Efficient Binocular Rendering of Volumetric Density Fields with Coupled Adaptive Cube-Map Ray Marching for Virtual Reality

Tianchen Xu, Xiaohua Ren, Jiale Yang, Bin Sheng, and Enhua Wu *Member, IEEE*

Fig. 1. Our real-time binocular rendering result of multiple translucent volumetric density fields with dynamic global-illumination (GI) effects in hybrid mesh-volume environment (some density fields include fluid simulation).

Abstract—Creating visualizations of multiple volumetric density fields is demanding in virtual reality (VR) applications, which often include divergent volumetric density distributions mixed with geometric models and physics-based simulations. Real-time rendering of such complex environments poses significant challenges for rendering quality and performance. This paper presents a novel scheme for efficient real-time rendering of varying translucent volumetric density fields with global illumination (GI) effects on high-resolution binocular VR displays. Our scheme proposes creative solutions to address three challenges involved in the target problem. Firstly, to tackle the doubled heavy workloads of binocular ray marching, we explore the anti-aliasing principles and more advanced potentials of ray marching on interior cube-map faces, and propose a coupled ray-marching technique that converges to multi-resolution cube maps with interleaved adaptive sampling. Secondly, we devise a fully dynamic ambient GI approximation method that leverages spherical-harmonics (SH) transform information of the phase function to reduce the huge amount of ray sampling required for GI while ensuring fidelity. The method catalyzes spatial ray-marching reuse and adaptive temporal accumulation. Thirdly, we deploy a two-phase ray-tracing algorithm with a tiled k-buffer to achieve fast processing of order-independent transparency (OIT) for multiple volume instances. Consequently, high-quality and high-performance real-time dynamic volume rendering can be achieved under constrained budgets controlled by developers. As our solution supports mixed mesh-volume rendering, the test results prove the practical usefulness of our approach for high-resolution binocular VR rendering on hybrid multi-volumetric and geometric environments.

Index Terms—Binocular views, density field, global illumination, ray marching, real-time rendering, virtual reality, volume rendering.

1 Introduction

Nowadays, volume-data rendering has become a significant part of modern real-time rendering systems for virtual reality (VR). It can convey translucent object perceptions of complex interior density distribution, such as clouds, smoke, and fog. Moreover, it can provide a form of geometry-data representation to present subsurface scattering effects for non-uniform interior densities, such as jade, turbid liquids, and ground glass. Transparency rendering of mesh surfaces can only handle constant or trivial (regular) density distributions. However, volume rendering is much more computationally intensive than mesh rendering, using either ray marching or volumetric path tracing approaches. Volumetric path tracing is commonly used for extremely high-quality requirements like offline rendering, while ray marching is more commonly used in real-time rendering because of its lightweight and cache-friendly process. Despite the possibility of introducing bias, ray marching is also capable of supporting global illumination (GI) by spherical

- *T.C. Xu is with the State Key Lab of CS, Institute of Software, Chinese Academy of Sciences (CAS) & University of CAS, Beijing, China, and Advanced Micro Devices (AMD), Inc., Shanghai, China. E-mail: xutc@ios.ac.cn, tianchen.xu@amd.com*
- *X.H Ren is with Multimedia Research Center, Tencent, Shenzhen, China.*
- *J.L. Yang is with the department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China, and Advanced Micro Devices (AMD), Inc., Shanghai, China.*
- *B. Sheng is with the department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China.*
- *E.H. Wu is with the State Key Lab of CS, Institute of Software, Chinese Academy of Sciences & University of CAS, Beijing, China, and FST, University of Macau, Macao SAR., China. E-mail: weh@ios.ac.cn*

Manuscript received August 28, 2023; revised August 28, 2023.

sampling on voxels. Therefore, this paper focuses on ray marching along the view and light directions. However, it is still a challenge to achieve real-time rendering of multi-volumes in high quality and performance for binocular VR views due to the following reasons.

Firstly, it is challenging to handle the view-direction ray marching in binocular views. Generally, ray marching needs to sample the volume data by many steps for each pixel, thus rapidly increasing the workload of ray marching with the screen resolution and ray sampling counts. The base ray sampling count is usually specified by developers according to the volume resolution and can be further optimized by adaptive sampling. However, emitting too sparse rays would result in breadth-direction aliasing artefacts. Inadequate sampling along the ray would cause observable depth-direction aliasing when moving to another view. On the other hand, by taking advantage of some texture-space imposter techniques [1], [2], we have a chance to decouple the ray marching consumption and screen resolution. Nevertheless, in VR applications, the ray marching workloads still need to be doubled to accommodate the binocular stereo views requirement. There is still potential for improvement to boost the ray-marching performance for binocular views based on the imposter strategy.

Secondly, it is also a challenging task to achieve real-time dynamic ambient lighting of GI. GI for mesh rendering is already computationally intensive, as it requires sampling light sources along all directions of the hemisphere and computing the reflections. In the light-direction ray marching approach, we need to sample both light sources and light phase functions along all directions on the full unit sphere for GI. The calculation of the phase function for each direction also requires ray marching. By exploiting temporally amortized sampling [3], [4], [5], we can distribute the intensive sampling workloads across multiple frames. However, different temporal sampling techniques and implementations can result in significant differences in rendering quality. Poor temporal sampling methods can lead to artefacts such as ghosting or noise due to slow convergence, or flickering due to instability in historical sample filtering. Both types of artefacts occur because of contradictory factors in how the contribution of current samples is weighted. Additionally, denoising on voxels of a volume requires more computational work than denoising on pixels of an image. Therefore, finding a single ideal solution to address all of these challenges is rare.

Thirdly, it is challenging to draw all translucent volume instances in the correct order, either far-to-near or near-to-far. Order-independent transparency (OIT) has been troublesome in computer graphics for some time, whereas it happens to be demanding in multi-volume rendering. For ordinary mesh rendering, stochastic OIT methods have become popular, which also uses sampling and filtering techniques. However, view direction and light-direction ray marching already calls for amortized sampling. We expect an OIT technique that would not further amplify noises. Fortunately, different from the generalized ray tracing, all rays emitted for OIT come from the camera and go along the view directions. We can leverage such a characteristic to accelerate the OIT of multi-volumes in a coarse view.

In this paper, we propose a novel solution for real-time rendering of multiple volumetric density instances with GI effects for binocular VR views. To overcome the challenges arising from this task, we introduce three major technical contributions. We employ the principle of anti-aliasing by ray marching on interior cube-map faces [6], using a ray coupling strategy, tile-granularity process, and adaptive sampling with interleaved ray steps. For the lighting, we develop a fully dynamic ambient GI approximation method that leverages spherical harmonics transforms and reuses coarsely estimated phase functions calculated by neighboring voxels to speed up convergence. To address the multi-volume OIT problem, we design a two-phase ray-tracing technique that uses ray tracing on a fixed number of coarse tile-level pixels and accelerates the fine per-pixel coverage evaluations with the hardware ray intersection function. Our solution achieves high fidelity and high performance real-time multi-volume rendering for high-resolution VR views, with adaptive techniques reducing sampling workloads and ensuring viewport independence.

2 Related Work

Volume rendering has emerged as a vibrant field for volume-data visualization and volumetric light transport simulation since its introduction in the seminal work by Drebin *et al.* that proposes a technique to render images of volumes containing mixtures of materials [7]. Over the years, numerous research works have been presented in this domain. In this paper, we focus on works closely related to ours and direct the reader to [8] for a comprehensive review of volume-data real-time rendering techniques and [9] for recent advancements in volumetric light transport simulation using Monte Carlo integration estimating methods.

Sampling and Integration: The attenuation and scattering of light occur during its propagation from the light sources to the participating media. The changes in radiance can be described using the radiative transfer equation (RTE) [10]. Specifically, the RTE calculates the radiance $L(\boldsymbol{x}, \boldsymbol{\omega})$ along the ray $(\boldsymbol{x}, -\boldsymbol{\omega})$, as described in [9]:

$$L(\boldsymbol{x}, \boldsymbol{\omega}) = \int_0^\infty T(\boldsymbol{x}, \boldsymbol{y}) \mu_s(y) L_s(\boldsymbol{y}, \boldsymbol{\omega}) dy, \tag{1}$$

where $\mu_s(y)$ with $\boldsymbol{y} = \boldsymbol{x} - y\boldsymbol{\omega}$ is the scattering coefficient. The transmittance $T(\boldsymbol{x}, \boldsymbol{y})$ in Equation 2, which accounts for the attenuation of light, is defined as

$$T(\boldsymbol{x}, \boldsymbol{y}) = e^{-\int_0^y \mu_t(\boldsymbol{x} - s\boldsymbol{\omega}) ds}, \tag{2}$$

where μ_t is the extinction coefficient. The in-scattered radiance $L_s(\boldsymbol{y}, \boldsymbol{\omega})$ in Equation 3, which integrates the incident radiance L_i from all directions $\bar{\boldsymbol{\omega}}$ on the unit sphere S^2, is defined as

$$L_s(\boldsymbol{y}, \boldsymbol{\omega}) = \int_{\mathrm{S}^2} f_p(\boldsymbol{\omega}, \bar{\boldsymbol{\omega}}) L_i(\boldsymbol{y}, \bar{\boldsymbol{\omega}}) d\bar{\boldsymbol{\omega}}, \tag{3}$$

where $f_p(\boldsymbol{\omega}, \bar{\boldsymbol{\omega}})$ is the phase function. To evaluate the radiance, we need to compute two line integrals (Equations 1 and 2) and a spherical integral (Equation 3).

Since the two line integrals are coupled, the proper selection of the sampling rate for each integral is essential to avoid aliasing. To address this issue, Engel *et al.* proposed a pre-integrated scheme to decouple the second integral from the first for independent sampling-rate selection [11]. Moreover, Bergner *et al.* introduced a spectral analysis method

to determine the max sampling rates [12], allowing for the design of an adaptive sampling strategy. Once the sampling rate is determined, these methods employ linear interpolation for integration. In contrast, Muõz opted for high-order quadrature rules to solve Equations 1 and 2, such as the two-order nested Simpson quadrature rule, and presented a comprehensive analysis of various rules [13].

Besides those quadrature schemes, Monte Carlo methods are widely used. One such method is an unbiased distance sampling approach proposed by Raab, which places fewer samples in regions where extinction makes them unnoticeable [14]. Kulla and Fajardo proposed a method to put more samples close to light source for decoupling ray marching from light calculations by computing a representation of the transmittance function [15]. Fraboni *et al.* presented a sampling method by exploiting redundancy across views to accelerate the rendering of heterogeneous participating media [16]. Lin *et al.* extended the spatiotemporal reservoir sampling for direct illumination to multi-dimensional volumetric path tracing [17]. Recently, Kettunen *et al.* presented an unbiased estimator to evaluate the transmittance equation [18] which can effectively perform ray marching while using rarely-sampled higher order terms to correct the bias.

Parallel and Real-time Techniques: This section explores the parallel and real-time techniques for volume rendering. Parker *et al.* presented a system that utilizes brute-force ray tracing for interactive volume visualization [19], which can efficiently run on multiprocessor machines. For real-time rendering of diffuse and glossy objects, recomputed radiance transfer based on spherical harmonics was proposed [20], which can also be applied to volume rendering. With the advancement of GPU technology, numerous GPU-based techniques for volume rendering have been developed. Kruger and Westermann addressed the integration of early ray termination and empty-space skipping into texture-based volume rendering on GPUs [21]. Smelyanskiy *et al.* performed a comprehensive analysis of three types of volume rendering methods [22]: ray casting, splatting, and shear-warping, which are suitable for modern parallel processing architectures. Based on their analysis, they concluded that ray-casting is the best method for medical applications and described a thread and data-parallel ray casting algorithm on both GPUs and multi-core CPUs. Crassin *et al.* introduced an efficient GPU-based method for rendering large voxel-grid datasets using an adaptive data representation based on the current view and occlusion information [23]. Recently, Morrical *et al.* presented an approach to render unstructured meshes using a combination of coarse spatial acceleration and hardware-accelerated ray tracing [24]. Moreover, Derin *et al.* devised a sparse volume rendering approach utilizing hardware ray tracing and block walking [25]. Xu *et al.* proposed a viewport-resolution independent ray marching scheme on the interior faces of cube maps that enhances performance on high-resolution displays with a simplified GI effect through major-direction irradiance approximation [6]. However, as a progressive work, the work does not investigate the anti-aliasing principle underlying the proposed method, and their ordinary OIT method for multi-volumes holds significant potential for further performance improvement.

3 Principles and Methods

Our volume rendering solution is based on ray marching in the directions of view and light. The work pipeline can be divided into 3 major phases, consisting of several rendering passes:

Lighting passes We launch a per-voxel lighting process with our ambient GI approximation method (see section 3.2).

View-direction ray marching passes We first perform a ray marching cast on the tile-level texels of a cube map to estimate the level of detail and sampling rates, and cull the tiles with all empty-density voxels along the ray. Then, we dispatch a fine ray marching pass with coupled rays converging to the cube-map pixel for binocular views (see sections 3.1 and 3.3).

Integration passes We draw the ray-marched cube imposters of all visible volume instances on the screen, and process the OIT by tracing the volume instances in near-to-far order, using our tiled k-buffer ray tracing (see sections 3.3 and 3.4).

Here, a cube map consists of six 2D square images to represent faces of a cube. Besides, a tile is a small rectangular region of pixels or texels, which can be regarded as a group of pixel workloads. Then, a tile-level pixel or texel is to hold the coarsely estimated information of a tile, serving as an approximation to the average values of the pixels in a tile. Additionally, a k-buffer is comprised of multi-layer 2D images in same size, storing an array of data for each pixel. Furthermore, a tiled k-buffer is a low-resolution k-buffer, where each texel is a tile-level pixel to represent a tile.

3.1 View-direction Ray Marching

To perform view-direction ray marching that accumulates the integral of lit voxel densities according to Equation 1, we propose a solution consisting of several detailed techniques. These techniques **adaptively fit the sampling rates for both breadth and depth frequencies in binocular views** by taking into account the grid resolution and density contents in the source volume data.

3.1.1 Anti-aliased Interior-face Cube-map Ray Marching

Firstly, we compare three ray-marching schemes of different workspaces: direct screen-space ray marching, ray marching onto the exterior cube-map faces, and ray marching cast to the interior cube-map faces. The concept of cube-map space ray marching was proposed by Xu *et al.* [6]. Their work suggests that the interior scheme is faster and reduces texel-aliasing but without explaining its principle. In our work, we provide both a qualitative analysis and a quantitative proof for the texel anti-aliasing property of the interior cube-map ray marching scheme.

As shown in Figure 2, we present a comparison between two ray marching schemes for volume rendering. The direct screen-space ray marching approach (a) has a uniform sampling rate, which can be wasteful when the screen resolution exceeds the volume resolution. On the other hand, the cube-map space ray marching approach aims to fit the volume resolution but suffers from a poor fit when using the straightforward exterior cube-map face scheme

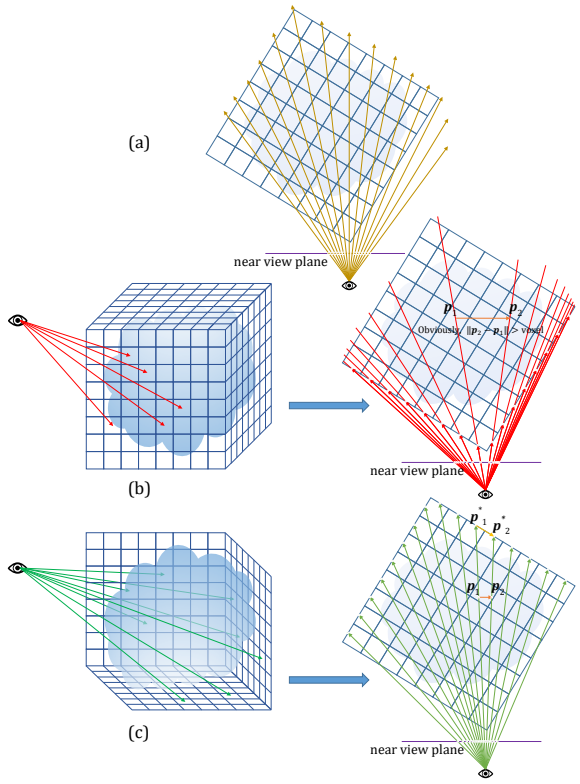


Fig. 2. (a) Direct screen-space ray marching, (b) Ray cast to the pixel center of the cube-map exterior faces - (left) 3D view, (right) cross-section view, and (c) ray cast to the pixel center of the cube-map interior faces - (left) 3D view, (right) cross-section view.

(b). To address this issue, we assume that the resolution per dimension of a cube-map should be equal to the resolution per dimension of the source volume grid. Contrary to the exterior-face scheme, which casts many rays (red arrow lines) onto the useless border regions, the interior-face scheme (c) ensures that the rays (green arrows) cast onto the pixel centers of the interior faces (blue-grid border at the back) are well-distributed. The max gap between two adjacent rays on the far cube-face plane is equal to the voxel size (denoted by s_{voxel}). As $\frac{1}{s_{\text{voxel}}}$ is the double max-frequency of the original volumetric density data, which satisfies the Nyquist sampling theorem, it has no chance to result in any case of texel-aliasing due to the frequency loss along the breadth directions on the view plane. We mathematically prove the anti-aliasing property of ray marching on interior cube-map faces in appendix A.

3.1.2 Ray Coupling Based on Rays-Convergence Property

In this section, we describe our strategy of spatial samples reuse, which involves ray coupling based on the rays-convergence property observed in the cube-map ray marching scheme. This property allows the rays from two camera points to be cast convergently to the same point on the cube-map interior face, as illustrated in Figure 3 (a). The rays-convergence property can be related to binocular stereo views, where the positions of an object in the view spaces of the two cameras are different but the positions in the world space are the same. By using a cube as the imposter of a volume instance for the binocular views, we can always find a couple of rays from the two cameras that focus on the same pixel center on the cube interior face in the world

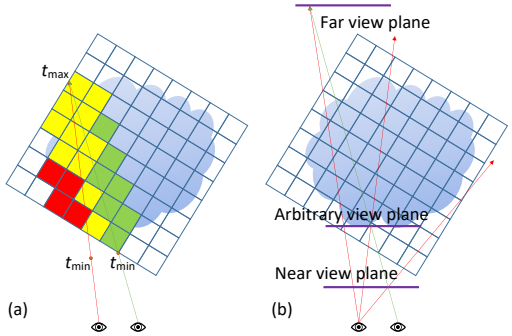


Fig. 3. (a) The couple of rays from the binocular cameras converge to the interior face of the cube map, and the red, green, and yellow cells represent the sample-impact voxels, and the yellow cells have shared samples; (b) in direct screen-space ray marching, the couple of rays can hardly converge to an optimal view plane.

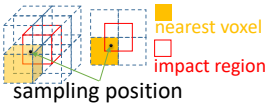
space. As a result, some ray-marching samples on the couple of rays only affect the same shared voxels (yellow cells in (a)). Thus, we can **spatially reuse** the sample data on one ray to the other for such samples.

In contrast, reusing ray-marching samples in the direct screen space ray marching scheme is challenging because of the difficulty to identify a uniform view plane for all rays. A couple of rays intersecting on this plane have a minimum distance, but such a plane is hard to find. As illustrated in Figure 3 (b), the couple of rays diverge after intersecting on the near view plane. Even using the far view plane for convergence is less optimal than the point on the cube because the convergence point can be far. Accurately determining the optimal view plane for all rays would require more computations, which could be even costlier than ray marching twice.

To achieve spatial sample reuse with ray coupling, we also need to couple the samples on the coupled binocular rays with minimized distance and keep the marching steps synchronized. Firstly, we evaluate the max ray lengths from the two cameras to the same pixel of the interior cube face, denoted by $t_{\max}$ values. Then, we measure the distances from the cameras to the volume entries respectively, as the min ray length values $t_{\min}$. Then, we take the max value of $(t_{\max} - t_{\min})$ as the joint ray length value t_{d} of the coupled rays and reevaluate the start ray length $t_{\min} = t_{\max} - t_{\text{d}}$. Thus, we perform ray marching starting from $t_{\min}$ and ending at $t_{\max}$ with synchronous steps for the coupled rays. When the ray marching jobs for the coupled rays are arranged into two neighboring threads, we can utilize the hardware cache to implicitly achieve the most data reuse. However, the thread for the shorter ray may sample the voxels outside the volume in the initial iterations. In such cases, we skip the sampling operations but keep the step.

In addition, we also implement a version to explicitly execute the coupled sampling by two phases of sampling loops in the same shader, which provides almost same performance to the implicit implementation. In the first phase, we perform ray marching along the coupled rays separately. After the samples on the coupled rays have overlaps in $2 \times 2 \times 2$ sampling range, we only execute the voxel sampling for the left ray, and share the raw samples to the right ray thread through wave-level thread shuffle. Mathematically, the phase-switch condition can be expressed

as $\lfloor (\boldsymbol{x}_0^{\text{Left}} + t\boldsymbol{v}^{\text{Left}})S + 0.5 \rfloor = \lfloor (\boldsymbol{x}_0^{\text{Right}} + t\boldsymbol{v}^{\text{Right}})S + 0.5 \rfloor$, where $\boldsymbol{x}_0^{\text{Left}}$ and $\boldsymbol{x}_0^{\text{Right}}$ are the ray origins in the texture space for the left and right cameras, $\boldsymbol{v}^{\text{Left}}$ and $\boldsymbol{v}^{\text{Right}}$ denote the ray directions, t is the ray-length parameter of the current samples, and S is the volume grid resolution per dimension. When two sampling positions are located among the same $2 \times 2 \times 2$ voxels, the couple of rays will share the same voxel data (as shown in the wrap figure). Specifically, we incorporate a 0.5 voxel offset to implement a round operator.



3.1.3 Adaptive Sampling-Rate Ray Marching

To further enhance performance without introducing any aliasing artefacts, we propose an adaptive ray-marching sampling method that employs **variable step amplification** (VSA) with **interleaved spatial reuse** and **tile-granularity variable levels of detail** (LODs) for both the depth and breadth directions.

To adjust the sampling step sizes dynamically during ray-marching in the *depth direction*, we consider 3 factors for adaptively amplifying the step: the **wave-length factor**, the **transparency factor**, and the **attenuation factor**. In addition, we take a user-specified step size as a baseline. Then, we formulate our variable step amplification model as follows:

$$\Delta t(t) = \frac{t_{\max} - t_{\min}}{N} \max\left(\frac{k_{\text{a}} t}{t_{\max}} f_\lambda^{w_\lambda}(t) f_{\text{T}}^{w_{\text{T}}}(t) f_{\text{A}}^{w_{\text{A}}}(t), 1 \right) \tag{4}$$

where N and t represent the baseline sampling count for the current tile and the current ray length, respectively. k_{a} is a constant adjustment coefficient, and f_λ, f_{T}, and f_{A} are the wave-length, transparency, and attenuation factors, respectively. Further, w_λ, w_{T}, and w_{A} denote their corresponding exponential weights, which can be pre-determined through parameter tuning to approximate the ground truth by ray marching with very small steps. In our implementation, we use $k_{\text{a}} = 3$.

The major adjustment factor, the wave-length factor f_{F}, is determined by the reciprocal frequency of the density field along the ray, which can be estimated using the derivative of the density:

$$f_\lambda(t) = \min\left(\frac{k_{\text{b}}}{|\rho'(t)|}, f_\lambda^{\max} \right) \tag{5}$$

where k_{b} denotes a constant adjustment coefficient for this factor, $f_\lambda^{\max}$ represents the upper bound value, and ρ denotes the sampled densities. We use Bergner *et al.*'s calculation to estimate derivative ρ' [12]. In our implementation, we use $f_\lambda^{\max} = 2$.

Additionally, we consider the transparency and attenuation factors to control the sampling steps. We skip less important samples with low density and high transparency. We also introduce the accumulated opacity as an attenuation factor. This factor amplifies the step when sufficient high opacity has accumulated, but reduces the step to prevent sampling-rate deficiency when high-density samples appear behind low-density samples. The transparency and attenuation factors are modeled using the inverted opacity and inverted transmittance, respectively.

$$\begin{cases} f_{\text{T}}(t) = 1 - a\tilde{\rho}(t + \Delta t) \\ f_{\text{A}}(t) = 1 - T(t) \end{cases} \tag{6}$$

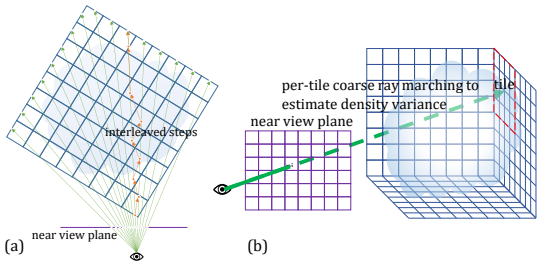


Fig. 4. (a) Depth-direction sampling with interleaved steps (b) In a coarse-phase process, we evaluate the variable MIP level for each tile.

where a represents the absorption coefficient, and $T(t)$ is the view-direction transmittance obtained by ray-marching until the current ray length t. It is to be noted that the next density value $\tilde{\rho}(t + \Delta t)$ is not available, and therefore we need to predict it using $\rho(t) + \rho'(t)$, where ρ' is estimated in a way similar to the wave-length factor calculation.

To further allow for step amplification while collecting adequate samples, we propose an **interleaved spatial reuse** strategy. This strategy leverages the characteristic that the breadth distance between the neighbor rays in interior cube-map faces can be smaller than the half voxel size for the near voxels. Thus, we group 2×2 neighbor rays in a quad and add interleaved offsets to retrieve 3 supplementary samples from the other rays for a ray step, as shown in Figure 4 (a). On GPU shaders, we use wave-level thread shuffle operations to share the raw samples among the threads in the quad, and recompute the trilinear interpolation of the supplementary samples for the current ray. Once the breadth distance between the neighbor rays get larger than the half voxel size, we no longer deploy interleaved steps to preserve quality for this quad, as the sampling step adjusted by Equation 4 is already large enough.

Regarding the ray-coupling mechanism in Section 3.1.2, which can be damaged by variable sampling steps, we synchronize the ray step sizes to ensure consistency in the current ray length between coupled rays. We take the max value as the optimal synchronization method.

In order to further boost performance by decreasing the LODs of the instances distant from the camera, we can adjust the shading rate of *breadth directions* using tile-granularity LODs. The shading rates cannot exceed the highest resolution level of the cube map, and we have proven aliasing avoidance in Section 3.1.1. Inspired by GPU variable-rate shading (VRS) technique [26], we divide the cube map into tiles that are 32×32 pixels per tile. For each tile, we assign a multi-image pyramid (MIP) level as the LODs value for the input volumetric density texture, stored into the tile-level pixels of a low-resolution cube map (as shown in Figure 4 (b)). The LODs value l can be 0, 1, and 2 to denote 1×1, 2×2, and 4×4 fine pixels per coarse texel, accordingly. Hence, a 32×32-pixel tile has at least 8×8 coarse texels, which allows occupancy of a full wave on wave-64 GPU architectures and occupancy of 2 full waves on wave-32 GPU architectures. To evaluate l, we calculate the partial derivatives of the viewport pixels with respect to the cube map tiles and take the ratio, $l = \lceil \log_2 \max\{ \frac{\partial \text{tiles}}{\partial \text{pixels}} \} \rceil$ [27]. To combine tiles of pixels at different MIP levels during the integration passes, we modify the 2×2 bilateral filter of [6]. We must load raw samples for the bilateral filter from

different MIP levels.

3.2 Ambient Global Illumination Approximation

We use per-voxel lighting for our volume rendering. This involves computing the in-scattering radiance for each voxel using Equation 3. The phase function, which can be anisotropic, is modeled using light transmittance - the attenuation rate from the light source into the voxel. It is anisotropic for incident directions but isotropic for outgoing directions (similar to the Lambert diffuse model) and commonly used in real-time volume rendering. To calculate the integral on the unit sphere in Equation 3, we can implement the phase-function calculation by ray marching towards the light with adaptive step sizes, as in Section 3.1.3. To decouple the light sources and the phase function, we deploy spherical harmonics (SH) transforms for both the light probe and the phase function, following the approach [19]:

$$\int_{\mathrm{S}^2} f_{\mathrm{p}} L_i d\bar{\boldsymbol{\omega}} \approx \sum_{j}^{n^2} \int_{\mathrm{S}^2} f_{\mathrm{p}} y_j(\bar{\boldsymbol{\omega}}) d\bar{\boldsymbol{\omega}} \cdot \int_{\mathrm{S}^2} L_i y_j(\bar{\boldsymbol{\omega}}) d\bar{\boldsymbol{\omega}} \tag{7}$$

The integral in Equation 3 can be expressed using the SH transform, as shown in Equation 7, where y_j represents the SH basis function of index j. Thus, we only need to focus on the SH coefficients of the phase function. Instead of using pre-computed radiance transfer (PRT) [20], [28] or zonal harmonics [29], [30] for the phase-function integral, we use a fully dynamic approximation achieved through temporal accumulation with spatial reuse of implicit information in the SH coefficients. Our per-voxel ambient GI has 3 render passes in its work pipeline:

Sample-generation pass We randomly generate a sampling direction for each voxel using the uniform spherical distribution and estimate the coarse phase function using a large ray-marching step (quadruple step in our tests).

SH-transform pass We spatially reuse the sampling directions of neighboring voxels and select the optimal direction using our simplified reservoir resampling to perform fine phase-function evaluation, transform the phase function into SH coefficients, and perform adaptive temporal accumulation.

Lighting-integration pass We integrate the SH lighting for ambient GI and perform single-direction ray marching for the direct directional light sources.

3.2.1 Temporal Accumulation with SH Coefficients

Our dynamically adaptive temporal accumulation involves weighted blending of the current SH coefficient contributions with the historically accumulated SH coefficients in an amortized way, as expressed by Equation 8.

$$\begin{cases} \mathrm{SH}(n) \approx (1-\lambda(n))\,\mathrm{SH}_{\mathrm{dst}}(n-1) \\ \qquad +\lambda(n) w_p(\bar{\boldsymbol{\omega}}) f_{\mathrm{p}}(\bar{\boldsymbol{\omega}}, n) y_j(\bar{\boldsymbol{\omega}}) \\ \mathrm{SH}_{\mathrm{dst}}(n-1) = \dfrac{w_{\mathrm{SH}}(n)}{w_{\mathrm{SH}}(n-1)} \mathrm{SH}(n-1) \\ w_{\mathrm{SH}}(n) = (1-\lambda(n))\, w_{\mathrm{SH}}(n-1) + \dfrac{\lambda(n)}{p} \end{cases} \tag{8}$$

Define the SH coefficient for short:

$$\mathrm{SH}(n) = \int_{\mathrm{S}^2} f_{\mathrm{p}}(\bar{\boldsymbol{\omega}}, n) y_j(\bar{\boldsymbol{\omega}}) d\bar{\boldsymbol{\omega}}$$

where $\mathrm{SH}(n)$ represents the SH coefficient at the current frame index n, w_{SH} denotes the weight for SH coefficient normalization, w_p denotes the probability weight for the phase function along a spherical direction $\bar{\boldsymbol{\omega}}$, $p = \frac{1}{4\pi}$ is the probability density function of uniform spherical-surface sampling, and λ refers to the weight of the current SH coefficient contribution blended with the historical SH coefficient. The initial evaluation model for λ is $\lambda(n) = \frac{1}{\min(n+1, N)}$, where N is the frame count regarded as convergence.

Because the historical SH coefficients can be staled with regard to the animated volumes, we add a calibration to the historical SH coefficient for the current new frame. When querying neighboring voxels ($3 \times 3 \times 3$ in our test) to reuse their sampling directions and coarsely estimated phase functions, we can also obtain coarse SH coefficients. To estimate the coarse SH coefficients, we compute $\tilde{\mu}\left(\frac{\tilde{f}_{\mathrm{p}}(n) y}{p}\right)$ and the variances of the $3\times3\times3$ SH coefficient contributions $\tilde{\sigma}^2\left(\frac{\tilde{f}_{\mathrm{p}}(n) y}{p}\right)$, which are used to clamp the historical SH coefficient for calibration (inspired by temporal anti-aliasing [3], [4], [5]):

$$\begin{cases} \mathrm{SH}_{\mathrm{clamped}}(n-1) = \mathrm{clamp}\left(\mathrm{SH}_{\mathrm{dst}}(n-1), \mathrm{SH}_{\min}, \mathrm{SH}_{\max}\right) \\ \mathrm{SH}_{\min} = \tilde{\mu} - \gamma(n)\tilde{\sigma} \\ \mathrm{SH}_{\max} = \tilde{\mu} + \gamma(n)\tilde{\sigma} \\ \gamma(n) = \dfrac{k_\gamma}{\left|\tilde{\mu}\left(\frac{\tilde{f}_{\mathrm{p}}(n) y}{p}\right) - \tilde{\mu}\left(\frac{\tilde{f}_{\mathrm{p}}(n-1) y}{p}\right)\right|} \end{cases} \tag{9}$$

where γ is the adaptive amplification function for variance clamp. The expression for γ is affected by the difference between the current and corresponding historical coarse SH coefficients, and k_γ is the adjustment parameter, which we set to 0.5 in our tests.

Further, we use $\mathrm{SH}_{\mathrm{dst}}$, $\mathrm{SH}_{\mathrm{clamped}}$, and cubic polynomial fitting with $\tilde{\mu}$, and linearly combine them to model an advanced calibrated historical SH coefficient:

$$\begin{cases} \mathrm{SH}^* = w_{\mathrm{dst}}\mathrm{SH}_{\mathrm{dst}} + w_{\mathrm{clamped}}\mathrm{SH}_{\mathrm{clamped}} \\ \qquad +w_{\mathrm{poly}}(\tilde{\mu}^3 + a\tilde{\mu}^2 + b\tilde{\mu} + c) \\ w_{\mathrm{dst}} + w_{\mathrm{clamped}} + w_{\mathrm{poly}} = 1 \end{cases} \tag{10}$$

We set $w_{\mathrm{dst}} = 0.125$, $w_{\mathrm{clamped}} = 0.875$, and $w_{\mathrm{poly}} = 0$ as the intial values of the weights. We further determine the weights in Equation 10 as well as a, b, and c via a simple training in advance by minimizing the L2 difference between SH^* values and the ground-truth values obtained from 400k rays per voxel in 10k frames captured across our all test case assets.

Then, we can replace $\mathrm{SH}_{\mathrm{dst}}$ with SH^* in Equation 8 and adjust $\lambda(n)$ to an advanced version. This is achieved by utilizing the difference between the current coarse SH coefficients and the corresponding historical ones:

$$\lambda(n) = \frac{1 + k_\lambda \left|\tilde{\mu}\left(\frac{\tilde{f}_{\mathrm{p}}(n) y}{p}\right) - \tilde{\mu}\left(\frac{\tilde{f}_{\mathrm{p}}(n-1) y}{p}\right)\right|}{\min(n+1, N)} \tag{11}$$

where we use $k_\lambda = 0.5$ in our test for adjustment.

Notably, as the calibration process is inexpensive and keeps the value unchanged if the value is in the toleration range, we apply the calibration every frame.

3.2.2 Simplified Reservoir Resampling with Spatial Reuse

Regarding the quality of convergence in temporal accumulation, if the frame rate is high, the SH temporal accumulation

will converge quickly, such that users cannot detect its appearance. However, if a VR application has a low frame rate due to intensive workloads, the convergence procedure may create some noise artefacts that can be noticeable to humans. Therefore, it is advisable to further shorten the frame count for convergence in our temporal accumulation by selecting more significant phase-function samples. As mentioned before, we already have a rough estimation of the phase functions from the neighboring voxels. Hence, we perform a simplified reservoir resampling [31], [32], [33] by reusing the neighboring sampling directions and coarse phase functions. The probability weight w_{p} in Equation 8 can be evaluated as follows:

$$w_{\mathrm{p}} = \frac{\sum_{\boldsymbol{x}}^{\boldsymbol{x}\in\Omega} \frac{\tilde{f}_{\mathrm{p}}(\boldsymbol{x})}{p}}{M\tilde{f}_{\mathrm{p}}^{*}} \tag{12}$$

where $\boldsymbol{x}$ denotes the voxel location and Ω represents the neighboring voxel domain ($3\times3\times3$ in our test). M refers to the total voxel count in domain Ω, $\tilde{T}(\boldsymbol{x})$ denotes the coarse phase functions at voxel $\boldsymbol{x}$, and $\tilde{f}_{\mathrm{p}}^{*}$ is the coarse phase-function value stored in the reservoir.

We simplify the procedure by utilizing the coarse phase functions from neighbor voxels as a sample distribution function. This eliminates the need for re-estimation of the phase function for the current voxel. Additionally, we eliminate the temporal reuse step of the spatial-temporal reservoir resampling (ReSTIR [17], [34], [35], [36]) because it may exacerbate the errors in practice.

3.2.3 Lighting Update for Multi-volumes

Updating lighting with ambient GI for each instance of multiple volumes every frame can be computationally expensive, and thus, an amortized strategy is necessary. By default, we update the lighting state of one visible volume per frame. As mentioned, for each voxel of each volume, we select a random direction to compute the phase function using ray marching. Once ray marching is complete for the current volume, we must locate the next volume along the ray. We use GPU ray tracing to hit other volume instances, and since the product operations for transmittance calculation are naturally order-independent as per Equation 2, we can use the "any-hit" mode of the GPU hardware ray tracing. This mode can complete all volume hits with a single traversal. The algorithm is illustrated below:

procedure UPDATEVOXELLIGHTING(v) ▷ each thread for a voxel v
  $(s, E) \leftarrow (1, 0)$ ▷ init shadow and irradiance
  $(N, F) \leftarrow$ (VisibleVolumeCount, FrameIndex)
  $i \leftarrow$ VisibleVolumes[rand(F) $\bmod N$] ▷ get Id
  $r \leftarrow$ CalcRayDesc(v, i)
  while TraceRay(r).CandidateHit not miss **do**
    $j \leftarrow$ GetHitVolume(j)
    $s \leftarrow$ LightRayMarch(r, j)
    $E \leftarrow$ Equation 7 and integral
  end while
end procedure

Our default approach for temporal accumulation is to compute one direction per voxel per frame. In VR applications, vertical synchronization (V-sync) is typically enabled for stable frames per second (FPS). Devices with surplus budgets can enable multiple sampling rays or lighting updates per frame, which reduces the frame count for faster temporal accumulation convergence.

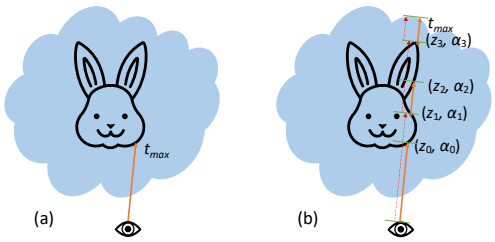


Fig. 5. (a) The max ray length t_{max} clamped by the opaque mesh depth, and (b) ray marching with the mesh k-buffer involved for mixing with the translucent mesh, where the red dash arrows visualize the range of the transmittance segments.

3.3 Mixed Mesh-Volume Rendering

Our solution can handle the mixed rendering of both meshes and volumes. For opaque meshes, we can easily use the depth buffer to handle the mesh occlusion to the volume by clamping the max ray length (t_{max}) of ray marching, as shown in Figure 5 (a). However, with cube-map ray marching, the occlusion stored in the cube map cannot be directly matched to screen-space pixels in the depth buffer, causing mesh-occlusion artefacts during cube imposter drawing. Xu *et al.* provided a solution of deploying a small 2×2 bilateral filter that compares depth values stored in the cube map and the screen-space depth buffer [6]:

$$\begin{aligned} \boldsymbol{s}(i) &= \frac{\boldsymbol{s}_{\mathrm{raw}}(i) w_{\mathrm{z}}(z_{\mathrm{cube}}(i), z_{\mathrm{buffer}}) w_{\mathrm{bilinear}}(i)}{\sum_{j=1}^{4} w_{\mathrm{z}}(z_{\mathrm{cube}}(j), z_{\mathrm{buffer}}) w_{\mathrm{bilinear}}(j)} \\ w_{\mathrm{z}}(z_{\mathrm{cube}}, z_{\mathrm{buffer}}) &= \max\left(1 - 0.5|z_{\mathrm{buffer}} - z_{\mathrm{cube}}|, 0\right) \end{aligned} \tag{13}$$

where $\boldsymbol{s}_{\mathrm{raw}}$ and w_{bilinear} represent the raw sample i gathered from the cube map and its bilinear weight, respectively. w_{z} is the edge-stopping function of the bilateral filter, whose z_{buffer} and z_{cube} represent linear depth values derived from the depth buffer and cube map, respectively. The depth value stored in the cube map is transferred from the depth buffer during the view-direction ray marching pass.

For translucent meshes, it is more challenging to handle the local order-independent transparency (OIT) problem with the volume. To resolve this, we extend the mesh depth buffer and depth cube maps into k-buffers [37]. We require the translucent objects to be enclosed as a prerequisite, and this condition is generally satisfied in most cases. We separate their color blending as the mesh and volume have different destinations (cube-map space and screen space, respectively). We insert the k-buffer depths of the mesh and cube k-buffer depths of the volume into each other in view-direction ray marching and integration phases respectively. We then sum the volume and mesh colors in the integration phase, $c = c^{\mathrm{V}} + c^{\mathrm{M}}$.

On the mesh side, we first encode the k-foremost depth and opacity values (z_i, α_i) into a depth k-buffer in near-to-far order. During ray marching, we insert the mesh opacity into the steps to calculate the transmittance impacted by the mesh in the cube-map space. Then, we store the segmented transmittance in a cube k-buffer, as shown in Figure 5 (b), using the algorithm below.

$(c^{\mathrm{V}}, T^{\mathrm{V}}) \leftarrow (0, 1)$ ▷ volume color and transmittance

$\hat{T}[K/2] \leftarrow 1$ ▷ init segmented transmittance
$(z_i, \alpha_i)[K] \leftarrow$ LoadMeshDepthKBuffer(all layers)
$t \leftarrow 0, \Delta t \leftarrow$ InitStep ▷ init current ray length
$i \leftarrow 0$ ▷ init current insertion layer
for $n \leftarrow 0$ to N **do**
  $(c, \rho) \leftarrow$ GetSample(Ray.Origin $+t$ * Ray.Direction)
  $c^{\mathrm{V}} \leftarrow$ BlendColor$(c^{\mathrm{V}}, T^{\mathrm{V}}, c, \rho)$
  $\sigma \leftarrow 1 - a\rho\Delta t$ ▷ a: absorption coeff.
  $(T^{\mathrm{V}}, \hat{T}_i) \leftarrow (\sigma T^{\mathrm{V}}, \sigma \hat{T}_i)$
  if $i < K/2$ **then**
    if $t + \Delta t >$ DepthToRayLength(z_{i*2}) **then**
      $T^{\mathrm{V}} \leftarrow (1 - \alpha_{i*2})(1 - \alpha_{i*2+1})T^{\mathrm{V}}$
      $t \leftarrow$ DepthToRayLength(z_{i*2+1})
      $i \leftarrow i + 1$
    end if
  end if
  $\Delta(t) \leftarrow$ Equation 4 with couple sync
  $t \leftarrow t + \Delta(t)$
end for

During the integration phase, when drawing the cube imposters, we resolve the mesh color blending, volume color, and total transmittance together in screen space for overlapping pixels between the mesh and volume. We load the mesh color from the k-buffer and blend it from near to far, inserting the volume-segmented transmittance $\hat{T}_{ij}$ with the following algorithm:

$(c, \alpha) \leftarrow 0$ ▷ init blended color and alpha
$(c^{\mathrm{M}}, T^{\mathrm{M}})[4] \leftarrow (0.xxx, 1)$ ▷ init mesh color samples
for $i \leftarrow 0$ to $K/2$ **do**
  $(c_0, \alpha_0) \leftarrow$ LoadMeshColorKBuffer(i * 2);
  $(c_1, \alpha_1) \leftarrow$ LoadMeshColorKBuffer(i * 2 + 1);
  $\hat{T}_{ij} \leftarrow$ GatherTransmSegCubeKBuffer(i);
  for all $j \in 2 \times 2$ texel quad **do**
    $(c_j^{\mathrm{M}}, T_j^{\mathrm{M}}) \leftarrow \hat{T}_{ij} T_j^{\mathrm{M}}(c_0\alpha_0, 1 - \alpha_0) + (c_j^{\mathrm{M}}, T_j^{\mathrm{M}})$
    $(c_j^{\mathrm{M}}, T_j^{\mathrm{M}}) \leftarrow T_j^{\mathrm{M}}(c_1 * \alpha_1, 1 - \alpha_1) + (c_j^{\mathrm{M}}, T_j^{\mathrm{M}})$
  end for
end for
$(c^{\mathrm{V}}, T^{\mathrm{V}}, z^{\mathrm{V}})[4] \leftarrow$ GatherVolumeCubeMap()
$z_{\text{buffer}} \leftarrow$ LoadMeshDepthKBuffer(0)
$T \leftarrow 1, w_{\text{sum}} \leftarrow 0$ ▷ init transmittance
for all $i \in 2 \times 2$ texel quad **do**
  $w = w_z(z_i^{\mathrm{V}}, z_{\text{buffer}}) * w_{\text{bilinear}}$ ▷ Equation 13
  $(c, T) \leftarrow (c_i^{\mathrm{V}} + c_i^{\mathrm{M}}, T_i^{\mathrm{V}} T_i^{\mathrm{M}})w + (c, T)$
  $w_{\text{sum}} \leftarrow w_{\text{sum}} + w$
end for
$(c, \alpha) \leftarrow (c/w_{\text{sum}}, 1 - T/w_{\text{sum}})$

Finally, we draw a full-screen quad for the remaining regions of the mesh that are not overlapped with the volume imposter, and then resolve the mesh OIT from the k-buffer in the ordinary way.

3.4 Global Ray-Traced OIT with Tiled K-buffer

Ray tracing is indeed a feasible solution to tackle order-independent transparency (OIT) problems by triggering ray closest hits to the objects for several times in near-to-far order. However, for a volume-instance OIT problem, per-pixel ray tracing is overkill, as we only expect to get the next volume instance occluded by the front instance. Therefore, we devise a coarse ray tracing method to reduce the rays for the OIT problem in multiple volume rendering. Inspired by ray-traced multi-layer shadow map [38], our basic idea is a two-phase ray tracing, which divides the viewport into a fixed number of tiles. Initially, the algorithm coarsely traces the volume instances in a front-to-back manner for each tile-level pixels. It then stores the hit volume instances in a tiled k-buffer in the same order. Finally, per-pixel hit testing is performed on each targeted volume instance to obtain the precise hit result for alpha blending.

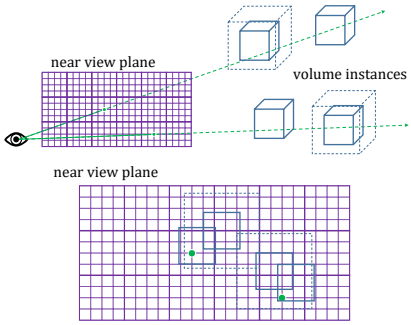


Fig. 6. Tile-granularity ray tracing for acceleration of OIT.

For the per-tile ray-tracing phase, as shown in Figure 6, we use 1080p (1920×1080) viewport as the baseline and make 8×8 pixels as a tile. Thus, the viewport can be divided into 240×135 tiles, as the layout. For higher-resolution viewports, such as 2K and 4K, we do not increase the number of tiles, just scaling up the tile size and keeping the 240×135 tile layout. First, we represent each volume instance as a bottom-level acceleration structure (BLAS) of cube, and build a top-level acceleration structure (TLAS). During the ray-tracing process, rays are traced for each tile from near to far, repeating the process K times. Front-face culling is enabled to ensure that only the interior faces of the BLAS instances are intersected. For each ray's closest hit, we store the hit volume instance ID into a k-buffer slot. Besides, as the rays are emitted from the center of each tile, it is necessary to adjust the size of each BLAS instance during the per-frame TLAS updating process. This adjustment ensures that a BLAS instance, which partially covers a tile, can fully cover the tile center. To implement the adjustment, we conservatively scale up the cube vertices of each BLAS instance in the perspective projection space to cover tile center with floor and ceil operations, and take the max vertex magnification. Using this magnification value, we multiply the world-view compound matrix by a xy-axes scaling transform at the volume center, and then multiply the result by the inverse view matrix to obtain the conservative world transform matrix of the instance.

For the per-pixel ray-tracing phase, we prebuild a TLAS for each BLAS in its local space. Throughout the ray-tracing time, ray queries are performed K times in a near-to-far order for each pixel. For every query, we load the corresponding tile k-buffer data of the current pixel to retrieve the volume instance ID, transform the ray into the local space of the target volume instance, and trace the ray to the prebuilt TLAS of the volume instance. Notably, due to the simplicity of the target TLAS, the pixel-level ray tracing can be highly efficient, and an any-hit test can be used instead of a closest-hit test, leveraging the property that the hit point is unique on the interior faces of a cube.

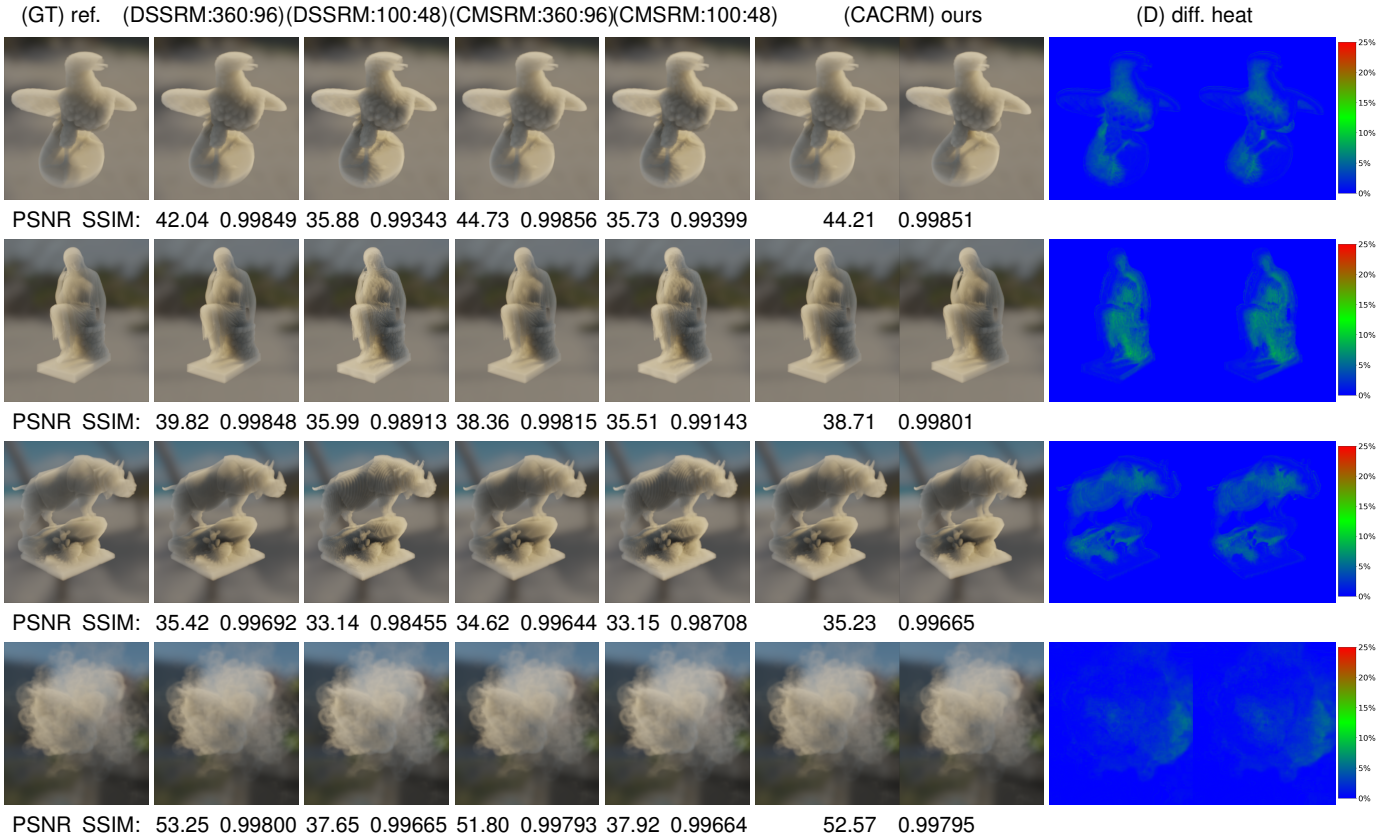


Fig. 7. (GT) Brute-force direct screen-space ray marching with dense uniform-steps of 1024 samples for view direction and 1024 samples for light directions as the ground-truth references, (DSSRM:360:96) direct screen-space ray marching with small uniform-steps of max 360 samples for view direction and 96 samples for light directions, (DSSRM:100:48) DSSRM with large uniform-steps of max 100 samples for view direction and 48 samples for light directions, (CMSRM:360:96) cube-map space ray marching with small uniform-steps of max 360 samples for view direction and 96 samples for light directions, (CMSRM:100:48) CMSRM with large uniform-steps of max 100 samples for view direction and 48 samples for light directions, (CACRM) coupled cube-map space ray marching with our adaptive-rate sampling, and (D) difference heat maps of the results from the ground truths to ours; volume grid size: 256^3, light volume grid size: 128^3; row 4 smoke shows a more transparent volume with low density near the view and high density at far (see the video for other views); except for our (CACRM) and (D), we only show the left views to save paper space.

4 Result Analyses

In this section, we will present our implemented results. In our experiments, we have set up several groups of comparative test cases to verify the quality and performance advantages of our scheme. In our test cases of single volume instances, the volume resolution of density data is 256^3 and the light volume grid size is 128^3 due to the following reasons. The light volume setting can be independent of the input density volume. The per-voxel lighting essentially costs more intensive computations than the view-direction ray marching, while the low-frequency volume lighting is visually less sensitive than the view-direction presentation. Hence, we reduce the light volume resolution accordingly. In terms of the test cases of multi-volumes, we further reduce the input volume resolution and light volume grid size of each instance to 128^3 and 96^3 respectively, so as to sustain the rational consumption of real-time rendering for the entire scene. We ran all test cases in a system equipped with an AMD Ryzen™ 7 5700X CPU with 8 cores of 3.4GHz, a Radeon™ RX 6800 GPU (core clock 1815MHz, memory clock 2000MHZ, and memory size 16GB), and a Oculus Quest 2 VR headset.

4.1 Quality Verification

To begin, we verify the quality of the view-direction ray marching results using direct screen space ray marching with dense uniform sampling steps of 1024 samples per view ray and 1024 samples per light ray to create a ground-truth reference. We compare the results with direct screen-space (DSSRM) and cube-map space ray marching (CMSRM) in small and large uniform steps (Figure 7 (DSSRM:360:96–CMSRM:100:48)), and our coupled adaptive cube map ray marching (CACRM) with ray coupling, variable step amplifications (VSA), and interleaved steps. The numbers appear in the image code name (METHOD:V:L) denote the sample counts for the view direction and the light directions, respectively. We use peak signal-to noise ratio (PSNR) and structure similarity index measure (SSIM) values to rate the quality in Figure 7. Our script captures 100,000 frame shots from various dynamic test cases, and we compute the average PSNR and SSIM values, which are: DSSRM:360:96 (41.17, 0.99677), DSSRM:100:48 (34.87, 0.99142), CMSRM:360:96 (39.64, 0.99781), CMSRM:100:48 (34.21, 0.99128), and our CACRM (39.72, 0.99758). Even with PSNR and SSIM values over 30dB and 0.98, respectively, larger steps and fewer samples (100 samples per view ray and 48 samples per light ray) can produce visible banding artefacts, as seen in Figure 7 (DSSSRM:100:48 and CMSRM:100:48). However, our adaptive sampling eliminates these artefacts with fewer samples, a benefit verified in the performance study. Also, we generate heat maps to visualize the pixel color differences between the results of the ground

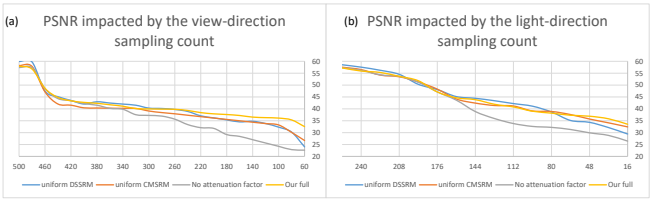


Fig. 8. (a) PSNR impacted by the varying view-direction sampling count, and (b) PSNR impacted by the light-direction sampling count.

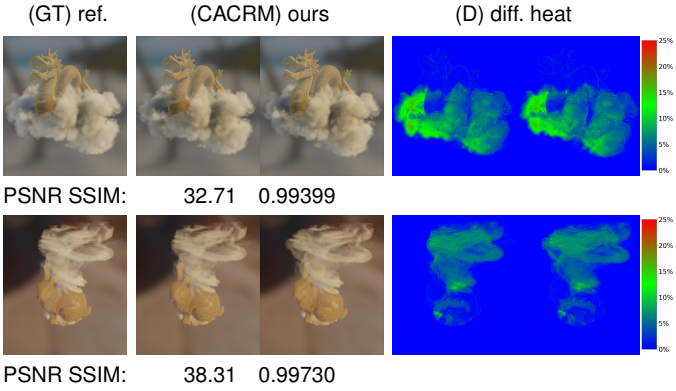


Fig. 9. Mixed translucent mesh and volume rendering with local OIT: (GT) Ground-truth references from DSSRM with k-buffers of meshes, (CACRM) results by our cube map-based method with k-buffers of meshes in binocular views, and (D) difference heat maps of the results from the ground truths to ours.

truth and ours. The heat maps are computed from the per-pixel L1 errors of the normalized RGB values, mapped to [0, 25%] in colors.

In Figure 8, we analyze the PSNR relations to the view and light direction sampling counts. Chart (a) shows that view direction sampling counts significantly impact quality, and cube-map ray marching can reduce aliasing for medium sampling counts. Adaptive sampling can further decrease the necessary sampling count. Chart (b) reveals that cube-map ray marching has better anti-aliasing capabilities for limited sampling budgets, and sampling locations of the light maps are the only quality impacts of different methods as the light-direction ray marching is done per voxel, independent of the view-direction ray marching.

To demonstrate the capabilities of our ray marching method for mixed translucent mesh and volume rendering, we use direct screen-space ray marching with k-buffers of the meshes as the ground-truth references (GT). We then compare these results with those generated by our proposed method (CACRM) using PSNR and SSIM metrics, as shown in Figure 9. The performance information is also presented in Table 1 of the next subsection.

Afterwards, we verify the quality of the lighting results with ambient GI. As illustrated in Figure 10 (GT), the ground-truth references are generated by brute-force uniform spherical sampling with $393,216$ rays per voxel (rpv) on the density field and light probe. In addition, we provide results obtained by precomputed radiance transfer (PRT) using the same sampling amount but with SH transforms, as well as the raw noisy results of 1 rpv (Raw) for additional references. Next, we present the comparative results of the naive SH lighting approximation along the major directions (normal directions from the density gradient) (M) [6], strict reservoir resampling with spatial and temporal reuse (ReSTIR) with our temporal accumulation, and our method of simplified reservoir resampling with adaptive SH temporal accumulation (SReSIR). To evaluate the quality, we generate the pixel-color differences (D) between the results from the ground truths to ours and attach the PSNR and SSIM of all tested methods. The average values are obtained from 100,000 frame shots: (PRT) 55.86, (M) 23.37, (ReSTIR) 34.87, and our (SReSIR) 33.92. Overall, the quality of volume GI approximations has a more significant impact on PSNR than view-direction ray marching tests, but PRT, strict ReSTIR, and our method exhibit no visual artefacts. The difference in quality relative to the ground truth is mainly due to the acceptable approximation of SH transforms and reservoir resampling. Our approach uses coarse ray marching to estimate weights and drops ineffective computations, resulting in quality scores that are comparable to strict ReSTIR. As a result, our method has no significant quality drawbacks compared to state-of-the-art approximation methods. Additionally, the final two rows of Figure 10 demonstrate examples of volume rendering mixed with 3D-mesh geometry.

Moreover, we demonstrate a significant use case for grid-based fluid rendering in Figure 11, where each voxel varies with time due to advection. Our GI method uses temporal accumulation, which requires several frames to converge. We stress tested our method's convergence by rendering dynamic fluid at low frame rates and found that the noise was visually acceptable at 60 FPS but pronounced at 30 FPS. As most VR applications run at 60 FPS or 120 FPS, our temporal accumulation method is applicable.

Furthermore, we have attempted to enhance real-time hair rendering by extending our volume GI method to opacity ambient occlusion (AO) maps from opacity shadow maps [40]. We combine the idea of the opacity shadow map, which creates a volumetric density field voxelized from a hair mesh, to produce the opacity AO map to achieve the GI effect. We apply three-view voxelization from the hair lines and head-mesh triangles on the GPU, based on [41]. Then, we use our GI method from section 3.2 to generate a volume AO map represented by SH coefficients in real time. Finally, we transform the positions on the hairs and mesh surface to the volume space and sampled the corresponding SH data in the volume AO map. Figure 12 displays the results, including images of hairs without shadow or AO, shaded hair using only a shadow map, ground-truth references from brute force sampling, and fully shaded results using our method. We also show the pure shadow and AO mapped onto the hair, and the differences of the results from the ground truths to ours.

In addition, all our dynamic examples are attached in the supplementary video, and more less important results are shown in appendix B.

4.2 Performance Evaluations

View-direction Ray Marching: We compared direct screen-space ray marching and cube-map ray marching with spectral adaptive sampling to our coupled cube-map ray marching with adaptive rate sampling for performance verification. Our test results, as shown in Table 1, revealed that cube-map ray marching is at least 80% faster than direct

This article has been accepted for publication in IEEE Transactions on Visualization and Computer Graphics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2023.3322416

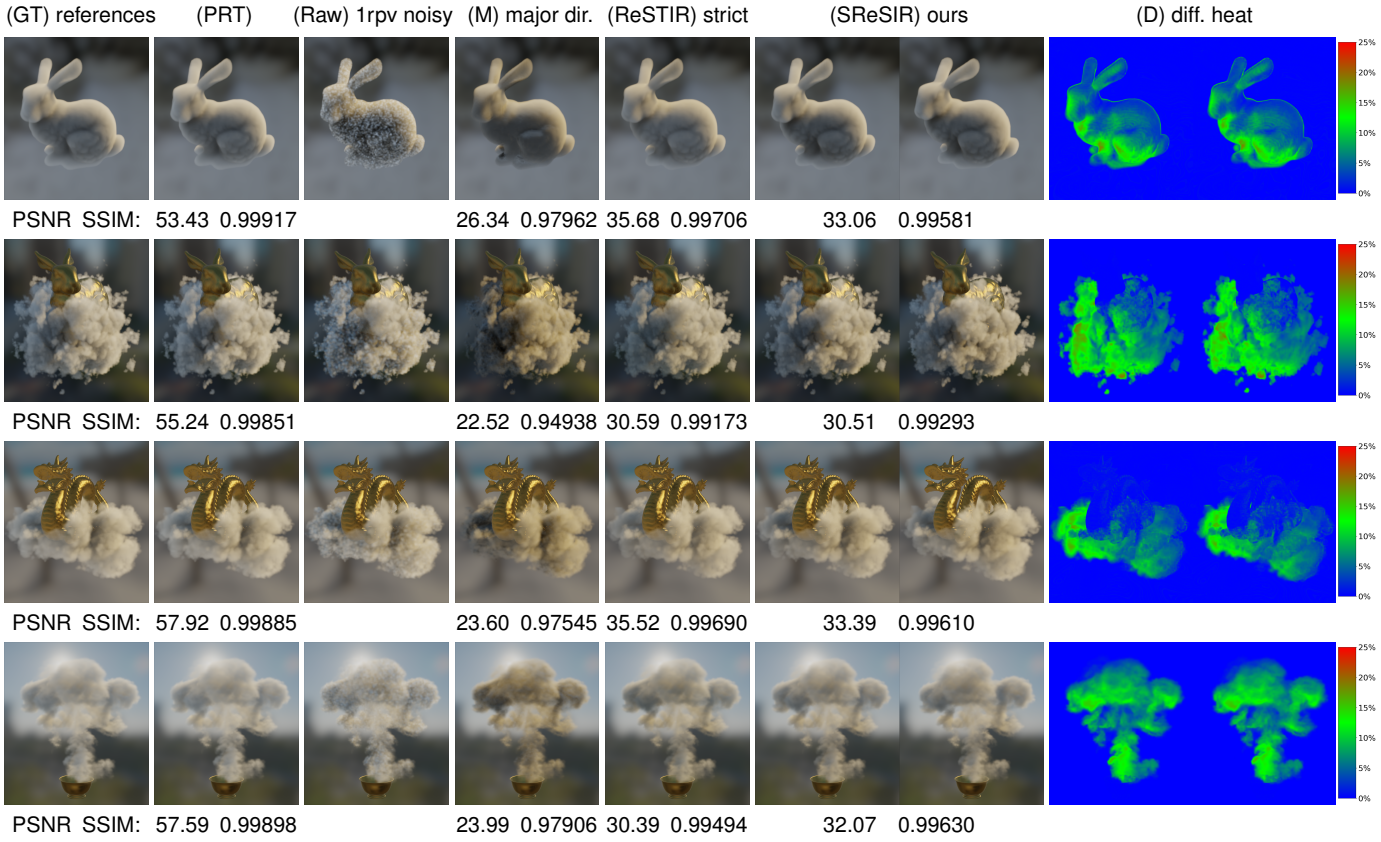


Fig. 10. (GT) Ground-truth references brute-forcefully generated by $393,216$ rpv, (PRT) pre-computed radiance transfer with $24,576$ rpv (same to light-probe size), (Raw) 1 rpv noisy results as the temporal accumulation inputs, (M) naïve SH lighting using SH coefficients along the major directions [6], (ReSTIR) strict ReSTIR with temporal accumulation, (SReSIR) our method of simplified reservoir resampling with our adaptive SH temporal accumulation, and (D) difference heat maps of the results from the ground truths to ours; rows 2–4 include the results of mixed mesh-volume rendering; except for our (SReSIR) and (D), we only show the left views to save paper space.

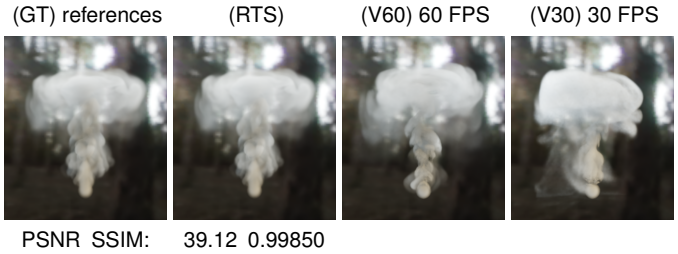


Fig. 11. Fluid rendering with per-voxel advected animation to test the convergence capability of our amortized scheme (all in the left views to save space): (GT) ground-truth references brute-forcefully generated by $393,216$ rpv (RTS) vertical synchronization (v-sync) off and real time step, no obvious noisy dots, (V60) v-sync to 60 FPS and fixed time-step = 0.017 sec/frame, weakly noisy dots, and (V30) v-sync to 30 FPS and fixed time-step = 0.033 sec/frame, noisy dots can be observed.

TABLE 1
Time-cost statistics (in ms) of our tested binocular applications for view-direction ray marching performance comparisons corresponding to Figure 7 (grid size 256^3 and light-map grid size 128^3)

Test case	View res.	DSSRM 360:96	DSSRM 100:48	CMSRM 360:96	CMSRM 100:48	Ours +VSA	Ours +InS	Ours +RCp
Eagle	1080	2.349	1.793	1.265	0.988	1.171	1.135	1.087
	2K	3.984	3.127	1.345	1.074	1.263	1.209	1.141
	4K	7.599	5.865	1.482	1.148	1.392	1.311	1.201
Pene.	1080	2.574	1.802	1.462	0.996	1.356	1.213	1.073
	2K	4.631	3.536	1.486	1.136	1.369	1.241	1.131
	4K	9.295	7.573	1.585	1.243	1.488	1.415	1.326
Jacq.	1080	2.581	1.793	1.481	1.012	1.354	1.257	1.106
	2K	4.132	3.622	1.507	1.153	1.397	1.291	1.134
	4K	8.662	7.858	1.592	1.312	1.535	1.513	1.472
Smoke	1080	2.142	1.423	1.243	0.873	1.146	1.112	0.921
	2K	3.613	2.852	1.451	0.928	1.307	1.156	1.099
	4K	6.953	5.146	1.493	0.972	1.349	1.277	1.118
Transl. mesh-vol.	1080	4.164	2.475	3.268	1.386	2.773	2.589	2.142
	2K	6.782	5.267	5.366	2.435	4.432	3.996	3.227
	4K	11.24	9.364	7.943	4.423	6.824	6.163	5.232

screen-space ray marching due to providing better sampling rates along breadth directions and is view-resolution independent. Our cube-map ray marching has a performance close to cases with large uniform steps while providing quality close to dense-step cases. Our VSA effectively reduces the sampling on the depth direction, and interleaved steps (InS) improve the reuse of neighbor rays. Ray coupling (RCp) further enhances the performance by sharing voxel memory accesses for binocular views. Regarding volume rendering mixed with a translucent mesh, our test results show that employing k-buffers for local OIT process incurs about 30% additional costs compared to an opaque mesh.

Ambient Lighting with GI: To compare performance, we listed the time consumption of our tested methods in Table 2. Since the GI part is viewport-independent, we did not separate the test cases based on different viewport resolutions. As shown in the table, the major-direction approximation method samples only the light probe for SH coefficients of global light sources, with attenuation along major directions. Although this straightforward simplification provides significant performance gains, it results in

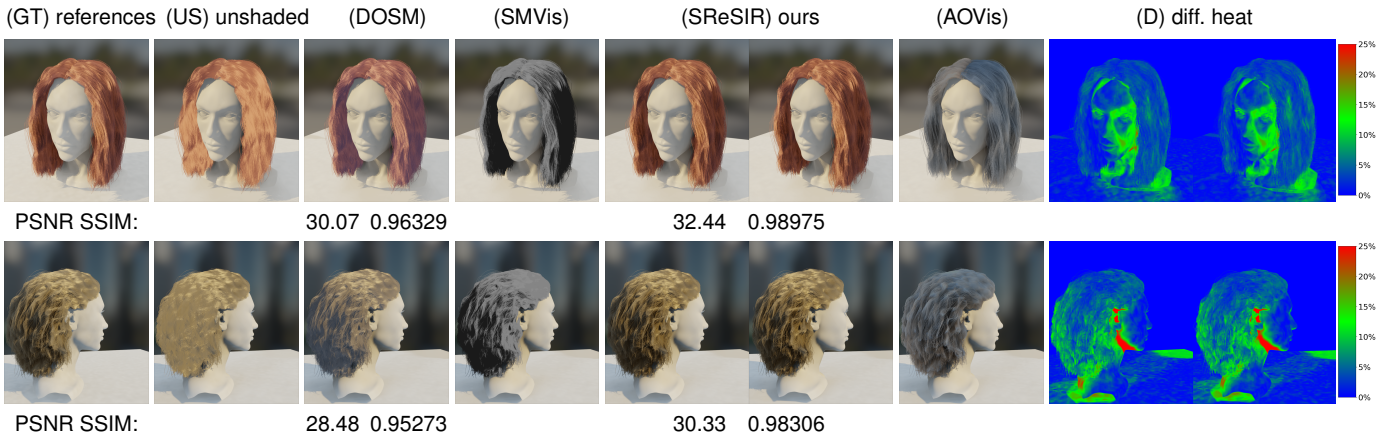


Fig. 12. Our volume GI method can be extended to opacity AO maps for real-time hair rendering: (GT) shaded hairs with opacity AO maps by $393,216$ rpv as the ground-truth references, (US) unshaded hairs without shadows or AO, (DOSM) shaded hairs with opacity shadow maps only (hemisphere ambient model instead of ambient GI) by deep opacity map [39], (SMVis) visualization of opacity shadow maps, (SReSIR) shaded hairs with our method, (AOVis) visualization of opacity AO maps by our method, and (D) difference heat maps of the results from the ground truths to ours; except for our (SReSIR) and (D), we only show the left views to save paper space.

an obvious fidelity loss. Our method performs at double the speed of the strict ReSTIR method by dropping many ineffective computations.

TABLE 2
Time-cost statistics (in ms) of our tested applications in Figure 10 for GI performance comparisons (light-map grid size 128^3)

Test case	Major dir.	Strict ReSTIR	Our method
Bunny	1.242	5.695	2.975
Cloud	1.368	5.759	3.092
Dragon cloud	1.147	4.886	2.379
Gasoline	0.986	4.277	1.994
Fluid smoke	1.353	5.874	3.117
Hairs	0.674	3.023	0.985
	0.795	3.245	1.135

Order-Independent Transparency: We proceeded to compare the performance of OIT methods. We increased the test cases using per-pixel ray tracing, per-pixel k-buffer sorting, and our coarse and fine ray traced OIT with tiled k-buffer, respectively. As displayed in Table 3, our method has clear performance advantages as screen resolution increases since our tile division strategy is independent of the viewport resolution.

TABLE 3
Time-cost statistics (in ms) of our tested binocular applications for OIT performance comparisons (grid size 128^3 and light-map grid size 96^3)

N volumes	View res.	K-buffer	Ray tracing	Our method
4	1080	0.599	0.613	0.503
	2K	1.086	1.134	0.879
	4K	2.276	2.354	0.962
16	1080	0.751	0.712	0.553
	2K	1.324	1.312	0.879
	4K	2.824	2.967	1.158
36	1080	0.851	0.882	0.857
	2K	1.579	1.612	0.979
	4K	3.617	3.698	1.426

Integrated Example: Finally, we recorded the overall performance of our entire solution, shown in Table 4. The video demonstrations are also available. Our approach satisfies high-performance and high-quality rendering requirements of multiple volumes for modern VR applications, as demonstrated in the teaser image (Figure 1). In contrast, the vanilla method can hardly achieve the expected real-time VR requirements for multiple volume instances.

TABLE 4
Frame rate statistics (in frame per second, average FPS) of our tested binocular applications (grid size 128^3 and light-map grid size 96^3)

N volumes	View res.	DSSRM+strict ReSTIR	Our sol.
4	1080	83	213
	2K	46	179
	4K	22	110
16	1080	64	174
	2K	28	145
	4K	14	107
36	1080	49	146
	2K	19	124
	4K	8	97

5 Conclusion

In conclusion, our proposed scheme delivers a novel and efficient method for real-time multi-volume rendering in VR binocular views with fully dynamic GI effects. The cube map space ray marching principles and properties that we have explored and exploited for spatial reuse methods, as well as the dynamic ambient GI approximation method that we have devised, are promising contributions that can potentially enhance the performance of multi-volume rendering while maintaining high quality. The hybrid method of ray tracing and k-buffer that we have proposed also demonstrates potential for boosting the performance of multi-volume drawing in the correct order. While there are some limitations to our approach, such as the lack of consideration for overlapping volumes and indirect lighting from meshes to volumes, our future work can focus on further improvements for lighting-update strategies for multiple volumes, adaptive light-grid resolution adjustments, indirect lighting, and integrating the properties of cube-map ray marching into machine learning for neural rendering. Overall, our proposed scheme provides an important step forward in the advancement of multi-volume rendering techniques for virtual reality applications.

REFERENCES

[1] K. E. Hillesland and J. C. Yang, "Texel shading," in *Proceedings of the 37th Annual Conference of the European Association for Computer Graphics: Short Papers*, ser. EG '16. Goslar, DEU: Eurographics Association, 2016, p. 73–76.

[2] E. Risser, *True Impostors*. Addison-Wesley Professional, 2008.

[3] L. Yang, D. Nehab, P. V. Sander, P. Sitthi-amorn, J. Lawrence, and H. Hoppe, "Amortized supersampling," *ACM Trans. Graph.*, vol. 28, no. 5, p. 1–12, dec 2009. [Online]. Available: https://doi.org/10.1145/1618452.1618481

[4] B. Karis, "High quality temporal anti-aliasing," in *ACM SIGGRAPH 2014 Courses: Advances in RealTime Rendering in Games*, 2014.

[5] A. Marrs, J. Spjut, H. Gruen, R. Sathe, and M. McGuire, "Adaptive temporal antialiasing," in *Proceedings of the Conference on High-Performance Graphics*, ser. HPG '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3231578.3231579

[6] T. Xu, W. Zeng, and E. Wu, "Viewport-resolution independent anti-aliased ray marching on interior faces in cube-map space," in *SIGGRAPH Asia 2021 Technical Communications*, ser. SA '21 Technical Communications. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3478512.3488598

[7] R. A. Drebin, L. Carpenter, and P. Hanrahan, "Volume rendering," *SIGGRAPH Comput. Graph.*, vol. 22, no. 4, p. 65¨C74, jun 1988. [Online]. Available: https://doi.org/10.1145/378456.378484

[8] J. Beyer, M. Hadwiger, and H. Pfister, "A Survey of GPU-Based Large-Scale Volume Visualization," in *EuroVis - STARs*, R. Borgo, R. Maciejewski, and I. Viola, Eds. The Eurographics Association, 2014.

[9] J. Novák, I. Georgiev, J. Hanika, and W. Jarosz, "Monte carlo methods for volumetric light transport simulation," *Computer Graphics Forum*, vol. 37, no. 2, pp. 551–576, 2018. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13383

[10] S. Chandrasekhar, *Radiative Transfer*. Courier Corporation, 1960.

[11] K. Engel, M. Kraus, and T. Ertl, "High-quality pre-integrated volume rendering using hardware-accelerated pixel shading," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, ser. HWWS '01. New York, NY, USA: Association for Computing Machinery, 2001, p. 9¨C16. [Online]. Available: https://doi.org/10.1145/383507.383515

[12] S. Bergner, T. Möller, D. Weiskopf, and D. J. Muraki, "A spectral analysis of function composition and its implications for sampling in direct volume visualization," *IEEE transactions on visualization and computer graphics*, vol. 12, no. 5, pp. 1353–1360, 2006.

[13] A. Muõz, "Higher order ray marching," *Computer Graphics Forum*, vol. 33, no. 8, pp. 167–176, 2014. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12424

[14] M. Raab, D. Seibert, and A. Keller, "Unbiased global illumination with participating media," in *Monte Carlo and Quasi-Monte Carlo Methods 2006*, A. Keller, S. Heinrich, and H. Niederreiter, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 591–605.

[15] C. Kulla and M. Fajardo, "Importance sampling techniques for path tracing in participating media," *Comput. Graph. Forum*, vol. 31, no. 4, p. 1519¨C1528, jun 2012. [Online]. Available: https://doi.org/10.1111/j.1467-8659.2012.03148.x

[16] B. Fraboni, A. Webanck, N. Bonneel, and J.-C. Iehl, "Volumetric multi-view rendering," *Computer Graphics Forum*, vol. 41, 04 2022.

[17] D. Lin, C. Wyman, and C. Yuksel, "Fast volume rendering with spatiotemporal reservoir resampling," *ACM Trans. Graph.*, vol. 40, no. 6, dec 2021. [Online]. Available: https://doi.org/10.1145/3478513.3480499

[18] M. Kettunen, E. D'Eon, J. Pantaleoni, and J. Novák, "An unbiased ray-marching transmittance estimator," *ACM Trans. Graph.*, vol. 40, no. 4, jul 2021. [Online]. Available: https://doi.org/10.1145/3450626.3459937

[19] S. Parker, M. Parker, Y. Livnat, P.-P. Sloan, C. Hansen, and P. Shirley, "Interactive ray tracing for volume visualization," in *ACM SIGGRAPH 2005 Courses*, ser. SIGGRAPH '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 15¨Ces. [Online]. Available: https://doi.org/10.1145/1198555.1198754

[20] P.-P. Sloan, J. Kautz, and J. Snyder, "Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments," *ACM Trans. Graph.*, vol. 21, no. 3, p. 527–536, jul 2002. [Online]. Available: https://doi.org/10.1145/566654.566612

[21] J. Kruger and R. Westermann, "Acceleration techniques for gpu-based volume rendering," in *IEEE Visualization, 2003. VIS 2003.* IEEE, 2003, pp. 287–292.

[22] M. Smelyanskiy, D. Holmes, J. Chhugani, A. Larson, D. M. Carmean, D. Hanson, P. Dubey, K. Augustine, D. Kim, A. Kyker, V. W. Lee, A. D. Nguyen, L. Seiler, and R. Robb, "Mapping high-fidelity volume rendering for medical imaging to cpu, gpu and many-core architectures," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, pp. 1563–1570, 2009.

[23] C. Crassin, F. Neyret, S. Lefebvre, and E. Eisemann, "Gigavoxels: Ray-guided streaming for efficient and detailed voxel rendering," in *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*, ser. I3D '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 15–22.

[24] N. Morrical, W. Usher, I. Wald, and V. Pascucci, "Efficient space skipping and adaptive sampling of unstructured volumes using hardware accelerated ray tracing," in *2019 IEEE Visualization Conference (VIS)*. Vancouver, BC, Canada: IEEE, 2019, pp. 256–260.

[25] M. O. Derin, T. Harada, Y. Takeda, and Y. Iba, "Sparse volume rendering using hardware ray tracing and block walking," in *SIGGRAPH Asia 2021 Technical Communications*, ser. SA '21 Technical Communications. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3478512.3488608

[26] *Vulkan®1.3.230 - A Specification*, The Khronos Group, Inc., September 2022. [Online]. Available: https://registry.khronos.org/vulkan/specs/1.3/html/

[27] A. Jindal, K. Wolski, K. Myszkowski, and R. K. Mantiuk, "Perceptual model for adaptive local shading and refresh rate," *ACM Trans. Graph.*, vol. 40, no. 6, dec 2021. [Online]. Available: https://doi.org/10.1145/3478513.3480514

[28] R. Wang, J. Zhu, and G. Humphreys, "Precomputed radiance transfer for real-time indirect lighting using a spectral mesh basis," in *Proceedings of the 18th Eurographics Conference on Rendering Techniques*, ser. EGSR'07. Goslar, DEU: Eurographics Association, 2007, p. 13–21.

[29] P.-P. Sloan, B. Luna, and J. Snyder, "Local, deformable precomputed radiance transfer," *ACM Trans. Graph.*, vol. 24, no. 3, p. 1216–1224, jul 2005. [Online]. Available: https://doi.org/10.1145/1073204.1073335

[30] K. Iwasaki, Y. Dobashi, F. Yoshimoto, and T. Nishita, "Precomputed radiance transfer for dynamic scenes taking into account light interreflection," in *Proceedings of the 18th Eurographics Conference on Rendering Techniques*, ser. EGSR'07. Goslar, DEU: Eurographics Association, 2007, p. 35–44.

[31] J. F. Talbot, D. Cline, and P. Egbert, "Importance resampling for global illumination," in *Proceedings of the Sixteenth Eurographics Conference on Rendering Techniques*, ser. EGSR '05. Goslar, DEU: Eurographics Association, 2005, p. 139–146.

[32] Y. Tokuyoshi, "Tiled reservoir sampling for many-light rendering," AMD, Tech. Rep. 21-11-ecdc, 2021.

[33] D. Lin, M. Kettunen, B. Bitterli, J. Pantaleoni, C. Yuksel, and C. Wyman, "Generalized resampled importance sampling: Foundations of restir," *ACM Trans. Graph.*, vol. 41, no. 4, jul 2022. [Online]. Available: https://doi.org/10.1145/3528223.3530158

[34] B. Bitterli, C. Wyman, M. Pharr, P. Shirley, A. Lefohn, and W. Jarosz, "Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting," *ACM Trans. Graph.*, vol. 39, no. 4, jul 2020. [Online]. Available: https://doi.org/10.1145/3386569.3392481

[35] Y. Ouyang, S. Liu, M. Kettunen, M. Pharr, and J. Pantaleoni, "ReSTIR GI: Path Resampling for Real-Time Path Tracing," *Computer Graphics Forum*, 2021.

[36] G. Boissé, "World-space spatiotemporal reservoir reuse for ray-traced global illumination," in *SIGGRAPH Asia 2021 Technical Communications*, ser. SA '21 Technical Communications. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3478512.3488613

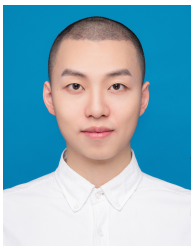
[37] A. A. Vasilakis and I. Fudos, "K+-buffer: Fragment synchronized k-buffer," in *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ser. I3D '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 143–150. [Online]. Available: https://doi.org/10.1145/2556700.2556702

[38] F. Xie, E. Tabellion, and A. Pearce, "Soft shadows by ray tracing multilayer transparent shadow maps," in *Proceedings of the 18th Eu-*

rographics Conference on Rendering Techniques, ser. EGSR'07. Goslar, DEU: Eurographics Association, 2007, p. 265–276.
[39] C. Yuksel and J. Keyser, "Deep Opacity Maps," *Computer Graphics Forum*, 2008.
[40] T.-Y. Kim and U. Neumann, "Opacity shadow maps," in *Rendering Techniques 2001*, S. J. Gortler and K. Myszkowski, Eds. Vienna: Springer Vienna, 2001, pp. 177–182.
[41] M. Schwarz and H.-P. Seidel, "Fast parallel surface and solid voxelization on gpus," *ACM Trans. Graph.*, vol. 29, no. 6, dec 2010. [Online]. Available: https://doi.org/10.1145/1882261.1866201

Tianchen Xu received his BSc degree and MSc degree in Software Engineering from University of Macau in summers of 2011 and 2014 respectively. Then, he worked for NVIDIA as a developer technology engineer from 2015 to 2016. He has been working for AMD as a software development engineer since 2016. Currently, he is pursuing PhD degree at State Key Lab., Institute of Software, Chinese Academy of Sciences (CAS) and University of CAS. His research interests include real-time rendering, character animation, and GPU-based fluid simulation.

Xiaohua Ren is currently a senior computer graphics researcher at Multimedia Research Center, Tencent. He obtained his PhD degree from the University of Macau in 2019. His research interests lie in fluid simulation, image/geometry processing, and deep learning.



Jiale Yang received his B.Eng. degree in Computer Science and Technology from Wuhan University. He is now a PhD student in the Department of Computer Science at Shanghai Jiao Tong University, and working for AMD. His research interests include computer graphics and deep learning.

Bin Sheng received the B.A. degree in English and the B.Eng. degree in computer science from the Huazhong University of Science and Technology, Wuhan, China, in 2004, and the M.Sc. degree in software engineering from the University of Macau, Taipa, Macau, in 2007, and the PhD degree in computer science and engineering from The Chinese University of Hong Kong, Shatin, Hong Kong, in 2011. He is currently a Full Professor with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China. He is an Associate Editor of the IEEE Transactions on Circuits and Systems for Video Technology. His current research interests include virtual reality and computer graphics.

Enhua Wu Enhua Wu received his BSc in 1970 from Tsinghua University, Beijing and stayed teaching there until 1980. He was awarded PhD degree from University of Manchester, UK in 1984, followed by working at the State Key Lab. of Computer Science, Chinese Academy of Sciences since 1985, and also as a full professor of University of Macau since 1997. He is an Associate Editor-in-Chief of JCST since 1995, and the editorial board member of CAVW, Visual Informatics. He is a fellow member of the China Computer Federation (CCF). His research interests include realistic image synthesis, physically based simulation, and virtual reality.