Point-Based Rendering for Homogeneous Participating Media with Refractive Boundaries

Beibei Wang, Nicolas Holzschuch

Point-Based Rendering for Homogeneous Participating Media with Refractive Boundaries

Beibei Wang[1] and Nicolas Holzschuch[2]
[1]Nanjing University of Science and Technology, [2]Inria, Univ. Grenoble-Alpes, CNRS, LJK

Abstract—Illumination effects in translucent materials are a combination of several physical phenomena: refraction at the surface, absorption and scattering inside the material. 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 homogeneous translucent materials with refractive boundaries. We start by placing light samples inside the translucent material and organizing them into a spatial hierarchy. At rendering, we gather light from these samples for each camera ray. We compute separately the sample contributions for single, double and multiple scattering, and add them. We present two implementations of our algorithm: an offline version for high-quality rendering and an interactive GPU implementation. The offline version provides significant speed-ups and reduced memory footprints compared to state-of-the-art algorithms, with no visible impact on quality. The GPU version yields interactive frame rates: 30 fps when moving the viewpoint, 25 fps when editing the light position or the material parameters.

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 to 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.

Combining photon mapping with beams and paths [1], [2] provide very good results but can take a long time to converge. With some materials, the initial results are quite noisy.

Point-Based Global Illumination [3] 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 homogeneous participating media. As in all point-based methods, 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 their contributions.

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. We present two different implementations: an offline version for high-quality rendering and an interactive version, with all steps running on the GPU. The offline version provides the same quality as existing algorithms, with significant speedups and reduced memory footprint. The GPU version provides interactive framerates: 30 frames per second when changing the view point position, 25 frames per second when changing the material parameters or the light source position, as both require recomputing the light samples.

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, and describe details specific to our GPU implementation in Section 4. Low level implementation details are described in Section 5. We validate results for each step of our algorithm in Section 6 and compare with previous work and reference solution.

2 Previous Work

Subsurface Scattering: Jensen et al. [4] 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. [5] introduced the *Directional Dipole*, relaxing the assumption that incoming light is orthogonal to the material surface. D'Eon

and Irving [6] 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 canceled by the large number of scattering events. Our algorithm targets a large range of translucent materials, from almost transparent to almost opaque. Jimenez et al. [7], [8] provide fast, real-time models to compute these sub-surface scattering effects in such high-albedo materials. In comparison, our algorithm is designed to work with arbitrary materials, from low to high albedo, from almost transparent to almost opaque. We compute both single and multiple scattering effects.

Donner et al. [9] precomputed surface response as a BSSRDF 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 a more compact and accurate representation.

Accurate Single Scattering: Inside participating media with refractive boundaries, single scattering effects can produce volume caustics, with complicated shapes. Walter et al. [10] introduced a method for accurate computation of single scattering effects in participating media, computing the entry point using Newton-Raphson optimization. Holzschuch [11] 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 [12] presented an extension of the Photon Mapping algorithm for participating media. Jarosz et al. [13] 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. [1], [14] extended this idea by tracing beams inside the media rather than simply photons. Křivánek et al. [2] improved the algorithm by automatically selecting between beams, points and paths in light transport simulation, using multiple importance sampling. Bitterli and Jarosz [15] further extended the idea by tracing photon planes and volume. Our algorithm is similar in scope, but works within a Bi-Directional Path Tracing framework rather than Photon Mapping.

Point-Based Global Illumination Christensen [3] 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. Arbree et al. [16] extended the approach for subsurface scattering. Yan et al. [17] 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 [18] proposed virtual point light method for fast global illumination computation. Hašan et al. [19] introduced spherical lights to avoid singularities. Novák et al. [20] used virtual point lights and virtual ray lights for light transport inside translucent materials. In later work [21] they 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. [22] introduced the lightcuts technique to cluster the virtual lights resulting in large speed-up. In later work, they extended the domain of cuts to include light-receiver pairs [23] and extended the method to include glossy reflection and subsurface scattering [24].

TABLE 1
Notations.

Material properties	
σ_a	absorption coefficient
σ_s	scattering coefficient
$\sigma_t = \sigma_s + \sigma_a$	extinction coefficient
$\ell = 1/\sigma_t$	mean free path
$\alpha = \sigma_s/\sigma_t$	single scattering albedo
$p(\omega, \omega_t)$	phase function
g	mean cosine of phase function
η	media refractive index
f_r	bidirectional scattering distribution function
Volume samples	
$\boldsymbol{x}_v$	position
$\hat{\boldsymbol{d}}_v$	direction of incoming light
I_v	intensity of node v
k_v	density factor of node v
S_v	surface area of node v
A_v	area of the octree leaf cell
$\hat{\boldsymbol{o}}$	Unit vector with same direction as $\boldsymbol{o}$: $\hat{\boldsymbol{o}} = \boldsymbol{o}/\|\boldsymbol{o}\|$
Camera ray samples	
$d_{\max}$	maximum depth along the ray
P_k	sample point on camera ray
d_k	depth of sample point P_k
$k_{\max}$	maximum number of sample points

Interactive / Real-time Scattering Many methods target rendering of participating media without a refractive interface, for example by ray-marching through light propagation volumes (Kaplanyan et al. [25]), using volumetric shadow mapping (Delalandre et al. [26]) or computing volume caustics by rendering large number or light rays as lines (Hu et al. [27], Sun et al. [28]). These approaches exploit screen-space coherence in the absence of a refractive interface between viewpoint and translucent material. We target instead materials enclosed inside a refractive interface, which breaks the screen-space coherence, but provides more interesting effects. Sun et al. [29] discretizes the scene into voxels and trace the curved paths of photons as they travel through the volume. Their method enables interactive relighting, but its accuracy is limited by the voxel grid resolution. Walter et al. [10] method for accurate single scattering provides interactive framerates (10 fps) if the refractive interface is made of flat surfaces, but not for curved surfaces or triangles with interpolated normals.

Xu et al. [30] present a real-time homogenous translucent material editing method by approximating the multiple scattering diffuse reflectance function and single scattering attenuation function with non-uniform piecewise polynomial basis. This method can achieve real-time frame-rate for translucent material under fixed illumination. Compared with this method, ours supports edition of material properties and light position. Zhang et al. [31] present a real-time rendering method for volumetric data set by combining precomputed radiance transfer method and photon mapping. Their targets are volumetric data set, while ours are homogenous media with surface boundaries.

3 Point-Based Light Transport in participation media

In this section, we review the Radiative Transfer Equation, introduce notations, and describe our algorithm.

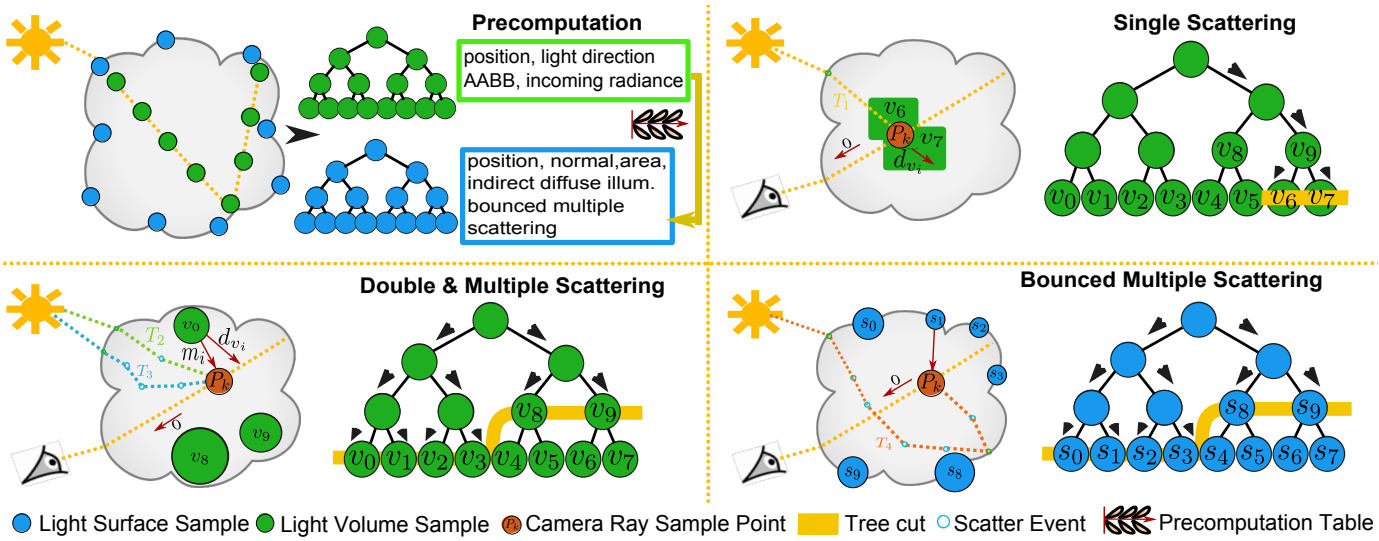


Fig. 1. Our algorithm: we begin by computing incoming light at volume and surface samples. We then compute Single-, Double- and Multiple scattering effects for each camera ray using these volume and surface samples.

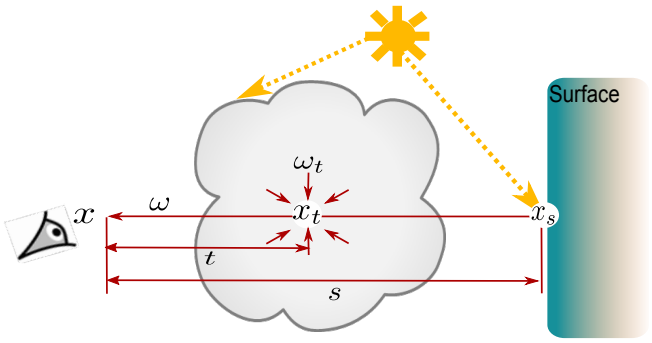


Fig. 2. Radiance that reaches a point x from direction ω as a sum of exitant radiance from the nearest surface x_s from this direction and in-scattered radiance from the medium among the whole length of the ray.

3.1 Radiative Transfer Equation

We consider a scene containing objects with translucent material. Each of these is assumed to be made of an homogeneous material, with index of refraction η, scattering coefficient σ_s, absorption coefficient σ_a and phase function $p(\hat{\mathbf{i}}, \hat{\mathbf{o}})$ (see Table 1). We note ℓ the mean-free path inside the material, with $1/\ell = \sigma_t = \sigma_s + \sigma_a$. In the remainder of this paper, we focus on a single translucent object, but the algorithm is generic enough to handle multiple objects.

Light transport within participating medium is described by the *Radiative Transfer Equation* [32], which defines the radiance that reaches a point x from direction $\boldsymbol{\omega}$ 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:

$$L(x, \boldsymbol{\omega}) = T_r(x \leftrightarrow x_s) L(x_s, \boldsymbol{\omega}) + \int_0^s T_r(x \leftrightarrow x_t) \sigma_s(x_t) L_i(x_t, \boldsymbol{\omega}) \, \mathrm{d}t, \tag{1}$$

where T_r is the transmittance, defined as:

$$T_r(x \leftrightarrow x_t) = e^{-\sigma_t \|x - x_t\|}, \tag{2}$$

s is the distance through the medium to the nearest surface at $x_s = x - s\boldsymbol{\omega}$, and $x_t = x - t\boldsymbol{\omega}$ with $t \in (0, s)$ (see Figure 2). $L(x_s, \boldsymbol{\omega})$ is the exit radiance from the nearest surface, which is governed by the *rendering equation* [33]. $L_i(x_s, \boldsymbol{\omega})$ is the in-scattering radiance at x_t from all direction $\boldsymbol{\omega}_t$ over the sphere of directions $\Omega_{4\pi}$ using the phase function p, defined as:

$$L_i(x_t, \boldsymbol{\omega}) = \int_{\Omega_{4\pi}} p(\boldsymbol{\omega}, \boldsymbol{\omega}_t) L(x_t, \boldsymbol{\omega}_t) \, \mathrm{d}\boldsymbol{\omega}_t. \tag{3}$$

3.2 Context

We place ourselves within a standard Point-Based Global Illumination (PBGI) 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 1).

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

In this paper, we focus on homogenous media with refractive boundaries. We separate between single-, double- and multiple-scattering effects, depending on the number of volume scattering events inside the translucent material (see Figure 1). *Single scattering* corresponds to a light paths 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 compute these effects separately because their appearance is different: single-scattering results in localized high-frequency volume caustics, double-scattering effects are still localized but blurrier, multiple scattering effects are mostly diffuse and present throughout the material.

Our algorithm only handles light paths with a single refraction on the camera ray, and a single refraction on the light ray. We can not handle multiple refractions caused by concave geometries.

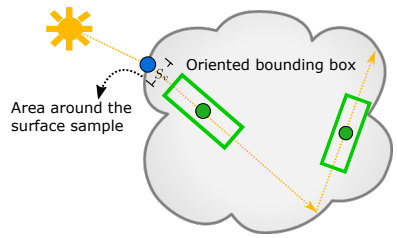


Fig. 3. Illustration for the volume samples. For each volume sample, we store the position, direction, area around its surface sample, interval along the ray and the incoming radiance.

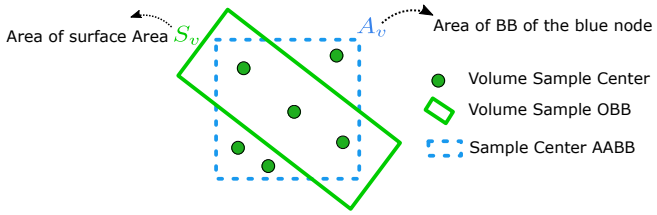


Fig. 4. Density factor: for each volume sample, we compute the ratio between the area of its leaf node A_v and the area S_v around the surface sample at the origin of the ray. A_v is larger in low density areas, smaller in high-density regions.

3.4 Pre-processing

3.4.1 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 [34]. 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 3.5.5).

Each light surface sample is defined by its position $\boldsymbol{x}_s$ and diffuse intensity I_s. We build a spatial hierarchy over these surface samples for future queries (an octree).

3.4.2 Light Surface Samples after Environment Interactions

We adapt the sampling process when the participating media is enclosed inside several layers of transparent refractive interfaces, as is common with liquids (see Figure 17): 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.

We treat the sampling differently when the participating media is enclosed by glossy reflective interface (see Figure 18). We shoot photons from the light source and importance sample the outgoing direction of glossy reflection. This is similar to the photon mapping, except we do not scatter the photons in the media. We store the photons when they interact with the media surface. For each blue noise surface sample, we average all the photons that are close to it, by averaging the direction and radiance.

3.4.3 Light Volume Samples

We then place volume samples, storing incoming light inside the translucent material (see Figure 3). 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.

For each volume sample point v, we store its position $\boldsymbol{x}_v$, incoming light direction $\boldsymbol{d}_v$, incoming radiance I_v and its volume of influence V_v, defined by the surface area around the sample point S_v and the interval along the ray, and approximated by an Oriented Bounding Box (OBB). We store both the shape of the actual OBB and the amount of space it encloses. The volumes of influence can overlap.

Surface refraction focuses the incoming rays, resulting in irregular volume samples density. Higher volume sample density corresponds to proximity with volume caustics. This allows us to compute illumination effects inside the medium with good accuracy.

We then organize these volume samples in a spatial hierarchy (an octree). We build this tree in two passes: first with a top-down traversal: we start with the bounding box of the material as the root of the octree, and subdivide regularly until there is at most 8 volume samples per cell. Second, with a bottom-up traversal, compute average position, direction, direction variance and incoming radiance for each cell, using values computed for its children.

During construction, for each volume sample v, we compute a dimensionless *density factor* k_v, depending on the ratio between the area around its surface sample S_v and the bounding box area of the volume samples in the octree leaf cell A_v (see Figure 4):

$$k_v = \left(\text{clamp}\left(\frac{A_v}{S_v}, 1, 8\right) * \text{clamp}\left(\sigma_t, 0.3, 0.8\right)\right)^2 \tag{4}$$

We use the size of the leaf node, A_v, as a proxy for sample density: the more dense the samples in an area, the smaller the leaf nodes. We increase contributions from samples in low-density areas, an conversely decrease the contributions from samples in high-density areas. Clamping avoids giving too much importance to any given individual samples.

3.5 Rendering

3.5.1 Sampling the camera ray

To render the scene, we shoot rays from the camera. They can reach the translucent object either 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 $\hat{\boldsymbol{o}}$. We then place sample points $\boldsymbol{P}_k$ 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. There are usually several camera rays per pixel, for anti-aliasing and indirect illumination computations.

We need different sampling strategies, depending on whether the camera ray has reached the participating media directly or after diffuse or glossy bounces:

- if the camera ray reaches the participating media directly or after only specular events (reflections and refractions), we need a large number of samples. We place $k_{\max}$ samples on the ray, with depth varying exponentially to place more samples near the surface (see Section 5.1).
- If the camera ray reaches the participating media after diffuse or glossy interactions, we need less samples. We sample it randomly with a smaller amount of samples.

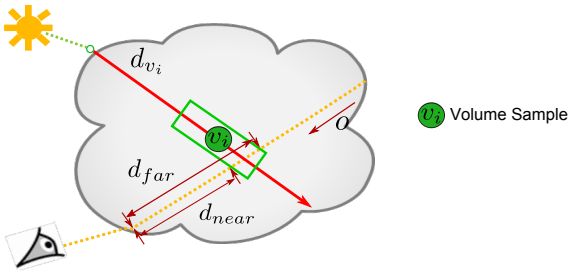


Fig. 5. Single scattering: we add the contributions from all volume samples whose OBB is intersected by the camera ray.

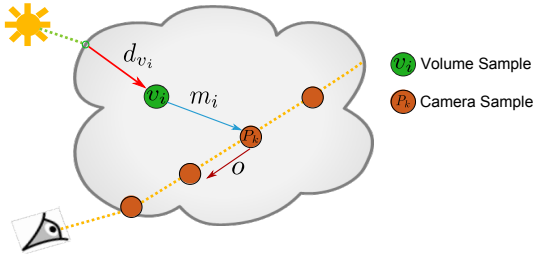


Fig. 6. Double scattering: we extract a tree-cut of volume samples and compute double scattering from each node of this tree-cut.

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 $\hat{\boldsymbol{o}}$ and a sample point $\boldsymbol{P}_k$ 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 $\boldsymbol{P}_k$.

3.5.2 Single scattering

Single scattering corresponds to light that enters the participating medium, is scattered exactly once, then leaves the participating medium. We compute it by summing the contributions of light volume samples whose OOBB is intersected by the camera ray.

We go through the hierarchy in a top-down manner, discarding nodes that are not intersected by the camera ray, until we reach the leaves. We then test the direction variance of the leaf node and the phase function value. If both the direction variance and the phase function are smaller than a threshold, we gather the contribution from the leaf node, otherwise, we gather the contribution from each volume sample in the leaf node. The contributions are scaled by the density factor k_v:

$$\text{single} = \alpha \sum_v p(\hat{\boldsymbol{o}}, \hat{\boldsymbol{d}}_v) \frac{I_v}{k_v} (\mathrm{e}^{-\sigma_t d_{\text{near}}} - \mathrm{e}^{-\sigma_t d_{\text{far}}}), \tag{5}$$

where d_{near} and d_{far} represent the two intersections with the sample volume of influence (see Figure 5).

3.5.3 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 $\boldsymbol{P}_k$ and the phase function between outgoing direction $\hat{\boldsymbol{o}}$ and the direction from $\boldsymbol{P}_k$ to the center of the node $\boldsymbol{v}_i$ (see Algorithm 1 for the full algorithm). This gives us a tree cut.

For each node in this tree cut, we compute double scattering from the volume sample $\boldsymbol{v}_i$ to $\boldsymbol{P}_k$: light that has entered the material, is scattered once at $\boldsymbol{v}_i$ in the direction of $\boldsymbol{P}_k$, reaches $\boldsymbol{P}_k$, is scattered at $\boldsymbol{P}_k$ and leaves in the direction $\hat{\boldsymbol{o}}$ (see Figure 6). We handle the weak singularities caused by $\frac{1}{r^2}$ by clamping.

$$\boldsymbol{r}_i = \boldsymbol{P}_k - \boldsymbol{v}_i, \qquad r_i = \|\boldsymbol{r}_i\|, \qquad \hat{\boldsymbol{r}}_i = \boldsymbol{r}_i / r_i,$$

$$\text{double}\,(\boldsymbol{P}_k) \quad = \quad \sigma_s^2 \sum_{v_i} \frac{\mathrm{e}^{-\sigma_t r_i} I_{v_i}}{r_i^2} p(\hat{\boldsymbol{o}}, \hat{\boldsymbol{r}}_i) p(\hat{\boldsymbol{r}}_i, \hat{\boldsymbol{d}}_{v_i}). \tag{6}$$

Algorithm 1 Octree subdivision for multiple scattering

P = camera sample position
$\hat{\boldsymbol{o}}$ = camera ray direction
C = octree cell
$C.\boldsymbol{r}$ = center of the cell
$C.\hat{\boldsymbol{d}}$ = incoming light direction
$\boldsymbol{u} \leftarrow P - C.\boldsymbol{r}$ ▷ vector joining sample and center of cell
if solidAngle$(P, C) > \varepsilon_1$ **then** subdivide
else if solidAngle$(P, C) > \varepsilon_2$ **and**
  $(p(\hat{\boldsymbol{o}}, \hat{\boldsymbol{u}}) > \pi/4$ **or** $p(\hat{\boldsymbol{u}}, C.\hat{\boldsymbol{d}}) > \pi/4))$ **then** subdivide
end if

3.5.4 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 $\boldsymbol{v}_i$, is scattered several times before reaching $\boldsymbol{P}_k$ and is scattered one last time into direction $\hat{\boldsymbol{o}}$.

We have precomputed material response in a table using Monte-Carlo simulation: we take a point light source sending photons in a single direction in an infinite medium. We simulate photon propagation in this medium, and accumulate the results. After convergence, we have a table that represents material response. To reduce storage costs, we exploit the symmetries: we store response in cylindrical coordinates $(\rho/\ell, z/\ell)$, since the problem has rotational symmetry around the direction of propagation; we index distances divided by the mean-free-path. At each point, we store outgoing radiance in spherical coordinates relative to the current frame, $L_o(\rho/\ell, z/\ell, \theta, \phi)$, giving a 4 dimension table (see Figure 7).

This table depends only on the albedo and phase function of the material. We have precomputed it for different parameter values and access the relevant table at run-time. We compress these tables to save memory (see Section 5.2).

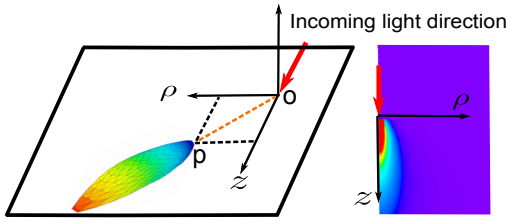


Fig. 7. Precomputed multiple scattering lobes. Left: a single lobe from our precomputed multiple scattering table. Right: total light intensity, as stored in the table, as a function of position. Material: Bumpy Sphere, $\alpha = \{0.9545, 0.6774, 0.4565\}$, $\ell = \{4.5455, 3.2258, 2.1739\}$, $g = 0.9$.

To compute multiple scattering from $\boldsymbol{v}_i$ to $\boldsymbol{P}_k$, we find the cylindrical coordinates of $\boldsymbol{P}_k$ around the $(\boldsymbol{v}_i, \boldsymbol{d}_i)$ axis, then extract the outgoing radiance:

$$\begin{aligned} z_k &= \boldsymbol{r}_i \cdot \hat{\boldsymbol{d}}_i \\ \rho_k &= \left\| \boldsymbol{r}_i - z_k \hat{\boldsymbol{d}}_i \right\| \\ \text{mult.}\,(\boldsymbol{P}_k) &= \sum_{v_i} L_o\left(\frac{\rho_k}{\ell}, \frac{z_k}{\ell}, T_{(\hat{\boldsymbol{v}}_i, \hat{\boldsymbol{d}}_i)}(\hat{\boldsymbol{o}})\right), \end{aligned} \tag{7}$$

where $T_{(\hat{\boldsymbol{v}}_i, \hat{\boldsymbol{d}}_i)}(\hat{\boldsymbol{o}})$ is the direction corresponding to $\hat{\boldsymbol{o}}$ in the frame defined by $(\hat{\boldsymbol{v}}_i, \hat{\boldsymbol{d}}_i)$.

3.5.5 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 already store diffuse illumination from the environment, as in Jensen and Buhler [34]. 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 $\boldsymbol{P}_k$, we begin by computing a tree cut over surface samples, using the same refinement criterion as in Section 3.5.3: the solid angle sustained by the node multiplied by the phase function from $\boldsymbol{P}_k$ to the center of the node. We then accumulate contributions from the surface samples s_j, multiplying the stored incoming diffuse radiance with surface BRDF f_r:

$$\boldsymbol{q}_j = \boldsymbol{P}_k - \boldsymbol{s}_j, \qquad q_j = \|\boldsymbol{q}_j\|, \qquad \hat{\boldsymbol{q}}_j = \boldsymbol{q}_j / q_j,$$

$$\text{bounced}\,(\boldsymbol{P}_k) = \sigma_s \sum_{s_j} \frac{\mathrm{e}^{-\sigma_t q_j} \boldsymbol{I}_{s_j}}{q_j^2} f_r(\hat{\boldsymbol{q}}_j) p(\hat{\boldsymbol{o}}, \hat{\boldsymbol{q}}_j), \tag{8}$$

3.5.6 Full solution

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

$$L = \text{single} + \sum_k \mathrm{e}^{-\sigma_t d_k} \left(\text{double}\,(\boldsymbol{P}_k) + \text{mult.}\,(\boldsymbol{P}_k) + \text{bounced}\,(\boldsymbol{P}_k)\right). \tag{9}$$

4 Fast GPU Implementation

We have implemented a fast version of this algorithm on the GPU, using Optix [35]. We adapted the algorithm to the specificity of the GPU for: precomputed multiple scattering (Section 4.1), hierarchical data structure storage (Section 4.2) and traversal (Section 4.3), as well as picture computation (Section 4.4).

4.1 Precomputed Multiple Scattering Table

The table storing precomputed material response to multiple scattering has to be stored in GPU memory. To reduce its memory footprint, we approximate each lobe by a lobe with symmetry of revolution around the axis joining the source point O and the current point P (see Figure 7). With this approximation, we store material response in a 3D table, indexed by $(\rho/\ell, z/\ell, \theta)$, where θ is the angle between the outgoing direction and the lobe axis. Figure 8 shows the difference between the approximated lobe representation used for GPU computations and the full representation. Enforcing lobe symmetry has a small impact on lobe direction and shape.

We repeat the computations for multiple values of α and g, and store the responses from all materials in a 5D table, for an overall cost of 600 MB. At run-time, we interpolate over α and g to extract the table corresponding to the current material.

4.2 Hierarchical Data Structure Storage

We compute illumination samples every time the lighting conditions change, and store them in a spatial hierarchy. As long as lighting conditions remain identical, we keep these illumination samples and their spatial hierarchy, and only recompute their contributions to the picture.

Individual illumination samples are stored together by groups of eight samples, in a table. For each sample, we store position, direction, incoming radiance, scaled surface area and interval length along the ray, using a structure of arrays.

We build two different hierarchical representations for single and multiple scattering. These trees have the same internal representation, but store different information at each node: for single scattering, we only store the AABB of the node. For multiple scattering, we store the average position, direction, incoming radiance, volume of influence and the depth of the node. Values for a node are the averages of the values for the children.

We store both trees as 1D buffers. Each node contains its bounding box, the index of its first child and the number of children. For compact representation, we store these information using two `float4` (see Figure 9):

- `min_childIdx` contains the $(x_{\min}, y_{\min}, z_{\min})$ part of the Bounding Box in the first three coordinates, and the index of the first child of the node in the fourth.
- `max_nChild` contains the $(x_{\max}, y_{\max}, z_{\max})$ part of the Bounding Box in the first three coordinates, and the number of children in the fourth.

Having separate data structures for single and multiple scattering data introduces some redundancy: we are storing the structure hierarchy twice. It increases data locality at each step, saving time.

4.3 Two-Steps Hierarchy Traversal

Traversing the spatial hierarchy is the main computational cost for our algorithm, both for single or multiple scattering computations. We designed a faster traversal algorithm, working in two steps:

- in a first step, we extract a coarse subtree from the hierarchy, starting from the tree root but stopping after a certain depth.
- in the second step, we restart our traversal, starting from the subtree computed at the first step.

The two-step traversal is obviously efficient for single scattering computations, where it allows us to discard a large subset of the hierarchy before the second pass. It is also useful for multiple scattering. Overall, it provides a 20 % performance improvement over the naive hierarchy traversal (see Section 6.3.6).

We further improve performance by reusing the results of the first pass between neighbouring pixels for single scattering

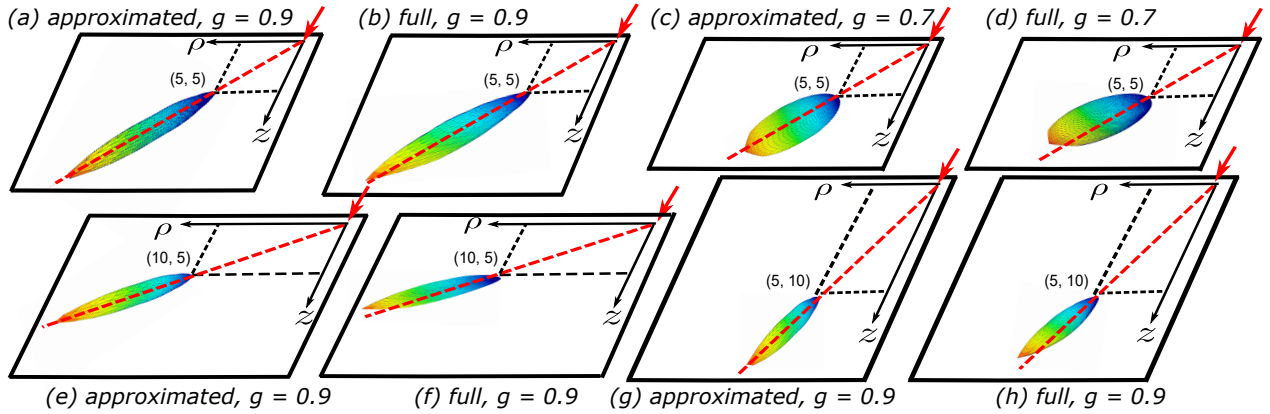


Fig. 8. Comparison of the lobe shapes between full and approximated representation. In approximated representation, we assume that the lobe is symmetric around the main axis (red dash line), while in the full representation, the peak value of the lobe is slightly off the main axis. We compare approximated and full lobe representation at location (5,5) for bumpy sphere material with g = 0.9 in (a) and (b), for bumpy sphere material with g = 0.7 in (c) and (d). We show the lobes at location (10,5) in (e) and (f), and at location (5,10) in (g) and (h) for bumpy sphere material with g = 0.9.

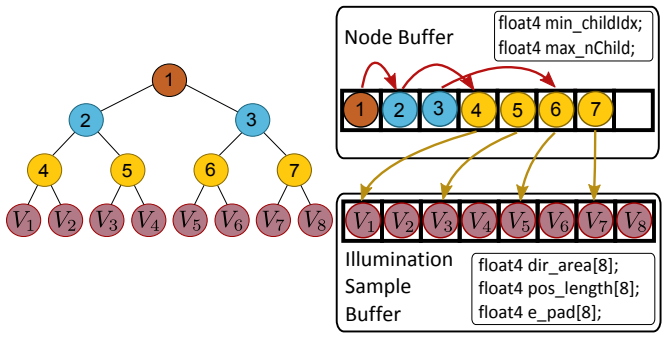


Fig. 9. Our Octree Data Structure: nodes are stored in a 1D array. Each node stores its bounding box, index to first child and number of children, encoded in two `float4`.

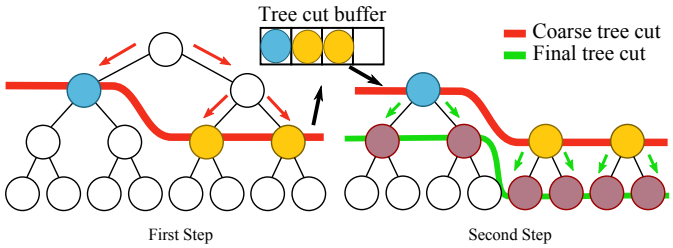


Fig. 10. Two-step traversal pipeline. In the first step, we get the coarse tree cut (red), and store the nodes in a tree cut buffer. In the second step, we continue traversing from the nodes in the tree cut buffer until the node satisfies our criterion for accuracy.

on primary rays: we compute the first pass only once for each group of 2×2 pixels, then do the second pass for each pixel independently, starting from the shared subtree. We do not apply these savings to secondary rays (rays that have bounced at least once) or when the 2×2 pixels block covers an edge of the material.

4.4 Rendering

In a first step, we trace rays from the camera through the scene. Each time a ray encounters the translucent material, we store it, with its starting point and end point, in an auxiliary buffer (see Figure 11). In a second step, we treat all the rays from this buffer, compute their single, double and multiple scattering contributions, and add these to the corresponding pixel value.

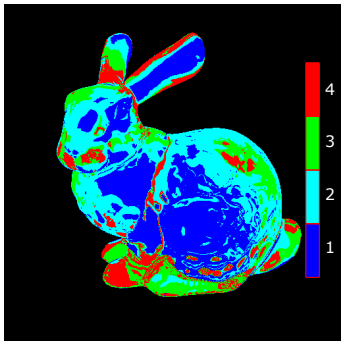


Fig. 11. Number of rays per pixel for the Bunny scene. Thinner regions of the Bunny require more rays per pixel, due to internal reflections. In thicker areas, a single ray is sufficient.

The auxiliary buffer contains all paths through the material, both primary rays (rays from the camera) and secondary rays (rays that reach the material after one or several bounces, including internal reflections in the material). Figure 11 shows the number of rays per pixel for a typical scene.

For each ray from the camera, after each bounce, we compute the attenuation caused by material absorption and BRDF reflectance, and stop tracing when the attenuation is above a certain threshold.

We add two separate optimizations:

- for *single scattering*, we found it becomes negligible compared to multiple scattering after roughly one mean-free-path. We only gather single scattering contributions over a material layer of thickness ℓ.
- for *multiple scattering*, the effects are mostly low frequency. We group neighbouring rays together, using 4×4 downsampling and interpolation.

5 Implementation Details

5.1 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 $\boldsymbol{P}_k$:

- Compute the exit point for this camera ray. This gives the maximum depth along this ray, $d_{\max}$.

- Compute $k_{\max}$ sample points $\boldsymbol{P}_k$, each defined by its depth along the ray d_k:

$$d_k = -\frac{1}{\sigma_t} \log\left(1 - \frac{\left\lfloor k(1 - e^{-\sigma_t d_{\max}}) \right\rfloor}{k_{\max}}\right). \quad (10)$$

This sampling scheme ensures that we always place $k_{\max}$ 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.

5.2 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 normalize coordinates by mean-free-path, to reuse computations.

We store particle contributions in a 2D grid, using cylindrical coordinates $(\rho/\ell, z/\ell)$ to take advantage of the symmetry of revolution around the z axis. ρ/ℓ is the distance from the z axis, and z/ℓ is the depth along this axis (see Figure 7). We use a grid with 1024×512 cells. Lobes are stored using spherical coordinates (θ, ϕ), sampled with 18 directions for θ and 36 for ϕ.

We compress this representation using a quadtree, depending on particle density. We discard nodes with no scattering events for further compression. The hierarchical representation allows adaptive compression of our precomputed multiple scattering, reducing the memory cost from 8.4 GB to approximately 60 MB. The actual compression rate depends on material properties (see Section 6.3.1).

6 Results

We have implemented our algorithm inside the Mitsuba Renderer [36], combined with an Optix previewer. We compared our algorithm against (i) photon mapping with Beam-Radiance Estimate (BRE) by Jarosz et al. [13], (ii) Bi-Directional Path Tracing (BDPT) and (iii) Unified points, beams and paths (UPBP) by Křivánek et al. [2], which we take as the reference. We used Mitsuba renderer for our algorithm and BRE, and SmallUPBP [37] 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 [11], implemented in the Mitsuba renderer. We compare our double and multiple scattering computations against VRL [20], also implemented in the Mitsuba renderer. We compare our GPU implementation with Separable Subsurface scattering (SSSS) [8].

All timings in this section are measured on a 2.67GHz Intel i7 (32 cores) with 32 GB of main memory and Nvidia Quadro M6000 GPU for real-time rendering using Optix (Cuda). Pictures rendered with the high-quality offline algorithms used a resolution of 1024×1024 pixels, except for the *Bumpy Sphere* and *Bunny*, where we used 512×512. Pictures rendered with the GPU implementation used a resolution of 512×512. We measure numerical differences between simulations using the Mean-Squared Error (MSE). We report the full computation time, including time for our algorithm as well as the rest of the picture.

All materials in our test scenes are homogenous materials, with Henyey-Greenstein phase functions and smooth refractive boundaries. Material properties were taken from Křivánek et al. [2], Narasimhan et al. [38] and Holzschuch [11] (see Table 2).

6.1 Individual Component Validation

We validate our algorithm by comparing its response for each individual component to reference solutions: Holzschuch's algorithm for single scattering [11], and UPBP by Křivánek et al. [2] for double and multiple scattering. Figure 12 shows this component comparison for the offline renderer, while Figure 13 shows the same comparison for the interactive GPU version.

Our algorithm provides a very good match with the reference for each component, at a fraction of the cost in terms of rendering time. On this specific scene, our offline algorithm takes 1 mn to compute contributions for all components together, compared to 6 h for UPBP, a 360× acceleration. The memory cost for our algorithm is also 4 times smaller than for UPBP. The interactive version is even faster, running at 30 ms per frame.

Our algorithm behaviour is quite robust with varying material properties. Fig. 14 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 ($g = 0.9$).

6.2 Comparison with previous work

6.2.1 Comparison with generic reference methods

Figures 15, 16 and 17 show a comparison between pictures computed using our two implementations and reference algorithms: Bi-Directional Path Tracing (BDPT), Photon Mapping with Beam-Radiance Estimate (BRE) and Unified Points, Beams and Paths (UPBP). We test our algorithm on a wide variety of materials, ranging from almost transparent (olive oil) to almost opaque (wax). Our offline algorithm produces pictures that are very close to the UPBP reference, while being an order of magnitude faster. BDPT can not connect paths through the refractive interface, missing single scattering effect. For a comparison with individual UPBP components, please refer to the supplemental. Total rendering time for Figure 17 was 34 mn: 18 mn for the participating media, and 16 mn for the rest of the scene.

The interactive GPU version gives pictures that are very close to the reference, while running interactively. The overall performance for the GPU version depends on material properties and environment. For simple scenes, our algorithm runs in less than 30 ms (more than 30 fps), allowing viewpoint edition in real-time. Adding a glass enclosure has a performance cost, but our algorithm is still interactive. We do not simulate surface caustics in our GPU implementation, as it would significantly degrade the rendering time, only volume caustics.

Table 3 gives computation times, memory costs and MSE between reference pictures and our pictures, for both the offline and interactive versions of our algorithm. This quantitative data confirms the qualitative visual impression: the pictures generated by our algorithm are very close to the reference images.

Our algorithm also handles glossy reflections in light paths (see Figure 18). We obtain volume caustics caused by the glossy reflection on the surface of the material.

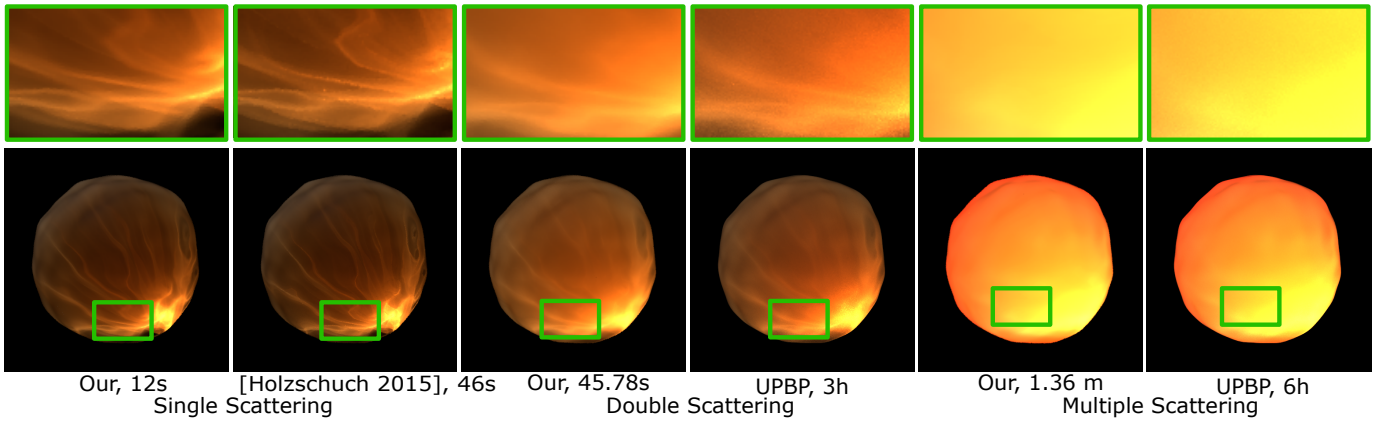
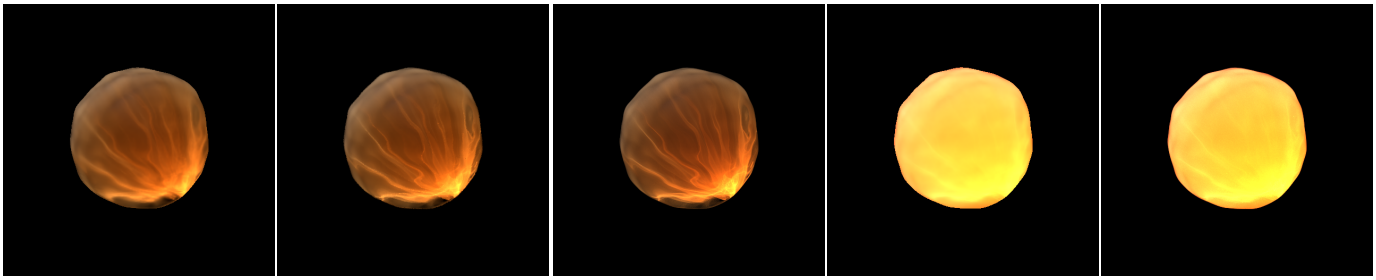


Fig. 12. Individual component validation on the Bumpy Sphere scene, high-quality offline rendering.



(a) Ours, 12.7 ms *(b) [Holzschuch 2015], 31 s* *(c) UPBP, Reference, 3h* *(d) Ours, 29.1 ms* *(e) UPBP, Reference, 1 h*

Fig. 13. Individual component validation on the Bumpy Sphere scene, interactive GPU implementation. (a) to (c): single scattering only. (d), (e): full solution.

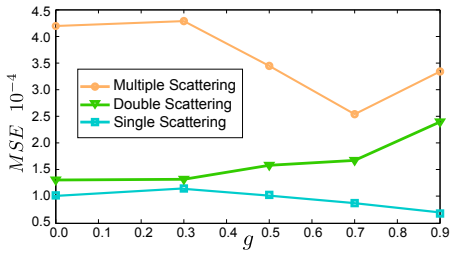


Fig. 14. Mean Square Error of single/double/multiple scattering as a function of g for the Bumpy Sphere Scene.

TABLE 2
Material parameters, memory cost after compression and precomputation times with 50 M particles, 8256 lobes, 36×18 directions for each lobe.

Name	α			ℓ			g	mem.	time
	R	G	B	R	G	B		MB	s
Oil	0.0042	0.4535	0.0995	9.7087	11.6279	2.7397	0.9	52.4	14
Wax	0.9803	0.9615	0.7500	0.6536	0.6250	0.5882	0.8	63.8	172
Milk	0.9999	0.9997	0.9991	0.8422	0.7521	0.6848	0.7	67.9	212
Skin	0.9584	0.8380	0.6778	1.2953	0.9524	0.6711	0.0	36.2	211
BmpS.	0.9545	0.6774	0.4565	4.5455	3.2258	2.1739	0.7	71.9	101

6.2.2 *Comparison with scattering-specific methods*

We compare our algorithm with methods specifically designed for rendering participating media: Virtual Ray Lights (VRL) [20], precomputed multiple scattering [39], Dipole approximation and Separable Sub-Surface Scattering (SSSS) [8].

Figure 19 shows a comparison between our algorithm and Virtual Ray Lights (Novák et al. [20]), for double and multiple scattering. With equal time, our method provides higher quality than VRL for both double and multiple scattering; the advantage is especially striking for multiple scattering. We ignore the visibility test between the lights rays and the camera rays both for VRL and our algorithm.

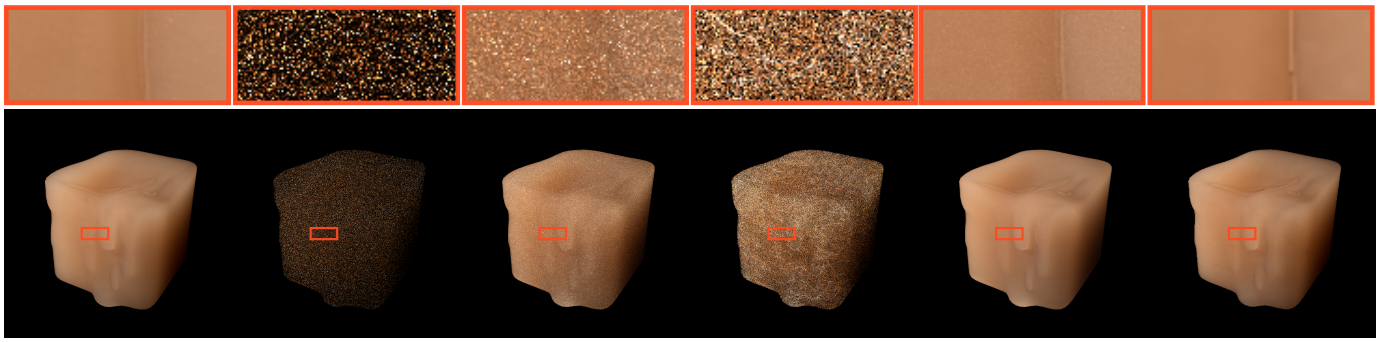
Moon et al. [39] proposed a method for precomputed multiple scattering, based on concentric spherical shells. Figure 20 shows a comparison between multiple scattering computed using our table and multiple scattering computed using their method, with different settings. In this picture, we only show the multiple scattering component, with no bounces on the surface. Using the same amount of memory, Moon et al. [39] method blurs the variations in multiple scattering. Increasing the memory allocated, they get results that are visually similar to ours. Figure 20(e) shows a visual comparison between the internal representation of multiple scattering for Moon et al. [39] and our method. Their method contains numerous artefacts, caused by the low resolution in the angular domain.

Figure 21 shows a comparison between our algorithm and algorithms designed for high-albedo sub-surface scattering: dipole approximation and Separable Sub-Surface Scattering (SSSS) [8]. Both methods produce pictures that are quite different from the reference, our algorithm being much closer. Timings for the dipole method correspond to a CPU implementation, so both methods are much faster than the equivalent for our algorithm; none of them has the option to increase quality at the expense of computation time.

6.3 Performance and Timings

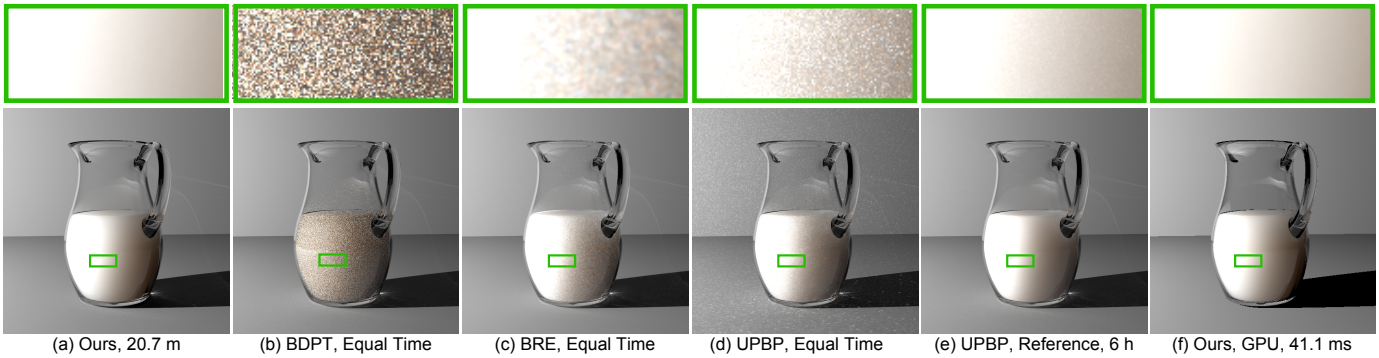
6.3.1 *Precomputed Multiple Scattering*

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



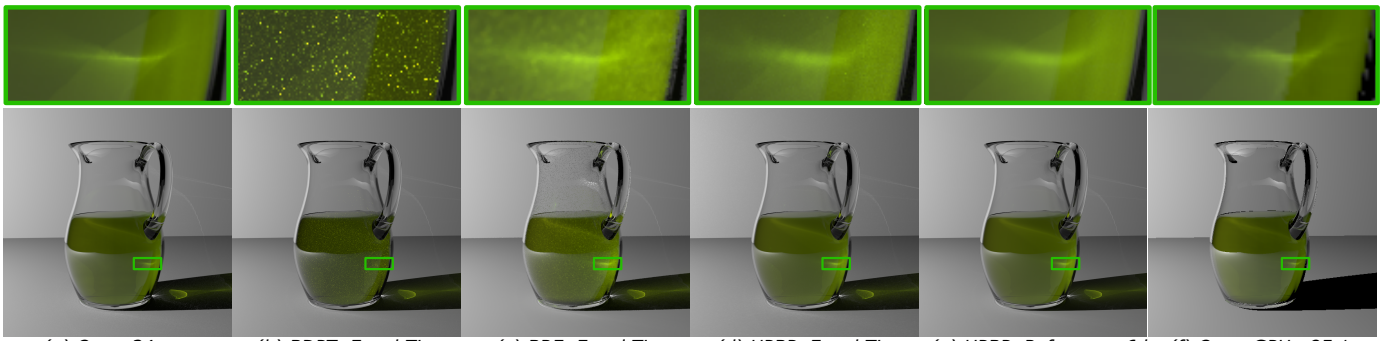
(a) Ours. 1.1 m (b) BDPT, Equal Time (c) BRE, Equal Time (d) UPBP, Equal Time (e) UPBP, Reference. 6 h (f) Ours, GPU, 15.5 ms

Fig. 15. Material: wax. For this material, with a large albedo α and a small mean free path ℓ, multiple scattering effects dominate.



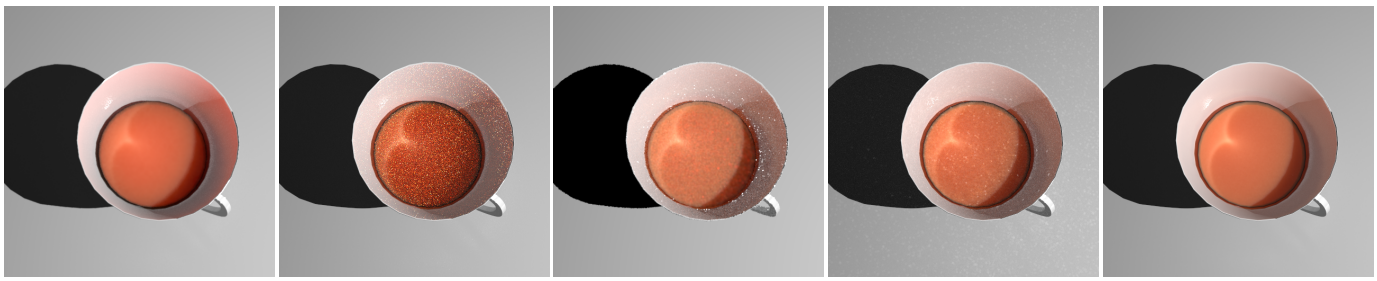
(a) Ours, 20.7 m (b) BDPT, Equal Time (c) BRE, Equal Time (d) UPBP, Equal Time (e) UPBP, Reference, 6 h (f) Ours, GPU, 41.1 ms

Fig. 16. Material: milk. For this material, with a very large albedo α and a small mean free path ℓ, multiple scattering effects dominate.



(a) Ours, 34 m (b) BDPT, Equal Time (c) BRE, Equal Time (d) UPBP, Equal Time (e) UPBP, Reference, 6 h (f) Ours, GPU, 85.1 ms

Fig. 17. Material: olive oil. For this material with low albedo α and large mean-free-path ℓ, low-order scattering effects dominate.



(a) Ours, 7.9 m (b) BDPT, Equal Time (c) BRE, Equal Time (d) UPBP, Equal Time (e) UPBP, Reference, 3 h

Fig. 18. Inside Material: bumpy sphere material with g = 0.0, and scale = 5. Note the caustics caused by light reflecting on the glossy cup surface. For this material, with a large albedo and a small mean free path, multiple scattering effects dominate. Outside Material: Phong model, specular exponent: 800.

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 (computing volume and surface samples). Rend. time: rendering time. Total computation time is a sum of these two.

		UPBP		Our Algorithm (offline version)						Error	Our Algorithm (GPU version)								Error
scene	max path.	mem. (GB)	rend. time (h)	ss# (K)	vs# (M)	ε_1	mem. (GB)	pr. time (s)	rend. time (m)	*MSE*	ss# (K)	vs# (K)	mem. (MB)	cam. (ms)	singl. (ms)	mul. (ms)	others. (ms)	total. (ms)	*MSE*
Oil	11	16.25	6	192	3.8	0.5	1.52	33	34	1.1e-3	9.7	153.3	607	10.2	42.8	18.2	13.9	85.1	2.8e-3
Wax	50	13.41	3	65.95	0.38	0.2	0.82	6	1.1	2.9e-4	65.9	323.8	601	1.5	1.9	6.1	6.0	15.5	7.1e-4
Milk	50	18.57	6	308	1.1	0.2	0.96	17	20.7	1.3e-3	77.1	930.2	672	10.5	2.5	14.7	13.4	41.1	2.9e-3
BumpS.	50	6.75	6	118	2.96	0.2	1.73	11	0.97	3.2e-4	14.8	103.7	577	0.88	13.84	6.7	7.68	29.1	6.6e-4

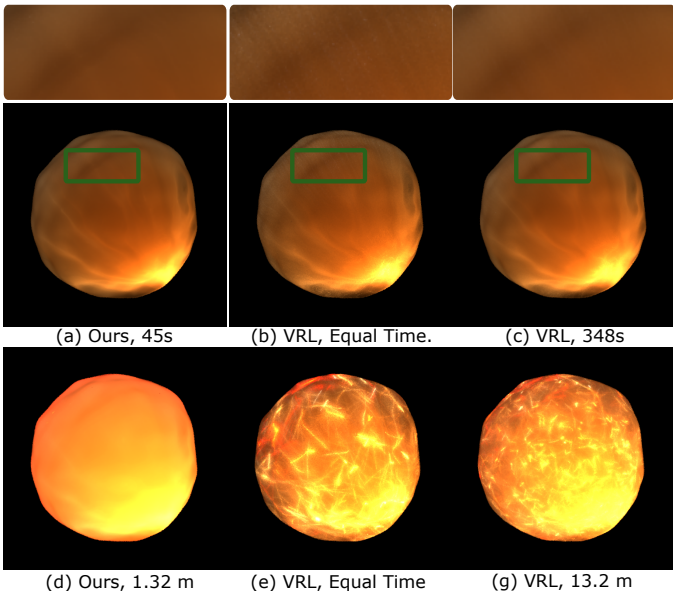


Fig. 19. Comparison with Virtual Ray Lights on the Bumpy Sphere scene. Top row: double scattering. Bottom row: multiple scattering. Our algorithm always provides pictures with better quality. The advantages are striking for multiple scattering.

TABLE 4
Editing material properties or light source position adds extra computation costs, to update the data structures.

	Material Editing Time				Light Editing			
scene	table. (ms)	v. tree. (ms)	s. tree (ms)	total. (ms)	vol. (ms)	v. tree. (ms)	s. tree. (ms)	total. (ms)
Oil	0.04	0.28	12.5	12.8	4.4	11.2	12.7	28.3
Wax	0.06	0.16	4.5	4.72	6.3	8.4	4.4	19.1
BumpS.	0.06	0.18	7.0	7.24	3.2	10.1	6.5	19.8

scenes (full representation). Computations are faster for materials with low albedo, as particles are more likely to be absorbed at each event. Memory cost after compression depend mostly on the anisotropy of the phase function: isotropic materials need less memory. The storage cost for uncompressed multiple scattering data would be 8.4 GB, so our compression algorithm reduces data size by two orders of magnitude.

6.3.2 Performance for Rendering

Table 3 gives computation time, memory cost and Mean Square Error for all our test scenes, for both offline CPU and interactive GPU implementations. Compared to reference solution, our CPU implementation gives a speed-up of 11× to 360×, with negligible differences. Memory costs are between 5× to 10× smaller. The GPU implementation requires between 15 ms and 85 ms, depending on scene complexity. The approximations introduced to make the algorithm faster have a small impact on accuracy, roughly doubling the MSE.

We report computation times for the entire scenes, with both the translucent material and its environment. On the milk and oil scene, a significant part of the computation time is related to the environment.

Figure 22 shows the computation time cost for each ray, in the CPU implementation, for each component on the Bumpy sphere scene as a function of phase function anisotropy g, with a constant number of volume samples. The cost for single scattering stays roughly constant. For double and multiple scattering, computation cost increases linearly with g, as the subdivision threshold ε_2 in Algorithm 1 depends on g. The cost per ray is independent of the albedo and mean-free-path of the materials.

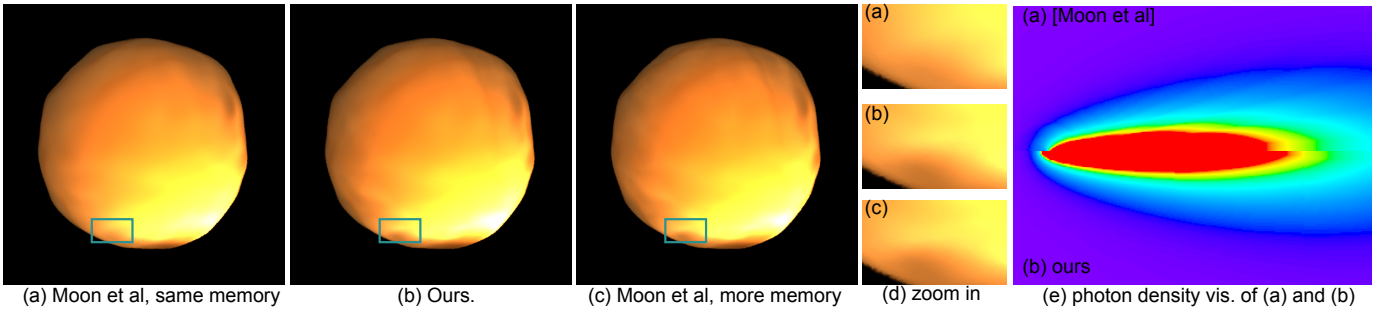
Figure 23 shows the rendering time cost and the error as a function of the number of Surface Samples. We use the number of surface samples as a proxy for the total number of samples, as volume samples are determined by surface samples. The impact on rendering time and computation error is non-linear: for a very small number of surface samples (7 K), we have both large computation time and large error. As we increase the number of samples, both computation time and error decrease, until we reach a sweet spot (approximately 66 K surface samples). Above this threshold, computation time increases, with no impact on quality. For a small number of samples, the tree hierarchy does not work: each leaf node is so large spatially that we have to access every single volume sample. For a large number of samples, we still have to compute the samples in the preprocessing stage, but they are not used in rendering, hence the lack of effect on quality.

Figure 24 shows the rendering time cost and the error as a function of Solid Angle Threshold ε_1. As we decrease the threshold, quality increases along with rendering time. A sweet spot seems to be $\varepsilon_1 \approx 0.1$.

6.3.3 Interactive Scene Editing

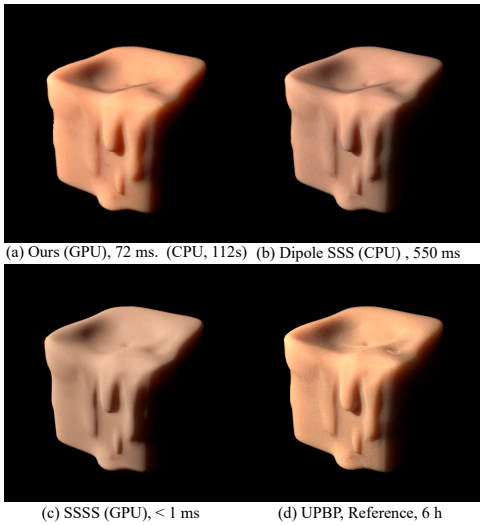
Our GPU implementation is especially fast when we only edit the viewpoint (see Table 3). Editing material properties or light position requires going through the precomputation steps: overall rendering is slower, but still interactive (see Table 4 and the companion video).

Editing material properties involves two things: first, extracting the 3D multiple scattering precomputed table from the 5D table we have for all materials, which takes less than 0.1 ms. Second, recomputing the illumination values for the volume and surface samples, taking 5 to 15 ms. Without an environment, we edit the



(a) Moon et al, same memory (b) Ours. (c) Moon et al, more memory (d) zoom in (e) photon density vis. of (a) and (b)

Fig. 20. Comparison with Moon et al. [39] on Bumpy Sphere Scene with $g = 0.95$ (unbounced multiple scattering only). Using the same amount of memory (a), their method blurs multiple scattering. Increasing memory costs (c), they get similar results to ours. The precomputed table (e) shows multiple artefacts due to the low angular resolution.



(a) Ours (GPU), 72 ms. (CPU, 112s) (b) Dipole SSS (CPU) , 550 ms

(c) SSSS (GPU), < 1 ms (d) UPBP, Reference, 6 h

Fig. 21. Comparison with Dipole SSS and SSSS on candle scene. Material: skin (scale = 4).

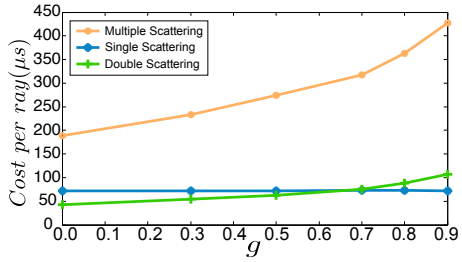


Fig. 22. Cost per ray of single/double/multiple scattering as a function of g for the Bumpy sphere scene.

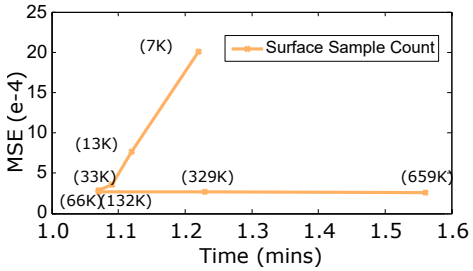


Fig. 23. Performance and error, depending on the number of Surface Samples. When the number of surface samples is too low, error and computation time are high. When it is too high, computation time increases without impact on error. The sweet spot is around 66 K surface samples.

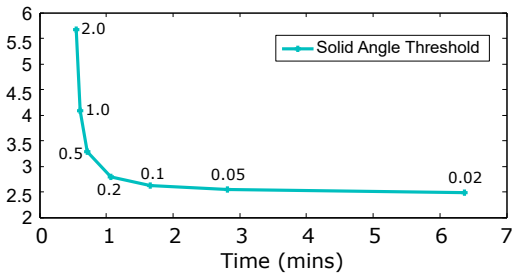


Fig. 24. Performance and error, depending on the Solid Angle Threshold for refinement (Wax Scene). Computation time and picture quality increase when we decrease the refinement threshold. A good compromise in terms of quality/computation time is $\varepsilon_1 \approx 0.1$.

material and render the result in less than 35 ms overall, or 27 frames per second.

Editing light source position requires tracing new rays inside the material and recomputing volume samples, then updating the volume and surface tree hierarchies. This is done on the GPU and adds an extra 20 to 30 ms to the computation time. It is a one-off computation, done only if the light source is edited. While it has an impact on performance, for scenes without an environment we can edit the light source and render the result in less than 50 ms, or 20 frames per second.

6.3.4 *Performance Analysis for GPU Implementation*

Figure 25 shows how the performance of our GPU implementation depend on different parameters: screen resolution, number of illumination samples, scene complexity. Unsurprisingly, computation time increases linearly with the number of pixels in the picture (the square of the image size). This effect is visible on all three steps of the algorithm, but the impact is stronger on ray-tracing, which includes computing the path buffer and rendering the rest of the picture. For very low resolutions (below 256^2) other effects dominate in scattering computations.

Computation time also depends strongly on the number of illumination samples. For multiple scattering, computation time increases linearly with the number of points, until we reach a limit. Beyond this limit, we keep extracting the same treecut, so new sample points have no impact. For single scattering, we always use the leaves of the volume samples hierarchy, so computation time increases linearly.

Since our algorithm is mostly image-based, rendering costs are roughly independent from scene complexity. Fig. 25 also shows the time per frame as a function of the number of polygons in the scene. All three steps have roughly constant computation time, independently of the complexity. The only exception is the cost of ray-tracing, for scenes below 500 K triangles.

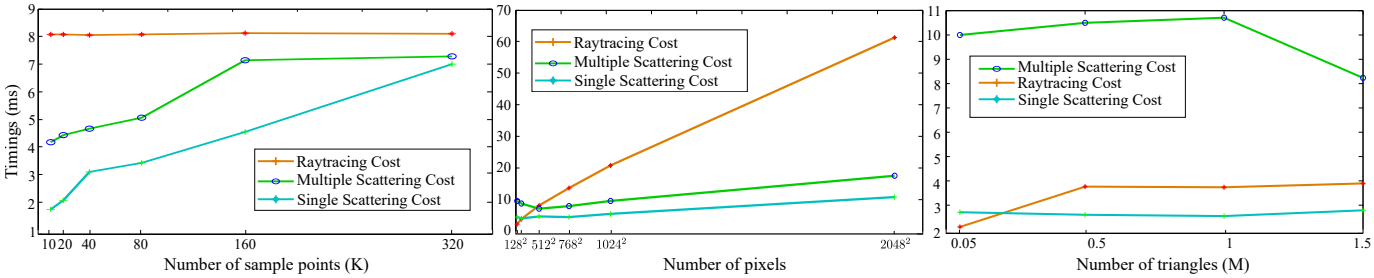


Fig. 25. Performance of the GPU implementation, over varying parameters: points, pixels and scene complexity.

6.3.5 Multiple Bounces

Performance for our interactive GPU implementation depends on the number of specular bounces we simulate inside the material. These can either be bounces along the ray from the camera, or from the light source. Figure 26 shows the visual impact of increasing the number of bounces: several phenomena only appear when we simulate multiple bounces both from the light and the camera. The effect is spectacular inside the ear, but also appears elsewhere in the scene.

Each added bounce comes with a performance cost: the full version, with 4 bounces on the path from the light source and 3 from the camera, is 6× slower than the simple version, with no internal reflection. Performance cost is sub-linear since we discard rays depending on their importance. Figure 11 shows the number of rays per pixel for 3 bounces on the camera ray.

6.3.6 Two-Pass Hierarchy Traversal

Our two-step hierarchy traversal method reduces the time for traversal by 30 %. The gain depends on the stopping depth used in the first step. Fig. 27 (a) shows the time for each step and the overall computation time, compared to the naive implementation, as a function of the stopping depth for the first step. Depth is measured relative to the leaves, so depth = 0 means a first step that descends through the entire hierarchy. The cost for the second step is then null, and the performance is equal to that of the direct implementation. Stopping the first traversal higher in the hierarchy reduces the cost of the first traversal and increases that of the second pass. Best performance is when the first step produces a set of trees of depth 3, with computation time reduced by 15 %.

Sharing the result of the first traversal over 2×2 tiles of neighbouring pixels for single scattering further improves performance (see Fig. 27 (b)). For a stopping depth of 3, computation time is reduced by 30 %.

7 Conclusion

We have presented a new method for computing light transport in participating media with refractive boundaries. Our algorithm begins by computing volume light samples inside the participating media, organizing 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. We have both an offline version of the algorithm, for high-quality pictures, and a GPU-based interactive version.

Both versions perform well for a large range of materials, from low albedo to high albedo, and for isotropic to highly anisotropic. We include indirect illumination from the scene.

Results from our algorithm are comparable to the reference solution, and include both low-order scattering and high-order scattering. Our offline implementation is an order of magnitude faster than reference solutions. Our interactive implementation is more accurate than existing fast approximations and allow interactive scene edition (light source, camera, object and material).

The main limitation for our algorithm is that we assume homogeneous materials. Extension to heterogeneous materials, with spatially varying scattering properties, will require future work.

Another avenue for future work for the GPU implementation is reducing the cost of the tree-cut buffer wit improved node encoding.

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 [3], [40], [41].

References

[1] W. Jarosz, D. Nowrouzezahrai, I. Sadeghi, and H. W. Jensen, "A comprehensive theory of volumetric radiance estimation using photon points and beams," *ACM Trans. Graph.*, vol. 30, pp. 5:1–5:19, Jan. 2011.

[2] J. Křivánek, I. Georgiev, T. Hachisuka, P. Vévoda, M. Šik, D. Nowrouzezahrai, and W. Jarosz, "Unifying points, beams, and paths in volumetric light transport simulation," *ACM Trans. Graph.*, vol. 33, pp. 1–13, Aug. 2014.

[3] P. Christensen, "Point-based approximate color bleeding," Tech. Rep. 08-01, Pixar Technical Notes, 2008.

[4] H. W. Jensen, S. Marschner, M. Levoy, and P. Hanrahan, "A practical model for subsurface light transport," in *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 2001)*, August 2001.

[5] J. R. Frisvad, T. Hachisuka, and T. K. Kjeldsen, "Directional dipole model for subsurface scattering," *ACM Trans. Graph.*, vol. 34, pp. 5:1–5:12, Dec. 2014.

[6] E. D'Eon and G. Irving, "A quantized-diffusion model for rendering translucent materials," *ACM Trans. Graph. (proc. Siggraph)*, vol. 30, pp. 56:1–56:14, July 2011.

[7] J. Jimenez, V. Sundstedt, and D. Gutierrez, "Screen-space perceptual rendering of human skin," *ACM Trans. Appl. Percept.*, vol. 6, pp. 23:1–23:15, Oct. 2009.

[8] J. Jimenez, K. Zsolnai, A. Jarabo, C. Freude, T. Auzinger, X.-C. Wu, J. von der Pahlen, M. Wimmer, and D. Gutierrez, "Separable subsurface scattering," *Comput. Graph. Forum*, vol. 34, p. 188197, Sept. 2015.

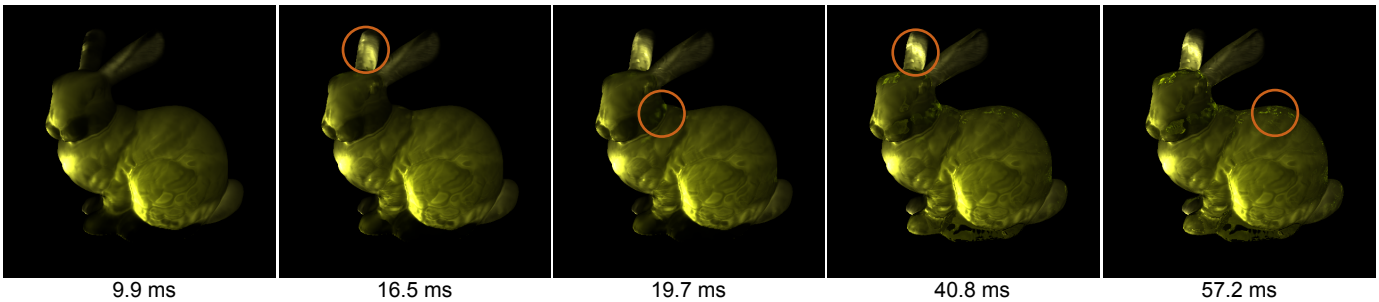
[9] C. Donner, J. Lawrence, R. Ramamoorthi, T. Hachisuka, H. W. Jensen, and S. Nayar, "An empirical bssrdf model," *ACM Trans. Graph. (proc. Siggraph)*, vol. 28, pp. 30:1–30:10, July 2009.

[10] B. Walter, S. Zhao, N. Holzschuch, and K. Bala, "Single scattering in refractive media with triangle mesh boundaries," *ACM Trans. Graph. (Proc. Siggraph)*, vol. 28, Aug. 2009.

[11] N. Holzschuch, "Accurate computation of single scattering in participating media with refractive boundaries.," *Comput. Graph. Forum*, vol. 34, no. 6, pp. 48–59, 2015.

[12] H. W. Jensen and P. H. Christensen, "Efficient simulation of light transport in scenes with participating media using photon maps," in *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pp. 311–320, 1998.

[13] W. Jarosz, M. Zwicker, and H. W. Jensen, "The beam radiance estimate for volumetric photon mapping," *Computer Graphics Forum (Proceedings of Eurographics)*, vol. 27, p. 557566, Apr. 2008.



9.9 ms | 16.5 ms | 19.7 ms | 40.8 ms | 57.2 ms

(a) Light Depth:1, Cam. Depth:1 (b) Light Depth:3, Cam. Depth:1 (c) Light Depth:4, Cam. Depth:1 (d) Light Depth:4, Cam. Depth:2 (e) Light Depth:4, Cam. Depth:3

Fig. 26. Increasing the number of bounces with our algorithm has an impact on performance, but is necessary to render several effects. Material: oil.

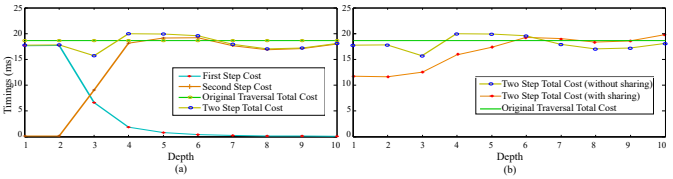


Fig. 27. (a)Performance for the two-step hierarchy traversal, as a function of stopping depth for the first step.(b) Computation time when sharing result of first traversal between neighbouring pixels, as a function of stopping depth for the first step.

[14] W. Jarosz, D. Nowrouzezahrai, R. Thomas, P.-P. Sloan, and M. Zwicker, "Progressive photon beams," *ACM Trans. Graph. (proc. SIGGRAPH Asia)*, vol. 30, no. 6, 2011.

[15] B. Bitterli and W. Jarosz, "Beyond points and beams: Higher-dimensional photon samples for volumetric light transport," *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, vol. 36, July 2017.

[16] A. Arbree, B. Walter, and K. Bala, "Single-pass scalable subsurface rendering with lightcuts," *Computer Graphics Forum (Proc. Eurographics 2008)*, vol. 27, no. 2, pp. 507–516, 2008.

[17] L.-Q. Yan, Y. Zhou, K. Xu, and R. Wang, "Accurate Translucent Material Rendering under Spherical Gaussian Lights," *Computer Graphics Forum*, 2012.

[18] A. Keller, "Instant radiosity," in *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, pp. 49–56, 1997.

[19] M. Hašan, J. Křivánek, B. Walter, and K. Bala, "Virtual spherical lights for many-light rendering of glossy scenes," *ACM Trans. Graph.*, vol. 28, pp. 143:1–143:6, Dec. 2009.

[20] J. Novák, D. Nowrouzezahrai, C. Dachsbacher, and W. Jarosz, "Virtual ray lights for rendering scenes with participating media," *ACM Trans. Graph. (proc. Siggraph)*, vol. 31, July 2012.

[21] J. Novák, D. Nowrouzezahrai, C. Dachsbacher, and W. Jarosz, "Progressive virtual beam lights," *Computer Graphics Forum (Proceedings of EGSR)*, vol. 31, no. 4, 2012.

[22] B. Walter, S. Fernandez, A. Arbree, K. Bala, M. Donikian, and D. P. Greenberg, "Lightcuts: A scalable approach to illumination," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 1098–1107, 2005.

[23] B. Walter, A. Arbree, K. Bala, and D. P. Greenberg, "Multidimensional lightcuts," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 1081–1088, 2006.

[24] B. Walter, P. Khungurn, and K. Bala, "Bidirectional lightcuts," *ACM Trans. Graph.*, vol. 31, no. 4, pp. 59:1–59:11, 2012.

[25] A. Kaplanyan and C. Dachsbacher, "Cascaded light propagation volumes for real-time indirect illumination," in *Symposium on Interactive 3D Graphics and Games*, pp. 99–107, 2010.

[26] C. Delalandre, P. Gautron, J.-E. Marvie, and G. François, "Transmittance function mapping," in *Symposium on Interactive 3D Graphics and Games*, pp. 31–38, 2011.

[27] W. Hu, Z. Dong, I. Ihrke, T. Grosch, G. Yuan, and H.-P. Seidel, "Interactive volume caustics in single-scattering media," in *Symposium on Interactive 3D Graphics and Games*, pp. 109–117, 2010.

[28] X. Sun, K. Zhou, S. Lin, and B. Guo, "Line space gathering for single scattering in large scenes," *ACM Trans. Graph. (proc. Siggraph)*, vol. 29, pp. 54:1–54:8, July 2010.

[29] X. Sun, K. Zhou, E. Stollnitz, J. Shi, and B. Guo, "Interactive relighting of dynamic refractive objects," *ACM Trans. Graph. (proc. Siggraph)*, vol. 27, no. 3, pp. 1–9, 2008.

[30] K. Xu, Y. Gao, Y. Li, T. Ju, and S. min Hu, "Real-time homogenous translucent material editing," in *in Eurographics*, 2007.

[31] Y. Zhang, Z. Dong, and K.-L. Ma, "Real-time volume rendering in dynamic lighting environments using precomputed photon mapping," *IEEE Transactions on Visualization & Computer Graphics*, vol. 19, pp. 1317–1330, 2013.

[32] S. Chandrasekhar, *Radiative transfer*. New York: Dover publications, 1960.

[33] J. T. Kajiya, "The rendering equation," *SIGGRAPH Comput. Graph.*, vol. 20, no. 4, pp. 143–150, 1986.

[34] H. W. Jensen and J. Buhler, "A rapid hierarchical rendering technique for translucent materials," *ACM Trans. Graph.*, vol. 21, pp. 576–581, July 2002.

[35] S. G. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison, and M. Stich, "Optix: A general purpose ray tracing engine," *ACM Trans. Graph. (proc. Siggraph)*, vol. 29, pp. 66:1–66:13, July 2010.

[36] W. Jakob, "Mitsuba renderer." http://www.mitsuba-renderer.org/, 2010.

[37] J. Křivánek, "SmallUPBP." http://www.smallupbp.com/, 2014.

[38] S. G. Narasimhan, M. Gupta, C. Donner, R. Ramamoorthi, S. K. Nayar, and H. W. Jensen, "Acquiring scattering properties of participating media by dilution," *ACM Trans. Graph.*, vol. 25, pp. 1003–1012, July 2006.

[39] J. T. Moon, B. Walter, and S. R. Marschner, "Rendering Discrete Random Media Using Precomputed Scattering Solutions," in *Rendering Techniques* (J. Kautz and S. Pattanaik, eds.), The Eurographics Association, 2007.

[40] T. Ritschel, T. Engelhardt, T. Grosch, H.-P. Seidel, J. Kautz, and C. Dachsbacher, "Micro-rendering for scalable, parallel final gathering," *ACM Trans. Graph. (Proc. SIGGRAPH Asia 2009)*, vol. 28, no. 5, 2009.

[41] B. Wang, X. Meng, and T. Boubekeur, "Wavelet point-based global illumination," *Computer Graphics Forum (Special Issue on EGSR 2015)*, vol. 34, no. 4, pp. 143–154, 2015.

PLACE PHOTO HERE

Beibei Wang is an Associate Professor at Nanjing University of Science and Technology. She received her PhD from Shandong University in 2014 and visited Telecom ParisTech from 2012 to 2014. She worked as a Postdoc in Inria from 2015 to 2017. She joined NJUST in March 2017. Her research interests include rendering and game development.

PLACE PHOTO HERE

Nicolas Holzschuch is a Senior Researcher at INRIA Grenoble Rhône-Alpes, and the scientific leader of the MAVERICK research team. He received his PhD from Grenoble University in 1996 and his Habilitation in 2007. He joined INRIA in 1997. His research interests include photorealistic rendering and real-time rendering, with an emphasis on material models and participating media.