A Compact Representation for Multiple Scattering in Participating Media using Neural Networks
Liangsheng Ge, Beibei Wang, Lu Wang, Nicolas Holzschuch

To cite this version:

HAL Id: hal-01809890
https://inria.hal.science/hal-01809890
Submitted on 7 Jun 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
A Compact Representation for Multiple Scattering in Participating Media using Neural Networks

Liangsheng Ge
Shandong University

Lu Wang
Shandong University

Beibei Wang
Nanjing University of Science and Tech.

Nicolas Holzschuch
Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LJK

Our Method | [Wang and Holzschuch 2017] | VRL, Equal time | Reference
---|---|---|---
60s | 60s | UPBP, 6h |
140s | 140s | UPBP, 6h |

Figure 1: Equal time comparison for global illumination in participating media: our method based on neural networks (left), storing the full table [Wang and Holzschuch 2017], and Virtual Ray Lights (VRL) [Novák et al. 2012]. Right: reference solution.

CCS CONCEPTS
• Computing methodologies → Rendering;

KEYWORDS
Participating Media, Multiple Scattering, Neural Networks

ACM Reference format:
DOI: 10.1145/3214745.3214758

1 OVERVIEW

Many materials, such as milk or wax, exhibit scattering effects; incoming light enters the material and is scattered inside, giving a translucent aspect. These effects are computationally intensive as they require simulating a large number of events. Full computations are expensive, even with accelerating methods such as Virtual Ray Lights. The dipole approximation [Jensen et al. 2001] is fast, but a strong approximation. Precomputing the material response for multiple scattering [Moon et al. 2007; Wang and Holzschuch 2017] integrates well with existing rendering algorithms, allowing separate computation for single- and double- scattering, and fast computation for multiple scattering. Their main issue is efficient storage for the precomputed multiple scattering data.

We present a method to encode multiple scattering effects using a neural network. We replace the precomputed multiple scattering table of [Wang and Holzschuch 2017] (40 MB) with a trained neural network, with a cost of 6490 bytes (1623 floats). At runtime, the neural network is used to generate multiple scattering. We demonstrate the effects combined with Virtual Ray Lights (VRL), but our approach can be integrated with other rendering algorithms.

2 PREVIOUS WORKS

Moon et al. [2007] precompute multiple scattering effects and store the result on a set of concentric spheres. Müller et al. [2016] extend the approach with varying sphere size, and use it to convert granular materials to heterogeneous media.

Wang et al. [2016] precompute multiple scattering in a 2D table, using the symmetry of revolution, and use it for point-based global
illumination with participating media. [Wang and Holzschuch 2017] combine this with many illumination simulation algorithms. Ren et al. [2013] used a multilayer acyclic feed-forward neural network to map scene data to indirect illumination.

3 NEURAL NETWORK MODEL FOR PRECOMPUTED MULTIPLE SCATTERING

3.1 Precomputing multiple scattering

First we compute multiple scattering effects assuming a light source with a dirac in position and direction in an infinite participating medium. The problem has symmetry of revolution: we store multiple scattering using cylindrical coordinates for position $(r, \rho, z)$ and spherical coordinates at each point for direction $(\theta, \phi)$ (see Figure 2). This step takes 8mn for a given material. [Wang and Holzschuch 2017] used this data to render participating media, combined with other algorithms for low-order scattering.

We treat it as a regression problem and train a neural network to learn the multiple scattering function $\Phi$, approximating it with $\Phi_N(\rho, z, \theta, \phi, w)$, where $w$ is the weights and biases of $\Phi_N$, found by minimizing:

$$E = \sum_i ||r_i - \Phi_N(\rho, z, \theta, \phi, w)||^2.$$  \hspace{1cm} (1)

3.2 Neural Network Model

We can see the precomputed scattering as a complex mapping from a 4 dimension domain (input coordinates, $(\rho, z, \theta, \phi)$) to a 3 dimension domain (the RGB channel), where the output function has an exponential falloff with the first two coordinates, and can be highly anisotropic for the last two coordinates (assuming anisotropic material).

We treat it as a regression problem and train a neural network to learn the multiple scattering function $\Phi$, approximating it with $\Phi_N(\rho, z, \theta, \phi, w)$, where $w$ is the weights and biases of $\Phi_N$, found by minimizing:

$$E = \sum_i ||r_i - \Phi_N(\rho, z, \theta, \phi, w)||^2.$$  \hspace{1cm} (1)

3.3 Neural Network Structure & Training

For our neural network, we used two fully connected hidden layers with 20 nodes each (see Figure 2). The networks were 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 precomputed table data from section 3.1 and use 70 % of the data for training, and the rest for validation. We normalize the input parameters to $[-1.0, 1.0]^3$ and shuffle them. We switch the output parameters to the log domain to make them well distributed. We train the models with 10,000 iterations. It takes 25 mn to train the network on a given material.

3.4 Neural Network Parameters

We normalize the input parameters to $[-1.0, 1.0]^3$ and shuffle them. We train the models with 10,000 iterations. It takes 25 mn to train the network on a given material. It takes 25 mn to train the network on a given material. We normalize the input parameters to $[-1.0, 1.0]^3$ and shuffle them. We train the models with 10,000 iterations. It takes 25 mn to train the network on a given material. We compute the loss as the difference between predicted radiance and computed radiance. The network is trained using the $L2$ error metric. We split the precomputed table data from section 3.1 and use 70 % of the data for training, and the rest for validation. We normalize the input parameters to $[-1.0, 1.0]^3$ and shuffle them. We switch the output parameters to the log domain to make them well distributed. We train the models with 10,000 iterations. It takes 25 mn to train the network on a given material.

REFERENCES


