Bounding Stratified Bernoulli Impulses for Ray Marching Gaussian Process Implicit Surfaces

JUNJIE CHEN, State Key Lab for Novel Software Technology, Nanjing University, China
ZHIMIN FAN, State Key Lab for Novel Software Technology, Nanjing University, China
LING-QI YAN, Mohamed bin Zayed University of Artificial Intelligence, United Arab Emirates
JUNQIU ZHU, Shandong University, China
YANWEN GUO, State Key Lab for Novel Software Technology, Nanjing University, China
KUN ZHOU, State Key Lab of CAD&CG, Zhejiang University, China
JIE GUO*, State Key Lab for Novel Software Technology, Nanjing University, China

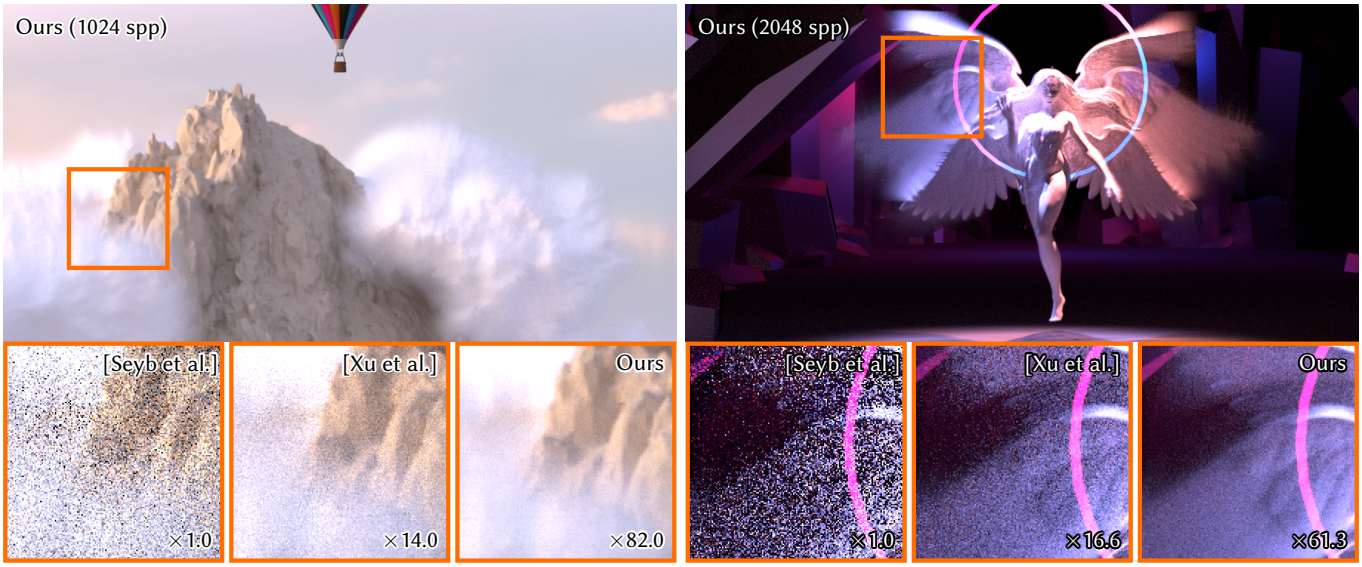


Fig. 1. We develop *bounded ray marching* with *stratified Bernoulli impulses*. The mountain, cloud (in the left scene), and the Angel (in the right scene) are all modeled by GPISes with different covariance kernels and mean functions, rendered with the same light transport framework. Equal-time (1 hour) comparison against previous works [Seyb et al. 2024; Xu et al. 2025] shows that our method offers substantial speedup, in terms of mean squared error (MSE). The right scene adapted from Angel model ©ida-faber and Neon stage ©AnixMoonLight.

The theory of light transport on Gaussian process implicit surface (GPIS) provides a unified framework for rendering surfaces, participating media, and the intermediate spectrum. However, previous approaches rely on brute-force ray marching for surface intersections, requiring full noise evaluations at each marching point, whether using multivariate Gaussian sampling or sparse convolution noise approximation. This imposes a severe limitation on the rendering efficiency.

In this paper, we derive bounds to significantly reduce the total number of full noise evaluations, leading to efficient ray marching for ray–surface intersections. We introduce stratified Bernoulli impulses, enabling a fast point-level bound for individual realizations to replace unnecessary full noise evaluations. To further reduce the number of point-level bound evaluations, we propose a region-level bound, leveraging a spatial acceleration structure to prune probabilistically empty regions, thereby avoiding unnecessary marching points in advance. By combining these two bounds, our bounded ray marching accelerates ray–surface intersections in GPIS, and consequently significantly improves overall GPIS rendering efficiency. Code for this paper are at https://github.com/Cchen-77/bounded-gpis.

CCS Concepts: • **Computing methodologies → Ray tracing**.

*The corresponding author.

Authors' Contact Information: Junjie Chen, State Key Lab for Novel Software Technology, Nanjing University, Nanjing, China, Zoltraak77@outlook.com; Zhimin Fan, State Key Lab for Novel Software Technology, Nanjing University, Nanjing, China, zhiminfan2002@gmail.com; Ling-Qi Yan, Mohamed bin Zayed University of Artificial Intelligence, Abu Dhabi, United Arab Emirates, Lingqi.Yan@mbzuai.ac.ae; Junqiu Zhu, Shandong University, Shandong, China, zhujunqiu@mail.sdu.edu.cn; Yanwen Guo, State Key Lab for Novel Software Technology, Nanjing University, Nanjing, China, ywguo@nju.edu.cn; Kun Zhou, State Key Lab of CAD&CG, Zhejiang University, Hangzhou, China, kunzhou@zju.edu.cn; Jie Guo, State Key Lab for Novel Software Technology, Nanjing University, Nanjing, China, guojie@nju.edu.cn.

ACM 1557-7368/2026/7-ART120
https://doi.org/10.1145/3811311

ACM Reference Format:
Junjie Chen, Zhimin Fan, Ling-Qi Yan, Junqiu Zhu, Yanwen Guo, Kun Zhou, and Jie Guo. 2026. Bounding Stratified Bernoulli Impulses for Ray Marching Gaussian Process Implicit Surfaces. *ACM Trans. Graph.* 45, 4, Article 120 (July 2026), 15 pages. https://doi.org/10.1145/3811311

1 INTRODUCTION

Classical light transport simulation assumes that scene components can be treated independently as either participating media or deterministic surfaces, with each governed by a distinct light transport equation [Chandrasekhar 1960; Kajiya 1986]. This assumption not only imposes a separation at the algorithmic level but also makes it challenging to render appearances that fall between deterministic surfaces and participating media, e.g., the Angel in Figure 1.

Recently, Seyb et al. [2024] proposed a unified rendering framework using their new light transport theory on Gaussian Process Implicit Surface (GPIS), covering deterministic surfaces, participating media, and the intermediate spectrum. Despite remarkable expressive power, its computational cost is significantly higher than conventional path tracing, as it relies on ray marching with expensive multivariate Gaussian sampling for ray–surface intersections. To make it practical, Xu et al. [2025] introduced a sparse convolution noise approximation of the Gaussian Process (GP). By replacing multivariate Gaussian sampling with sparse convolution noise evaluations, it significantly reduces the computational cost. Nevertheless, sparse convolution noise evaluation is still required for each marching point. Furthermore, the use of brute-force ray marching [Seyb et al. 2024; Xu et al. 2025] produces many marching points, the number of which is linear in ray length. This leads to numerous noise evaluations, leaving the ray–surface intersection a key performance bottleneck for rendering GPIS.

In this paper, we present *bounded ray marching* that dramatically reduces the number of marching points requiring full noise evaluations, significantly accelerating ray–surface intersection for GPIS. Our key observation is that an inexpensive bound, rather than a precise but costly noise evaluation, is sufficient to guarantee non-intersection. To make feasible and efficient bounds, we adopt *stratified Bernoulli impulses* (SBIs) for sparse convolution noise [Tavernier et al. 2019], which enables categorizing realizations using impulse-weight corresponding binary sequences, allowing precomputations for each category. Based on SBIs, we then develop an efficient *point-level bound* to skip unnecessary noise evaluations at individual marching points. Furthermore, we propose a *region-level bound* to prune regions unlikely to contain intersections, thereby reducing the total number of point-level bound evaluations. These bounds reduce full noise evaluations by up to 97.4%, providing a highly efficient ray marching procedure for GPIS. As shown in our experiments, our bounded ray marching visibly accelerates ray–surface intersections for GPIS, and consequently, GPIS rendering.

In summary, our main contributions include:

- A new impulse type, stratified Bernoulli impulses, enabling the classification of 1D sparse convolution noise (consequently, 1D GP realization) by impulse-weight-corresponding binary sequence, and supporting precomputation.
- A point-level bound tailored for stratified Bernoulli impulses, allowing early skipping of full noise evaluation for marching points guaranteed non-intersection.
- A region-level bound via a mean-guided sparse voxel octree, pruning regions unlikely to contain intersections and facilitating selective evaluation of only the "non-empty" GPIS within each region when multiple GPISes are overlapped.

2 RELATED WORK

Gaussian process implicit surface. Gaussian process implicit surfaces have been applied to surface reconstruction [Dragiev et al. 2011; Williams and Fitzgibbon 2006], where the GP is conditioned on observed data. The resulting deterministic surface is typically obtained from the posterior GP mean, while the GP (co)variance provides a measure of uncertainty [Poethkow et al. 2013; Pöthkow et al. 2011].

Rather than producing a deterministic implicit surface directly from the GP mean, Seyb et al. [2024] proposed a light transport framework that samples GP realizations on-the-fly and ensemble-averages over the light transport on the corresponding implicit surfaces, thereby explicitly accounting for uncertainty during rendering. Xu et al. [2025] introduced the sparse convolution noise approximation of GP, reducing the computational cost of sampling GP realization. In addition, they enable next-event estimation (NEE) for GPIS with specular underlying BRDFs, improving the efficiency of the Monte Carlo light transport estimator.

Despite these improvements, GPIS rendering performance is still limited by the cost of ray–surface intersections. Seyb et al. [2024] limited ray segment lengths using prescribed scene bounds (e.g., a bounding box), thereby reducing the total number of points requiring GP evaluation. They also proposed principles for discarding regions of space that are unlikely to contain zero crossings by first ray marching with point-wise occupancy probabilities.

In this paper, we adopt a similar probability-based heuristic, constructing a spatial acceleration structure to prune probabilistically empty regions. We also propose ray marching with efficient point-level bounds for individual realizations, rather than point-wise occupancy probabilities aggregated over all realizations.

Sparse convolution noise. Sparse convolution noise [Lewis 1989] is a type of procedural noise defined as the convolution of a compact kernel with an impulse process. By carefully designing the kernel, sparse convolution noise can achieve a wide range of spatial and spectral characteristics, enabling rich and controllable noise patterns [Galerne et al. 2017; Lagae et al. 2010, 2009; van Wijk 1991].

Lagae et al. [2009] introduced Gabor noise, i.e., sparse convolution noise with Gabor kernels, and developed a framework to generate it efficiently. Lagae et al. [2011] further sped up isotropic Gabor noise and introduced multi-resolution techniques to enable efficient evaluation for cases with spatially varying parameters. Tavernier et al. [2019] revisited the ingredients of Gabor noise: kernel, impulse distribution, and impulse weight, envisioned the alternatives, and discussed their impacts. Though these improvements are originally for Gabor noise, they can be directly applied to any sparse convolution noise.

Despite its expressive power, evaluating sparse convolution noise is computationally expensive, as each point evaluation requires multiple kernel evaluations over nearby impulses. In this paper, we show that 1D sparse convolution noise with stratified Bernoulli impulses is particularly well-suited for precomputation, allowing us to leverage precomputed information to avoid unnecessary full evaluations and improve efficiency.

Ray marching for implicit surfaces. Tracing implicit surfaces relies on root-finding methods to locate zero crossings along rays. Among numerical approaches, ray marching and its variants are the most widely used. Brute-force ray marching [Perlin and Hoffert 1989] advances along a ray using fixed step sizes and requires only point evaluations of the function, without relying on any analytical properties. Hart [1996] introduced sphere tracing, which accelerates marching by adapting the step size according to a Lipschitz bound. Keinert et al. [2014] further accelerated sphere tracing by proposing a safe over-relaxation scheme. Galin et al. [2020] proposed segment tracing, which computes the Lipschitz bound locally and adaptively, accelerating the overall ray marching process. Bán and Valasek [2025] introduced a discrete Lipschitz field from local polynomial approximations to infer safe ray-marching step sizes, enabling efficient and conservative rendering without prior knowledge of Lipschitz constants. Moinet and Neyret [2025] accelerated sphere tracing for sums of bounded functions based on a procedural Bounding Volume Hierarchy. For functions defined by construction trees, tree pruning algorithms [Barbier et al. 2025; Hubert-Brierre et al. 2025; Uchytil and Storti 2023] can be employed to reduce the complexity of point evaluations.

In this paper, we draw inspiration from the ray-marching acceleration techniques for deterministic implicit surfaces and adapt them to the stochastic implicit surface, GPIS, where surfaces are generated on the fly. Specifically, we introduce efficient point-level bounds for individual realizations, thereby avoiding expensive noise evaluations whenever possible.

Acceleration structures for implicit surfaces. In addition to adaptive step-size and point evaluation strategies, spatial acceleration structures are often employed to further speed up ray marching. Kalra and Barr [1989] employed an octree to prune empty regions, thereby avoiding unnecessary function evaluations along the ray. Sparse structures [Laine and Karras 2010; Museth 2013] enable efficient representation of function values in regions of interest, allowing large empty regions to be skipped. Uchytil and Storti [2023] employed an N^3-tree sparse volume data structure to store pruning information as well as model geometry. Zanni [2024] utilized a tile-based acceleration structure built per frame to identify active primitives, enabling the construction of a coherent pruned blobtree view shared within each workgroup.

In this paper, we propose region-level bounds as a criterion to discard probabilistically empty regions, as in Seyb et al. [2024]. Consequently, the remaining regions form a new volume that is deterministic under the fixed criterion. We then employ a spatial acceleration structure, following previous work on implicit surfaces, to efficiently maintain and traverse regions retained by our pruning criterion.

3 PRELIMINARIES

3.1 Gaussian Process

A Gaussian process $f \sim \mathcal{GP}(\mu, \kappa)_{\mathbb{R}^3}$ is a distribution over functions f characterized by its mean function $\mu(\boldsymbol{x}) = \mathbb{E}(f(\boldsymbol{x}))$ and covariance function $\kappa(\boldsymbol{x}, \boldsymbol{x}') = \mathrm{Cov}(f(\boldsymbol{x}), f(\boldsymbol{x}'))$. In the function-space view, a Gaussian process can be interpreted as a collection of Gaussian variables [Williams and Rasmussen 2006], where each variable corresponds to a function value. For any finite collection of input points $\mathrm{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \subset \mathbb{R}^3$, the corresponding function values follow a joint Gaussian distribution:

$$f(X) \sim \mathcal{N}(\mu(\mathrm{X}), \kappa(\mathrm{X}, \mathrm{X})), \tag{1}$$

where $\mu(\mathrm{X}) = [\mu(\boldsymbol{x}_1), \ldots, \mu(\boldsymbol{x}_n)]^\top$, and $\kappa(\mathrm{X}, \mathrm{X}) = [\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{i,j=1}^n$.

3.2 Sparse Convolution Noise Approximation of GP

In addition to the function-space view, a GP can also be approximated using sparse convolution noise, yielding a weight-space interpretation with local bases.

A realization f from $\mathcal{GP}(\mu, \kappa)$ can be decomposed into a deterministic mean component and a stochastic zero-mean component:

$$f(x) = \mu(x) + \psi(x), \qquad \psi(x) \sim \mathcal{GP}(0, \kappa). \tag{2}$$

While ψ can be regarded as an "infinite-dimensional multivariate Gaussian" in the function-space view, its convolution representation offers a more useful perspective [Noack et al. 2024; Thiebaux 1976]:

$$\psi(x) = \int_\Omega h(\boldsymbol{x}, \boldsymbol{s}) W(\boldsymbol{s}) \mathrm{d}\boldsymbol{s}, \tag{3}$$

where $\Omega \subseteq \mathbb{R}^3$ denotes the sub-domain of interest for ψ, $W(\boldsymbol{s})$ is the Gaussian white noise, and the convolution kernel $h(\cdot)$ is chosen such that it induces the covariance function:

$$\kappa(\boldsymbol{x}, \boldsymbol{y}) = \int_\Omega h(\boldsymbol{x}, \boldsymbol{s}) h(\boldsymbol{y}, \boldsymbol{s}) \mathrm{d}\boldsymbol{s}. \tag{4}$$

Under sparse convolution approximation [Lagae et al. 2009; Lewis 1989; Xu et al. 2025], Eq. (3) becomes:

$$\psi(\boldsymbol{x}) = \sum_i w_i h(\boldsymbol{x}, \boldsymbol{q}_i), \tag{5}$$

where $\boldsymbol{q}$ and w denote the position and weight of the impulse, respectively. Impulse positions $\{\boldsymbol{q}_i\}$ are uncorrelated and drawn from a Poisson point process with density λ (also refer to as impulse density), and i.i.d. random weights $\{w_i\}$ with $\mathbb{E}[w] = 0$ and $\mathbb{E}[w^2] = \frac{1}{\lambda}$.

To enable efficient evaluation of Eq. (5), previous studies [Lagae et al. 2009; Xu et al. 2025] typically introduce a uniform grid with a cell size equal to the kernel truncated radius. The noise is then evaluated using only the impulses within the cell containing the query point and its adjacent cells. Impulses in each cell are generated on the fly using a random seed derived from the cell index, combined with a realization-dependent seed offset.

Furthermore, the number of impulses per cell can be fixed to a constant N [Tavernier et al. 2019; Xu et al. 2025], and we use this convention for the rest of the paper. Tavernier et al. [2019] also investigated the distributions of impulse weights and positions for 2D Gabor noise, which can be applied to sparse convolution noise in our context. Importantly, Tavernier et al. [2019] introduced impulses with Bernoulli weights and a jittered grid distribution of positions.

Table 1. List of important symbols.

Symbol	Description
$\psi(\cdot)$	Realization of the Gaussian process
μ	Mean function
κ	Correlation function
$\boldsymbol{h}$	Convolution kernel
N	Maximum number of impulses per cell
C	Cell size
c	Subcell size
λ	Impulse density
$\boldsymbol{p}$	Marching point
$\boldsymbol{q}$	Impulse position
ψ_i	Contribution from i-th impulse
w	Weight of the impulse
S	Sample interval of the impulse
d	Impulse–marching-point distance for minimal contribution
$\underline{\psi}(\cdot)$	Lower bound function defined for $\psi(\cdot)$
$\underline{\psi}_i(\cdot)$	Lower bound on the contribution of i-th impulse
B	Bound map
$\mathbf{s}$	Binary sequence of impulse weights
$\mathbf{b}$	Binary sequences block
M	Block size
o	Offset of the block within the sequence

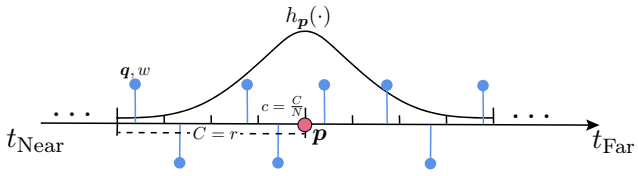


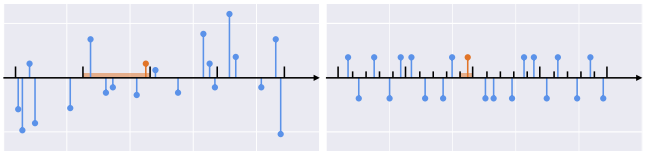
Fig. 2. Visual explanation of the important symbols for 1D sparse convolution noise. The ray segment from t_{Near} to t_{Far} is partitioned into cells of size $C = r$, where r is the kernel truncated radius. Each cell is subdivided into N subcells of size $c = \frac{C}{N}$, and a single impulse is sampled within each subcell.

We will make further explanation and show how such impulses can be encoded in binary form to facilitate precomputation in Sec. 4.

3.3 Light Transport on GPIS

A Gaussian process implicit surface (GPIS) is the distribution of zero-level sets of a Gaussian process. Each realization $f \sim \mathcal{GP}(\mu, \kappa)$ generates a implicit surface at the zero level set $\{\boldsymbol{x} \mid f(\boldsymbol{x}) = 0\}$.

Seyb et al. [2024] defined light transport on GPIS as the ensemble average of light transport over all realizations. Building on this theory, they propose a rendering framework that is well-suited for integration into path tracing. Specifically, GP realizations are sampled incrementally along each ray traversal. From the ray origin to the endpoint, they perform ray marching under the function-space view of Gaussian processes, sampling a multivariate Gaussian at the marching points and detecting ray–surface intersections via ray marching. The realization sampled for the current path segment is then stored and used as conditioning information when sampling realizations for subsequent path segments. To prevent the computational cost from growing unboundedly as memory accumulates during path tracing, Seyb et al. [2024] introduced the *Renewal+* memory model, which retains only the intersection point and its corresponding gradient from the last path segment. We give a more in-depth derivation of the light transport equation of Seyb et al. [2024] in Appendix A.



(a) Gaussian Impulses (b) Stratified Bernoulli Impulses

Fig. 3. Illustrating Gaussian impulses and our proposed SBIs. Stick weights correspond to impulse weights. We use the orange color to highlight a particular impulse and its sample interval. For a Gaussian impulse, the sample interval covers the entire cell, while for SBI, it is confined to the corresponding subcell.

3.4 Ray Marching with 1D Sparse Convolution Noise

Seyb et al. [2024] employed ray marching with the function-space view, multivariate Gaussian sampling, which attains a time complexity of $O(n^3)$, where n is the number of marching points. Xu et al. [2025] demonstrated that ray marching with the weight-space view, sparse convolution noise, provides a more efficient alternative: the $O(n^3)$ Gaussian sampling can be replaced with the $O(Nn)$ sparse convolution noise evaluations. Moreover, since only the realization values along the ray traversal are required, 1D sparse convolution noise offers the most efficiency [Xu et al. 2025]. For 1D sparse convolution noise, Ω reduces to the sub-domain along the ray, and the uniform grid used for efficient evaluation simplifies to a uniform partition of the ray into cells, each corresponding to a ray interval, with impulses sampled independently within each cell. Some important symbols are listed in Table 1 and visually explained in Figure 2.

Beyond sampling realizations, conditioning is also necessary in Seyb et al. [2024]'s framework. Unfortunately, classical GP conditioning relies on manipulating the mean and covariance for Gaussian sampling, which is not applicable for 1D sparse convolution noise. Thus, path-wise conditioning is employed [Wilson et al. 2021; Xu et al. 2025], which applies conditioning in two stages: at each marching point, we sample an unconditional realization and then apply a conditioning update afterward. With the *Renewal+* memory model, this update can be computed in constant time.

4 STRATIFIED BERNOULLI IMPULSES

In the sparse convolution noise approximation, a Gaussian process realization is determined by the positions and weights of sampled impulses. One possible choice for the weights, also adopted in Xu et al. [2025, Algorithm 1], is to use Gaussian weights. While this choice is consistent with the classical weight-space view of Gaussian processes [Williams and Rasmussen 2006], the Gaussianity of the sparse convolution arises from the central limit theorem [Kallenberg

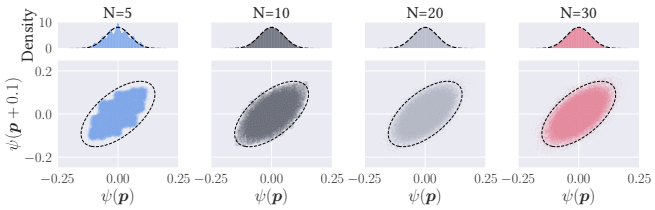


Fig. 4. Evaluating the Gaussianity of a 1D sparse convolution noise with SBIs. We employ a stationary squared-exponential kernel with a length scale of 0.1. We visualize the normalized 1-point distribution (top row) and the 2-point joint distribution (bottom row). As the number of impulses-per-cell N increases, it rapidly converges toward a Gaussian shape.

1997] rather than the Gaussian distribution of the weights. This allows us to move beyond Gaussian weights.

In this paper, we regularize the realizations, thereby classifying them into a finite number of categories and computing a bound for each of them. Specifically, we adopt impulses with only two weights, either $+\frac{1}{\sqrt{\lambda}}$ or $-\frac{1}{\sqrt{\lambda}}$, and place them in a stratified way within each cell. Such a design also appears in Tavernier et al. [2019], where it is referred to as impulses with Bernoulli weights and a jittered grid point distribution. In this paper, we refer to this design as *stratified Bernoulli impulses* (SBIs). Beyond the convergence properties discussed in Tavernier et al. [2019], we identify algorithmic advantages of SBIs: impulses follow a stratum-wise order instead of random placement, and we can encode their weights as binary 0/1 values corresponding to their signs. As in Lagae et al. [2009]; Xu et al. [2025], we uniformly divide the ray into cells. Then, we further split each cell into subcells, with one impulse in each subcell. Thus, 1D GP realizations can be classified by binary sequences over subcells, where each bit encodes the weights of all impulses in its subcell. This binary-sequence-based classification enables efficient bound computation and allows faster ray marching over GP realizations. Figure 3 explains these two types of impulses visually.

Validation. Now, we verify that 1D sparse convolution noise with SBIs is still a consistent approximation of a Gaussian process.

Our choice of SBIs preserves the derivations of Xu et al. [2025, Appendix C], ensuring that the covariance of the sparse convolution noise matches that of the target Gaussian process. In Figure 4, we show how the sparse convolution approximation rapidly converges to a Gaussian process as the impulse density increases.

Figure 5 further compares the convergence behavior of the sparse convolution approximation using SBIs and Gaussian impulses by measuring the Cramér–von Mises (CvM) statistic [1] [Anderson and Darling 1952] of the normalized 1-point distribution compared to the Gaussian distribution. As shown, an increase in impulse density results in a decrease in CvM, which indicates that the sparse convolution approximation with SBIs converges to the target Gaussian distribution as expected. Although SBIs exhibit a larger CvM statistic compared to Gaussian impulses, the impact is visually negligible in the final renderings even at a relatively low impulse density, as

[1]The Cramér–von Mises statistic measures the goodness of fit between an empirical CDF F_n and a target CDF F^*. For one sample case, $T = n\omega^2 = n\int_{-\infty}^{\infty}(F_n(x) - F^*(x))^2\,\mathrm{d}F^*(x)$. Larger values of T indicate greater discrepancy and thus a worse fit.

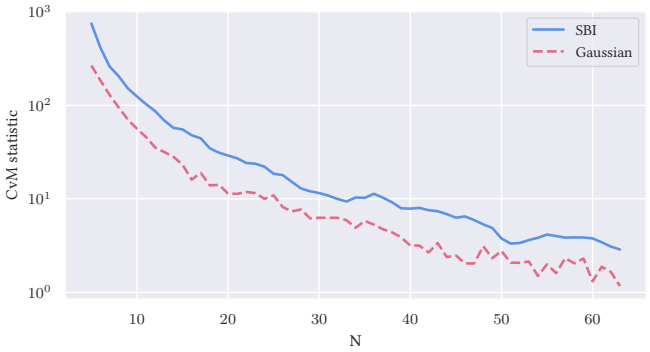


Fig. 5. Comparing the Cramér–von Mises (CvM) statistic of the normalized one-point distribution against a standard Gaussian distribution for the two impulse types, using 4 million samples for each impulses-per-cell setting. We employ the same covariance function as in Figure 4.

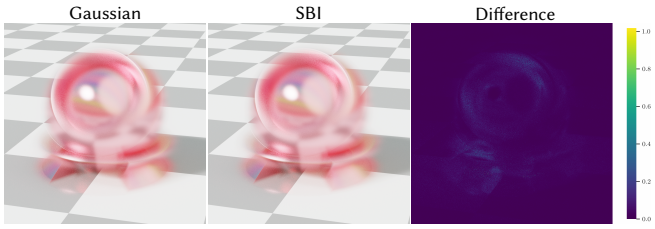


Fig. 6. Rendering the Knob scene featuring a GPIS with a stationary covariance function at 1024 spp. At $N = 10$, we compare renderings with Gaussian impulses and SBIs. The difference map shows the L2 norm of the difference, which verifies that SBIs maintain visually indistinguishable rendering even when CvM suggests suboptimal convergence under a low impulses-per-cell.

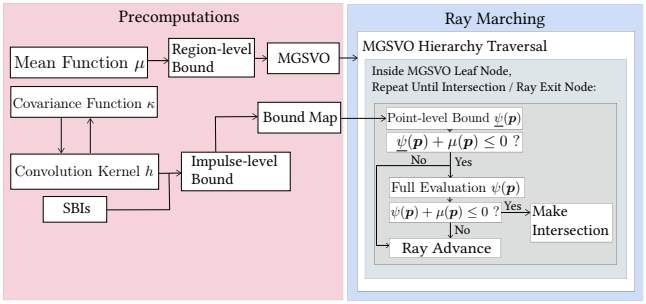


Fig. 7. An overview of bounded ray marching. For clarity, we assume the ray originates outside the surface, where the realization value is positive, and omit the path-wise updating term.

demonstrated in Figure 6. Moreover, as we will demonstrate later, SBIs enable precomputed bounds, which make it possible to use larger N for improved Gaussian coverage without significantly compromising rendering efficiency.

5 BOUNDED RAY MARCHING

Conventional ray marching methods evaluate sparse convolution noise at each marching point. However, such a precise evaluation

is often unnecessary since a conservative bound on the value suffices to guarantee non-intersection. This motivates us to develop efficient point-level bounds for individual realizations and region-level bounds across all realizations. Since evaluating these bounds is significantly cheaper than a full noise evaluation, we only perform the full evaluation when these bounds indicate that an intersection may exist. An overview of our pipeline is shown in Figure 7.

In what follows, we consider a ray from t_{Near} to t_{Far}, with cells of size C (Figure 2). This yields a total of KN impulses, where $K = \left\lceil \frac{t_{\text{Far}} - t_{\text{Near}}}{C} \right\rceil + 2$ is the maximum number of cells that contribute. Without loss of generality, we work with lower bounds.

5.1 Impulse-level Bound

Since a realization of sparse convolution noise is a sum over contributions from individual impulses, we can bound each impulse's contribution independently first and then obtain an overall bound:

$$\underline{\psi}(\boldsymbol{p}) = \sum_{i=1}^{i=KN} \underline{\psi}_i(\boldsymbol{p}), \tag{6}$$

where $\boldsymbol{p}$ is a marching point and $\underline{\psi}_i(\boldsymbol{p})$ is the lower bound of the contribution from i-th impulse along the ray.

Let $\boldsymbol{q}_i$ and w_i denote the position and weight of the i-th impulse along the ray, and let its sample interval S_i denote the segment of the ray over which this impulse is sampled. Let $h_{\boldsymbol{p}}(\boldsymbol{q})$ denote the convolution kernel at the marching point $\boldsymbol{p}$, and we assume it is *symmetric*, which is the case for many commonly used kernels (e.g., squared-exponential and Matérn kernels [Matérn 1960; Williams and Rasmussen 2006]). Therefore, $h_{\boldsymbol{p}}(\boldsymbol{q})$ depends only on the distance $\|\boldsymbol{p} - \boldsymbol{q}\|$ between each impulse and $\boldsymbol{p}$, i.e.,

$$h_{\boldsymbol{p}}(\boldsymbol{q}) = h_{\boldsymbol{p}}(\|\boldsymbol{p} - \boldsymbol{q}\|). \tag{7}$$

We assume $h_{\boldsymbol{p}}(\|\boldsymbol{p} - \boldsymbol{q}\|)$ to *decrease monotonically* with $\|\boldsymbol{p} - \boldsymbol{q}\|$. As a result, a negative impulse reaches its minimum contribution at the nearest point in S_i to $\boldsymbol{p}$. Likewise, a positive impulse reaches its minimum at the farthest point. We thus compute the distance d_i between $\boldsymbol{p}$ and S_i that yields the minimum contribution as:

$$d_i = \operatorname*{argmin}_{u \in \{\|\boldsymbol{p}-\boldsymbol{x}\|, \boldsymbol{x} \in \mathrm{S}_i\}} w_i h_{\boldsymbol{p}}(u) = \begin{cases} \min_{\boldsymbol{x} \in \mathrm{S}_i} \|\boldsymbol{p} - \boldsymbol{x}\|, & w_i < 0, \\ \max_{\boldsymbol{x} \in \mathrm{S}_i} \|\boldsymbol{p} - \boldsymbol{x}\|, & w_i \geq 0. \end{cases} \tag{8}$$

Then, we obtain the lower bound of the i-th impulse's contribution:

$$\boxed{\underline{\psi}_i(\boldsymbol{p}) = w_i h_{\boldsymbol{p}}(d_i).} \tag{9}$$

For stratified Bernoulli impulses, the i-th impulse is sampled in the i-th subcell. This allows us to compute the impulse's minimum and maximum distances to the marching point by considering the nearest and farthest subcell endpoints from the marching point, respectively (as shown in Figure 8). Since we aim to precompute bounds independent of the marching point's position, thereby reducing overall memory requirements, we conservatively assume marching points lie at one of the subcell endpoints. Using the above analysis, we can rewrite Eq. (8) as

$$d_i = \begin{cases} \max\left(0, \left(|i - i_{\boldsymbol{p}}| - 1\right) c\right), & w_i < 0, \\ \left(|i - i_{\boldsymbol{p}}| + 1\right) c, & w_i \geq 0, \end{cases} \tag{10}$$

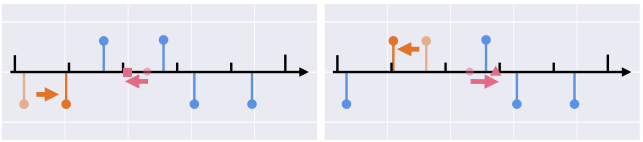
where $i_{\boldsymbol{p}}$ denotes the index of the subcell containing the marching point and $c = \frac{C}{N}$ is the subcell size.

5.2 Point-level Bound with Bound Maps

Building on the above impulse-level bound, we now precompute and populate a bound map (Figure 9) to accelerate retrieval of the overall lower bounds. We assume a *stationary* convolution kernel and a limited number (e.g., 16) of impulses-per-cell. Support for larger impulses-per-cell is discussed later in this subsection, while extensions for non-stationary kernels are presented in section 6.

Base bound map construction. Since we have assumed our convolution kernel is symmetric, we only precompute bounds for binary sequences encoding the impulse weights on the right side of the marching point and handle the left side using reversed sequences.

Since we set cell size C equal to the kernel truncated radius, each marching point receives contributions from N impulses on its right side, excluding the one in its own subcell. Thus, we design the base bound map B as an array of size 2^N. Each entry maps a binary sequence $\mathbf{s}$ of length N which encodes the weights of the impulses from the right side, to the lower bound of their contribution. We compute each entry of the bound map by summing the corresponding impulse-level lower bounds based on Eq. (6), considering only



(a) Negative Impulse (b) Positive Impulse

Fig. 8. Illustration of the construction of the impulse-level bound. In each subfigure, five subcells are arranged from left to right. The red point indicates the marching points, and d_i is evaluated for the orange-colored impulses. We show the movement of impulses and marching points to obtain d_i. A square marker is used for the marching point to denote its movement for negative impulses, while a triangular marker is used for positive impulses.

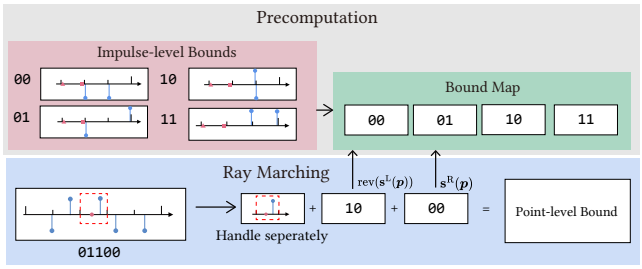


Fig. 9. Example of bound map construction and point-level bound computation. The red point denotes the marching point. Five impulses are considered in total, with two impulses on each side of the marching point. This results in four entries in the bound map for symmetric kernels. We precompute each entry by applying impulse-level bounds to each impulse, with the corresponding impulse weights determined by the binary sequence.

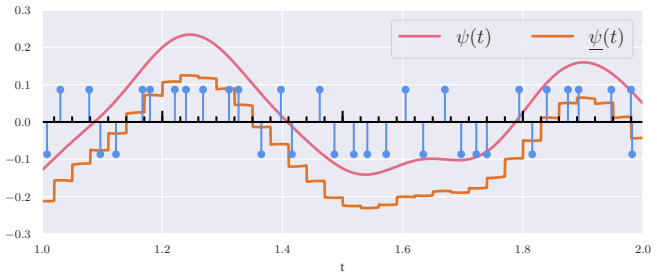


Fig. 10. Illustration of the point-level bound. We use a squared-exponential kernel with a length scale of 0.1 and an amplitude of 0.1, along with a normalization factor, resulting in a Gaussian process with a standard deviation of 0.1. We set the impulses-per-cell $N = 10$. Corresponding impulses are shown as sticks, with stick lengths scaled by a factor of 5 (i.e., stick length = impulse weight ×5) for clarity. Marching points within the same subcell correspond to an identical binary sequence, leading to plateau-like regions with a slight slope. In contrast, crossing a subcell boundary changes the binary sequence and causes a sudden variation in the bound.

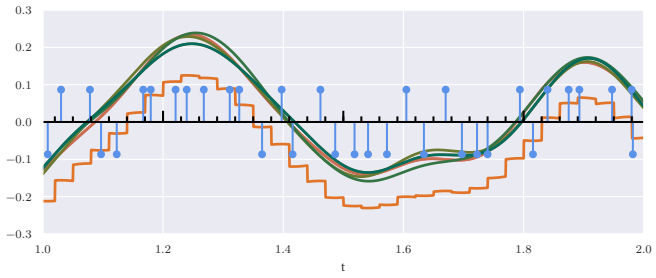


Fig. 11. Realizations sharing the same impulse weights correspond to the same point-level bound (omit the impulse within the same subcell of the marching point). The point-level (lower) bound is clearly lower than all realizations. The figure also indicates that the realization shape is more strongly affected by impulse weights than by impulse positions.

the N impulses nearby on the right side of the marching point:

$$B(\mathbf{s}) = \sum_{j=1}^{N} \underline{\psi}_{i_{\boldsymbol{p}}+j}(\boldsymbol{p}), \tag{11}$$

with bit $\mathbf{s}_i$ encoding the weight of $(i_{\boldsymbol{p}} + j)$-th impulse. Inserting Eq. (9) and Eq. (10) together into Eq. (11) yields:

$$B(\mathbf{s}) = \sum_{j=1}^{N} w_{i_{\boldsymbol{p}}+j} h\left(d_{i_{\boldsymbol{p}}+j}\right) = \frac{1}{\sqrt{\lambda}} \sum_{j=1}^{N} \begin{cases} -h\left((j-1)\,c\right), & \mathbf{s}_j = 0, \\ h\left((j+1)\,c\right), & \mathbf{s}_j = 1. \end{cases} \tag{12}$$

Hence, we can express the lower bound for $\boldsymbol{p}$ as

$$\boxed{\underline{\psi}(\boldsymbol{p}) = w_0 h(\|\boldsymbol{p} - \boldsymbol{q}_0\|) + B\left(\mathrm{rev}(\mathbf{s}^{\mathrm{L}}(\boldsymbol{p}))\right) + B\left(\mathbf{s}^{\mathrm{R}}(\boldsymbol{p})\right),} \tag{13}$$

where $\boldsymbol{q}_0, w_0$ denote the impulse's position and weight within the subcell of $\boldsymbol{p}$. We handle its contribution separately to avoid counting it twice. The binary sequences $\mathbf{s}^{\mathrm{L}}(\boldsymbol{p})$ and $\mathbf{s}^{\mathrm{R}}(\boldsymbol{p})$ encode the weights of impulses in the subcells to the left and right of the marching point, respectively. The reversal operator $\mathrm{rev}(\cdot)$ reverses the order of bits relative to the subcell order. Example illustrations of the resulting point-level lower bound is shown in Figure 10, Figure 11.

Scaling to long sequences. Until now, we have considered a restricted number of impulses within the kernel truncated radius. However, storing a single bound map for entire sequences becomes infeasible as N grows, due to its $O(2^N)$ space complexity. To support long sequences, we split the entire binary sequence into fixed-length blocks of size M (16 bits each in our implementation) and precompute bounds independently for each block. We then store them in the blocked bound map:

$$B(\mathbf{b}, o) = \frac{1}{\sqrt{\lambda}} \sum_{j=1}^{M} \begin{cases} -h\left((j-1+Mo)c\right), & \mathbf{b}_j = 0, \\ h\left((j+1+Mo)c\right), & \mathbf{b}_j = 1, \end{cases} \tag{14}$$

where $\mathbf{b}$ denotes the sequence block of length M and o is the block index within the full sequence.

For bound queries, we divide the binary sequence $\mathbf{s}$ of length N into blocks of length $M = \{\mathbf{b}^1, \ldots \mathbf{b}^{\lceil \frac{N}{M} \rceil}\}$, where $\mathbf{b}^o$ denotes the o-th block. We can then compute lower bound for $\mathbf{s}$ as $\sum_{o=1}^{\lceil \frac{N}{M} \rceil} B(\mathbf{b}^o, o)$.

When N is not divisible by M, we compute the sub-map for the final block $B\left(\cdot, \lceil \frac{N}{M} \rceil\right)$ using only the remaining $(N \bmod M)$ bits of the sequence, corresponding to the incomplete tail block:

$$B\left(\mathbf{b}, \left\lceil \frac{N}{M} \right\rceil\right) = \frac{1}{\sqrt{\lambda}} \sum_{j=1}^{N \bmod M} \begin{cases} -h\left((j-1+Mo)c\right), & \mathbf{b}_j = 0, \\ h\left((j+1+Mo)c\right), & \mathbf{b}_j = 1. \end{cases} \tag{15}$$

The space complexity is $O\left(\lceil \frac{N}{M} \rceil 2^M\right)$ and the time complexity of bound computation is $O\left(\lceil \frac{N}{M} \rceil\right)$. The parameter M provides a trade-off between memory footprint and speed.

Extensions to general kernels. So far, we have assumed stationary convolution kernels in the construction of bound maps. However, non-stationary kernels are often required in practice for modeling complex scenes. Since the non-stationary kernels are typically modeled by input-dependent parameters, we can discretize such parameter space and precompute the bound map for each parameter sub-space. For details, we defer discussions to Sec. 6.

5.3 Ray Marching with Point-level Bounds

To accelerate ray marching, we use point-level lower and upper bounds, computed in the same manner, to avoid unnecessary evaluations. We first test whether these bounds already rule out an intersection at each marching point:

$$\begin{cases} \underline{\psi}(\boldsymbol{p}) + \mu(\boldsymbol{p}) + U(\boldsymbol{p}) > 0, & \text{if } \boldsymbol{p} \text{ is outside the implicit surface,} \\ \bar{\psi}(\boldsymbol{p}) + \mu(\boldsymbol{p}) + U(\boldsymbol{p}) < 0, & \text{if } \boldsymbol{p} \text{ is inside the implicit surface,} \end{cases} \tag{16}$$

where $U(\boldsymbol{p})$ is the pathwise update term for GP conditioning [Wilson et al. 2021]. If the condition holds, we skip the costly evaluation. Otherwise, we evaluate $\psi(\boldsymbol{p})$ to check for an intersection. When a marching point indicates that an intersection occurs, we locate the intersection position by linearly interpolating between it and the previous marching point. If the previous marching point's full evaluation is skipped due to the point-level bound, we recompute its full evaluation for interpolation instead of using the point-level bound directly.

Note that each bound query depends on the binary sequences $\mathbf{s}^{\mathrm{L}}$ and $\mathbf{s}^{\mathrm{R}}$ defined around the current point p. As the ray advances, these sequences shift along the ray. To minimize redundant computation, we incrementally update $\mathbf{s}^{\mathrm{L}}$ and $\mathbf{s}^{\mathrm{R}}$ with a sliding window. As a result, the time complexity for generating these sequences is $O(KN)$. This corresponds to an amortized complexity of $O(\lceil \frac{\Delta s}{c} \rceil)$ per marching point, where Δs is the marching step size. In total, a point-level bound evaluation retains a time complexity of $O(\lceil \frac{\Delta s}{c} \rceil + \lceil \frac{N}{M} \rceil)$, in contrast to the $O(N)$ complexity incurred by a full evaluation.

5.4 Empty Pruning with Region-level Bounds

Ray marching with point-level bounds provides acceleration by skipping full noise evaluations when they are unnecessary. Although this reduces the average cost per marching point, the total number of marching points remains unchanged, as it still grows linearly with $t_{\mathrm{Far}} - t_{\mathrm{Near}}$, making it a significant computational bottleneck.

Probabilistically empty regions for GPIS. To reduce the total number of marching points, we discuss how to prune empty regions that are unlikely to intersect the surface, avoiding even the evaluation of the point-level bound. We can leverage spatial acceleration structures such as Volumetric Dynamic B+ tree (VDB) [Museth 2013] for deterministic implicit surfaces. For GPIS, however, we can hardly define the empty regions, as different realizations produce different empty regions. Thus, we instead consider *probabilistically empty regions*, as in Seyb et al. [2024], where the probability of containing intersections under any realization is negligible.

According to the three-sigma rule of the Gaussian distribution, for a point $\boldsymbol{x}$, we have

$$\mu(\boldsymbol{x}) - 3\sqrt{\kappa(\boldsymbol{x},\boldsymbol{x})} \le \psi(\boldsymbol{x}) \le \mu(\boldsymbol{x}) + 3\sqrt{\kappa(\boldsymbol{x},\boldsymbol{x})}, \tag{17}$$

holding with nearly 100% probability. Extending this pointwise bound to any region V, we have the region-level bound:

$$\underline{\mu}(V) - 3\bar{\sigma}(V) \le \psi(\boldsymbol{x}) \le \bar{\mu}(V) + 3\bar{\sigma}(V), \tag{18}$$

where $\underline{\mu}(V) = \min_{\boldsymbol{x}\in V} \mu(\boldsymbol{x})$ and $\bar{\mu}(V) = \max_{\boldsymbol{x}\in V} \mu(\boldsymbol{x})$ denotes the lower and upper bounds of the mean function over V, respectively. $\bar{\sigma}(V) = \max_{\boldsymbol{x}\in V} \sqrt{\kappa(\boldsymbol{x},\boldsymbol{x})}$ is the upper bound of the GP standard deviation within V. With this, we identify any region with $\underline{\mu}(V) > 3\bar{\sigma}(V)$ or $\bar{\mu}(V) < -3\bar{\sigma}(V)$ as a probabilistically empty region.

Mean-guided sparse voxel octree. To efficiently identify and skip the probabilistically empty regions, we propose the mean-guided sparse voxel octree (MGSVO), following the sparse voxel octree design of Laine and Karras [2010]. The MGSVO is primarily a sparse octree, with each node v corresponding to a voxel V_v. A node in the MGSVO is stored if and only if it is non-empty or has at least one non-empty sibling.

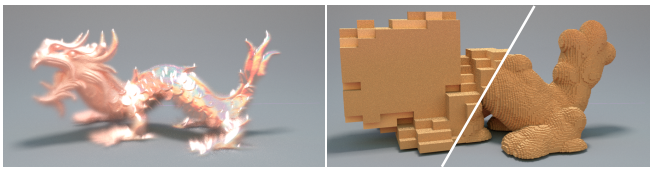


Fig. 12. 3D visualization showing all non-empty voxels for two distinct levels of MGSVO, built with rigorous bounds (Eq. (19), Eq. (20), and Eq. (21)).

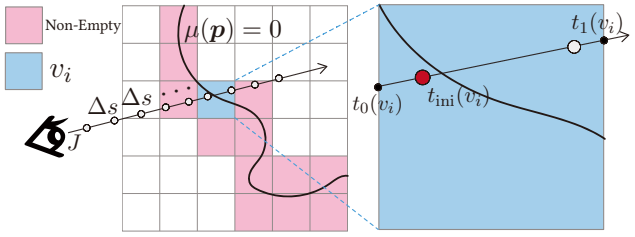


Fig. 13. 2D illustration of ray marching with MGSVO. For clarity, we omit the hierarchy traversal process and show only the finest level of the MGSVO.

We treat a node v as empty if $\underline{\mu}(V_v) > 3\bar{\sigma}(V)$ or $\bar{\mu}(V_v) < -3\bar{\sigma}(V)$. For the finest nodes, corresponding to the voxel $V_v = \boldsymbol{c} + [-\frac{e}{2}, \frac{e}{2}]^3$, where $\boldsymbol{c}$ is the voxel center and e is the voxel edge length, the most straightforward approach is to approximate $\underline{\mu}(v)$, $\bar{\mu}(v)$ and $\bar{\sigma}(v)$ by sampling multiple points inside the voxel. In particular, we can use the Lipschitz condition if available to obtain a rigorous bound, as in Barbier et al. [2025]: if μ is λ-Lipschitz continuous, i.e., $|\mu(x) - \mu(y)| \le \lambda\|x - y\|, \forall x, y$, the value of μ at any point inside the voxel can differ from the value at the voxel center $\boldsymbol{c}$ by at most λ times the distance from the center to that point. Since the farthest point from the center is a corner, at a distance of $\frac{\sqrt{3}e}{2}$, we obtain

$$\underline{\mu}(V_v) = \mu(\boldsymbol{c}) - \lambda\frac{\sqrt{3}e}{2}, \tag{19}$$

$$\bar{\mu}(V_v) = \mu(\boldsymbol{c}) + \lambda\frac{\sqrt{3}e}{2}, \tag{20}$$

which is especially useful when μ is a signed distance function (SDF), the most commonly used case, as SDFs are 1-Lipschitz [Evans 1998]. In contrast, the Lipschitz continuity of the GP standard deviation is often not explicitly available in practice. Nevertheless, a rigorous bound can still be obtained by falling back to a global upper bound.

$$\bar{\sigma}(v) = \max_{\boldsymbol{x}\in\mathbb{R}^3} \sqrt{\kappa(\boldsymbol{x},\boldsymbol{x})} = \bar{\sigma}. \tag{21}$$

In Figure 12, we provide a 3D visualization of all non-empty MGSVO nodes at two different levels.

Ray marching with MGSVO. We embed the MGSVO into ray marching by performing a hierarchy traversal [Laine and Karras 2010] over all non-empty leaf nodes. Each non-empty leaf node v_i corresponds to a ray sub-interval $[t_0(v_i), t_1(v_i)) \subseteq [t_{\mathrm{Near}}, t_{\mathrm{Far}}]$. We then perform ray marching with point-level bounds in the interval $[t_0(v_i), t_1(v_i))$. To mitigate aliasing caused by the voxel structure, rather than using $t_0(v_i)$ directly to determine the initial marching point, we first sample a uniform jitter $J \in [0, \Delta s]$ for the entire ray, which produces a jittered distance sequence $\{\, t_{\mathrm{Near}} + \Delta s\, j + J \mid j = 0, 1, \dots \}$. We then take the first one that falls inside $[t_0(v_i), t_1(v_i))$:

$$t_{\mathrm{ini}}(v_i) = t_{\mathrm{Near}} + \Delta s \left\lceil \frac{t_0(v_i) - t_{\mathrm{Near}} - J}{\Delta s} \right\rceil + J, \tag{22}$$

to compute the initial marching point for node v_i. We show a 2D illustration of ray marching with MGSVO in Figure 13.

Disentangled region-level bounds. The conventional approach for representing multiple GPIS objects in the same scene is to take the minimum over the GPIS realizations across all objects. One can construct an *entangled region-level bound* by aggregating the minimum mean function and the maximum standard deviation over all objects. However, this formulation only permits pruning of regions that are probabilistically empty with respect to all objects, which severely limits its pruning effectiveness.

Since different GPIS objects typically occupy distinct spatial regions with only partial overlap, we instead apply the region-level bound independently to each object, leading to *disentangled region-level bounds*. As a result, a given region may be probabilistically empty for some objects even if it is not empty in the physical sense. Leveraging this observation, we maintain, for each MGSVO leaf node, a list of GPIS objects that are not probabilistically empty within that node. During querying, evaluations of these GPIS objects are pruned, avoiding the need to consider all objects.

6 Extensions to General Kernels

In this section, we show how bound maps can be extended for non-stationary kernels by adding more dimensions. For clarity, we work with the base bound map in the following discussion, and the same applies to the blocked bound map.

6.1 Parametric non-stationary kernels

A common way to model non-stationary kernels is to introduce input-dependent kernel parameters [Heinonen et al. 2016; Noack et al. 2024]:

$$h(\boldsymbol{x}, \boldsymbol{y}) = h(\boldsymbol{x}, \boldsymbol{y}; \theta(\boldsymbol{x}, \boldsymbol{y})) \tag{23}$$

where $\boldsymbol{x}$ corresponds to the kernel center. $\theta(\boldsymbol{x}, \boldsymbol{y})$ denotes the input-dependent kernel parameters.

6.2 Local parametric non-stationary kernels

Unfortunately, when the kernel parameters depend on $\boldsymbol{y}$, the sparse convolution noise depends not only on the impulses' relative positions with respect to the marching point, but also on the impulses' absolute position, which affects kernel parameters. To facilitate bound precomputation, we assume that the kernel parameters depend only on the kernel center (marching points), i.e., $\theta(\boldsymbol{x}, \boldsymbol{y}) = \theta(\boldsymbol{x})$, and we refer to this type of kernel as local parametric non-stationary kernels.

Isotropic kernels with monotonic parameter dependence. If the kernel is further isotropic, we have

$$h(\boldsymbol{x}, \boldsymbol{y}; \theta(\boldsymbol{x})) = h(\|\boldsymbol{x} - \boldsymbol{y}\|; \theta(\boldsymbol{x})). \tag{24}$$

For clarity of discussion, we begin with the case where $\theta(\boldsymbol{x}) \in \mathbb{R}$ only has a single scalar parameter. We further assume that $h(\boldsymbol{x}, \boldsymbol{y}, \theta)$ is monotonic in θ and isotropic with fixed θ.

We incorporate θ into the bound map by discretizing its range into a set of equally spaced values $\Theta = \{\theta_1, \theta_2, \dots \theta_{\max}\}$ (max = 16 in our implementation) and include the index of these values as an additional dimension, i.e., $B(i(\theta), \mathbf{s})$, where $\theta_{i(\theta)}, \theta_{i(\theta)+1}$ bracket θ.

Without loss of generality, we assume h increases with θ. Therefore, we have the bounds for the kernel centered at $\boldsymbol{x}$:

$$h\left(\|\boldsymbol{x} - \boldsymbol{y}\|; \theta_{i(\theta(\boldsymbol{p}))}\right) \le h\left(\|\boldsymbol{x} - \boldsymbol{y}\|; \theta(\boldsymbol{p})\right) \le h\left(\|\boldsymbol{x} - \boldsymbol{y}\|; \theta_{i(\theta(\boldsymbol{p}))+1}\right), \tag{25}$$

which becomes tighter as $|\Theta|$ increases. We can then precompute the bound map using such bounds as follows:

$$B(k, \mathbf{s}) = \sum_{j=1}^{N} \frac{1}{\sqrt{\lambda}} \begin{cases} -h\left((j-1)c; \theta_{k+1}\right), & \mathbf{s}_j = 0, \\ h\left((j+1)c; \theta_k\right), & \mathbf{s}_j = 1. \end{cases} \tag{26}$$

During query, we first compute $i(\theta(\boldsymbol{p}))$ and then compute the bounds in a similar manner as Eq. (13):

$$\begin{aligned} \underline{\psi}(\boldsymbol{p}) =& w_0 h(\|\boldsymbol{p} - \boldsymbol{q}_0|; \theta(\boldsymbol{p})) + B\left(i(\theta(p)), \mathrm{rev}(\mathbf{s}^{\mathrm{L}}(\boldsymbol{p}))\right) + \\ & B\left(i(\theta(p)), \mathbf{s}^{\mathrm{R}}(\boldsymbol{p})\right) \end{aligned} \tag{27}$$

For multi-parametric, non-stationary kernels, we add separate dimensions to the bound map for each kernel parameter, discretizing each dimension independently and processing them in the same way as the single-parameter case.

Anisotropic kernels. Anisotropies are typically defined using a matrix $\Sigma(\boldsymbol{x})$:

$$h(\boldsymbol{x}, \boldsymbol{y}; \theta(\boldsymbol{x})) = h\left((\boldsymbol{x} - \boldsymbol{y})^\top \Sigma(\boldsymbol{x})(\boldsymbol{x} - \boldsymbol{y}); \theta(\boldsymbol{x})\right) \tag{28}$$

For a ray with direction $\boldsymbol{d}$, we can define the effective length-scale along the ray with direction $\boldsymbol{d}$ as

$$l_{\mathrm{eff}}(\boldsymbol{d}) = \frac{1}{\sqrt{\boldsymbol{d}^\top \Sigma^{-1}(\boldsymbol{x}) \boldsymbol{d}}}. \tag{29}$$

We can then treat the effective length scale itself as a single parameter and rewrite the kernel as

$$h(\boldsymbol{x}, \boldsymbol{y}; \theta(\boldsymbol{x})) = h(\|\boldsymbol{x} - \boldsymbol{y}\|; l_{\mathrm{eff}}(\boldsymbol{d}), \theta(\boldsymbol{x})) = h\left(\frac{\|\boldsymbol{x} - \boldsymbol{y}\|^2}{l_{\mathrm{eff}}^2(\boldsymbol{d})}; \theta(\boldsymbol{x})\right) \tag{30}$$

Based on this formula, we precompute the bound map as in Eq. (26) and query it using the effective length-scale computed on the fly.

Kernel decomposition. Note that we often decompose a parametric non-stationary kernel into a scalar and an unscaled part, i.e.,

$$h(\boldsymbol{x}, \boldsymbol{y}; \theta(\boldsymbol{x})) = N(\theta(\boldsymbol{x}))\, \hat{h}(\boldsymbol{x}, \boldsymbol{y}; \theta(\boldsymbol{x})), \tag{31}$$

where the unscaled part $\hat{h}$ often exhibits a more explicit monotonic dependency on the parameters. The most common case in our context is when $N(\theta(\boldsymbol{x}))$ serves as a normalization factor, scaled by an amplitude parameter, to ensure that the integral of h yields the desired covariance function. In such cases, we can construct the bound map using only $\hat{h}$ and scale by $N(\theta(\boldsymbol{x}))$ at query time.

6.3 Bound map for multi-resolution sparse convolution noise

The procedural evaluation of sparse convolution noise introduces a uniform grid with a cell size equal to the kernel truncated radius [Lagae et al. 2009; Xu et al. 2025]. This enables evaluating the noise using only impulses within the cell of the query point and direct neighboring cells. However, in the non-stationary case, the uniform grid cell size must be the maximum kernel radius, which forces a

larger impulses-per-cell to meet the impulse density requirement for smaller kernels.

Like Xu et al. [2025], we use a multi-resolution grid following Lagae et al. [2011] to address this increase in impulses-per-cell. Specifically, the grid with resolution of res has a cell size $C_{\text{res}} = 2^{\text{res}} r_{\min}$, where $r_{\min}$ is the minimal kernel truncated radius. The noise evaluation $\psi(\boldsymbol{p})$ then becomes a linear combination of noises with same impulses-per-cell on two consecutive grid resolutions $\text{res}_0, \text{res}_1 (\text{res}_1 = 2\text{res}_0)$, whose cell sizes bracket the kernel truncated radius at $\boldsymbol{p}$:

$$\psi_{\text{multi}}(\boldsymbol{p}) = \alpha_{\text{res}_0}(\boldsymbol{p})\psi_{\text{res}_0}(\boldsymbol{p}) + \alpha_{\text{res}_1}(\boldsymbol{p})\psi_{\text{res}_1}(\boldsymbol{p}) \tag{32}$$

where $\alpha_{\text{res}_0}(\boldsymbol{p})$ and $\alpha_{\text{res}_1}(\boldsymbol{p})$ are weight functions.

We bound ψ_{multi} by considering the contributions from different resolutions separately. This requires bounds from different resolutions, so we include the resolution as an additional dimension in the bound map, i.e., $B(i(\theta), \mathbf{s}, \text{res})$. Hence, we rewrite Eq. (27) as:

$$\begin{aligned}\underline{\psi}_{\text{res}}(\boldsymbol{p}) =& w_0^{\text{res}} h(\|\boldsymbol{p} - \boldsymbol{q}_0^{\text{res}}|; \theta(\boldsymbol{p})) + B\left(i(\theta(p)), \text{rev}(\mathbf{s}^{\text{L}}(\boldsymbol{p})), \text{res}\right) + \\ & B\left(i(\theta(p)), \mathbf{s}^{\text{R}}(\boldsymbol{p}), \text{res}\right)\end{aligned} \tag{33}$$

where w_0^{res} and $\boldsymbol{q}_0^{\text{res}}$ denote the impulse weight and position for the grid at resolution res.

Since the querying resolution depends on the spatially varying parameters, it is not necessary to precompute bounds with the full range of θ at every specific resolution. Instead, at each resolution, we precompute bounds only with the subset of θ values that are relevant, which yields a tighter bound within the same memory usage.

Notably, when only the non-stationary length scale l is present, Xu et al. [2025] introduces further approximation, treating ψ_{res_0} and ψ_{res_1} as two noises with stationary kernels of different length-scales. Under this approximation, both subcell size and kernel length-scale are scaled by 2^{res}:

$$c_{\text{res}} = \frac{C_{\text{res}}}{N} = \frac{r_0 2^{\text{res}}}{N} = c_0 2^{\text{res}}, \tag{34}$$

$$l_{\text{res}} = l_0 2^{\text{res}}, \tag{35}$$

where $c_0 = \frac{r_0}{N}, l_0$ denote the minimum subcell size and length-scale. Then the influence of parameters and resolution in the bound map computation cancels out:

$$B(k, \mathbf{s}, \text{res}) = \sum_{j=1}^{N} \frac{1}{\sqrt{\lambda}} \begin{cases} -h\left(\frac{(j-1)c_{\text{res}}}{l_{\text{res}}}\right) = -h\left(\frac{(j-1)c_0}{l_0}\right), & \mathbf{s}_j = 0, \\ h\left(\frac{(j+1)c_{\text{res}}}{l_{\text{res}}}\right) = h\left(\frac{(j+1)c_0}{l_0}\right), & \mathbf{s}_j = 1. \end{cases} \tag{36}$$

Thus, in this case, we do not need to introduce additional dimensions.

7 RESULTS

We have implemented our proposed method in the Tungsten renderer [Bitterli 2018], building on the GPIS implementation of Seyb et al. [2024]. All experiments are conducted on a PC equipped with an Intel Core i9-12900K CPU (16 cores, 4.5GHz). In what follows, we refer to Seyb et al. [2024]'s method as the *function-space method*, and Xu et al. [2025]'s method, with Gaussian impulses, as the *1D sparse-convolution method*. We use 1024 spp renderings from the 1D sparse-convolution method as references. Next-event estimation (NEE) is enabled when generating the references, while disabled when making comparisons among the three methods to focus on the speedup from the improvement in ray–surface intersections. We evaluate the function-space method [Seyb et al. 2024] with a batch size of 8 marching points and fixed object bounding boxes, as in Xu et al. [2025], and evaluate the 1D sparse-convolution method and our bounded ray marching method, both employing the multi-resolution approximation Eq. (32). Unless otherwise specified, we set impulses-per-cell $N = 10$. We always employ rigorous bounds (Eq. (19), Eq. (20), and Eq. (21)) for $\bar{\mu}$, $\underline{\mu}$, and $\bar{\sigma}$. The maximum depth of MGSVO is chosen to ensure that leaf nodes are no smaller than a single marching step size (set to 8 in our experiments). For each method, we compute the mean squared error (MSE) against the references. We take the function-space method as the baseline and estimate speedup by the ratio of MSEs.

Table 2. Spatial variation of non-stationary parameters for each scene, normalized by object bounding diameter and mean-function scaling.

Scene	Amplitude	Length Scale
Dragon	0.02%–0.61% (top → bottom)	0.6%–4.5% (left → right)
Lion (Left)	0.57%–1.44% (top → bottom)	9.2%–28.9% (top → bottom)
Lion (Right)	0.06%–1.44% (top → bottom)	9.2%–2.0% (top → bottom)
Cloud	0.45%–1.5% (bottom → top)	1.5%–15.2% (bottom → top)

Speedup. We use equal-time (10 min) renderings to compare the function-space method [Seyb et al. 2024], the 1D sparse-convolution method [Xu et al. 2025], and our bounded ray marching method. As shown in Figure 14, under a 10-minute equal-time setting, the function-space method yields highly noisy renderings. The 1D sparse-convolution method achieves substantial speedups over the function-space method. Furthermore, by eliminating unnecessary full noise evaluations, our method achieves significantly higher speedups compared to the 1D sparse-convolution method. Notably, the speedups achieved by bounded ray marching vary across scenes. This variation arises because our method tends to skip marching points with a higher probability of no intersection, and the number of such marching points differs between scenes. Consequently, the resulting speedups are scene-dependent. Table 2 summarizes the spatial variation of the non-stationary parameters for each scene.

We also profile with the statistics collected at 1 spp. As shown in Table 3, ray–surface intersections, including normal samplings, dominate the runtime. The 1D sparse-convolution method leads to many full noise evaluations, which is a major bottleneck of ray intersections. Our bounded ray marching method significantly reduces the full noise evaluations, thereby accelerating ray–surface intersections. Moreover, our bounded ray marching also skips some unnecessary mean function evaluations due to the region-level bound, providing additional acceleration.

Impulses-per-cell. In Figure 15, we compare the number of samples per pixel (spp) in equal time. While the computational cost of evaluating sparse convolution noise scales linearly with the impulses-per-cell N, the cost of our bound computations is largely insensitive to N. This is because traversing the MGSVO does not depend on

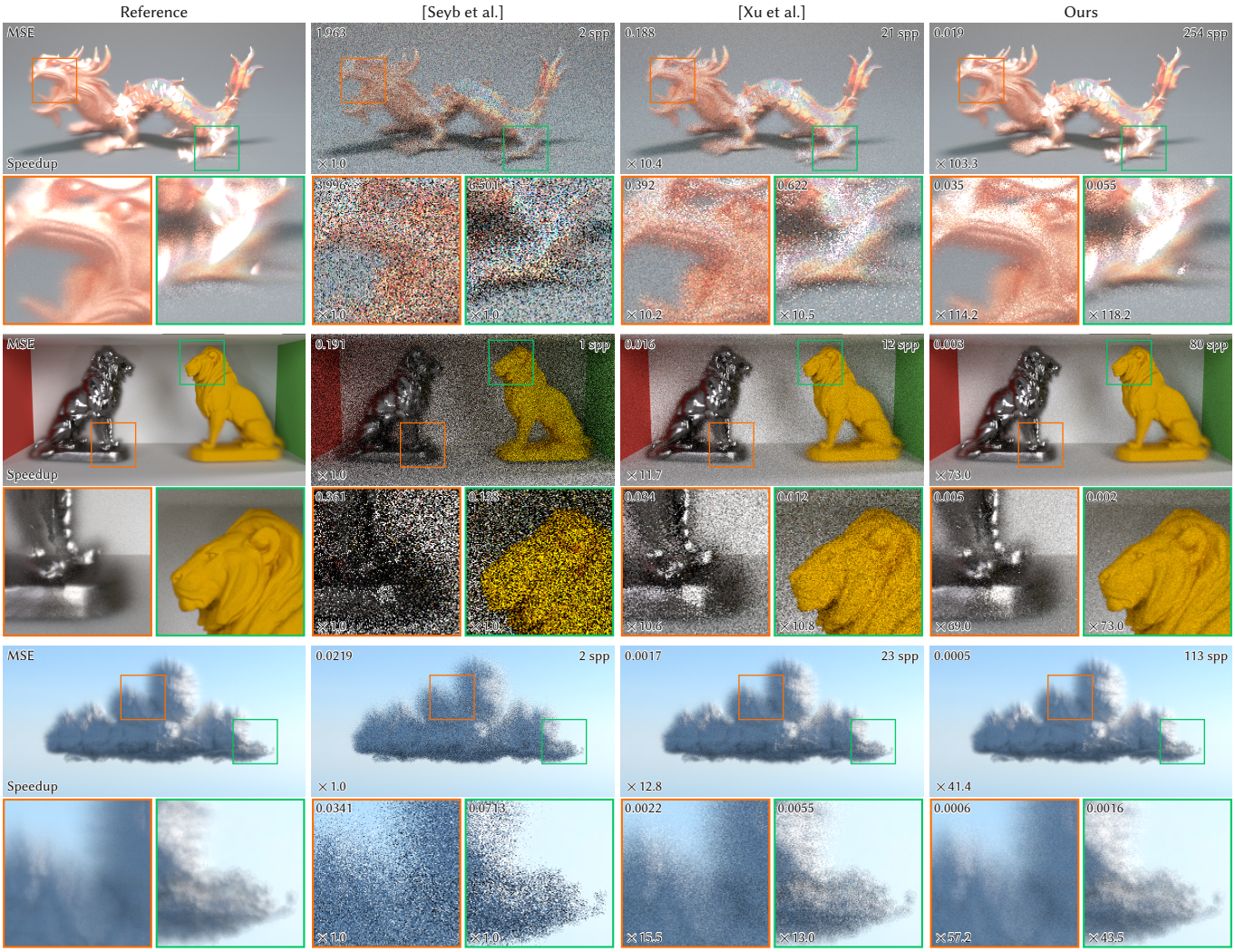


Fig. 14. We evaluate the function-space method [Seyb et al. 2024], the 1D sparse-convolution method [Xu et al. 2025], and our bounded ray marching method by comparing equal-time (10 min) renderings. The spp and speedup for the full image and for image patches are shown in the corners. We employ SDF mean functions and non-stationary squared-exponential covariance functions (and thus the convolution kernels) in all three scenes. We refer to the three scenes from top to bottom as Dragon, Lion, and Cloud.

N, and querying the bound map scales linearly with $\frac{N}{M}$ without requiring explicit kernel evaluations, often making it even cheaper than other overheads. These factors allow our method to scale better with respect to N than the 1D sparse-convolution method. As a result, we can increase N to improve the approximation accuracy without significantly reducing efficiency.

Ablation study. We evaluate the performance impact of replacing Gaussian impulses with SBIs. As shown in Figure 16, using SBIs only yields slightly better performance than Gaussian impulses, which is due to the lower computational cost of sampling a Bernoulli variable compared to a Gaussian one.

We then evaluate the speedup obtained by enabling only the point-level bound or only the region-level bound. Both bounds individually provide substantial speedup over Seyb et al. [2024]'s method. As shown in Figure 17, the point-level bound already offers a significant speedup, while the region-level bound yields an even larger improvement, as it directly prunes probabilistically empty regions in advance. With a relatively small impulses-per-cell $N = 10$, combining both bounds yields only a marginal improvement over using the region-level bound alone in our experiment. Nevertheless, as N increases, the point-level bound yields a significant additional speedup on top of the region-level bound alone.

For a closer look at the bound contributions, we further conduct a runtime profiling in Figure 18. With the point-level bound, we can already significantly reduce the computational cost of full noise evaluations. However, the evaluation of the mean function and other overheads subsequently emerges as the new bottleneck. By

Table 3. Rendering statistics. We count the marching points that perform full noise evaluations or are skipped by the point-level bound. We estimate runtime percentages relative to the total overall runtime for (1) ray–surface intersections, which subsume (2) full noise evaluations, (3) point-level (PL) bound evaluations, and (4) mean function evaluations. We also report the memory usage of the bound map and MGSVO. All statistics were collected at 1 spp.

Scene	Method	#Marching Points (M)		Runtime Breakdown				Memory (MB)		Rendering Time (s)
		Full Eval.	PL Bound Skip	Ray–Surf Inter.	Full Eval.	PL Bound	Mean Func.	Bound Map	MGSVO	
Dragon	[Xu et al.]	180.70 (100.0%)	0.00 (0.0%)	99.61%	74.12%	0.00%	17.00%	0.00	0.00	31.33
	Ours	4.77 (2.6%)	7.43 (4.1%)	96.84%	26.39%	11.81%	20.47%	32.00	20.41	2.28
Lion	[Xu et al.]	169.69 (100.0%)	0.00 (0.0%)	99.47%	70.69%	0.00%	19.96%	0.00	0.00	29.73
	Ours	8.48 (5.0%)	19.57 (11.5%)	97.72%	22.27%	14.77%	23.54%	24.00	58.89	4.64
Cloud	[Xu et al.]	158.04 (100.0%)	0.00 (0.0%)	99.78%	70.28%	0.00%	14.71%	0.00	0.00	28.43
	Ours	13.36 (8.6%)	25.37 (16.0%)	99.69%	26.66%	16.06%	20.55%	40.00	30.51	5.68

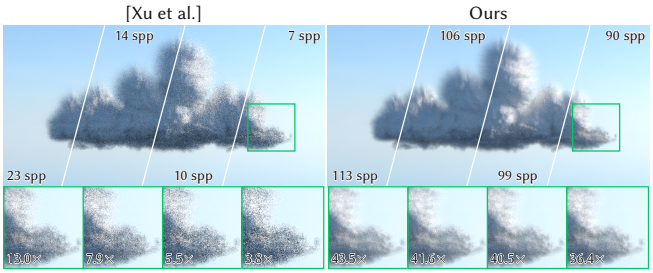


Fig. 15. Equal-time (10 min) renderings of two methods: the 1D sparse-convolution (left) and our bounded ray marching variant (right). Each method is shown as a horizontally split image, where each segment corresponds to a different number of impulses per cell: $N = 10, 20, 30, 40$ from left to right. Cropped regions are shown below for closer comparison, and we slightly abuse "speedup" to show the final impact on rendering quality.

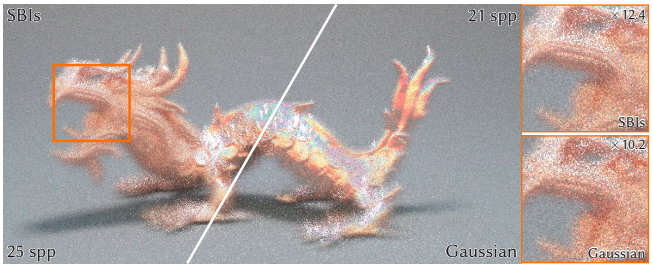


Fig. 16. Without bounds, replacing Gaussian impulses with SBIs leads to only a marginal improvement, which arises from the relatively lower cost of sampling Bernoulli variables compared to Gaussian ones.

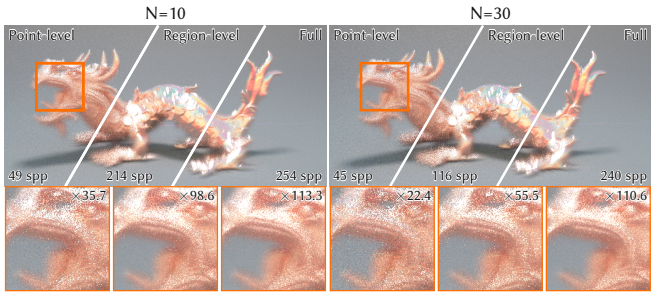


Fig. 17. Comparing equal-time (10 min) renderings of the Dragon scene using full bounded ray marching, only point-level bound, and only region-level bound, within two different impulses-per-cell settings ($N = 10$, $N = 30$). Both bounds offer substantial speedup against Xu et al. [2025].

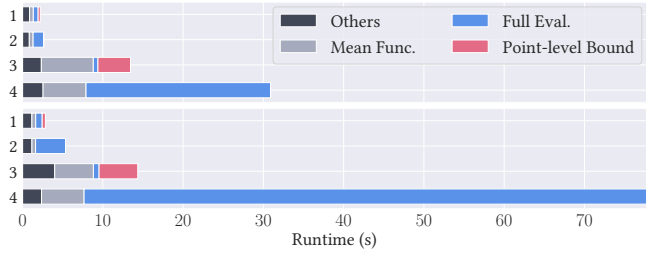


Fig. 18. Visualizing runtime breakdowns of a 1 spp rendering for the scene Dragon, with two impulses-per-cell settings: $N = 10$ (top) and $N = 30$ (bottom). Within each sub-figure, the bars from top to bottom correspond to the following methods: (1) full bounded ray marching, (2) region-level only variant, (3) point-level only variant, and (4) 1D sparse convolution.

leveraging the region-level bound, we can prune not only full noise evaluations but also mean function evaluations and pathwise updates, leading to even greater efficiency. When N is small, since full noise evaluation is no longer a major bottleneck, further applying the point-level bound yields only marginal additional improvement. With increased N, the computational cost of full noise evaluations escalates, re-emerging as the dominant bottleneck, and the point-level bound can once again provide noticeable performance gains.

We also evaluate the advantage of the disentangled region-level bounds over the entangled region-level bounds. As shown in Figure 19, while the entangled region-level bound already delivers substantial speedups, applying the bound in a disentangled manner to each GPIS yields further performance gains.

Impact of covariance. In Figure 20, we illustrate how, under the same mean function, different covariance functions (and thus different convolution kernels) impact the rendering efficiency of our method. When the (co)variance is very small, the resulting uncertainty is negligible, and only regions very close to the surface are considered probabilistically non-empty by the region-level bounds. Hierarchical traversal on the MGSVO then advances the ray directly to the vicinity of the surface, after which only a minimal number

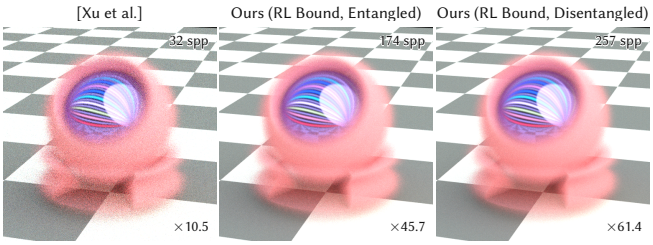
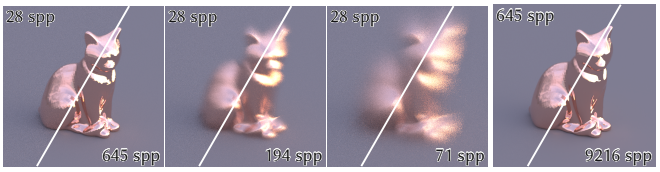


Fig. 19. Equal-time (10 min) rendering comparison of a scene composed of the overlap of two GPISes. The inner and outer parts correspond to different GPISes with different underlying BRDFs and covariance functions.



(a) [Xu et al.] vs. Ours (b) Ours vs. Mesh

Fig. 20. (a) Equal-time (3 min) comparison between the 1D sparse-convolution method (left of the split) and our method (right of the split). The three split images show scenes with identical mean functions, yielding similar geometric structures, but with increasing (co)variance from left to right. (b) Comparison between our method and the traditional path tracing on mesh-based representations when (co)variance is very small.

Table 4. Statistics of marching points and MGSVO memory usage for Figure 20 collected from 1-spp renderings. Rows correspond to scenes shown from left to right in the figure. We categorize marching points into those performing full noise evaluations, those pruned by the region-level (RL) bound, and those skipped by the point-level (PL) bound.

# Marching Points (M)			MGSVO (MB)
Full Eval.	PL Bound Skip	RL Bound Prune	
0.40 (0.6%)	0.35 (0.5%)	61.36 (98.8%)	3.28
5.36 (8.8%)	2.41 (3.9%)	53.13 (87.2%)	22.07
19.64 (33.5%)	10.08 (17.2%)	28.8 (49.2%)	66.70

of marching points remain to be evaluated. As the covariance increases, probabilistically empty regions shrink, and the discrepancy between precise evaluations and point-level bounds grows. Consequently, the efficiency of our method decreases. We also report the corresponding statistics in Table 4.

8 LIMITATION AND FUTURE WORK

Assumptions. The derivation of the point-level bound relies on a set of assumptions on the convolution kernel h: symmetry, monotonic decay with distance, and, for non-stationary kernels, local parametric with monotonic dependence on the parameters. These assumptions restrict the class of convolution kernels within our framework. Nevertheless, future work could relax these constraints, e.g., by using two bound maps, one for impulses in front of the marching point along the ray and one for those behind, to handle non-symmetric kernels. General-purpose bounding techniques like interval/affine arithmetics [Knoll et al. 2009; Velázquez-Armendáriz et al. 2009], polynomial bases [Fan et al. 2025], and neural networks [Liu et al. 2024] could also be employed.

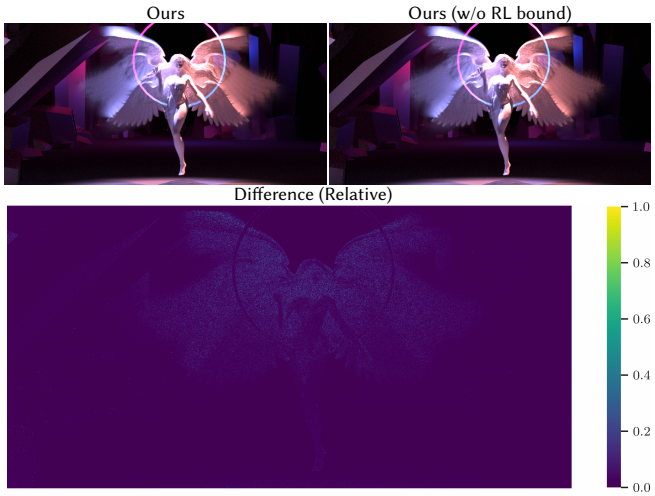


Fig. 21. High-spp (2048 spp) rendering comparison between our method and its variant without region-level bound. The results are visually indistinguishable. The difference map shows the L2 norm of the difference normalized by the L2 norm of the reference (Ours).

Bias. The region-level bound is a high-confidence bound rather than a strict one, which theoretically introduces bias. Nevertheless, in practice, we did not observe significant visual impacts in our experiments, as shown in Figure 21. The construction of the MGSVO requires bounds of the mean function μ within the voxel corresponding to each node. When μ satisfies a Lipschitz condition, we can derive rigorous bounds (Eq. (19) and Eq. (20)). Otherwise, while a global upper bound is available for $\bar{\sigma}$, we could fall back to bounding the mean function via a sampling approximation, which may become unreliable when μ exhibits high-frequency variations. Furthermore, our disentangled region-level bounds effectively treat the probabilistically empty regions of multiple objects independently. However, mathematically, overlapping low-probability occupancy regions across many objects can collectively result in regions with high overall occupancy probability (i.e., $1-(1-p)^n \gg p$). This may have a significant impact in scenarios with a very large number of objects. We leave a thorough investigation of such an impact for future work.

Efficiency. The effectiveness of our bounded ray marching relies on the presence of high-probability empty regions: regions where the point-level bound is sufficient to identify non-intersections, and regions that can be directly pruned by the region-level bound. Consequently, the point-level and region-level bounds are less effective for scenes with high variance or near-zero mean everywhere, as high-probability empty regions become rare (the third column of Figure 20). To address this limitation, one can leverage the precomputation capabilities of SBIs. While currently used to develop

the point-level bound, SBIs could also enable precomputations of additional properties for individual realizations. For example, precomputing a Lipschitz bound could facilitate an adaptive marching step size strategy to accelerate ray marching in challenging scenes.

Trade-offs. Our method involves several efficiency–memory trade-offs, including the block size M for long sequences, the discretization level of the parameter space for non-stationary kernels (which serves as an additional dimension in the bound map), and the maximum depth for MGSVO. In particular, when the parameter space remains exceptionally large even after being restricted to a single resolution, a finer discretization may be required to achieve sufficiently tight point-level bounds, which in turn leads to increased memory consumption.

1D sparse convolution noise. Our bounded ray marching is built on 1D sparse convolution noise, which carries inherent limitations in the context of GPIS light transport. Not every covariance function admits a corresponding analytic convolution kernel for 1D sparse convolution noise [Noack et al. 2024]. Although we enable non-stationary anisotropic kernels for the bound map, when applied to solve GPIS light transport, the convolution kernels are still restricted to being globally anisotropic, i.e., $\Sigma(\boldsymbol{x}) = \Sigma$, due to the requirement of sampling surface normals [Xu et al. 2025, Section 6.4].

9 CONCLUSION

In this paper, we accelerate ray–surface intersections for GPIS, thereby speeding up GPIS rendering. We show that full GP realizations are often unnecessary during ray marching, and that efficiently obtainable auxiliary information can be exploited to safely skip expensive evaluations at an early stage. In particular, point-level bounds provide coarse yet informative constraints on individual realizations, while region-level bounds allow large spatial regions to be confidently ruled out as non-intersecting.

More broadly, our approach demonstrates a reusable principle for reasoning about stochastic realizations. By decomposing the realization generation process into stages, each stage reveals partial but actionable information about the shape of the realization, enabling meaningful decisions to be made before a full realization is constructed. This suggests a general strategy for accelerating inference and rendering tasks involving expensive stochastic processes. In our context, considering the region-level bound serves as the first stage, relying only on the mean and (co)variance functions to rule out non-intersecting regions. Evaluating the point-level bound forms the second stage, using sampled stratified Bernoulli impulse (SBI) weights, and finally, the full realization is evaluated. We also believe that the potential of SBIs has not been fully explored, and that SBI weights could provide additional information beyond the point-level bound. For example, using SBI weights to construct and precompute a Lipschitz bound could indicate how the realization varies within a limited interval.

Acknowledgments

We would like to thank the anonymous reviewers for their valuable suggestions. This work was supported by Nanjing Major Science and Technology Special Projects (No. 202209003), the National Natural Science Foundation of China (No. 62032011), and the Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China (No. JYB2025XDXM118)

The Angel scene in Figure 1 and Figure 21 is adapted from assets provided on CGTrader (user ida-faber) and Sketchfab (user Anix-MoonLight). The model in Figure 20 and the HDRIs used in Figure 1 and Figure 14 are obtained from Poly Haven. Knob scenes (Figure 6 and Figure 19) are modified from the test scenes from Seyb et al. [2024]. The Dragon scene is acquired from Xu et al. [2025].

References

T. W. Anderson and D. A. Darling. 1952. Asymptotic Theory of Certain "Goodness of Fit" Criteria Based on Stochastic Processes. *The Annals of Mathematical Statistics* 23, 2 (1952), 193 – 212. https://doi.org/10.1214/aoms/1177729437

Wilhem Barbier, Mathieu Sanchez, Axel Paris, Élie Michel, Thibaud Lambert, Tamy Boubekeur, Mathias Paulin, and Theo Thonat. 2025. Lipschitz Pruning: Hierarchical Simplification of Primitive-Based SDFs. *Computer Graphics Forum* 44, 2 (2025), e70057. https://doi.org/10.1111/cgf.70057

Benedikt Bitterli. 2018. Tungsten Renderer. https://github.com/tunabrain/tungsten/

Róbert Bán and Gábor Valasek. 2025. Generalized Lipschitz Tracing of Implicit Surfaces. *Computer Graphics Forum* 44, 1 (2025), e15202. https://doi.org/10.1111/cgf.15202

Subrahmanyan Chandrasekhar. 1960. *Radiative Transfer.* Dover Publications.

Stanimir Dragiev, Marc Toussaint, and Michael Gienger. 2011. Gaussian process implicit surfaces for shape estimation and grasping. In *2011 IEEE International Conference on Robotics and Automation*. 2845–2850. https://doi.org/10.1109/ICRA.2011.5980395

Lawrence C. Evans. 1998. *Partial Differential Equations* (1 ed.). Graduate Studies in Mathematics, Vol. 19. American Mathematical Society.

Zhimin Fan, Chen Wang, Yiming Wang, Boxuan Li, Yuxuan Guo, Ling-Qi Yan, Yanwen Guo, and Jie Guo. 2025. Bernstein Bounds for Caustics. *ACM Trans. Graph.* 44, 4, Article 63 (July 2025), 15 pages. https://doi.org/10.1145/3731145

B. Galerne, A. Leclaire, and L. Moisan. 2017. Texton Noise. *Computer Graphics Forum* 36, 8 (2017), 205–218. https://doi.org/10.1111/cgf.13073

Eric Galin, Eric Guérin, Axel Paris, and Adrien Peytavie. 2020. Segment Tracing Using Local Lipschitz Bounds. *Computer Graphics Forum* 39, 2 (2020), 545–554. https://doi.org/10.1111/cgf.13951

John C. Hart. 1996. Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer* 12, 10 (Dec. 1996), 527–545. https://doi.org/10.1007/s003710050084

Markus Heinonen, Henrik Mannerström, Juho Rousu, Samuel Kaski, and Harri Lähdesmäki. 2016. Non-Stationary Gaussian Process Regression with Hamiltonian Monte Carlo. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 51)*, Arthur Gretton and Christian C. Robert (Eds.). PMLR, Cadiz, Spain, 732–740. https://proceedings.mlr.press/v51/heinonen16.html

Pierre Hubert-Brierre, Eric Guérin, Adrien Peytavie, and Eric Galin. 2025. Accelerating Signed Distance Functions. *Computer Graphics Forum* 44, 7 (2025), e70258. https://doi.org/10.1111/cgf.70258

James T. Kajiya. 1986. The rendering equation. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '86)*. Association for Computing Machinery, New York, NY, USA, 143–150. https://doi.org/10.1145/15922.15902

Olav Kallenberg. 1997. *Foundations of modern probability.* Springer New York, NY. https://doi.org/10.1007/b98838

D. Kalra and A. H. Barr. 1989. Guaranteed ray intersections with implicit surfaces. In *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '89)*. Association for Computing Machinery, New York, NY, USA, 297–306. https://doi.org/10.1145/74333.74364

Benjamin Keinert, Henry Schäfer, Johann Korndörfer, Urs Ganse, and Marc Stamminger. 2014. Enhanced Sphere Tracing. In *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*, Andrea Giachetti (Ed.). The Eurographics Association. https://doi.org/10.2312/stag.20141233

A. Knoll, Y. Hijazi, A. Kensler, M. Schott, C. Hansen, and H. Hagen. 2009. Fast Ray Tracing of Arbitrary Implicit Surfaces with Interval and Affine Arithmetic. *Computer Graphics Forum* 28, 1 (2009), 26–40. https://doi.org/10.1111/j.1467-8659.2008.01189.x

Ares Lagae, Sylvain Lefebvre, Rob Cook, Tony DeRose, George Drettakis, David S Ebert, John P Lewis, Ken Perlin, and Matthias Zwicker. 2010. State of the art in procedural noise functions. *EG 2010-State of the Art Reports* (2010), 1–19.

Ares Lagae, Sylvain Lefebvre, George Drettakis, and Philip Dutré. 2009. Procedural noise using sparse Gabor convolution. *ACM Trans. Graph.* 28, 3, Article 54 (July 2009), 10 pages. https://doi.org/10.1145/1531326.1531360

Ares Lagae, Sylvain Lefebvre, and Philip Dutre. 2011. Improving Gabor Noise. *IEEE Transactions on Visualization and Computer Graphics* 17, 8 (2011), 1096–1107. https:

//doi.org/10.1109/TVCG.2010.238
Samuli Laine and Tero Karras. 2010. Efficient sparse voxel octrees. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (Washington, D.C.) *(I3D '10)*. Association for Computing Machinery, New York, NY, USA, 55–63. https://doi.org/10.1145/1730804.1730814
J. P. Lewis. 1989. Algorithms for solid noise synthesis. *SIGGRAPH Comput. Graph.* 23, 3 (July 1989), 263–270. https://doi.org/10.1145/74334.74360
Stephanie Wenxin Liu, Michael Fischer, Paul D. Yoo, and Tobias Ritschel. 2024. Neural Bounding. In *ACM SIGGRAPH 2024 Conference Papers* (Denver, CO, USA) *(SIGGRAPH '24)*. Association for Computing Machinery, New York, NY, USA, Article 98, 10 pages. https://doi.org/10.1145/3641519.3657442
Bertil Matérn. 1960. Spatial Variation. *Meddelanden från Statens skogsforskningsinstitut* 49: 5 (1960).
Mathéo Moinet and Fabrice Neyret. 2025. Fast sphere tracing of procedural volumetric noise for very large and detailed scenes. *Computer Graphics Forum* 44, 2 (2025), e70072. https://doi.org/10.1111/cgf.70072
Ken Museth. 2013. VDB: High-resolution sparse volumes with dynamic topology. *ACM Trans. Graph.* 32, 3, Article 27 (July 2013), 22 pages. https://doi.org/10.1145/2487228.2487235
Marcus M. Noack, Hengrui Luo, and Mark D. Risser. 2024. A unifying perspective on non-stationary kernels for deeper Gaussian processes. *APL Machine Learning* 2, 1 (Feb. 2024), 010902. https://doi.org/10.1063/5.0176963
K. Perlin and E. M. Hoffert. 1989. Hypertexture. *SIGGRAPH Comput. Graph.* 23, 3 (July 1989), 253–262. https://doi.org/10.1145/74334.74359
Kai Poethkow, Christoph Petz, and Hans-Christian Hege. 2013. Approximate Level-Crossing Probabilities for Interactive Visualization of Uncertain Isocontours. *International Journal for Uncertainty Quantification* 3, 2 (2013), 101–117. https://doi.org/10.1615/Int.J.UncertaintyQuantification.2012003958
Kai Pöthkow, Britta Weber, and Hans-Christian Hege. 2011. Probabilistic Marching Cubes. *Computer Graphics Forum* 30, 3 (2011), 931–940. https://doi.org/10.1111/j.1467-8659.2011.01942.x
Dario Seyb, Eugene d'Eon, Benedikt Bitterli, and Wojciech Jarosz. 2024. From microfacets to participating media: A unified theory of light transport with stochastic geometry. *ACM Trans. Graph.* 43, 4, Article 112 (July 2024), 17 pages. https://doi.org/10.1145/3658121
Vincent Tavernier, Fabrice Neyret, Romain Vergne, and Joëlle Thollot. 2019. Making Gabor Noise Fast and Normalized. In *Eurographics 2019 - Short Papers*, Paolo Cignoni and Eder Miguel (Eds.). The Eurographics Association. https://doi.org/10.2312/egs.20191009
H. Jean Thiebaux. 1976. Anisotropic Correlation Functions for Objective Analysis. *Monthly Weather Review* 104, 8 (1976), 994 – 1002. https://doi.org/10.1175/1520-0493(1976)104<0994:ACFFOA>2.0.CO;2
Christopher Uchytil and Duane Storti. 2023. A Function-Based Approach to Interactive High-Precision Volumetric Design and Fabrication. *ACM Trans. Graph.* 43, 1, Article 3 (Sept. 2023), 15 pages. https://doi.org/10.1145/3622934
Jarke J. van Wijk. 1991. Spot noise texture synthesis for data visualization. *SIGGRAPH Comput. Graph.* 25, 4 (July 1991), 309–318. https://doi.org/10.1145/127719.122751
Edgar Velázquez-Armendáriz, Shuang Zhao, Miloš Hašan, Bruce Walter, and Kavita Bala. 2009. Automatic bounding of programmable shaders for efficient global illumination. *ACM Trans. Graph.* 28, 5 (Dec. 2009), 1–9. https://doi.org/10.1145/1618452.1618488
Christopher KI Williams and Carl Edward Rasmussen. 2006. *Gaussian processes for machine learning*. Vol. 2. MIT press Cambridge, MA.
Oliver Williams and Andrew Fitzgibbon. 2006. Gaussian Process Implicit Surfaces. https://www.microsoft.com/en-us/research/wp-content/uploads/2006/06/gpc.pdf
James T. Wilson, Viacheslav Borovitskiy, Alexander Terenin, Peter Mostowsky, and Marc Peter Deisenroth. 2021. Pathwise conditioning of Gaussian processes. *J. Mach. Learn. Res.* 22, 1, Article 105 (Jan. 2021), 47 pages.
Kehan Xu, Benedikt Bitterli, Eugene d'Eon, and Wojciech Jarosz. 2025. Practical Gaussian Process Implicit Surfaces with Sparse Convolutions. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 44, 6 (Dec. 2025). https://doi.org/10.1145/3763329
Cédric Zanni. 2024. Synchronized Tracing of Primitive-based Implicit Volumes. *ACM Trans. Graph.* 44, 1, Article 6 (Nov. 2024), 15 pages. https://doi.org/10.1145/3702227

A Light transport on GPIS

The light transport with realization f is given by the classical surface rendering equation [Kajiya 1986]:

$$L^f(\boldsymbol{x},\boldsymbol{\omega}) = \int_{\mathcal{S}^2} \rho^f(\boldsymbol{x}_s^f, \mathbf{n}_s^f) L^f(\boldsymbol{x}_s^f, \boldsymbol{\omega}_s) \mathrm{d}\boldsymbol{\omega}_s, \tag{37}$$

where $L^f(\boldsymbol{x},\boldsymbol{\omega})$ is the radiance arriving at $\boldsymbol{x}$ from the direction $\boldsymbol{\omega}$. $\boldsymbol{x}_s^f$ is the closest intersection along the ray originating at $\boldsymbol{x}$ in the direction $\boldsymbol{\omega}$, and its corresponding normal is $\mathbf{n}_s^f$. $\rho^f(\boldsymbol{x}_s^f) = \rho^f(\boldsymbol{x}_s^f, \boldsymbol{\omega}, \boldsymbol{\omega}_s, \mathbf{n}_s^f)|\mathbf{n}_s^f \cdot \boldsymbol{\omega}|$ is the cosine-weighted BRDF.

The light transport on GPIS is defined by ensemble-averaging the total radiance over all realizations:

$$\left\langle L^f(\boldsymbol{x},\boldsymbol{\omega})\right\rangle_\zeta = \int_{\mathcal{GP}_{|\zeta}} L^f(\boldsymbol{x},\boldsymbol{\omega}) \mathrm{d}\gamma(f \mid \zeta), \tag{38}$$

where ζ denotes the set of constraints on the Gaussian process (e.g., observed function values or gradients) and $\gamma(f \mid \zeta)$ denotes the classical Wiener measure under the constraint ζ, i.e., the probability density of sampling $f \sim \mathcal{GP}$.

Since path tracing only requires realization values along the ray's transversal, the light transport equation for GPIS could be written as [Seyb et al. 2024]:

$$\left\langle L^f(\boldsymbol{x},\boldsymbol{\omega})\right\rangle_\zeta = \left\langle \int_{\mathcal{S}^2} \rho^f(\boldsymbol{x}_s^f, \mathbf{n}_s^f) \left\langle L^{f'}(\boldsymbol{x}_s^f, \boldsymbol{\omega}_s)\right\rangle_{\zeta'} \mathrm{d}\boldsymbol{\omega}_s \right\rangle_\zeta \tag{39}$$

where ζ' is an augmented set of constraints that extends the original set ζ by incorporating the new observations along the ray. To prevent the computational cost from growing unboundedly as ζ accumulates during path tracing, Seyb et al. [2024] propose the *Renewal+* memory model, making

$$\zeta' = \left(f(x_s^f) = f'(x_s^f) = 0\right) \wedge \left(\nabla f(x_s^f) = \nabla f'(x_s^f)\right). \tag{40}$$

They also introduce a function-space, ray-marching method to solve Eq. (39). This method first samples a multivariate Gaussian conditioned on ζ over n marching points to determine the intersection point $\boldsymbol{x}_s^f$ and the gradient component $\nabla_z f(\boldsymbol{x}_s^f)$ along the ray. The remaining two orthogonal components $\nabla_x f(\boldsymbol{x}_s^f), \nabla_y f(\boldsymbol{x}_s^f)$ are then sampled conditioned on ζ and the previously sampled values along the segment, exploiting the fact that they are still jointly Gaussian with the function value. Finally, $\mathbf{n}_s^f$ is obtained by normalizing $\nabla f(\boldsymbol{x}_s^f)$. This procedure has a total time complexity of $O(n^3)$ per path segment due to the cost of sampling a multivariate Gaussian and conditioning for $\nabla_x f(\boldsymbol{x}_s^f)$ and $\nabla_y f(\boldsymbol{x}_s^f)$.