Point-Based Light Transport for Participating Media with Refractive Boundaries
Beibei Wang, Jean-Dominique Gascuel, Nicolas Holzschuch

To cite this version:
Beibei Wang, Jean-Dominique Gascuel, Nicolas Holzschuch. Point-Based Light Transport for Participating Media with Refractive Boundaries. Eurographics Symposium on Rendering 2016 - Experimental Ideas & Implementations, Jun 2016, Dublin, Ireland. 10.2312/sre20161216. hal-01327239v2

HAL Id: hal-01327239
https://inria.hal.science/hal-01327239v2
Submitted on 19 Jan 2017

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.
Point-Based Light Transport for Participating Media with Refractive Boundaries

Beibei Wang∗ Jean-Dominique Gascuel∗ Nicolas Holzschuch∗

∗INRIA; Université Grenoble-Alpes, LJK; CNRS, LJK

Abstract
Illumination effects in translucent materials are a combination of several physical phenomena: absorption and scattering inside the material, refraction at its surface. Because refraction can focus light deep inside the material, where it will be scattered, practical illumination simulation inside translucent materials is difficult. In this paper, we present an a Point-Based Global Illumination method for light transport on translucent materials with refractive boundaries. We start by placing volume light samples inside the translucent material and organising them into a spatial hierarchy. At rendering, we gather light from these samples for each camera ray. We compute separately the samples contributions to single, double and multiple scattering, and add them. Our approach provides high-quality results, comparable to the state of the art, with significant speed-ups (from 9× to 60× depending on scene complexity) and a much smaller memory footprint.

1. Introduction
Participating media are frequent in real-world scenes, whether it is milk, fruit juices, oil or muddy water in river or ocean scenes. Incoming light interacts with these participating media in complex ways: it is refracted at the boundary, absorbed and scattered as it travels inside the medium. These physical phenomena have different contributions on the overall aspect of the material: refraction focuses light in some parts of the medium, creating high-frequency events, or volume caustics. Scattering blurs incoming light, spreading its contribution. Absorption reduces light intensity as it travels inside the medium.

This complex interplay between these different phenomena makes simulating light transport in participating media a difficult and ongoing research problem. It is especially difficult for materials with relatively low albedo and large mean-free-path, as scattering events inside the medium are more visible. Directional phase functions and refraction at the interface add to the computational complexity.

Figure 1: Our algorithm (a), compared with Bi-Directional Path Tracing (BDPT) (b), Photon Mapping with Beam-Radiance Estimate (BRE) (c) and Unified Points, Beams and Paths (UPBP) (d) (e). Our algorithm is up to 60× faster than UPBP, with similar quality. Material: olive oil, α = {0.0042, 0.4535, 0.0995}; ℓ = {9.7087, 11.6279, 2.7397}. For this material with low albedo α and large mean-free-path ℓ, low-order scattering effects dominate.
Recent approaches, combining photon mapping with beams and paths [JNSJ11, KGH’14] provide very good results, but can still take a long time to converge. With some materials, the initial results are still noisy.

Point-Based Global Illumination [Chr08] is widely used for light transport simulation in surface scenes. The basic idea is to decorrelate the scene complexity from the illumination computation by replacing the scene with a shaded point cloud when computing indirect illumination, only keeping the polygonal representation for direct visibility from the camera. The algorithm scales well with scene complexity and provides noise-free results.

In this paper, we introduce a point based method for global illumination in participating media. As in all point-based method, we begin by computing light samples. The difference is that they are placed inside the medium. These volume sample points are organized in a spatial hierarchy. For each camera ray, we compute illumination from the light samples, separating single, double and multiple scattering contributions. Single scattering is computed directly, finding light samples that are closer to the camera ray. To compute double scattering, we traverse the spatial hierarchy to obtain the best tree cut and gather the contributions from these nodes. For multiple scattering, we use a precomputed table storing the resulting contribution. We then add the contributions from single, double and multiple scattering. For indirect lighting and multiple scattering after several bounces on the refractive surface, we also use surface samples, organized in a separate spatial hierarchy, and add the contribution from these samples.

Our algorithm has the advantage of fast convergence, with little noise during simulation. It provides a natural compromise between computation time and quality, by acting on the number of samples. Compared to existing algorithms, it is both faster and with a smaller memory footprint.

In the next section, we review previous work on light simulation in participating media. We then present our algorithm for Point-Based Global Illumination in Participating Media in Section 3. In Section 5 we validate results for each step of our algorithm (single, double and multiple scattering) and compare with previous work and reference solution. Finally, we conclude in Section 7 and discuss avenues for future work.

2. Previous Work

Subsurface Scattering: Jensen et al. [JMLH01] introduced the dipole method to Computer graphics for practical rendering of participating media. The dipole method works better with high-albedo materials, where multiple scattering effects dominate. Frisvad et al. [FHK14] introduced the Directional Dipole, relaxing the assumption that incoming light is orthogonal to the material surface. D’Eon and Irving [DI11] introduced quantized diffusion, improving the accuracy for rendering high-absorption materials. All these methods are designed for materials with high albedo, where directional effects are cancelled by the large number of scattering events. Our algorithm targets a large range of translucent materials, from almost transparent to almost opaque.

Donner et al. [DLR’09] precomputed surface response as a BSS-RDF for a large range of participating media. Surface response is encoded using elliptic coordinates over directions. We store material response to multiple scattering at the volume level instead of the surface level, and separate between double- and multiple-scattering, resulting in more accurate and more compact representation.

Accurate Single Scattering: Inside participating media with refractive boundaries, single scattering effects can produce volume caustics, with complicated shapes. Walter et al. [WZH09] introduced a method for accurate computation of single scattering effects in participating media, computing the entry point using Newton-Raphson optimization. Holzschuch [Hol15] improved both accuracy and speed by computing the extent of the influence of each triangle over the camera ray. We use this algorithm as a reference for single scattering.

Photon Mapping: Jensen and Christensen [JCC08] presented an extension of the Photon Mapping algorithm for participating media. Jarosz et al. [JZJ08] extended the algorithm by using a beam around the camera ray to gather the radiance from the photon map, resulting in faster computations and less noise in the results. Jarosz et al. [JNSJ11, JNT’11] extended this idea by tracing beams inside the media rather than simply photons. Krivánek et al. [KGH’14] improved the algorithm by automatically selecting between beams, points and paths in light transport simulation, using multiple importance sampling. Our algorithm is similar in scope, but works within a Bi-Directional Path Tracing framework rather than Photon Mapping.

Point-Based Global Illumination: Christensen [Chr08] introduced Point-Based Global Illumination as a way to evaluate diffuse light transport by representing direct illumination using a mesh-less hierarchy of points, along with a Z-buffer inspired approach to solve for visibility at each receiver. Arbee et al. [AWB08] extended the approach for subsurface scattering. Yan et al. [YWXZ12] used Gaussian spherical light sources instead of point lights. Both approaches focused more on materials with high albedo, where multiple scattering effects dominate.

Virtual Point Lights: Keller [Ke97] proposed virtual point light method for fast GI computation. Hašan et al. [HWKB09] improved it by introducing spherical lights to avoid singularity. Novák et al. [NNDJ12b] used virtual point lights and virtual ray lights for light transport inside translucent materials. Novák et al. [NNDJ12a] replaced virtual ray lights with virtual beam lights to remove singularities. They used importance sampling for the transfer between camera rays and virtual light sources, while we use a spatial hierarchy and precompute multiple scattering effects. Walter et al. [WFA’05] introduced the lightcuts technique to cluster the virtual lights resulting in large speed-up. Walter et al. [WABG06] extended the domain of cuts to light-receiver pairs and Walter et al. [WKB12] extended it to glossy reflection and subsurface scattering.

3. Point-Based Light Transport in participation media

3.1. Radiative Equation

We consider a scene containing objects with translucent material. Each of these is assumed to be made of an homogeneous material,
Light transport within participating medium is described by the radiative transfer equation [Cha60], which defines the radiance that reaches a point $x$ form direction $\omega$ as a sum of exitant radiance from the nearest surface from this direction and in-scattered radiance from the medium among the whole length of the ray. This can be expressed as:

$$L_i(x, \omega) = T_i(x \leftrightarrow x_i)L_i(x_i, \omega) + \int_0^s T_i(x \leftrightarrow x_i)\sigma_t(x_i)I_v(x_i, \omega)dt,$$

where $T_i$ is the transmittance, defined as:

$$T_i(x \leftrightarrow x_i) = e^{-\sigma_t \ell - \sigma_a s},$$

$s$ is the distance through the medium to the nearest surface at $x_i = x - s\omega$, and $x_i = x - t\omega$ with $t \in (0, s)$. $L_i(x_i, \omega)$ is the exit radiance from the nearest surface, which is governed by the rendering equation [Kaj86]. $I_v(x_i, \omega)$ is the in-scattering radiance at $x_i$ from all direction $\omega_i$ over the sphere of directions $\Omega_\omega$ using the phase function $p$, defined as:

$$I_v(x_i, \omega) = \int_{\Omega_\omega} p(\omega, \omega_i)L_i(x_i, \omega_i)d\omega_i.$$

### 3.2. Context

We place ourselves within a standard Point-Based Global Illumination framework, with Bi-Directional Path-Tracing: in a precomputation step, we place light samples in the scene. The main difference is that we also place volume samples inside the translucent material, along with the usual surface samples (see Figure 2).

At rendering time, we trace rays from the camera. These rays traverse the scene and are reflected by surfaces, until they reach the translucent object. Inside the translucent object, we compute contributions from the light samples to this particular ray. The way we sample a given ray depends on the nature of the reflections it has encountered: rays directly reaching the translucent object are sampled more than rays reaching it after diffuse reflections.
3.3. Notations

We separate between single-, double- and multiple-scattering effects, depending on the number of volume scattering events inside the translucent material. Single scattering corresponds to a light path with only one scattering event inside the material, double scattering to paths with two scattering events, and multiple scattering to paths with more than two scattering events. We only count the number of scattering events, independently of the number of internal reflections on the specular surface. A path coming from the light source, being refracted, then reflected several times on the internal surface of the material before having a single scattering event and leaving the material corresponds to single scattering.

3.4. Light Surface Samples

In a first step, we place blue-noise surface samples at the surface of the object, as suggested by Jensen and Buhler [JB02]. These surface samples store indirect diffuse illumination from the scene, as well as indirect illumination from inside the translucent material after multiple bounces on the surface (see Section 3.10).

We adapt the sampling process when the participating media is enclosed inside several layers of transparent refractive interfaces, as is common with liquids (see, e.g., Figures 1 and 10): we place blue-noise sample points on the outer layer of transparent material, connect these sample points to the light source, then trace the refracted ray inside the transparent layers until we reach the participating media. We then process these samples to remove samples that are too close to other samples, ensuring a minimal distance between samples on the surface of the participating media.

Each light surface sample is defined by its position \( x \), and diffuse intensity \( I_d \). We build a spatial hierarchy over these surface samples for future queries (an octree).

3.5. Light Volume Samples

We then place volume samples, storing incoming light inside the translucent material. They will be used in subsequent steps to compute illumination inside the volume.

We start with the surface samples we computed in the previous section. We connect these points to the light source, and compute the refracted ray inside the material. We follow this refracted ray, including internal reflections on the enclosing surface, and sample it to create the volume samples.

Each volume sample is defined by its position \( x \), the direction of incoming light \( d_i \), incoming light intensity \( I_i \), and a bounding box. The bounding box is computed using the surface area around the surface sample we started with, and the length of the sampling interval on the incoming ray.

We build a spatial hierarchy over the light volume samples for future queries (an octree). Each node of the hierarchy stores the average position, average direction of incoming light and combined bounding box of its descendants.

3.6. Sampling the camera ray

We shoot rays from the camera. They reach the translucent object directly or after several reflections or refractions. For each camera ray reaching the surface of the translucent object, we compute the refracted ray inside the object and its direction \( o \). We then place sample points \( P_k \) on the camera ray; each one is defined by its depth \( d_k \) along the ray, measured from the entry point. We will compute outgoing radiance at these sample points and combine these values together to get the radiance for this camera ray. Please note that there are usually several camera rays per pixel, for anti-aliasing and indirect illumination computations.

We use two different sampling strategies for sampling the camera ray inside the participating media:

- if the camera ray reaches the participating media directly or after only specular events (reflections and refractions), we place \( k_{\text{max}} \) samples on the ray, with depth varying exponentially to place more samples near the surface (see Section 4.1).
- If the camera ray reaches the participating media after diffuse interactions, we sample it randomly with a smaller amount of samples. Contributions from these rays will be averaged over the diffuse reflections, resulting in a converged value in the end.

In the remainder of this section, we consider a camera ray with direction inside the material \( o \) and a sample point \( P_k \) on the camera ray, at a depth \( d_k \). We compute single, double and multiple scattering from volume and surface light samples, including absorption between the entry point and \( P_k \).

3.7. Single scattering

Single scattering corresponds to light that enters the participating medium, is scattered exactly once, then leaves the participating medium. With our framework, it corresponds to the light volume samples that are close to the camera ray sample point \( P_k \).

We compute it by traversing the spatial hierarchy of volume samples. If the bounding box of a node does not contain \( P_k \), we stop the traversal. If it does contain it, we continue the traversal until we reach the leaf nodes.

We then average the contributions from all leaf nodes \( v_i \) whose bounding box contain \( P_k \):

\[
\text{single}(P_k) = e^{-\sigma d_k} \frac{1}{N} \sum_{v_i} \sigma_i \rho(o, d_i) I_{v_i}.
\]

3.8. Double Scattering

We compute double scattering directly from the hierarchy of light volume samples. The algorithm is similar to PBGI: we traverse the hierarchy of volume samples until we reach a node that satisfies our criterion for accuracy. Our stopping criterion uses the solid angle sustained by the node from point \( P_k \) and the phase function between outgoing direction \( o \) and the direction from \( P_k \) to the center of the node \( v_i \). This gives us a tree cut.

For each node in this tree cut, we compute double scattering from the volume sample \( v_i \) to \( P_k \): light that has entered the material, is scattered once at \( v_i \), in the direction of \( P_k \), reaches \( P_k \), is scattered at

\[
\text{double}(P_k) = e^{-2\sigma d_k} \frac{1}{N} \sum_{v_i} \sigma_i \rho(o, d_i) I_{v_i}.
\]
We use the same tree cut to compute multiple scattering, using a 3.9. Multiple Scattering by Precomputation

We use the same tree cut to compute multiple scattering, using a precomputed table. Multiple scattering corresponds to light that has reached \( v_i \), is scattered several times before reaching \( P_k \) and is scattered one last time into direction \( o \).

\[
\begin{align*}
\rho_k &= \| (P_k - P_i) - z_k d_o \| \\
\text{mult.} (P_k) &= e^{\sigma d_o} \sum_{v_j} \frac{e^{-\sigma t_j} I_{s_j}}{q_j^2} f_j(t_j) \sigma(o, t_j).
\end{align*}
\]

3.10. Bounced Multiple Scattering

Our precomputed table for multiple scattering assumes an infinite medium. Our computations do not account for the light that has bounced on the internal surface of the material. To correct for this missing light, we add diffuse the indirect illumination from inside the material to our surface samples.

At each surface sample, we are already storing diffuse illumination from the environment, as was done by Jensen and Buhler to compute subsurface scattering [JBO2]. We add to this the light coming from inside the material and reflected by the surface. This is done simply by computing the position of the surface sample relative to the volume sample and extracting the incoming radiance from the precomputed table. For simplicity, we store incoming illumination at the surface samples as a diffuse value.

For each camera sample \( P_k \), we begin by computing a tree cut over surface samples, using the same refinement criterion as in Section 3.8: the solid angle sustained by the node multiplied by the phase function from \( P_k \) to the center of the node. We then accumulate contributions from the surface samples, multiplying the stored incoming diffuse radiance with surface BRDF \( f_j \):

\[
\begin{align*}
q_j &= \| P_k - s_j \| \\
t_j &= \| (P_k - s_j)/\|P_k - s_j\| \\
\text{bounced} (P_k) &= e^{\sigma d_o} \sum_{s_j} \frac{e^{-\sigma t_j} I_{s_j}}{q_j^2} f_j(t_j) \sigma(o, t_j).
\end{align*}
\]

3.11. Full solution

The full solution for a given ray is the sum of all contributions from all sample points:

\[
L = \sum_k \text{single} (P_k) + \text{double} (P_k) + \text{mult.} (P_k) + \text{bounced} (P_k).
\]

4. Implementation Details

4.1. Camera Ray Samples

When the camera ray reaches the translucent material directly or after specular events (reflections or refractions), we use the following algorithm to place sample points \( P_i \):

- Compute the exit point for this camera ray. This gives the maximum depth along this ray, \( d_{\text{max}} \).
Compute \( k_{\text{max}} \) sample points \( P_i \), each defined by its depth along the ray \( d_i \):

\[
d_i = -\frac{1}{\sigma_t} \log \left( 1 - \frac{k(1 - e^{-\sigma_t d_{\text{max}}})}{k_{\text{max}}} \right).
\]  

This sampling scheme ensures that we always place \( k_{\text{max}} \) samples, even if object width is small along this ray, it also places more samples closer to the surface, where illumination effects are less attenuated by absorption.

When the camera ray reaches the translucent material after diffuse interactions, we sample it randomly with a small number of samples, as low as one sample per camera ray.

### 4.2. Precomputed Multiple Scattering

We compute multiple scattering in our precomputed table in two steps. First, we trace particles in an infinite medium, starting from a point light source shooting particles along the \( z \) axis. Particles propagate in a straight line in the medium, are scattered and absorbed using standard Monte-Carlo procedure.

We store particle contribution in a 2D grid, exploiting the symmetry of revolution around the \( z \) axis. We use cylindrical coordinates \((\rho, z)\), where \( \rho \) is the distance from the \( z \) axis, and \( z \) is the depth along this axis (see Figure 4). We discretize cylindrical coordinates into a grid of \( 512 \times 512 \) cells. Each cell stores a lobe representing radiance from multiple scattering at this point. These lobes are stored using spherical coordinates \((\theta, \phi)\), sampled with 18 directions for \( \theta \) and 36 for \( \phi \).

In a second step, we compress this representation using a quadtree in a bottom-up approach: if four neighbouring grid cells contain almost identical lobes, we replace them with a single cell, storing the same lobe. The hierarchical representation allows adaptive compression of our precomputed multiple scattering, reducing the memory cost from 8.4 GB to approximately 100 MB. The actual compression rate depends on material properties (see Section 5.2).

### 5. Results

We have implemented our algorithm inside the Mitsuba Renderer [Jak10]. We compared our algorithm against (i) photon mapping with Beam-Radiance Estimate (BRE) by Jarosz et al. [JJJ08], (ii) Bi-Directional Path Tracing (BDPT) and (iii) Unified points, beams and paths (UPBP) by Krivánek et al. [KGH′14], which we take as the reference. We used Mitsuba for our algorithm and BRE, and SmallUPBP [Kfi14] for BDPT and UPBP. For BRE, we use the default value of 120 as the look up size, which is the number of photons that is fetched in photon map queries. For UPBP, we only set the path length and target rendering time.

We also compared our single scattering computations against the reference solution from Holzschuch [Hol15], implemented in the Mitsuba renderer.

All timings in this section are measured on a 2.67GHz Intel i7 (32 cores) with 32 GB of main memory. All pictures were rendered at a resolution of \( 1024 \times 1024 \) pixels, except for the Bumpy Sphere, where we used \( 512 \times 512 \). We rendered the picture using tiles of \( 32 \times 32 \) pixels. We measure numerical differences between simulations using the Mean-Squared Error (MSE).

All materials in our test scenes are homogenous materials, with Henyey-Greenstein phase functions and smooth refractive boundaries. Material properties were taken from Krivánek et al. [KGH′14], Narasimhan et al. [NGD′06] and Holzschuch [Hol15].

### 5.1. Individual Component Validation

We start by comparing the response from each individual component in our algorithm to the reference solution on the Bumpy sphere test scene (see Figure 5). We used the algorithm from Holzschuch [Hol15] as our reference for single scattering, and UPBP by Krivánek et al. [KGH′14] as our reference for double and multiple scattering.

Our algorithm provides a very good match with the reference for each component, at a fraction of the cost in terms of rendering time.

For single scattering, the difference with Holzschuch’s algorithm [Hol15] depends on the number of volume samples we use. With 5.66 M samples, our algorithm computes single scattering using 1.2 GB in 15s, a 3x speedup compared to [Hol15]. Zooming in, we observe a slight degradation in quality for our algorithm; the algorithm from [Hol15] finds sharper volume caustics. Increasing the number of volume samples brings our result closer to those of [Hol15], with similar computation time and an increased memory cost (see Figure 6).

For double and multiple scattering, our algorithm gives results that are almost identical to the reference, at a fraction of the cost. On this specific scene, our algorithm takes 4 mn to compute contributions for all components together, compared to 6 h for UPBP, a 90x acceleration. The memory cost for our algorithm is also 5 times smaller than for UPBP.

Our algorithm behaviour is quite robust with varying material properties. Figure 7 shows the Mean-Square Error for each component on the Bumpy Sphere Scene as a function of \( g \) with a constant number of volume samples. The error for each component stays within the same order of magnitude as we go from isotropic materials (\( g = 0 \)) to strongly anisotropic materials (\( g = 0.9 \)).

### 5.2. Precomputed Multiple Scattering

The precomputed table used for multiple scattering is an important part of our algorithm. Table 2 gives the computation time and memory cost (after compression) for all the materials in our test scenes. Materials with a high albedo require longer computations, as particles are less likely to be absorbed at each event. Memory costs after compression are harder to predict, and appear to depend both on the albedo and the directionality of the phase function. The uncompresses multiple scattering data is 8.4 GB, so our bottom-up compression algorithm reduces data size by a factor of at least 50.

### 5.3. Comparison to Other Methods

In Figures 1, 8, 9 and 10, we compare our algorithm to Bi-Directional Path Tracing, Photon Mapping with Beam-Radiance
Our, 1.2GB, 15s
Holzschuch 2015, 46s
Our, 45.78s
UPBP, 3h
Our, 2.52mins
UPBP, 6h

Figure 5: Individual component validation on the Bumpy Sphere scene.

Our, 1.2GB, 15s
Our, 2.35GB, 46s
Holzschuch 2015, 46s
Our, 2.35GB, 46s

Figure 6: Single scattering validation on Bumpy Sphere scene.

Figure 7: Mean Square Error of single/double/multiple scattering as a function of $g$ for the Bumpy Sphere Scene.

Estimate and Unified Points, Beams and Paths. Our algorithm produces images that are very close to our reference, UPBP, while being an order of magnitude faster. Bi-Directional Path Tracing has problems connecting paths through the refractive interface, missing single scattering effects. Photon mapping with Beam Radiance Estimate gives images with more noise than UPBP when low order scattering effects dominate, and is comparable when multiple scattering effects dominate.

We use two different renderers in our tests, Mitsuba and Small-UPBP. They agree on most of our test scenes, except for Figure 10. As this difference is also present with photon mapping with Beam Radiance Estimate (see (c) in Figure 11), even after convergence, it could be related to scene format differences between the two renderers.

Our algorithm cannot handle all light paths. If the light is reflected by a highly glossy reflection surface before entering the volume, we miss the volume caustic because the surface was treated as diffuse (see Figure 12).

5.4. Performance and Timings

Table 3 gives computation time, memory cost and Mean Square Error for all our test scenes. Our algorithm gives a speed-up of 9× to 60× compared to the reference solution, with negligible differences. It also has a much smaller memory footprint.

We report computation times for the entire scenes, with both the translucent material and its environment. On some scenes, most of the computation time is related to the environment, not to our algorithm itself.

6. Discussion

Novák et al. [NNDJ12b] and Novák et al. [NNDJ12a] deal with refraction effects, but are limited to multiple scattering. The other previous work using using lightcuts deal with single and multiple scattering, but do not address the issues caused by refraction at the interface. The second scattering in our method is computed in a similar manner as lightcuts, except the threshold to traverse the hierarchy. In our method, each surface sample has area and each volume sample has volume, which means all the samples are coarse representation of the geometry, the same as the basic idea of PBGI. We use solid angle as a metric to traverse the hierarchy to compute the tree-cut. In lightcuts, a bounded cluster error is used to determine the light cut.

There are some similarities between our precomputed multiple
scattering and [MWM07] approach. However, there are several strong differences. First, we deal with a continuous anisotropic media, instead of the discrete isotropic media of [MWM07]. Second, Moon et al. [MWM07] store the probability density on a set of concentric spheres. We exploit the symmetry of revolution of the problem to reduce it to two dimensions, and store outgoing radiance on a planar 2D hierarchy.

The main limitation of our method is that we assume homogeneous translucent materials. Extension to heterogeneous materials, with spatially varying scattering properties, is relatively straightforward but will require future work. Another limitation is that we separated incoming light into two components: direct and specular, which contribute to volume samples, and indirect diffuse, which contributes to diffuse surface samples. Extension to glossy indirect lighting is an avenue for future work. Our algorithm inherits both the good and bad points from PBGI. Our algorithm is fast, as a number of samples are cached. However, this leads to larger memory cost than Monte Carlo based methods.

The memory cost of our precomputed table used for multiple scattering is still relatively high. More compact representations, for example using a sum of spherical gaussians for the lobes could be interesting [TS06, XSD∗13].

Finally, we do not consider visibility when summing the contributions from volume and surface samples to a camera sample. This is not an issue in our test scenes, but could be in other scenes. The solution would be to use microbuffers [Chr08, REG∗09, WMB15].

7. Conclusion
We have presented an extension of Point-Based Global Illumination for light transport in participating media with refractive boundaries. Our algorithm begins by computing volume light samples inside the participating media, organising them in a spatial hierarchy. These volume samples are used to compute single, double and multiple scattering, using tree cuts and a precomputed table for multiple scattering.

Our algorithm performs well for a large range of materials, from low albedo to high albedo, and from isotropic to highly anisotropic.

© 2016 The Author(s)
Our algorithm is integrated inside a bi-directional path tracing framework and computes illumination exchanges between the translucent material and the environment. In future work, we want to improve indirect lighting integration, using directional surface samples. We also want to improve the compression ratio for our precomputed multiple scattering table, and to work on heterogeneous participating media.

Acknowledgements. We thank the reviewer for the valuable comments. We thank Guillaume Loubet for the conversations.

References


Computation time and memory costs for our test scenes. ss#: number of surface samples. vs#: number of volume samples. pr. time: preprocessing time, to compute volume and surface samples. Rend. time: rendering time, computing illumination for all camera samples. Total computation time is a sum of these two.

Table 3: Computation time and memory costs for our test scenes. ss#: number of surface samples. vs#: number of volume samples. pr. time: preprocessing time, to compute volume and surface samples. Rend. time: rendering time, computing illumination for all camera samples. Total computation time is a sum of these two.

Figure 12: A scene designed to underline the limitation of our algorithm on handling highly glossy reflection surfaces. Volume Material: espresso, $\alpha = [0.6186, 0.5529, 0.4911], \ell = [2.2852, 1.9550, 1.6534]$. For this material (high albedo, small mean free path), high order scattering effects dominate. Our algorithm finds volume caustics with a diffuse cup (a) and (b), but not with a glossy material (c) and (d), because it was treated as diffuse.

(a) Our, diffuse cup (b) UPBP, Reference (c) Our, glossy cup (d) UPBP, Reference
B. Wang, J. Gascuel & N. Holzschuch / Point-Based Light Transport for Participating Media with Refractive Boundaries


