Deep Real-time Volumetric Rendering Using Multi-feature Fusion

Jinkai Hu*
State Key Lab of CAD&CG, Zhejiang University
Hangzhou, China
568246876@qq.com

Chengzhong Yu*
Tokyo University of Science
Tokyo, Japan
yuchengzhongue4@gmail.com

Hongli Liu
Tencent, Inc.
Shanghai, China
bot@cuc.edu.cn

Lingqi Yan
Computer Science at UC Santa Barbara
Santa Barbara, United States of America
lingqi@cs.ucsb.edu

Yiqian Wu
State Key Lab of CAD&CG, Zhejiang University
Hangzhou, China
onethousand@zju.edu.cn

Xiaogang Jin†
State Key Lab of CAD&CG, Zhejiang University; ZJU-Tencent Game and Intelligent Graphics Innovation Technology Joint Lab
Hangzhou, China
jin@cad.zju.edu.cn

Figure 1: Multi-feature RPNN (MRPNN) renders multi-scattered cloud illumination in real time at 1024×1024 resolution, producing results close to the ground truth (a). Thanks to its novel network design, MRPNN supports configurable shading parameters (b), while the prior work only accepts fixed ones. G configures Henyey-Greenstein phase function. In addition, our network correctly handles the shadow boundary (c), which was previously a failure case.

ABSTRACT

We present Multi-feature Radiance-Predicting Neural Networks (MRPNN), a practical framework with a lightweight feature fusion neural network for rendering high-order scattered radiance of participating media in real time. By reformulating the Radiative Transfer Equation (RTE) through theoretical examination, we propose *transmittance fields*, generated at a low cost, as auxiliary information to help the network better approximate the RTE, drastically reducing the size of the neural network. The light weight network efficiently estimates the difficult-to-solve in-scattering term and allows for configurable shading parameters while improving prediction accuracy. In addition, we propose a frequency-sensitive stencil design in order to handle non-cloud shapes, resulting in accurate shadow boundaries. Results show that our MRPNN is able to synthesize indistinguishable output compared to the ground truth. Most importantly, MRPNN achieves a speedup of two orders of magnitude compared to the state-of-the-art, and is able to render high-quality participating material in real time.

CCS CONCEPTS

• **Computing methodologies** → **Ray tracing**; **Neural networks**.

KEYWORDS

Participating media, volumetric rendering, radiative transfer equation, real time, variable phase function

ACM Reference Format:
Jinkai Hu, Chengzhong Yu, Hongli Liu, Lingqi Yan, Yiqian Wu, and Xiaogang Jin. 2023. Deep Real-time Volumetric Rendering Using Multi-feature Fusion. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Proceedings (SIGGRAPH '23 Conference Proceedings), August 06–10, 2023, Los Angeles, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3588432.3591493

*Authors contributed equally.
†Corresponding author.

SIGGRAPH '23 Conference Proceedings, August 06–10, 2023, Los Angeles, CA, USA

ACM ISBN 979-8-4007-0159-7/23/08…$15.00
https://doi.org/10.1145/3588432.3591493

1 INTRODUCTION

Participating media exists as a ubiquitous form of material observed widely in the real world, for example, in cloud, milk, jade, skin, and so on. Correctly handling the interaction between these materials and light greatly enhances the realism of rendering results. This subject has been the focus of numerous investigations, in the context of general light transport (e.g. bidirectional path tracing [Lafortune and Willems 1996], Metropolis methods [Pauly et al. 2000]), or specifically tailored to rendering participating media (e.g. photon beams [Jarosz et al. 2008], photon surfaces [Deng et al. 2019] and volumetric path guiding [Herholz et al. 2019]). However, since light could hardly be absorbed before undergoing thousands of bounces in the volume of participating media, especially those of low absorption rates such as clouds, the rendering process takes considerable time to converge to noise-free results. Researchers have been pursuing more effective approaches in recent years. While some of these approaches refer to diffusion theory and/or pre-computation, the line of research starting from Kallweit et al. [2017] introduced the Radiance-Predicting Neural Networks (RPNN). They achieved notable performance improvements and showed how neural networks (NN) can be used to estimate in-scattering irradiance.

Despite their accomplishments, the brute-force architecture of RPNN and its variants limits their performance and robustness. First, because RPNN receives only a few density samples as input and uses a fully connected structure, it is difficult for the network to infer the underlying physical principle of light transport. One clue is that RPNN does not correctly handle shadows (see Fig. 1). In order to map this extremely complicated function, excess neural connections were used, which resulted in bloated network structures and lower performance. Second, the shading parameters in RPNN are hardcoded. Changes in phase values or albedos triggers retraining of the network. The straightforward structure of RPNN merely encodes all scenarios in which different shading parameters push the dimension to an impractical level.

Our motivation is to create a network architecture that better approximates the solution to the RTE. We started by reformulating the RTE through theoretical examination. The investigation reveals that the media could be decomposed into multiple features, rather than being directly fed into the networks. The decomposed features include a set of density fields, pre-integrated phase values, and albedos (see Fig. 10). The decomposition leads to a smarter and fewer-shot network architecture with faster inference and helps achieve configurable parameters. Additionally, we suggest a new sample stencil with two parts, each concentrating on either low frequency (shadow-aware) or high frequency (diffusion-aware) information. The network thus achieves better results in shadow boundaries.

Based on the observations above, we offer Multi-feature Radiance-Predicting Neural Networks (MRPNN), a framework for fusing features extracted from the RTE and predicting the radiance in real time using a lightweight network.

Through experiments, we verify that our method achieves an order of magnitude performance boost over RPNN, configurable shading and better generalization capability in non-cloud shapes, which can never be achieved by previous work. With MRPNN, we are able to generate realistic volumetric rendering in real time.

In particular, our paper makes the following contributions:

- Reformulation of the radiance-predicting problem from an intricate mapping to a simpler one through multi-feature input, allowing us to significantly simplify the neural network structure while enabling dynamically adjustable albedos and phase parameters.
- A lightweight radiance-predicting framework that requires much less computation to achieve a real-time volumetric rendering solution, with better results on non-cloud shapes and the shadow boundary, and better quality compared to RPNN.

2 RELATED WORK

Monte Carlo Integration. Various approaches based on Monte Carlo (MC) integration have been developed to render participating media. Solving the RTE, i.e. finding all potential light paths connecting the light sources, the camera, and the medium vertices, is central to these approaches. For example, bidirectional path tracing [Lafortune and Willems 1996] generates rays from both the light sources and the camera to explore the path space. In order to efficiently find those paths with higher contribution, Pauly et al. [2000] introduced Metropolis Light Transport to media rendering. To reduce the estimation variance and increase light path utilization, the photon-based method was introduced to benefit efficiency [Jarosz et al. 2008], and further improved by Jarosz et al. [2011]. These schemes were unified into one framework to achieve a more robust integrator [Křivánek et al. 2014]. Although these methods are unbiased and they all speed up the rendering process considerably, they are still time consuming and can only be used for offline rendering [Kallweit et al. 2017].

Diffusion Theory. Diffusion estimators, in addition to MC integrators, are another type of RTE solvers for efficiently capturing multi-scattered volumetric illumination [Stam 1995]. The intention of diffusion-based methods is to address the multi-scatter issue of MC integrators. A foretype, the Flux-Limited Diffusion (FLD) proposed by Koerner et al. [2014] is built based on Classical Diffusion Approximations (CDA) for more accuracy. Diffusion-based approaches [Jensen et al. 2001] address the high-order problem, but they are hampered by other concerns that prevent them from being used in real time. Their computational costs, for example, are polynomially dependent on grid resolution, which is also non-constant. However, with insufficient resolution, the bias would be distinguishable. Furthermore, because they naturally do not support progressive rendering, it is difficult to distribute the computational load across frames.

Learning-based methods. Several works have employed learning-based approaches for rendering tasks. To enhance the efficiency of rendering BSSRDFs, neural networks have been employed to predict the exit point position of an object after internal scattering [Vicini et al. 2019], or to directly estimate the contributions of all possible paths between two specific points in homogeneous media [Leonard et al. 2021]. Mildenhall et al. [2020] utilize neural radiance fields to represent scenes and perform rendering through ray marching. Neural radiance fields are further utilized to accelerate the convergence of path tracing [2021]. These methods offer valuable insights into the application of learning techniques for rendering purposes.

Radiance Prediction Neural Networks. Researchers have been looking for approximate but more efficient MLP-based methods to achieve faster performance. The research starting from Kallweit et al. [2017] has drawn our attention. They were the first to introduce neural networks to estimate the challenging in-scattering term. They have successfully avoided the time-consuming process of tracing numerous light paths, and thus is able to synthesize images of clouds within minutes. The performance of this method is further improved by $2 \sim 3\times$ by caching the latent vectors [Panin and Nikolenko 2019].

In essence, the aforementioned neural solutions both consider light transport as a mapping from a density field to a radiance field. Since the mapping can be very complex, the neural network must be *large enough* to approximate it, which makes evaluation time-consuming. For example, consider the intricate visibility relationships among the voxels. Since we are requiring the network to produce Dirac weights for the intermediate voxels, this task is incredibly challenging for network estimators. However, it is easy with ray-marching. The task of the network may be streamlined by using the simple-to-compute features as additional input. Sec. 3 and 4 begin with theoretical deduction and then conclude with the detailed framework. We demonstrate that with proper decomposition of the input features and tailored neural network structure, it can converge with a very limited number of training data, indicating that it understands the underlying physical principle of light transport rather than brute force. In Sec. 5, we validate our claim through a series of experiments.

3 RENDERING WITH MULTI-FEATURES

As shown in Alg. 1, our framework contains four steps: 1) Sample a light direction (for ambient light), or get the direction of a given directional light. 2) Prepare the transmittance features. 3) Perform delta tracing, predict and accumulate 1-SPP radiance. 4) Repeat from Step 1 (for ambient light), or repeat from Step 3 (for directional light).

In essence, we assume neural networks can correctly predict in-scattering radiance at any arbitrary location within media with appropriately prepared inputs, allowing us to accumulate it along the light path. Given a starting point x, ray direction ω and light vector l, we generate several sample points $\mathbf{u}$ along the ray using delta tracking [Woodcock et al. 1965]. Then, at each sample point, we predict the in-scattering radiance and accumulate it. Before prediction, we sample the multi-feature $\mathbf{s}$ according to a **stencil** pattern, which assembles the descriptor and will be fed into the neural network. The *descriptor* is the network's input, which specifies the optical context surrounding a given location. The *stencil*, accordingly, is a set of discrete relative positions that describes the points.

In the following sections, we will first demonstrate how to extract the multiple features from the Radiance Transport Equation (RTE) [Kajiya and Herzen 1984] by reformulating it as the sum of contributions from paths of different lengths. The results indicate that the extracted features may be useful in network estimation. We then demonstrate how to decompose the transmittance fields (see Sec. 3.2), which is necessary for network compacting. Then, in order to support configurable shading parameters, we extract phase and albedo (see Sec. 3.3). Following that, we propose a new frequency-aware stencil for collecting data from the sample point's surrounding discrete points while achieving better shadow boundaries (see Sec. 3.4). Finally, we present the lightweight radiance prediction framework (see Sec. 3.5).

ALGORITHM 1: MRPNN::Render(x, ω, l)

```
L ← 0
for N := 1 to TotalSamples do
    (u, hit) ← GetSamplePointWithDeltaTracking(x, ω)
    if hit == true then
        s ← ApplyStencil(u, ω, l)
        descriptor ← SampleAndGenerateDescriptor(s)
        L ← L + PredictRadiance(descriptor) + DirectLight(x, l)
    else
        L ← L + SampleSkyBox(ω)
    end if
end for
return L/TotalSamples
```

3.1 RTE

We begin the theoretical investigations of the RTE by reviewing its classic formulation. The RTE depicts the differential change of the radiance L traveling through a medium at position x in direction ω:

$$\begin{aligned}\frac{\partial}{\partial \omega} L(x, \omega) &= L_e(x, \omega) - \mu_t(x) L(x, \omega) \\ &+ \mu_s(x) \int_\Omega p(\omega, \omega_i) L(x, \omega_i) \mathrm{d}\omega_i.\end{aligned} \tag{1}$$

The right-hand side of RTE breaks down into 3 terms:

(1) **Self-emission term.** L_e represents the differential increment of light contributed by the particles, such as chemiluminescence and nuclear luminescence, which are rare in real life. We ignore this term for the sake of simplicity.
(2) **Extinction term.** The extinction term is defined by the total amount of light being absorbed or out-scattered, where $\mu_t = \mu_a + \mu_s$ is the extinction coefficient; μ_a and μ_s are the absorption and scattering coefficients, respectively.
(3) **In-scattering term.** The in-scattering term collects the inbound light scaled by the phase function p from the spherical neighbor Ω.

We use the term albedo (denoted by ς) to refer to the amount of light being re-scattered rather than absorbed when a collision occurs. We assume that the albedo of the medium is *constant* through the space, i.e. $\frac{\mu_s(x)}{\mu_t(x)} \equiv \varsigma$. Ignoring the emission term, the integral form [Arvo 1993] of Eq. (1) is simplified to:

$$L(x, \omega) = \int_x^z T(x, u) \mu_s(u) S(u, \omega) \mathrm{d}u + T(x, z) L_s(z, \omega), \tag{2}$$

where $z = \lim_{t \to \infty} x - t \cdot \omega$, L_s is the in-coming radiance from z toward ω, and the *transmittance* term T is:

$$T(x, y) = e^{-\int_x^y \mu_t(u) \mathrm{d}u}, \tag{3}$$

and the integral part of the *in-scattering* term S is:

$$S(x,\omega) = \int_\Omega p(\omega,\omega_i)L(x,\omega_i)\mathrm{d}\omega_i. \tag{4}$$

Here we assume that the lights are distant (e.g. the sun). Other light sources may be effective, but they are out of our scope. Under this condition, the L_s term from Eq. (2) can be written as:

$$L_s(x,\omega) = \delta(\langle l,\omega\rangle + 1)\cdot I, \tag{5}$$

where $\delta(\langle l,\omega\rangle + 1)$ is the Dirac function asserting that light is only coming from direction l, and I is the light intensity. We will also use $T(x,\omega)$ in short of $\lim_{b\to\infty} T(x, x+b\cdot\omega)$.

Plugging Eq. (2) and Eq. (5) into Eq. (4), we get:

$$\begin{aligned} S(x,\omega) &= \int_\Omega \left[\left(p\int_x^z T\mu_t\varsigma S\mathrm{d}u\right) + pTL_s\right]\mathrm{d}\omega_i \\ &= \varsigma\int_\Omega p\int_x^z T\mu_t S\mathrm{d}u\mathrm{d}\omega_i + p(\omega,l)T(x,l)\cdot I. \end{aligned} \tag{6}$$

Note that the S term appears on the both side of Eq. (6). Following the notation of Veach [1998], we employ an operator $\mathbf{K}$ which is defined as $(\mathbf{K}h)(x,\omega) = \int p\int T\mu_t h(x,\omega)\mathrm{d}u\mathrm{d}\omega_i$ and a set of in-scattering intensity field $\mathcal{S}_{i,i\geq 0}$ being the order of scattering. The RTE can be reformulated as:

$$\begin{aligned} &\mathcal{S}_0 = p(\omega,l)T(x,l)\cdot I, \quad \mathcal{S}_i = \mathbf{K}\mathcal{S}_{i-1}, \\ &\mathcal{S} = \sum_{i=0}^{\infty}\varsigma^i\mathcal{S}_i = \mathcal{S}_0 + \varsigma\mathcal{S}_1 + \varsigma^2\mathcal{S}_2 + \varsigma^3\mathcal{S}_3\ \cdots. \end{aligned} \tag{7}$$

Eq. 7 shows that the RTE can be rewritten as an additive series. When given approximated $\mathcal{S}_i$ as hints, the network can infer the final result $\mathcal{S}$ more easily.

Splitting the in-scattering field. To approximate $\mathcal{S}_i$, we immediately hit the first problem. Note that for $i > 0$, the operator $\mathbf{K}$ involves intricate integration over both position x and direction ω, raising a problem of too many dimensions, which causes difficulties during both training and runtime. To simplify the problem, we'd better separate the directionally invariant part $\mathcal{S}'_i$ and the remaining part P_i, where $\mathcal{S}_i(x,\omega) = P_i(x,\omega)\cdot\mathcal{S}'_i(x)$. Note that P_i is impossible to be separated through formulation; its intention is a scale factor that accounts for the phase function (directionally invariant part), which will be later discussed.

Directionally invariant part. According to Eq. (7), the phase term p in operator $\mathbf{K}$ is ignored, such that $\mathcal{S}'_i$ is directionally invariant:

$$\begin{aligned} &\mathcal{S}'_0 = T(x,l)\cdot I, \quad \mathcal{S}'_i = \mathbf{K}'(\mathcal{S}'_{i-1}), \\ &\mathcal{S} = \sum_{i=0}^{\infty}\varsigma^i P_i\mathcal{S}'_i. \end{aligned} \tag{8}$$

Eq. (8) reveals the three valuable physically-based features hidden beneath the RTE: the directionally invariant part $\mathcal{S}'_i$, phase P_i and albedo ς^i. It should be noted that S is not a simple combination of the extracted features, but a neural network conditioned on these features that can well approximate S. To this end, our next goal is to extract features from the three terms.

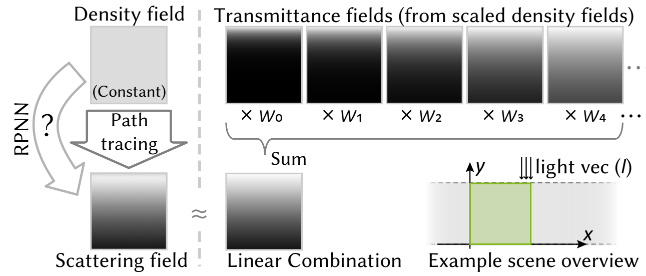


Figure 2: A toy example case. The light enters from $+y$, and the density field extends to infinity in $\pm x$. Left: we are using the RPNN to predict the scattered field from a constant density field, which is hard. Right: given the hint transmittance fields, the result could be easily deduced.

3.2 Transmittance Field

To approximate the directionally invariant part $\mathcal{S}'_i$ in Eq. (8), we propose to generate certain features by applying $T(x,l)$ (defined in Eq. (3)) to density fields. Fig. 2 demonstrates our intuition, integrating the scattering field over a homogeneous volume with a *constant* density field. In this simple setting, an unbiased estimator produces non-constant radiance. Given that the input is spatially constant, it would be hard for a network to infer the relationship (e.g., RPNN). Considering that light travels farther into the media due to diffusion, which is similar to traveling through a down-scaled density volume, we first increasingly down-scale the constant density field. Then we generate a set of fields by applying $T(x,l)$ to those down-scaled density fields. The scattering field can be easily approximated using a linear combination of the generated fields, indicating that we can leverage $T(x,l)$ and the scaled density fields to approximate $\mathcal{S}'_i$. Furthermore, as the diffusion operator $\mathbf{K}'$ in Eq. (8) could be approximated by applying $T(x,l)$ to density mipmap [Wrenninge et al. 2011], we propose to use scaled density mipmaps instead of scaled density fields.

Given the above analysis, we apply $T(x,l)$ to scaled density mipmaps, yielding the so-called ***transmittance field***:

$$\widetilde{\mathcal{S}}_i = e^{-\int_x^y \beta^{(i+1)}\mu_i(u)\mathrm{d}u}, \tag{9}$$

where μ_i denotes the i-th level of mipmaps generated from the density field, which is down-scaled by a hyper-parameter $\beta^{(i+1)}$ $(0 < \beta < 1)$. Therefore, we can use our transmittance fields $\widetilde{\mathcal{S}}_i$ to hint the directionally invariant part $\mathcal{S}'_i$. This intuition will be further validated by experiments. By providing transmittance fields, the network better understands the mapping. Since the approximated transmittance fields are being increasingly smoother in the spatial domain, we can use gradually reduced spatial resolution to calculate and store them, yielding the so-called *mipmaps*. More details will be addressed in Sec. 4.1.

3.3 Phase and Albedo

The remaining components of Eq. (8) to be addressed are the *phase* term P_i and *albedo* term ς^i.

Phase. Reminiscent that the goal of the phase feature is to determine a scale factor to assist the network in learning the contribution of phase functions. In practice, we adopt $P_i = p'(\omega, \mathbf{v}_i - \mathbf{u}) \cdot p'(\mathbf{v}_i - \mathbf{u}, l)$ as the phase feature [1], where $\mathbf{v}_i$ is the i^{th} stencil point and p' is the cumulative version of phase function (detailed implementation will be given in the supplementary material) over the volume of the point.

Albedo. In Eq. (8), those powers of albedo (ς^i) will only appear as the weights of the combination of the $\mathcal{S}_i$. We use ς as the feature of the albedo (and the remaining $\{\varsigma^2, \varsigma^3, ...\}$ are superfluous).

We should mention that by separating the features as in Eq. (8), the phase and albedo are configurable parameters that can be dynamically adjusted in testing.

3.4 Frequency-Sensitive Stencil Pattern

Inputting the entire field into the network can lead to difficult optimization and over-parameterization of the network. To solve this problem, we need a stencil pattern sampling to collect information from the sample point's surrounding discrete points. We design a frequency-sensitive stencil pattern according to the decomposition of global-local multiple scattering [Zinke et al. 2008]. Our stencil has two parts, one for high-frequency shadow boundary and the other for low-frequency diffusive scattering. We use stratified uniform spherical distributions for the low-frequency part, and use a cone shape towards light direction for the high-frequency part. Compared to a naïve uniform stencil pattern (e.g., the grid-lattice used in RPNN), our frequency-sensitive method can better predict non-cloud shaped media and shadow boundaries.

The i^{th} layer in the stencil is denoted by $Q_i = \{\boldsymbol{q}_{i,1}, \boldsymbol{q}_{i,2}, ..., \boldsymbol{q}_{i,N_i}\}$, where $\boldsymbol{q}_{i,j}$ is the j^{th} stencil point in the i^{th} layer and N_i is the number of the points in this layer. Our stencil consists of totally $K = 12$ layers, which is split into a *low-frequency part* $Q_1, ..., Q_M$ and a *high-frequency part* $Q_{M+1}, ..., Q_K$ where $M = 8$. Each layer corresponds to a mip-level m_i. The details of each single layer of stencil design will be given in the supplementary material.

3.5 Two-Stage Network

Our design motivation is based on the network's ability for estimating the in-scattering radiance, where the network is a function:

$$\widetilde{S}_x^* (\boldsymbol{z}, \Theta) : \mathbb{R}^d \to \mathbb{R}, \tag{10}$$

which maps a descriptor $\boldsymbol{z}$ and a set of parameters Θ to an estimated radiance value in the spectral channel x.

To adapt the proposed multi-feature inputs and frequency-sensitive stencil, we separate the entire network into two stages, namely the *feature* and the *albedo* stage (see Fig. 3). S'_i (transmittance field) and P'_i (phase) are responsible for estimating $\mathcal{S}_i$ and will be fed into the first stage. ς^i is responsible for combining $\mathcal{S}_i$ of different spectra and will be fed into the second stage. Because the *feature* stage is albedo-independent, it only needs to be run once in spite of the number of spectral channels, which reduces the computational cost of the network. As we use the RGB color space, the *albedo* stage will be executed 3 times to compose the final RGB output (i.e., for each of $\varsigma_{x, x\in\{r,g,b\}}$). Note that thanks to the assistance of RTE-based features, our network is light-weight, contains only $\sim 50k$ parameters (approximately 83% of LeNet-5's, 0.2% of ResNet-50's and 0.03% of VGG-16's), and has fast inference speed (frame cost $\leq$ 33.3ms). More details can be found in Sec. 4.3.

[1] The phase feature of the central point $\mathbf{v}_0 = \mathbf{u}$ is $P_i = p'(\omega, l)$

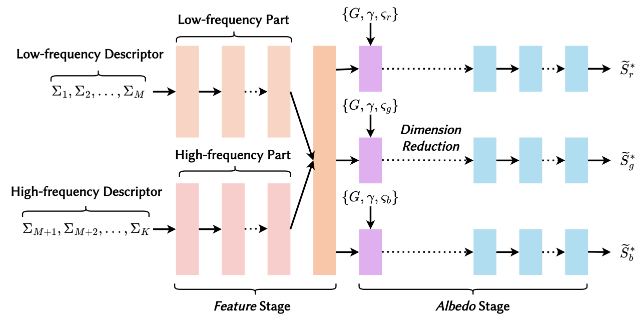


Figure 3: An overview of the network structure.

4 IMPLEMENTATION DETAILS

In this section, we will first show how to implement feature extraction (Sec. 4.1). Then, we demonstrate the detailed construction of the network's inputs (Sec. 4.2). Following that, we present the architecture of our lightweight network (Sec. 4.3). Finally, we display our training details (Sec. 4.4).

4.1 Feature Extraction

Density mipmaps. The mipmap layers are generated efficiently by the hardware's bi-linear interpolation. In our case, the 9 layers of mipmaps begin at the input μ_0 fields with a resolution of 256^3 and end at μ_8 with with a resolution of 1^3.

Transmittance fields. We employ the ray-marching [Drebin et al. 1988] technique to compute the transmittance fields. For each voxel, we generate several uni-spaced sample points towards the light source depending on the voxel resolution. The above process is executed on each mip-level i of the density maps, where the density field is scaled by $\beta^{(i+1)}$ $(0 < \beta < 1)$ during the estimation (which is suggested in Sec. 3.1), and we emprically set $\beta = 0.578$.

Phase function. We use the Henyey-Greenstein (HG) phase function as the volume-averaged phase function. The HG phase function is dynamically controlled by a hyper-parameter G. In addition to the HG phase function, our framework is capable of rendering other phase functions, e.g., the Lorenz-Mie phase function (see Fig. 9). Detailed implementation will be given in the supplementary material.

4.2 Descriptor

In response to the stencil, the descriptor is built utilizing features that describe a point and its illumination context in the volume. The j^{th} feature out of stencil Q_i (or $\boldsymbol{q}_{i,j}$) is:

$$\boldsymbol{F}_{i,j} = \{F_{i,j}^{\mu}, F_{i,j}^{S}, F_{i,j}^{P}\},$$

where $F_{i,j}^{\mu}$ and $F_{i,j}^{S}$ are the density and the scaled-transmittance samples at the stencil point $v_{i,j} = u + \boldsymbol{q}_{i,j}$ with the corresponding mip-level m_i, and $F_{i,j}^{P}$ is the phase. In practice, we use $log\left(F_{i,j}^{\mu} + 1\right)$

and $log\left(F_{i,j}^{P}+1\right)$ to compress the ranges of the values (which is trivial and will be omitted in the rest).

The features in each stencil layer form a single-layer descriptor

$$\Sigma_i = \{\boldsymbol{F}_{i,1}, \boldsymbol{F}_{i,2}, ..., \boldsymbol{F}_{i,N_i}\}, \tag{11}$$

and finally the descriptor is established through:

$$z = \{\Sigma_1, \Sigma_2, ..., \Sigma_M, \Sigma_{M+1}, ..., \Sigma_K, G, \varsigma^{\alpha}, \gamma\}, \tag{12}$$

where G is the parameter of the HG phase function, ς is the albedo, α is a hyper-parameter (detailed in Subsec. 4.4), and $\gamma = \cos^{-1}(\omega \cdot l)$ is the angle between the view and the light. We assume that the density outside the volume boundary is 0. However, the scaled-transmittance fields should be given special consideration. More information can be found in the supplement file.

4.3 Network Architecture

Feature processing. We have introduced multi-features (density, scaled-transmittance, and phase) as the input to the network. To handle these inputs more efficiently, we employ the Squeeze-and-excitation Module (a.k.a., the *SE module* [Hu et al. 2018]) for each layer. The ablation tests in Sec. 5.2 show that the SE module produces more stable and better results than other methods (e.g., fully-connected layers).

Feature stage. Its purpose is to compute the latent space vectors for the input feature. We introduce a two-part network in the feature stage to fuse stencils from different frequencies in order to investigate the high- and low-frequency parts separately. It is based on 2 individual sub-networks in connection with the low-frequency part of the descriptor $\Sigma_1, ..., \Sigma_M$ and the high-frequency part $\Sigma_{M+1}, ..., \Sigma_K$. Each layer of descriptor is first passed into the SE module, then is combined with the output of the former block by addition. We feed individual stencil layers progressively to the network with a residual connection.

Albedo stage. In this stage, our goal is to introduce albedo to the network, fuse the low- and high-frequency information and compute the final output. We first use a SE module to scale the input latent vectors from the previous stage. In order to accelerate the inference process, a dimension reduction operate is perform. Finally we use several fully-connected layers with residual connection to compute the output.

We will provide the low-level details of network architecture and hyper-parameter setting in the supplementary material.

4.4 Training

To train our network, we employ a supervised learning scheme. We define the collection of descriptors and labels (ground-truth) by:

$$\mathcal{D} = \{(\boldsymbol{z}_1, S(\boldsymbol{u}_1, l_1)), ..., (\boldsymbol{z}_N, S(\boldsymbol{u}_N, l_N))\}, \tag{13}$$

where N is the size of the collection. Our goal is to find a Θ that minimizes the average loss between predicted and target values:

$$\Theta^* \in \underset{\Theta}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} \mathscr{L}\left(\widetilde{S}^*(\boldsymbol{z}_i; \Theta), S(\boldsymbol{u}_i, l_i)\right). \tag{14}$$

We use a mean-square-error (MSE) based loss function as per RPNN's design with albedo term ς_x included:

$$\mathscr{L}_{\mathcal{B}} = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \left(log\left(\frac{\widetilde{S}^*(\boldsymbol{z}_i; \Theta)}{\varsigma_x^{\alpha}} + 1\right) - log\left(\frac{S(\boldsymbol{u}_i, l_i)}{\varsigma_x^{\alpha}} + 1\right)\right)^2, \tag{15}$$

where $\mathcal{B}$ is a minibatch during training and $\alpha \geq 1$ is a parameter. On the one hand, the log transforms [Yeo and Johnson 2000] considerably condense the range of the radiance, which speeds up the training [Bako et al. 2017]. It also prevents the artifacts of bright mutations induced by high-frequency phase functions. On the other hand, introducing α normalizes the loss function, and when $\varsigma_x > 0$, the input data is closer to normal distribution [Box and Cox 1964]. We recommend using $\alpha = 4$ by experimenting with range $[1, 8]$ and a step size of 0.5. In this situation, the skewness of all training label $log\left(\frac{S(\mathbf{u}_i, l_i)}{\varsigma_x^{\alpha}} + 1\right)$ is much closer to 0.

Data Generation. Using manual and procedure methods, we generated 26 cloud-shaped volumetric models and created 7 others in non-cloud shape for training, validating and testing, all of which were resized to suit their bounding box, and stored with a resolution of $[512]^3$. 6 cloud-shaped models and 7 non-cloud shaped models were chosen at random for testing, these models are not used for training or validation. And the remaining 20 models in cloud-shaped were used for training and validation.

We generated 250k groups of samples $(\boldsymbol{z}_i, S(\boldsymbol{u}_i, l_i))$ for each model, totaling 5M. Then, in a 4:1 ratio, we partitioned these into two non-overlapping subsets for training and validation.

For each model, we randomly generated 50 groups of different parameters $\{l_j, \epsilon_j\}$, namely the light direction, and a parameter that scales entire density field. And for each sample we randomly select parameters $\{G_j, \varsigma_j\}$, where G_j is the parameter G in the HG phase function, ς_j is a single channel of the albedo.

Training Configuration. We employ the *Stochastic Gradient Descent* technique with Adabound optimizer [Luo et al. 2019], with a learning rate $\alpha = 0.001$ and $\alpha^* = 0.1$. To enhance generalization ability, we use a minibatch of the size of $|\mathcal{B}| = 64$. We also use a validation set to evaluate and monitor the convergence process in order to select a superior training outcome.

5 EXPERIMENTS AND ANALYSIS

We tested our Multi-feature RPNN against the naïve RPNN as the baseline, and an unbiased path tracer as the reference. All tests were carried out on an Nvidia RTX 2080 GPU with a 1024^2 resolution (exceptions explicitly noted). Our training time is about 4 days. The resolution of volumetric data is 1024^3. "Cloud" refers to a cloud-shaped configuration, and "Model" to one created from a specified geometry. All configurations are demonstrated in Fig. 4.

Objective 1: an equivalent bias. In the first two columns of Tab. 1, we show that the MRPNN preserves an equivalent bias level compared with RPNN. The bias was measured through the RMSE of the renderings. Based on the bias test, we have proof-of-concept that the auxiliary transmittance fields are critical for minimizing the network's complexity in response to our motivation in Sec. 3.2. A speedier framework is achievable since the network's dimensionality and necessary samples are considerably decreased.

Table 1: Bias and performance test results. Note that the neural network-based approaches are cost-consistent with change of light direction, whereas the reference (MC estimator) fluctuates in time. In the convergence test, we used 64SPP for both RPNN and MRPNN, and adaptive samples depending on the noise for the reference. Note that rendering takes much less time (≤ 0.5ms) than network inference. (See supplementary material for full table)

Model	Light Dir.	Bias		Frame Cost (ms)		Convergence Boost	
		Ours	RPNN	Ours	RPNN	RPNN	Ref.
Cloud0	Side	1.55e-2	1.85e-2	5.2	597.5	114.9 ×	1812.5 ×
	Front	2.21e-2	1.75e-2				1812.5 ×
	Back	3.52e-2	4.28e-2				1562.5 ×
Model0	Side	1.80e-2	2.18e-2	5.0	447.0	89.4 ×	837.5 ×
	Front	1.86e-2	2.20e-2				628.1 ×
	Back	2.08e-2	6.83e-2				487.5 ×

Table 2: Parameter scale and feature comparisons. Supported features are marked ✓and unsupported ✗. Note that the LUTs column refers to whether a variant uses the volume-averaged phase function.

Model	Parameters	μ	S	P	LUTs	High Freq.	SE-Module
MRPNN (Ours)	49.7 k	✓	✓	✓	✓	✓	✓
MRPNN-Var1	49.7 k	✓	✓	✓	✗	✓	✓
MRPNN-Var2	49.7 k	✓	✗	✗	✗	✓	✓
MRPNN-Var3	49.7 k	✓	✓	✗	✗	✓	✓
MRPNN-Var4	49.7 k	✓	✗	✓	✗	✓	✓
MRPNN-Var5	49.7 k	✓	✓	✓	✓	✗	✓
MRPNN-Wide	80.4 k	✓	✓	✓	✓	✓	✗
MRPNN-Narrow	47.3 k	✓	✓	✓	✓	✓	✗
RPNN	1296.4 k	✓	✗	✗	✗	✗	✗
RPNN-Var1	1296.7 k	✓	✓	✗	✗	✗	✗
RPNN-Var2	1296.7 k	✓	✗	✓	✗	✗	✗
RPNN-Var3	1298.3 k	✓	✓	✓	✗	✗	✗

Objective 2: real-time performance. In the last four columns of Tab. 1, we tested MRPNN's performance against RPNN and the reference. Each frame's cost is made up of the network's inference time and the rendering time. It's worth noting that rendering time for MRPNN and RPNN is nearly identical, which is significantly less (frame cost ≤ 0.5ms) than inference time. Our MRPNN was able to render in real time (frame cost ≤ 33.3ms), and is order(s) of magnitude faster than both RPNN and the reference, which is to be expected given the network's compact design.

In addition, Tab. 2 compares the parameter scale for MRPNN and RPNN. The variations of both approaches differ in the descriptors and structure, which will be used in later ablation tests. The parameter scale of MRPNN is only approximately 3.8% of RPNN's, which explains the faster inference speed.

Objective 3: generalization ability in varying shading parameters. With an initial intent to generalize the shading parameters which were previously hardcoded into the network in RPNN, we tested our framework under various phase parameters G and albedos as listed in Tab. 3. Note that RPNN does not handle non-uniform ($\varsigma \neq \{1.0, 1.0, 1.0\}$) albedos or varying G and is marked "n/a".

Table 3: Check experiment of the biases with different shading parameters.

RMSE $\times 10^2$		Side		Front		Back	
Model	Parameter	Ours	RPNN	Ours	RPNN	Ours	RPNN
Cloud0	$\varsigma = \{1.0, 1.0, 1.0\}$	1.55	1.85	2.21	1.75	3.52	4.28
	$\varsigma = \{0.96, 0.98, 1.0\}$	1.38	n/a	2.08	n/a	2.57	n/a
	$\varsigma = \{0.8, 0.9, 1.0\}$	1.51	n/a	2.49	n/a	1.55	n/a
Model0	$\varsigma = \{1.0, 1.0, 1.0\}$	1.80	2.18	1.86	2.20	2.08	6.83
	$\varsigma = \{0.96, 0.98, 1.0\}$	1.35	n/a	1.57	n/a	2.60	n/a
	$\varsigma = \{0.8, 0.9, 1.0\}$	1.20	n/a	2.21	n/a	1.76	n/a
Cloud0	$G = 0.857$	1.55	1.85	2.21	1.75	3.52	1.75
	$G = 0.5$	2.13	n/a	2.00	n/a	3.91	n/a
	$G = 0.0$	3.10	n/a	1.94	n/a	4.04	n/a
Model0	$G = 0.857$	1.80	2.18	1.86	2.20	2.08	2.20
	$G = 0.5$	1.43	n/a	1.74	n/a	1.93	n/a
	$G = 0.0$	2.32	n/a	1.53	n/a	2.40	n/a

Objective 4: ability of generalization in non-cloud objects. In the second row of Tab. 1, we test an artist-created model, which is a non-cloud object with regular boundaries. By comparing the RMSE biases with that of the reference, we can see that in the majority of cases MRPNN quality is better than that of RPNN. Also we achieve the same level of bias as clouds for rendering non-cloud objects. It can therefore be said that MRPNN outperforms RPNN in the generalized rendering of non-cloud objects.

Faster convergence. We tested MRPNN and RPNN's training convergence based on the same number of nodes (200 as the original RPNN), Adabound optimizer, initial learning rate and weight decay, data configuration, training set size, and validation sets with the split ratio of 4:1. Fig. 5 shows the results, where our MRPNN achieves apparently better convergence. This profits from the parameter scale of our network: RPNN uses approximately 1.3 million trainable parameters due to the density-only design, while MRPNN contains only about 50,000 trainable parameters. Consequently, MRPNN learns faster while converging with less error.

5.1 Examining the SE Modules

In Sec. 4.3 we propose to use SE modules to help the network fuse the feature channels. To support this, we implement two other networks. MRPNN-Narrow is achieved by removing the SE Module in MRPNN, and MRPNN-Wide is achieved by replacing the SE Module in MRPNN with fully-connected layers, detailed architectures are listed in the appendix.

As shown in Fig. 5, MRPNN-Narrow has no evident quality or speed change in runtime, but its convergence is slower than MRPNN. In MRPNN-Wide, features are fused at each layer by brute force using large matrices, causing a $2 \sim 3$ times slower runtime performance. Also, due to considerably increased trainable weights, the training convergence is slower. Given the above, the SE module can better fuse the inputting features.

5.2 Ablation Tests

In addition to the wide and narrow variants, we implemented several more variants, each of which disables some features while ensuring the same number of parameters (as detailed in Tab. 2). Fig. 6 presents the results.

Volume-averaged phase functions (LUTs for short). MRPNN-Var1 with non-volume-averaged phase converges slightly slower, indicating that LUTs provide the network with more precise context.

Transmittance fields. We removed the transmittance fields in MRPNN-Var2 and MRPNN-Var4. Note that both variants are considerably harder to converge, which again supports our proposal to introduce the transmittance fields.

Split stencil. We removed the high-frequency part of the stencil in MRPNN-Var5. As expected again, the convergence rate decreases. The high-frequency part not only addresses the shadow boundary, but also speeds up the training.

Modified RPNN. In addition to our network, we integrate the features into the RPNN to validate their effectiveness. We implement (approximately) the same number of trainable parameters. Based on the results, we can conclude that our proposals are even effective in the original RPNN architecture. This again supports our claims.

6 DISCUSSION AND FUTURE WORK

In this paper, we offer a novel framework to render high-fidelity participating media in real time, which is nearly 2 orders of magnitude faster than the state-of-the-art thanks to the compact network architecture and fewer required samples. Also, our stencil design addresses the shadow boundary. Furthermore, our framework is able to generalize different shading parameters by concatenating new features, as suggested by our investigation into the RTE.

Our approach has limitations. First, the use of mipmaps may cause issues in highly fragmented portions of the volume, resulting in more biased brightness (see Fig. 7). Introducing new auxiliary features, such as the variance of down-scaled densities, may aid in addressing this issue. Second, because the scaled transmittance fields in our case are directionally consistent, stochastic progressive sampling and denoisers are disabled (if any) (see Fig. 8). Designing new ambient-oriented features, such as spherical harmonics encoded with lighting information [Kaplanyan and Dachsbacher 2010], could be a solution. Finally, we assumed that the albedos and phase parameters are homogeneous over the entire volume, which could not fulfill all potential types of materials. A more in-depth examination of the RTE and a better design of the descriptor may alleviate this issue.

ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China (Grant No. 62036010), and the Key R&D Program of Zhejiang (No. 2023C01047). Ling-Qi Yan is supported by gift funds from Adobe, Intel, Meta and XVerse.

REFERENCES

James Arvo. 1993. Transfer equations in global illumination. *Global Illumination, SIGGRAPH '93 Course Notes* 2 (1993).

Steve Bako, Thijs Vogels, Brian McWilliams, Mark Meyer, Jan Novák, Alex Harvill, Pradeep Sen, Tony DeRose, and Fabrice Rousselle. 2017. Kernel-predicting convolutional networks for denoising Monte Carlo renderings. *ACM Trans. Graph.* 36, 4 (2017), 97:1–97:14.

George EP Box and David R Cox. 1964. An analysis of transformations. *Journal of the Royal Statistical Society: Series B (Methodological)* 26, 2 (1964), 211–243.

Xi Deng, Shaojie Jiao, Benedikt Bitterli, and Wojciech Jarosz. 2019. Photon surfaces for robust, unbiased volumetric density estimation. *ACM Trans. Graph.* 38, 4 (2019), 46:1–46:12.

Robert A Drebin, Loren Carpenter, and Pat Hanrahan. 1988. Volume rendering. *ACM Siggraph Computer Graphics* 22, 4 (1988), 65–74.

Sebastian Herholz, Yangyang Zhao, Oskar Elek, Derek Nowrouzezahrai, Hendrik P. A. Lensch, and Jaroslav Křivánek. 2019. Volume path guiding based on zero-variance random walk theory. *ACM Trans. Graph.* 38, 3 (2019), 25:1–25:19.

Jie Hu, Li Shen, and Gang Sun. 2018. Squeeze-and-excitation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 7132–7141.

Wojciech Jarosz, Derek Nowrouzezahrai, Robert Thomas, Peter-Pike J. Sloan, and Matthias Zwicker. 2011. Progressive photon beams. *ACM Trans. Graph.* 30, 6 (2011), 181:1–181:12.

Wojciech Jarosz, Matthias Zwicker, and Henrik Wann Jensen. 2008. The beam radiance estimate for volumetric photon mapping. In *ACM SIGGRAPH 2008 classes.* 1–112.

Henrik Wann Jensen, Stephen R. Marschner, Marc Levoy, and Pat Hanrahan. 2001. A practical model for subsurface light transport. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2001, Los Angeles, California, USA, August 12-17, 2001.* ACM, 511–518.

James T. Kajiya and Brian Von Herzen. 1984. Ray tracing volume densities. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1984, Minneapolis, Minnesota, USA, July 23-27, 1984.* ACM, 165–174.

Simon Kallweit, Thomas Müller, Brian McWilliams, Markus H. Gross, and Jan Novák. 2017. Deep scattering: rendering atmospheric clouds with radiance-predicting neural networks. *ACM Trans. Graph.* 36, 6 (2017), 231:1–231:11.

Anton Kaplanyan and Carsten Dachsbacher. 2010. Cascaded light propagation volumes for real-time indirect illumination. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games.* 99–107.

David Koerner, Jamie Portsmouth, Filip Sadlo, Thomas Ertl, and Bernd Eberhardt. 2014. Flux-limited diffusion for multiple scattering in participating media. *Computer Graphics Forum* 33, 6 (2014), 178–189.

Jaroslav Křivánek, Iliyan Georgiev, Toshiya Hachisuka, Petr Vévoda, Martin Šik, Derek Nowrouzezahrai, and Wojciech Jarosz. 2014. Unifying points, beams, and paths in volumetric light transport simulation. *ACM Trans. Graph.* 33, 4 (2014), 103:1–103:13.

Eric P Lafortune and Yves D Willems. 1996. Rendering participating media with bidirectional path tracing. In *Eurographics Workshop on Rendering Techniques.* Springer, 91–100.

Ludwig Leonard, Kevin Höhlein, and Rüdiger Westermann. 2021. Learning multiple-scattering solutions for sphere-tracing of volumetric subsurface effects. *Computer Graphics Forum* 40, 2 (2021), 165–178.

Liangchen Luo, Yuanhao Xiong, Yan Liu, and Xu Sun. 2019. Adaptive gradient methods with dynamic bound of learning rate. In *Proceedings of the 7th International Conference on Learning Representations.* New Orleans, Louisiana.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: representing scenes as neural radiance fields for view synthesis. In *Computer Vision - ECCV 2020 - 16th European Conference (Lecture Notes in Computer Science, Vol. 12346).* Springer, 405–421.

Thomas Müller, Fabrice Rousselle, Jan Novák, and Alexander Keller. 2021. Real-time neural radiance caching for path tracing. *ACM Trans. Graph.* 40, 4 (2021), 36:1–36:16.

Mikhail Panin and Sergey I. Nikolenko. 2019. Faster RPNN: rendering clouds with latent space light probes. In *SIGGRAPH Asia 2019 Technical Briefs, SA 2019, Brisbane, QLD, Australia, November 17-20, 2019.* ACM, 21–24.

Mark Pauly, Thomas Kollig, and Alexander Keller. 2000. Metropolis light transport for participating media. In *Eurographics Workshop on Rendering Techniques.* Springer, 11–22.

Jos Stam. 1995. Multiple scattering as a diffusion process. In *Eurographics Workshop on Rendering Techniques.* Springer, 41–50.

Eric Veach. 1998. *Robust Monte Carlo Methods for Light Transport Simulation.* Ph. D. Dissertation. Stanford University, Stanford, CA, USA. Advisor(s) Guibas, Leonidas J. AAI9837162.

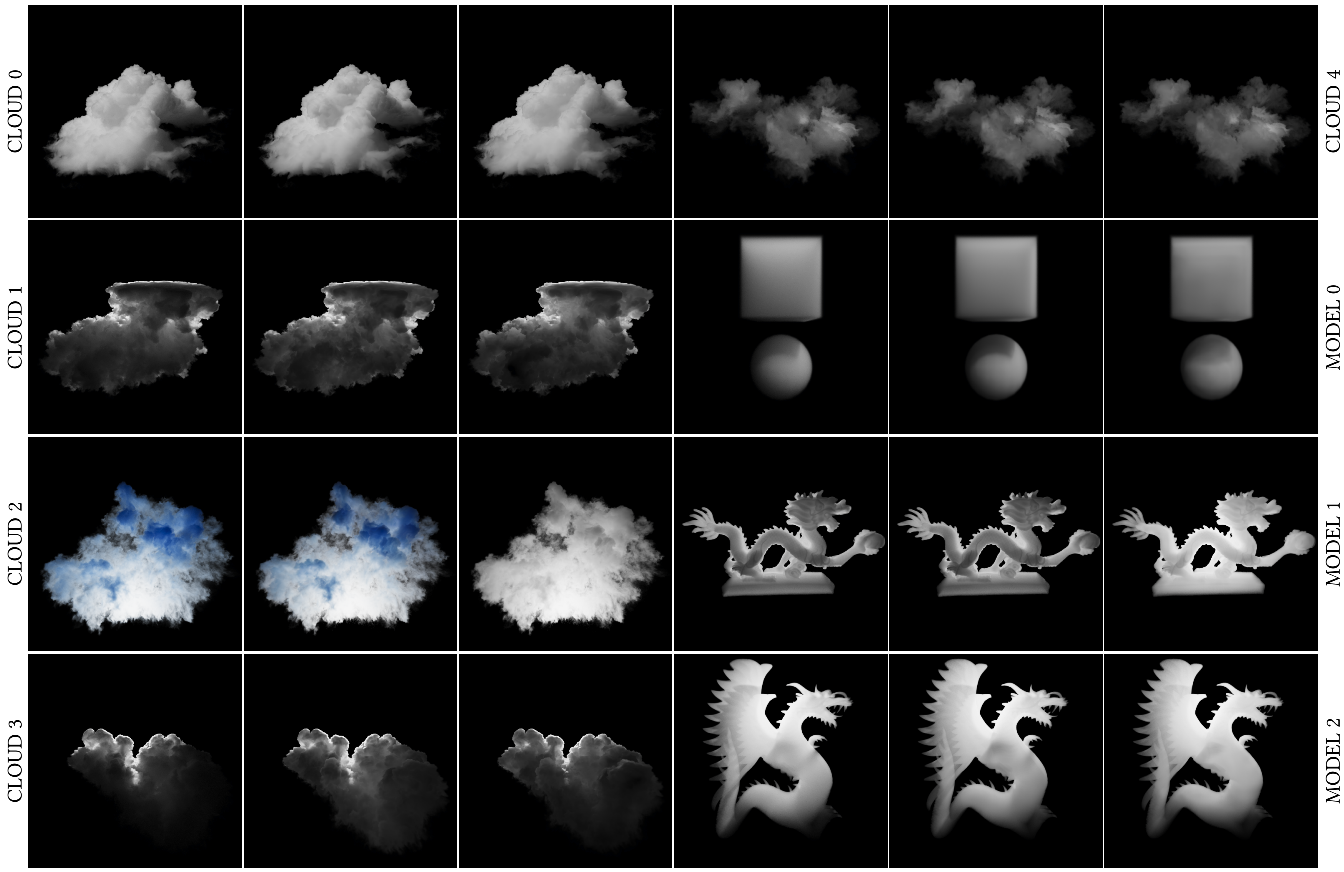
D. Vicini, V. Koltun, and W. Jakob. 2019. A learned shape-adaptive subsurface scattering model. *ACM Trans. Graph.* 38, 4 (2019), 127:1–127:15.

E Woodcock, T Murphy, P Hemmings, and S Longworth. 1965. Techniques used in the GEM code for Monte Carlo neutronics calculations in reactors and other systems of complex geometry. In *Proc. Conf. Applications of Computing Methods to Reactor Problems*, Vol. 557. 557–579.

M. Wrenninge, N. B. Zafar, A. Bouthors, J. Tessendorf, and G. Graham. 2011. Production Volume Rendering: Systems. In *ACM SIGGRAPH (Courses).*

In-Kwon Yeo and Richard A Johnson. 2000. A new family of power transformations to improve normality or symmetry. *Biometrika* 87, 4 (2000), 954–959.

Arno Zinke, Cem Yuksel, Andreas Weber, and John Keyser. 2008. Dual scattering approximation for fast multiple scattering in hair. *ACM Trans. Graph.* 27, 3 (2008), 32:1–32:10.



(a) Reference **(b) Ours** **(c) RPNN** **(d) Reference** **(e) Ours** **(f) RPNN**

Figure 4: Render configurations and comparisons. All scenes are rendered with $\varsigma = RGB(1.0, 1.0, 1.0)$, $G = 0.857$ (except $\varsigma = RGB(0.8, 0.9, 1.0)$ in CLOUD2 and $G = 0.5$ in MODEL1).

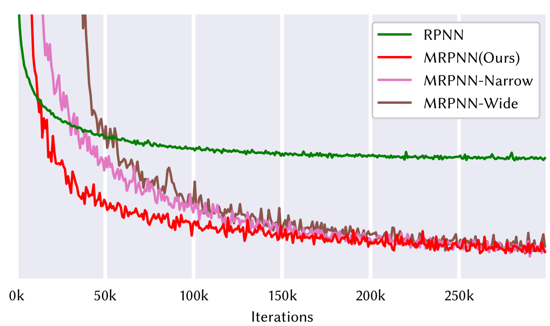


Figure 5: Convergence of validation errors (log-transformed).

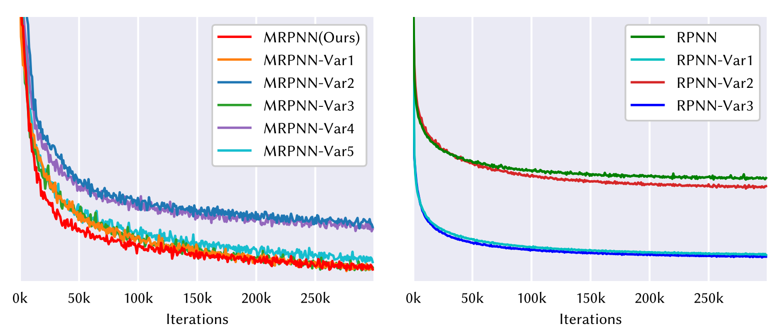
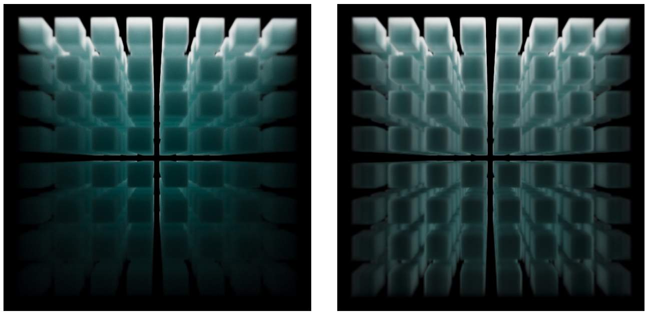


Figure 6: Validation errors of the ablation tests.



(a) Reference **(b) MRPNN**

Figure 7: Because of mipmapping, the network assumes fragmented volumes to be continuous lower-density volumes, resulting in imprecise overall illuminance.

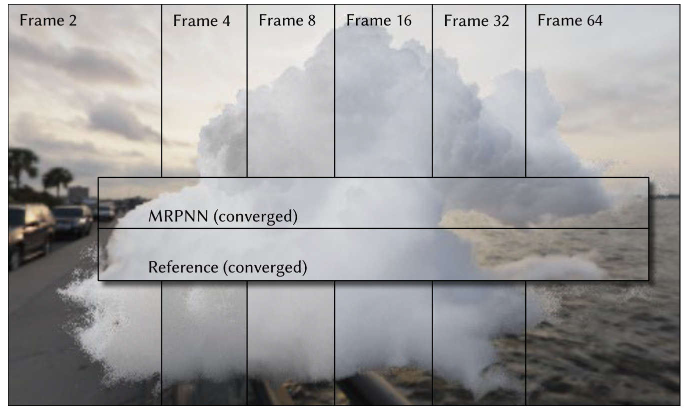


Figure 8: MRPNN produces random overall biases before convergence in ambient lighting, which can't be addressed by denoisers.

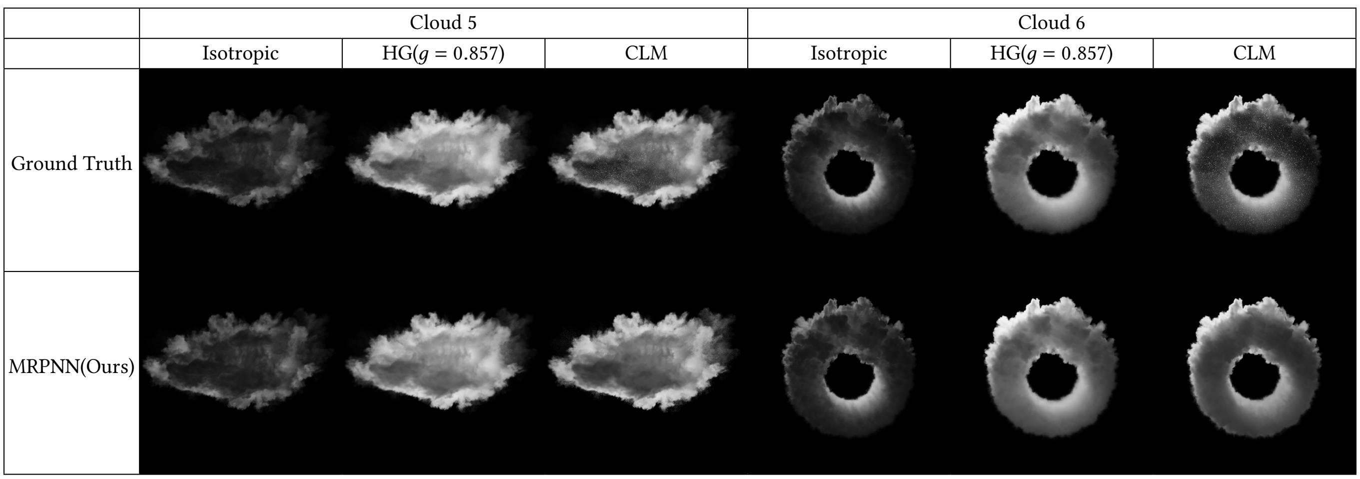


Figure 9: MRPNN renders with configurable phase functions on the fly, avoiding the need to retrain the network. The results show promising compatibility of MRPNN in phase functions. CLM is an abbreviation for Chopped Lorenz-Mie.

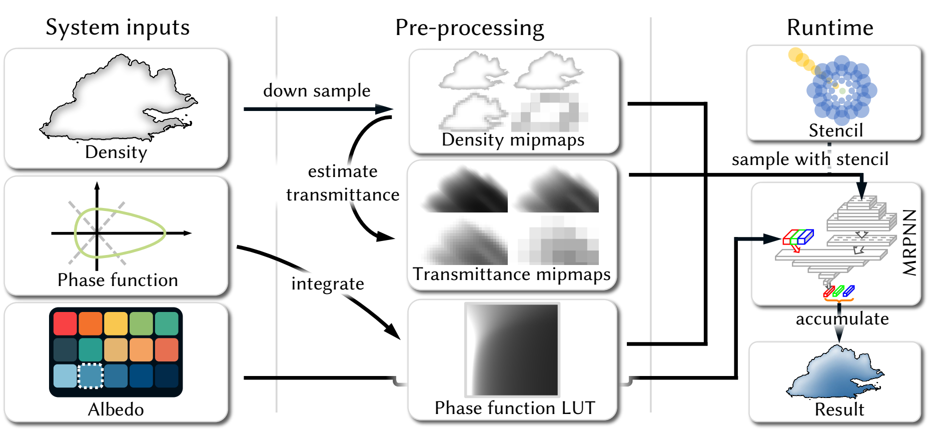


Figure 10: Framework overview. The system takes a density field, phase function, and albedo as inputs. Then, in pre-processing, we recursively down sample the density field and generate the corresponding transmittance fields. The volume-averaged phase function's LUT is also generated in this phase. Finally, in runtime, we sample the density and transmittance fields and the phase with the stencil, and the assembled descriptor is then passed to MRPNN network for estimation.