# Interactive Simulation of Scattering Effects in Participating Media Using a Neural Network Model

Liangsheng Ge, Beibei Wang, Lu Wang, Xiangxu Meng and Nicolas Holzschuch

Rendering participating media is important to the creation of photorealistic images. Participating media has a translucent aspect that comes from light being scattered inside the material. For materials with a small mean-free path, multiple scattering effects dominate. Simulating these effects is computationally intensive, as it requires tracking a large number of scattering events inside the material. Existing approaches precompute multiple scattering events inside the material and store the results in a table during rendering time, this table is used to compute the scattering effects. While these methods are faster than explicit scattering computation, they incur higher storage costs. In this paper, we present a new representation for double and multiple scattering effects that uses a neural network model. The scattering response from all homogeneous participating media is encoded into a neural network in a preprocessing step. At run time, the neural network is then used to predict the double and multiple scattering effects. We demonstrate the effects combined with Virtual Ray Lights (VRL), although our approach can be integrated with other rendering algorithms. Our algorithm is implemented on a GPU. Double and multiple scattering effects for the entire participating media space are encoded using only 23.6 KB of memory. Our method achieves a rendering times of 50 ms per frame in typical scenes and provides results almost identical to the reference.

**Index Terms**—Participating Media, Multiple Scattering, Real-time, Neural Network

## 1 Introduction

Many materials, such as milk or wax, exhibit so-called scattering effects; incoming light enters the material and is scattered inside, giving a translucent aspect. Rendering these effects is computationally intensive, as it requires simulating a large number of events.

A full computation using ray-tracing or photon mapping is expensive, even with accelerating methods such as Virtual Ray Lights [1] or the state-of-the-art method Unified Points, Beams and Paths (UPBP) [2]. The dipole approximation [3] is fast, but involves too much approximation of material behavior. Precomputing the material response for multiple scattering [4], [5], [6] integrates well with existing rendering algorithms, allowing separate computation for single- and double- scattering along with fast computation for multiple scattering. The main issue with these methods concerns efficient storage for the precomputed multiple scattering data.

In this paper, we present a method to encode multiple scattering effects using a neural network model for the entire participating media space. We treat the multiple scattering response as a six-dimensional function: two dimensions for the material parameters (albedo $\alpha$ and anisotropy $g$), two dimensions for the spatial position, and two dimensions for the outgoing direction relative to the incoming direction. A straightforward approach of this kind would involve storing the entire space of materials in a single table; such a table, however would exceed 50 GB, which is impractical for rendering, especially at interactive rates. To solve this issue, we treat it as a regression problem and train a neural network model to learn the multiple scattering function. Regarding storage memory, our algorithm provides far more compact representation (only 23.6 KB, 11.8 KB for multiple scattering and 11.8 KB for double scattering) compared to explicitly storing the multiple and double scattering response [4]. For rendering, we reconstruct an explicit 4D table for each material using this network. This 4D table costs about 20 MB (including both double and multiple scattering); while more expensive than the neural network model, it is already compact enough to fit on the GPU. We then use the reconstructed table for multiple scattering rendering in a GPU-friendly pipeline. The neural network provides pictures that are identical to those produced by explicitly storing the material response, with a cost of only 50 ms per frame. In this paper, we focus on dense media, where multiple scattering effects dominate.

We review related work in the next section. We then describe our algorithm in Section 3, with implementation details presented in Section 4. In Section 5, we compare our method with previous works and reference solutions. Finally, we conclude this paper in Section 6.


- *L. Ge, L. Wang and X. Meng are with School of Software, Shandong University, 250101, Jinan, China. L. Wang is corresponding author. E-mail: luwang_hcivr@sdu.edu.cn*
- *B. Wang is with School of Computer Science and Engineering, Nanjing University of Science and Technology, 210094, Nanjing, China. Joint First Author.*
- *N. Holzschuch is with Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LJK, 38000 Grenoble, France.*


## 2 Previous work

### 2.1 Photon Mapping

Chandrasekhar [7] introduced the *radiative transfer equation*to describe radiation transport in participating media. Photon mapping has been used as a means of efficiently solving this equation. Most recently, Křivánek et al. [2] combined photons with beams and paths and automatically selected between these representations

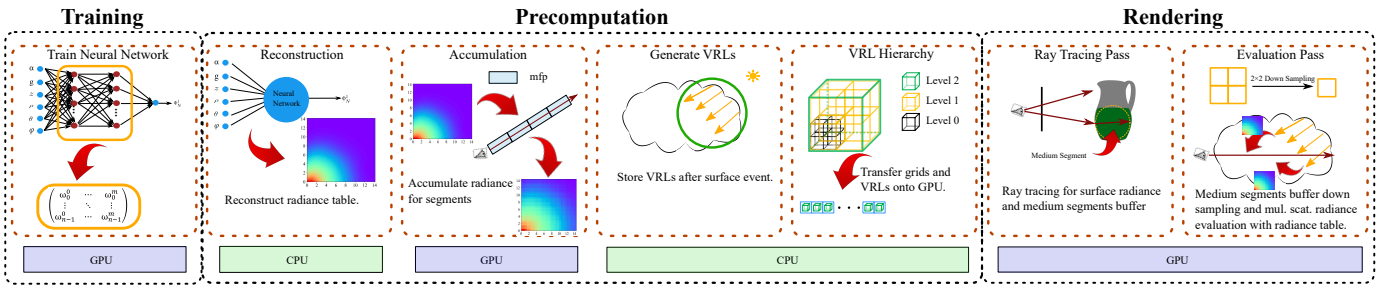



Fig. 1. In the training stage, we train a neural network model to represent multiple scattering as a function of medium parameter, position and orientation. The trained model can be used for any homogeneous participating media. In the precomputation stage, we reconstruct multiple scattering radiance from our neural network, accumulating the radiance in the table for query segment from the query point. We then generate the VRLs build hierarchy. The rendering stage is conducted in two passes: a ray tracing pass for surface lighting and camera segments in the medium, and a multiple scattering evaluation pass for the down-sampled camera segments in the medium and get the final result in the end.

using multiple importance sampling. Bitterli and Jarosz [8] further extended this concept by tracing photon planes and volume.

While both of these methods provide high-quality simulations of light transport in participating media, they usually require a long time to converge. By contrast, our method provides similar quality with much faster convergence, as it represents the multiple scattering with neural networks and even an interactive frame rate via an efficient GPU implementation.

### 2.2 Virtual Ray Lights

Novák et al. [1] proposed virtual ray lights (VRL) for simulating light transport inside translucent materials, using importance sampling for the transfer between camera rays and virtual light sources.

VRL can be used to achieve high-quality simulation of multiple scattering, although this comes at the cost of a large number of virtual rays and a consequent slow convergence speed, especially for highly scattering media. Comparatively speaking, combining our method with VRL is much faster while still providing similar results.

### 2.3 Diffusion Theory

Dipole-based methods are very efficient at representing multiple scattering effects in high-albedo materials. The material multiple scattering response is encoded into a surface function that can be queried efficiently. Jensen et al. [3] introduced the method to computer graphics, after which Jensen and Buhler [9] presented a faster version relying on precomputed incoming radiance on the surface. D'Eon and Irving [10] improved the accuracy of the Dipole method using quantized diffusion, while Frisvad et al. [11] introduced the *Directional Dipole*, which takes into account the orientation of incoming light relative to the surface. Subsequently, Habel et al. [12] combined photon beams and the diffusion model. All dipole methods approximate the material response as isotropic, and approximate the object as a flat surface. Our algorithm, like that of Wang and Holzschuch [4], handles arbitrary geometry and phase functions, and is thus more accurate than diffusion-based methods.

### 2.4 Precomputation-based Method

The idea behind precomputation methods involves precomputing the multiple scattering response and storing it in a structure, to be accessed at rendering time. Donner et al. [13] precomputed the material response on the surface as a function of material properties, surface position and outgoing direction. Instead of computing response on the surface, Wang et al. [14] and Wang and Holzschuch [6] assumed an infinite medium and stored multiple scattering response in a precomputed table, as a function of position and outgoing direction. Wang and Holzschuch [4] then combined this precomputed table with several illumination simulation algorithms. Our algorithm is based on Wang and Holzschuch [4], but provides a far more compact representation that contains all material parameters.

Precomputation has also been used for discrete media. Moon et al. [5] computed the visual aspect of sand by storing the probability density in a set of concentric spheres or shells. Meng et al. [15] converted discrete grains to continuous homogeneous media in order to decouple the precomputation from the grain pack. Muller et al. [16] converted discrete mixing granular materials to heterogeneous media.

In a departure from the above methods, we here represent the multiple scattering effects using a neural network model, which is more compact than their naive table representation. The extra cost associated with reconstructing from the neural network is proven to be negligible.

### 2.5 Neural Networks

Ren et al. [17] used a multilayer acyclic feed-forward neural network to map scene data, such as position, view direction and light direction to the indirect illumination. Both their approach and ours rely on the multilayer perceptron; while they use it for global illumination representation, and we use it for multiple scattering effects.

Kallweit et al. [18] devised a technique for efficiently synthesizing images of atmospheric clouds using a combination of Monte Carlo integration and neural networks, in which the spatial and directional distribution of radiant is pre-learned from from tens of cloud exemplars. To render a new scene, these authors sampled the visible points of the cloud and, for each, extracted a hierarchical 3D descriptor of the cloud geometry with respect to the shading location and light source. This method is highly decoupled from the shape of the scene. By contrast, our method avoids handling the scene shape by assuming an infinite medium.

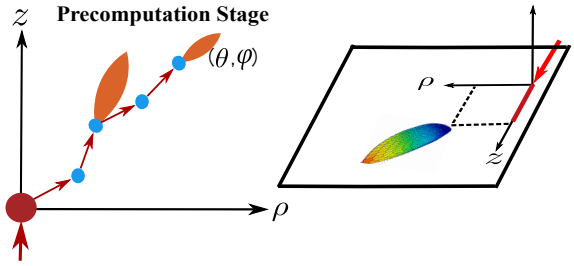



Fig. 2. Parametrization of the multiple scattering effects.

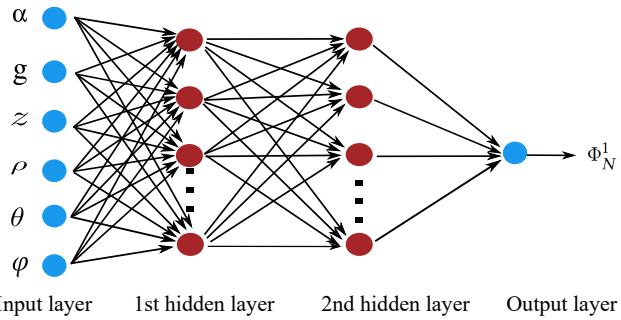



Fig. 3. The structure of our neural network. We use a two-layer (hidden layer) model with 50 nodes in each layer. The activation function is *tanh*.

## 3 Neural Network Model For Precomputed Multiple Scattering

### 3.1 Notations and Context

We only consider a homogeneous material with index of refraction $\eta$, scattering coefficient $\sigma_s$, absorption coefficient $\sigma_a$ and phase function $p(\omega, \omega_t)$. We use $\ell$ to denote the mean-free path inside the material, with $1/\ell = \sigma_t = \sigma_s + \sigma_a$.

We distinguish 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.

Our algorithm is designed to work with any rendering framework. We here describe integration with Virtual Ray Lights (VRL).

### 3.2 Multiple Scattering Function

First, we precompute multiple scattering effects in a table, assuming a light source with a dirac in position and direction in an infinite participating medium. The problem is characterized by symmetry of revolution: we parametrize multiple scattering using cylindrical coordinates for position $r(\rho, z)$ and spherical coordinates at each point with direction $(\theta, \varphi)$ (see Figure 2). We sample the homogenous media space with varying anisotropy $g$ and scattering albedo $\alpha$. All spatial dimensions are normalized by the material mean-free path, to reduce the number of parameters. Finally, we obtain the following multiple scattering function parametrization: $R(\rho, z, \theta, \varphi, g, \alpha)$. This is a function that maps from a six-dimensional domain to one dimension (multiple scattering intensity).

We compute this function using Monte Carlo-based simulation. We shoot photons from the light source, let them travel in the medium (being scattered or absorbed), and then accumulate their contributions in a table. It is too expensive to store the entire function in this discrete manner.

### 3.3 Neural Network Model

The multiple scattering function $R(\rho, z, \theta, \varphi, g, \alpha)$ maps from a six-dimensional domain to one dimension, where the output function has an exponential falloff with the spatial coordinates and can be highly anisotropic for the angular coordinates (depending on the anisotropy of the material) for a given material with property $g, \alpha$.

We treat this as a regression problem and train a neural network to learn the multiple scattering function $\Phi$, approximating it with $\Phi_N(\rho, z, \theta, \varphi, \alpha, g, \mathbf{w})$, where $\mathbf{w}$ denotes the weights and biases of $\Phi_N$, found by minimizing:

$$E = \sum_i \|r_i - \Phi_N(\rho, z, \theta, \varphi, \alpha, g, \mathbf{w})\|^2 . \quad (1)$$

### 3.4 Data Set

To get the data set for the multiple scattering function, we first sample the material space $g, \alpha$, then use Monte Carlo by shooting photons into the sampled medium and collecting the multiple scattering data for sampled positions and directions. We set the $\ell$ as 1, then sample the anisotropy parameter $g$ with {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 0.99} and the albedo $\alpha$ with {0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 0.99, 0.995 }. This results in 156 different materials. For each sampled material, we shoot 500 million photons and collect the scatter events in a table with a maximum size of $24 \times \ell$. We take 100 samples for the spatial parameters $\rho$ and $z$ and sample the angular parameters $\theta$ and $\varphi$ every 10 degrees.

The computational cost differs for each material; in total, it takes 8.5 hours to compute the values for all the materials.

We preprocess the radiance in the table by applying a logarithm operator for improved training performance: $\Phi'(\rho, z, \theta, \varphi, g, \alpha) = log(\Phi(\rho, z, \theta, \varphi, g, \alpha) + 1)$.

### 3.5 Neural Network Structure and Training

We used a neural network model to learn the multiple scattering function. We use two fully connected hidden layers with 50 nodes for each layer (see Figure 3), and employ *tanh* as the activation function.

We normalize the input parameters into [0, 1] and shuffle them. The networks are then optimized using the ADAM optimizer in TensorFlow with a learning rate of 0.01. We compute the loss for the network as the difference between predicted radiance and computed radiance. The network is trained using the *L2* error metric; we split the entire database into training (70 %) and testing (20 %) subsets. 10% of the dataset is not used. The model is trained using the mini-batch approach: in total, we have 9513 batches, each with a batch size of 30K. During the training, the mini-batches are chosen sequentially.

We train our network with 64 epochs. It takes 45 minutes to train the network on a 2.20GHz Intel(R) Xeon(R) CPU E5-2699 (44 cores) machine with Nvidia TITAN RTX GPU. Through the implementation of our neural network model, we decrease the storage cost for all materials from 50.0 GB to 11.8 KB.

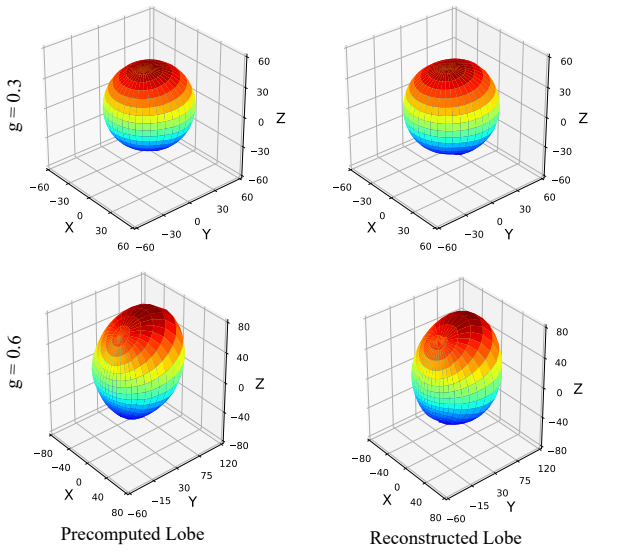



Fig. 4. Comparison between the lobe simulated using the Monte Carlo method and the one reconstructed by our neural network. Material: Bumpy-sphere, g = 0.3 and g = 0.6, lobe position (4.8, 4.8).

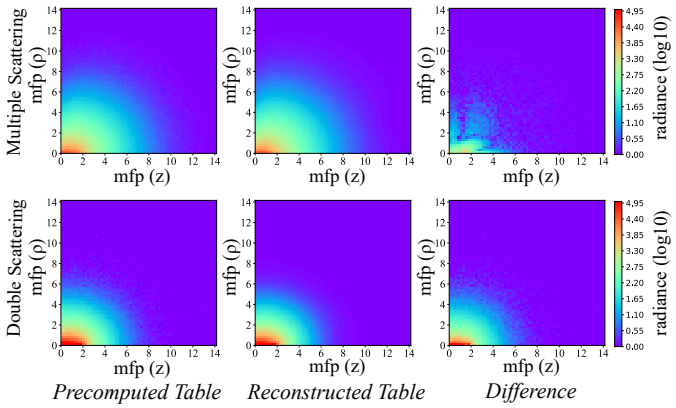



Fig. 5. Comparison between the reconstructed radiance and the radiance simulated using the Monte Carlo method. Material: Lucy, g = 0.0.

### 3.6 Double Scattering Neural Network

Another neural network is trained to express the response for double scattering, which has a higher frequency than multiple scattering. As representing them together in one neural network would introduce bias, we here represent them with two separate neural networks.

For double scattering, we employ the same structure as the multiple scattering neural network, with a different data set, where only the first two scattering events are stored in the precomputed table. We also change the parameters, setting the learning rate to 0.001 and the number of mini-batches to 19521 with size 4K and 1024 epochs. The size of our double scattering model is the same as that of our multiple scattering model.

### 3.7 Multiple Scattering Table Extraction and Accumulation

Before rendering, for a specific material, we first reconstruct the multiple scattering and double scattering table from the neural network model, convert these tables to a 4D look-up table, and then sum them up. More precisely, we extract the precomputed table data from the network for a given material and apply an exponential operator to reconstruct the radiance: $\hat{\Phi}(\rho, z, \theta, \varphi, g, \alpha) = \exp(\hat{\Phi}'_N(\rho, z, \theta, \varphi, g, \alpha)) - 1$, where $\hat{\Phi}$ is the reconstructed multiple (or double) scattering radiance we stored in the table, while $\hat{\Phi}'_N$ is the value reconstructed from the neural network.

Figure 4 presents a comparison between the lobes reconstructed using our algorithm and the raw data from the Monte Carlo simulation, for multiple scattering.

Figure 5 depicts a comparison of both double and multiple scattering between the outgoing radiance reconstructed using our algorithm and the value computed using Monte Carlo simulation, for an isotropic material.

Once this step is complete, we have a table representing the outgoing radiance at a point $(z_i, \rho_i)$, with a specific direction $(\theta_i, \phi_i)$. We obtain the *Segment-to-Point* contribution with this table. Starting from this table, we encode the length of the camera ray by accumulating the contribution along direction $(\theta_i, \phi_i)$ with length 5× mfp and thereby create a segment-to-segment table. The accumulated table represents the multiple scattering from a point $(z_i, \rho_i)$, with a specific direction $(\theta_i, \phi_i)$ and length 5× mfp. We use the GPU to accelerate this accumulation step. It should be briefly noted here that we tried training the *Segment-to-Segment* contribution with a neural network directly, but found that it tended to overestimate contributions (see Section 5.5 for a more detailed discussion).

### 3.8 Rendering

Our rendering algorithm is based on Virtual Ray Lights [1]. We trace rays from the light source and store the first ray after an interaction with the surface of the medium; this ray will be the Virtual Ray Light. Each VRL has the following attributes: position of the ray origin, ray direction, light ray length before the next surface event, and radiance at the ray origin. We organize these VRLs into a hierarchy of regular grids (as detailed in Section 4.3). We then transfer the data to the GPU.

Rendering is done on the GPU, gathering the contributions of the VRLs from other nodes using the segment-to-segment table. We use one specific level of the grid hierarchy and discard the unimportant nodes that have a large solid angle. The contribution from a VRL to a camera ray is computed in a manner similar to [4]:

$$z = (\boldsymbol{P} - \boldsymbol{v}) \cdot \boldsymbol{d}, \tag{2}$$

$$\rho = \|(\boldsymbol{P} - \boldsymbol{v}) - z\boldsymbol{d}\|, \tag{3}$$

$$\text{mult.}(\boldsymbol{P}, \boldsymbol{\omega}_t) = W\, \hat{\Phi}\left(\frac{\rho}{\ell}, \frac{z}{\ell}, T_{(\boldsymbol{v},\boldsymbol{d})}(\boldsymbol{\omega}_t)\right). \tag{4}$$

where $(\boldsymbol{v}, \boldsymbol{d})$ is the position and orientation of the segment-to-segment precomputed table, $(\rho, z)$ are the cylindrical coordinates around the axis of propagation, and $T_{(\boldsymbol{v},\boldsymbol{d})}(\boldsymbol{\omega}_t)$ is the direction corresponding to $\boldsymbol{\omega}_t$ in the frame defined by $(\boldsymbol{v}, \boldsymbol{d})$. Moreover, $W$ is the factor denoting the ratio of the camera segment length to the accumulated segment length (5× mfp) and is defined as:

$$W = \frac{\sum_{i=1}^{i=k} e^{-i\ell}}{\sum_{i=1}^{i=5} e^{-i\ell}}, \tag{5}$$

where $k$ is the length of the camera segment divided by 5 × mfp.

**Algorithm 1** Multiple Scattering Estimation

**Input:**
$S$ = camera segment ray in medium
$G$ = all grid cells included in current thread
$A$ = accumulated segment table
**Output:**
$R$ = multiple scattering radiance

**for all** *grid* cell $\in G$ **do**
  **// discard grid cells with small contribution**
  **if** solidAngle(grid*cell*,S) $> \varepsilon$ **then** discard this *grid* cell
  **end if**
  $V$ = all VRLs in *grid* cell
  **for all** *vrl* $\in V$ **do**
    **// compute relative coordinate between** *vrl* **and** $S$
    *index* = relativeCoordinate(*vrl*, $S$)
    **// weight of segment length (see Section 3.8)**
    $R$ += $A$[*index*] $\times$ power(*vrl*) $\times$ weight($S$)
  **end for**
**end for**

## 4 Implementation Details

### 4.1 GPU Pipeline

We implement our rendering process on the GPU, using Optix. It runs in three passes:

1) Compute direct illumination and store the first segment inside the medium into a buffer;
2) Gather double and multiple scattering from these Virtual Ray Lights (see Alg. 1);
3) Approximate single scattering using the algorithm proposed by Jensen et al. [3].

For the first step, we trace a path from the camera, with multiple bounces (the maximum depth is provided in Table 2). For the second step, we downsample scattering illumination with one sample per $2 \times 2$ tile, as multiple scattering trends to be smooth. We then interpolate the multiple scattering radiance computed in the second pass and add it to the direct illumination and single scattering to obtain the final result (see Figure 1).

### 4.2 Multiple Scattering Table Representation

We use a regularly sampled 4D structure to store the double and multiple scattering radiance tables. In the spatial domain, we sample both $z$ and $\rho$ per mean-free path (mfp), with a maximum distance of 32 mfp. In the directional domain, we sample both $\theta$ and $\phi$ every 10 degrees. Both tables are stored as one-dimensional buffers on the GPU.

To reduce memory and access costs, we cull low values in the tables. For each cell, we check the precomputed lobe; if most of the directions (90%) for this lobe have low radiance values, this lobe is discarded.

This radiance table is used as an intermediate representation before the neural network computations are run, since these computations tend to be expensive. Each query requires three matrix multiplications ($6 \times 50$, $50 \times 50$ and $50 \times 1$) and several trigonometric function evaluations, and there are multiple queries for each sample.

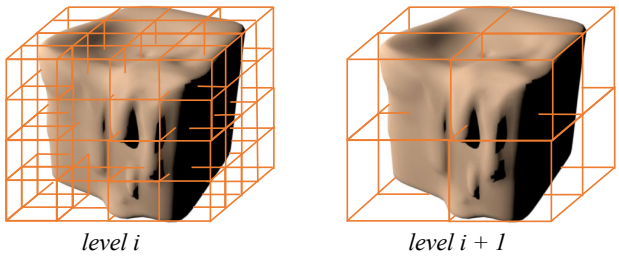



Fig. 6. Two grid layers in our grid representation. The layer $i+1$ is computed from layer $i$ by merging eight grid cells together.

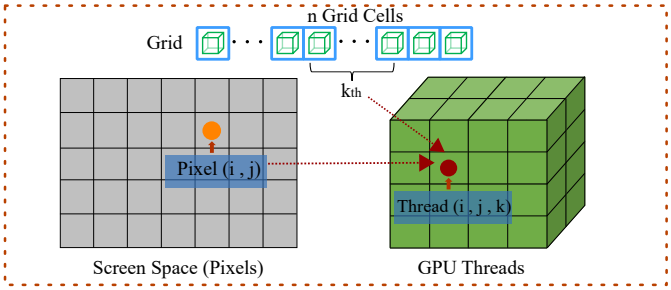



Fig. 7. The mapping between the pixels, grid cells and 3D threads.

### 4.3 VRL Grid Representation

We organize our virtual ray lights into a hierarchy of regular grids. Each grid in the hierarchy is regular, with all cells being identical in size. Each grid cell stores the index of all virtual ray lights that are located inside, along with a representative virtual ray light for the cell, and the bounding box of the virtual ray lights.

We build two to five levels of regular grids, depending on the relative size of the object and the material mean-free path.

At rendering time, we select the appropriate grid level and transfer only this level to the GPU. We store all virtual ray lights in a one-dimensional buffer and reorder them to ensure that the virtual ray lights in each grid cell are stored continuously.

We compute the rendering via the use of threads. The threads are organised in 3D, with two dimensions used to indicate the pixel coordinates of the camera ray's origin, while the third dimension represents several grid cells. Each GPU thread gathers the contribution made by the grid cells to the pixel. We discard grid cells during the traversal if the solid angle subtended by the cell and the camera ray is smaller than a predefined threshold.

## 5 Results

We trained our neural network model with Tensorflow on a 2.20GHz Intel(R) Xeon(R) CPU E5-2699 (44 cores) machine with an Nvidia TITAN RTX GPU. For rendering, we used Virtual Ray Lights (VRL) implementation inside the Mitsuba Renderer [19]. All timings in this section are measured on a 2.20GHz Intel(R) Xeon(R) CPU E5-2699 (44 cores) with 32 GB of main memory and an Nvidia TITAN RTX GPU for real-time rendering using Optix (Cuda). Unless otherwise specified, all timings correspond to pictures with $512 \times 512$ pixels; for the Dragon and Lucy scenes, we used $768 \times 576$ pixels. Reference solutions are computed using UPBP inside smallUPBP [20]. We also compare our results with those of Wang and Holzschuch [4].

All materials in our scenes are homogeneous, with Henyey-Greenstein phase functions and refractive boundaries. Material properties are derived from Křivánek et al. [2], Narasimhan et al. [21] and Holzschuch [22] (see Table 1).

We use our neural network model for multiple and double scattering evaluation, combined with the single scattering approximation proposed by Jensen et al. [3]. In the single scattering evaluation, we use 100 regular samples along each camera ray and connect each of these samples with the light sources, without considering refraction at the medium boundary.

TABLE 1
Parameters for the materials used in this paper.

| Name | $\alpha$ | | | $\ell$ | | | g | $\eta$ |
|---|---|---|---|---|---|---|---|---|
| | R | G | B | R | G | B | | |
| BumpS. | 0.955 | 0.677 | 0.457 | 4.55 | 3.23 | 2.17 | 0.9 | 1.50 |
| Candle | 0.980 | 0.962 | 0.750 | 0.65 | 0.63 | 0.59 | 0.8 | 1.45 |
| Dragon | 0.882 | 0.938 | 0.980 | 0.29 | 0.18 | 0.33 | 0.5 | 1.50 |
| Milk | 0.999 | 0.999 | 0.999 | 0.84 | 0.75 | 0.68 | 0.7 | 1.30 |
| Oil | 0.398 | 0.453 | 0.032 | 9.71 | 11.63 | 2.74 | 0.9 | 1.50 |
| Lucy | 0.936 | 0.939 | 0.941 | 0.21 | 0.20 | 0.19 | 0.0 | 1.50 |
| Cloud | 0.909 | 0.909 | 0.909 | 0.909 | 0.909 | 0.909 | 0.0 | 1.0 |

### 5.1 Quality Validation

Figure 8 presents a comparison between our method, competing methods (Wang and Holzschuch [4] and Virtual Ray Lights (VRL)), and a reference image computed using UPBP. To facilitate better comparison, only multiple scattering results are displayed here. Our algorithm produces pictures that are virtually identical to the reference image and to Wang and Holzschuch [4], while being several orders of magnitude faster.

Figures 9, 10, 11 and 20 present the different scattering components in our method: multiple scattering only, double and multiple scattering, and double and multiple scattering combined with the single scattering approximation (full solution). Increasing the number of scattering components increases the computation time, which goes from roughly 50 ms per frame for multiple scattering only to roughly 150 ms per frame for the full solution. The visual importance of each component depends on the material: for a high-albedo material such as milk (Figure 9), with $\alpha = 0.999$, most of the visual appearance comes from multiple scattering, while for lower-albedo values ($\alpha \approx 0.9$), the visual impact of the double and multiple scattering effects is more visible (Figures 10, 11 and 20).

Figure 10 presents a comparison between our method, competing methods by Wang and Holzschuch [4] and Virtual Ray Lights, and a reference image computed using UPBP, for the full solution (single-, double-, multiple- scattering and other surface lighting). Overall, the end result of our method appears similar to the reference, and the difference image confirms this. We zoom in on two interesting areas: a thin area on the ears and a thick area on the body. We find that our method with full solution works well in both cases. Combining Wang and Holzschuch [4] and UPBP provides a result that is closer to the reference, but with a computation time measured in minutes rather than milliseconds. Moreover, VRL produces results with a large number of artifacts, due to the insufficient count of VRLs.

Figure 11 presents results computed using our method with an index-matching material. In this situation, there is no refraction at the boundaries of the material, which allows us to use path-tracing for the reference solution. For this material, single and multiple scattering effects bring the result closer to the reference, for a total cost of only 150 ms per frame.

### 5.2 Comparison to Dipole-based Methods

We further compare our method to the fast approximation of multiple scattering effects using the Dipole approximation [3], as implemented in the Mitsuba Renderer. Figure 20 presents a side-by-side comparison between our method, UPBP for reference and the dipole diffusion method. For our method, the results shown are for multiple scattering only, multiple scattering with double scattering and full solution. For the Dipole-based method, we compute single scattering using UPBP and double & multiple scattering using the Dipole approximation. Overall, our method appears similar to the reference at an interactive frame rate, benefiting from the GPU acceleration. By contrast, the dipole diffusion method overestimates the thin areas, e.g. the hand.

### 5.3 Performance Measurement

Table 2 reports timings, memory cost and Perceived Signal-Noise Ratio (PSNR) for our method: first with multiple scattering only, then with double and multiple scattering, and finally with single, double and multiple scattering (full solution). We also report computation times for the two competing methods, Wang and Holzschuch [4] and VRL, and the reference solution. For the reference, we always used 6 h of computation time.

Even with all scattering effects included, our method achieves interactive framerates, at approximately 150 ms per frame. With multiple scattering effects only, our method achieves interactive framerates of around 50 ms per frame. The increase in computation time when more scattering effects are included is not only related to more effects being computed: double scattering also requires more virtual ray lights to avoid artefacts. The increase in computation time corresponds with an improvement in quality, as measured by the Perceived Signal-Noise Ratio. Adding single scattering approximation has a smaller impact on performance while increasing the visual quality. Users can select which scattering components to include depending on their performance budget and quality requirements.

Wang and Holzschuch [4] used a precomputed table for each material. For this algorithm, changing the material parameters results in additional computation time being required to generate the table. By contrast, our method provides a single neural network for all homogeneous participating media, regardless of their parametersparameters; changing material parameters is instantaneous. Our method requires 23.6 KB to store parameters for the entire space of participating media, compared to 100 MB for a single material in Wang and Holzschuch [4].

Virtual Ray Lights significantly underperform in these test scenes: the equal time requirements results in a small number of virtual lights, insufficient for the method.

Tables 3 and 4 present the cost of each step in our method, first with double and multiple scattering (Table 3), then with multiple scattering only (Table 4). For precomputation, both methods require only a couple of seconds (1 s – 4 s). With double scattering, it is necessary to reconstruct two precomputed tables, thus doubling the reconstruction cost; more virtual ray lights are also required, increasing the VRL generation and organization time. For rendering time, the first step, tracing rays inside the material, is independent from the scattering effects. The evaluation

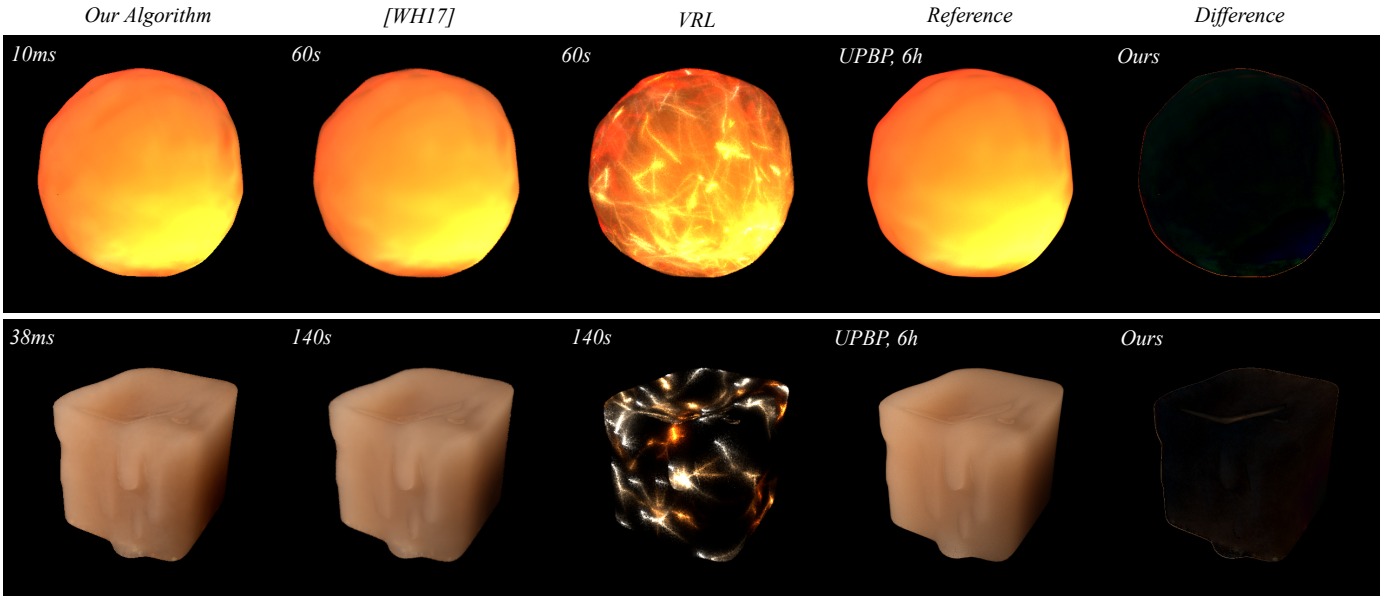



Fig. 8. Our algorithm (multiple scattering represented with neural network models), compared to Wang and Holzschuch [4] (multiple scattering represented with a naive table), Virtual Ray Lights (VRL) [1] and Unified Points, Beams and Paths (UPBP) [2]. Using neural networks to represent multiple scattering produces results that are almost identical to the reference at a real-time frame rate.


Fig. 9. With high-albedo, small mean-free path materials like milk, our algorithm produces images that are very close to the reference, at a fraction of the cost. It should be noted here that GPU ray-tracing does not handle transparency effects, which causes the difference in shadows. Material: Milk.

step is more expensive when double scattering is included, due to the increased number of virtual ray lights.

### 5.4 Grid Hierarchy Validation

Grid hierarchy is an important component of our algorithm. To validate its influence, we here illustrate a performance and error comparison between two cases: with and without grid hierarchy for varying mean-free path (mfp) in Figure 12. Without a grid hierarchy in place, decreasing the mean-free path has a significant impact on performance; with the grid hierarchy, however, there is no visible impact on performance when mfp decreases. The impact of the grid hierarchy on quality is minimal at around 5 %.

### 5.5 Segment-to-Segment Neural Network Model

As noted above, we initially attempted to use a neural network to generate *Segment-to-Segment* multiple scattering directly, rather than generating these contributions as the sum of *Segment-to-Point* contributions. Figure 13 presents a comparison between results computed by training a neural network on *Segment-to-Segment* contributions, the results of our method and the reference. As can be seen from the figure, the *Segment-to-Segment* contributions obtained by directly training a neural network overestimate the contributions.

To train our neural network on the *Segment-to-Segment* multiple scattering effects, we constrain the length of the camera segment to 5 mfp, then perform an accumulation for the camera segment from the *Segment-to-Point* contribution. The accumulation tends to amplify the differences between lobes and makes the neural network less accurate.

### 5.6 Neural Network Parameters

Figures 15 and 16 illustrate the impact of the neural network parameters on loss: namely, number of epochs, number of nodes for each layer, and the choice of the activation function (i.e. using *tanh* or Rectified Linear Unit (ReLU)).

Increasing the number of epochs decreases the loss, as does increasing the number of nodes in each layer. In practice, we used 50 nodes and selected *tanh* for the activation function.

### 5.7 Rendering Parameters

Figure 21 presents the impact of changing the scale of the mean-free path on the candle scene. From left to right, the mean-free

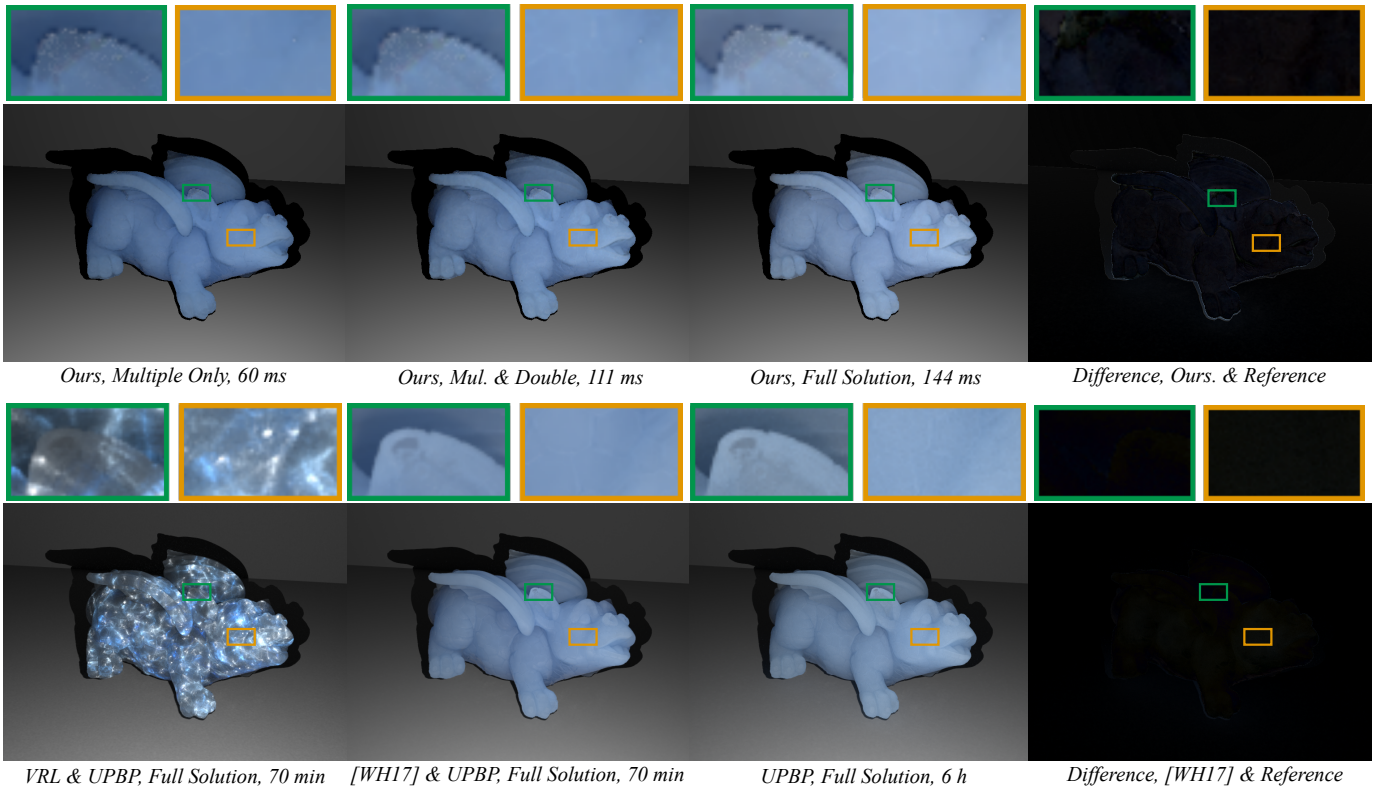



Fig. 10. When multiple scattering effects are dominant, our algorithm produces images that are close to the reference: comparison with Wang and Holzschuch [4], VRL and UPBP on the Dragon Scene (full solution). Material: Dragon.


Fig. 11. To facilitate a meaningful comparison with Path Tracing (PT), we used a material with a matching refraction index, meaning that there is no refraction at the interface. As we assume a single interface and semi-infinite material when computing scattering, our algorithm tends to overestimate extinction (full solution). Material: Cloud.

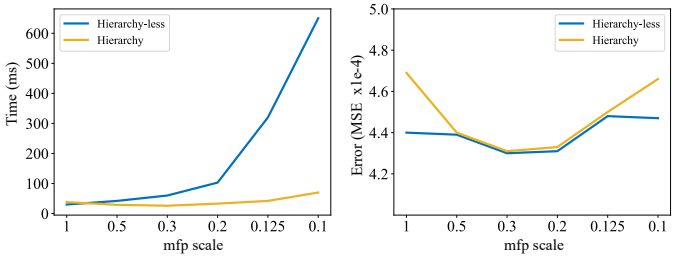



Fig. 12. Comparisons between our method with and without the grid hierarchy on the Candle Scene with varying mean-free path (mfp).

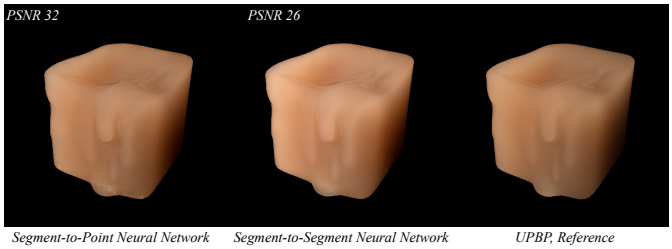



Fig. 13. Using a neural network on *Segment-to-Segment* precomputed multiple scattering, as opposed to *Segment-to-Point* in our algorithm, overestimates scattering effects and produces images that are lighter than the reference.

path is divided by 10, making the material less transparent. For all values of the mean-free path, our method provides results that are similar to the reference. The rendering cost increases slightly as mfp decreases, from 26 ms to 65 ms, even though the number of virtual ray lights used increases from 40,000 to 2 million. This

TABLE 2
Computation time, memory costs and error for our test scenes.

| | | Ours (Multiple only) | | | Ours (Multiple + Double) | | | Ours (Full Solution) | | | [WH17] | | | VRL | | Reference |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| scene | resolution | VRLs K | time ms | *PSNR* | VRLs K | time ms | *PSNR* | VRLs K | time ms | *PSNR* | VRLs K | *PSNR* | Memory MB | VRLs K | *PSNR* | time h |
| BumpS. | 512 × 512 | 3 | 10 | 31.4 | - | - | - | - | - | - | 3 | 34.8 | 32 | 1 | 24.5 | 6 |
| Candle | 512 × 512 | 40 | 38 | 32.1 | - | - | - | - | - | - | 40 | 36.3 | 33 | 3 | 14.1 | 6 |
| Oil | 512 × 512 | 1 | 20 | 22.6 | 1 | 20 | 22.6 | 1 | 35 | 23.2 | - | - | - | - | - | 6 |
| Dragon | 768 × 576 | 40 | 60 | 26.3 | 150 | 111 | 29.8 | 150 | 144 | 28.6 | 40 | 37.3 | 78 | 8 | 23.5 | 6 |
| Milk | 512 × 512 | 80 | 50 | 24.5 | 250 | 130 | 24.6 | 250 | 135 | 24.5 | 60 | 32.9 | 33 | 10 | 21.5 | 6 |
| Lucy | 768 × 576 | 80 | 27 | 23.7 | 250 | 44 | 27.6 | 250 | 69 | 30.5 | 80 | 36.3 | 43 | 1.5 | 19.8 | 6 |

TABLE 3
Cost of each step for our test scenes with multiple and double scattering evaluation.

| scene | Precomputation (M&D) | | | | Rendering (M&D) | | |
|---|---|---|---|---|---|---|---|
| | Recons. s | Acc. ms | VRL G. s | Tot. s | Ray. ms | M.&D. Eval. ms | Tot. ms |
| Drag. | 1.39 | 3 | 0.51 | 1.90 | 34 | 77 | 111 |
| Milk | 1.34 | 3 | 2.78 | 4.12 | 8 | 122 | 130 |
| Lucy | 1.33 | 3 | 0.60 | 1.93 | 13 | 31 | 44 |
| Cloud | 1.45 | 3 | 0.86 | 2.31 | 1 | 81 | 82 |

TABLE 4
Cost of each step for our test scenes for multiple scattering only. VRL G. includes VRL generation and VRL hierarchy organization. Recons. and Acc. refer to radiance table reconstruction and accumulation respectively. M. Scat. Eval. indicates multiple scattering evaluation.

| scene | Precomputation (M.) | | | | Rendering (M.) | | |
|---|---|---|---|---|---|---|---|
| | Recons. s | Acc. ms | VRL G. s | Tot. s | Ray. ms | M. Eval. ms | Tot. ms |
| BumpS. | 0.75 | 3 | 0.16 | 0.91 | 5 | 5 | 10 |
| Candle | 0.75 | 3 | 1.72 | 2.47 | 4 | 34 | 38 |
| Drag. | 0.72 | 3 | 0.16 | 0.88 | 34 | 26 | 60 |
| Milk | 0.75 | 3 | 1.00 | 1.75 | 8 | 42 | 50 |
| Lucy | 0.71 | 3 | 0.24 | 0.95 | 13 | 14 | 27 |
| Cloud | 0.78 | 3 | 0.3 | 1.08 | 1 | 29 | 30 |

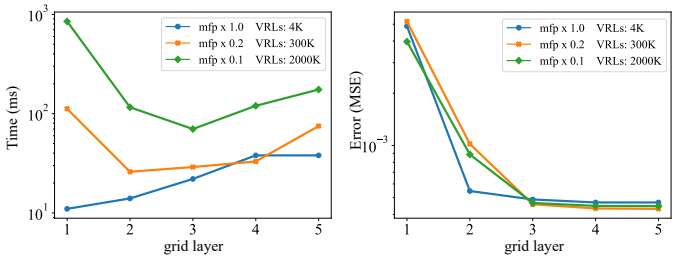



Fig. 14. Performance and Error (MSE) as a function of grid layer on the Candle Scene (multiple scattering only) with different mfp .

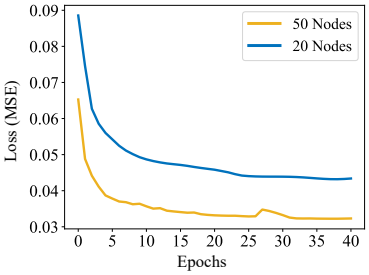



Fig. 15. Error (MSE) as a function of the number of epochs, for different node sizes for each layer.

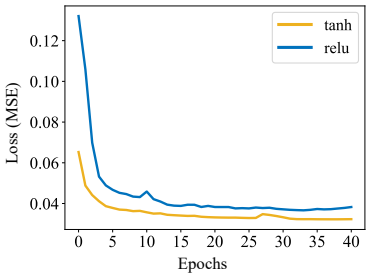



Fig. 16. Error (MSE) as a function of the number of epochs for different activation functions.

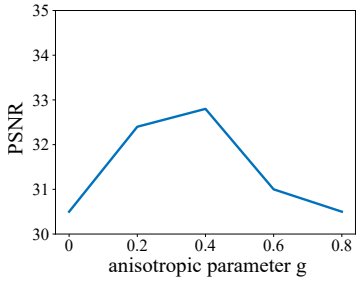



Fig. 17. *PSNR* as a function of varying anisotropic parameter: *g*. Material: Lucy.

confirms the benefits of our hierarchical representation.

Figure 14 presents the impacts of the number of grid layers on the performance and the error (MSE) of the Candle Scene with different mean-free paths. Increasing the number of grid layers always increases the quality of the simulation; however the impact is less significant for this scene after the third layer. Moreover, the impact on computation time is more varied: when mfp is small, increasing the number of grid layers decreases the computation time. This is due to the number of virtual ray lights involved, as smaller mean-free paths (0.1 or 0.2) require a larger number of virtual ray lights. At the top grid layer, there are a large number of grid cells assigned to each thread, which has a significant impact on the time cost. Increasing the number of grid layers results in a smaller number of cells being assigned to each thread, which improves the performance. Further increasing the number of layers (from 3 to 5) results in larger grid cells, which in turn results in a smaller number of cells being discarded, meaning that costs increase. For large mean-free paths (1), the number of virtual ray-lights is smaller and does not affect the performance for the top layer. In this situation, only discarding cells has an impact on performance, with the result that computation time increases with the number of grid layers.

Figure 17 illustrates the impact of anisotropic parameter *g*

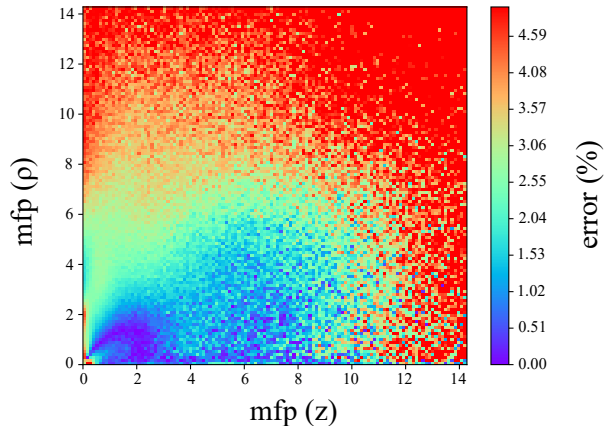



Fig. 18. The energy error ratio between the precomputed table and the table reconstructed by our neural network. The error ratio is evaluated by summing up the difference between two lobes at the sampled position in the two tables and then dividing by the precomputed table value. Material: Candle.

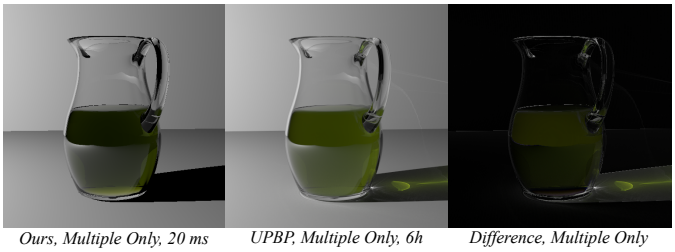



Fig. 19. Comparison between our method and references on a medium with a low albedo (Oil). On this material, low-order scattering effects dominate. Our method tends to underestimate multiple scattering, due to the inaccuracy of neural network representation for extremely small values caused by the low albedo in the media.

on the rendering quality for our algorithm on the Lucy Scene. The PSNR (Peak Signal-Noise Ratio) stays consistently above 30, which confirms that the rendering quality is almost independent of the value of $g$, although there are some variations.

### 5.8 Discussion and Limitations

Our neural network cannot guarantee energy conservation. Nevertheless, the energy error introduced remains acceptable. Figure 18 presents the error ratio on the Candle Scene. The error ratio is evaluated by summing up the difference between lobes at the sampled positions in the precomputed table and those in the table reconstructed by the neural network, then dividing by the precomputed table value. The maximum error ratio is around 5%. Large error ratios are concentrated in areas with small scattering values, i.e. from the source. The neural network tends to underperform in areas with small values. These are also areas where scattering is less visible, resulting in little visual difference in our experiments.

Our method performs better for materials where multiple scattering effects are dominant. Figure 19 presents a comparison with reference for a medium where the albedo is small and where the mean-free path is large. Our approximation tends to underestimate scattering effects with this material; this is because the neural network tends to underperform on small values, which is caused by the low albedo.

## 6 Conclusion

We have presented a neural network model to represent double and multiple scattering events in the entire participating media space. The model provides a highly compact representation for precomputed double and multiple scattering, and can also be combined with many existing rendering algorithms, providing similar results for a fraction of the memory cost. We use our precomputed data for multiple scattering and an approximated solution for single scattering. Our model is able to represent scattering effects for any type of participating media, but is especially interesting for materials with a large albedo and small mean-free path, where higher-order scattering effects tend to dominate.

In future work, we aim to use our neural network directly on the GPU, rather than being required to reconstruct the 4D table first, by using the new APIs that support neural networks on the GPU. We also want to train new models in order to extend the range of parameters for learning, enabling inclusion of the surface property and layer thickness, as this will allow the model to be used in combination with more rendering algorithms, such as subsurface scattering or more complex participating media, such as layered media.

**Acknowledgments.** We thank the reviewer for their valuable comments. This work has been partially supported by the National Key R&D Program of China under grant No. 2017YFB0203000, the National Natural Science Foundation of China under grant No. 61802187 and 61872223, the Natural Science Foundation of Jiangsu under grant No. BK20170857, the Fundamental Research Funds for the Central Universities No. 30918011320 and ANR project ANR-15-CE38-0005 ”Materials”.

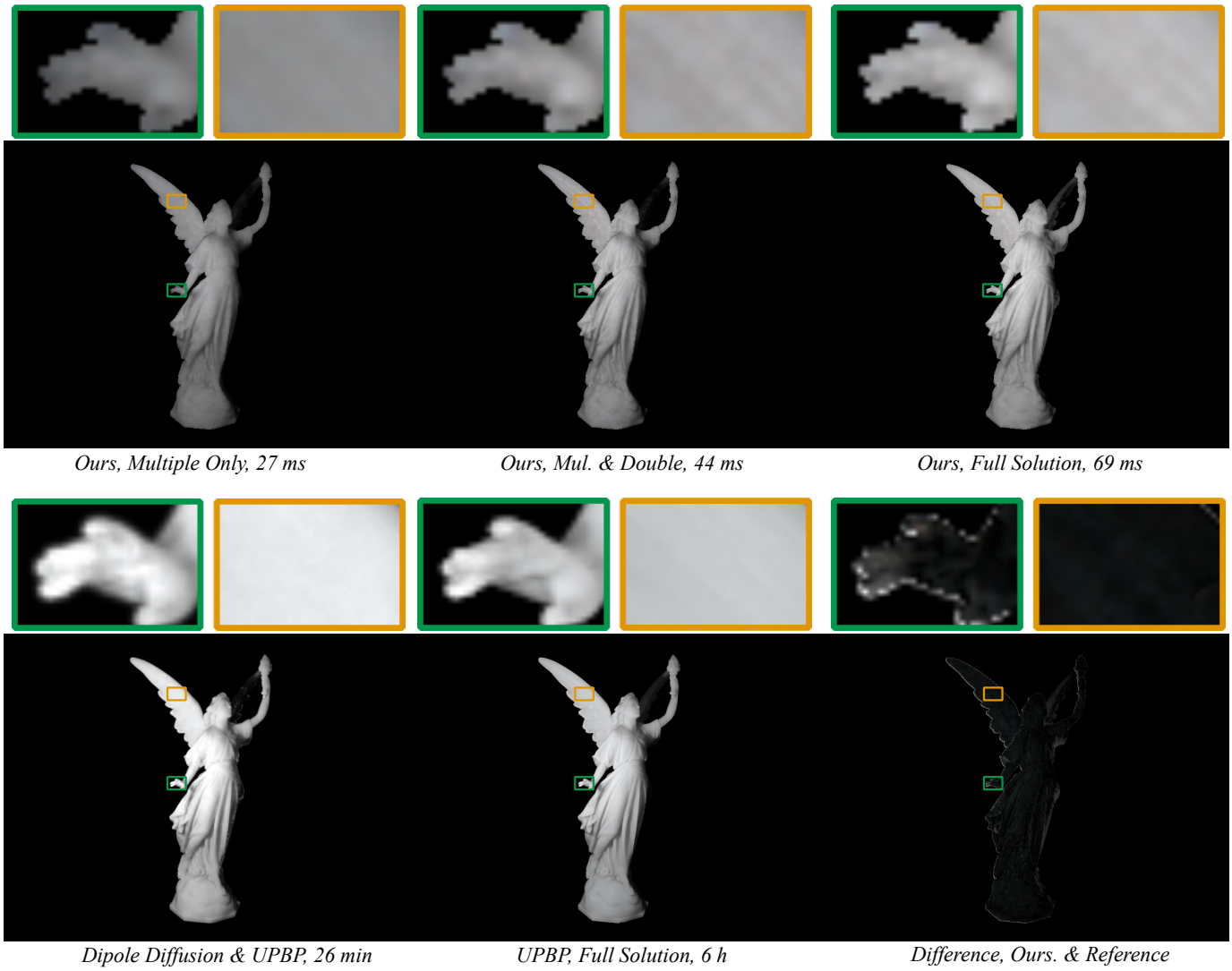



Fig. 20. Comparison with dipole-based methods for the full solution (including single and double scattering). Material: Lucy.

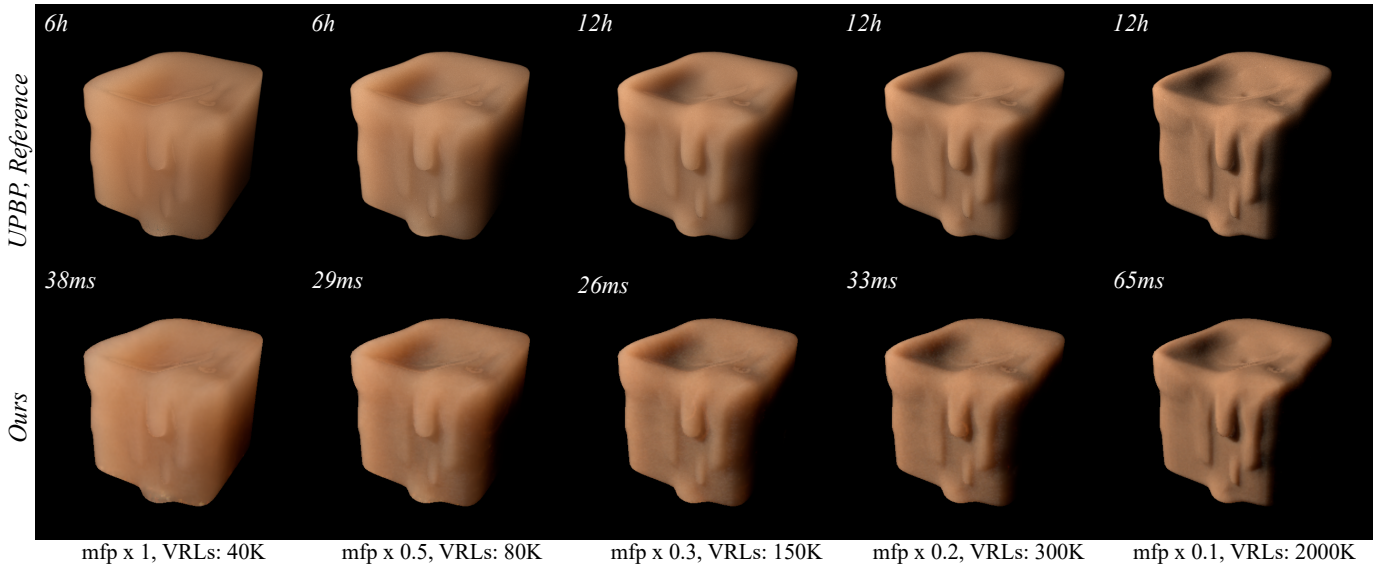



Fig. 21. Comparisons between our method with UPBP on the Candle Scene with varying mfp.

## References


[1] 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.

[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. (proc. SIGGRAPH)*, vol. 33, pp. 1–13, Aug. 2014.

[3] H. W. Jensen, S. R. Marschner, M. Levoy, and P. Hanrahan, "A practical model for subsurface light transport," in *SIGGRAPH*, pp. 511–518, ACM, 2001.

[4] B. Wang and N. Holzschuch, "Precomputed multiple scattering for light simulation in participating medium," in *ACM SIGGRAPH 2017 Talks*, SIGGRAPH '17, pp. 35:1–35:2, 2017.

[5] 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.

[6] B. Wang and N. Holzschuch, "Point-based rendering for participating media with refractive boundaries," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 10, pp. 2743 – 2757, 2018.

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

[8] 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.

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

[10] 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.

[11] 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, Nov. 2014.

[12] R. Habel, P. H. Christensen, and W. Jarosz, "Photon beam diffusion: A hybrid monte carlo method for subsurface scattering," *Comput. Graph. Forum (proc. of EGSR)*, vol. 32, June 2013.

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

[14] B. Wang, J.-D. Gascuel, and N. Holzschuch, "Point-Based Light Transport for Participating Media with Refractive Boundaries," in *Eurographics Symposium on Rendering 2016 (EI&I)*, (Dublin, Ireland), June 2016.

[15] J. Meng, M. Papas, R. Habel, C. Dachsbacher, S. Marschner, M. Gross, and W. Jarosz, "Multi-scale modeling and rendering of granular materials," *ACM Trans. Graph.*, vol. 34, pp. 49:1–49:13, July 2015.

[16] T. Müller, M. Papas, M. Gross, W. Jarosz, and J. Novák, "Efficient rendering of heterogeneous polydisperse granular media," *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, vol. 35, p. 168:1–168:14, December 2016.

[17] P. Ren, J. Wang, M. Gong, S. Lin, X. Tong, and B. Guo, "Global illumination with radiance regression functions," *ACM Trans. Graph.*, vol. 32, pp. 130:1–130:12, July 2013.

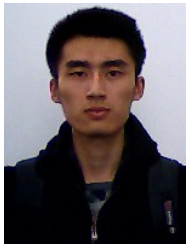
[18] S. Kallweit, T. Müller, B. Mcwilliams, M. Gross, and J. Novák, "Deep scattering: Rendering atmospheric clouds with radiance-predicting neural networks," *ACM Trans. Graph.*, vol. 36, pp. 231:1–231:11, Nov. 2017.

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

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

[21] 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. (proc. Siggraph)*, vol. 25, pp. 1003–1012, July 2006.

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

**Liangsheng Ge** is a second-year Masters student at Shandong University. He received his Bachelor Degree from Nanjing Tech University in June 2017. His research interests include rendering techniques.

**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.

**Lu Wang** is a Professor at the School of Software, Shandong University. She received her PhD from Shandong University in 2009. Her research interests include photorealistic rendering and high performance rendering.

**Xiangxu Meng** is a professor in the School of Software, Shandong University. He received his PhD from the Institute of Computing Technology, Chinese Academy of Sciences, in 1998. His current research interests include human-computer interaction, virtual reality, computer graphics, and visualization.

**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.