

Interactive Path Tracing and Reconstruction of Sparse Volumes

NIKOLAI HOFMANN, NVIDIA, Germany

JON HASSELGREN, NVIDIA, Sweden

PETRIK CLARBERG, NVIDIA, Sweden

JACOB MUNKBERG, NVIDIA, Sweden



Fig. 1. Our volume renderer coupled with a temporally stable neural denoiser is running interactively on the DISNEY CLOUD [DisneyAnimation 2020] scene, consisting of a sparse voxel grid with 188M unique voxels. Rendering and inference times at 1920×1080 on a NVIDIA RTX 3090: 61 ms for 4 spp including denoising (middle) and about 60 s for the reference rendering (right).

We combine state-of-the-art techniques into a system for high-quality, interactive rendering of participating media. We leverage *unbiased* volume path tracing with multiple scattering, temporally stable neural denoising and NanoVDB [Museth 2021], a fast, sparse voxel tree data structure for the GPU, to explore what performance and image quality can be obtained for rendering volumetric data. Additionally, we integrate neural adaptive sampling to significantly improve image quality at a fixed sample budget. Our system runs at interactive rates at 1920×1080 on a single GPU and produces high quality results for complex dynamic volumes.

CCS Concepts: • **Computing methodologies** → **Ray tracing**; **Neural networks**; *Image processing*.

Additional Key Words and Phrases: volumetric rendering, path tracing, denoising, image reconstruction, machine learning

Authors' addresses: Nikolai Hofmann, NVIDIA, Germany; Jon Hasselgren, NVIDIA, Sweden; Petrik Clarberg, NVIDIA, Sweden; Jacob Munkberg, NVIDIA, Sweden.

© 2021 Association for Computing Machinery.

This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, <https://doi.org/10.1145/3451256>.

ACM Reference Format:

Nikolai Hofmann, Jon Hasselgren, Petrik Clarberg, and Jacob Munkberg. 2021. Interactive Path Tracing and Reconstruction of Sparse Volumes. *Proc. ACM Comput. Graph. Interact. Tech.* 4, 1 (May 2021), 19 pages. <https://doi.org/10.1145/3451256>

1 INTRODUCTION

Path tracing [Kajiya 1986] is the de-facto standard for photo-realistic image synthesis and is widely adopted in the visual effects industry [Georgiev et al. 2018]. It comes with high computational costs, and has mostly been applied to offline applications, where rendering times can be measured in minutes or hours per frame. Recent advances in ray tracing hardware and image filtering enables high-quality *interactive* rendering of triangular geometry with ray/path tracing techniques.

However, accurately rendering participating media, such as clouds, explosions, and smoke, remains a computationally complex problem as it cannot leverage ray traversal hardware primitives for triangle meshes. Combined with the high-dimensional problem of integrating multiple scattering, a considerable gap to bringing such renderings into the interactive domain exists.

We combine state-of-the-art rendering and neural denoising techniques to show that interactive performance at high image quality can be obtained with *unbiased* volume path tracing. We build a high-performance GPU volume path tracer by leveraging NanoVDB [Musetth 2021], which provides GPU acceleration of OpenVDB [Musetth 2013; Museth et al. 2013] assets and algorithms, an industry standard library for manipulating sparse dynamic volumes. By incorporating adaptive sampling, empty space skipping, and stochastic filtering, we achieve very high performance. The path tracer is combined with a temporally stable neural denoiser, which we adapt for dynamic, path-traced volumetric data. A separate small network predicts the sample density to maximize image quality at a given sample count.

Previous work on interactive volume rendering, which typically relies on biased ray-marching methods and single-scattering lighting models, can drastically change the visual appearance, as illustrated in Figure 3. Our system achieves significantly higher image quality while maintaining interactive frame rates at 1920×1080 resolution on a single GPU. Figure 1 shows one example of our approach applied to a cloud with 188M voxels, and Figure 2 shows a gasoline explosion including emission.

The contributions of our work are as follows:

- A system for high-quality volume rendering with multiple scattering and interactive performance.
- A spatio-temporal denoising architecture fine-tuned for volume data at low sample counts.
- An ablation study of the quality impact of design choices for the renderer and denoiser.

1.1 Background

Rendering with Participating Media. An extensive overview over existing techniques and challenges for rendering with participating media is given in a state of the art report by Novák et al. [Novák et al. 2018]. The change in radiance through a differential volume segment, caused by absorption, in-scattering, out-scattering, and emission, is given by the *radiative transfer equation* (RTE) [Chandrasekar 1960]:

$$(\omega \cdot \nabla)L(x, \omega) = \mu_a(x)L(x, \omega) - \mu_s(x)L(x, \omega) + \mu_s(x)L_s(x, \omega) + \mu_e(x)L_e(x, \omega), \quad (1)$$

where $\mu_a(x)$ and $\mu_s(x)$ are the absorption and scattering coefficients at point x in the volume, respectively. The in-scattered radiance L_s is collected over all directions on the unit sphere (S_x)



Fig. 2. Converged volumetric rendering of a dense GASOLINE EXPLOSION [JangaFX 2020] cloud, lit by an environment light probe and emission. In this configuration, 1 spp renders in 11.9 ms at 1920×1080 resolution on a NVIDIA RTX 3090 GPU.

around x :

$$L_s(x, \omega) = \int_{S_x} f_p(\omega, \tilde{\omega}) L_i(x, \tilde{\omega}) d\tilde{\omega}, \quad (2)$$

where $f_p(\omega, \tilde{\omega})$ is the *phase function*, describing the directional scattering probabilities in the medium. Emission L_e is modulated by the absorption coefficient to force all radiance into $W/m^2/sr$ units and to only occur where energy is absorbed first. The corresponding integral formulation, which is ultimately solved via Monte Carlo integration during rendering, is the following:

$$L(x, \omega) = \int_0^\infty T(x, y) [\mu_a(y) L_e(y, \omega) + \mu_s(y) L_s(y, \omega)] dy. \quad (3)$$

The *transmittance* $T(x, y)$ between two points, i.e., the loss of radiance due to absorption and out-scattering, is given via the Beer-Lambert law [Lambert 1760]:

$$T(x, y) = e^{-\int_x^y \mu_t(x+t\omega) dt}, \quad (4)$$

where $\mu_t(x) = \mu_a(x) + \mu_s(x)$ and t is the distance along the ray segment between points x and y . Example renderings showcasing these effects are shown in Figures 1 and 2.

Null Collision Algorithms. In order to approximate the integral in Equation 3 with Monte Carlo integration, we need a technique that allows estimating transmittance and, ideally, also a technique that distributes samples proportional to it. Ray marching [Perlin and Hoffert 1989] is a common approach, which consists of stepping along the ray in fixed increments and accumulating optical thickness. While conceptually simple, this algorithm assumes piecewise constant extinction between marching steps, which introduces bias, even if the stepping is randomly jittered [Raab et al. 2008]. This systematic bias becomes especially apparent around thin features of the volume, which are likely to be missed when stepping with higher step size than the Nyquist frequency, i.e., half the



Fig. 3. Sources of bias for volumetric rendering at converged state: single scattering only (top, left) versus fully integrating multiple scattering (top, right) on the SMOKE PLUME [JangaFX 2020] scene. Furthermore, thin features of the DISNEY CLOUD [DisneyAnimation 2020] may be missed when employing ray marching (bottom, left), over unbiased null-collision trackers (bottom, right). Note how the ray marching bias persists, even despite we randomly jittered the marching steps.

voxel size in a discrete grid, which is very costly in practice. Therefore, to retain high quality and remain unbiased, we rather employ ratio tracking [Novák et al. 2014] for transmittance estimation and delta tracking [Galtier et al. 2013; Woodcock et al. 1965] for distance sampling. These methods were originally designed for free path sampling in heterogeneous media in the field of neutron transport and operate by homogenizing the heterogeneous medium via padding with fictitious matter that has no effect on light transport. Doing so enables analytically sampling for collisions, while rejecting the so-called *null-collisions* with fictitious matter and continue unaffected in order

to return to the correct free-path distribution [Novák et al. 2018]. See Figure 3 for a visual example of possible sources of bias when limiting rendering to single scattering or employing ray marching.

Denoising. Monte Carlo integration approximates the underlying integral through stochastic sampling, and the sample mean is an unbiased estimator of the correct value. Unfortunately the variance converges only slowly with more samples, which causes renders with low sample counts to exhibit high amounts of noise. This motivates the use of reconstruction techniques to reduce residual noise in the resulting renderings instead of accumulating additional samples for the best trade-off between quality and performance. A multitude of recent works [Bako et al. 2017; Chaitanya et al. 2017; Hasselgren et al. 2020; Kuznetsov et al. 2018; Vogels et al. 2018] take a data-driven approach and use neural networks to yield highly effective denoising filters for Monte Carlo rendering, mostly outperforming their hand-crafted counterparts. In this work, we heavily rely on the usage of such neural denoisers to achieve higher frame rates, while staying in a reasonable error range.

1.2 Related Work

Real-time Volumetric Rendering. A multitude of previous works aim to bring effects from volume rendering into the context of real-time rendering [Delalandre et al. 2011; Gautron et al. 2011], such as video games [Lagarde and Golubev 2018; Vos 2014; Wronski 2014], especially regarding single scattering [Engelhardt and Dachsbacher 2010; Gautron et al. 2011; Wyman and Ramsey 2008]. Since all of these approaches operate in a highly runtime constrained environment, they often rely on some form of ray marching with (varying) high step sizes and temporal filtering [Karis 2014] to achieve best possible frame rates. In order to reduce the cost of tracing shadow rays, pre-integrating transmittance and other volumetric properties into a low-resolution frustum-aligned volume texture, analogous to shadow maps [Williams 1978] for volumes [Chen et al. 2011], and relying on temporal integration has shown promising results as well [Hillaire 2015]. Potentially including a fallback to ray marching to more effectively cover large distances [Bauer 2019]. Note that most of these techniques are aimed at creating dynamic clouds and god rays, and are limited to fairly low-frequency single-scattering lighting.

Offline Volumetric Rendering. We refer the reader to the state of the art report by Novák et al. [2018] and focus on the, for our work, most relevant papers and emission sampling. Capturing volumetric lighting by importance sampling heterogeneous emission grids has been shown to yield promising results [Villemin et al. 2013]. Simon et al. [2017] propose to further improve the sampling of heterogeneous, emissive volumes by both integrating along the ray during DDA traversal and efficiently computing the PDF for an importance sampled position in an emission grid to enable the usage of *multiple importance sampling (MIS)* [Veach and Guibas 1995]. Furthermore, it has been shown that by decomposition of the medium into constant and residual parts, expensive lookups of spatially varying coefficients may be omitted due to sampling the constant component in closed form, reducing overall rendering costs, while staying unbiased [Kutz et al. 2017]. Similarly to our work, Kallweit et al. [2017] employ neural networks to speed up the rendering process. They propose to terminate long paths and predict in-scattered radiance via a hierarchy of multilayer perceptrons for high-quality cloud rendering.

Denoising. There has been substantial progress in denoising for both offline and real-time rendering. We refer the interested reader to Zwicker et al. [2015] for an extensive survey. Here, we will focus on recent denoising research that applies deep learning to craft the denoisers, which we also deploy in this work. Furthermore, recent work indicate that deep learning based denoisers tend to outperform hand-crafted algorithms in terms of quality [Vogels et al. 2018].

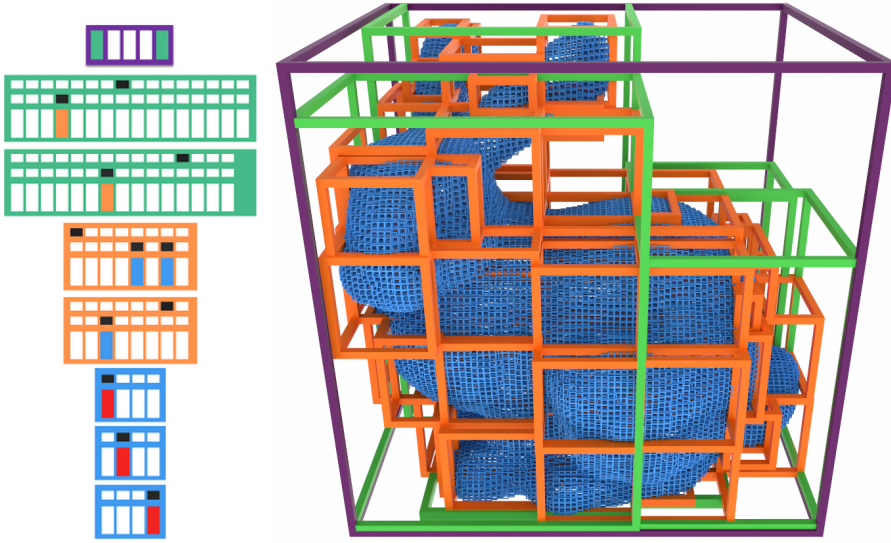


Fig. 4. Visualization of the internal tree structure of (non-overlapping) nodes in NanoVDB [Museth 2021]. The root of the tree is visualized in purple, while the respective lower nodes are visualized in green, orange and blue respectively. Each node may encode up to $8 \times 8 \times 8$ references to the level below, while the lowest level (blue) contains data. But note that data might alternatively be stored in a node higher up the tree as well, efficiently encoding regions of space with constant value.

This development started with denoisers [Bako et al. 2017; Vogels et al. 2018] for production rendering, employing large U-nets [Ronneberger et al. 2015] to predict unique filter kernels for denoising each pixel. These *kernel predicting networks* [Bako et al. 2017] tend to perform better than direct image prediction, because they can effectively enforce certain criteria such as energy preservation and reduce color shifts, as the same kernel is applied on each color component. Recently, these methods have further been refined to include temporal reuse [Hasselgren et al. 2020; Vogels et al. 2018] and learned adaptive sampling [Hasselgren et al. 2020; Kuznetsov et al. 2018]. Runtime performance has also improved by several orders of magnitude [Chaitanya et al. 2017; Hasselgren et al. 2020; Meng et al. 2020] by optimizing the network architecture and migrating from high level frameworks to customized shader kernels. This makes real-time neural denoising a possibility with per-frame times of approximately 10 ms at a resolution of 1920×1080 pixels on high end GPUs. Most previous work focus on path tracing of triangle meshes. A notable exception is Hofmann et al. [2020] who propose a denoiser specifically for volumetric medical data. Their architecture is similar to the aforementioned neural denoisers, but is trained uniquely on volume data. As such it is reasonable to assume that most features of general denoisers can be applied successfully to volume rendering as well.

In this work, we leverage this recent progress in denoising research and fine-tune an efficient *spatio-temporal* denoiser for sparse volumetric data at low sample counts. More specifically, we base our denoiser on the architecture proposed by Hasselgren et al. [2020], as it includes kernel prediction, an efficient temporal feedback loop, and runs in real-time.

2 RENDERER

Path Tracer. We implemented an unbiased volumetric path tracer with multiple scattering purely in software, using Direct3D 12 compute shaders in the Falcor [Benty et al. 2020] real-time rendering

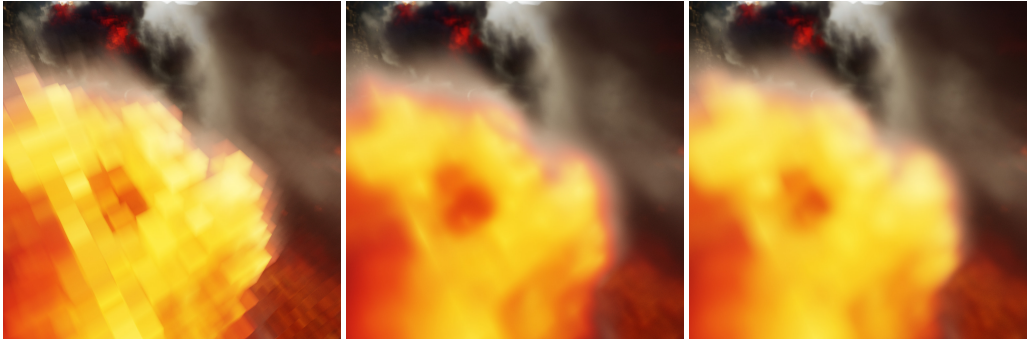


Fig. 5. Nearest neighbor (left, 7.2 ms), trilinear (middle, 31.9 ms) and stochastic (right, 7.9 ms) neighborhood sampling in an explosion cloud. Timings are GPU frame times for 1 spp at 1024×1024 pixels on a NVIDIA RTX 3090. Note how the stochastic interpolation scheme slightly overestimates emissive highlights due to a non-linear mapping of temperature to color when compared to trilinear interpolation, but at nearly a quarter of the cost.

framework. At each path vertex, we use *next-event estimation* (NEE), i.e., trace an importance sampled shadow ray towards the light probe, relying on a mipmap hierarchy and the warping approach from Clarberg et al. [2005] to leverage the GPU’s texture units. We use the Henyey-Greenstein phase function [Henyey and Greenstein 1941] to model both isotropic and anisotropic scattering properties and importance sample the next scattering direction at a path vertex. In order to combine samples from both NEE and accidental light probe hits from free paths, we rely on *multiple importance sampling* (MIS) [Veach and Guibas 1995]. We realize unbiased distance sampling along a path via delta tracking and transmittance estimation along shadow rays via ratio tracking [Novák et al. 2018], while applying *Russian roulette* (RR) to paths with low transmittance or throughput. We empirically chose to apply RR to paths with throughput, or shadow rays with transmittance, below 0.1. We use global majorants, but note that localized majorants could further accelerate the estimation.

Our scenes are currently lit from both an HDR environment light probe and emissive volumes, but extending the shading to support other types of light sources is straightforward. In order to isolate the problem definition and reduce complexity, our scenes contain one (animated) volume, such as an explosion or smoke simulation. We load sparse OpenVDB or NanoVDB grids containing density and optionally temperature fields, supplying a still frame or pre-compiled animations. See Figure 4 for an illustration of the data structure. Although they tend to be slightly faster to access, we refrain from employing dense 3D textures as volume descriptions due to excessive memory requirements, which would effectively prohibit the ability to render high-quality animated volumes on the GPU. For example, solely the cloud shown in Figure 1 would consume approximately 3 GB of GPU memory when stored densely, compared to 585 MB for the sparse voxel tree.

Volume Filtering. Since our volumes are discrete grids, they exhibit a regular structure, especially when viewed up close. The straightforward solution is to smooth the blocky structure via linear or cubic interpolation, but this comes with a significant runtime cost. In our tests, trilinear interpolation imposed at least a $3.5\times$ runtime cost compared to nearest neighbor sampling, due to the sparseness of the data. Thus, we opt to solve the interpolation problem via Monte Carlo integration instead and stochastically select a neighboring voxel based on the fraction of the index coordinate, which we consider equivalent to importance sampling of (linear) interpolation, similar to Ernst et al. [2006]. Since the PDF cancels out, we can simply substitute the (*index-space*) nearest neighbor lookup at P

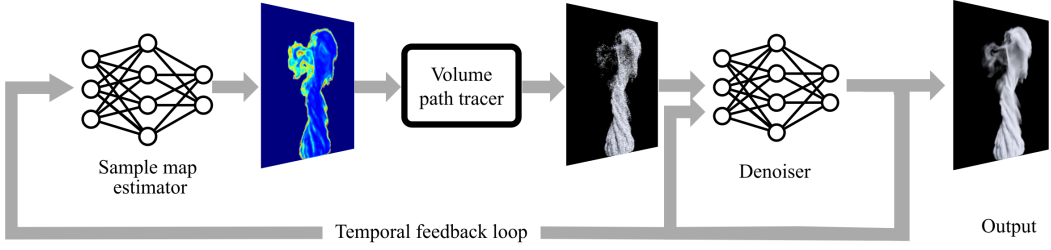


Fig. 6. For interactive path tracing and reconstruction of sparse volumes, we couple neural adaptive sampling, adaptive rendering and neural denoising. Note that the adaptive sampler and denoiser both rely on a temporal feedback loop, i.e., the denoised result from the previous time step is used to drive the adaptive sampler and is fed as an input guide to the denoiser. We do not require an initial sampling pass, but rely uniquely on the temporal feedback loop to estimate adaptive sampling. We train the denoiser and adaptive sampler jointly in this configuration, with loss computed only on the final output.

with $\text{floor}(P + u - 0.5f)$, where u is a 3 component vector of random samples in $[0, 1]$. Note that we refer to *index-space* as the space, where moving by 1 unit along an axis results in exactly arriving at the neighboring voxel in the tree. This yields similar converged results to trilinear interpolation at the marginal cost of drawing additional samples. Note that in order to avoid the expensive neighborhood lookups and compute the PDF, this interpolation scheme assumes a linear relation between neighboring voxels, which might cause a change in appearance when applying a non-linear mapping on top, such as black body emission [Dicke et al. 1965] or arbitrary transfer functions. See Figure 5 for an illustration of this effect. We argue that the performance gain strongly outweighs the slight variation in appearance when compared to trilinear interpolation. And since there is no reliable information about neighborhood relations at sub-voxel resolution in a discrete voxel grid in the first place, we consider both approaches to be equally viable.

Emission Sampling. To incorporate emissive effects from, e.g., fire or explosions, we load a second grid containing temperature values, which are then mapped to color via implementing either a black body emitter or an arbitrary transfer function. For rendering explosions for example, we rely on a simple transfer function $(r, g, b) = (t, t^2, t^4)^2$, where t is a temperature value in $[0, 1]$. Analogous to line integration proposed by Simon et al. [2017], we perform temperature lookups at all collisions during delta tracking, i.e., both real and null, to increase the sampling rate of the emission grid and improve the integration of thin features. Each emission lookup is scaled by the probability of the real collision to correctly weigh the samples. We experimented with collecting emission along both scatter and shadow rays as well, weighting each by 0.5, but observed only marginal gains from considering emission along shadow rays at an approximately 10% performance penalty, which we do not consider worthwhile.

To further improve sampling of thin, bright emissive features, we implemented *forward next event estimation* (FNEE) [Simon et al. 2017]. We importance sample the downsampled emission grid via the same warping approach we employ for environment maps from Clarberg et al. [2005], adapted to three dimensions. We observed improved convergence rates with FNEE enabled around bright emissive spots, but at a considerable increase in runtime costs (30 – 50%, scene dependent) and higher variance overall due to divergence when choosing between sampling the environment map and emission grid during shading. We assume this approach to be more applicable for scenes where surfaces are mainly lit from an emissive volume, as was previously demonstrated, but was not the case in our experiments. For these reasons, we decided to not use FNEE in our interactive renderer.

Work Load Balancing. Due to stochastic sampling of collisions and the high dimensional problem of multiple scattering, the variance of (even neighboring) paths may be significant, not only in terms of scattering positions and directions, but also path lengths. Effectively mapping such a workload to the *single instruction, multiple threads (SIMT)* architecture of modern GPUs is not straightforward. Motivated by Aila et al. [2009], we aim to increase occupancy of the GPU by implementing both their *persistent threads* and *dynamic fetch* load balancing schemes. Thus, we split the rendering task into two steps: an initial pass over each pixel to queue primary rays into a global work queue and a second pass to consume all rays from the queue, decoupled from pixels and effectively allocating all the GPU's resources. This approach not only increases occupancy for rendering a single sample per pixel, but also allows for slightly sub-linear scaling for tracing multiple samples per pixel in a single pass, simply by queuing multiple primary rays per pixel initially.

Adaptive Sampling. Additionally, our load balancing setup allows for a straightforward implementation of adaptive sampling at negligible cost. We adaptively allocate global queuing and framebuffer memory based on an input sample density map containing per-pixel sample counts, which we reduce using prefix sum operations. In the initial pass, we lookup the sample count for each pixel in the density map and append the corresponding number of primary rays to the work queue.

Empty Space Skipping. In order to avoid unnecessary volume lookups, we clip all rays against the bounding box of the volume and additionally utilize the *hierarchical digital differential analyzer (HDDA)* from Museth et al. [2014] to quickly step through empty space. When the first non-empty voxel is found, we start delta tracking until either a real collision occurs, or we have escaped the volume bounds. This results in a considerable speedup due to both culling rays from the queue that hit the bounding box, but will miss the (sparse) volume, and by advancing all primary rays through empty space. We therefore extend our queuing scheme to hold an additional floating point number, signaling the parametric distance t along the ray to start the path at. Fortunately, this initial traversal comes at a relatively small runtime cost, mainly due to the hierarchical nature of the DDA, high branching factor of the NanoVDB tree and high coherence between camera rays. We refer the reader to Section 4.1 for a more detailed breakdown of rendering performance.

3 DENOISING

To remove the residual noise from the volumetric rendering at low sample counts, we take a data-driven approach and train a denoiser on a large corpus of animated volume rendering clips. Our system targets interactive rendering of dynamic volumes, so we require a robust, temporally stable neural network capable of denoising volumes from low input sample counts. Ideally, the denoiser should run in less than 10 ms at a resolution of 1920×1080 pixels.

3.1 Network Architecture

We leverage recent work in neural denoising, which has been successfully applied to medical volumetric data [Hofmann et al. 2020]. To meet the runtime performance requirements of our system, we adapt the neural denoising architecture by Hasselgren et al. [2020]. Our system and data flow is shown in Figure 6. The raw output samples of the volume path tracer are denoised by a network, which uses a temporal recurrence loop to effectively reuse data between frames in an animation. We also optionally support adaptive sampling through a sample map estimator network [Hasselgren et al. 2020], that relies entirely on temporally reused data and therefore does not require an initial sampling pass.

This denoiser network is illustrated in Figure 7 (right). The network takes the tone mapped noisy image, alongside *feature guides*, e.g., positions, depth, density, and the tone mapped denoised output from the previous time step as input. We derive all our feature guides from the primary scatter

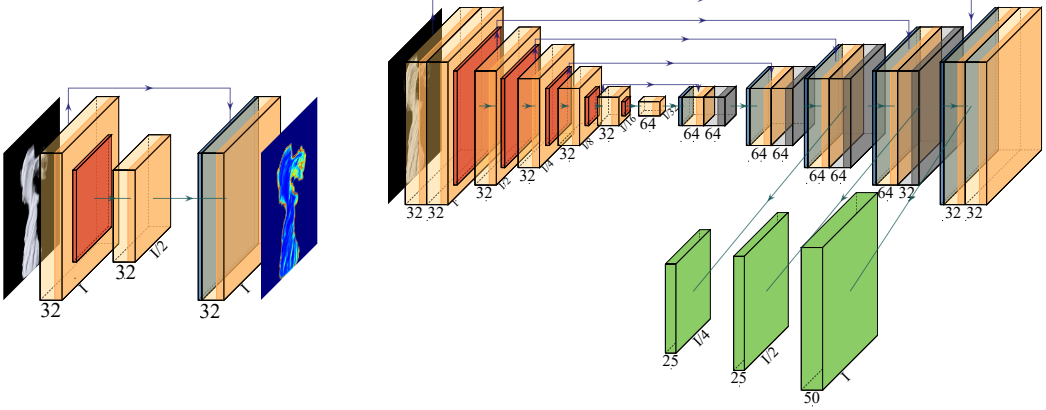


Fig. 7. Our denoiser architecture (right) is adapted from Hasselgren et al. [2020]. It is a U-net (upper part) with three levels of hierarchical kernel prediction (shown in green). The kernels are applied to 4 \times , 2 \times , and full resolution versions of the noisy input. In addition, at the finest level, an additional kernel is predicted and applied to the denoised history buffer. Our adaptive sample map estimator network (left) is a small U-net with three layers. The input is the denoised frame from the previous timestep, and the network outputs a sample count per pixel, indicating where to trace more paths.

position along the camera ray and compute both mean and standard deviation over multiple samples per pixel in order to better cope with the stochastic nature of the sampling process. See Figure 8 for an illustration of our input feature vector.

The network is a U-net [Ronneberger et al. 2015], e.g., a fully convolutional encoder and decoder hierarchy that lets it see multiple scales of the noisy input. Instead of directly predicting an output color, it outputs a set of weights for filter *kernels*, which are applied at three spatial scales on the noisy HDR input image: at full resolution, 2 \times , and 4 \times downsampled versions. The hierarchical kernel prediction approach has previously been shown to be more robust against color shifts, converges faster, and can be implemented efficiently for interactive denoising [Hasselgren et al. 2020; Vogels et al. 2018].

We also experimented with separating color samples into a *primary* and *secondary* buffer, similar to how shading is sometimes split into direct and indirect components. In this case, we used a heuristic to guide direct, high-frequency, shading into the primary buffer and indirect, low-frequency, shading into the secondary buffer. We then predicted unique kernels for each of these two components. Unfortunately, we observed only minor quality improvements of this separation, so could not motivate the added cost of two kernels in our use case. Please refer to the accompanying video for a visualization of these buffers.

We optionally augment the system with a learned adaptive sampler. This network component is illustrated in Figure 7 (left). We train the adaptive sampler and denoiser networks jointly, letting the two networks work in unison together with adaptive volume path tracing. The end result is more rays guided towards high variance regions in the volumes, which improves image quality at a given sample budget. Please refer to Hasselgren et al. [2020] for a detailed description of this sampling and denoising architecture.

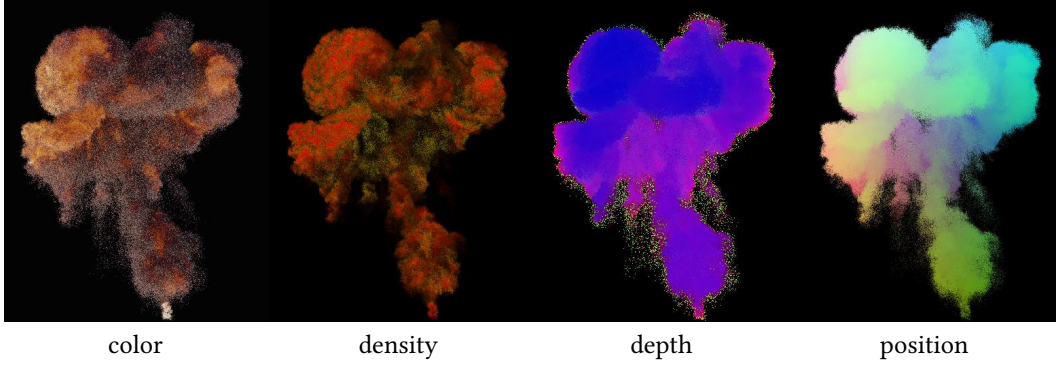


Fig. 8. Illustration of our full input vector for denoising at 4 spp uniform, with features gathered at the primary scattering position. From left to right: color buffer (rgb), mean (r) and standard deviation (g) of density, mean (r) and standard deviation (g) of depth including the fraction of rays that scattered (b), and object space position (rgb).

3.2 Training and Dataset

We train the denoiser using animated clips of eight frames. We generated 1000 clips from a set of volumes¹ at input sample counts in the range of 1–32 spp, and trained against corresponding reference renderings with 4096 spp. An excerpt from the training data is shown in Figure 9.

In each clip, we light the volume from a randomly selected probe from a set of 40 HDR probes from HDRIHaven [Haven 2020], randomize intensity, and use random camera positions, targets and focal lengths. We also randomize a multitude of parameters of the volumes: volumetric density, albedo, anisotropy and emissive strength, in order to ensure a high-quality dataset. We train our networks for 1000 epochs, unless otherwise noted, and in each epoch and training step, we use random crops of size 256×256 pixels and augment with random 90-degree rotations and xy-flips. Our dataset consists mostly of animated volumes, but we purposely also include a fraction ($\frac{1}{14}$) of static frames in the training set to make the network more robust for varying degrees of motion. Our networks train in roughly 24 hours on a single NVIDIA V100 GPU.

Objective Function. We train our networks using *symmetric mean absolute percentage error* (SMAPE) [Vogels et al. 2018] on linear HDR color values. Given a reference \mathbf{r} and denoised estimate \mathbf{d} :

$$\text{SMAPE}(\mathbf{r}, \mathbf{d}) = \frac{1}{3N} \sum_{p \in N} \sum_{c \in C} \frac{|\mathbf{d}_{p,c} - \mathbf{r}_{p,c}|}{|\mathbf{d}_{p,c}| + |\mathbf{r}_{p,c}| + \epsilon}. \quad (5)$$

Here, N is the number of pixels in the image, and C are the three color channels. We use $\epsilon = 0.01$. To improve temporal stability, we include a loss term with SMAPE computed on the temporal differences Δ_r and Δ_d of the last and second last timesteps:

$$\text{tSMAPE}(\Delta_r, \Delta_d, \mathbf{r}, \mathbf{d}) = \frac{1}{3N} \sum_{p \in N} \sum_{c \in C} \frac{|\Delta_{\mathbf{d}_{p,c}} - \Delta_{\mathbf{r}_{p,c}}|}{|\mathbf{d}_{p,c}| + |\mathbf{r}_{p,c}| + \epsilon}. \quad (6)$$

Our combined objective function is $1.0 \cdot \text{SMAPE} + 0.3 \cdot \text{tSMAPE}$.

We also experimented with an adversarial loss, following Hofmann et al. [2020]. Please refer to Section 4.2 for detailed evaluations.

¹Sources: TurboSquid [TurboSquid 2020], Disney Animation [DisneyAnimation 2020] and JangaFX [JangaFX 2020]

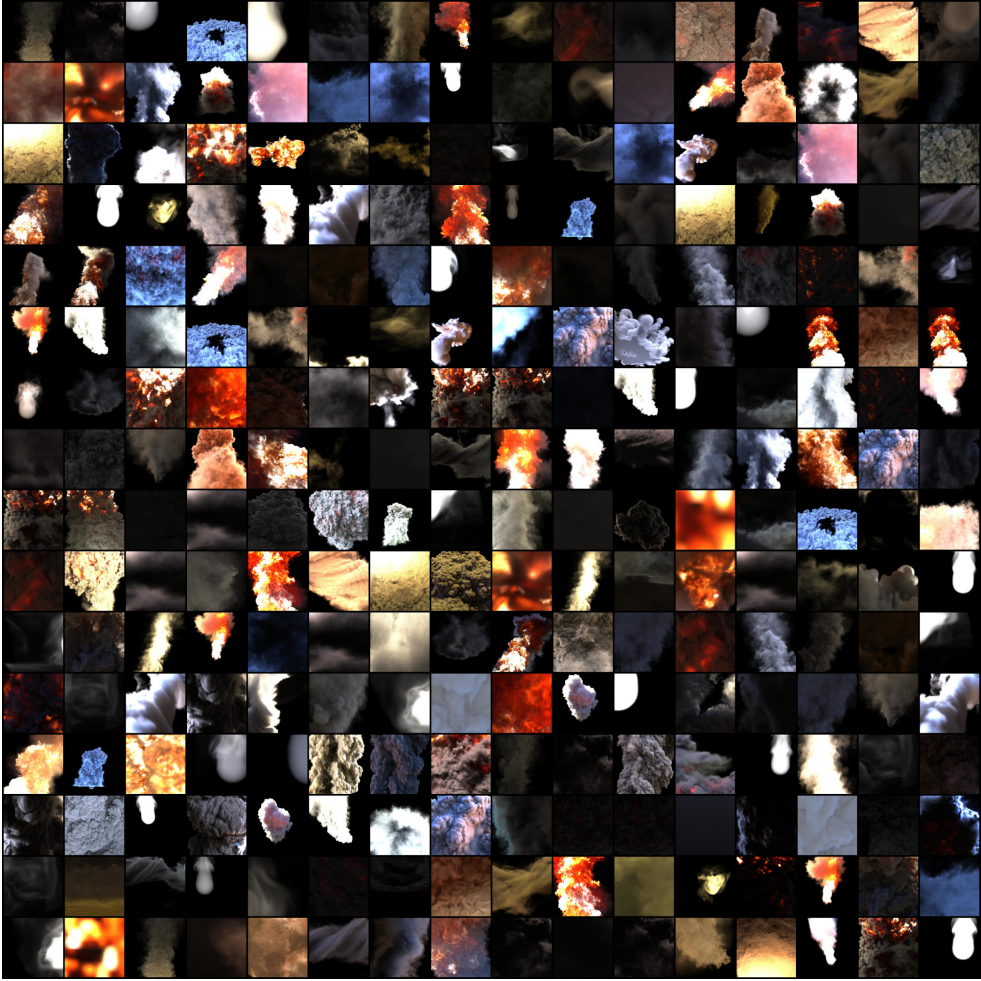


Fig. 9. Excerpt of 256 non-augmented frames randomly selected from our volumetric denoising training dataset.

4 EVALUATION

In the following, we provide an extensive survey of both interactive rendering performance and the effects of multiple variations regarding denoising quality in an ablation study.

4.1 Runtime Performance

Performance Breakdown. We implemented the adaptive sampler, renderer and denoiser in the Falcor [Benty et al. 2020] real-time rendering framework. We report runtime performance on a set of six different volumes in Table 1. All measurements were performed on a NVIDIA RTX 3090 at 1920×1080 , and we render and reconstruct all six examples at interactive rates. Note that performance depends on the scene and the number of pixels covered by the volume. Subsequently, we evaluate the performance impact of our optimizations and the influence of volume parameters.

4 spp uniform						
Scene	#Voxels	#Paths	Render	Denoise	Σ	FPS
DUST DEVIL	2.5M	1.4M	8.6	7.3	15.9	63
GASOLINE EXPLOSION	3.6M	1.6M	8.9	7.3	16.2	61
BUILDING IMPLOSION	4.0M	2.8M	8.4	7.3	15.7	64
MIDAIR EXPLOSION	5.4M	2.7M	16.5	7.3	23.8	42
SMOKE PLUME	11.4M	1.6M	20.1	7.3	27.4	36
DISNEY CLOUD	188.4M	5.3M	53.4	7.3	60.7	16

4 spp adaptive						
Scene	#Voxels	#Paths	Render	Denoise	Σ	FPS
DUST DEVIL	2.5M	5.3M	27.8	7.3 + 3.1	38.2	26
GASOLINE EXPLOSION	3.6M	6.1M	31.4	7.3 + 3.1	41.8	24
BUILDING IMPLOSION	4.0M	8.3M	22.6	7.3 + 3.1	33.0	30
MIDAIR EXPLOSION	5.4M	8.3M	50.1	7.3 + 3.1	60.5	17
SMOKE PLUME	11.4M	6.1M	79.5	7.3 + 3.1	89.9	11
DISNEY CLOUD	188.4M	8.5M	86.4	7.3 + 3.1	96.8	10

Table 1. Runtime performance (ms) at 4 spp (max 128 bounces) at a resolution of 1920×1080 pixels on a NVIDIA RTX 3090 GPU. In the total time (Σ), we include our neural denoising and adaptive sampling networks, which run at constant times of 7.3ms and 3.1ms per frame, respectively. Adaptive sampling provides a significant quality boost at equal sample count, but comes at increased runtime cost, as more paths will be traced into the volume.

Optimizations. We evaluate the effect of continuously enabling different components in our renderer from Section 2 on the DISNEY CLOUD scene, as displayed in Figure 1, starting from a simple path tracer using trilinear interpolation, but without next-event estimation. Enabling next-event estimation (NEE) roughly doubles the runtime cost, due to additional shadow rays, but leads to significantly increased convergence rates, so we consider it critical. Performance numbers, indicating an overall speedup of over $5\times$ compared to a simple path tracer with NEE and trilinear interpolation, are shown in the table below.

DISNEY CLOUD, 4 spp (ms)	On	Off	Speedup
Next-event estimation	278.0	150.8	$\times 0.54$
Stochastic interpolation	87.2	278.0	$\times 3.19$
Load balancing	63.2	87.2	$\times 1.38$
Empty space skipping	53.4	63.2	$\times 1.18$

Volume Parameters. The volume parameters play a critical role in runtime performance, due to causing a significant variation in the dimensionality of the underlying integrand. For example, when rendering with high density and albedo, rays will scatter often and without losing much energy, resulting in highly divergent and very long paths, where Russian roulette is unable to efficiently cull paths. In contrast, when rendering a thin smoke cloud with low albedo, paths will scatter less often and lose energy quickly, resulting in a more coherent workload and shorter paths overall. We provide a small parameter sweep over what we consider to be the parameters with the most notable effect on runtime in the table below, i.e., volume albedo, density, and maximum number of bounces per path. Note how load balancing in combination with Russian roulette results in the same performance when limiting the number of bounces per path to 16 or 128.

Volume albedo	0.25	0.5	0.75	1.0
4 spp uniform (ms)	20.7	34.6	53.4	115.1
Density scale	0.1	0.25	0.5	1.0
4 spp uniform (ms)	24.7	39.9	62.6	105.1
Max bounces	1	4	16	128
4 spp uniform (ms)	19.6	38.3	53.4	53.4

4.2 Denoiser Ablation Studies

In this section, we summarize our findings in trying to fine tune the denoiser architecture for sparse volumes. To evaluate our trained networks, we use four animated volumes with 100 frames each. Please refer to the accompanying video to see the full sequences. We show visual quality results in Figure 10 and a larger set of comparisons in the supplemental material. We report errors using three commonly used error metrics, relative mean-square error (relMSE), SMAPE (Eq. 5), and PSNR. We used a weighted combination of SMAPE and temporal SMAPE (Eq. 6) as objective function during training.

Feature Guides. We observe that the additional feature guides (Figure 8) consistently improve quality, but the improvements are quite modest, as can be seen in the table below.

Influence of guides	relMSE	SMAPE	PSNR
Color	0.0199	0.0238	35.37
Color + guides	0.0185	0.0234	35.48

Network Size. To study denoising quality as function of network size, we created network variants with twice and four times the number of trainable weights, but noted only a small increase in quality. The improvements are slightly higher with feature guides enabled.

Network size	relMSE	SMAPE	PSNR
Color	0.0199	0.0238	35.37
Color 2x weights	0.0185	0.0236	35.47
Color 4x weights	0.0198	0.0238	35.51
Color + guides	0.0185	0.0234	35.48
Color + guides 2x weights	0.0175	0.0228	35.73
Color + guides 4x weights	0.0211	0.0230	35.75

For a real-time scenario, we conclude that our baseline small network is a good compromise of speed vs. quality. Reducing the size further was not beneficial from a runtime performance perspective, as the optimized GPU convolution kernels we apply at inference-time run at equal speed with 16 and 32 features.

Adaptive Sampling. To train joint adaptive sampling and denoising, we need to provide training data at sample rate. Therefore, the evaluation in the table below is trained on a different dataset (500 clips with per-sample data), and the test set is similarly slightly different (same four animations, but with input provided at sample rate). We note that adaptive sampling provides a significant quality boost by focusing more samples into high variance regions of the volume. Our adaptive variants have roughly equal quality of twice the number of samples per pixel of a uniform rendering. We also compare our data-driven approach to a naive implementation of adaptive sampling, which simply allocates one sample per pixel for background pixels and the desired number of samples for non-background pixels, which is then normalized to fit the given sample budget.

Uniform vs. Adaptive	relMSE	SMAPE	PSNR
Uniform 4 spp	0.0186	0.0229	35.28
Uniform 8 spp	0.0123	0.0194	36.75
Adaptive (naive) 2 spp	0.0265	0.0253	35.13
Adaptive (naive) 4 spp	0.0167	0.0201	37.26
Adaptive (ours) 2 spp	0.0180	0.0226	35.68
Adaptive (ours) 4 spp	0.0115	0.0188	37.27

However, this also comes at an increased runtime cost, as more paths will be traced into the more complex parts of the volume, where traversal cost is higher.

Adversarial Loss. We implemented the adversarial loss function described in the DCGAN paper [Radford et al. 2015], and trained the discriminator alongside our network weights. While improvements are quite modest in terms of error metrics, we observed improved sharpness and details in the denoised images. Adversarial losses only incur a performance cost during training and do not affect inference, so we recommend using it as a free additional improvement.

Adversarial loss	relMSE	SMAPE	PSNR
Color	0.0199	0.0238	35.37
Color + GAN	0.0189	0.0236	35.43

Motion Vectors. Our target is dynamic volumes, so if we have motion vectors (or velocity information) available, we can reproject the denoised result from the previous frame to align with the current timestep. This approach is commonly used in denoisers for surface path tracers, where accurate motion vectors are readily available. Unfortunately, for volumes, the motion information is much noisier, and we found it very hard to find publicly available volume assets containing motion data. Therefore, our current evaluation of the influence of motion vectors is limited to a single animation, and is only a first indication of the quality impact. When training a denoiser for this single smoke animation consisting of 200 distinct frames, we did not see a clear benefit from reprojecting data, neither when using the raw motion vectors nor motion vectors pre-filtered by another small neural network. We speculate that the dataset is too small, so the network overfits to the current animation. We leave a larger evaluation of temporal reprojection for future work.

Motion data	relMSE	SMAPE	PSNR
Without motion	0.0030	0.0130	42.57
With motion	0.0038	0.0133	42.35
With filtered motion	0.0039	0.0127	42.55

Generalization. During training data generation, we randomize a multitude of parameters, i.e., scene and animation frame, light probe and intensity, camera position, target and focal length, volumetric density, albedo, anisotropy, and emissive strength, in order to ensure a high quality dataset. Along with the indirect prediction of filter kernels and applying data augmentations during training, we observe robust ability to generalize onto unknown data. To verify, we evaluate two identical configurations of our neural denoising network, where the first one was solely trained on the evaluation sequence (*Overfit*) and the second one on all available data except the evaluation sequence (*Holdout*):

Generalization	relMSE	SMAPE	PSNR
Overfit	0.0030	0.0130	42.57
Holdout	0.0042	0.0148	41.42

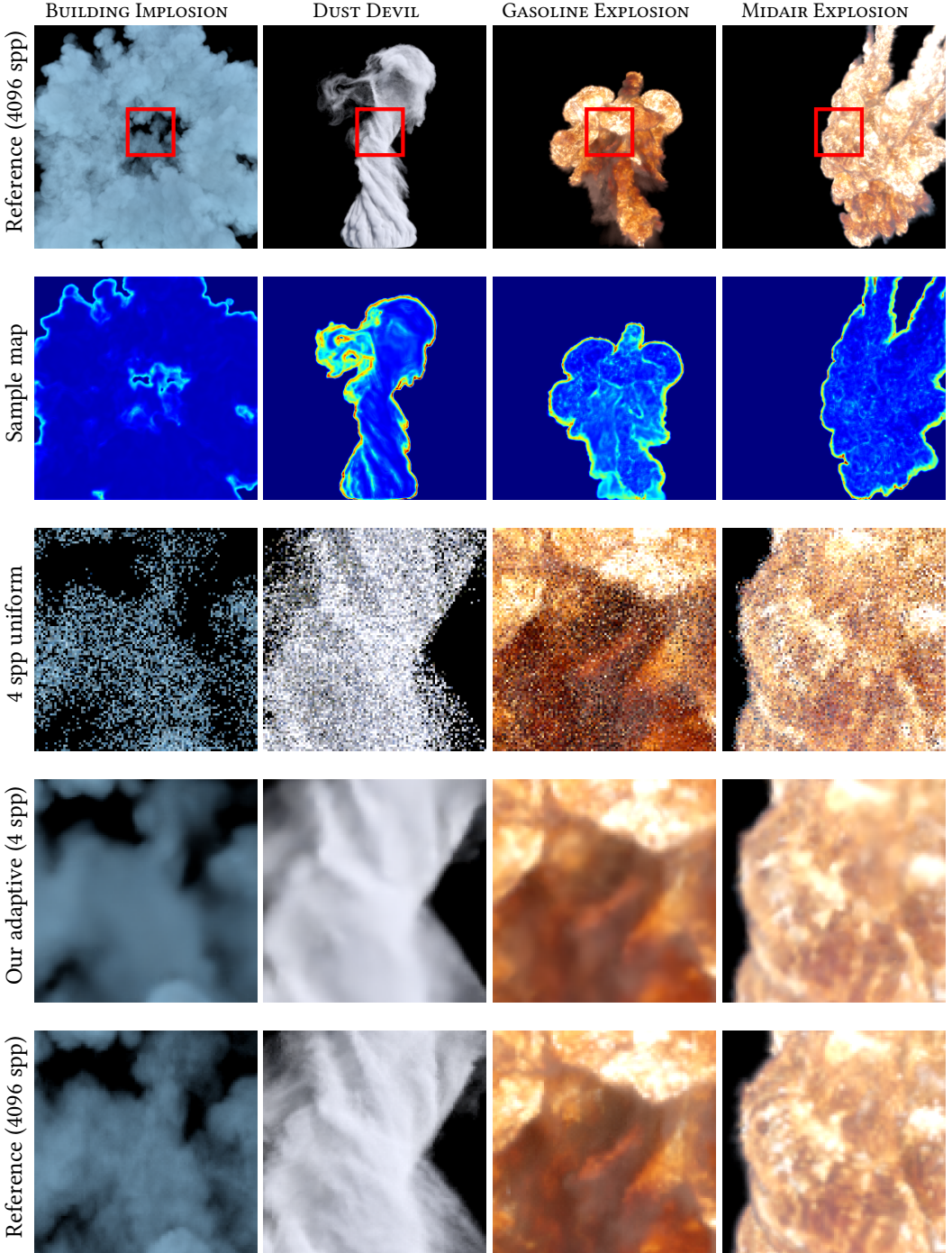


Fig. 10. Denoising quality from an average of 4 spp on four frames from our test set. Please refer to the accompanying video for quality in motion and evaluation of temporal stability, and the supplemental material for full-resolution images including pre-denoised output with both uniform and adaptive sampling.

5 CONCLUSIONS AND FUTURE WORK

We have shown that unbiased interactive volume rendering is feasible on modern GPUs, and for simpler volumes we achieve real-time performance. We limited the evaluation to single volumes and in future work, we want to integrate our volume renderer into a full path tracer to allow for light interactions between volumes and triangle meshes. We consider our renderer to be easily extendable to handle surface interactions as well, although that would require retraining the denoising networks and generating new training data, which comes at high computational costs. Rendering multiple non-overlapping volumes is trivially possible, although correctly handling arbitrarily overlapping, or nested, volumes would require further care and adjustments to the renderer.

Since we found additional feature guides to only yield very modest improvements in denoising quality, we infer that classic edge-stoppers, as commonly used for denoising surface-only renderings, don't work as well in a volume. This is due to the noisy nature of the guides by virtue of stochastic volume sampling, see Figure 8, in comparison to surface-only rendering, where guides are virtually noise free. We'd thus like to encourage future work on experimenting with different features, or encoding of the guide data, to more effectively guide the denoising process.

While our current performance is encouraging, real-time rendering would still require improvements, especially in the larger volumes with high density and albedo. There is potential for future work in improving the ray traversals, taking advantage of the hierarchical volume data-structures, and perhaps applying level-of-detail for higher order bounces. However, the biggest opportunity seems to be better utilization of existing samples. In future work, we would like to explore volume motion flow in further detail, or additional physical properties derived from the underlying simulation, which could potentially increase the effective sample count through temporal reprojection.

ACKNOWLEDGMENTS

We want to acknowledge Jan Novák, Ken Museth and Nathan Morrical for their constructive input on the paper. Additionally, we thank the reviewers, Alex Evans, Tomas Akenine-Moller, Matt Pharr and Pontus Andersson for their insightful and helpful feedback on the draft, and Aaron Lefohn for supporting this research. This research was conducted at NVIDIA's real-time rendering research group.

REFERENCES

- Timo Aila and Samuli Laine. 2009. Understanding the efficiency of ray traversal on GPUs. In *Proceedings of the conference on High Performance Graphics 2009*. 145–149.
- Steve Bako, Thijs Vogels, Brian McWilliams, Mark Meyer, Jan Novák, Alex Harvill, Pradeep Sen, Tony Deroose, and Fabrice Rousselle. 2017. Kernel-predicting convolutional networks for denoising Monte Carlo renderings. *ACM Trans. Graph.* 36, 4 (2017), 97–1.
- Fabian Bauer. 2019. Creating the Atmospheric World of Red Dead Redemption 2: A Complete and Integrated Solution. In *ACM SIGGRAPH*.
- Nir Benty, Kai-Hwa Yao, Petrik Clarberg, Lucy Chen, Simon Kallweit, Tim Foley, Matthew Oakes, Conor Lavelle, and Chris Wyman. 2020. The Falcor Rendering Framework. <https://github.com/NVIDIAGameWorks/Falcor>
- Chakravarty R Alla Chaitanya, Anton S Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. 2017. Interactive reconstruction of Monte Carlo image sequences using a recurrent denoising autoencoder. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1–12.
- S Chandrasekar. 1960. Radiative Transfer Dover Publications. *New York* (1960).
- Jiawen Chen, Ilya Baran, Frédo Durand, and Wojciech Jarosz. 2011. Real-time volumetric shadows using 1D min-max mipmaps. In *Symposium on Interactive 3D Graphics and Games*. 39–46.
- Petrik Clarberg, Wojciech Jarosz, Tomas Akenine-Möller, and Henrik Wann Jensen. 2005. Wavelet importance sampling: efficiently evaluating products of complex functions. In *ACM SIGGRAPH 2005 Papers*. 1166–1175.

- Cyril Delalandre, Pascal Gautron, Jean-Eudes Marvie, and Guillaume François. 2011. Transmittance Function Mapping. In *Symposium on Interactive 3D Graphics and Games* (San Francisco, California) (I3D '11). Association for Computing Machinery, New York, NY, USA, 31–38. <https://doi.org/10.1145/1944745.1944751>
- Robert H Dicke, P James E Peebles, Peter G Roll, and David T Wilkinson. 1965. Cosmic Black-Body Radiation. *The Astrophysical Journal* 142 (1965), 414–419.
- DisneyAnimation. 2020. A large and highly detailed volumetric cloud data set. <https://www.disneyanimation.com/datasets/?drawer=/resources/clouds/> Accessed: 2020-12-10.
- Thomas Engelhardt and Carsten Dachsbacher. 2010. Epipolar sampling for shadows and crepuscular rays in participating media with single scattering. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*. 119–125.
- Manfred Ernst, Marc Stamminger, and Gunther Greiner. 2006. Filter importance sampling. In *2006 IEEE Symposium on Interactive Ray Tracing*. IEEE, 125–132.
- Mathieu Galtier, Stephane Blanco, Cyril Caliot, Christophe Coustet, Jérémie Dauchet, Mouna El Hafi, Vincent Eymet, Richard Fournier, Jacques Gautrais, Anais Khuong, et al. 2013. Integral formulation of null-collision Monte Carlo algorithms. *Journal of Quantitative Spectroscopy and Radiative Transfer* 125 (2013), 57–68.
- Pascal Gautron, Cyril Delalandre, and Jean-Eudes Marvie. 2011. Extinction transmittance maps. In *SIGGRAPH Asia 2011 Sketches*. 1–2.
- Iliyan Georgiev, Thiago Ize, Mike Farnsworth, Ramón Montoya-Vozmediano, Alan King, Brecht Van Lommel, Angel Jimenez, Oscar Anson, Shinji Ogaki, Eric Johnston, et al. 2018. Arnold: A brute-force production path tracer. *ACM Transactions on Graphics (TOG)* 37, 3 (2018), 1–12.
- J Hasselgren, J Munkberg, M Salvi, A Patney, and A Lefohn. 2020. Neural Temporal Adaptive Sampling and Denoising. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 147–155.
- HDRI Haven. 2020. Free HDRIs for Everyone. <https://hdrihaven.com/> Accessed: 2020-12-10.
- Louis G Henyey and Jesse L Greenstein. 1941. Diffuse radiation in the galaxy. *The Astrophysical Journal* 93 (1941), 70–83.
- Sébastien Hillaire. 2015. Physically Based and Unified Volumetric Rendering in Frostbite. *SIGGRAPH Advances in Real-Time Rendering course* (2015), 570–610.
- Nikolai Hofmann, Jana Martschinke, Klaus Engel, and Marc Stamminger. 2020. Neural Denoising for Path Tracing of Medical Volumetric Data. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 3, 2 (2020), 1–18.
- JangaFX. 2020. Download Free VDB Animations: Explosions, Fire, And More! <https://jangafx.com/software/embergen/download/free-vdb-animations/> Accessed: 2020-12-10.
- James T Kajiya. 1986. The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*. 143–150.
- Simon Kallweit, Thomas Müller, Brian McWilliams, Markus Gross, and Jan Novák. 2017. Deep scattering: Rendering atmospheric clouds with radiance-predicting neural networks. *ACM Transactions on Graphics (TOG)* 36, 6 (2017), 1–11.
- B Karis. 2014. High quality temporal anti-aliasing. *Advances in Real-Time Rendering for Games, SIGGRAPH Courses* (2014).
- Peter Kutz, Ralf Habel, Yining Karl Li, and Jan Novák. 2017. Spectral and decomposition tracking for rendering heterogeneous volumes. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1–16.
- Alexandr Kuznetsov, Nima Khademi Kalantari, and Ravi Ramamoorthi. 2018. Deep Adaptive Sampling for Low Sample Count Rendering. *Computer Graphics Forum* 37 (2018), 35–44.
- Sébastien Lagarde and Evgenii Golubev. 2018. The Road toward Unified Rendering with Unity’s High Definition Render Pipeline. In *ACM SIGGRAPH*.
- Johann Heinrich Lambert. 1760. *Photometria sive de mensura et gradibus luminis, colorum et umbrae*. Klett.
- Xiaoxu Meng, Quan Zheng, Amitabh Varshney, Gurprit Singh, and Matthias Zwicker. 2020. Real-time Monte Carlo Denoising with the Neural Bilateral Grid. In *Eurographics Symposium on Rendering - DL-only Track*, Carsten Dachsbacher and Matt Pharr (Eds.).
- Ken Museth. 2013. VDB: High-resolution sparse volumes with dynamic topology. *ACM transactions on graphics (TOG)* 32, 3 (2013), 1–22.
- Ken Museth. 2014. Hierarchical digital differential analyzer for efficient ray-marching in OpenVDB. In *ACM SIGGRAPH 2014 Talks*. 1–1.
- Ken Museth. 2021. NanoVDB: A GPU-Friendly and Portable VDB Data Structure For Real-Time Rendering And Simulation. (2021). In submission.
- Ken Museth, Jeff Lait, John Johanson, Jeff Budsberg, Ron Henderson, Mihai Alden, Peter Cucka, David Hill, and Andrew Pearce. 2013. OpenVDB: an open-source data structure and toolkit for high-resolution volumes. In *Acm siggraph 2013 courses*. 1–1.
- Jan Novák, Iliyan Georgiev, Johannes Hanika, and Wojciech Jarosz. 2018. Monte Carlo methods for volumetric light transport simulation. In *Computer Graphics Forum*, Vol. 37. Wiley Online Library, 551–576.

- Jan Novák, Andrew Selle, and Wojciech Jarosz. 2014. Residual ratio tracking for estimating attenuation in participating media. *ACM Trans. Graph.* 33, 6 (2014), 179–1.
- Ken Perlin and Eric M Hoffert. 1989. Hypertexture. In *Proceedings of the 16th Annual Conference on Computer graphics and interactive techniques*. 253–262.
- Matthias Raab, Daniel Seibert, and Alexander Keller. 2008. Unbiased global illumination with participating media. In *Monte Carlo and Quasi-Monte Carlo Methods 2006*. Springer, 591–605.
- Alec Radford, Luke Metz, and Soumith Chintala. 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434* (2015).
- Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*. Springer, 234–241.
- Florian Simon, Johannes Hanika, Tobias Zirr, and Carsten Dachsbacher. 2017. Line integration for rendering heterogeneous emissive volumes. In *Computer Graphics Forum*, Vol. 36. Wiley Online Library, 101–110.
- TurboSquid. 2020. *3D Models for Professionals*. <https://www.turbosquid.com/>. Accessed: 2020-12-10.
- Eric Veach and Leonidas J Guibas. 1995. Optimally combining sampling techniques for Monte Carlo rendering. In *Proceedings of the 22nd Annual Conference on Computer graphics and interactive techniques*. 419–428.
- Ryusuke Villemin, Christophe Hery, and Pixar Animation Studios. 2013. Practical illumination from flames. *Journal of Computer Graphics Techniques* 2, 2 (2013).
- Thijs Vogels, Fabrice Rousselle, Brian McWilliams, Gerhard Röthlin, Alex Harvill, David Adler, Mark Meyer, and Jan Novák. 2018. Denoising with kernel prediction and asymmetric loss functions. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–15.
- Nathan Vos. 2014. Volumetric light effects in killzone: Shadow fall. *GPU Pro* 5 (2014), 127–147.
- Lance Williams. 1978. Casting curved shadows on curved surfaces. In *Proceedings of the 5th annual conference on Computer graphics and interactive techniques*. 270–274.
- 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.
- Bartłomiej Wronski. 2014. Volumetric fog: Unified compute shader based solution to atmospheric scattering. In *ACM SIGGRAPH*.
- Chris Wyman and Shaun Ramsey. 2008. Interactive volumetric shadows in participating media with single-scattering. In *2008 IEEE Symposium on Interactive Ray Tracing*. IEEE, 87–92.
- Matthias Zwicker, Wojciech Jarosz, Jaakko Lehtinen, Bochang Moon, Ravi Ramamoorthi, Fabrice Rousselle, Pradeep Sen, Cyril Soler, and S-E Yoon. 2015. Recent advances in adaptive sampling and reconstruction for Monte Carlo rendering. In *Computer Graphics Forum*, Vol. 34. Wiley Online Library, 667–681.