Gaussian Integral Linear Operators for Precomputed Graphics

HAOLIN LU, Max Planck Institute for Informatics, Germany
YASH BELHE, University of California San Diego, USA
GURPRIT SINGH, Max Planck Institute for Informatics, Germany
TZU-MAO LI, University of California San Diego, USA
TOSHIYA HACHISUKA, University of Waterloo, Canada

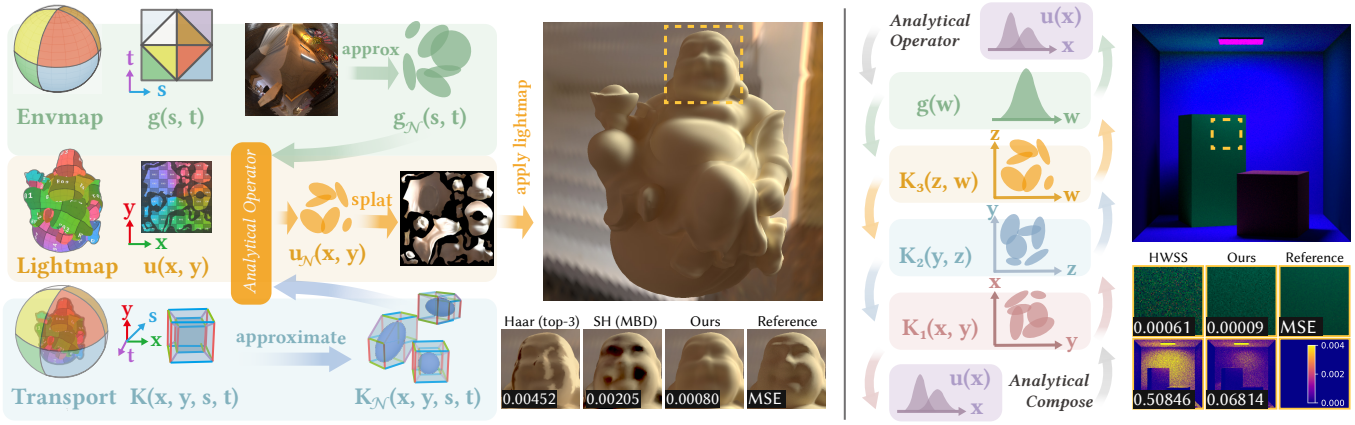


Fig. 1. *Integral linear operators* are widely used across many graphics problems. **(left)** In relighting, the environment map can be represented as a 2D function $g(s,t)$, and the resulting surface irradiance can be parameterized as another 2D function $u(x,y)$. The light transport can then be described by a 4D kernel function $K(x,y,s,t)$, such that $u(x,y) = \iint K(x,y,s,t)g(s,t)\,\mathrm{d}s\,\mathrm{d}t$. In our method, the kernel K is approximated as a 4D Gaussian mixture. Given an environment map represented as a 2D Gaussian mixture, the resulting irradiance can be analytically computed as another 2D Gaussians mixture, which are then splatted to generate the corresponding lightmap. Unlike typical precomputed radiance transfer (PRT) techniques, our approach decouples kernel storage from the mesh or spatial grid, allowing for more flexible representation. In an equal-memory comparison, it can achieve higher-quality relighting results. **(right)** Our method naturally supports the sequential application of multiple operators, as both input and output are consistently represented in the same basis family of Gaussians. Furthermore, we show that the composition of two operators can also be analytically computed and remains closed under the Gaussian family. This property also allows the problem to be solved in reverse order, which is beneficial in certain scenarios. As an application, we can solve the fluorescent material interactions in spectral rendering using our closed-form approximation, resulting in significant color noise reduction under equal time.

Integral linear operators play a key role in many graphics problems, but solutions obtained via Monte Carlo methods often suffer from high variance. A common strategy to improve the efficiency of integration across various inputs is to precompute the kernel function. Traditional methods typically rely on basis expansions for both the input and output functions. However, using fixed output bases can restrict the precision of output reconstruction and limit the compactness of the kernel representation. In this work, we introduce a new method that approximates both the kernel and the input function using Gaussian mixtures. This formulation allows the integral operator to be evaluated analytically, leading to improved flexibility in kernel storage and output representation. Moreover, our method naturally supports the sequential application of multiple operators and enables closed-form operator composition, which is particularly beneficial in tasks involving chains of operators. We demonstrate the versatility and effectiveness of our approach across a variety of graphics problems, including environment map relighting, boundary value problems, and fluorescence rendering.

CCS Concepts: • **Computing methodologies** → **Rendering**.

Authors' addresses: Haolin Lu, Max Planck Institute for Informatics, Germany; Yash Belhe, vicini@google.com, University of California San Diego, USA; Gurprit Singh, Max Planck Institute for Informatics, Germany; Tzu-Mao Li, University of California San Diego, USA; Toshiya Hachisuka, University of Waterloo, Canada.

Additional Key Words and Phrases: linear operator, integral operator, gaussian, precomputed radiance transfer, partial differential equations

1 INTRODUCTION

Many problems in computer graphics, such as light transport and boundary value problems, can be formulated using an *integral linear operator* $\mathcal{K}$. This operator maps an input function $g(\mathbf{y})$ to an output function $u(\mathbf{x})$ using an integral,

$$u(\mathbf{x}) = \mathcal{K}[g(\mathbf{y})] = \int K(\mathbf{x},\mathbf{y})g(\mathbf{y})\,\mathrm{d}\mathbf{y}, \tag{1}$$

where $K(\mathbf{x},\mathbf{y})$ is a kernel function that defines an operator. There is typically no analytical solution to the above integral, and one needs to rely on numerical approaches to estimate the output $u(\mathbf{x})$.

Monte Carlo (MC) integration is one such numerical approach which performs pointwise estimation of $u(\mathbf{x}_p)$ as an average over stochastic estimates, given a specific point $\mathbf{x}_p$. The resulting estimate is noisy due to stochastic sampling, and it is computationally costly due to multiple evaluations of the kernel $K(\mathbf{x},\mathbf{y})$. Instead, many prior works in computer graphics have explored a method to *precompute* the integral linear operator for a given kernel function, allowing it to approximately take any arbitrary input function during runtime. A

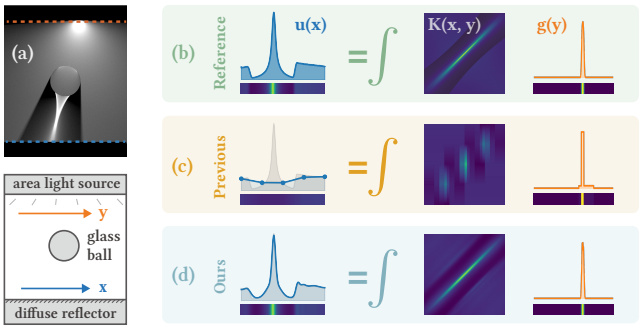


Fig. 2. Toy 1D light transport. **(a)** A 1D diffuse reflector is illuminated by a 1D area light, with a glass ball in between. **(b)** The reflector's irradiance $u(x)$ is obtained by applying a radiance transfer operator K to the light emission $g(y)$. **(c)** A common approach expands both input and output in predefined bases – for example, Haar wavelets with thresholding for g and tent functions for u here – so that the kernel can be represented by a coefficient matrix. However, a poor choice of output basis can severely limit the accuracy of the approximation. **(d)** Instead, we approximate the operator directly with 2D Gaussians using the same memory as in (c). This allows u to be solved analytically without predefined bases, and the result is accurate as long as the approximations of K and g are good. Note that handling general 3D caustics is more involved in terms of fitting the signals and oeprators, and this example is meant to be purely didactic.

prominent example is precomputed radiation transfer (PRT) [Sloan et al. 2002], which precomputes and approximates the light transport operator $\mathcal{K}$ for a given fixed scene. An input environment map $g(\mathbf{y})$ (i.e., a light source) is allowed to change at runtime, enabling efficient evaluation of the solution of light transport $u(\mathbf{x})$ for a given scene.

In PRT, the input $g(\mathbf{y})$ and the output $u(\mathbf{x})$ are both approximated by *basis expansions*, so that the kernel can be approximated as a matrix to convert a vector of coefficients of the input basis to a vector of coefficients of the output basis. Different bases have been proposed to improve its accuracy and efficiency in different scenarios [Ng et al. 2003; Tsai and Shih 2006; Xu et al. 2013, 2022]. Beyond light transport, functional maps [Ovsjanikov et al. 2012] formulate the operator $\mathcal{K}$ for various geometry processing tasks using the same basis expansion idea. Recent neural approaches [Azizzadenesheli et al. 2024] represent the operator $\mathcal{K}$ by a neural network by taking an encoding of the input $g(\mathbf{y})$ and samples of the expected output values $u(\mathbf{x})$ and fits a network to approximate the operator $\mathcal{K}$.

All those precomputation approaches involve several approximations. Basis expansions in PRT and functional maps approximate both the input and the output as a set of coefficients of given basis functions so that the kernel becomes a matrix. This formulation introduces approximation errors when basis expansions cannot represent functions well. For instance, spherical harmonics used in PRT are known to be inaccurate at representing high-frequency features. A subtle, but important issue is that an accurate basis expansion of the input alone (e.g., spherical harmonics expansion of a smooth environment map) may not yield an accurate basis expansion of the output (e.g., sharp contact shadows under a smooth illumination). Fig. 2 illustrates a didactic example with a 2D operator. In the subfigure (c), the output is expanded using the tent basis, which struggles to capture sharp features. Since the input can change arbitrarily at runtime and the corresponding output is available only after evaluating the integral linear operator, it is challenging to determine a priori which basis functions can accurately represent the output $u(\mathbf{x})$. Neural approaches do not fundamentally circumvent this problem, as they still require a suitable encoding of the input function, and their generalization beyond trained examples is unpredictable.

We propose a new representation of integral linear operators based on analytical integrals with a basis expansion of the *kernel*. Similar to basis expansions, our representation approximates both the input and the output as sums of basis functions. The key difference from the existing basis expansions is that we also apply a basis expansion to the kernel $K(\mathbf{x}, \mathbf{y})$ as a higher dimensional function and utilize analytical integration of a product of basis functions. We propose to use Gaussian as a basis which allows us to perform such analytical integration. The basis expansion of the output in our method emerges from the analytical integral of the operator in a posterior manner given the input basis and the kernel basis. This property allows the output basis to change according to the input basis and the kernel basis on the fly, making it adaptive to the basis expansions for both. One can also concatenate multiple operators into a single operator by analytical integration over multiple kernels. Since the only approximations we introduce are basis expansions of the input and the kernel, the output becomes accurate whenever the input and the kernel are accurately represented by basis expansions. In contrast, the existing basis expansion approaches need to define a set of bases for the output apriori, which fundamentally limits the accuracy of the output even when the input and the kernel are accurately represented. Our representation of the kernel as a higher dimensional function is also often more compact than a matrix for coefficients in the existing basis expansion approaches whenever there is spatial coherence in the space of the variables of the $K(\mathbf{x}, \mathbf{y})$. As shown in Fig. 2(d), under equal memory constraint, directly approximating the kernel K with 2D Gaussians yields a more accurate operator approximation and, thus, a more precise output prediction. We demonstrate the practical improvements our new operator representation brought in multiple applications. To summarize, our technical contributions are:

- We propose directly representing kernel functions as high-dimensional Gaussian mixtures, enabling closed-form evaluation of integral operators.
- We show that our Gaussian mixture representation supports closed-form composition of multiple integral operators.
- We demonstrate how our theory applies to various graphics applications, supported by practical infrastructures, including the tile-based differentiable rasterizer, culling, an unbiased stochastic summation estimator, etc.

An open-source implementation of our method is available under https://github.com/suikasibyl/gilo.

2 RELATED WORK

Precomputed radiance transfer. PRT [Sloan et al. 2002] precomputes the light transport kernel K for efficient relighting with the environment map. A significant body of research has focused on

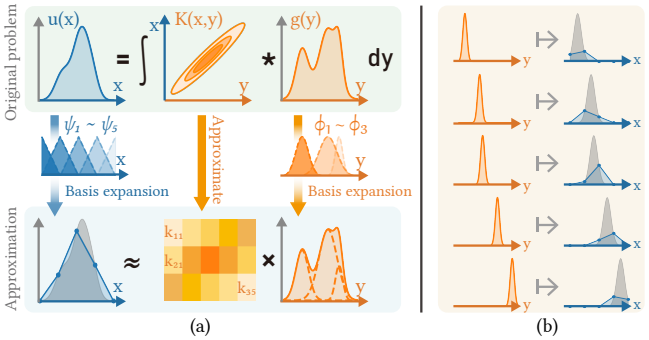


Fig. 3. Traditional methods use fixed output bases. **(a)** In classical methods, the integral linear operator is approximated by expanding both the input g and output u using basis functions ϕ_i and ψ_j respectively. Here, the input bases ϕ_i are Gaussians, while the output bases ψ_j are tent functions defined on a uniform grid. Then the kernel K can be represented as a coefficient matrix k_{ij}. **(b)** In many cases, the input g can be highly dynamic, leading to significant variation in the output u. Consequently, designing a fixed set of output bases ψ_j that can accurately represent all possible outputs is generally challenging. In the plot, the gray curve represents the ground-truth output, while the blue curve shows an inaccurate approximation reconstructed from fixed tent basis functions.

developing various basis functions for the input (directional) domain $\mathcal{Y}$, including spherical harmonics (SH), spherical Gaussians (SG) [Tsai and Shih 2006; Xu et al. 2013], nonlinear wavelets [Ng et al. 2003], and even neural basis [Xu et al. 2022]. The main focus of those prior work is to estimate an integral $u(\mathbf{x}_i) = \int K(\mathbf{x}_i, \mathbf{y}) g(\mathbf{y})\, \mathrm{d}\mathbf{y}$ evaluated at a given point $\mathbf{x}_i$. We instead focus on a closed-form solution to the operator $\mathcal{K}$ itself, yielding an output function $u(\mathbf{x})$ rather than a single value $u(\mathbf{x}_i)$ at $\mathbf{x}_i$.

Křivánek et al. [2004] proposed using an adaptive mesh to capture high-frequency details in the output through training-time adaptive subdivision. While adaptive to a given scene, their method ultimately still uses a fixed set of output basis functions at runtime. In contrast, our approach dynamically adapts the output basis to the input basis on the fly. Clustered principal component analysis [Sloan et al. 2003] and moving basis decomposition [Silvennoinen and Sloan 2021] exploit spatial coherence to compress coefficients at different $\mathbf{x}_i$, implicitly performing a basis expansion in the space $\mathcal{X}$ of the output function $u(\mathbf{x})$. These approaches are effective with SH coefficients and assume a fixed set of basis across different $\mathbf{x}_i$. Spherical Gaussians and nonlinear wavelets introduce varying basis functions across different $\mathbf{x}_i$, thus it is nontrivial to express them by principal component analysis or moving basis decomposition. As we will demonstrate later, our method often compactly captures the correlation between $\mathbf{x}$ and $\mathbf{y}$ by considering the basis expansion in the higher dimensional space of the kernel $K(\mathbf{x}, \mathbf{y})$.

Functional maps. Functional maps [Ovsjanikov et al. 2012] model non-rigid shape matching as linear operators between function spaces on manifolds. Both input and output functions are represented in predefined bases, typically Laplacian eigenbases, and the operator is expressed as a matrix, that linearly maps input coefficients to output coefficients. Unlike in light transport, here the kernel K may not be explicitly defined or directly estimated; instead, it can be implicitly determined by the choice of descriptor functions. For instance, correspondences can be improved by enforcing additional constraints [Hu et al. 2021; Magnet et al. 2022; Nogneng and Ovsjanikov 2017; Ren et al. 2018] or by designing handcrafted [Huska et al. 2023; Salti et al. 2014; Yan et al. 2023] or learned features [Halimi et al. 2019; Litany et al. 2017; Roufosse et al. 2019]. We assume the kernel K is well-defined and can be estimated pointwise in the space of $\mathcal{X} \times \mathcal{Y}$, which motivates us to directly approximate the kernel itself with high-dimensional Gaussians.

Neural approaches. Neural methods have also been used to solve operator problems. Operator learning [Anandkumar et al. 2019; Kovachki et al. 2021; Li et al. 2021; Lu et al. 2021] fits general functional mappings using neural networks, but are typically not tailored for linear integral operators. In graphics, neural networks have been used to approximate light transport [Rainer et al. 2022; Ren et al. 2013], and some methods preserve linearity by maintaining the basis expansion of the input g [Lyu et al. 2022; Raghavan et al. 2023; Yang et al. 2023]. While these methods demonstrate strong approximation capabilities, they typically represent the kernel K implicitly based on neural encoding of the input function and require relatively expensive inference. Moreover, these methods usually opt for point-wise output queries at specific $\mathbf{x}$, making them less compatible with the recursive composition of multiple operators. Our method explicitly approximates the kernel K, naturally preserves linearity, and is directly compatible with recursive operators.

Gaussian in graphics. Gaussian distributions are widely used in rendering for their ability of compact function approximation [Jakob et al. 2011], and support for efficient sampling [Herholz et al. 2018; Hua et al. 2023; Vorba et al. 2014; Yan et al. 2016] and product sampling [Herholz et al. 2016; Xia et al. 2020]. Key properties and operations of high-dimensional Gaussians are provided in the supplementary material. Dodik et al. [2022] used high-dimensional Gaussians to capture spatial-directional correlations, and we also aim to capture the input-output correlations via Gaussians. Kerbl et al. [2023] introduced a differentiable tile-based rasterizer for efficient reconstruction and rendering of Gaussians, which we used in our pipeline. Our work is inspired by recent advances in high-dimensional Gaussian fitting [Diolatzis et al. 2024; Gao et al. 2025; Yang et al. 2024]. We exploit a less used feature of Gaussians; they support closed-form integration of two Gaussians with different dimensions, whereas most methods typically consider the integral of two Gaussians with the same dimensionality. To the best of our knowledge, this feature has not been explored previously, even beyond the domain of computer graphics.

3 INTEGRAL LINEAR OPERATORS IN GRAPHICS

Integral linear operators (Eq. (1)) serve as a unifying abstraction for many problems in computer graphics. For example, the rendering equation uses a radiance transfer kernel K to map emission g to outgoing radiance u, while boundary value problems involve a Poisson kernel $\mathcal{P}$ [Krantz et al. 1999] that maps boundary conditions g to a solution u. In both cases, the kernel can typically only be computed pointwise and its integral is estimated by MC integration, making

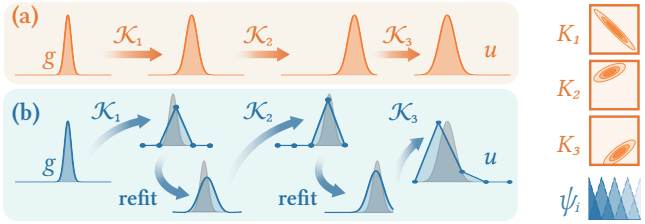


Fig. 4. Challenges in applying multiple operators. **(a)** We consider the scenario where an input function g is sequentially transformed by three operators: $\mathcal{K}_1$, $\mathcal{K}_2$, and $\mathcal{K}_3$. In this specific case where all operators are exact 2D Gaussians, our approach can compute the output of each operator precisely, without any need for reprojection or incurring approximation error. **(b)** In traditional methods, if Gaussians are used as input bases ϕ and tent functions are used as output bases ψ, the resulting output cannot be directly reused as input for the next operator. Instead, it must be re-projected onto the input basis ϕ, introducing additional approximation error and computational overhead. The gray curves show the reference result, while the blue curves are the inaccurate approximation obtained.

it advantageous to precompute the kernel for a variety of inputs. A common strategy is to approximate both input $g(\mathbf{y})$ and output $u(\mathbf{x})$ as weighted sums of chosen basis functions ϕ and ψ, as illustrated in Fig. 3:

$$g(\mathbf{y}) \approx \sum_i \mathrm{G}_i \phi_i(\mathbf{y}), \tag{2}$$

$$u(\mathbf{x}) \approx \sum_j \mathrm{U}_j \psi_j(\mathbf{x}), \tag{3}$$

where G_i and U_i are scalar coefficients. At runtime, the output $u(\mathbf{x})$ is approximated as:

$$u(\mathbf{x}) \approx \int K(\mathbf{x},\mathbf{y}) \underbrace{\sum_i \mathrm{G}_i \phi_i(\mathbf{y})}_{\approx g(y)} \mathrm{d}\mathbf{y} = \sum_i \mathrm{G}_i \int K(\mathbf{x},\mathbf{y}) \phi_i(\mathbf{y}) \,\mathrm{d}\mathbf{y} \tag{4}$$

$$\approx \sum_i \mathrm{G}_i \underbrace{\left(\sum_j k_{ij} \psi_j(\mathbf{x}) \right)}_{\approx \int K(\mathbf{x},\mathbf{y}) \phi_i(\mathbf{y}) \,\mathrm{d}\mathbf{y}} = \sum_j \underbrace{\left(\sum_i \mathrm{G}_i k_{ij} \right)}_{\mathrm{U}_j} \psi_j(\mathbf{x}), \tag{5}$$

where the linear integral operator K is applied to each input basis ϕ_i, and the result is projected onto the predefined output bases ψ_j. The kernel can be compactly represented as a precomputed coefficient matrix k_{ij}, which encodes the contribution of each ϕ_i to each ψ_j. Approximation of the product integral with the kernel (Eq. (5)) introduces additional approximation error besides basis expansions. Designing bases ψ_j to reduce this additional error is challenging since the output u depends on the runtime input g.

The challenge becomes even more pronounced in scenarios that involve recursively applying a sequence of integral linear operators $\mathcal{K}_l$, $l = 1, \ldots, N$:

$$u = \mathcal{K}_N[\mathcal{K}_{N-1}[\cdots \mathcal{K}_1[g]]], \tag{6}$$

where the sequence of kernels is determined only at runtime, allowing precomputation only for individual kernels. For example, in spectral rendering, each interaction with a surface material acts as a linear operator; and in layered materials, each layer's BSDF acts as a linear operator. Since the sampled path is determined by a stochastic random walk, the sequence of operators applied varies across paths and is only determined after the path has been sampled.

As illustrated in Fig. 4, applying multiple operators introduces several challenges. First, when the input and output bases ϕ and ψ belong to different families, additional re-projection steps are required to bridge them, increasing both computational cost and approximation error. While using the same basis family for both input and output can avoid re-projection, designing the output bases ψ_j also becomes even more difficult, as the outputs are not only dependent on the dynamic input g, but also on the stochastic sequence of kernels applied K_l. The approximation error will accumulate throughout the sequence, so it would be even more important to keep the approximation error of every operator as low as possible.

4 Analytical Integral Linear Operator with a Higher Dimensional Basis Expansion

4.1 High-dimensional basis for the kernel

In many applications, the kernel K is either known analytically or can be estimated pointwise, while the output u is generally unknown and varies dynamically with the input. Instead of attempting to improve the output basis expansion, we turn our attention to the kernel K and expand it directly using *high-dimensional* basis functions over the joint output-input domain $(\mathbf{x}, \mathbf{y})$:

$$K(\mathbf{x},\mathbf{y}) \approx \sum_i \mathrm{K}_i \cdot \sigma_i(\mathbf{x},\mathbf{y}), \tag{7}$$

where K_i are scalar coefficients, and σ_i are basis functions defined over an N_k dimensional space, with $N_k = N_\mathbf{x} + N_\mathbf{y}$.

Similar to the previous work, we also approximate input g using basis expansion as Eq. (2), and compute output u as:

$$\mathcal{K}[g] \approx \int \left(\sum_i \mathrm{K}_i \sigma_i(\mathbf{x},\mathbf{y}) \right) \left(\sum_i \mathrm{G}_i \phi_i(\mathbf{y}) \right) \mathrm{d}\mathbf{y} \tag{8}$$

$$= \sum_i \sum_j \mathrm{K}_i \mathrm{G}_j \int \sigma_i(\mathbf{x},\mathbf{y}) \phi_j(\mathbf{y}) \,\mathrm{d}\mathbf{y} \tag{9}$$

$$= \sum_i \sum_j \mathrm{K}_i \mathrm{G}_j \xi_{ij}(\mathbf{x}). \tag{10}$$

While the output u is still represented as basis expansion, its basis $\xi(\mathbf{x})$ are determined *a posteriori* based on the input as:

$$\xi_{ij}(\mathbf{x}) = \int \sigma_i(\mathbf{x},\mathbf{y}) \phi_j(\mathbf{y}) \,\mathrm{d}\mathbf{y}. \tag{11}$$

In this paper, we propose to use $N_\mathbf{y}$-dimensional Gaussians as input bases ϕ and N_K-dimensional Gaussians as kernel bases σ, enabling the integral ξ to be analytically evaluated as an unnormalized $N_\mathbf{x}$-dimensional Gaussian, as detailed in the supplementary material. A toy example with $N_\mathbf{x} = N_\mathbf{y} = 1$ is shown in Fig. 5. Unlike Eq. (5), our method avoids manually designing an output basis ψ and eliminates additional approximation error of the product integral in Eq. (5) since the output basis ξ is derived in closed form.

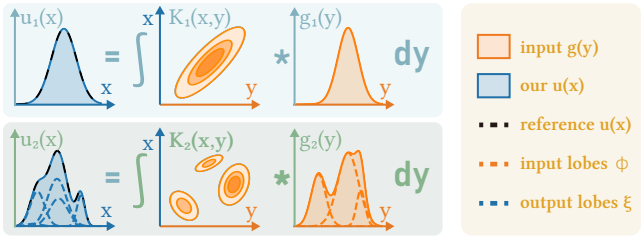


Fig. 5. Our method, Single operator. **(1st row)** The integral of a 2D Gaussian kernel with a 1D Gaussian input yields a weighted 1D Gaussian output. **(2nd row)** When both the kernel K and input g are represented as Gaussian mixtures, the output u is also a Gaussian mixture, with each lobe computed in closed form. Since our output Gaussian lobes are dynamically determined by the Gaussian mixtures of the input and kernel, they adapt to the output shapes better unlike traditional methods (Fig. 3).

4.2 Operator composition

We now turn to another key application scenario of our method, where multiple operators $\mathcal{K}_l$ are applied sequentially, as described in Eq. (6) and Fig. 1 (right). When each kernel K_l is represented using our high-dimensional Gaussians, the output of each operator remains a Gaussian mixture. The output can thus be used directly as the input for the next operator, without incurring additional computational cost or approximation error beyond the kernel basis expansion, as shown in Fig. 6(a).

In certain scenarios, it is also useful to compose the operators (i.e., to find a composite operator $\mathcal{K}_{12} = \mathcal{K}_1 \circ \mathcal{K}_2$) such that: $\mathcal{K}_{12}[g] = \mathcal{K}_1[\mathcal{K}_2[g]]$. The new kernel K_{12} is defined by the following integral:

$$K_{12}(\mathbf{x}, \mathbf{z}) = \int K_1(\mathbf{x}, \mathbf{y}) K_2(\mathbf{y}, \mathbf{z})\, \mathrm{d}\mathbf{y}. \tag{12}$$

This arises, for example, in spectral rendering, where a light path may interact with several materials before reaching the light source. We demonstrate that when each kernel K_l is approximated as an N_K-dimensional Gaussian, the composite operator can also be computed in closed form, result in a new group of Gaussians. This property enables efficient evaluation of multi-operator problems in reverse order, as illustrated in Fig. 6(b). The full derivation and resulting expressions are provided in the supplementary material.

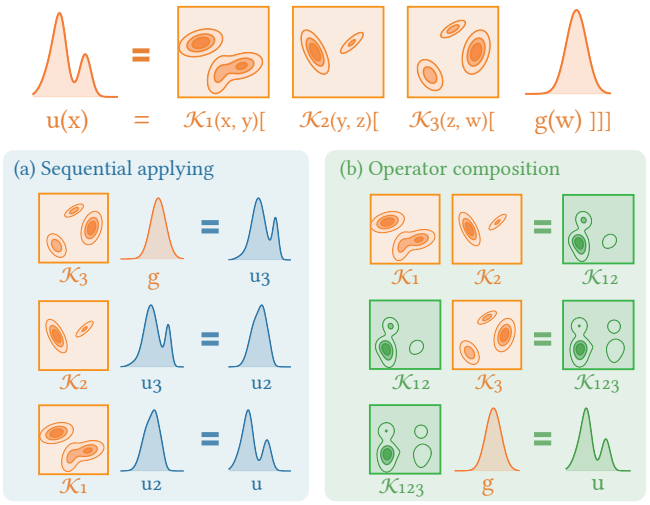


Fig. 6. Our method, Multiple operators. We discuss two ways to solve problems involving multiple operators. **(a)** A straightforward approach is to apply the operators sequentially to the input: first, apply operator $\mathcal{K}_3$ to the input g, yielding the intermediate output u_3. Then, apply operators $\mathcal{K}_2$ and $\mathcal{K}_1$ successively to obtain the final output u, using the closed-form solution illustrated in Fig. 5. **(b)** Alternatively, the problem can be solved in reverse order by first composing $\mathcal{K}_1$ and $\mathcal{K}_2$ to obtain an equivalent operator $\mathcal{K}_{12}$, and then further composing it with $\mathcal{K}_3$ to form $\mathcal{K}_{123}$. Finally, by analytically applying $\mathcal{K}_{123}$ to input g, we obtain the same output u. This pipeline is enabled by another key feature of our method: *Gaussian operators can be analytically composed*, and the result remains a Gaussian mixture.

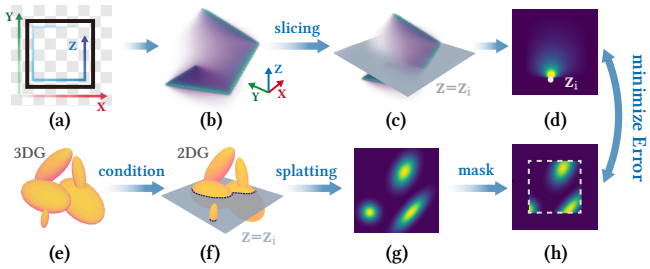


Fig. 7. Fitting pipeline of precomputed Poisson kernel. (a) Given a rectangular Dirichlet boundary, we parameterize the image domain by (x, y) and the boundary curve by z. (b) The corresponding Poisson kernel is a 3D function $\mathcal{P}(x, y, z)$. (c) Each slice of $\mathcal{P}$ at $z = z_i$ represents (d) the solution to the PDE with a Dirac delta source placed at boundary point z_i. (e) We approximate $\mathcal{P}$ using a 3D Gaussian mixture. (f) During training, we condition this mixture on z_i to obtain 2D Gaussians, (g) which are rasterized onto the image plane using a differentiable rasterizer. (h) We then mask out regions outside the domain and minimize the error between the rendered Gaussians and the corresponding kernel slice.

5 APPLICATION: ONE INTEGRAL LINEAR OPERATOR

While our theoretical formulation provides a general foundation, the way in which the kernel is approximated, and how its analytical properties are used, varies by application. In the following sections, we discuss three main application scenarios.

5.1 Precomputed Poisson kernel

We first start from a lower-dimensional application: the Dirichlet problem for the Laplace's equation in 2D, formulated as

$$u(x, y) = \int \mathcal{P}(x, y, z) \cdot g(z) \mathrm{d}z, \tag{13}$$

where the Poisson kernel $\mathcal{P}$ maps the boundary condition g to the harmonic solution u. By precomputing the Poisson kernel $\mathcal{P}$ for a fixed boundary geometry, we enable efficient computation of the solution $u(x, y)$ for varying input boundary conditions $g(z)$,

Fitting. As shown in Fig. 7, for a 2D boundary value problem, the Poisson kernel forms a 3D function, which we approximate using a 3D Gaussian mixture. For precomputation, each iteration

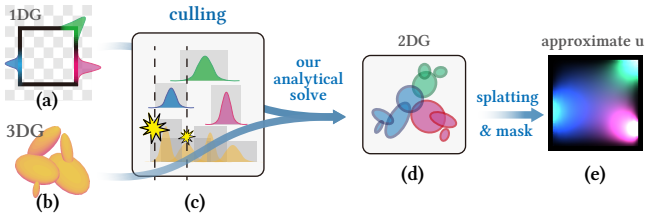


Fig. 8. Runtime pipeline of precomputed Poisson kernel. At runtime, (a) we approximate the boundary values using 1D Gaussians and combine them with (b) the learned 3D Gaussian operators. (c) After culling low-contribution pairs, (d) we compute 2D Gaussians using our analytical integral operator, and (e) render the final result via Gaussian splatting.

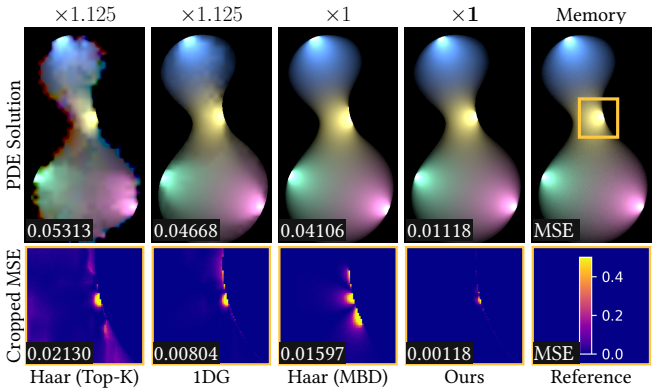


Fig. 9. Comparison with Fixed Output Basis. We compare approximate solutions under equal memory budgets for fitting the Poisson kernel $\mathcal{P}$. *Haar Top-K* retains the top 6 1D Haar coefficients (by absolute value) per vertex on a 25×45 grid. *1DG* uses 4 1D Gaussians per vertex on the same grid. *Haar MBD* compresses the first 64 1D Haar coefficients using 3 moving basis vectors learned on a coarse 5×9 grid, with per-vertex coefficients defined on the full 25×45 grid. *Our method* uses 1200 3D Gaussians and achieves higher quality results, particularly near narrow sources.

randomly samples a boundary point z_i and minimizes the error between the conditioned Gaussians and the reference solution. The reference Poisson kernel is precomputed on a 512^3 grid using the Walk on Spheres (WoS) method [Muller 1956], as detailed in the supplementary material. For efficient rendering and backpropagation, we use a differentiable tile-based rasterizer [Kerbl et al. 2023], mixing the Gaussians via additive accumulation instead of alpha blending. To handle discontinuities from the boundary geometry, we apply a mask over the interior domain, avoiding wasting too many unnecessary Gaussians on fitting sharp boundary curves.

Runtime. At runtime, we approximate the boundary condition $g(\mathbf{y})$ using a 1D Gaussian mixture. The output u can then be analytically computed as a 2D Gaussian mixture using our closed-form operator. To accelerate rendering, we discard the output Gaussian ξ_{ij} if the three-sigma confidence regions of the input Gaussian ϕ_j and the kernel Gaussian σ_i do not overlap. The remaining Gaussians are splatted and masked as in training. The entire pipeline is shown in Fig. 8.

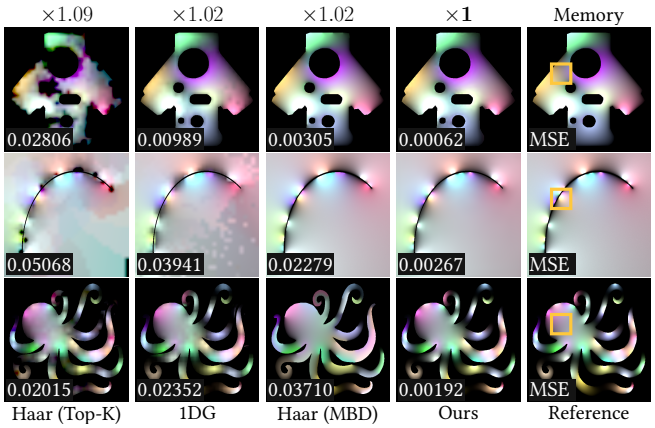


Fig. 10. More Involved Scenes. Our approach also handles internal occluders, open boundary curves, and detailed shapes while achieving higher-quality reconstructions. The first two rows use 1500 Gaussians, and the third uses 3000. Training times for these three problems are 5.25, 13.15, and 3.30 minutes with our method, compared to 8.30 minutes for MBD in all scenes.

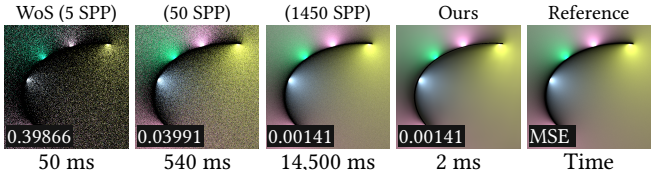


Fig. 11. Comparison with Walk on Spheres. As a precomputation-based method, our runtime solution is noise-free. In contrast, Walk on Spheres (WoS), being a Monte Carlo method, exhibits significant noise and requires considerably more time (×7250 here) to achieve comparable visual quality.

Comparison. In Fig. 9 and Fig. 10, we compare our method against three alternatives that expand the input function g using different basis functions: (1) nonlinear Haar wavelets [Ng et al. 2003], (2) 1D Gaussian basis functions, and (3) truncated Haar wavelets compressed via moving basis decomposition (MBD) [Silvennoinen and Sloan 2021]. Under an equal memory budget, our method delivers higher reconstruction quality, particularly in boundary regions with sharp or localized sources. Both our model and the MBD method are trained on an NVIDIA RTX 4090 GPU, while the other basis methods are computed within seconds on an Intel i7-13700.

When comparing with other precomputed methods, we use equal memory settings to emphasize both approximation accuracy and compactness. Unlike Monte Carlo integration, where accuracy can improve over time by drawing more samples, the accuracy of precomputed methods is fixed given the chosen approximation. Therefore, our comparisons prioritize reconstruction quality over runtime performance. That said, our method also achieves competitive performance and can easily outperform Walk on Spheres, thanks to the precomputation of the Poisson kernel, as shown in Fig. 11.

Ablation of culling. Figure 12 illustrates how our culling strategy can substantially reduce the number of Gaussians that need to be

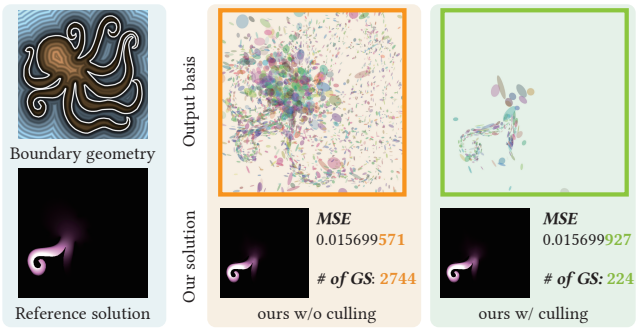


Fig. 12. Influence of culling. We place a single 1D Gaussian source on one tentacle of the Octopus. While it should technically interact with all 3D Gaussians in the kernel, most contribute negligibly. By culling kernel Gaussians outside the 3-sigma regions, only 8% of the output Gaussians are retained, with a minimal MSE increase of just 0.002%.

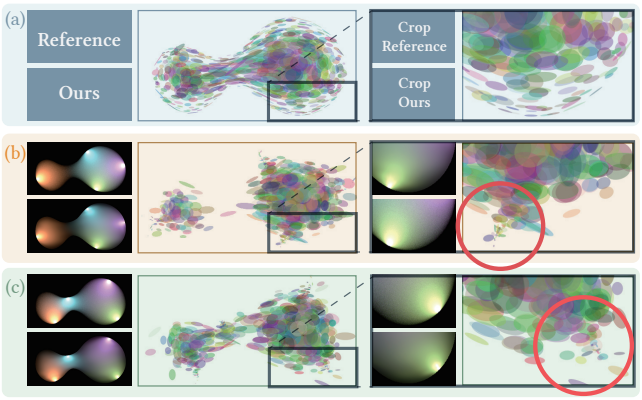


Fig. 13. Distribution of output basis. **(a)** shows the marginal Gaussians of our fitted kernel. Near the domain boundaries, the Gaussians exhibit anisotropic shapes aligned with the edges. However, this does not imply that high-frequency details near the boundary cannot be represented. **(b)** Given a specific input boundary condition, the resulting output Gaussians are visualized. In regions with strong localized sources, many small Gaussians are automatically allocated to capture high-frequency details, demonstrating the adaptive nature of our method. **(c)** For a different input, the output basis again adapts, concentrating smaller Gaussians in regions with fine detail. For clarity, we visualize an over-distilled model using only 600 Gaussians, which does not reflect the full reconstruction capacity of our approach.

rasterized, while maintaining nearly identical output quality. In the Octopus scene with 500 input 1D Gaussian sources, rendering without culling takes 11 ms, whereas enabling culling reduces the time to just 1 ms, achieving a 11× inference speedup.

Adaptive output basis. In Fig. 13, we illustrate how the output basis is dynamically influenced by the input. When the kernel is accurately approximated, the output basis exhibits adaptive behavior: allocating more, smaller Gaussians in regions with high-frequency content. This adaptivity explains the superior efficiency of our method compared to those relying on a fixed output basis.

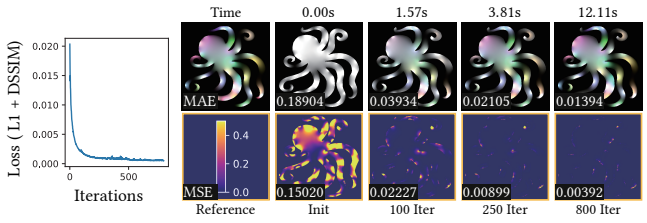


Fig. 14. Inverse boundary value problem. Our approach is fully differentiable and allows direct optimization of the boundary condition to match a given solution. Thanks to precomputation, it converges in seconds.

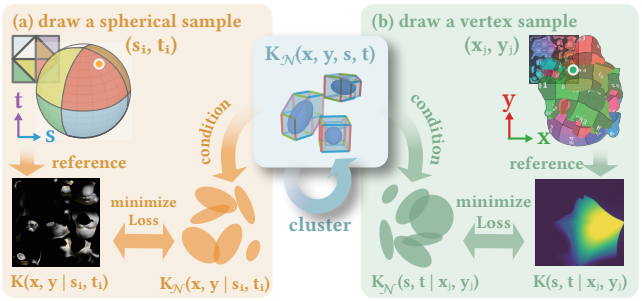


Fig. 15. Fitting pipeline of Prt. To fit the 4D Gaussian mixture, each iteration consists of two steps: **(a)** A directional sample (s_i, t_i) is drawn from the sphere. The corresponding reference slice $K(x, y \mid s_i, t_i)$ $K(s, t \mid x_j, y_j)$ represents the lightmap under a directional light from direction (s_i, t_i). We condition the 4D Gaussian mixture on (s_i, t_i) to obtain the predicted irradiance $K_N(x, y \mid s_i, t_i)$ and minimize the error against the reference. **(b)** Then, a vertex sample is drawn on the mesh, with texture coordinates (x_j, y_j). The corresponding reference slice, $K(s, t \mid x_j, y_j)$, is the transport function at that vertex, describing the radiance contribution from each direction. We condition the 4D Gaussian mixture on (x_j, y_j) to obtain the predicted $K_N(s, t \mid x_j, y_j)$ and minimize the error accordingly. To reduce redundancy, we periodically interleave optimization with Gaussian clustering.

Solving inverse problem. Our runtime pipeline, as shown in Fig. 8, is fully differentiable. Therefore, by fixing the kernel Gaussians and optimizing the boundary Gaussians, it can also be applied to inverse problems: optimizing the boundary values to match a given solution. As illustrated in Fig. 14, our method can efficiently reconstruct the boundary values within seconds.

5.2 Precomputed radiance transfer

Next, we apply our method to precomputed radiance transfer,

$$u(x, y) = \iint K(x, y, s, t)\, g(s, t)\, \mathrm{d}s\, \mathrm{d}t, \tag{14}$$

where the radiance transfer kernel K maps the environment map g to the surface irradiance u. As shown in Fig. 1 (left), we parameterize the environment map using octahedral mapping as $g(s, t)$ and the geometry surface using texture coordinates as $u(x, y)$. The radiance transfer kernel thus forms a 4D function $K(x, y, s, t)$, which we approximate using a 4D Gaussian mixture in our approach.

Fitting. As illustrated in Fig. 15, we fit the 4D Gaussian mixture also by extracting 2D slices from the 4D space and minimizing

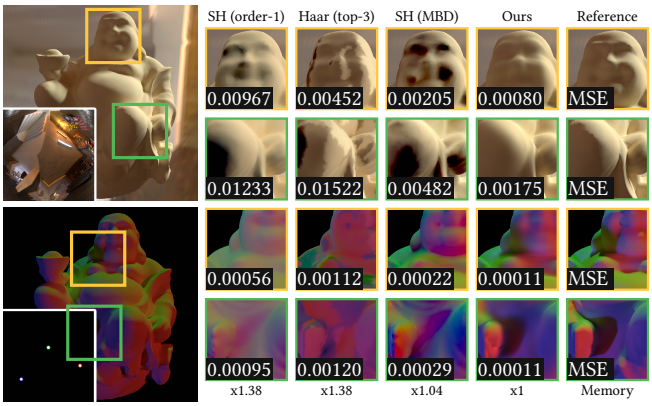


Fig. 16. Diffuse Global Illumination with Low Memory Budget. We approximate diffuse transport with multiple bounces in the Buddha scene and relight it under two different environment maps, using different kernel approximations with equal memory. Our method most faithfully preserves both shadow shapes and color fidelity compared to other approaches.

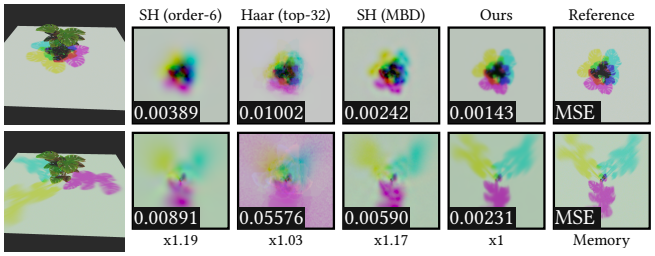


Fig. 17. Diffuse Global Illumination with High Memory Budget. We approximate diffuse transport with multiple bounces for the underlying plane using a high memory budget. Notably, our method is able to reconstruct sharp shadows cast by small light sources and even capture fine details such as the gaps between leaves.

Fig. 18. Glossy Global Illumination. We approximate glossy transport with multiple bounces under a fixed camera view. Under an equal memory budget, our method outperforms baseline approaches in reproducing specular highlights across different environment maps.

the corresponding image reconstruction error. Specifically, we use a combination of L1 and DSSIM losses. Unlike the setup in Section 5.1, where slicing is performed solely along the input space $\mathbf{z}$, the PRT case involves two types of slicing within each iteration to better capture the structure of the high-dimensional kernel. First, we sample an input direction (s_i, t_i) and minimize the error between the resulting lightmap $K(x, y \mid s_i, t_i)$ and the conditional Gaussians $K_{\mathcal{N}}(x, y \mid s_i, t_i)$. Next, we sample an mesh vertex (x_j, y_j) and minimize the error between its directional contribution $K(s, t \mid x_j, y_j)$ and the conditional kernel $K_{\mathcal{N}}(s, t \mid x_j, y_j)$. For a fair comparison with the baseline methods, we follow the typical PRT setup and only render the conditional transfer function $K(s, t \mid x_j, y_j)$ at each mesh vertex j. The other type of slice, $K(x, y \mid s_i, t_i)$, is then obtained by interpolation across vertices. This ensures that our method is trained with the same amount of information as the baselines. Please refer to the supplementary material for further details.

To reduce redundancy in the Gaussian mixture, we periodically interleave optimization with Gaussian clustering [Goldberger and Roweis 2004] to merge similar components. Each surface patch in the lightmap is trained independently, using the same masking strategy described in Section 5.1, to prevent interference between Gaussians from different patches.

Our approach also requires fitting the environment map with 2D Gaussian mixtures during precomputation. We also use the differentiable rasterizer and clustering for this step; further details are provided in the supplementary material.

Runtime. Our runtime pipeline is illustrated in Fig. 1(left). We approximate the input environment map using a 2D Gaussian mixture and compute the output irradiance Gaussians for each surface patch. The output Gaussians are then splatted and masked onto a lightmap, which defaults to a resolution of 2048^2 in our comparisons. The mesh is subsequently rendered using this lightmap.

Equal memory comparison. We compare our method with alternative approaches under an equal memory budget to approximate the transport operator. In Fig. 16, we use 10,618 4D Gaussians to approximate the diffuse multi-bounce light transport of a Buddha mesh with 20,769 vertices. Each 4D Gaussian stores a mean (4 floats), covariance (10 floats), and RGB weight (3 floats), following the parameterization of Diolatzis et al. [2024]. This results in an average of 8.7 floats per vertex. In comparison, even first-order spherical harmonics (SH) require 12 floats, showing the compactness of our representation. Despite the lower memory budget, our method achieves higher relighting accuracy, better preserving shading details.

In Fig. 17, we approximate the light transport of a rectangular plane using 72,649 4D Gaussians— a significantly larger budget than in Fig. 16 — to demonstrate our method's capability to handle large surface patches and capture fine details. The baseline uses a 100×100 uniform grid, while our method achieves comparable resolution with an effective storage of 123.5 floats per vertex, on par with order-6 spherical harmonics or retaining the top 32 nonlinear Haar coefficients. Our method also exhibits superior relighting accuracy in this high-budget, high-detail scenarios.

In Fig. 18, we approximate glossy transport assuming a fixed camera view for simplicity. Our approach again outperforms others in reconstructing detailed highlights while using very compact storage. To demonstrate that our efficiency does not solely come from the

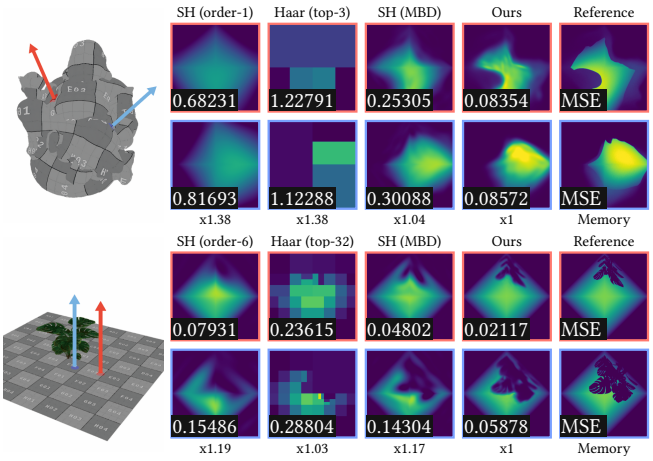


Fig. 19. Vertex Transfer Function. We visualize the reconstructed transfer functions at selected vertices using different methods. Our approach consistently provides significantly more accurate approximations of the sliced transfer functions, obtained by conditioning the 4D Gaussian mixture on vertex texture coordinates, across various scenes and memory budgets.

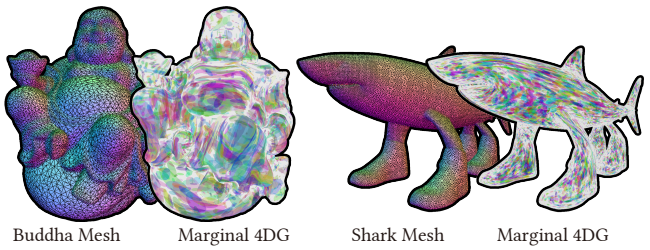


Fig. 20. Distribution of Marginalized 4D Gaussians. We visualize our learned 4D Gaussian kernels by marginalizing them onto the texture coordinate domain (x, y). Ideally, regions exhibiting high spatial-directional coherence are assigned fewer and larger Gaussians, while areas with greater complexity or variation receive more and smaller ones. The anisotropy of each Gaussian indicates the local structure of the transfer function, revealing how regions with similar directional responses are grouped. Our training algorithm effectively places 4D Gaussians to form a compact and expressive representation. Notably, denser clusters of smaller Gaussians appear around geometric corners, while elongated anisotropic Gaussians align with highlight regions, capturing their directional similarity.

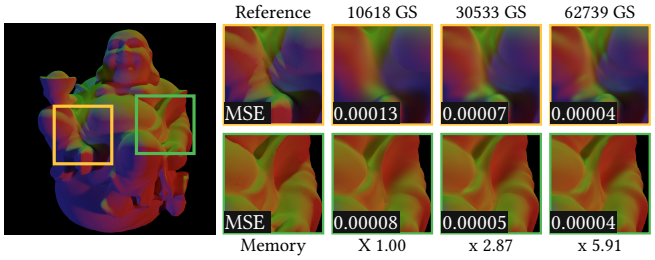


Fig. 21. Number of Gaussians vs. Transfer Detail. Increasing the number of 4D Gaussians used in the kernel approximation allows more details to be preserved. For example, in the model with fewer Gaussians, some clothing wrinkles appear smoothed out, whereas higher-Gaussian-count models capture these fine details more accurately.

octahedral parameterization or the use of Gaussian bases, we also compare against an alternative that approximates transport using per-vertex 2D Gaussians, with the same octahedral mapping for the environment map. Under the same memory budget, only one 2D Gaussian can be allocated per vertex. Despite we have used adaptive initialization for 2DG, a single lobe remains highly sensitive to initialization and prone to mode collapse in many cases, leading to poor output quality.

For all comparisons, we provide the best possible approximation of the environment map, as our main goal is to validate the accuracy of the operator. For example, in the Buddha scene, MBD compresses the kernel using up to 17th-order spherical harmonics. Thus, we use the same order to approximate the input environment map, using 972 floats in total. For our method, depending on the complexity, we approximate the environment map using 100 (i.e. 800 floats), 40, 20, or 3 two-dimensional Gaussians.

Kernel approximation accuracy. Better relighting results directly stem from more accurate kernel approximations. In Fig. 19, we visualize how each method approximates a slice of the transport kernel at the same mesh vertex. Our approach achieves significantly better approximation under equal memory constraints, primarily because all 4D Gaussians are shared across vertices within the same patch. As a result, each vertex effectively fits the conditional transport using a much larger number of basis functions.

Distributions of 4D Gaussian kernel. Efficiently reusing Gaussians across vertices is crucial for the compactness of our approach. Ideally, regions with more complex spatial-directional signals should be represented by a larger number of 4D Gaussians, while areas with more coherent signals can share the same set of Gaussians. In Fig. 20, we visualize the spatial distribution of 4D Gaussians marginalized onto the lightmap space (x, y). This demonstrates how our training strategy effectively captures correlations across the spatial and directional domains in a compact manner.

Number of Gaussians. Existing basis representations typically can improve approximation accuracy by increasing the number of coefficients, allowing higher-order basis functions to be retained. In Fig. 21, we similarly demonstrate that using a larger number of Gaussians can also lead to more accurate fitting. Since our approach is mesh-independent, it offers great flexibility in balancing storage requirements and relighting accuracy — even under one float per vertex if necessary.

Training cost. In Table 1, we report the training times of different methods on the Shark scene. Our approach requires comparatively more training time, as it employs a brute-force strategy to derive the final set of Gaussians. Specifically, we begin by initializing five 4D Gaussians per mesh vertex and optimize them in a phase referred to as the *initial train*. Following this, we iteratively distill the model by clustering the Gaussians to 75% of their current count and fine-tuning the clustered representation. This distillation process

Table 1. Training time. We report the training times for all methods on the glossy Shark scene. SH coefficients and Moving Basis Decomposition (MBD) training are accelerated using GPU shaders, while the nonlinear Haar basis computation and selection run on CPU only. Both the per-vertex 2D Gaussian method and our approach use the differentiable rasterizer for training, with Gaussian clustering performed on CPU in our method. All models are trained on a cluster with hybrid Nvidia A40 and A100 GPUs, and the reported times represent the total GPU-hours of all parallel jobs.

Methods	SH + MBD	Haar	2DG	Ours
Total training time (hours)	3.84	21.86	43.48	26.08
		Distillation (8 iters)		
Ours Breakdown	Initial train	Clustering	Retraining	
Training hour	6.40	3.13	17.18	

Fig. 22. Runtime Culling. We marginalize the learned 4D Gaussians onto the environment map domain (s, t). At runtime, given a 2D Gaussian mixture approximation of the environment map, we evaluate every pair of input 2D Gaussians and marginalized 2D Gaussians. Pairs are culled if their 3-sigma confidence regions do not overlap.

Table 2. Impact of Culling. We report the number of output Gaussians before and after applying our culling pass, along with the corresponding increase in MSE. A significant portion of Gaussians can be discarded with minimal impact on accuracy, as evidenced by only a slight increase in MSE.

Scene	**# input GS→ # output GS**	**+ culling**	**# output GS**	**MSE**
Buddha	3 → 31,854	15,660	-50.84%	+0.0034%
	20 → 212,360	169,415	-20.22%	+0.0872%
	40 → 424,720	262,768	-38.13%	+0.3203%
Shark	3 → 108,618	24,828	-77.14%	+0.0015%
	20 → 724,120	372,523	-48.55%	+0.0073%
	40 → 1,448,240	537,795	-62.86%	+0.0124%

is repeated 8–9 times until the number of Gaussians is reduced to approximately 0.5 per vertex on average. On the positive side, our method allows for joint optimization over all positions on a texture patch at each step, which is significantly more efficient than per-vertex serialized optimization, as used in *2DG*.

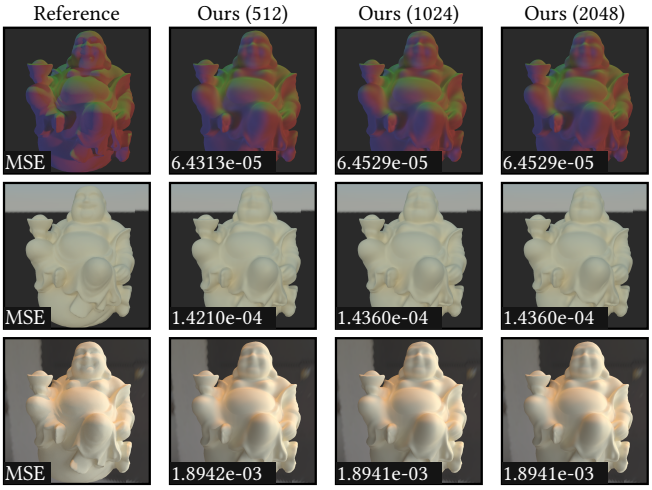
Ablation of culling. To reduce the number of output 2D Gaussians, we use the same culling strategy that avoids generating from all possible pairs between kernel Gaussians and input Gaussians. As shown in Fig. 22, we first marginalize the kernel Gaussians into the environment map domain (s, t), turning them into 2D Gaussians. We then compare their confidence regions with those of the input

Table 3. Optimize Runtime Performance. We report the runtime cost of rendering a 2048^2 lightmap using varying numbers of 2D Gaussians to approximate the environment map. While the Naïve implementation is computationally expensive, our culling and merging performance optimizations significantly reduce inference time.

Scene	**# input 2DG**	**basic**	**+ culling**	**+ merging**
Buddha	3	37.59 ms	30.02 ms	5.51 ms
	20	184.42 ms	118.75 ms	73.23 ms
	40	306.65 ms	148.49 ms	93.82 ms
Shark	3	25.80 ms	19.85 ms	5.05 ms
	20	154.15 ms	83.96 ms	60.76 ms
	40	264.56 ms	122.84 ms	92.96 ms

Table 4. Impact of Lightmap Resolution. We demonstrate how varying lightmap resolutions affect rendering cost and relighting quality. In the Buddha test scene, lower lightmap resolutions significantly reduce inference time while preserving, or sometimes even improving, the MSE.

	Lightmap resolution		
Input 2DG	512^2	1024^2	2048^2
3	1.18 ms	2.04 ms	5.51 ms
20	6.75 ms	19.50 ms	73.23 ms
40	10.37 ms	26.63 ms	93.82 ms



2D Gaussians that approximate the environment map. If the 3-sigma confidence regions of a pair do not overlap, we discard the output of that pair during rasterization. As shown in Table 2, this approach allows us to cull a large number of output Gaussians in practice, resulting in only a minimal increase in MSE. At the same time, it significantly improves rendering speed, as demonstrated in the *+ culling* column of Table 3.

Inference cost and performance optimization. In addition to culling, we further improve inference speed by merging the Gaussians from all patches into a single splatting pass. To preserve the masking

Table 5. Runtime Performance. We measure the average frametime (ms) in our actual renderer running on an RTX 3070 Laptop GPU. For evaluation, we use the simplest environment map containing only 3 input Gaussians, and render lightmaps at a resolution of 512^2 for our method.

Scene	SH	MBD	Ours
Buddha	1.82 (order-1)	7.04 (4 basis, 324 coeffs each)	5.68
Shark	2.39 (order-1)	14.7 (8 basis, 121 coeffs each)	6.25

effect, each Gaussian is assigned an index indicating its corresponding patch. During rasterization, we discard pixels that fall outside the assigned patch region. This approach eliminates the need to perform N times splatting for N patches, leading to a significant efficiency gain, as shown in the *+ merging* column of Table 3.

Another important factor affecting rendering performance is the resolution of the lightmap we splat onto. While the previous comparisons use a 2048^2 resolution lightmap, Table 4 shows that using lower-resolution lightmaps can significantly boost rendering speed, with only minimal additional error, making it an effective trade-off between quality and performance.

We implement the runtime rendering for both our method and all baseline methods in a custom Vulkan renderer. The runtime performance is reported in Table 5, and additional timings are provided in the supplementary material.. Although our method runs at a lower framerate, it delivers much better visual quality. We also believe there is still significant room for further optimization of our method. For example, by constructing a hierarchy of Gaussian operators or applying visibility culling to discard Gaussians that are not visible in the current viewport.

6 APPLICATION: MULTIPLE OPERATORS

We now demonstrate how our approach can be applied to scenarios involving multiple operators.

6.1 Spectral rendering and fluorescence

In this application, we consider spectral rendering with fluorescent materials, which involves a re-radiation process,

$$u_o(\lambda_o) = \int K(\lambda_o, \lambda_i)\, \mu_i(\lambda_i)\, \mathrm{d}\lambda_i, \tag{15}$$

where the re-radiation kernel K maps the incident spectrum μ_i (before interacting with the material) to the outgoing spectrum u_o (after the interaction). Although this involves only a 2D kernel, multiple operators may be applied sequentially, since a light path can include several light–material interactions.

Background. Spectral rendering represents the intensity of light as a continuous function over wavelength $u(\lambda)$. As illustrated in Fig. 23, each light path begins with a spectral source function $g(\lambda)$ and is successively transformed by a sequence of linear operators $\mathcal{K}_l$, $l = N, \ldots, 1$ representing material interactions along the path. At the camera, the resulting spectrum is integrated against the CIE color matching functions $\bar{x}(\lambda)$, $\bar{y}(\lambda)$, $\bar{z}(\lambda)$ to compute XYZ values:

$$X := \int \bar{x}(\lambda)u(\lambda)\,\mathrm{d}\lambda;\; Y := \int \bar{y}(\lambda)u(\lambda)\,\mathrm{d}\lambda;\; Z := \int \bar{z}(\lambda)u(\lambda)\,\mathrm{d}\lambda, \tag{16}$$

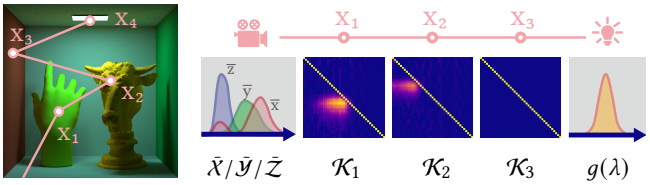


Fig. 23. Spectral integral. In spectral rendering, the contribution of each light path is computed as a recursive integral over wavelength. A photon with spectral luminance $g(\lambda)$ is successively transformed by a sequence of linear operators $\mathcal{K}$, each corresponding to a material interaction at a path vertex. At the camera, the resulting spectrum is integrated against the CIE color matching functions $\bar{x}(\lambda)$, $\bar{y}(\lambda)$, and $\bar{z}(\lambda)$ to compute final tristimulus values. Fluorescent materials exhibit nonzero off-diagonal elements in their operator kernels $K(\lambda_o, \lambda_i)$, as seen in K_1 and K_2, enabling wavelength shifts. In contrast, typical (non-fluorescent) materials, such as K_3, have kernels with only diagonal components, preserving the input wavelength.

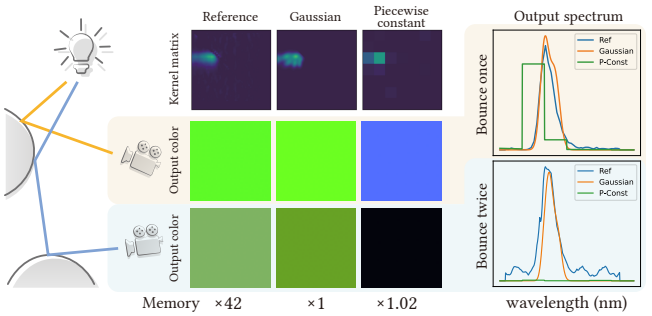


Fig. 24. *Compressed re-radiation matrix.* Gaussian mixtures have proven to be effective for approximating the re-radiation matrix, offering more accurate output spectral and consequently more faithful output colors. In contrast, under equal memory budget, a piecewise-constant representation will introduce significant bias in the resulting color. With multiple bounces, the error will accumulate and lead to more significant artifacts.

which are then converted to RGB via a linear transformation. In summary, the contribution of each path can be expressed as:

$$X/Y/Z := \bar{\mathcal{X}}/\bar{\mathcal{Y}}/\bar{\mathcal{Z}}\,[\mathcal{K}_1[\mathcal{K}_2 \cdots [\mathcal{K}_N[g]]]], \tag{17}$$

where $\bar{\mathcal{X}}/\bar{\mathcal{Y}}/\bar{\mathcal{Z}}$ denote operators of integrating against the respective color matching functions $\bar{x}(\lambda)$, $\bar{y}(\lambda)$, and $\bar{z}(\lambda)$.

The influence of materials on the light spectrum can be characterized by a kernel function $K(\lambda_o, \lambda_i)$. The resulting spectrum u_o can then be computed by applying the kernel K to the input spectrum u_i, i.e. $u_o(\lambda_o) = \int K(\lambda_o, \lambda_i)u_i(\lambda_i)\,\mathrm{d}\lambda_i$, where λ_i and λ_o denote the input and output wavelengths, respectively. Specifically, a typical BSDF assumes that the wavelength remains unchanged during interaction, so the kernel is non-zero only when $\lambda_o = \lambda_i$:

$$K(\lambda_o, \lambda_i) = \begin{cases} f(\lambda_i) \cdot \delta(\lambda_o = \lambda_i), & \text{if } \lambda_i = \lambda_o, \\ 0, & \text{if } \lambda_i \neq \lambda_o, \end{cases} \tag{18}$$

where δ is the Dirac delta function. In contrast, fluorescent materials shift the wavelength upon interaction, requiring an extension of the BRDF to *Bi-spectral Bidirectional Reflectance and Re-radiation Distribution Function* (BBRRDF) [Hullin et al. 2010], which includes

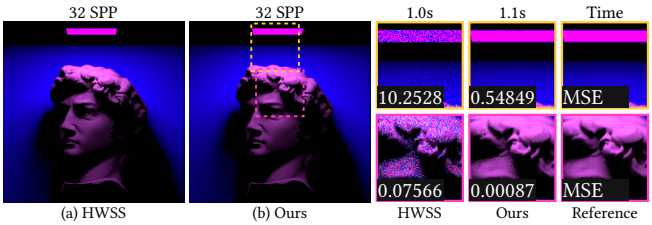


Fig. 25. Direct illumination. We render David with a fluorescent material, illuminated by light with spectral energy centered around 450 nm, with direct illumination only. With 32 samples per pixel, our method significantly reduces color noise with minimal overhead, even in non-fluorescent regions, where HWSS struggles to sample wavelengths near 450 nm.

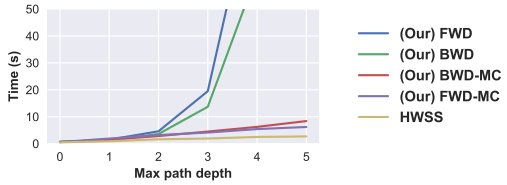


Fig. 26. Increasing time with path depth. Rendering times for 32 samples per pixel with varying maximum path depths. Both our forward (FWD) and backward (BWD) methods experience exponential growth in computation time, making them impractical at higher depths. However, when combined with a Monte Carlo estimator, our method maintains low rendering times.

off-diagonal components. The integral in Eq. (17) is typically estimated using Monte Carlo methods such as *Hero Wavelength Spectral Sampling* (HWSS) [Mojzík et al. 2018; Wilkie et al. 2014].

Compact representation. Compact spectral representations are essential in spectral rendering [Jakob and Hanika 2019; Peters et al. 2019], as dense spectra — often requiring hundreds of floats per spectrum — can lead to prohibitive storage demands and poor caching performance, especially when used with textures. Fluorescent materials exacerbate this issue by introducing a two-dimensional re-radiation matrix, which multiplies the memory burden. Hua et al. [2023] addressed this by compressing the signal using 2D Gaussian mixtures, demonstrating that it provides an accurate approximation with bounded bias. As shown in Fig. 24, under equal memory constraints, the Gaussian mixture representation can better preserve the output color, whereas a piecewise-constant alternative fails.

Closed-form spectral integral. Despite Hua et al. [2023] compressing the re-radiation matrix with 2D Gaussians, the spectral integral in Eq. (17) is still estimated using hero-wavelength spectral sampling. This approach can lead to noticeable color noise in challenging scenarios, particularly with fluorescent materials, where the stratification of wavelength samples often breaks down, as shown in Fig. 25(a). To address this, we approximate all components — emission spectra g, material kernels K, and color matching functions $\bar{x}, \bar{y}, \bar{z}$ — using 1D or 2D Gaussian mixtures. This enables a closed-form evaluation of the spectral contribution in Eq. (17).

The classical functional mapping method (Eq. (5)) can also handle scenarios involving multiple operators through straightforward matrix multiplication. However, when using a fixed output basis, its expressiveness is limited, just like the piecewise constant case shown in Fig. 24, especially under equal-memory constraints. In contrast, our approach enables analytical solutions using a Gaussian-based matrix approximation, which has been shown to be more accurate.

Belcour et al. [2025] also address fluorescence using closed-form Gaussian representations, approximating re-radiation matrices with axis-aligned 2D Gaussians and enforces an upper-triangular structure by explicitly modeling the Heaviside function. In contrast, our approach supports non-axis-aligned cases and accounts for multiple bounces, but does not explicitly incorporate the Heaviside function.

Diagonal and off-diagonal operator representation. The diagonal components[1] of the material kernel K pose a challenge for our 2D Gaussian representation, as they correspond to measure-zero sets in 2D and cannot be directly modeled with 2D Gaussians. To address this, we represent the diagonal part separately using a 1D Gaussian mixture $\tilde{f}_{\text{diag}}$, yielding a hybrid approximation:

$$\tilde{K}(\lambda_i, \lambda_o) = \underbrace{\tilde{K}_{\text{off}}(\lambda_o, \lambda_i)}_{\text{2D Gaussians}} + \underbrace{\tilde{f}_{\text{diag}}(\lambda_i)}_{\text{1D Gaussians}} \cdot \delta(\lambda_o = \lambda_i), \tag{19}$$

where the off-diagonal term $\tilde{K}_{\text{off}}$ is omitted for non-fluorescent materials. We show that this hybrid operator representation retains closed-form integration and analytical operator composition, as shown in the supplementary material. As illustrated in Fig. 25, our method significantly reduces color noise under direct illumination.

Operator application strategies. For multi-bounce paths, we can analytically solve Eq. (17) in two directions, as discussed in Fig. 6. The first approach starts from the input g, applies the last operator to obtain $u_3 = \mathcal{K}_3[g]$, and then recursively applies the preceding operators, such as $\mathcal{K}_2[u_3]$, and so on. Each intermediate result remains a Gaussian mixture, enabling exact and efficient propagation without basis projection. However, this strategy requires storing the material information for all vertices along the path during runtime. Alternatively, we can compute the *throughput* of the current path by composing the operators corresponding to each material interaction. For example, we first composite two operators into $\mathcal{K}_{12} = \mathcal{K}_1 \circ \mathcal{K}2$ such that $\mathcal{K}_{12}[g] = \mathcal{K}_1[\mathcal{K}_2[g]]$, and then further extend this to $\mathcal{K}_{123} = \mathcal{K}_{12} \circ \mathcal{K}_3$, and finally compute $\mathcal{K}_{123}[g]$. We refer to the first strategy as the *backward*, and the second as the *forward* method.

Stochastic summation. Another challenge in multi-bounce paths is the exponential growth in the number of basis functions with the number of operators. As shown in Eq. (10), the number of output basis functions equals the product of the input and kernel basis counts. This leads to exponential increases in computation time, as illustrated in Fig. 26. To mitigate this, we apply a stochastic method to sample from the output Gaussians after each operator:

$$u(\mathbf{x}) = \sum_{i=1}^{I} \sum_{j=1}^{J} \mathrm{U}_{ij} \cdot \xi_{ij}(\mathbf{x}) = \mathbb{E}\left[\frac{1}{N} \sum_{k=1}^{N} \frac{U_k \cdot \xi_k(\mathbf{x})}{p(k)} \right], \tag{20}$$

[1]Technically, we treat the re-radiation matrix as a continuous 2D function rather than a discretized matrix; however, we still borrow the terms diagonal and off-diagonal to describe the Dirac delta components, as defined in Eq. (18).

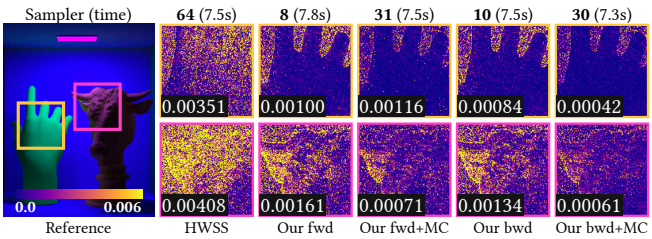


Fig. 27. Global illumination. We render a scene with two different fluorescent materials under an equal time budget of approximately 7.5 seconds, using a maximum path depth of 4. Hero-wavelength spectral sampling (HWSS) achieves the highest sample count (64 SPP) but suffers from significant color noise. In contrast, our forward (FWD) and backward (BWD) methods significantly reduce color noise, though with fewer samples per pixel. By combining these methods with stochastic summation, our approach achieves a favorable balance between spectral accuracy and rendering performance.

where we importance sample N Gaussians from the full set of $I \times J$ using the distribution p. Although this reintroduces some noise, each sample represents a Gaussian rather than a point, resulting in improved convergence [Billen and Dutré 2016; Deng et al. 2019; Salesin and Jarosz 2019].

Implementation. We implemented our approach in PBRT-v4 [Pharr et al. 2023], using fluorescence hero-wavelength spectral sampling [Mojzík et al. 2018] with Gaussian mixture compression [Hua et al. 2023] as the baseline (HWSS). For both HWSS and our stochastic Gaussian sampling, we used $N = 4$ point- or Gaussian-spectrum samples for each light path. Our sampling process is implemented using weighted reservoir sampling (WRS) to minimize additional computational overhead. All results in this section were tested on an Intel i7-13700 CPU.

Comparison. For direct illumination, as shown in Fig. 25, our method significantly reduces color noise with minimal computational overhead. For global illumination, as shown in Fig. 27, both forward (FWD) and backward (BWD) methods introduce substantial overhead, leading to fewer samples within the same time budget. However, they can still be beneficial in scenes dominated by color noise from spectral integration. By combining our approach with stochastic summation (FWD/BWD + MC), we mitigate the exponential growth in computation and achieve a greater overall variance reduction.

7 DISCUSSION AND FUTURE WORK

Comparing generalizability to neural methods. While neural networks are powerful, the inherent analytical properties of our method, and the baselines to which we compare, offer advantages in training and generalization. For instance, these methods preserve linearity, allowing training on simple basis inputs like $[1, 0, 0, \dots]$, $[0, 1, 0, \dots]$, etc., and generalization to arbitrary inputs like $[3, 1, 4, \dots]$ via linearity. In contrast, standard brute-force neural networks lack this property and require extensive training on a wide variety of input vectors, demanding significantly more data and effort.

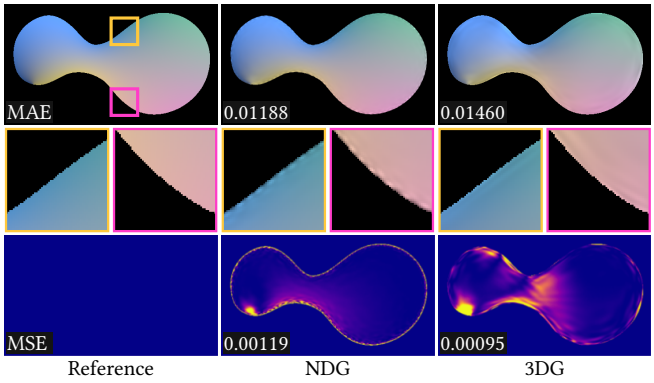


Fig. 28. Influence of Gaussian parameterization. The parameterization of Gaussians can influence the results, even when using the same initialization, training hyperparameters, loss functions, and number of iterations. For example, NDG [Diolatzis et al. 2024] uses the upper triangular components of the Cholesky decomposition to parameterize the covariance matrix, while 3DG [Kerbl et al. 2023] adopts a formulation based on separate rotation and scaling matrices. At runtime, the NDG parameterization tends to produce zigzag-like aliasing but achieves lower absolute error, whereas the 3DG parameterization exhibits ringing artifacts yet yields better performance in terms of squared error.

Similarly, in the multi-operator setting, our method supports analytical composition, so each kernel can be learned independently, and arbitrary compositions can be performed at runtime. In contrast, neural networks must learn the composition themselves, requiring supervision on all potential kernel combinations, which makes the training process significantly more complex.

Output- and perception-aware operator learning. Typical PRT methods follow a largely deterministic pipeline—for example, when using spherical harmonics or Haar wavelets, the coefficients are obtained via explicit basis projection. In contrast, our method's behavior depends heavily on how the kernel is approximated. As shown in Fig. 28, different parameterizations can lead to markedly different optimized kernels. We use the NDG parameterization for most results, as it typically gives lower MSE and supports any-dimensional Gaussians. For Fig. 13, however, we use 3DG for better visualization.

Although we employ an L1 + DSSIM loss for all our precomputation, the loss function that yields the most perceptually pleasing results remains an open question. A promising avenue for future work is to optimize the kernel not by minimizing its sliced error, but by directly maximizing the fidelity of the final rendered output.

Fitting high-dimensional Gaussians. While our mathematical formulation supports arbitrary input dimension $N_\mathbf{y}$ and output dimension $N_\mathbf{x}$, fitting kernels in high-dimensional spaces remains a practical challenge. In current applications, we minimize errors between 2D slices of high-dimensional data via conditional Gaussians, leveraging tile-based rasterizers for efficient differentiation. However, when reference slices are unavailable — such as in fitting kernels defined on 3D point clouds — this approach breaks down. Being able to fit all kinds of high-dimensional kernels would significantly broaden the applicability of our approach.

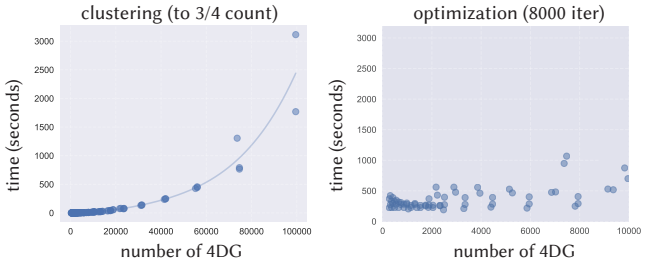


Fig. 29. Efficiency of Clustering and Optimization. We evaluate the training time cost with respect to the number of 4D Gaussians used in the Shark scene. The task of clustering Gaussians down to 75% of their current number exhibits a significant increase in computational cost as the number of Gaussians grows. In contrast, optimizing Gaussian parameters using a differentiable rasterizer scales more favorably with the number of Gaussians, demonstrating better computational efficiency at larger scales.

Clustering and densification. As discussed in Table 1, a substantial portion of the training cost for our PRT model arises from the clustering and retraining of Gaussians. Currently, we use a greedy clustering algorithm with $O(N^2 \log N)$ time complexity and $O(N^2)$ space complexity, where N is the number of Gaussians. This approach becomes increasingly expensive as the number of Gaussians grows, as shown in Fig. 29. One potential improvement is to adopt a more scalable, parallel clustering strategy to reduce computational overhead. Moreover, adopting the densification technique proposed by Kerbl et al. [2023] may enable training from fewer initial Gaussians, thus alleviating or even eliminating the need for expensive distillation. Balancing clustering and densification in high-dimensional Gaussian fitting presents an interesting direction for future work.

Runtime performance. While our method achieves higher quality under equal memory, its runtime performance is generally slower than the baselines. Fortunately, there is significant room to improve it. For example, better culling strategies, like visibility and occlusion culling, can skip Gaussians not visible in PRT applications. Hierarchical Gaussian operators can also help, by using coarser levels for large input Gaussians and finer levels for smaller ones, reducing the number of output Gaussians when the input is large.

Parameterization of domains. Our method performs best when the kernel function K can be well represented by an N_k-dimensional Gaussian mixture. In practice, the structure of K strongly depends on the choice of input ($\mathcal{Y}$) and output ($\mathcal{X}$) parameterizations. In our examples, we simply use the default parameterizations, but exploring parameterizations more aligned with Gaussian assumptions could improve compactness and accuracy. Another relevant concern is that certain parameterizations can introduce extra discontinuities into the domain — for example, the octahedral mapping of spherical space. Investigating extensions of our analytical operator to support other basis families, e.g., spherical Gaussians, better suited to these domains is also an interesting direction.

8 CONCLUSION

We present a novel method for closed-form integral linear operators by expanding kernels directly into high-dimensional Gaussian mixtures. Our approach compactly captures correlations across input-output spaces and adaptively expands the output basis according to the input structure. This formulation enables efficient, analytical operator integration and composition, and applies broadly across graphics problems involving single or multiple operators. We believe this formulation opens new avenues for scalable, accurate integral linear operator approximations in graphics and related fields.

ACKNOWLEDGMENTS

This research was partially funded by NSERC Discovery Grants (RGPIN-2020-03918), NSF Grants 2238839 and 2341952, and gifts from Adobe and Google. We thank the anonymous reviewers for their constructive feedback, Qingqin Hua and Chao Wang for helpful discussion, and Xingchang Huang for proofreading. Scene assets are courtesy of Sketchfab users Giora, Alex CGW, andrea.notarstefano, and Kirk Hiatt.

REFERENCES

Anima Anandkumar, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Nikola Kovachki, Zongyi Li, Burigede Liu, and Andrew Stuart. 2019. Neural Operator: Graph Kernel Network for Partial Differential Equations. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations.* https://openreview.net/forum?id=fg2ZFmXFO3

Kamyar Azizzadenesheli, Nikola Kovachki, Zongyi Li, Miguel Liu-Schiaffini, Jean Kossaifi, and Anima Anandkumar. 2024. Neural operators for accelerating scientific simulations and design. (April 2024).

Laurent Belcour, Alban Fichet, and Pascal Barla. 2025. A Fluorescent Material Model for Non-Spectral Editing & Rendering. In *Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers (SIGGRAPH Conference Papers '25).* Association for Computing Machinery, New York, NY, USA, Article 135, 9 pages. https://doi.org/10.1145/3721238.3730721

Niels Billen and Philip Dutré. 2016. Line Sampling for Direct Illumination. *Computer Graphics Forum* (2016). https://doi.org/10.1111/cgf.12948

Xi Deng, Shaojie Jiao, Benedikt Bitterli, and Wojciech Jarosz. 2019. Photon surfaces for robust, unbiased volumetric density estimation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 38, 4 (July 2019). https://doi.org/10/gf6rx9

Stavros Diolatzis, Tobias Zirr, Alexander Kuznetsov, Georgios Kopanas, and Anton Kaplanyan. 2024. N-Dimensional Gaussians for Fitting of High Dimensional Functions. In *ACM SIGGRAPH 2024 Conference Papers* (Denver, CO, USA) *(SIGGRAPH '24).* Association for Computing Machinery, New York, NY, USA, Article 126, 11 pages. https://doi.org/10.1145/3641519.3657502

Ana Dodik, Marios Papas, Cengiz Öztireli, and Thomas Müller. 2022. Path Guiding Using Spatio-Directional Mixture Models. *Computer Graphics Forum* 41, 1 (2022), 172–189. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14428 https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14428

Zhongpai Gao, Benjamin Planche, Meng Zheng, Anwesa Choudhuri, Terrence Chen, and Ziyan Wu. 2025. 6DGS: Enhanced Direction-Aware Gaussian Splatting for Volumetric Rendering. In *The Thirteenth International Conference on Learning Representations.* https://openreview.net/forum?id=sUvBTEYXGt

Jacob Goldberger and Sam Roweis. 2004. Hierarchical Clustering of a Mixture Model. In *Advances in Neural Information Processing Systems*, L. Saul, Y. Weiss, and L. Bottou (Eds.), Vol. 17. MIT Press. https://proceedings.neurips.cc/paper_files/paper/2004/file/36e729ec173b94133d8fa552e4029f8b-Paper.pdf

Oshri Halimi, Or Litany, Emanuele Rodola, Alex M Bronstein, and Ron Kimmel. 2019. Unsupervised learning of dense shape correspondence. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 4370–4379.

Sebastian Herholz, Oskar Elek, Jens Schindel, Jaroslav Křivánek, and Hendrik P. A. Lensch. 2018. A unified manifold framework for efficient BRDF sampling based on parametric mixture models. In *Proceedings of the Eurographics Symposium on Rendering: Experimental Ideas & Implementations* (Karlsruhe, Germany) *(SR '18).* Eurographics Association, Goslar, DEU, 41–52. https://doi.org/10.2312/sre.20181171

Sebastian Herholz, Oskar Elek, Jiří Vorba, Hendrik Lensch, and Jaroslav Křivánek. 2016. Product Importance Sampling for Light Transport Path Guiding. *Computer Graphics Forum* 35, 4 (2016), 67–77. https://doi.org/10.1111/cgf.12950 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12950

Ling Hu, Qinsong Li, Shengjun Liu, and Xinru Liu. 2021. Efficient deformable shape correspondence via multiscale spectral manifold wavelets preservation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 14536–14545.

Q. Hua, V. Tázlar, A. Fichet, and A. Wilkie. 2023. Efficient Storage and Importance Sampling for Fluorescent Reflectance. *Computer Graphics Forum* 42, 1 (2023), 47–59. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14716 https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14716

Matthias B. Hullin, Johannes Hanika, Boris Ajdin, Hans-Peter Seidel, Jan Kautz, and Hendrik P. A. Lensch. 2010. Acquisition and analysis of bispectral bidirectional reflectance and reradiation distribution functions. *ACM Trans. Graph.* 29, 4, Article 97 (July 2010), 7 pages. https://doi.org/10.1145/1778765.1778834

Martin Huska, Serena Morigi, and Giuseppe Recupero. 2023. Geometric texture transfer via local geometric descriptors. *Appl. Math. Comput.* 451 (2023), 128031.

Wenzel Jakob and Johannes Hanika. 2019. A Low-Dimensional Function Space for Efficient Spectral Upsampling. *Computer Graphics Forum (Proceedings of Eurographics)* 38, 2 (March 2019).

Wenzel Jakob, Christian Regg, and Wojciech Jarosz. 2011. Progressive expectation-maximization for hierarchical volumetric photon mapping. In *Proceedings of the Twenty-Second Eurographics Conference on Rendering* (Prague, Czech Republic) *(EGSR '11)*. Eurographics Association, Goslar, DEU, 1287–1297. https://doi.org/10.1111/j.1467-8659.2011.01988.x

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics* 42, 4 (July 2023). https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/

Nikola B. Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew M. Stuart, and Anima Anandkumar. 2021. Neural Operator: Learning Maps Between Function Spaces. *CoRR* abs/2108.08481 (2021).

Steven George Krantz, Steve Kress, and R Kress. 1999. *Handbook of complex variables*. Springer.

Jaroslav Křivánek, Sumanta Pattanaik, and Jiří Žára. 2004. Adaptive mesh subdivision for precomputed radiance transfer. In *Proceedings of the 20th Spring Conference on Computer Graphics* (Budmerice, Slovakia) *(SCCG '04)*. Association for Computing Machinery, New York, NY, USA, 106–111. https://doi.org/10.1145/1037210.1037226

Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. 2021. Fourier Neural Operator for Parametric Partial Differential Equations. In *International Conference on Learning Representations*. https://openreview.net/forum?id=c8P9NQVtmnO

Or Litany, Tal Remez, Emanuele Rodola, Alex Bronstein, and Michael Bronstein. 2017. Deep functional maps: Structured prediction for dense shape correspondence. In *Proceedings of the IEEE international conference on computer vision*. 5659–5667.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. 2021. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence* 3, 3 (2021), 218–229.

Linjie Lyu, Ayush Tewari, Thomas Leimkühler, Marc Habermann, and Christian Theobalt. 2022. Neural Radiance Transfer Fields for Relightable Novel-View Synthesis with Global Illumination. In *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XVII* (Tel Aviv, Israel). Springer-Verlag, Berlin, Heidelberg, 153–169. https://doi.org/10.1007/978-3-031-19790-1_10

Robin Magnet, Jing Ren, Olga Sorkine-Hornung, and Maks Ovsjanikov. 2022. Smooth non-rigid shape matching via effective dirichlet energy optimization. In *2022 International Conference on 3D Vision (3DV)*. IEEE, 495–504.

Michal Mojzík, Alban Fichet, and Alexander Wilkie. 2018. Handling Fluorescence in a Uni-directional Spectral Path Tracer. *Comput. Graph. Forum (Proc. EGSR)* 37, 4 (2018), 77–94.

Mervin E Muller. 1956. Some continuous Monte Carlo methods for the Dirichlet problem. *The Annals of Mathematical Statistics* (1956), 569–589.

Ren Ng, Ravi Ramamoorthi, and Pat Hanrahan. 2003. All-frequency shadows using non-linear wavelet lighting approximation. *ACM Trans. Graph.* 22, 3 (July 2003), 376–381. https://doi.org/10.1145/882262.882280

Dorian Nogneng and Maks Ovsjanikov. 2017. Informative Descriptor Preservation via Commutativity for Shape Matching. *Comput. Graph. Forum* 36, 2 (May 2017), 259–267. https://doi.org/10.1111/cgf.13124

Maks Ovsjanikov, Mirela Ben-Chen, Justin Solomon, Adrian Butscher, and Leonidas Guibas. 2012. Functional maps: a flexible representation of maps between shapes. 31, 4, Article 30 (July 2012), 11 pages. https://doi.org/10.1145/2185520.2185526

Christoph Peters, Sebastian Merzbach, Johannes Hanika, and Carsten Dachsbacher. 2019. Using moments to represent bounded signals for spectral rendering. *ACM Trans. Graph.* 38, 4, Article 136 (July 2019), 14 pages. https://doi.org/10.1145/3306346.3322964

Matt Pharr, Wenzel Jakob, and Greg Humphreys. 2023. *Physically Based Rendering: From Theory to Implementation (4th ed.)* (4th ed.). The MIT Press, San Francisco, CA, USA. 1266 pages.

Nithin Raghavan, Yan Xiao, Kai-En Lin, Tiancheng Sun, Sai Bi, Zexiang Xu, Tzu-Mao Li, and Ravi Ramamoorthi. 2023. Neural Free-Viewpoint Relighting for Glossy Indirect Illumination. *Computer Graphics Forum (Proc. EGSR 2023)* (2023).

Gilles Rainer, Adrien Bousseau, Tobias Ritschel, and George Drettakis. 2022. Neural Precomputed Radiance Transfer. *Computer Graphics Forum (Proceedings of the Eurographics conference)* 41, 2 (April 2022). http://www-sop.inria.fr/reves/Basilic/2022/RBRD22

Jing Ren, Adrien Poulenard, Peter Wonka, and Maks Ovsjanikov. 2018. Continuous and orientation-preserving correspondences via functional maps. *ACM Trans. Graph.* 37, 6, Article 248 (Dec. 2018), 16 pages. https://doi.org/10.1145/3272127.3275040

Peiran Ren, Jiaping Wang, Minmin Gong, Stephen Lin, Xin Tong, and Baining Guo. 2013. Global illumination with radiance regression functions. *ACM Trans. Graph. (Proc. SIGGRAPH)* 32, 4 (2013), 130.

Jean-Michel Roufosse, Abhishek Sharma, and Maks Ovsjanikov. 2019. Unsupervised deep learning for structured shape matching. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 1617–1627.

Katherine Salesin and Wojciech Jarosz. 2019. Combining point and line samples for direct illumination. *Computer Graphics Forum (Proceedings of EGSR)* 38, 4 (July 2019), 159–169. https://doi.org/10/gf6rx6

Samuele Salti, Federico Tombari, and Luigi Di Stefano. 2014. SHOT: Unique signatures of histograms for surface and texture description. *Computer vision and image understanding* 125 (2014), 251–264.

Ari Silvennoinen and Peter-Pike Sloan. 2021. Moving Basis Decomposition for Precomputed Light Transport. *Computer Graphics Forum* 40, 4 (2021), 127–137. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14346 https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14346

Peter-Pike Sloan, Jesse Hall, John Hart, and John Snyder. 2003. Clustered principal components for precomputed radiance transfer. *ACM Trans. Graph.* 22, 3 (July 2003), 382–391. https://doi.org/10.1145/882262.882281

Peter-Pike Sloan, Jan Kautz, and John Snyder. 2002. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM Trans. Graph.* 21, 3 (July 2002), 527–536. https://doi.org/10.1145/566654.566612

Yu-Ting Tsai and Zen-Chung Shih. 2006. All-frequency precomputed radiance transfer using spherical radial basis functions and clustered tensor approximation. *ACM Trans. Graph.* 25, 3 (July 2006), 967–976. https://doi.org/10.1145/1141911.1141981

Jirí Vorba, Ondrej Karlík, Martin Sik, Tobias Ritschel, and Jaroslav Krivánek. 2014. On-line learning of parametric mixture models for light transport simulation. *ACM Trans. Graph. (Proc. SIGGRAPH)* 33, 4 (2014), 101:1–101:11.

A. Wilkie, S. Nawaz, M. Droske, A. Weidlich, and J. Hanika. 2014. Hero wavelength spectral sampling. In *Proceedings of the 25th Eurographics Symposium on Rendering* (Lyon, France) *(EGSR '14)*. Eurographics Association, Goslar, DEU, 123–131. https://doi.org/10.1111/cgf.12419

Mengqi (Mandy) Xia, Bruce Walter, Christophe Hery, and Steve Marschner. 2020. Gaussian Product Sampling for Rendering Layered Materials. *Computer Graphics Forum* 39, 1 (2020), 420–435. https://doi.org/10.1111/cgf.13883 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13883

Kun Xu, Wei-Lun Sun, Zhao Dong, Dan-Yong Zhao, Run-Dong Wu, and Shi-Min Hu. 2013. Anisotropic spherical Gaussians. *ACM Trans. Graph.* 32, 6, Article 209 (Nov. 2013), 11 pages. https://doi.org/10.1145/2508363.2508386

Zilin Xu, Zheng Zeng, Lifan Wu, Lu Wang, and Ling-Qi Yan. 2022. Lightweight Neural Basis Functions for All-Frequency Shading. In *SIGGRAPH Asia 2022 Conference Papers* (Daegu, Republic of Korea) *(SA '22)*. Association for Computing Machinery, New York, NY, USA, Article 14, 9 pages. https://doi.org/10.1145/3550469.3555386

Ling-Qi Yan, Miloš Hašan, Steve Marschner, and Ravi Ramamoorthi. 2016. Position-normal distributions for efficient rendering of specular microstructure. *ACM Trans. Graph.* 35, 4, Article 56 (July 2016), 9 pages. https://doi.org/10.1145/2897824.2925915

Yuhuan Yan, Mingquan Zhou, Dan Zhang, and Shengling Geng. 2023. Scale-invariant Mexican Hat wavelet descriptor for non-rigid shape similarity measurement. *Scientific Reports* 13, 1 (2023), 2518.

Haotian Yang, Mingwu Zheng, Wanquan Feng, Haibin Huang, Yu-Kun Lai, Pengfei Wan, Zhongyuan Wang, and Chongyang Ma. 2023. Towards Practical Capture of High-Fidelity Relightable Avatars. In *SIGGRAPH Asia 2023 Conference Papers* (Sydney, NSW, Australia) *(SA '23)*. Association for Computing Machinery, New York, NY, USA, Article 23, 11 pages. https://doi.org/10.1145/3610548.3618138

Zeyu Yang, Hongye Yang, Zijie Pan, and Li Zhang. 2024. Real-time Photorealistic Dynamic Scene Representation and Rendering with 4D Gaussian Splatting. In *International Conference on Learning Representations (ICLR)*.