Adaptive Ray Marching for Rendering Gaussian Process Implicit Surfaces

ZHIQIAN ZHOU, University of California, Irvine, USA
DARIO SEYB, Valve, USA
SHUANG ZHAO, University of Illinois Urbana-Champaign, USA

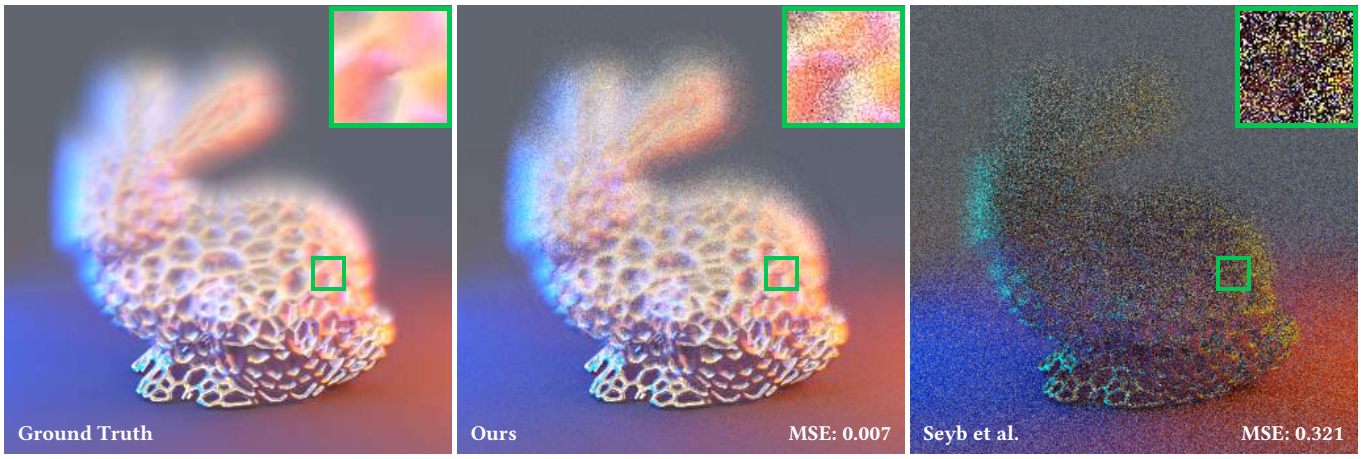


Fig. 1. GPIS is a probabilistic representation of 3D shapes, bridging the gap between microfacets, participating media and measurement uncertainties. Despite its usefulness, it has been historically challenging to render. Our method accelerates computing the ensemble averaged light transport by orders of magnitude compared to Seyb et al. [2024], by adaptively tracing the Gaussian process and drawing values using an online sampler. The figure renders a bunny-shaped Gaussian field with our method and Seyb et al. [2024] at equal time (7.5 minutes).

Gaussian Process Implicit Surfaces (GPIS) represent geometry as a distribution over implicit functions. Modeling an object's appearance as the expected rendering of a GPIS yields a unified framework that captures diverse light-transport effects including microfacet-like reflections and volumetric scattering. Despite this generality, computing GPIS–ray intersections requires sampling conditional multivariate Gaussian distributions along each ray and remains prohibitively expensive. We introduce an online sampling algorithm that draws these distributions incrementally, and an adaptive marching scheme that takes large steps where the surface is provably absent, minimizing the probability of missed intersections. Together, these ideas reduce rendering MSE by up to 46× at equal time compared to existing methods.

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

Additional Key Words and Phrases: Gaussian process implicit surfaces, ray marching, implicit surface rendering, online sampling

ACM Reference Format:
Zhiqian Zhou, Dario Seyb, and Shuang Zhao. 2026. Adaptive Ray Marching for Rendering Gaussian Process Implicit Surfaces. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers (SIGGRAPH Conference Papers '26), July 19–23, 2026, Los Angeles, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3799902.3811101

Authors' Contact Information: Zhiqian Zhou, zhou.zq@uci.edu, University of California, Irvine, USA; Dario Seyb, dario.seyb@gmail.com, Valve, USA; Shuang Zhao, shzhao@illinois.edu, University of Illinois Urbana-Champaign, USA.

SIGGRAPH Conference Papers '26, Los Angeles, CA, USA

ACM ISBN 979-8-4007-2554-8/2026/07
https://doi.org/10.1145/3799902.3811101

1 Introduction

Implicit surfaces are an attractive representation for graphics because they decouple geometry from tessellation and support robust queries such as inside–outside tests and smooth normals. *Gaussian Process Implicit Surfaces (GPIS)* [Williams and Fitzgibbon 2006] extend this idea by modeling the implicit function as a distribution rather than a single deterministic field, providing a systematic way to describe stochastic surfaces and enabling principled conditioning on sparse observations.

Recently, Seyb et al. [2024] showed that modeling object appearance as the expected rendering of a GPIS yields a unified framework capable of capturing diverse light-transport effects including microfacet-like reflections and volumetric scattering. Despite this generality, GPIS rendering remains far more expensive than conventional surface or volume rendering. The bottleneck is GPIS–ray intersection: along each ray, the renderer must determine where a random implicit function crosses zero, which requires sampling high-dimensional multivariate Gaussian distributions and makes the computation prohibitively costly.

We tackle this bottleneck with two mutually reinforcing ideas: an *online sampling* algorithm that draws correlated Gaussian samples incrementally as the ray advances, and an *adaptive marching* scheme that uses probabilistic bounds to take large steps where the surface is provably absent. Together, they dramatically reduce the per-ray cost while preserving the probabilistic correctness of GPIS rendering.

Concretely, we make three contributions:

(1) **Online sampling.** We devise a method for drawing correlated Gaussian samples when query locations are provided on the fly, enabling sequential ray marching while preserving the correct joint distribution across adaptively chosen points (Section 4.1).

(2) **Probabilistic range bounds.** We develop high-probability bounds for GP functions over continuous intervals (Section 4.2). These bounds, in turn, guide the computation of ray-marching step sizes.

(3) **Adaptive ray marching.** We introduce a practical ray marching algorithm for GPIS–ray intersection that uses adaptive step sizes, substantially reducing the number of sample points needed per ray while maintaining probabilistic correctness (Section 4.2).

In Section 5, we show that our method reduces rendering error by up to 46× at equal time compared to Seyb et al. [2024].

2 Related Work

Stochastic surfaces for rendering. Stochastic implicit surfaces offer a convenient way to model geometric uncertainty and fine-scale structure. Early work in graphics explored modeling and rendering implicit surfaces driven by stochastic processes [Gamito 2009]. Recently, Seyb et al. [2024] derived a unified framework to model and render a diverse set of appearances, including microfacets with correlated microstructures and non-exponential media. This technique leverages *Gaussian Process Implicit Surfaces* (GPIS) [Williams and Fitzgibbon 2006], which represent stochastic surfaces as zero level sets of stochastic functions governed by Gaussian processes [Williams and Rasmussen 2006]. In this paper, we introduce a technique to greatly accelerate GPIS rendering via adaptive ray marching. Our technique imposes no additional approximation to the underlying formulation and therefore preserves the generality and physical accuracy offered by this framework.

Concurrently, Xu et al. [2025] proposed a practical GPIS formulation based on sparse convolutions, substantially improving efficiency while retaining control over spatial correlation and conditioning. We consider this technique orthogonal to ours.

Stochastic surfaces for reconstruction and uncertainty visualization. Stochastic geometries have also been demonstrated to be suitable for many applications in 3D reconstruction [Dragiev et al. 2011; Sellán and Jacobson 2022, 2023; Miller et al. 2024]. Brodlie et al. [2012] survey techniques and design considerations for uncertainty visualization. For uncertain scalar fields, Pfaffelmoser et al. [2011] visualize positional and geometrical variability of isosurfaces, while Fout and Ma [2012] propose fuzzy volume rendering as a perceptual metaphor for uncertainty in volumetric data. These applications further motivate efficient rendering methods. In this paper, we focus only on the (forward) rendering problem.

Radiative transport in stochastic media. Previously, ensemble-average light transport through stochastic media has been explored via precomputed scattering and multi-scale representations [Moon et al. 2007; Meng et al. 2015; Müller et al. 2016]. In addition, generalized radiative-transfer models [Jarabo et al. 2018; Bitterli et al. 2018] capable of capturing spatial correlation and non-exponential attenuation have been introduced.

Ray-implicit-surface intersection. Identifying intersection points between a ray and an implicit surface—a key operation needed to render the surface—is often conducted by *marching* along the ray with steps chosen from bounds of the corresponding implicit function. Sphere tracing [Hart 1996] uses a signed-distance bound to offer high efficiency. Subsequent work has improved sphere tracing and extended sphere tracing to deformed signed-distance fields and particle-based fluid surfaces [Keinert et al. 2014; Seyb et al. 2019; Wu et al. 2022]. For more general implicit functions, intersections can be obtained from bounds on the rate of change [Kalra and Barr 1989], and tighter *local Lipschitz* or *forward inclusion* bounds enable larger steps [Galin et al. 2020; Aydinlilar and Zanni 2023]. In contrast, interval/affine arithmetic and range analysis provide inclusion guarantees for implicit queries [Mitchell 1990; Knoll et al. 2009; Sharp and Jacobson 2022].

In this paper, we use bound-based marching by leveraging a Lipschitz-like bound derived from GP statistics to enable adaptive marching step sizes.

3 Preliminaries

We introduce our notations, review the Gaussian-process background in Section 3.1, and state the problem of GPIS rendering in Section 3.2.

Notations. For any (potentially stochastic) 3D function $g(\boldsymbol{x})$ and a fixed set of points $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n \in \mathbb{R}^3\}$, we define

$$g(X) := \begin{bmatrix} g(\boldsymbol{x}_1) \\ \vdots \\ g(\boldsymbol{x}_n) \end{bmatrix}, \tag{1}$$

as a (column) vector comprised of the values of g evaluated at these points. We note that, when g is stochastic, so will be $g(X)$.

Given two sets of points $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ and $Y = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m\}$, for any $h(\boldsymbol{x}, \boldsymbol{y})$, we define $h(X, Y)$ as an $n \times m$ (block) matrix with its (i, j)-th element being

$$h(X, Y)_{ij} := h(\boldsymbol{x}_i, \boldsymbol{y}_j). \tag{2}$$

3.1 Gaussian Process

In this paper, we consider a **Gaussian process** (GP) [Williams and Rasmussen 2006] as a probability distribution $\mathcal{GP}(\mu, k)$ of 3D scalar-valued functions characterized by a differentiable *mean function* $\mu : \mathbb{R}^3 \mapsto \mathbb{R}$ and a twice-differentiable *covariance (kernel) function* $k : \mathbb{R}^3 \times \mathbb{R}^3 \mapsto \mathbb{R}$.

For any stochastic function $f : \mathbb{R}^3 \mapsto \mathbb{R} \sim \mathcal{GP}(\mu, k)$, its values $f(X)$ evaluated at a fixed set of 3D locations $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ follow a multivariate Gaussian distribution:

$$f(X) \sim \mathcal{N}\big(\mu(X),\, k(X, X)\big). \tag{3}$$

Covariance functions. The covariance (kernel) function $k(\boldsymbol{x}, \boldsymbol{y})$ must be symmetric and positive semi-definite. When k only depends on $(\boldsymbol{x} - \boldsymbol{y})$, it is considered *stationary*. A commonly used stationary kernel is the *radial basis function (RBF)* kernel (also known as the

squared exponential kernel):

$$k(\boldsymbol{x}, \boldsymbol{y}; l) = \sigma^2 \exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{y}\|^2}{2l^2}\right). \tag{4}$$

Our method makes no assumption on the exact form of the kernel k.

Derivatives of GPs. For any $f \sim \mathcal{GP}(\mu, k)$, its gradient function ∇f is known to follow another vector-valued GP:

$$\nabla f \sim \mathcal{GP}\left(\nabla\mu,\ \frac{\partial^2 k}{\partial \boldsymbol{x}\partial \boldsymbol{y}}\right). \tag{5}$$

Moreover, for any $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, it holds that

$$\begin{bmatrix} f(X) \\ \nabla f(X) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu(X) \\ \nabla\mu(X) \end{bmatrix}, \begin{bmatrix} k(X,X) & \frac{\partial k}{\partial \boldsymbol{y}}(X,X)^\top \\ \frac{\partial k}{\partial \boldsymbol{x}}(X,X) & \frac{\partial^2 k}{\partial \boldsymbol{x}\partial \boldsymbol{y}}(X,X) \end{bmatrix}\right). \tag{6}$$

Conditioning (posterior GPs). Given $f \sim \mathcal{GP}(\mu, k)$ and observations $v_i \in \mathbb{R}$ at locations $\boldsymbol{c}_i \in \mathbb{R}^3$ for $i = 1, \ldots, m$, let $C := \{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_m\}$, $V := [v_1, \ldots, v_m]^\top$, and $\zeta := \big(f(C) = V\big)$. The conditioned function $f_{|\zeta}$ remains a Gaussian process:

$$f_{|\zeta} \sim \mathcal{GP}(\mu_{|\zeta},\ k_{|\zeta}), \tag{7}$$

where

$$\begin{aligned} \mu_{|\zeta}(\boldsymbol{x}) &= \mu(\boldsymbol{x}) + k(\boldsymbol{x}, C)\, k(C,C)^{-1}\big(V - \mu(C)\big), \\ k_{|\zeta}(\boldsymbol{x}, \boldsymbol{y}) &= k(\boldsymbol{x}, \boldsymbol{y}) - k(\boldsymbol{x}, C)\, k(C,C)^{-1} k(C, \boldsymbol{y}), \end{aligned} \tag{8}$$

for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^3$.

Sampling GPs. Drawing a full sample function from a GP generally requires discretization. Given a set of points $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, one samples $f(X)$ from the multivariate Gaussian in Equation 3 by: (i) Drawing ξ from the n-dimensional standard normal distribution $\mathcal{N}(0, I_n)$; (ii) Setting

$$f(X) \leftarrow \mu(X) + L\xi, \tag{9}$$

where L is the Cholesky decomposition of $k(X,X)$.

3.2 Gaussian Process Implicit Surface

To obtain a unified appearance model capable of capturing both surface-like (e.g., reflection) and volumetric (e.g., subsurface scattering) light-transport effects, Seyb et al. [2024] have introduced a technique that synthesizes object appearances as expected renderings of stochastic surfaces. In the following, we briefly revisit some key ingredients of their technique.

Implicit surface. An implicit surface is a set of points where a scalar-valued function $f : \mathbb{R}^3 \to \mathbb{R}$ takes the value of zero: $\{\boldsymbol{x} \in \mathbb{R}^3 \mid f(\boldsymbol{x}) = 0\}$. If f is stochastic and subject to a GP, the corresponding implicit surface is called a **Gaussian process implicit surface** (GPIS), which is a distribution over (infinitely many) possible surfaces.

GPIS-ray intersection. To render a GPIS, a key operation is to compute ray-GPIS intersections. Precisely, given a (fixed) ray $r(t) = \boldsymbol{x}_0 + t\boldsymbol{\omega}$, finding its first intersection with a GPIS f amounts to obtaining the *free-flight distance* $\tau := \inf\{t > 0 \mid f(\boldsymbol{x}_0 + t\boldsymbol{\omega}) = 0\}$, which is not deterministic but a random variable described by a *free-flight distribution.*

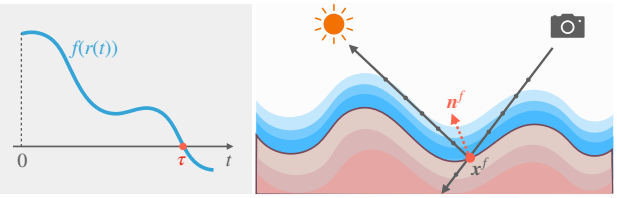


Fig. 2. The implicit surface f becomes random when f is a Gaussian process. In the path tracing algorithm, each path progressively realizes f along the ray to find a (random) intersection $\boldsymbol{x}^f$, as well as a (random) surface normal $\boldsymbol{n}^f$. Image on the left shows an example realization of f along the ray.

Sampling this free-flight distribution is equivalent to first realizing (sampling) the underlying random field, and then computing the intersection with the resulting implicit surface. This realization can further be restricted to the 1D subspace which the ray spans, meaning that we only need to realize a 1D Gaussian process. The intersection then becomes the distribution of the first root in the 1D Gaussian process.

Noting that $f(\boldsymbol{x}_0 + t\boldsymbol{\omega})$, as a 1D function of t, follows a 1D GP, Seyb et al. [2024] have proposed to discretize the ray into a sequence of evaluation points X (with even spacing) and realize the values of f at these locations by sampling a multivariate Gaussian using Equation 9. Lastly, f is interpolated linearly between the evaluation points to locate the first zero crossing (if any) that gives τ.

The surface normal at the intersection point $\boldsymbol{x}_0 + \tau\boldsymbol{\omega}$ is given by $n_s = \frac{\nabla f(\boldsymbol{x}_0+\tau\boldsymbol{\omega})}{\|\nabla f(\boldsymbol{x}_0+\tau\boldsymbol{\omega})\|}$, where ∇f is sampled using Equation 6.

In contrast, our method takes a geometric tracing approach, marching along the ray adaptively at large yet conservative intervals. This drastically reduces the number of GP samples, and reduces computation by about a cubic factor.

Ensemble averaged light transport. The light transport equation for a given surface f is given by Kajiya [1986]:

$$L^f(p, \boldsymbol{\omega}_0) = L_e^f(p, \boldsymbol{\omega}_0) + \int_{\mathcal{S}^2} \rho(\boldsymbol{x}^f, n^f, \boldsymbol{\omega}_0, \boldsymbol{\omega}_s) L^f(\boldsymbol{x}^f, \boldsymbol{\omega}_s) \mathrm{d}\boldsymbol{\omega}_s \tag{10}$$

which integrates over all incident directions $\boldsymbol{\omega}_s$ on the unit sphere $\mathcal{S}^2$. The BSDF ρ is evaluated at the shading location $\boldsymbol{x}^f$ with surface normal n^f. The emitted radiance is denoted by L_e^f, and the incident radiance by L^f.

For a GPIS, its *ensemble averaged light transport* [Seyb et al. 2024] is defined by the light transport averaged over all realizations of f:

$$\big\langle L^f(p, \boldsymbol{\omega}_0) \big\rangle_\zeta = \int_{\mathcal{GP}} L^f(p, \boldsymbol{\omega}_0) \mathrm{d}\gamma_{\mu,k}(f \mid \zeta). \tag{11}$$

where $\gamma_{\mu,k}(f \mid \zeta)$ is the Gaussian measure [Williams and Rasmussen 2006] of sampling $f \sim \mathcal{GP}(\mu_{|\zeta}, k_{|\zeta})$. Seyb et al. [2024] further show that this equation can be computed by progressively sampling f along light paths in a path-tracing algorithm instead of realizing f in the entire 3D domain. Figure 2 illustrates this process along a single light path. We give a brief description of the radiance estimator:

(1) A ray $r(t)$ is shot from the camera into the scene;

(2) The GPIS-ray intersection τ is sampled for $r(t)$;

(3) The local gradient at the intersection $g := \nabla f(r(\tau))$ is sampled, giving the surface normal $n_s = g/\|g\|$;

(4) Like in path tracing, a scatter direction $\boldsymbol{\omega}_s$ is sampled with BSDF ρ, and the incoming radiance is recursively estimated.

Note that each set of samples is conditioned on the sample history in that path. Seyb et al. [2024] also proposed a *Renewal+* model to approximate the full sample history of the current path with a single pair of value and gradient at the last bounce location. We refer the reader to their paper for details of the rendering algorithm.

Despite its mathematical elegance, this algorithm is usually too slow for practical use, as each GPIS-ray intersection requires the ray to be finely discretized, making the realization of function values—which uses Equation 9 and runs in $O(n^3)$ time—very time-consuming.

Motivated by these practical limitations, we introduce a new technique to drastically reduce the computational cost—which we will present in Section 4.

4 Our Method

We introduce an efficient ray-marching method for GPIS-ray intersection. At a high level, the method marches along the ray with adaptive step sizes, incrementally samples the Gaussian process, until a zero crossing is bracketed. This requires drawing correlated GP samples at ray locations that are selected on the fly, so we begin by describing an online Gaussian-process sampler in Section 4.1. We then build on this sampler to develop the adaptive ray-marching procedure in Section 4.2.

4.1 Online Gaussian Process Sampler

We now describe our method to draw correlated samples from a Gaussian process, where the sample locations are selected on the fly, rather than predetermined all at once.

To sample the Gaussian process at the given stream of points $X := \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, we need to draw $f(\boldsymbol{x}_i)$ immediately after $\boldsymbol{x}_i$ is given, while ensuring $f(X) \sim \mathcal{N}(\mu(X), k(X,X))$. To this end, we sequentially determine each $f(\boldsymbol{x}_i)$ conditioned on the already sampled $f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_{i-1})$ as follows.

For any $V := [v_1, \ldots, v_n]^\top \in \mathbb{R}^{n\times 1}$, it holds that

$$\begin{aligned}\mathbb{P}\big[f(X) = V\big] &= \prod_{i=1}^{n} \mathbb{P}\big[f(\boldsymbol{x}_i) = v_i \mid \underbrace{f(X_{i-1}) = V_{i-1}}_{=:\zeta_{i-1}}\big] \\ &= \prod_{i=1}^{n} \mathcal{N}\big(v_i \,;\, \mu_{|\zeta_{i-1}}(\boldsymbol{x}_i),\ k_{|\zeta_{i-1}}(\boldsymbol{x}_i, \boldsymbol{x}_i)\big),\end{aligned} \tag{12}$$

where $X_i := \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i\}$ and $V_i := [v_1, \ldots, v_i]^\top$. Thus, to draw a sample of $f(X)$, we start by sampling its first component $f(\boldsymbol{x}_1)$ from the Gaussian distribution $\mathcal{N}(\mu(\boldsymbol{x}_1), k(\boldsymbol{x}_1, \boldsymbol{x}_1))$. Then, for $i = 2, \ldots, n$, we iteratively draw the i-th component $f(\boldsymbol{x}_i)$ from the distribution $\mathcal{N}\big(\mu_{|\zeta_{i-1}}(\boldsymbol{x}_i),\ k_{|\zeta_{i-1}}(\boldsymbol{x}_i, \boldsymbol{x}_i)\big)$.

In practice, naïvely computing the mean $\mu_{|\zeta_{i-1}}(\boldsymbol{x}_i)$ and the variance $k_{|\zeta_{i-1}}(\boldsymbol{x}_i, \boldsymbol{x}_i)$ in Equation 12 via Equation 8 requires inverting the (dense) matrix $k(X_{i-1}, X_{i-1})$, which takes $O(i^3)$ time for each i. This gives a total time complexity of $O(n^4)$ for sampling $f(X)$.

To improve the efficiency of this process, we leverage the Cholesky decomposition $L_i \in \mathbb{R}^{i\times i}$ of $k(X_i, X_i)$. Specifically, we rewrite Equation 8 (with $C = X_i$, $V = V_i$, and $\boldsymbol{x} = \boldsymbol{y} = \boldsymbol{x}_{i+1}$) as

$$\begin{aligned}\mu_{|\zeta_i}(x_{i+1}) &= \mu(x_{i+1}) + \beta_i^\top L_i^{-1}(V_i - \mu(X_i)), \\ k_{|\zeta_i}(x_{i+1}, x_{i+1}) &= k(x_{i+1}, x_{i+1}) - \beta_i^\top \beta_i,\end{aligned} \tag{13}$$

where

$$\beta_i := L_i^{-1}\, k(X_i, \boldsymbol{x}_{i+1}). \tag{14}$$

We note that, provided the decomposition L_i, the right-hand side of Equation 13 can be evaluated in $O(i^2)$ time since β_i can be obtained using backward substitution (instead of general linear solve).

In the following, we discuss how L_i can be efficiently obtained for each i.

Incremental construction of Cholesky decomposition. We now describe how to efficiently compute the Cholesky decomposition L_i of $k(X_i, X_i)$ for $i = 1, 2, \ldots, n$. Instead of computing from scratch, we exploit the relation between $k(X_i, X_i)$ and $k(X_{i+1}, X_{i+1})$ as follows.

Since $X_{i+1} = X_i \cup \{\boldsymbol{x}_{i+1}\}$, the matrix $k(X_{i+1}, X_{i+1}) \in \mathbb{R}^{(i+1)\times(i+1)}$ contains $k(X_i, X_i) \in \mathbb{R}^{i\times i}$ in its top-left corner. Precisely,

$$k(X_{i+1}, X_{i+1}) = \begin{bmatrix} k(X_i, X_i) & k(X_i, \boldsymbol{x}_{i+1}) \\ k(\boldsymbol{x}_{i+1}, X_i) & k(\boldsymbol{x}_{i+1}, \boldsymbol{x}_{i+1}) \end{bmatrix}. \tag{15}$$

Based on this relation, applying a rank-1 modification [Gill et al. 1974] produces

$$L_{i+1} = \begin{bmatrix} L_i & 0 \\ \beta_i^\top & \sqrt{k(\boldsymbol{x}_{i+1}, \boldsymbol{x}_{i+1}) - \beta_i^\top \beta_i} \end{bmatrix}, \tag{16}$$

where β_i follows the definition in Equation 14.

Since Equation 16 can be evaluated in $O(i^2)$ time for each i, $L_2, \ldots, L_n$ (from the base case $L_1 = \sqrt{k(\boldsymbol{x}_1, \boldsymbol{x}_1)}$) can be obtained in $O(n^3)$ time. Combined with the $O(i^2)$ per-step cost of evaluating Equation 13, the total cost of our online sampler is $O(n^3)$—a factor-of-n improvement over the naïve $O(n^4)$ approach.

4.2 Ray-GPIS Intersection

Based on our incremental sampling method discussed in Section 4.1, we now introduce a technique to compute ray-GPIS intersections efficiently. As stated in Section 3.2, given a ray $(\boldsymbol{x}_0, \boldsymbol{\omega})$, computing the intersection amounts to sampling the first zero crossing of $f(\boldsymbol{x}_0 + t\boldsymbol{\omega})$, which follows a 1D GP. For brevity, we imply the ray parameters and write $f(t) := f(\boldsymbol{x}_0 + t\boldsymbol{\omega})$ and $\mu(t) := \mu(\boldsymbol{x}_0 + t\boldsymbol{\omega})$.

We first present our algorithm for finding zero crossings and then detail how the adaptive step size is determined.

Our algorithm. To find the first zero crossing of f, our algorithm (summarized in Algorithm 1) marches along the ray using adaptive step sizes. At the i-th marching step with $t = t_i$ and sampled $v_i = f(t_i)$, our method performs the following operations.

(1) Determine the step size Δ_i (which we will discuss later).

(2) Let $t_{i+1} = t_i + \Delta_i$ and sample $v_{i+1} = f(t_{i+1})$ using our online sampler introduced in Section 4.1.

(3) If $v_i \cdot v_{i+1} > 0$, we consider the interval $[t_i, t_{i+1}]$ to contain no root of f and continue to the $(i+1)$-th marching step.

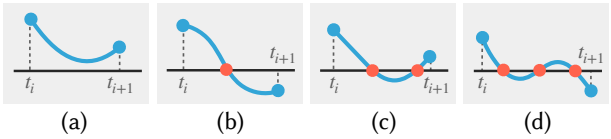


Fig. 3. Four different situations of zero crossing over an interval: (a) having no zero crossing, (b) endpoints opposite sign with exactly one zero crossing, (c) endpoints same sign with zero crossing(s), (d) endpoints opposite sign with more than one zero crossings. If the interval is chosen too large, (c) and (d) could cause errors in root finding.

(4) Otherwise, if $v_i \cdot v_{i+1} \le 0$, we consider the interval to contain exactly one root and use a bracketed root solver (e.g. bisection or Dekker-Brent [Brent 1971]) to locate the root τ.

We note that, when executing the root solver in Step (4), we apply our online sampler for each evaluation of f (and append the result to condition future evaluations).

Adaptive step size. A key component of our algorithm presented above is to determine the marching step size Δ_i in Step (1). Our goal is to take large marching steps without introducing too much risk of missing a root (see Figure 3). Precisely, we would like to pick Δ_i such that the probability for f to have a root in the interval $[t_i, t_i + \Delta_i]$ is no more than some user-specified tolerance $\eta > 0$.

To quantitatively obtain the relation between the sampled $v_i = f(t_i)$, its mean $\mu(t_i)$, and Δ_i, we make the following assumptions:

- The mean μ and covariance k functions to be continuously differentiable (with respect to spatial locations);
- μ is Lipschitz-continuous with $\Lambda_\mu := \sup_{\boldsymbol{x}\neq\boldsymbol{y}} |\mu(\boldsymbol{x})-\mu(\boldsymbol{y})|/\|\boldsymbol{x}-\boldsymbol{y}\|$;
- Both k and $\partial^2 k/\partial \boldsymbol{x}\partial \boldsymbol{y}$ are uniformly bounded by $\sigma^2 := \sup_{\boldsymbol{x}} |k(\boldsymbol{x})|$ and $M = \sup_{\boldsymbol{x},\boldsymbol{y}} \left|\frac{\partial^2 k}{\partial \boldsymbol{x}\partial \boldsymbol{y}}(\boldsymbol{x},\boldsymbol{y})\right|$ respectively.

In practice, these bounds are usually obtained by preprocessing scene data, or analytically computed for procedural functions. For example, for signed-distance functions, we have $\Lambda_\mu = 1$.

Then, assuming without loss of generality that $f(t_i) > 0$, it holds that

$$\begin{aligned} \inf_{[t_i,t_{i+1}]} f &\ge \mu(t_i) - \Lambda_\mu \Delta_i - Q_\eta \sigma, \\ \inf_{[t_i,t_{i+1}]} f &\ge f(t_i) - \left(\Lambda_\mu + C_\eta \sqrt{M}\right) \Delta_i. \end{aligned} \tag{17}$$

with a probability no less than $(1-\eta)$. Here, Q_η and C_η are constants that control the tail probability η; we refer the reader to the supplemental document for their exact definitions. For a fixed η, C_η depends only on η, while Q_η also depends on $\sqrt{M}/\Lambda_\mu$ but grows very slowly. In practice, when $\eta = 0.001$, $C_\eta \approx 4.5$; $Q_\eta \approx 4.1$ when $\sqrt{M}/\Lambda_\mu = 1$ and $Q_\eta \approx 5.4$ when $\sqrt{M}/\Lambda_\mu = 1000$.

Lastly, we set the step size Δ_i according to Equation 17, clamped to a lower bound $\Delta_{\min}$ (so that we eventually do find a zero crossing):

$$\Delta_i = \max\left(\frac{|\mu(t_i)| - Q_\eta \sigma}{\Lambda_\mu}, \frac{|f(t_i)|}{\Lambda_\mu + C_\eta\sqrt{M}}, \Delta_{\min}\right). \tag{18}$$

We validate our adaptive marching algorithm and demonstrate its accuracy and efficiency in Section 5.

ALGORITHM 1: GP root finding algorithm

Input: Stochastic function f, ray $(\boldsymbol{x}_0, \boldsymbol{\omega})$, maximal distance $t_{\max}$
Output: Intersection distance t (or $\perp$ if no intersection)

$v_0 \leftarrow$ `OnlineSampler`$(f, \boldsymbol{x}_0)$; `// Section 4.1`
$X \leftarrow \{\boldsymbol{x}_0\}; V \leftarrow \{v_0\}$; `// conditions` $f(X) = V$
$t_0 \leftarrow 0$;
for $i = 0, 1, 2, \ldots$ **do**
  $\Delta_i \leftarrow$ `AdaStepSize`(f, t_i, v_i) ; `// Equation 18`
  $t_{i+1} \leftarrow \min(t_{\max}, t_i + \Delta_i)$;
  $\boldsymbol{x}_{i+1} \leftarrow \boldsymbol{x}_0 + t_{i+1}\boldsymbol{\omega}$;
  $v_{i+1} \leftarrow$ `OnlineSampler`$(f, \boldsymbol{x}_{i+1} \mid X, V)$; `// Section 4.1`
  $X \leftarrow X \cup \{\boldsymbol{x}_{i+1}\}; V \leftarrow V \cup \{v_{i+1}\}$;
  if $v_i v_{i+1} \le 0$ **then**
    Find the root t using bracketed solver (or linear interpolation) within $[t_i, t_{i+1}]$;
    return t;
  if $t_i \ge t_{\max}$ **then**
    break;
return $\perp$;

Discussion

Exact sampling from a Gaussian process (GP) at n sample locations typically requires a dense Cholesky decomposition, which costs $O(n^3)$ time. As a result, both the baseline and our method have the same asymptotic per-ray intersection complexity, $O(n^3)$, where n denotes the number of GP samples (conditioning points) used along the ray. The key performance difference is in the effective n: the baseline relies on a dense uniform grid of sample locations to maintain robustness, whereas our adaptive strategy attains the same intersection accuracy using far fewer samples. In practice, this reduction in n dominates runtime, making our method substantially faster.

5 Results

We evaluate our GPIS-ray intersection method as a drop-in replacement in a GPIS rendering pipeline. We first validate the adaptive step size in a controlled setting (Section 5.1), then evaluate the accuracy and efficiency of our GPIS-ray intersection in isolation (Section 5.3), and finally measure end-to-end rendering improvements (Section 5.4).

Experiment setup. We implemented our method in C++ and integrated it into the rendering system by Seyb et al. [2024], which also serves as our baseline for comparison. Unless otherwise stated, both methods use double-precision arithmetic and share the same renderer, integrator, and scene configurations. All experiments are conducted on a workstation with an AMD Ryzen 9 7900X CPU (with 12 cores) and 32 GB of RAM.

To ensure fair comparisons, the baseline and our method use the same minimum step size $\Delta_{\min}$, chosen to satisfy $\mathrm{corr}(r(t), r(t + \Delta_{\min})) \ge 0.95$ and to be small enough to capture the high-frequency details of μ. The baseline uses the default parameters in the publicly available implementation by Seyb et al. [2024].

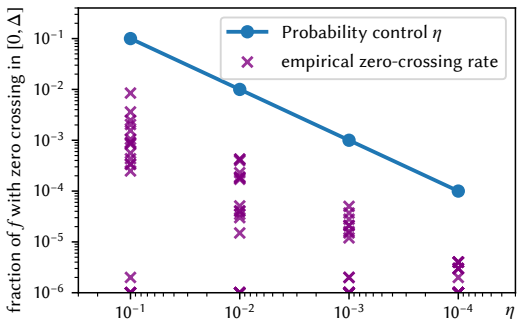


Fig. 4. Validation of the adaptive step-size. For each η, we randomize RBF hyper parameters and known value $f(0)$, then predict the stepsize Δ with risk of zero crossing on $[0, \Delta]$ less than η. Scatter points show the measured zero-crossing rate on the predicted safe interval $[0, \Delta]$ from Monte Carlo samples. We repeat 20 trials for each η. The blue line η is the target bound. All measured probabilities of zero crossings are below η.

Baseline method. We briefly recap the baseline algorithm (detailed in Section 3). The ray is cropped to a finite interval $[0, t_{\max}]$ using the approximate scene bound and discretized into a uniform grid with spacing $\Delta_{\min}$. Grid points with negligible marginal probability of lying near the zero level set are pruned via a fixed threshold. The remaining points are divided into segments of at most 32 points each, with inter-segment correlation approximated by the *Renewal+* model [Seyb et al. 2024]: only the value and derivative at each segment boundary are retained to couple neighboring segments. The algorithm then processes each segment sequentially, sampling the GP on its grid points and checking for a zero crossing.

Ground truth. Reference distributions and images in Section 5.3 and Section 5.4 are produced as follows. For stationary kernels, we use the *weight-space view* [Rahimi and Recht 2007] implemented by Seyb et al. [2024], which provides an independent validation of our method. For non-stationary kernels, where no weight-space method is available, we run our method with a very small conservative step size as reference.

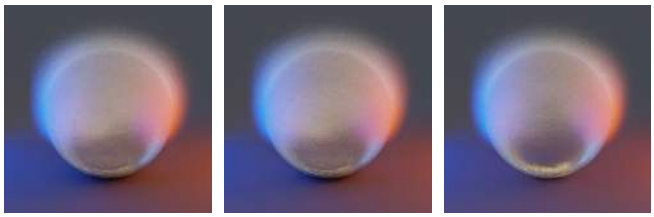
5.1 Empirical validation of adaptive step size

Because our step size is derived from a formal probabilistic analysis, it is guaranteed by construction under the stated assumptions, regardless of the specific form of the mean function or covariance function. Nonetheless, we empirically validate it on Gaussian processes with an RBF kernel to confirm that the resulting step sizes are indeed conservative.

To stress-test the step size, we choose a worst-case mean function $\mu(x) = 1 - x$ that saturates the Lipschitz bound ($\Lambda_\mu = 1$). Because μ decreases at the maximum admissible rate, it maximizes the probability of a zero crossing within each forward step.

For each threshold $\eta \in \{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$, we run 20 randomized trials. In each trial, we randomly sample RBF kernel hyperparameters and the initial known value $f(0)$. Given η and the bounds σ, M derived from the analytic kernel, we compute the step size Δ from Section 4.2, which guarantees that the probability of f having a zero crossing within $[0, \Delta]$ is at most η. We then draw 1,000,000 GP realizations on $[0, \Delta]$ and measure the fraction that exhibits at least one zero crossing.

Results. Figure 4 plots the empirical zero-crossing probability against η on log–log axes. Across all thresholds and randomized kernel settings, the observed frequency remains below η, confirming that the theoretically derived bound is conservative in the RBF case.



(a) 0.5×, 32m 5s (b) Ours, 15m 32s (c) 2×, 7m 13s

Fig. 5. Step-size ablation with render time shown under each image. We multiply the adaptive step size by a constant factor before applying the unchanged clamp $\Delta_{\min}$. A conservative 0.5× step size produces a result nearly indistinguishable from our default. A doubled step size, however, introduces visible differences because overly aggressive marching can skip zero crossings.

5.2 Effect of different step sizes

In the following, we examine how sensitive the rendering is to the marching step size. We use a GPIS with a non-stationary kernel and a thin-shell mean geometry, which makes missed intersections visually apparent. Starting from our adaptive step size, we multiply the step size by a constant factor before clamping with the unchanged minimum step size $\Delta_{\min}$.

Results. Figure 5 compares three settings: a conservative one taking 0.5× the adaptive step size, our default adaptive setting, and an aggressive doubled step-size setting, all with the same $\Delta_{\min}$. The conservative setting produces little visible change, suggesting that the default setting is already sufficiently conservative for this scene and that additional samples mostly increase computation rather than image quality. The aggressive setting, while faster, shows visible differences in the rendered shell. This indicates that increasing the step size beyond our probabilistic bound can miss some zero crossings and change the resulting intersection distribution. Thus, the default setting is already conservative and close to the largest usable step size before noticeable errors appear.

5.3 GPIS-ray intersection

In this section, we evaluate the correctness and efficiency of the GPIS-ray intersection method, separated from the downstream path-tracing algorithm.

A single ray query returns a free-flight distance t (or $t = t_{\max}$ if no intersection occurs within the scene bounds). By repeating the stochastic query many times on the same ray, we obtain an empirical free-flight distribution, which we compare to a high-accuracy reference.

We quantify distributional error with the Wasserstein distance [Villani et al. 2008] (earth mover's distance), which measures how

Table 1. Free-flight distribution accuracy and cost. For each camera ray, we repeatedly perform the stochastic GPIS–ray intersection query to estimate a distribution of free-flight distances, and compare it to a high-accuracy reference. We report results averaged over 32 randomly sampled camera rays: Wasserstein distance W_1 to the reference distribution, the average number of GP samples per ray, and run time per ray intersection. Lower is better for all metrics. The baseline method couldn't achieve an error as low simply by lowering the step size, because it discards some correlation along the ray (see [Seyb et al. 2024, Supplemental Sec. 3.2]). The "Baseline (equal time)" variant uses a larger fixed step size to match the run time of ours, illustrating the speed–accuracy tradeoff.

	W_1 Error	#Samples	Time/Ray (µs)
Ours	0.00044	6.0	37.9
Baseline	0.00509	74.8	2970.7
Baseline (equal time)	0.05664	12.6	37.6

much probability mass must be moved, and how far, to transform one distribution into another.

To summarize the experiment procedure:

(1) Randomly select a pixel whose camera ray intersects the GPIS.

(2) For this ray, query each method (ground truth, ours, baseline) for an intersection distance τ (or $t_{\max}$ if none).

(3) Repeat 100,000 times per method to build empirical distributions of τ.

(4) Compute the Wasserstein distance between each method's distribution and the ground truth.

We repeat this experiment on 32 randomly chosen camera rays across multiple object types and covariance functions.

Results. Table 1 shows results of this experiment including the errors of the free-flight distribution, the number of GP samples needed per intersection, and the GPIS-ray intersection time cost, averaged over tests. Besides the default baseline, we also report a *Baseline (equal time)* variant that uses a larger fixed step size to approximately match our run time, illustrating the accuracy–efficiency tradeoff.

Both our method and the default baseline stay close to the reference distribution, indicating that our method does not introduce extra bias despite using far fewer GP samples. When the baseline step size is increased to match our run time ("Baseline (equal time)"), its distributional error grows substantially, demonstrating that our method achieves a better accuracy–efficiency tradeoff. Efficiency-wise, our method requires substantially fewer GP samples per ray on average, translating into significantly lower per-ray intersection time. This speedup directly improves end-to-end rendering efficiency, which we evaluate next.

5.4 Rendering comparison

We evaluate the impact of faster GPIS-ray intersection on end-to-end path-traced renders.

For each scene, we render with both methods in the same framework under an equal time budget; the only difference is the GPIS-ray intersection routine.

We select scenes with diverse appearance characteristics and lighting, and include multiple types of stationary kernels and non-stationary kernels. We refer the reader to Seyb et al. [2024] for a catalog of supported kernel forms and how they can be authored. For each scene, we render a high-sample-count reference image and report MSE to this reference. We also record the achieved samples per pixel (spp) within the time budget.

Results. Figure 1 and Figure 6 show the equal-time comparison alongside a reference image (labeled as *Ground Truth*), with render time, number of samples, and image mean square error shown at the bottom of each render. Since Section 5.3 already evaluates bias (systematic error caused by finite GP samples), we focus here on rendering variance under a fixed time budget. Because our method reduces the cost of each GPIS-ray intersection, it traces more paths (higher spp) in the same time, resulting in consistently lower image error (MSE) across all tested scenes.

6 Discussion and Conclusion

Limitations and future work. Our probabilistic bounds are conservative in high-variance regions, which can limit step sizes and reduce speedups in complex scenes. We also focus on GPIS defined by standard kernels and conditioning schemes; extending the approach to more expressive priors or large-scale datasets may require new approximations. Integrating tighter, data-dependent bounds and scalable GP inference is a promising direction for further reducing queries per ray.

Conclusion. We have introduced an efficient ray marching algorithm for GPIS–ray intersection—the central bottleneck in GPIS-based rendering. By combining online Gaussian sampling with probabilistic step-size bounds, our method dramatically reduces per-ray cost while preserving correctness, improving rendering MSE by up to 46× at equal time.

Acknowledgments

We thank the anonymous reviewers for their constructive suggestions. This work was partially supported by NSF grant 2553564.

References

Melike Aydinlilar and Cédric Zanni. 2023. Forward inclusion functions for ray-tracing implicit surfaces. *Computers & Graphics* 114 (2023), 190–200.

Benedikt Bitterli, Srinath Ravichandran, Thomas Müller, Magnus Wrenninge, Jan Novák, Steve Marschner, and Wojciech Jarosz. 2018. A radiative transfer framework for non-exponential media. (2018).

Richard P. Brent. 1971. An algorithm with guaranteed convergence for finding a zero of a function. *The computer journal* 14, 4 (1971), 422–425.

Ken Brodlie, Rodolfo Allendes Osorio, and Adriano Lopes. 2012. A review of uncertainty in data visualization. *Expanding the frontiers of visual analytics and visualization* (2012), 81–109.

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*. IEEE, 2845–2850.

Nathaniel Fout and Kwan-Liu Ma. 2012. Fuzzy volume rendering. *IEEE Transactions on Visualization and Computer Graphics* 18, 12 (2012), 2335–2344.

Eric Galin, Eric Guérin, Axel Paris, and Adrien Peytavie. 2020. Segment tracing using local lipschitz bounds. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 545–554.

Manuel Gamito. 2009. *Techniques for stochastic implicit surface modelling and rendering*. Ph. D. Dissertation. University of Sheffield.

Philip E Gill, Gene H Golub, Walter Murray, and Michael A Saunders. 1974. Methods for modifying matrix factorizations. *Mathematics of computation* 28, 126 (1974), 505–535.

John C Hart. 1996. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer* 12, 10 (1996), 527–545.

Adrian Jarabo, Carlos Aliaga, and Diego Gutierrez. 2018. A radiative transfer framework for spatially-correlated materials. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–13.

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. doi:10.1145/15922.15902

Devendra Kalra and Alan H Barr. 1989. Guaranteed ray intersections with implicit surfaces. *ACM Siggraph Computer Graphics* 23, 3 (1989), 297–306.

Benjamin Keinert, Henry Schäfer, Johann Korndörfer, Urs Ganse, and Marc Stamminger. 2014. Enhanced sphere tracing. (2014).

Aaron Knoll, Younis Hijazi, Andrew Kensler, Mathias Schott, Charles Hansen, and Hans Hagen. 2009. Fast ray tracing of arbitrary implicit surfaces with interval and affine arithmetic. In *Computer Graphics Forum*, Vol. 28. Wiley Online Library, 26–40.

Johannes Meng, Marios Papas, Ralf Habel, Carsten Dachsbacher, Steve Marschner, Markus Gross, and Wojciech Jarosz. 2015. Multi-scale modeling and rendering of granular materials. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 1–13.

Bailey Miller, Hanyu Chen, Alice Lai, and Ioannis Gkioulekas. 2024. Objects as volumes: A stochastic geometry view of opaque solids. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 87–97.

Don P Mitchell. 1990. Robust ray intersection with interval arithmetic. In *Proceedings of Graphics Interface*, Vol. 90. 68–74.

Jonathan T Moon, Bruce Walter, and Stephen R Marschner. 2007. Rendering discrete random media using precomputed scattering solutions. In *Proceedings of the 18th Eurographics conference on Rendering Techniques*. 231–242.

Thomas Müller, Marios Papas, Markus Gross, Wojciech Jarosz, and Jan Novák. 2016. Efficient rendering of heterogeneous polydisperse granular media. *ACM Transactions on Graphics (TOG)* 35, 6 (2016), 1–14.

Tobias Pfaffelmoser, Matthias Reitinger, and Rüdiger Westermann. 2011. Visualizing the positional and geometrical variability of isosurfaces in uncertain scalar fields. In *Computer graphics forum*, Vol. 30. Wiley Online Library, 951–960.

Ali Rahimi and Benjamin Recht. 2007. Random features for large-scale kernel machines. *Advances in neural information processing systems* 20 (2007).

Silvia Sellán and Alec Jacobson. 2022. Stochastic Poisson Surface Reconstruction. *ACM Transactions on Graphics* (2022).

Silvia Sellán and Alec Jacobson. 2023. Neural Stochastic Screened Poisson Reconstruction. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)* (2023).

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 Transactions on Graphics (Proceedings of SIGGRAPH)* 43, 4 (July 2024). doi:10/gt5nh9

Dario Seyb, Alec Jacobson, Derek Nowrouzezahrai, and Wojciech Jarosz. 2019. Nonlinear sphere tracing for rendering deformed signed distance fields. *ACM Transactions on Graphics* 38, 6 (2019).

Nicholas Sharp and Alec Jacobson. 2022. Spelunking the deep: Guaranteed queries on general neural implicit surfaces via range analysis. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–16.

Cédric Villani et al. 2008. *Optimal transport: old and new*. Vol. 338. Springer.

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. In *Gaussian Processes in Practice*.

Tong Wu, Zhiqiang Zhou, Anlan Wang, Yuning Gong, and Yanci Zhang. 2022. A Real-Time Adaptive Ray Marching Method for Particle-Based Fluid Surface Reconstruction.. In *EGSR (ST)*. 71–79.

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). doi:10.1145/3763329

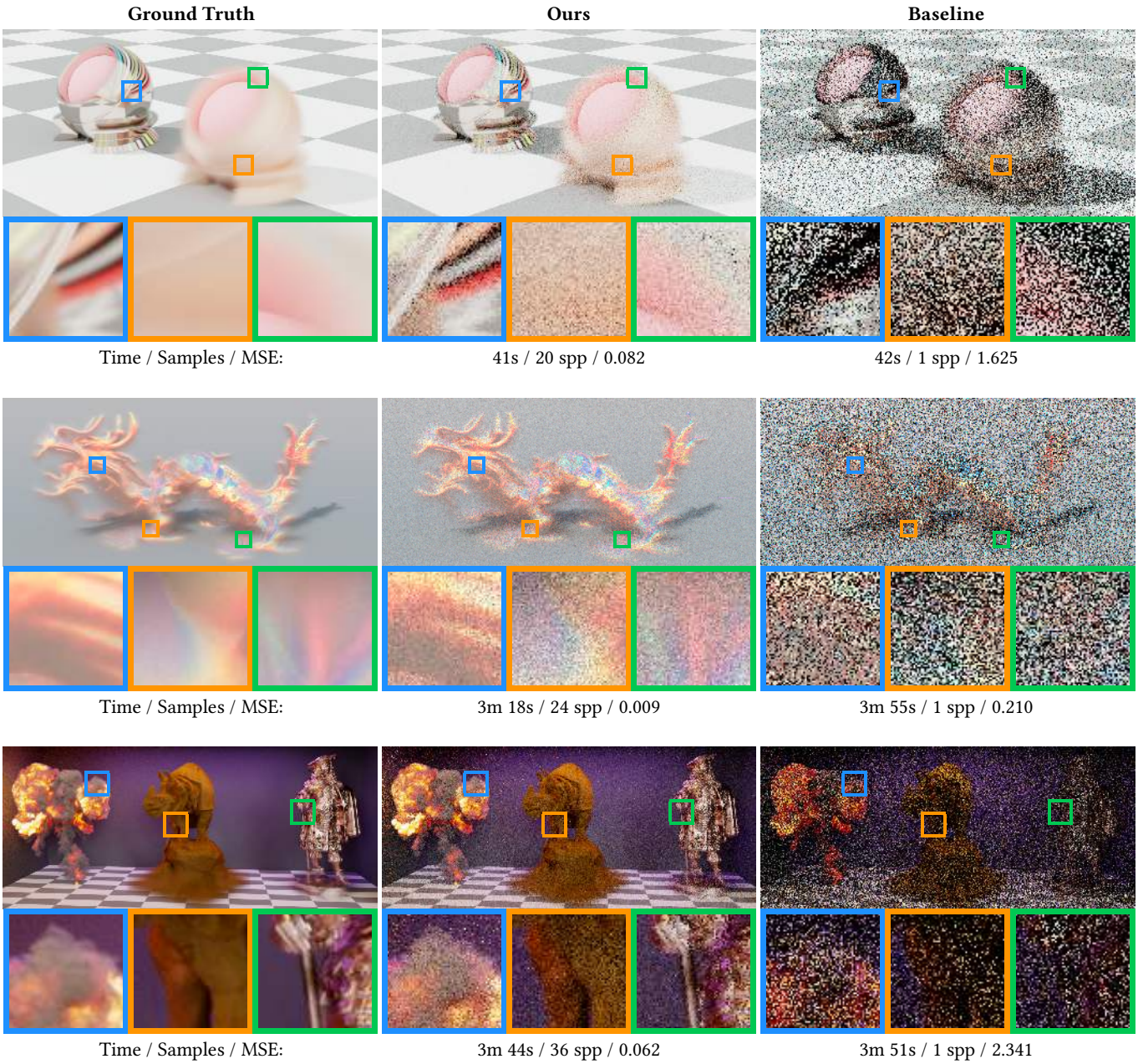


Fig. 6. We evaluate our GPIS-ray intersection method in a path-tracing renderer, comparing it to the baseline method across multiple scenes with various types of covariance functions. We render each scene with both methods at equal time, using the same renderer and scene settings. Our method enables more samples per pixel in the same time budget, resulting in significantly lower image noise.