARLÍK, O., ŠIK, M., RITSCHEL, T., AND KŘIVÁNEK, J. 2014. On-line learning of parametric mixture models for light transport simulation. *ACM Trans. Graph. 33*, 4 (July), 101:1–101:11. URL: http://doi.acm.org/10.1145/2601097.2601203, doi:10.1145/2601097.2601203. 25, 26, 27, 28, 32, 33, 35

## Index of Supplemental Materials

Supplemental materials can be found at http://www.jcgt.org/published/0006/03/02/doublehierarchies_supplemental.zip. Images included in the manuscript are included in the supplemental material in EXR format. The source code and supplemental materials can also be found on the website of the authors.

## Author Contact Information

Norbert Bus
Télécom-ParisTech
Dpt. IDS,
46 Rue Barrault
Paris, 75013 France
norbert.bus@telecom-paristech.fr
http://www.telecom-paristech.fr/~nbus

Tamy Boubekeur
Télécom-ParisTech
Dpt. IDS,
46 Rue Barrault
Paris, 75013 France
tamy.boubekeur@telecom-paristech.fr
http://www.telecom-paristech.fr/~boubek

Norbert Bus, Tamy Boubekeur, Double Hierarchies for Directional Importance Sampling, *Journal of Computer Graphics Techniques (JCGT)*, vol. 6, no. 3, 25–37, 2017
http://jcgt.org/published/0006/03/02/

| | | | |
|---|---|---|---|
| Received: | 2017-02-09 | | |
| Recommended: | 2017-05-08 | Corresponding Editor: | Wenzel Jakob |
| Published: | 2017-08-28 | Editor-in-Chief: | Marc Olano |