SGGF: SIMULATION-GUIDED GAUSSIAN FIELD FOR RELIGHTABLE VOLUMETRIC EFFECTS

Anonymous authors
Paper under double-blind review

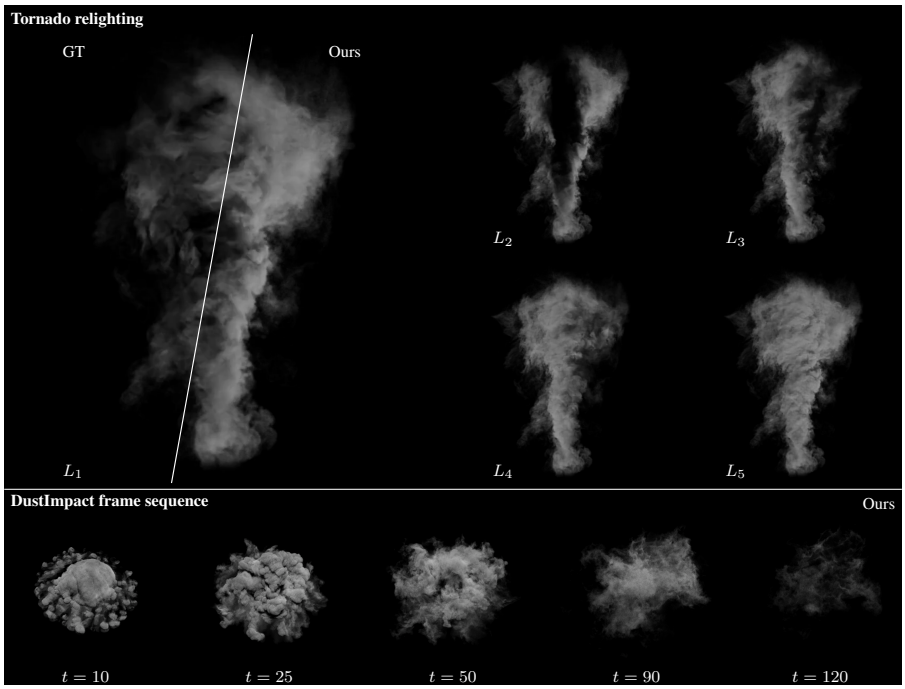

Figure 1: **Relighting and animation.** Top: Tornado under five lights, with a GT/ours comparison at left. Bottom: DustImpact over time at fixed camera and light, translated for display centering at a common scale.

ABSTRACT

Recent advances in agent-assisted workflows are making it increasingly practical to create complex 3D content with substantially less manual scene construction. When coupled with mature physics-based simulators, they provide a promising workflow for creating rich dynamic volumetric effects. Yet the resulting simulated volumes remain expensive to render, especially when repeatedly viewed from different cameras or under changing illumination. We present SGGF, a simulation-guided Gaussian field for efficient rendering and relighting of dynamic volumetric scenes. Trained per sequence from density fields and offline reference images, a shared spatiotemporal field generates Gaussian attributes without explicit cross-frame primitive correspondences. We use simulated density to guide opacity learning and visibility computed from the Gaussians to guide shading under changing illumination. To improve training efficiency, we begin with sparse samples and progressively increase their density, adjusting Gaussian sizes and opacity along the way. Experiments on public and agent-authored volumetric effect scenes evaluate rendering quality and the quality–efficiency tradeoff across sampling budgets. Under a GPU-resident configuration, SGGF renders 128 consecutive frames at 22.40 fps, evaluating the shared field and performing shading and rasterization at each frame.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

1 INTRODUCTION

Agent-assisted content creation connects textual intent to executable scene construction through scripts and tool calls (MCP for Blender Contributors, 2026). For dynamic volumetric effects, this workflow can use simulators to produce evolving three-dimensional structures from emission sources and simulation controls. However, generating an asset does not make its appearance inexpensive to render: attenuation and scattering must still be evaluated throughout the volume (Kajiya & von Herzen, 1984; Lafortune & Willems, 1996). Supporting subsequent playback, viewpoint changes, and lighting adjustment therefore calls for an efficient, relightable representation of the generated content.

We target this rendering stage of agent-assisted volumetric content creation. Simulation tools such as Houdini and EmberGen provide time-varying density fields in VDB sequences (JangaFX, n.d.; SideFX, n.d.). Rather than recovering their evolution from images, we use this known structure to learn rendering attributes from density fields and reference images rendered offline. Reference generation and training form a sequence-specific preparation stage; the fitted representation then supports fast rendering and relighting of the asset.

Existing neural methods accelerate scattering estimation or replace volume integration with learned shading. Yet sampling-based approaches retain per-pixel integration costs (Hu et al., 2023), while rasterized alternatives rely on static-asset formulations or approximate shading (Olajos et al., 2026; Li et al., 2026). These trade-offs motivate a representation that combines efficient image formation with known density evolution and light-dependent visibility.

Gaussian splatting offers efficient image formation (Kerbl et al., 2023), while representative dynamic Gaussian methods focus on reconstructing scenes from images through persistent primitives or canonical deformation (Luiten et al., 2024; Yang et al., 2024b; Wu et al., 2024). In simulator-driven content creation, the density sequence already describes structural changes such as expansion, fragmentation, and dissipation. This known evolution supplies frame-specific spatial support, allowing us to learn rendering attributes without recovering motion through cross-frame primitive correspondences. This suggests separating the changing support from the shared parameters that predict rendering attributes.

The remaining challenge is to connect that support to volumetric appearance. Known density constrains attenuation, but does not directly determine the opacity of discrete Gaussian kernels. Likewise, local density and light direction do not specify how the surrounding medium occludes illumination. A useful representation must combine the supplied structure with learned corrections and nonlocal visibility information. This motivates using density to guide opacity and geometry-derived visibility to condition appearance, with both learned from reference images.

To support efficient rendering and relighting of dynamic volumetric assets created in agent-assisted workflows, we introduce *Simulation-Guided Gaussian Field* (SGGF). Current-frame volume samples query a shared multiresolution 4D hash grid that jointly encodes spatial coordinates and time. Lightweight decoder heads map the resulting features to Gaussian geometry, density-guided residual opacity, and appearance for rasterization. A light-space visibility cue conditions the appearance decoder for relighting (Bi et al., 2024). Because rendering samples are evaluations of shared parameters, their density can increase during training without introducing independent parameters for every primitive. We use this separation for progressive training, coordinating sample density, kernel scales, and opacity-prior coefficients.

Our contributions are as follows: (i) We propose a simulation-guided rendering framework for agent-assisted creation of dynamic volumetric content. Through sequence-specific learning from simulator-generated density sequences and offline reference images, the framework produces an asset representation that supports efficient animation rendering and relighting. (ii) We introduce a shared spatiotemporal Gaussian attribute field that uses simulator-provided density evolution to define each frame's spatial support, without requiring cross-frame primitive correspondences. Density-guided residual opacity and visibility-conditioned appearance map the supplied volumetric structure to Gaussian rendering attributes. (iii) We develop a progressive training strategy that jointly schedules sampling density, Gaussian kernel scales, and opacity-prior coefficients, exploiting the separation between shared field parameters and rendering samples to improve training efficiency.

2 Related work

Agent-assisted volumetric creation and rendering. Tool-using AI agents can streamline scene authoring by translating textual instructions into scripts and operations in content-creation software (MCP for Blender Contributors, 2026). Coupled with volumetric simulators such as Houdini Sparse Pyro (SideFX, n.d.), these workflows can produce dynamic assets whose evolving density structure is explicitly available. Rendering their appearance, however, still requires accounting for attenuation and scattering throughout the volume. Density-based volume ray tracing and participating-media path tracing provide a physical foundation for this computation (Kajiya & von Herzen, 1984; Lafortune & Willems, 1996), but high-quality light-transport estimation remains costly. This motivates rendering representations that exploit simulator-provided structure to support efficient playback and relighting after offline preparation.

Neural acceleration for volumetric rendering. Neural methods reduce rendering costs by approximating scattering or learning efficiently evaluable asset representations. Deep Scattering predicts scattered radiance from density descriptors (Kallweit et al., 2017). MRPNN combines density and transmittance features in a lightweight network, achieving real-time rendering in its reported settings while retaining delta tracking and per-pixel sample accumulation (Hu et al., 2023). RNA instead precomputes complex shading and scattering into a feature-grid and MLP representation of a relightable asset (Mullia et al., 2025). Environmental Volumetric Neural Shading of Clouds avoids volume integration using mesh geometry, triplane features, and thickness, but its evaluation focuses on a static cloud asset with a particular sky model (Olajos et al., 2026). Neural Six-way Lightmaps supports real-time volumetric animation through approximate lightmap-based shading, without guaranteeing high-fidelity agreement with volumetric path tracing (Li et al., 2026). SGGF targets evolving simulated assets, learning a shared Gaussian field from density sequences and offline reference images for efficient rendering and relighting.

Gaussian splatting and volumetric consistency. 3D Gaussian Splatting enables efficient radiance-field rendering through projected Gaussian primitives and alpha compositing (Kerbl et al., 2023). Its screen-space opacity formulation approximates volumetric image formation rather than exactly integrating a view-independent extinction field. Subsequent work analytically integrates Gaussian density to compute Beer–Lambert transmittance within a rasterizer (Talegaonkar et al., 2025). Celarek et al. (2025) examine these approximations and report additional costs for more accurate integration and self-attenuation. Don't Splat Your Gaussians reconstructs scattering media with volumetric kernels, accelerating voxel-based path tracing but retaining costly light-transport sampling (Condor et al., 2025). SGGF retains efficient splatting and learns density-guided opacity and appearance from reference images.

Dynamic neural fields and Gaussian reconstruction. Hash encodings combine grid features with compact decoders (Müller et al., 2022), while ResFields adds temporal capacity through residual network weights (Mihajlovic et al., 2024). Dynamic Gaussian representations include persistent primitives (Luiten et al., 2024), canonical deformation (Yang et al., 2024b; Wu et al., 2024; Xu et al., 2024), and joint four-dimensional primitives (Yang et al., 2024a). Swift4D separates static and dynamic components and uses a four-dimensional hash representation for deformation (Wu et al., 2025). These methods primarily reconstruct scenes from images. GauSmoke combines Eulerian density and velocity fields with Lagrangian Gaussians and advects their states for the volume reconstruction (Zhang et al., 2026). For simulator-generated assets, density evolution is already available. SGGF uses this structure to define sampling support and an opacity prior, while learning Gaussian attributes through a shared spatiotemporal field without requiring velocity estimation or state advection.

Neural relighting and visibility cues. NRHints conditions neural appearance on shadow and highlight hints (Zeng et al., 2023). RNG introduces neural Gaussian appearance and a shadow cue, together with depth refinement and a hybrid forward-deferred fitting strategy (Fan et al., 2025). GS^3 computes shadow cues through light-space Gaussian splatting and learned refinement (Bi et al., 2024). Building on light-space Gaussian splatting, we derive a visibility cue that conveys light-dependent occlusion to a shared spatiotemporal appearance decoder, helping it model shading, including shadows and self-shadowing, under changing illumination.

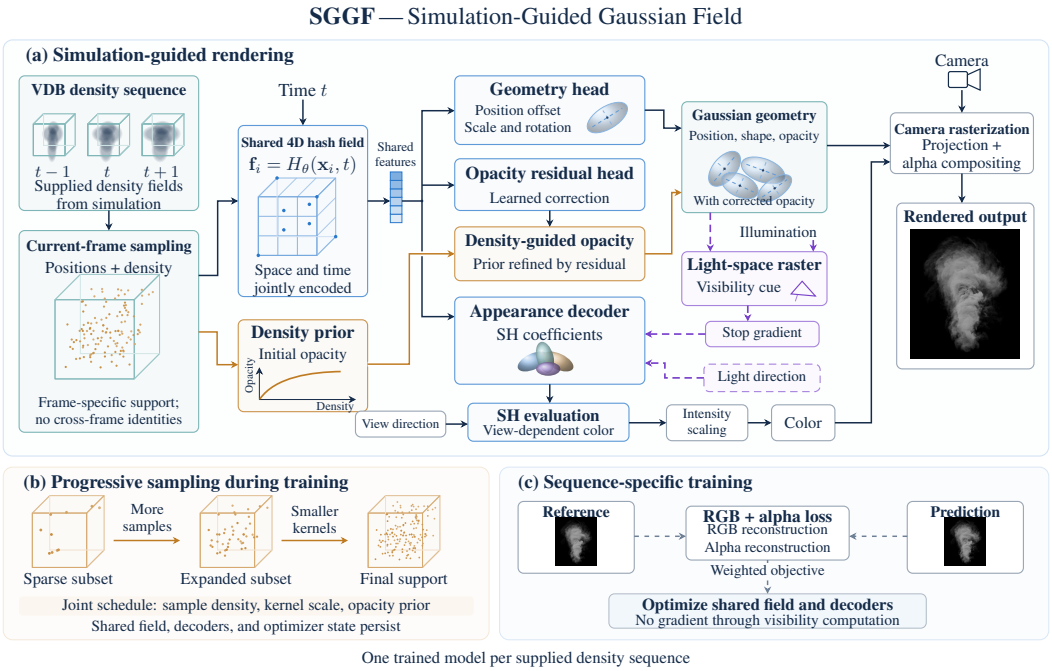

Figure 2: SGGF overview. (a) Density samples query a shared 4D hash field. Decoders predict Gaussian geometry, opacity residuals, and SH coefficients. A density prior guides opacity, while light-space rasterization supplies a detached visibility cue for appearance. SH evaluation and intensity scaling produce colors for camera rasterization. (b) Training expands nested samples with coordinated kernel scales and opacity-prior coefficients. (c) Reference RGB and alpha supervise the shared field and decoders per sequence. Purple dashed arrows indicate illumination conditioning; gray dashed arrows indicate supervision. Structural illustrations are schematic; displayed RGB images are actual Tornado renderings and the matched reference.

3 Method

3.1 Problem formulation and volume support

Given a sequence of density fields $D_t : \Omega \to \mathbb{R}_{\geq 0}$ over a spatial domain Ω, we seek a shared representation of its appearance across time, viewpoints, and illumination. The density evolution is supplied; the learned quantities are rendering attributes. At time t, let

$$\mathcal{P}_t = \{(\mathbf{x}_i^t, \rho_i^t)\}_{i=1}^{N_t}, \qquad \rho_i^t = D_t(\mathbf{x}_i^t), \tag{1}$$

denote samples from the current density support. Their positions anchor the rendering primitives, while their densities constrain opacity. Both the support and its cardinality may change with t; sample indices do not denote identities shared across frames. Thus, expansion and dissipation enter through the supplied density evolution rather than a learned trajectory of persistent primitives. A shared field maps these samples to Gaussian attributes, yielding $\mathcal{G}_t(\Lambda)$ under illumination Λ. The image at camera c is $\widehat{\mathbf{I}}_{t,c,\Lambda} = \mathcal{R}_c(\mathcal{G}_t(\Lambda))$, where $\mathcal{R}_c$ denotes Gaussian rendering. Each sequence is fitted to calibrated reference images. Figure 2 summarizes this construction.

3.2 Shared spatiotemporal Gaussian field

To share learnable parameters across the changing sample sets $\mathcal{P}_t$, we represent Gaussian attributes through a spatiotemporal feature field $H_\theta : \Omega \times \mathcal{T} \to \mathbb{R}^d$, where $\mathcal{T}$ is the sequence's time domain. Each sample queries this field at its spatial position and time, and shared decoders map the resulting features to Gaussian attributes. We parameterize H_θ by extending the multiresolution spatial hash encoding of Müller et al. (2022) to four dimensions, jointly encoding space and time at each

resolution level. The resulting feature is

$$\mathbf{f}_i^t = H_\theta(\mathbf{x}_i^t, t) = \bigoplus_{\ell=1}^{L} H_\theta^{(\ell)}(\mathbf{x}_i^t, t). \tag{2}$$

Shared decoders map $\mathbf{f}_i^t$ to a position offset $\Delta\mathbf{x}_i$, scale logits $\mathbf{a}_i$, rotation quaternion $\mathbf{q}_i$, and opacity residual r_i. Omitting the frame superscript, geometry is

$$\boldsymbol{\mu}_i = \mathbf{x}_i + \Delta\mathbf{x}_i, \qquad \mathbf{s}_i = K\ \mathrm{sigmoid}(g_s\mathbf{a}_i) + b, \tag{3}$$

$$\Sigma_i = R(\widehat{\mathbf{q}}_i)\,\mathrm{diag}(\mathbf{s}_i^2)R(\widehat{\mathbf{q}}_i)^\top, \tag{4}$$

where K controls kernel scale, g_s and b are the scale activation gain and bias, and $\widehat{\mathbf{q}}_i$ is normalized. Volume-derived position anchors and bounded Gaussian scales are intended to provide a structured starting point for early optimization. The field and decoders contain the trainable parameters; the Gaussians are their evaluations on $\mathcal{P}_t$. This separates the learned representation from its rendering discretization: changing the sample set changes the instantiated primitives without assigning new trainable parameters to them. Temporal sharing is therefore defined through H_θ, without requiring cross-frame sample correspondence.

3.3 Density-guided opacity

The supplied density constrains where attenuation should occur, but its conversion to Gaussian opacity depends on the rendering discretization. We factor opacity into a density-derived prior and a learned correction:

$$\tau_i^0 = \rho_i C/K^2, \qquad \alpha_i^0 = \mathrm{clip}(1-\exp(-\tau_i^0), \epsilon, 1-\epsilon), \tag{5}$$

$$\alpha_i = \mathrm{sigmoid}\left(\mathrm{logit}(\alpha_i^0) + g_\alpha r_i\right). \tag{6}$$

Here C controls the prior and g_α scales the residual. A zero residual recovers the prior, providing a density-informed starting estimate for opacity optimization. The residual branch learns corrections around this estimate, increasing or decreasing opacity as needed to fit the reference images. The corrected opacity is used in both camera and light-space rasterization. The coefficients K and C control discretization, rather than representing recovered physical extinction.

3.4 Visibility-conditioned appearance and relighting

Appearance depends on both local structure and attenuation through the surrounding volume. Conditioning only on position, time, and light direction leaves the decoder to infer this nonlocal relationship implicitly. We instead derive a scalar visibility cue from the current Gaussian geometry and opacity. Let $\mathcal{G}_t^{\mathrm{geom}} = \{(\boldsymbol{\mu}_i, \Sigma_i, \alpha_i)\}$ and let $\mathcal{V}_i$ denote the light-space visibility operator, evaluated by shadow splatting (Bi et al., 2024):

$$v_i = \mathcal{V}_i(\mathcal{G}_t^{\mathrm{geom}}, \Lambda), \qquad h_i = \mathrm{stopgrad}(v_i). \tag{7}$$

The detached scalar h_i serves as a visibility cue that exposes light-dependent occlusion to the appearance field. The decoder receives the shared feature, light direction $\boldsymbol{\ell}_i$, and visibility cue, and predicts spherical-harmonic (SH) coefficients:

$$\mathbf{b}_i = A_\psi([\mathbf{f}_i; \boldsymbol{\ell}_i; h_i]), \qquad \mathbf{c}_i = I\,\mathcal{S}(\mathbf{b}_i, \mathbf{d}_i). \tag{8}$$

Camera direction $\mathbf{d}_i$ enters SH evaluation $\mathcal{S}$, followed by intensity scaling I. The visibility cue conditions the decoder instead of directly multiplying its output, allowing local features and nonlocal attenuation cues to jointly determine appearance. At fixed t, changing Λ changes visibility and color while preserving $\mathcal{G}_t^{\mathrm{geom}}$. This factorization supports relighting without relearning the geometry for each light. Visibility conditioning approximates appearance; it is not a complete multiple-scattering solver.

3.5 Progressive sampling and optimization

Decoupling the shared field from its rendering samples allows us to progressively refine the rendering discretization during training. We observed that training initially captures coarse volumetric

structure before resolving fine details, echoing the low-frequency spectral bias of MLPs (Rahaman et al., 2019). Motivated by this observation, we adopt a sparse-to-dense sampling schedule with coordinated Gaussian kernel scales. The sparse initial stages reduce field-query and rasterization costs, improving training efficiency while fitting the overall appearance; subsequent densification supports the recovery of finer structures. Let S_e denote the sampling reduction factor (SR) at epoch e, and S_f its final value. Let $\mathcal{P}_t^{\text{final}}$ denote the position–density sample set at time t retained under the final budget S_f. A fixed selection function $u : \Omega \to [0, 1]$ defines nested subsets:

$$\mathcal{P}_t^{(e)} = \{(\mathbf{x}, \rho) \in \mathcal{P}_t^{\text{final}} : u(\mathbf{x}) \leq S_f/S_e\}. \tag{9}$$

Decreasing S_e expands the support while retaining the same learnable field and decoders. Because sample density affects coverage and accumulated opacity, we coordinate the kernel scale K_e and prior coefficient C_e during dense refinement. The subscript ref denotes a fixed reference sampling configuration: S_{ref} is its sampling reduction factor, and K_{ref} and C_{ref} are the corresponding kernel scale and prior coefficient. We set

$$r_e = \min(1, S_e/S_{\text{ref}}), \qquad K_e = K_{\text{ref}} r_e^{1/3}, \quad C_e = C_{\text{ref}} r_e. \tag{10}$$

Reference values remain fixed before schedule activation. This compensates approximately for changing point spacing and accumulated opacity; Appendix D gives the derivation. Evaluation uses $\mathcal{P}_t^{\text{final}}$.

Camera-space splatting with front-to-back alpha compositing (Kerbl et al., 2023) produces RGB $\widehat{\mathbf{I}}$ and alpha $\widehat{A}$. For reference RGB $\mathbf{I}$ and alpha A, we optimize a weighted appearance and opacity objective:

$$\mathcal{L} = \lambda_{\text{rgb}} \|\widehat{\mathbf{I}} - \mathbf{I}\|_1 + \lambda_{\text{ssim}}(1 - \text{SSIM}(\widehat{\mathbf{I}}, \mathbf{I})) + \lambda_\alpha \|\widehat{A} - A\|_1. \tag{11}$$

Gradients update the shared field and decoders through camera rendering. The visibility cue has no separate supervision, and gradients do not pass through its visibility computation.

4 Experiments

4.1 Datasets and reference rendering

Our five volumetric effects combine public assets with agent-assisted creation. Tornado, DustImpact, and DustShockwave use JangaFX's EmberGen VDB collection (JangaFX, n.d.). Codex with GPT-5.6 Sol constructs Mushroom and a volumetric Cornell-Box-like scene (CBox) in Houdini Sparse Pyro from text prompts (SideFX, n.d.). The exported density sequences and offline reference images train SGGF per asset. Creation prompts appear in Appendix H.

Training and test sets use disjoint camera viewpoints and both cover the complete temporal sequence. Reference images are rendered with Blender Cycles at 1024×1024, with 512 samples per pixel and denoising enabled. Sequence lengths, view counts, and scene-specific lighting configurations are summarized in Table 3 in Appendix A.

4.2 Experimental setup

Implementation details. VDB sequences (OpenVDB Contributors, 2026) are preprocessed into sparse position–density arrays in the cameras' world coordinate system and preloaded into GPU memory. Each training iteration selects a frame and sample subset without reloading or traversing VDB data. The 4D hash encoding interpolates the 16 vertices of each cell; shared MLPs decode attributes, with degree-2 SH for appearance. A deterministic coordinate hash implements $u(\mathbf{x})$, and optimizer state is retained across sampling stages. References are alpha-composited over black, with loss weights $(\lambda_{\text{rgb}}, \lambda_{\text{ssim}}, \lambda_\alpha) = (0.8, 0.2, 1)$.

Hardware and evaluation. Training and inference were performed on NVIDIA RTX A6000 GPUs. GPU timings include the complete forward rendering workload and rasterization, excluding model loading, CPU readback, and file output. We report PSNR and SSIM, with alpha metrics in the appendix. MRPNN uses the authors' rendering implementation (Hu et al., 2023); scene-specific configurations and metric conventions are detailed in Appendix A.

4.3 RENDERING QUALITY

We compare SGGF with MRPNN on eight sampled frames and three held-out views per scene, using identical camera and lighting settings and the same reference images. Table 1 reports PSNR and SSIM averaged over the resulting 24 images per scene. SGGF achieves broadly comparable rendering quality to MRPNN, with higher PSNR in some scenes. Additional metrics and scene-specific evaluation settings are provided in Appendices A.1 and B.1.

	PSNR (dB)↑		SSIM↑	
Scene	SGGF	MRPNN	SGGF	MRPNN
DustImpact	**41.124**	38.056	0.9661	**0.9714**
DustShockwave	35.515	**36.095**	0.9488	**0.9495**
Tornado	**35.870**	34.426	**0.9759**	0.9702

Table 1: Multi-frame quality: 24 images per scene. These quality-only results are separate from the sampling-sweep and dynamic-playback timings.

Figure 3 shows representative held-out views of three volumetric effects. Full error maps, detail crops, and temporal panels appear in Appendix F; the Grid4D comparison is in Appendix G.

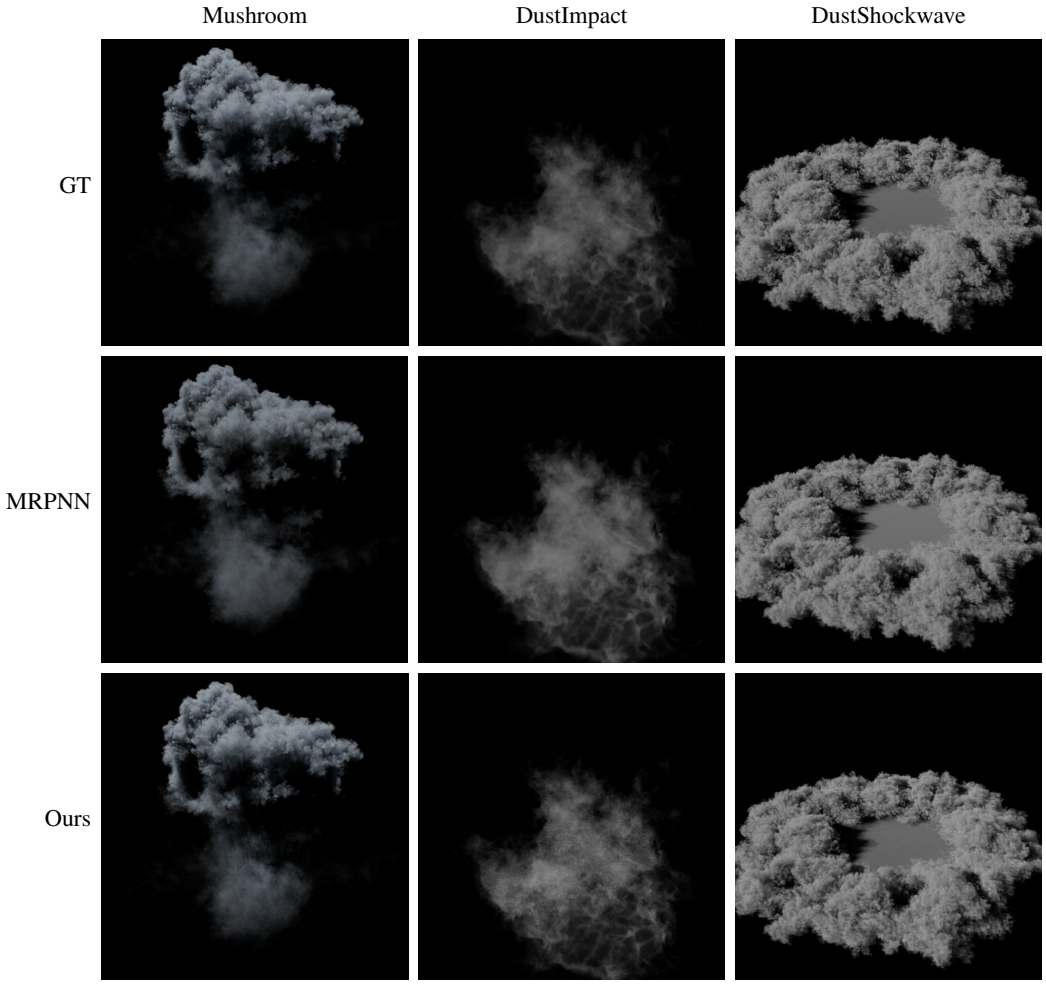

Figure 3: Qualitative comparison across three volumetric scenes. Rows: GT, MRPNN, and SGGF. Corresponding error maps and display conventions appear in Appendix F.

4.4 RENDERING EFFICIENCY

Figure 4 compares rendering quality and latency on a single frame at a fixed viewpoint. We vary the sampling reduction factor (SR) for SGGF, with smaller factors retaining more Gaussians, and the samples per pixel for MRPNN. At a sampling reduction factor of 2, SGGF achieves 34.866 dB PSNR in 28.60 ms, while MRPNN at 64 spp achieves 34.946 dB in 530.83 ms. At these comparable quality levels, SGGF provides an 18.56-fold speedup, with a PSNR difference of only 0.080 dB. Appendix B.2 provides the complete measurements and runtime breakdown.

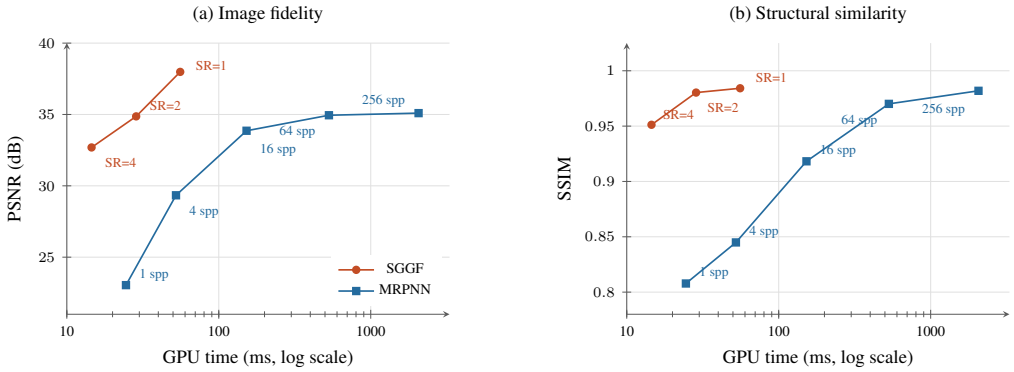

Figure 4: Quality–latency tradeoff for a single Tornado frame at a fixed viewpoint. For SGGF, SR denotes the sampling reduction factor: smaller values retain more Gaussians. For MRPNN, spp denotes samples per pixel. Lines connect measured configurations.

Dynamic playback. A separate mixed-precision configuration renders 128 consecutive Tornado frames at 1024×1024 with fixed camera and light, achieving **22.40 fps**. Median/P95 wall latency is 43.98/47.98 ms, with a sampled device-memory peak of 30.13 GiB. All per-frame input points are GPU-resident; field queries, visibility, and rasterization run each frame. Preloading, CPU readback, and display are excluded. Appendix B.4 provides the complete latency and memory record.

4.5 ABLATION STUDIES

We evaluate opacity, progressive training, and visibility conditioning on 128 Tornado frames from a fixed held-out viewpoint. Results use the final SR=2 support, without test-based checkpoint selection. Table 2 summarizes two experimental groups; full metrics and protocols appear in Appendices B.5 and B.6.

Variant	PSNR (dB)↑	SSIM↑
Opacity and training schedule		
Fixed dense	**36.3404**	**0.96266**
Progressive	35.7007	0.96229
Constant prior + residual	35.3458	0.96245
Density prior only	31.8603	0.90689
Paired visibility conditioning		
w/o visibility cue	28.9215	0.94788
w/ visibility cue	**36.3472**	**0.96270**

Table 2: Ablations averaged over 128 held-out images. Both visibility variants share initialization, architecture, and training settings; the control replaces only the decoder's visibility input with one.

Opacity and progressive training. Replacing the density prior with a constant reduces PSNR by 0.995 dB, while removing the learned opacity residual leads to a larger drop of 4.480 dB, highlighting the importance of both components. Progressive training reduces training time by 11.7%, at the cost of a 0.64 dB decrease in PSNR.

Visibility conditioning. The control retains shadow rasterization but replaces the appearance MLP's visibility input with one. Under matched training settings, using the visibility cue improves mean PSNR by 7.43 dB. Figure 5 illustrates the resulting shading improvement on this sequence.

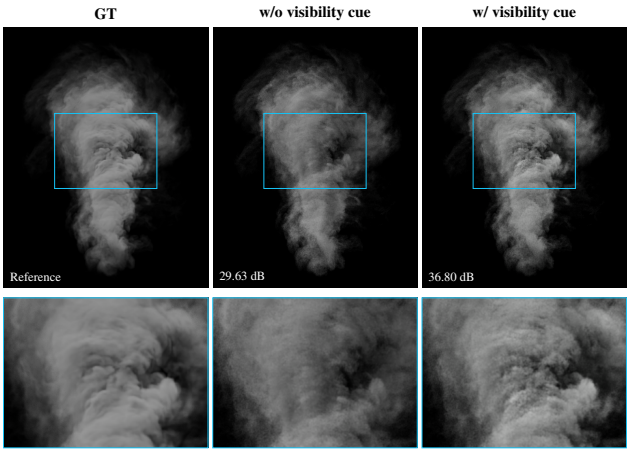

Figure 5: Visibility conditioning on a held-out Tornado image. Top: GT, without and with the visibility cue; bottom: matched detail crops. Without the visibility cue, the rendering appears overly smooth and loses shading details. The visibility cue enables more accurate reproduction of lighting and shadow details. Labels report single-image PSNR.

5 Limitations and Discussion

SGGF, like most Gaussian splatting methods, requires sequence-specific training and reference images. Its shared field does not replace the per-frame density support: preprocessed sample arrays still impose substantial storage and GPU-memory costs. Reported dynamic throughput assumes full-sequence GPU residency. Splatting may smooth thin structures or produce coverage artifacts, and visibility conditioning does not enforce exact volumetric light transport. Future work could combine temporal support compression with adaptive residency and asynchronous streaming to support longer or denser assets.

6 Conclusion

We presented SGGF, a shared Gaussian attribute field that uses known density evolution for efficient, relightable rendering in agent-assisted volumetric creation. Experiments assess density-guided opacity, visibility conditioning, and progressive sampling, with per-frame input storage remaining a practical constraint.

AI Use Statement

Generative AI tools assisted with volume asset generation in Houdini, implementation, experiment planning, interpretation of existing diagnostics, and manuscript polishing. The Mushroom and the volumetric Cornell-Box-like scene (CBox) were generated using Codex with GPT-5.6 Sol to construct Houdini simulations; the asset-generation prompts are provided in Appendix H.

REPRODUCIBILITY STATEMENT

Section 3 specifies the representation, image formation, and training schedule. Section 4 defines the hardware and timing scope. The appendices provide full tabulated measurements, evaluation configurations, image-processing conventions, and additional visual comparisons.

REFERENCES

Zoubin Bi, Yixin Zeng, Chong Zeng, Fan Pei, Xiang Feng, Kun Zhou, and Hongzhi Wu. GS^3: Efficient relighting with triple gaussian splatting. In *SIGGRAPH Asia 2024 Conference Papers*, 2024. doi: 10.1145/3680528.3687576. URL https://gsrelight.github.io/.

Adam Celarek, George Kopanas, George Drettakis, Michael Wimmer, and Bernhard Kerbl. Does 3D Gaussian Splatting need accurate volumetric rendering? *Computer Graphics Forum*, 2025. doi: 10.1111/cgf.70032. URL https://arxiv.org/abs/2502.19318.

Jorge Condor, Sébastien Speierer, Lukas Bode, Aljaž Božič, Simon Green, Piotr Didyk, and Adrián Jarabo. Don't splat your gaussians: Volumetric ray-traced primitives for modeling and rendering scattering and emissive media. *ACM Transactions on Graphics*, 44(1), 2025. doi: 10.1145/3711853. URL https://arxiv.org/abs/2405.15425.

Jiahui Fan, Fujun Luan, Jian Yang, Miloš Hašan, and Beibei Wang. RNG: Relightable neural gaussians. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025. URL https://whois-jiahui.fun/project_pages/RNG/.

Jinkai Hu, Chengzhong Yu, Hongli Liu, Ling-Qi Yan, Yiqian Wu, and Xiaogang Jin. Deep real-time volumetric rendering using multi-feature fusion. In *ACM SIGGRAPH 2023 Conference Proceedings*. ACM, 2023. doi: 10.1145/3588432.3591493. URL https://github.com/What-a-stupid-username/MRPNN.

JangaFX. Free VDB volumes created with EmberGen, n.d. URL https://jangafx.com/software/embergen/download/free-vdb-animations. Accessed September 25, 2026.

James T. Kajiya and Brian P. von Herzen. Ray tracing volume densities. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 165–174. ACM, 1984. doi: 10.1145/800031.808594.

Simon Kallweit, Brian McWilliams, Markus Gross, and Jan Novák. Deep scattering: Rendering atmospheric clouds with radiance-predicting neural networks. *ACM Transactions on Graphics*, 2017. URL https://la.disneyresearch.com/publication/deep-scattering/.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), 2023. URL https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/.

Eric P. Lafortune and Yves D. Willems. Rendering participating media with bidirectional path tracing. In *Rendering Techniques '96*, pp. 91–100. Springer, 1996. URL https://graphics.cs.kuleuven.be/publications/BiDirPart/.

Wei Li, Hanxiao Sun, Tao Huang, Haoxiang Wang, Tongtong Wang, Zherong Pan, and Kui Wu. Real-time neural six-way lightmaps, 2026. URL https://arxiv.org/abs/2604.03748.

Jonathon Luiten, Georgios Kopanas, Bastian Leibe, and Deva Ramanan. Dynamic 3d gaussians: Tracking by persistent dynamic view synthesis. In *International Conference on 3D Vision (3DV)*, 2024. URL https://dynamic3dgaussians.github.io/.

MCP for Blender Contributors. MCP for Blender: Community plugin for language-model tool access, 2026. URL https://github.com/ahujasid/mcp-for-blender. Software repository, accessed September 24, 2026.

Marko Mihajlovic, Sergey Prokudin, Marc Pollefeys, and Siyu Tang. ResFields: Residual neural fields for spatiotemporal signals. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/3bfee3bc6639c36e6e7b058db909f760-Abstract-Conference.html.

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics*, 41(4), 2022. doi: 10.1145/3528223.3530127. URL https://nvlabs.github.io/instant-ngp/.

Krishna Mullia, Fujun Luan, Xin Sun, and Miloš Hašan. RNA: Relightable neural assets. *ACM Transactions on Graphics*, 44(1), 2025. doi: 10.1145/3695866. URL https://arxiv.org/abs/2312.09398.

Rikard Olajos, Michael Doggett, and Prashant Goswami. Environmental volumetric neural shading of clouds for real-time rendering. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 9(4), 2026. doi: 10.1145/3820020. URL https://doi.org/10.1145/3820020.

OpenVDB Contributors. OpenVDB: An open source data structure and toolkit for high-resolution volumes, 2026. URL https://www.openvdb.org/. Project documentation, accessed September 19, 2026.

Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred A. Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 5301–5310. PMLR, 2019. URL https://proceedings.mlr.press/v97/rahaman19a.html.

SideFX. Introduction to Pyro and Pyro Solver, n.d. URL https://www.sidefx.com/docs/houdini/pyro/intro.html. Houdini documentation, accessed September 26, 2026.

Chinmay Talegaonkar, Yash Belhe, Ravi Ramamoorthi, and Nicholas Antipa. Volumetrically consistent 3D Gaussian rasterization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10953–10963, 2025. URL https://openaccess.thecvf.com/content/CVPR2025/html/Talegaonkar_Volumetrically_Consistent_3D_Gaussian_Rasterization_CVPR_2025_paper.html.

Guanjun Wu, Taoran Yi, Jiemin Fang, Lingxi Xie, Xiaopeng Zhang, Wei Wei, Wenyu Liu, Qi Tian, and Xinggang Wang. 4d gaussian splatting for real-time dynamic scene rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 20310–20320, 2024. URL https://arxiv.org/abs/2310.08528.

Jiahao Wu, Rui Peng, Zhiyan Wang, Lu Xiao, Luyang Tang, Jinbo Yan, Kaiqiang Xiong, and Ronggang Wang. Swift4D: Adaptive divide-and-conquer Gaussian Splatting for compact and efficient reconstruction of dynamic scene. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/17bae482066287ce54814514cf12f248-Abstract-Conference.html.

Jiawei Xu, Zexin Fan, Jian Yang, and Jin Xie. Grid4D: 4D decomposed hash encoding for high-fidelity dynamic Gaussian splatting. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/df8b8c12afcaf65639aa686ececbf00c-Paper-Conference.pdf.

Zeyu Yang, Hongye Yang, Zijie Pan, and Li Zhang. Real-time photorealistic dynamic scene representation and rendering with 4D Gaussian Splatting. In *International Conference on Learning Representations*, 2024a. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/26230ff5299de0929d03ed3576c3bbf9-Abstract-Conference.html.

Ziyi Yang, Xinyu Gao, Wen Zhou, Shaohui Jiao, Yuqing Zhang, and Xiaogang Jin. Deformable 3d gaussians for high-fidelity monocular dynamic scene reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 20331–20341, 2024b. URL https://arxiv.org/abs/2309.13101.

Chong Zeng, Guojun Chen, Yue Dong, Pieter Peers, Hongzhi Wu, and Xin Tong. Relighting neural radiance fields with shadow and highlight hints. In *ACM SIGGRAPH 2023 Conference Proceedings*, 2023. doi: 10.1145/3588432.3591482. URL https://arxiv.org/abs/2308.13404.

Wenran Zhang, Yuxiang Cai, Letian Huang, Dongwei Ye, Jie Guo, and Bo Ren. GauSmoke: Hybrid physics-optical gaussian splatting for sparse smoke reconstruction. In *ACM SIGGRAPH 2026 Conference Papers*, 2026. doi: 10.1145/3799902.3811148. URL https://ren-bo.net/papers/zwr_sig2026.pdf.

A Evaluation protocols

This appendix summarizes the datasets and evaluation settings for multi-frame rendering quality, dynamic playback, and progressive training. Table 3 lists the five simulation sequences, including their lengths, training/test view counts, and lighting configurations.

For all five scenes, we use checkpoints saved at epoch 200, with 400 training iterations per epoch.

Scene	Frames	Train/test views per frame	Lighting
Tornado	128	384/96	Random directional light
DustImpact	199	384/96	Random directional light
DustShockwave	155	384/96	Random directional light within a spherical band
CBox	154	384/96	Point light sampled along an orbit
Mushroom	144	80/20	Fixed directional light and volumetric emission

Table 3: Sequence lengths, train/test views per frame, and lighting configurations for the datasets introduced in Section 4.1.

A.1 Multi-frame quality evaluation

We evaluate Tornado, DustImpact, and DustShockwave on a fixed subset of 24 images per scene, using three viewpoints excluded from training. Table 4 reports the results for this subset. MRPNN is evaluated at 256 samples per pixel (spp) with a 512^3 density grid. Metrics are computed per image and averaged over the 24 images for each scene. SSIM uses an 11×11 Gaussian window with a standard deviation of 1.5, zero padding, and population variance.

A.2 Dynamic playback evaluation

The GPU-resident playback experiment in Section 4.4 is conducted on Tornado with a fixed camera and lighting configuration. All 128 point clouds, generated with a sampling reduction factor of 2, are preloaded onto the GPU. Playback selects the resident point cloud for each frame without per-frame host-to-device point transfers.

A.3 Progressive training evaluation

On Tornado, we compare fixed dense sampling with the coordinated progressive schedule described in Section 4.5. The progressive schedule increases the sampling budget from approximately 62,600 to 3.683 million Gaussians per frame on average. The measured training costs are 8.3732 GPU-hours for fixed dense sampling and 7.3942 GPU-hours for progressive training, including initialization and resume overhead.

B Complete quantitative results

B.1 Multi-frame quality across volumetric scenes

We evaluate a frozen subset of eight time samples and three held-out views per scene: 24 images each for DustImpact, DustShockwave, and Tornado. Each method uses the same cameras, per-record lights, 1024×1024 references, and black background. MRPNN uses the authors' renderer (Hu et al., 2023) at 256 spp with a 512^3 density grid. Scene-specific model and rendering settings are given in Appendix A.1.

Scene	Method	PSNR↑	SSIM↑	Alpha MAE↓
DustImpact	SGGF	**41.124**	0.9661	0.00426
	MRPNN	38.056	**0.9714**	**0.00221**
DustShockwave	SGGF	35.515	0.9488	0.00448
	MRPNN	**36.095**	**0.9495**	**0.00399**
Tornado	SGGF	**35.870**	**0.9759**	0.00376
	MRPNN	34.426	0.9702	**0.00206**

Table 4: Multi-frame quality on frozen subsets per scene. PSNR is in dB. MRPNN's alpha is computed by independent density integration.

On DustImpact, SGGF exceeds this MRPNN configuration by 3.068 dB PSNR, while its SSIM is lower. On DustShockwave, SGGF is lower by 0.581 dB PSNR; MRPNN also has higher SSIM and lower alpha MAE. On Tornado, SGGF with a sampling reduction factor of 2 exceeds MRPNN at 256 spp by 1.444 dB in PSNR and 0.00564 in SSIM; MRPNN has lower alpha MAE (0.00206 versus 0.00376). The results show scene-dependent behavior rather than a uniform quality advantage. Overall, SGGF achieves rendering quality broadly comparable to the volume-path-tracing-based MRPNN renderer.

B.2 Quality–latency tradeoff

We evaluate the quality–latency tradeoff on a single frame at a fixed viewpoint, varying the Gaussian count for SGGF and the samples per pixel for MRPNN. Tables 5 and 6 report the results.

Gaussians (M)	PSNR (dB)↑	SSIM↑	GPU time (ms)↓
1.890	32.689	0.9512	14.55
3.779	34.866	0.9803	28.60
7.559	37.984	0.9842	55.75

Table 5: Quality and rendering latency of SGGF at different Gaussian counts.

Samples per pixel	PSNR (dB)↑	SSIM↑	GPU time (ms)↓
1	23.041	0.8078	24.51
4	29.337	0.8449	52.30
16	33.859	0.9182	152.32
64	34.946	0.9701	530.83
256	35.094	0.9819	2064.12

Table 6: Quality and rendering latency of pretrained MRPNN at different sampling budgets.

Increasing the Gaussian count improves reconstruction quality at a higher rendering cost. At approximately 35 dB PSNR, SGGF renders in 28.60 ms compared with 530.83 ms for MRPNN, yielding an 18.56-fold speedup with a PSNR difference of 0.080 dB.

B.3 Runtime breakdown

Table 7 reports the GPU time for field evaluation, shadow computation, color decoding, and camera rasterization. Shadow computation is the dominant stage, accounting for approximately 57–60% of the profiled stage time.

B.4 GPU-resident dynamic sequence rendering

We benchmark one warm pass over 128 consecutive Tornado frames on a single RTX A6000, with a fixed camera and lighting configuration. Both the output images and shadow maps have a resolution of 1024×1024. Each frame contains 3.550–3.994 million Gaussians, with all input point clouds

Gaussians (M)	Field	Shadow	Color	Camera raster	Total
1.890	3.46	8.30	0.37	2.54	14.55
3.779	5.86	16.75	0.70	5.34	28.60
7.559	9.90	33.43	1.38	11.03	55.75

Table 7: GPU runtime in milliseconds. Stage sums may differ slightly from the measured total latency.

preloaded onto the GPU. Field evaluation, geometry and appearance decoding, shadow computation, and camera rasterization are performed for each frame.

The runtime uses FP16 hash tables, features, and MLP weights, and FP32 input points. Table 8 summarizes its performance.

Measurement	Value
Rendering throughput	22.40 fps
Median frame latency	43.98 ms
95th-percentile frame latency	47.98 ms
Sampled GPU memory peak	30.13 GiB
Resident input point clouds	7.02 GiB

Table 8: GPU-resident dynamic rendering on Tornado. Latency is measured in wall-clock time. Timings exclude input preloading and CPU readback.

B.5 Density-prior and progressive-training ablations

We evaluate four variants on all 128 Tornado frames from the same held-out viewpoint. Each variant is trained on two RTX A6000 GPUs for 200 epochs. All variants share the same final sampling support, averaging 3.683 million Gaussians per frame, with a kernel-scale coefficient of 1. For variants using the density-based opacity prior, its coefficient is set to 0.58. Table 9 reports the mean image-quality metrics and measured training time.

Variant	PSNR (dB)↑	SSIM↑	Alpha MAE↓	Training time (h)↓
Fixed dense	**36.3404**	**0.96266**	**0.004522**	4.1866
Progressive	35.7007	0.96229	0.004591	**3.6971**
Constant prior + residual	35.3458	0.96245	0.004927	4.3384
Density prior only	31.8603	0.90689	0.015654	4.3384

Table 9: Ablations on Tornado. Fixed dense and Progressive use a density prior with a learned residual; the other variants use fixed dense sampling. Training time includes initialization and resume overhead.

Density prior and learned residual. Replacing the density prior with a constant lowers PSNR by 0.995 dB, while removing the learned opacity residual lowers PSNR by 4.480 dB. Both components contribute to reconstruction quality.

Progressive training. The progressive run takes 3.6971 hours compared with 4.1866 hours for fixed dense sampling, using 11.69% less training time at the same final update count, with a PSNR decrease of 0.640 dB.

B.6 Visibility cue ablation

We compare the full model with a control that replaces the per-Gaussian visibility input to the color MLP with a constant one during both training and evaluation. Both variants retain shadow

computation and share the same initialization, network architecture, parameter count, and training settings.

Both variants are trained on the full 128-frame Tornado sequence using two RTX A6000 GPUs. They share a sampling reduction factor of 2, a kernel-scale coefficient of 1, and a density-based opacity-prior coefficient of 0.58. We evaluate the final checkpoints on 128 held-out images, one per frame, all from the same held-out viewpoint.

Variant	PSNR (dB)↑	SSIM↑	Alpha MAE↓
w/o visibility cue	28.9215	0.94788	0.004562
w/ visibility cue	**36.3472**	**0.96270**	**0.004526**

Table 10: Visibility cue ablation on Tornado, averaged over 128 held-out images.

Visibility conditioning improves mean PSNR by 7.426 dB and SSIM by 0.01482, with improvements in both metrics on all 128 frames. Alpha MAE remains similar, indicating that the main benefit lies in reproducing RGB appearance. Figure 5 illustrates the improvement in shading detail.

C Representation and rendering details

Density samples, cameras, and lights share a common world coordinate system defined by the volume-to-world transform. The density threshold and sample rate determine the spatial support. Frame indices are placed at temporal bin centers before normalization. Increasing the number of rendering samples does not change the number of learnable field and decoder parameters.

For pixel p, let $a_i(p)$ denote the projected opacity of Gaussian i, ordered from front to back. The rendered color is

$$\widehat{\mathbf{I}}(p) = \sum_i T_i(p) a_i(p) \mathbf{c}_i + T_{\text{end}}(p) \mathbf{c}_{\text{bg}}, \qquad T_i(p) = \prod_{j<i} (1 - a_j(p)). \tag{12}$$

The remaining transmittance is $T_{\text{end}}(p) = \prod_i (1 - a_i(p))$, and the output alpha is $1 - T_{\text{end}}(p)$. At a fixed time, changing illumination recomputes visibility and appearance. Changing time first updates the density support and decoded geometry.

D Interpretation of the kernel schedule

Under approximately uniform three-dimensional sampling, the typical point spacing scales as $S_e^{1/3}$, where S_e is the sampling reduction factor. This motivates the kernel-scale adjustment in Eq. (10).

In the low-opacity regime, the projected Gaussian footprint scales approximately as K_e^2, while the prior opacity scales as $\rho C_e / K_e^2$. Their aggregate contribution therefore scales as $N_e \rho C_e$, where N_e is the sample count. Since $N_e \propto 1/S_e$, choosing $C_e \propto S_e$ approximately compensates for changes in sampling density. This approximation does not enforce exact conservation under occlusion, opacity saturation, or learned residual corrections.

The schedule operates on nested sample subsets, retaining earlier samples as the sampling budget increases.

E MRPNN sampling convergence

We use the authors' public implementation and pretrained weights (Hu et al., 2023), retaining the full multiple-scattering network and applying Linux compatibility and dataset-camera adaptations. The input density is resampled from VDB to a 512^3 grid, while the output image resolution is 1024×1024. These spatial resolutions are independent of the samples-per-pixel (spp) budget.

Table 11 reports measurements on Tornado. We retain the original cameras, per-image directional lights, and Cycles references rendered with a bounce limit of four. The reported quality scores and RGB output times are means over these three views.

spp	PSNR (dB)	SSIM	RGB output time (s)
256	34.591	0.97374	2.164
1024	34.669	0.98007	8.557

Table 11: Pretrained MRPNN measurements on one Tornado frame and three views at 1024×1024. RGB output time includes rendering, synchronization, and device-to-host readback; it is not a GPU-resident inference-latency measurement.

Increasing the sampling budget fourfold improves mean PSNR by 0.078 dB and SSIM by 0.00633 on this subset. Alpha MAE is 0.001641 for both settings. Alpha is computed separately by density integration and is not an output of the radiance-prediction network. The RGB timings exclude approximately 4.8 s of initialization, separate alpha computation, display conversion, and image saving. They characterize high-sample quality exports and do not establish real-time performance or dynamic volume-update cost. The three-view RGB-export protocol and the single-frame GPU protocol use different sample sets and timing boundaries.

F Additional qualitative results

F.1 Image-space errors and detail comparisons

Figure 6 compares reference images, MRPNN, and SGGF using error maps and matched detail crops. Images are displayed on a black background, with RGBA images composited using their own alpha. For display-RGB values normalized to $[0, 1]$, the pixelwise error is

$$E(\mathbf{p}) = \frac{1}{3} \sum_{c \in \{R,G,B\}} \left| I_c(\mathbf{p}) - I_c^{\mathrm{ref}}(\mathbf{p}) \right|. \tag{13}$$

The same error definition is used for the additional scenes below.

F.2 Additional volumetric scenes

Figure 7 supplements Figure 3 with error maps for Mushroom, DustImpact, and DustShockwave. Each scene is shown at a held-out viewpoint, with MRPNN evaluated at 256 samples per pixel.

F.3 Temporal rendering

Figure 8 shows five time steps for Mushroom, CBox, and DustImpact, with camera and illumination fixed within each sequence. Mushroom is shown from a training viewpoint, DustImpact from a held-out viewpoint, and CBox from a centered camera under an overhead point light.

G Supplementary comparison with Grid4D

We compare Grid4D (Xu et al., 2024) and SGGF on Tornado frame 63 at three matched viewpoints, using the same cameras and 1024×1024 reference images. Grid4D learns Gaussian deformations from images, whereas SGGF additionally uses simulation-provided density support. On these views, SGGF achieves higher PSNR and SSIM, with an average PSNR improvement of 4.64 dB (Table 12).

View	Grid4D PSNR	Grid4D SSIM	SGGF PSNR	SGGF SSIM
82	39.215	0.9746	44.578	0.9898
83	38.363	0.9702	43.966	0.9888
97	40.028	0.9813	42.991	0.9886
Mean	39.202	0.9754	43.845	0.9891

Table 12: Rendering quality on three views of Tornado frame 63. PSNR (dB) and SSIM are evaluated on full-frame RGB images; higher is better.

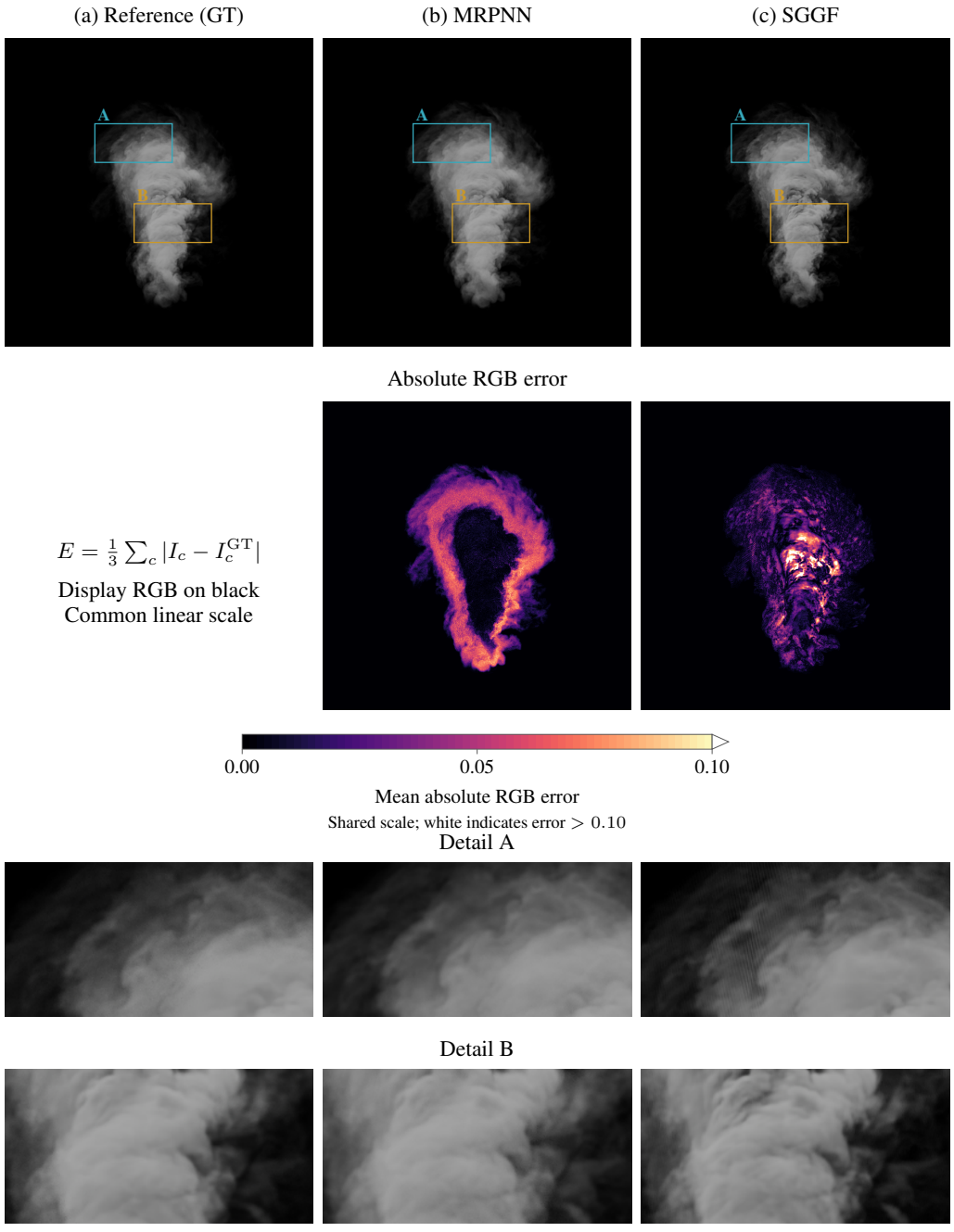

Figure 6: Reference, MRPNN, and SGGF with matched detail crops. Row 1: full images with regions A and B marked. Row 2: mean absolute RGB error on a shared scale, saturated above 0.10. Rows 3–4: enlarged views of the same 256×128-pixel regions. Our residual errors are concentrated in shadowed regions, suggesting limitations in approximating indirect illumination: the visibility cue explicitly captures occlusion toward the light source, while more complex scattering contributions must be learned by the appearance decoder from reference images.

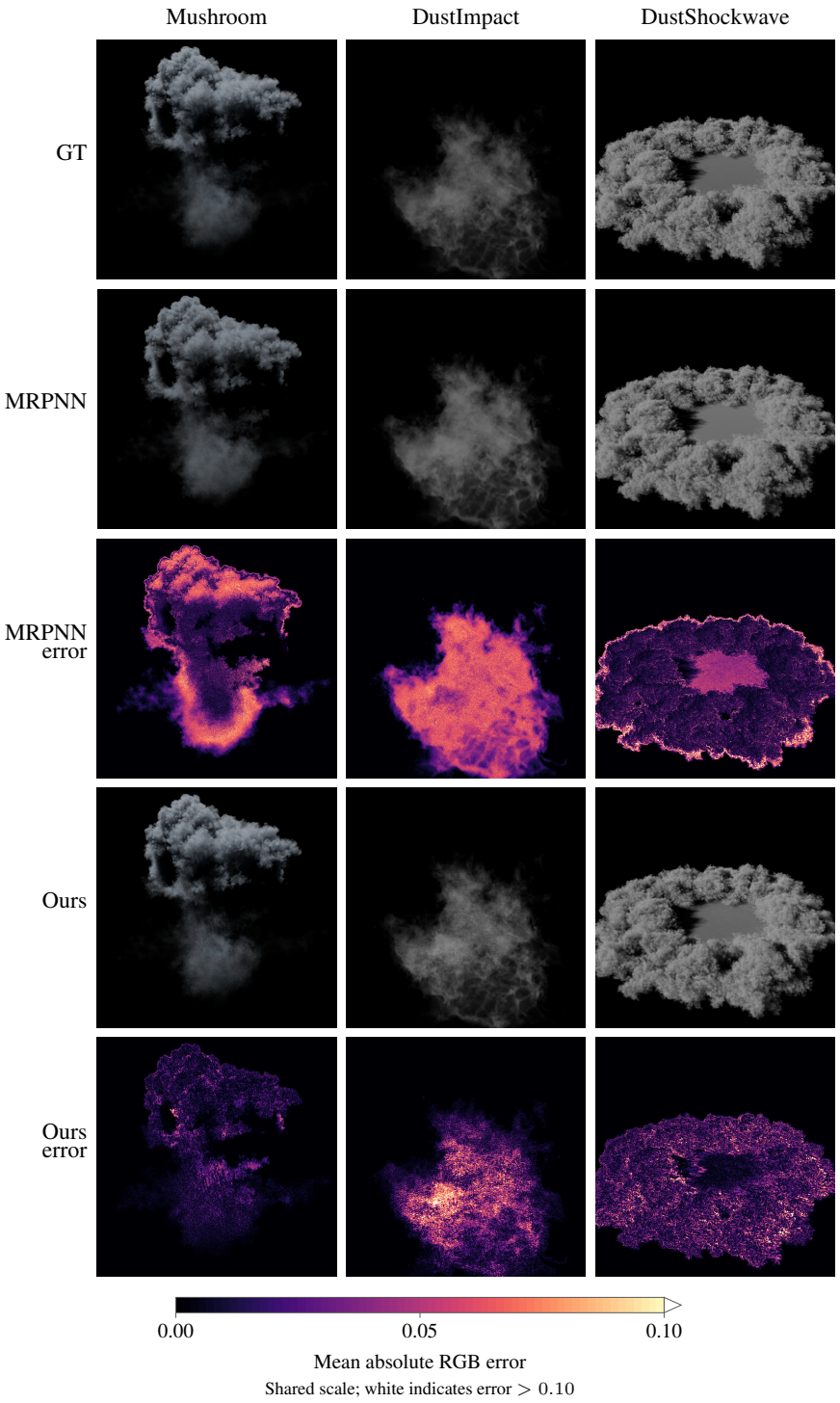

Figure 7: Additional scene comparisons. Rows: GT, MRPNN, MRPNN error, SGGF, and SGGF error. Error maps share a linear scale from 0 to 0.10, with white indicating larger errors.

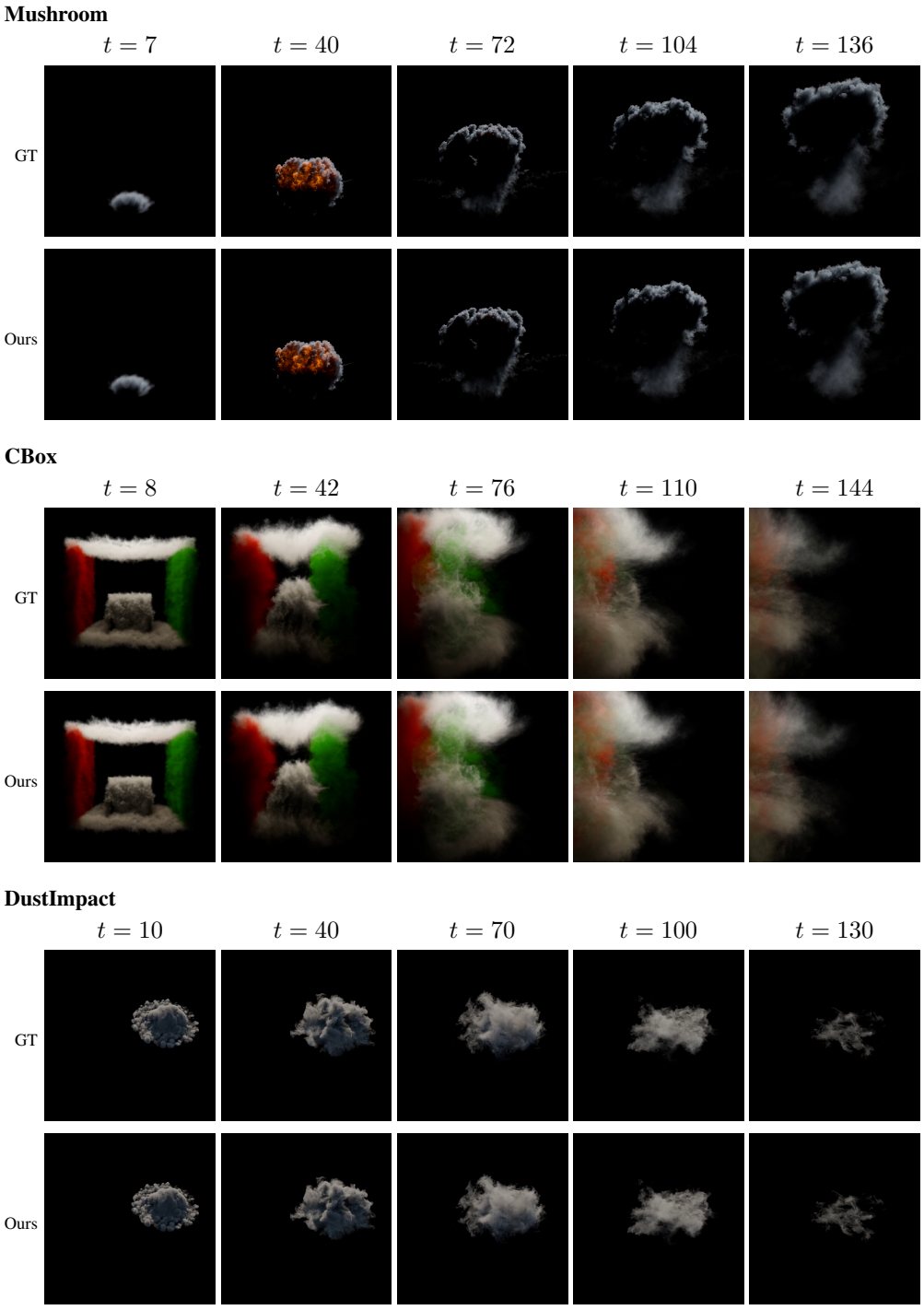

Figure 8: Temporal rendering comparisons, with GT above the corresponding predictions. Each panel shows the full 1024×1024 frame at a common display size. Images are extracted from H.264 videos for qualitative visualization.

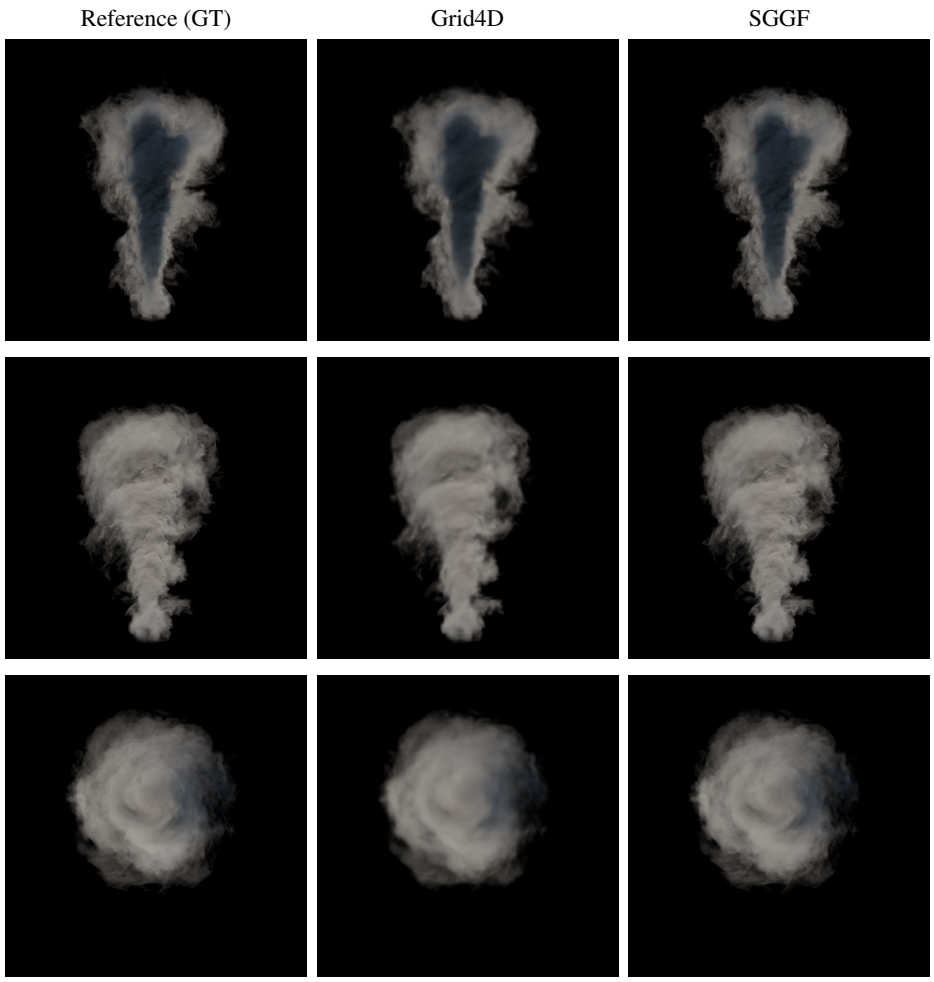

Figure 9: Qualitative comparison on Tornado. Rows correspond to the three evaluated viewpoints; columns show the reference, Grid4D, and SGGF.

H PROMPTS FOR AGENT-ASSISTED VOLUME ASSET GENERATION

We used Codex with GPT-5.6 Sol to construct the Mushroom and CBox simulations in Houdini Sparse Pyro and export their VDB sequences. We provide English versions of the prompts used to guide this process. The shared requirements are combined with the corresponding scene-specific prompt. The prompts retain the simulation and visual requirements while omitting machine-specific execution details. They describe the requested asset-generation behavior; the resulting VDB sequences serve as inputs to the rendering method.

H.1 SHARED REQUIREMENTS

> Use Houdini Sparse Pyro for simulation and Blender Cycles for rendering. Provide scripts and a unified configuration that can rebuild the scene from scratch. Set the voxel size to 0.02 scene units, and export a VDB sequence and a preview video.
>
> Fine-scale details must be advected by the velocity field. Do not add stationary world-space noise as a post-processing effect. Keep the coordinate system, camera, and exposure fixed, use a black background, and include no visible meshes or collision meshes.

H.2 MUSHROOM

Create a compact mushroom-cloud explosion lasting approximately six seconds. Begin with a brief, asymmetric burst from multiple jets, forming a low, broad mushroom cap and a short stem early in the animation.

The appearance should be dominated by dense, dark-gray, curling smoke. Keep orange firelight mostly concealed inside the smoke, with localized glimpses through it. Show clearly defined large-, medium-, and small-scale vortices and self-shadowing.

After injection stops, let the cloud spread naturally outward and gradually become thinner. Do not stretch it into a tall, narrow smoke column or make it fade abruptly. Emphasize a substantial volumetric appearance and a cinematic explosion aesthetic.

H.3 CBOX

Create an all-smoke Cornell Box with a red left wall, a green right wall, white top and bottom boundaries, and central geometric forms made of smoke. Leave the front and back open.

Fill the interiors of all volumes. Their boundaries should exhibit natural entrainment structures at multiple scales; avoid hollow shells, solid-looking skins, and fuzzy noise artifacts.

Use a main light above the scene to produce pronounced vertical shadows, without frontal fill lighting. First show the initial arrangement, then apply horizontal wind so that the smoke curls, mixes, and gradually dissipates. All colored smoke must share the same velocity and pressure fields.

Emphasize the initial composition and dense smoke with substantial volume and layered structure.