

Supplemental Material

Don't Splat your Gaussians: Volumetric Ray-Traced Primitives for Modeling and Rendering Scattering and Emissive Media

JORGE CONDOR, Meta Reality Labs, Switzerland and IDSIA-USI Lugano, Switzerland

SÉBASTIEN SPEIERER, Meta Reality Labs, Switzerland

LUKAS BODE, Meta Reality Labs, Switzerland

ALJAŽ BOŽIČ, Meta Reality Labs, Switzerland

SIMON GREEN, Meta Reality Labs, United Kingdom

PIOTR DIDYK, IDSIA-USI Lugano, Switzerland

ADRIÁN JARABO, Meta Reality Labs, Spain

ACM Reference Format:

Jorge Condor, Sébastien Speierer, Lukas Bode, Aljaž Božič, Simon Green, Piotr Didyk, and Adrián Jarabo. 2024. Supplemental Material Don't Splat your Gaussians: Volumetric Ray-Traced Primitives for Modeling and Rendering Scattering and Emissive Media. *ACM Trans. Graph.* 0, 0, Article 0 (2024), 11 pages. <https://doi.org/0.0>

This supplemental document includes additional details on the derivations of our volumetric model, as well as its adjoint derivatives for inverse rendering. I also includes additional details on the optimization process for radiance fields, and additional results on both synthetic and captured datasets.

1 VOLUMETRIC PRIMITIVES

Inspired by recent works using Gaussian primitives for reconstructing radiance fields [Kerbl et al. 2023], and following a similar medium definition as Knoll et al. [2021], we model media using sets of primitives. Each primitive \mathcal{P}_i represents a statistical aggregate of matter with identical emission, cross section and phase function, and with density $\rho_i(\mathbf{x})$ defined by a three-dimensional un-normalized kernel $K_i(\mathbf{x})$. Given these primitives, we can model the distribution of matter, and therefore the extinction probability as a mixture of these

Authors' addresses: Jorge Condor, jorge.condor@usi.ch, Meta Reality Labs, Giesshübelstrasse 30, Zurich, Zurich, Switzerland, 8045 and IDSIA-USI Lugano, Via Giuseppe Buffi 13, Lugano, Ticino, Switzerland, 6900; Sébastien Speierer, speierers@meta.com, Meta Reality Labs, Giesshübelstrasse 30, Zurich, Switzerland, 8045; Lukas Bode, lboede@meta.com, Meta Reality Labs, Giesshübelstrasse 30, Zurich, Switzerland, 8045; Aljaž Božič, aljaz@meta.com, Meta Reality Labs, Giesshübelstrasse 30, Zurich, Switzerland, 8045; Simon Green, simongreen@meta.com, Meta Reality Labs, London, United Kingdom, 8045; Piotr Didyk, piotr.didyk@usi.ch, IDSIA-USI Lugano, Via Giuseppe Buffi 13, Lugano, Switzerland, 6900; Adrián Jarabo, ajarabo@meta.com, Meta Reality Labs, Zaragoza, Spain.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM. ACM 0730-0301/2024/0-ART0
<https://doi.org/0.0>

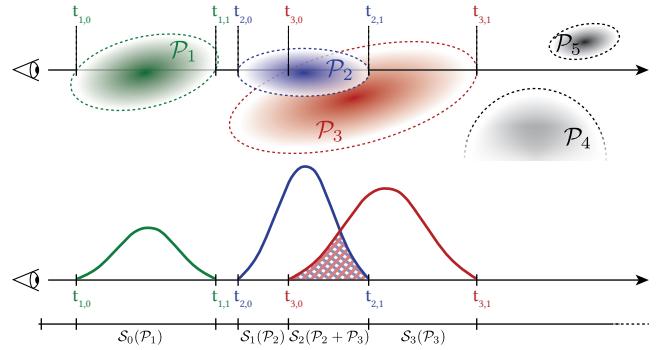


Fig. S.1. **Top:** Flatland representation of a medium modeled using $N_{\text{total}} = 5$ primitives \mathcal{P}_i with $i \in [1, 5]$. For the depicted ray from the camera only three primitives contribute directly to the ray. **Bottom:** The primitives are projected in 1D along the ray, and their density, emission and in-scattering are integrated over that line, which can be separated in disjoint segments defined by the boundaries of the primitives. The contribution of each segment S_k is the integral over the primitives overlapping on that segment.

volumetric primitives following

$$\mu_t(\mathbf{x}) = \sum_{i=1}^N \sigma_i K_i(\mathbf{x}), \quad (1)$$

with N the number of primitives affecting \mathbf{x} , and σ_i its cross-section. For kernels with infinite support, N is the total number of primitives $N_{\text{total}} = N$. In our case, we assume that all kernels have limited support, so typically $N < N_{\text{total}}$. The remaining optical properties are computed analogously.

1.1 The RTE with volumetric primitives

By plugging Equation (1) into the volumetric rendering equation, and omitting the boundary condition at \mathbf{x}_s for simplicity, we get

$$\begin{aligned} L(\mathbf{x}_0, \omega) &= \int_0^s T(\mathbf{x}, \mathbf{x}_t) \sum_{i=1}^N \sigma_i K_i(\mathbf{x}_t) L_{o,i}(\mathbf{x}_t, \omega) dt \\ &= \sum_{i=0}^N \sigma_i \int_{t_{i,0}}^{t_{i,1}} T(\mathbf{x}, \mathbf{x}_t) K_i(\mathbf{x}_t) L_{o,i}(\mathbf{x}_t, \omega) dt, \end{aligned} \quad (2)$$

with $t_{i,0} \geq 0$ and $t_{i,1} \leq t$ the boundaries of the segment defined by \mathbf{x}_t when intersecting the footprint of the primitive \mathcal{P}_i (see Figure S.1), and $L_{o,i}(\mathbf{x}, \omega)$ the outgoing radiance from primitive \mathcal{P}_i , defined as

$$L_{o,i}(\mathbf{x}_t, \omega) = (1 - \alpha_i) Q_i(\omega) + \alpha_i \int_{S^2} f_{textp,i}(\omega' \rightarrow \omega) L(\mathbf{x}, \omega') d\omega', \quad (3)$$

where α_i , $Q_i(\omega)$ and $f_{textp,i}(\omega' \rightarrow \omega)$ are the single scattering albedo, emission, and phase function of primitive \mathcal{P}_i , respectively. Note that they have no spatial dependency.

With the formulation of extinction in Equation (1), we can now express transmittance as the product of the transmittance of individual volumetric primitives, or as the sum of the optical depths of contributing kernels to the given segment:

$$\begin{aligned} T(\mathbf{x}_0, \mathbf{x}_t) &= \exp \left(- \int_0^t \sum_{i=1}^N \sigma_i K_i(\mathbf{x}_{t'}) dt' \right) \\ &= \exp \left(- \sum_{i=1}^N \int_0^t \sigma_i K_i(\mathbf{x}_{t'}) dt' \right) \\ &= \exp \left(- \sum_{i=1}^N \tau_i(\mathbf{x}, \mathbf{x}_t) \right) \end{aligned} \quad (4)$$

with $\tau_i(\mathbf{x}_a, \mathbf{x}_b)$ the optical depth from primitive \mathcal{P}_i in the range $t \in [a, b]$ defined as

$$\tau_i(\mathbf{x}_a, \mathbf{x}_b) = \sigma_i \int_{\max(a, t_{i,0})}^{\min(b, t_{i,1})} K_i(\mathbf{x}_{t'}) dt'. \quad (5)$$

Equation (4) can be solved with traditional stochastic tracking methods (e.g., Woodcock tracking [Woodcock 1965]). However, if we chose the kernel K carefully so that the line integral in Equation (5) has a closed-form solution, we can compute the transmittance $T(\mathbf{x}, \mathbf{x}_t)$ analytically. This is similar to regular tracking methods [Sutton et al. 1999].

Unfortunately, directly solving Equation (2) together with our transmittance estimator (4) involves a nested loop over potentially all N primitives intersected by the ray, leading to a computational complexity of $O(N^2)$. In the following, we propose an alternative formulation for Equation (2) more suitable for rendering.

1.2 Integrating through segments

Given the limited support of the kernels defining our primitives (or alternatively, our ability to clip kernels with infinite support), solving Equation (2) results into the summation of multiple line-integrals in potentially overlapping segments along the ray defined by $\mathbf{r}(t) = \mathbf{x}_0 - \omega t$, where each segment is the primitives' footprint along the ray. As we show in the simple example in Figure S.1 (bottom), the boundaries of each primitive $t_{i,0}$ and $t_{i,1}$ subdivide the ray into segments $\mathcal{S}_k = [\hat{t}_{k,0}, \hat{t}_{k,1}]$. Each of these segments \mathcal{S}_k might overlap with zero, one, or multiple primitives denoted with the set $\{\mathcal{P}_i | i \in \mathcal{S}_k\}$, where \mathcal{S}_k is the per-segment set of indices, with $|\mathcal{S}_k| \leq N$ the number of primitives overlapping in segment \mathcal{S}_k . Without loss of generality, let us assume that the segments are ordered, so that $\hat{t}_{k,0} = \hat{t}_{k-1,1}$ for $k > 0$, with $\hat{t}_{0,0} = \mathbf{x}$.

With that definition, we can rewrite Equation (2) as the summation of the radiance at each segment, following

$$L(\mathbf{x}_0, \omega) = \sum_{k=1}^M \int_{\hat{t}_{k,0}}^{\hat{t}_{k,1}} T(\mathbf{x}, \mathbf{x}_t) \sum_{i \in \mathcal{S}_k} \sigma_i K_i(\mathbf{x}_t) L_{o,i}(\mathbf{x}_t, \omega) dt, \quad (6)$$

with $M \leq 2N - 1$ the number of segments along the ray. Similarly, leveraging that the segments are ordered along the ray, we can rewrite Equation 4 as a recursive operator, defining the transmittance after \mathcal{S}_k as

$$\begin{aligned} T_k(\mathbf{x}_0, \mathbf{x}_t) &= T_{k-1}(\mathbf{x}_0, \mathbf{x}_t) \exp \left(- \sum_{i \in \mathcal{S}_k} \tau_i \left(\mathbf{x}_{\hat{t}_{k,0}}, \mathbf{x}_{\min(t, \hat{t}_{k,1})} \right) \right), \\ &= T_{k-1}(\mathbf{x}_0, \mathbf{x}_t) T(\mathbf{x}_{\hat{t}_{k,0}}, \mathbf{x}_t), \end{aligned} \quad (7)$$

with $T_0(\mathbf{x}_0, \mathbf{x}_t) = 1$, $\tau_i(\mathbf{x}_a, \mathbf{x}_b) = 0$ for $a \geq b$, and $T_M(\mathbf{x}_0, \mathbf{x}_t) = T(\mathbf{x}_0, \mathbf{x}_t)$. Plugging Equation (7) into (6) we thus get

$$L(\mathbf{x}_0, \omega) = \sum_{k=1}^M T_{k-1}(\mathbf{x}_0, \mathbf{x}_t) L_k(\mathbf{x}_{\hat{t}_{k,0}}, \omega), \quad (8)$$

with $L_k(\mathbf{x}_{\hat{t}_{k,0}}, \omega)$ the outgoing radiance at the segment \mathcal{S}_k defined as

$$L_k(\mathbf{x}_{\hat{t}_{k,0}}, \omega) = \int_{\hat{t}_{k,0}}^{\hat{t}_{k,1}} T(\mathbf{x}_{\hat{t}_{k,0}}, \mathbf{x}_t) \sum_{i \in \mathcal{S}_k} [\sigma_i K_i(\mathbf{x}_t) L_{o,i}(\mathbf{x}_t, \omega)] dt. \quad (9)$$

Note that this form of Equation (2) actually introduces additional computational complexity, since in its worst case scenario (all primitives overlap, leading to a segment with N overlapping primitives), the computational cost of Equation (8) is $O(N^3)$. However, in realistic scenarios the number of overlapping primitives $|\mathcal{S}_k|$ at a given segment \mathcal{S}_k is not high. This approach has benefits for closed-form integration of the inner integral of the summation in certain scenarios, as we will show later for emissive media. Note that this approach is similar to how production renderers deal with complex overlapping media [Fong et al. 2017, Ch.6]; the key difference is that our approach approximates the media with primitives, which allows having closed-form transmittance and sampling expressions.

1.3 Sampling transmittance

In order to solve Equation (6), specially the in-scattering integral, we need to sample the distance t along the ray. The most common approach is to sample t with probability distribution function (PDF) $p(t) = \mu_t(\mathbf{x}_t) T(\mathbf{x}_0, \mathbf{x}_t)$, and therefore with cumulative distribution function (CDF) $P(t) = T(\mathbf{x}_0, \mathbf{x}_t)$. While this is trivial for homogeneous media, for the most general case of heterogeneous media a tracking null-scattering-based approach is required [Miller et al. 2019; Novák et al. 2018], which requires a careful choice of majorant to maximize performance [Kutz et al. 2017; Misso et al. 2023].

While our volumetric primitive-based definition of the media allows the use of tracking estimators for distance sampling, we found that we can leverage the recursive formulation of transmittance (7) and pose the problem as an iterative search problem, where we uniformly sample the transmittance with a random variable $\xi \in (0, 1)$, and search for the distance t so that $\xi(t) = T(\mathbf{x}_0, \mathbf{x}_t)$. This approach

is very similar to regular tracking [Sutton et al. 1999], which we extend to handle overlapping kernels efficiently. In particular, we first search for a segment S_k so that $(1 - \xi) \in [T_{k-1}, T_k]$. Then, we invert the transmittance inside the segment, solving for $t \in [\hat{t}_{k,0}, \hat{t}_{k,1}]$ equation

$$\log(1 - \xi) = - \sum_{i \in S_k} \tau_i(\mathbf{x}_{\hat{t}_{k,0}}, \mathbf{x}_t). \quad (10)$$

Depending on the type of kernel $K_i(\mathbf{x})$ being used, and the amount of overlaps in segment S_k this equation may have a closed-form analytical solution. However, for cases where Equation (10) does not have a simple analytical solution (e.g. many overlapping kernels at the same segment), we rely on numerical root-finding. Here, we propose two methods.

Newton-Raphson Solver. Our first choice, given its efficiency, is the Newton-Raphson iterative method [Brown and Martin 2003]. In particular, we iteratively find the distance t by solving

$$t_{j+1} = t_j - \frac{\sum_{i \in S_k} \tau_i(\mathbf{x}_{\hat{t}_{k,0}}, \mathbf{x}_{\min(t_j, \hat{t}_{k,1})})}{\sum_{i \in S_k} \sigma_i K_i(\mathbf{x}_{t_j})}, \quad (11)$$

with j the iterator, and t_r and t_{r+1} the current and next solution. As long as extinction is positive, the Newton-Raphson solver is guaranteed to converge [Brown and Martin 2003]. We found that we can achieve good precision with just a few iterations (1-3), given that our initial estimates of the sampled distance are usually quite close to the final solution already (we use half the segment length as initial estimate, and clip the distance to the segment bounds during the iteration). The Newton-Raphson solver works particularly well on kernels with limited support and/or short tails (i.e. Epanechnikov, bicubic, triangular...). However, long-tailed kernels, or those with infinite support (i.e. Gaussian) will occasionally suffer from numerical instability when attempting to sample segments on the very edge of the primitive shells. In practice, this translates into a small amount of rendering artifacts, as sampled distance will clip to one of the segment limits. To overcome this, at a small performance cost, we suggest using the slower, but more stable, bisection method.

Bisection Solver. Given a fixed segment where we know the root will be present (integration segment limits t_a and t_b), we first compute its midpoint as $t_c = \frac{t_a + t_b}{2}$. Then, we re-compute accumulated optical depth up to the midpoint $\sum_{i \in S_k} \tau_i(\mathbf{x}_{\hat{t}_{k,a}}, \mathbf{x}_{\hat{t}_{k,c}})$. If the predicted optical depth falls within a margin of $-\log(1 - \xi)$, the recursion ends. Otherwise, depending on whether the difference between them is positive or negative (either the proposed distance under or overshoots the target), we will update either t_a or t_b with t_c , and start the process again. While fairly simple, due to the normally short segment length no more than 4-8 iterations are usually needed to obtain reasonable estimates. Even when analytical solutions to the inverse sampling problem exist, due to the numerical inaccuracy or expensive cost of some inverse analytical functions, we can still revert to the solvers even on segments with a single primitive.

1.4 Emissive media

So far, we have dealt with the emission $Q(\mathbf{x}, \omega)$ and in-scattering $S(\mathbf{x}, \omega)$ in a unified fashion inside $L_o(\mathbf{x}_t, \omega)$. However, these two

terms are generally not correlated (emissive media is generally non-scattering, and vice-versa) and the behaviour of both terms is different. Thus, a common approach is to compute these two terms as separate integrals.

In the case of emissive media, we can assume a medium with no scattering (i.e., with $\alpha = 0$), and where all contribution is due to the source term $Q(\mathbf{x}, \omega)$. In that case, leveraging our assumption of constant reduced emission $\bar{Q}_i(\omega) = (1 - \alpha_i)Q_i(\omega)$ in primitive P_i and expanding the transmittance term $T(\mathbf{x}_{\hat{t}_{k,0}}, \mathbf{x}_t)$, we transform Equation (9) as

$$L_k(\mathbf{x}_{\hat{t}_{k,0}}, \omega) = \int_{\hat{t}_{k,0}}^{\hat{t}_{k,1}} e^{\int_{\hat{t}_{k,0}}^{x_t} \sum_{i \in S_k} \sigma_i K_i(\mathbf{x}_t) dt'} \sum_{i \in S_k} \left[\sigma_i K_i(\mathbf{x}_t) \bar{Q}_i(\omega) \right] dt. \quad (12)$$

For a single primitive inside the segment, this integral has closed form depending on the kernel. However, for multiple overlapping primitives it does not, and we need to compute Equation (12) numerically. For our radiance field reconstruction application, we approximate (12) using a simpler light transport model.

Discussion. We have derived a general framework for radiative transport in primitive-based media, defined using arbitrary kernel functions. Any kernel could be used within our framework as long as 1) they have limited support or their decay is such that they can be bounded or clipped efficiently and 2) closed-form solutions to their line integrals exist or can be numerically computed efficiently.

2 ADJOINT DERIVATIONS

Here we include the full derivation of the adjoints for both the VPPT and VPRF integrators. For simplicity, in the following formulas we omit the set P of kernel primitives for all summations and products.

2.0.1 Adjoint VPRF. The image formation model of this integrator iterates over all segments along the ray, gathering the kernel primitives contributing to each segment P and compute the emission as follows:

$$L_e = \sum_i \left(E_i \cdot (1 - \prod_j e^{-\tau_j}) \cdot \frac{\tau_i}{\sum_j \tau_j} \right).$$

With E_i the emission of the primitive i in the segment and τ_i the integrated density of the primitive i for that segment. In the adjoint formulation of this integrator, we are interested in the derivative δL_e , defined as

$$\begin{aligned} \delta L_e &= \delta \sum_i \left(E_i \cdot (1 - \prod_j e^{-\tau_j}) \cdot \frac{\tau_i}{\sum_j \tau_j} \right) \\ &= \sum_i \delta \left(E_i \cdot (1 - \prod_j e^{-\tau_j}) \cdot \frac{\tau_i}{\sum_j \tau_j} \right). \end{aligned}$$

Now using the product rule, we can explode the within the loop and differentiate each term individually:

$$\begin{aligned}
& \delta \left(E_i \cdot \left(1 - \prod_j e^{-\tau_j} \right) \cdot \frac{\tau_i}{\sum_j \tau_j} \right) = \\
& \delta(E_i) \cdot \left(1 - \prod_j e^{-\tau_j} \right) \cdot \tau_i \cdot \frac{1}{\sum_j \tau_j} \quad (a) \\
& + E_i \cdot \delta \left(1 - \prod_j e^{-\tau_j} \right) \cdot \tau_i \cdot \frac{1}{\sum_j \tau_j} \quad (b) \\
& + E_i \cdot \left(1 - \prod_j e^{-\tau_j} \right) \cdot \delta(\tau_i) \cdot \frac{1}{\sum_j \tau_j} \quad (c) \\
& + E_i \cdot \left(1 - \prod_j e^{-\tau_j} \right) \cdot \tau_i \cdot \delta \left(\frac{1}{\sum_j \tau_j} \right) \quad (d)
\end{aligned}$$

Terms (a) and (c) don't need any special handling to be differentiated. For (b) we apply the product rule:

$$\delta \left(1 - \prod_j e^{-\tau_j} \right) = -\delta \left(\prod_j e^{-\tau_j} \right) = \sum_j -\delta(e^{-\tau_j}) \frac{\prod_k e^{-\tau_k}}{e^{-\tau_j}}$$

For the term (d), we can use the chain rule as follow:

$$\delta \left(\frac{1}{\sum_j \tau_j} \right) = \frac{-1}{(\sum_j \tau_j)^2} \cdot \delta \left(\sum_j \tau_j \right) = \sum_j -\delta(\tau_j) \cdot \frac{1}{(\sum_k \tau_k)^2}$$

The segment emission L_e is also multiplied by the path throughput β , which accumulate transmittance along the ray as follow:

$$L = \beta \cdot L_e = \prod_s T_s \cdot L_e$$

with $\prod_s T_s$ the path throughput as the product of the transmittance of the previously visited segments. To compute the derivatives we can again apply the chain rule:

$$\begin{aligned}
\delta L &= \delta \left(\prod_s T_s \right) \cdot L_e + \prod_s T_s \cdot \delta(L_e) \\
&= \sum_j \delta(T_j) \frac{\prod_k T_k}{T_j} \cdot L_e + \prod_s T_s \cdot \delta(L_e)
\end{aligned}$$

With the formulations above, we can implement the adjoint of this image formation model as presented in Listing 1.

```

1 def primitive_tracing(ray, max_depth, P = []):
2     depth = 0
3     t0 = 0.0 # Current segment start time
4     while depth < max_depth:
5         V = [(p, p.t1) for p in P] # List primitive exit points
6         p = ray_intersect(ray)      # Find next primitive
7         V.append((p, p.t0))        # Add primitive entry point
8
9         # Process all exit points up to this intersection
10        while not V.is_empty():
11            # Find next vertex to process
12            v = min(V, key=lambda x: x.t)
13
14            # Process segment according to application
15            process_segment(ray, G, t0, v.t)
16
17            t0 = v.t # Move on to next segment
18            if p == v.p: break # Check if we have reached p
19            P.remove(v.p) # Exiting v.p
20            V.remove(v)
21
22        P.append(p) # Ray is now entering primitive p
23        ray = ray.move_to(p.t0) # Update ray position
24        depth += 1

```

Listing 1. Pseudo-code implementation of the adjoint logic for *VPRF*.

2.0.2 Adjoint VPPT. In this integrator, a scattering event S is always associated to a set of kernel primitives contributing to the density in that point in space. As commonly used in the RTE, we are required to compute the single scattering albedo A_S of this scattering event, which will be multiplied with the path throughput β , the transmittance T_S and the density of the scattering event μ_S , to finally weight any further incoming radiance L_i .

This can be summarized in the following equation:

$$L = \beta \cdot A_S \cdot T_S \cdot \mu_S \cdot L_i$$

and the corresponding derivatives:

$$\begin{aligned}
\delta L &= \delta(\beta \cdot A_S \cdot T_S \cdot \mu_S \cdot L_i) \\
&= \delta(\beta) \cdot A_S \cdot T_S \cdot \mu_S \cdot L_i \\
&+ \beta \cdot \delta(A_S) \cdot T_S \cdot \mu_S \cdot L_i \\
&+ \beta \cdot A_S \cdot \delta(T_S) \cdot \mu_S \cdot L_i \\
&+ \beta \cdot A_S \cdot T_S \cdot \delta(\mu_S) \cdot L_i \\
&+ \beta \cdot A_S \cdot T_S \cdot \mu_S \cdot \delta(L_i)
\end{aligned}$$

Computing δA_S . When given a set of kernel primitives contributing to the density at a scattering event, we compute the sum of their respective albedo values a_i , weighted by their being their PDF value p_i at the scattering location. This can be written as follow:

$$A_S = \sum_i a_i p_i \cdot \frac{1}{\sum_i p_i}$$

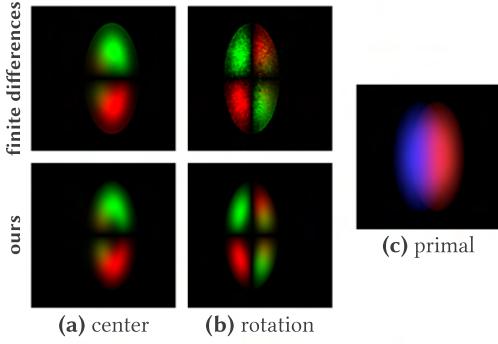


Fig. S.2. Validation of the gradients computed by *VPPT*, compared with finite differences, on a couple of overlapping Gaussian kernels with an isotropic scattering function. Primal image can be visualized on the right.

In order to compute the derivative of this term, we can apply the product and chain rules to put the derivative terms under a single sum, translating to a single loop in the adjoint implementation:

$$\begin{aligned}\delta A_S &= \delta \left(\sum_i a_i p_i \cdot \frac{1}{\sum_i p_i} \right) \\ &= \delta \left(\sum_i a_i p_i \right) \cdot \frac{1}{\sum_i p_i} + \delta \left(\frac{1}{\sum_i p_i} \right) \cdot \sum_i a_i p_i \\ &= \sum_i \delta(a_i p_i) \cdot \frac{1}{\sum_i p_i} - \frac{1}{(\sum_i p_i)^2} \cdot \delta(\sum_i p_i) \cdot \sum_i a_i p_i \\ &= \sum_i \left(\delta(a_i p_i) \cdot \frac{1}{\sum_j p_j} - \delta(p_i) \cdot \frac{1}{(\sum_j p_j)^2} \cdot \sum_j a_j p_j \right)\end{aligned}$$

Computing $\delta(T_S)$. The transmittance of a path segment is given by the product of the transmittance of the kernel primitive along that segment as shown here, with τ_i the optical depth of each kernel:

$$T_S = \prod_i e^{\tau_i}$$

To transform this chain of products into a sum, we simply apply the product rule once again:

$$\delta T_S = \delta \left(\prod_i e^{\tau_i} \right) = \sum_i \delta(e^{\tau_i}) \cdot \frac{(\prod_j e^{\tau_j})}{e^{\tau_i}}$$

Computing $\delta\mu_S$. This one is simple since it is already defined as a sum:

$$\delta\mu_S = \sum_i \delta\mu_i$$

We validate the accuracy of the computed gradients using our adjoint derivation in Figure S.2, by comparing it to finite differences.

3 MIXTURE MODEL REGRESSION

Depending on asset complexity, we present two different pipelines to regress our mixture models. In the case of relatively simple assets, we can leverage the accuracy of our analytic transmittance gradient computation to setup an image-based tomographic reconstruction. However, akin to previous primitive-based volumetric representations (i.e. 3DGS), starting from a random set of primitives and optimizing end-to-end can result in degraded regression quality when the complexity of the reference asset is high. In order to improve the pipeline, we include two additional pre-processing steps based on traditional mixture model regression literature to provide a better starting point to our tomographic reconstruction pipeline.

Randomly initialized reconstruction. We first render a set of tomographic images of our asset as training data. We start on a single primitive, and progressively adds new primitives throughout the optimization via random splitting of the optimized set every few iterations. Similarly to 3DGS, we alternate this procedure with pruning steps to eliminate small or very shallow primitives. As optimization loss, we use a combination of L1, D-SSIM and an anisotropy regularization. Full details of this pipeline are presented in Section ??, which we use in that case for the substantially more complex inverse scattering optimization. We use this approach to regress the *Smoke* asset; in Figure ??, we compare between the original grid-based asset and our reconstructed GMM, showcasing the quality and compactness of our representation. Additionally, since our approach is image-based, it can potentially be used in other tomographic applications, where explicit density representations (voxelgrids) are not available.

EM-initialized reconstruction. We first use the Expectation - Maximization algorithm [Moon 1996] to obtain a dense GMM representation of the discrete voxel-grid based distribution we want to model (40k Gaussians from 10M sampling points). While the quality of the regressed asset is very high, it normally produces high amounts of large overlapping kernels, which hampers the efficiency of our rendering approach. To reduce the complexity in a principled way, we perform an aggregation step following hierarchical clustering using the Kullback–Leibler divergence as a measure of similarity, similar to Jakob et al. [Jakob et al. 2011]. The amount of aggregation can vary depending on the desired complexity of the mixture. Finally, we employ the tomographic reconstruction method described above to further distill our mixture and regularize its anisotropy to enhance performance. We use this procedure for both the *Cloud* and *Dust Explosion* assets.

4 OPTIMIZATION AND RENDERING OF RADIANCE FIELDS - IMPLEMENTATION DETAILS

4.1 Initialization

We warm start our optimization on previously trained scenes using 3D Gaussian splatting [Kerbl et al. 2023] (3DGS). Our different image formation model impedes us from directly rendering their sets of primitives, but they represent a better, denser initialization for our method than random initializations or the relatively sparse SfM point sets.

Parameter	Initial LR	Final LR
Color Emission	0.1	0.01
Means	8e-3	4e-5
Rotations	8e-3	2e-5
Scales	5e-4	1e-5
Opacity	1e-3	3e-5

Table 1. List of parameters and their initial and final learning rate values. They are exponentially annealed throughout the optimization until the 3072th iteration.

4.2 Optimization Process

Rendering. Each iteration renders rays in batches of 8 full images for high-resolution, real datasets, and 16 for synthetic ones. Due to our reliance on D-SSIM loss, we cannot render random rays from the reference dataset for each batch, which would be more stable from the optimization point of view. We use 4 levels of SHs to model the anisotropic color emission, and optimize for a fixed 4096 iterations. As opposed to 3DGS, we use the same parameters in all our experiments, for both synthetic and real datasets. Fine-tuning per dataset could substantially improve the presented results.

Optimization overview. We start by solely optimizing for the opacity to improve our 3DGS initialization. This is followed by an emission optimization-only step, compensating for a slightly different color function. Both are done for 64 iterations, where we freeze primitive parameters and optimize the alpha and zeroth level of the spherical harmonics emission respectively. We then proceed to optimize for each parameter at different learning rates, using the previously introduced *Bounded Adam* optimizer (with learning rates on Table 1) and means bounded by the scene scale defined by SfM. We strictly control learning rate annealing through an exponential decay function akin to the one in 3DGS. [2023], but use it for all parameters as opposed to solely for the primitives' mean. All of our scenes use the same hyperparameters, settings and procedure.

Primitive pruning and cloning strategies. During the optimization process, we use different strategies to help avoid underreconstruction and improve its efficiency. We prune primitives approaching the density lower bound, as we deem their small contribution to the scene reduces our performance for little gain in quality. We heuristically found a value of $\sigma_{\text{threshold}} = 1e-5$ to work well for all scenes. We prune every 200 iterations until the 3600th. We couple this with random spawning strategy, where we randomly clone existing primitive means with a small amount of added random perturbation. The new primitives have a fixed starting density of 0.6 and scale of 0.02, to avoid potentially new blocking artifacts substantially impacting the optimization while still helping with holes. We practically found that cloning 10% of the primitives every 100 iterations gives ample time to the optimization to decide whether the new primitives improve the reconstruction or otherwise, in which case they will be pruned. We keep cloning until the 2400th iteration. In our experience, this stochastic process of adding noise (and later denoising through pruning strategies) was more robust than 3DGS's gradient-based splitting and cloning, where the choice of this heuristic threshold is difficult to control. Gradient-based

thresholds heavily depend on the scale of the scene being modelled; extending to new scenes requires expensive iteration and testing over different values. Similarly to 3DGS, we initially optimize solely the diffuse (zeroth order) SH color emission, which we progressively upsample every 300 iterations, resulting in a more stable training. For the results displayed in Tables 2, 3 and 4 we run our pipeline for 4096 iterations (plus the 128 of initial opacity and diffuse color), although, as in 3DGS, longer training times might produce better results.

4.3 Loss Function

Similarly to 3DGS, we use a λ weighted combination of a mean-absolute error (L1) loss and D-SSIM [Wang et al. 2004] (13).

$$\mathcal{L} = (1 - \lambda) \frac{1}{N} \sum_{i=0}^N |(x - x_i)| + \lambda(1 - \text{SSIM}(x, x_i)). \quad (13)$$

In our experiments, we use $\lambda = 0.2$.

4.4 Rendering parameters

In order to strike a good balance between optimization speed and final mixture quality, we render at 1 sample per ray, but deactivate typical performance trade-offs like Russian Roulette. We also use unlimited primitive depths and employ analytic ellipsoidal shells. Optimizing with some of this parameters on (particularly, the different shells) might produce mixtures that are both more efficient and compensate for some of the artifacts they might generate; however in order to better compare to previous works, we decided to follow a safer approach and maximize quality.

5 OPTIMIZATION AND RENDERING OF RADIANCE FIELDS - QUANTITATIVE COMPARISON TO PREVIOUS WORK

For accurate comparison to previous works, in the results presented below we deactivate Russian Roulette, uncap the maximum depth, use 128spp for rendering and employ analytical ellipsoid shells. This improves quality (the difference between 1 and 128spp is usually around 1.5 PSNR due to better sampling of edges and small features) at the cost of performance. All images presented unless otherwise stated were generated using 8spp.

REFERENCES

- Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. 2022. Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields. *Proceedings of CVPR* (2022).
- Forrest B. Brown and William R. Martin. 2003. Direct sampling of Monte Carlo flight paths in media with continuously varying cross-sections. <https://api.semanticscholar.org/CorpusID:38456058>
- Julian Fong, Magnus Wrenninge, Christopher Kulla, and Ralf Habel. 2017. Production volume rendering. In *ACM SIGGRAPH 2017 Courses*.
- Wenzel Jakob, Christian Regg, and Wojciech Jarosz. 2011. Progressive Expectation-Maximization for Hierarchical Volumetric Photon Mapping. *Computer Graphics Forum (Proceedings of EGSR)* 30, 4 (June 2011). <https://doi.org/10/dtwcj>
- 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/>
- Aaron Knoll, Gregory P. Johnson, and Johannes Meng. 2021. *Path Tracing RBF Particle Volumes*. Apress, Berkeley, CA, 713–723. https://doi.org/10.1007/978-1-4842-7185-8_44

Table 2. Quantitative comparison of radiance field rendering on synthetic datasets (NeRF-Blender)

Method	Bulldozer		Ficus		Hotdog		Drums		Ship		Materials		Mic		Chair	
	PSNR↑	SSIM↑	PSNR↑	SSIM↑	PSNR↑	SSIM↑	PSNR↑	SSIM↑	PSNR↑	SSIM↑	PSNR↑	SSIM↑	PSNR↑	SSIM↑	PSNR↑	SSIM↑
NeRF [Mildenhall et al. 2020]	32.54	0.961	30.13	0.964	36.18	0.974	25.01	0.925	28.65	0.856	29.62	0.949	32.91	0.980	33.00	0.967
Plenoxels [?]	34.10	0.975	31.83	0.976	36.43	0.980	25.35	0.933	29.62	0.890	29.14	0.949	33.26	0.985	33.98	0.977
iNGP-Base [Müller et al. 2022]	35.64	0.981	30.29	0.972	37.01	0.982	24.575	0.930	30.60	0.892	28.955	0.944	35.40	0.989	35.06	0.984
Mip-NeRF 360 (Big) [Barron et al. 2022]	36.10	0.980	33.19	0.979	37.71	0.982	25.60	0.931	31.26	0.893	29.90	0.949	36.52	0.991	35.65	0.983
3D Gaussian Splatting (30k) [Kerbl et al. 2023]	36.073	0.982	35.49	0.986	38.07	0.985	26.27	0.954	31.65	0.905	30.49	0.960	36.77	0.992	35.53	0.987
Ours - Gaussian Kernels	32.11	0.957	33.37	0.973	34.24	0.973	24.74	0.921	29.11	0.872	29.49	0.949	31.64	0.971	29.78	0.957
Ours - Epanechnikov Kernels	33.60	0.972	31.92	0.970	36.14	0.977	25.56	0.941	29.69	0.883	29.33	0.952	33.21	0.983	32.68	0.972

Table 3. Quantitative comparison of radiance field rendering on real datasets (MipNeRF-360)

Method	bicycle		garden		stump		room		counter		kitchen		bonsai	
	PSNR↑	SSIM↑	PSNR↑	SSIM↑	PSNR↑	SSIM↑	PSNR↑	SSIM↑	PSNR↑	SSIM↑	PSNR↑	SSIM↑	PSNR↑	SSIM↑
NeRF [Mildenhall et al. 2020]	21.76	0.455	23.11	0.546	21.73	0.453	28.56	0.843	25.67	0.775	26.31	0.749	26.81	0.792
Plenoxels [?]	21.91	0.496	23.49	0.606	20.66	0.523	27.59	0.842	23.62	0.759	23.42	0.648	24.67	0.814
iNGP-Base [Müller et al. 2022]	22.19	0.491	24.60	0.649	23.63	0.574	29.27	0.855	26.44	0.798	28.55	0.818	30.34	0.890
Mip-NeRF 360 (Big) [Barron et al. 2022]	24.31	0.685	26.88	0.809	26.18	0.74	31.47	0.910	29.45	0.892	31.99	0.917	33.40	0.938
3D Gaussian Splatting (30k) [Kerbl et al. 2023]	25.25	0.771	27.41	0.868	26.55	0.775	30.63	0.914	28.70	0.905	30.32	0.922	31.98	0.938
Ours - Gaussian Kernels	22.07	0.536	26.14	0.807	23.83	0.614	29.86	0.895	28.23	0.879	29.44	0.890	31.68	0.929
Ours - Epanechnikov Kernels	21.56	0.515	25.23	0.747	23.81	0.623	30.21	0.901	27.96	0.866	28.76	0.864	30.95	0.917

Table 4. Quantitative comparison of radiance field rendering on real datasets (Tanks&Temples and Deep Blending)

Method	truck		train		Dr. Johnson		Playroom	
	PSNR↑	SSIM↑	PSNR↑	SSIM↑	PSNR↑	SSIM↑	PSNR↑	SSIM↑
NeRF [Mildenhall et al. 2020]	21.76	0.455	19.40	0.376	23.11	0.546	21.73	0.453
Plenoxels [?]	23.22	0.774	18.93	0.663	23.14	0.787	22.98	0.802
iNGP-Base [Müller et al. 2022]	23.26	0.779	20.17	0.666	27.75	0.839	19.48	0.754
Mip-NeRF 360 (Big) [Barron et al. 2022]	24.91	0.857	19.52	0.660	29.14	0.901	29.66	0.900
3D Gaussian Splatting (30k) [Kerbl et al. 2023]	25.19	0.879	21.10	0.802	28.77	0.899	30.04	0.906
Ours - Gaussian Kernels	23.93	0.850	20.24	0.743	27.37	0.863	28.75	0.892
Ours - Epanechnikov Kernels	23.98	0.845	20.43	0.750	27.52	0.864	28.89	0.895

- Peter Kutz, Ralf Habel, Yining Karl Li, and Jan Novák. 2017. Spectral and decomposition tracking for rendering heterogeneous volumes. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1–16.
- Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *ECCV*.
- Bailey Miller, Iliyan Georgiev, and Wojciech Jarosz. 2019. A null-scattering path integral formulation of light transport. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 38, 4 (July 2019). <https://doi.org/10/gf6rzb>
- Zackary Misso, Yining Karl Li, Brent Burley, Daniel Teece, and Wojciech Jarosz. 2023. Progressive null-tracking for volumetric rendering. In *ACM SIGGRAPH Conference Papers*. <https://doi.org/10/kmdw>
- T.K. Moon. 1996. The expectation-maximization algorithm. *IEEE Signal Processing Magazine* 13, 6 (1996), 47–60. <https://doi.org/10.1109/79.543975>
- Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM Trans. Graph.* 41, 4 (2022).
- Jan Novák, Iliyan Georgiev, Johannes Hamka, and Wojciech Jarosz. 2018. Monte Carlo methods for volumetric light transport simulation. *Computer Graphics Forum* 37, 2 (2018), 551–576.

- T M Sutton, F B Brown, F G Bischoff, D B MacMillan, C L Ellis, J T Ward, C T Ballinger, D J Kelly, and L Schindler. 1999. The Physical Models and Statistical Procedures Used in the RACER Monte Carlo Code. (7 1999). <https://doi.org/10.2172/767449>
- Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing* 13, 4 (2004), 600–612. <https://doi.org/10.1109/TIP.2003.819861>
- E Woodcock. 1965. Techniques used in the GEM code for Monte Carlo neutronics calculations in reactors and other systems of complex geometry. *Proceedings of the conference on applications of computing methods to reactor problems*, 1965 557 (1965). <https://cir.nii.ac.jp/crid/1573668925582592640>



Fig. S.3. Qualitative results from randomly picked views of the test set using our optimization and rendering pipeline, in synthetic datasets (Blender Synthetic Dataset [Mildenhall et al. 2020]). From top to bottom: *Bulldozer*, *Mic*, *Ship*, *Hotdog*, *Materials*, *Chair*, *Ficus*, *Drums*. These were rendered at 16spp, analytic bounding ellipsoids and unlimited depth.

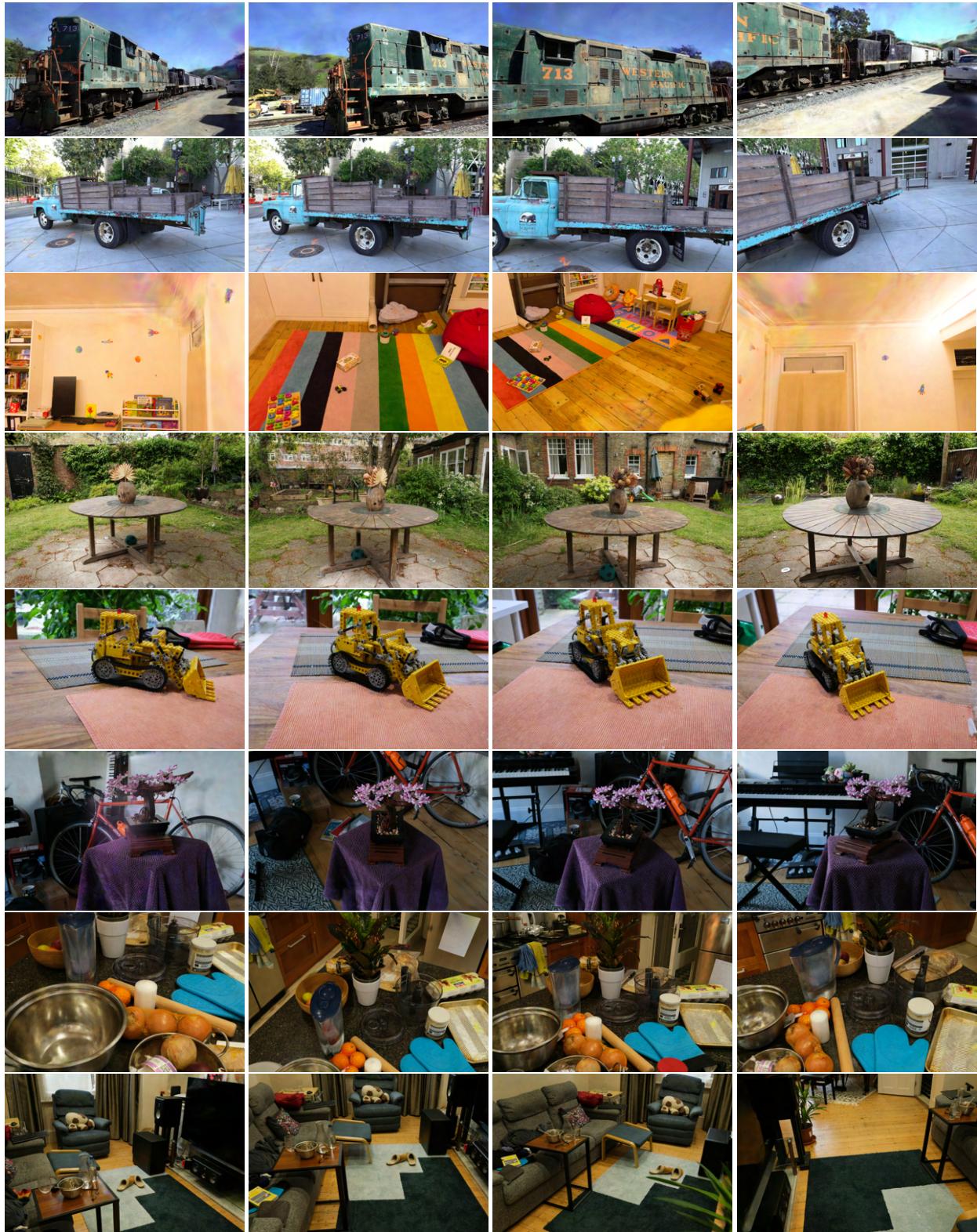


Fig. S.4. Qualitative results from the first 4 views of the test set using our optimization and rendering pipeline, in real datasets. From top to bottom: *train*, *truck* (Tanks&Temples dataset), *playroom* (Deep Blending dataset), *garden*, *kitchen*, *bonsai*, *counter* and *room* (Mip-NeRF360 datasets). Rendered at 8spp, analytic bounding ellipsoids and unlimited depth.

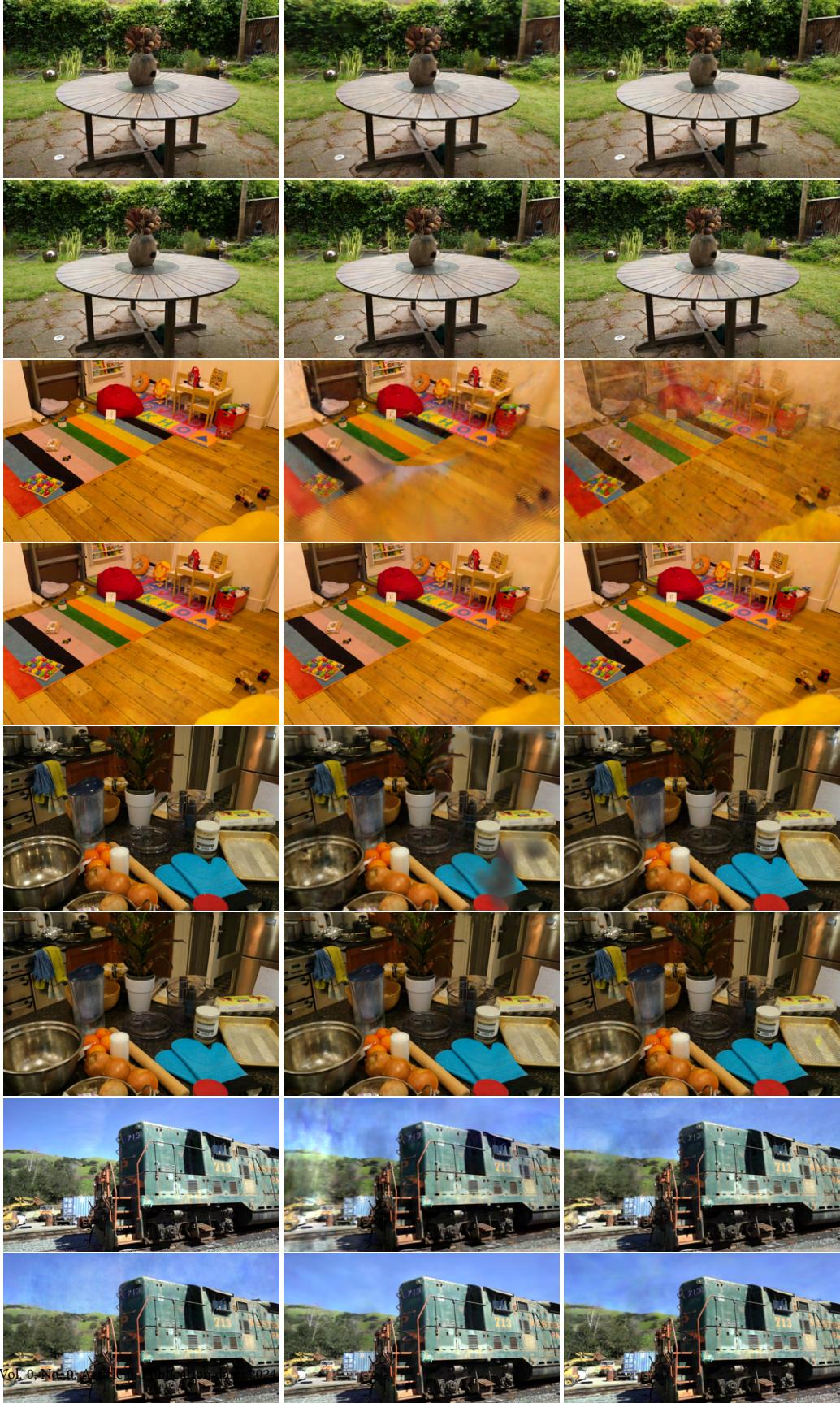


Fig. S.5. Expanded insets from Figure 11. From left to right: reference image, Plenoxels [?], Instant NGPs [Müller et al. 2022], (first row); Mip-NeRF360 [Barron et al. 2022], 3DGS [Kerbl et al. 2023] and ours (second row). Datasets, from top to bottom: *garden* (Mip-NeRF360 dataset), *playroom* (Deep Blending dataset), *counter* (Mip-NeRF360 dataset) and *train* (Tanks&Temples dataset).

Table 5. Performance analysis of our method on different quality settings. Quality presets use low-poly icosphere shells, 1 sample per pixel and 64 maximum depth. Performance presets limit maximum depth for a maximum quality drop of ~2-3 PSNR. In practice, this means limiting recursion to a maximum between 16 to 32 bounces depending on the dataset and type of kernel. All datasets feature a maximum of 300k primitives. Analogously to timings reported by Kerbl et al. [2023], we measure rendering times on an RTX A6000. Reported timings are the result of averaging over the entirety of their respective datasets.

Dataset	Gaussian Kernels		Epanechnikov Kernels	
	Quality	Performance	Quality	Performance
Real Datasets (resolution)		Execution time (ms)		
truck (979x576)	14.574	13.176	8.951	5.823
train (979x576)	22.672	16.348	14.131	11.923
Dr. Johnson (1332x876)	20.552	13.970	13.491	10.550
playroom (1264x832)	21.050	13.576	15.231	14.803
bicycle (1237x822)	11.478	8.889	4.940	4.588
garden (1297x840)	13.582	13.174	5.534	5.080
stump (1245x825)	10.225	8.409	4.779	4.555
room (1558x1038)	57.245	27.698	42.623	23.869
counter (1558x1038)	107.953	84.378	76.031	64.722
kitchen (1558x1038)	64.100	33.444	45.030	40.873
bonsai (1558x1038)	45.259	22.462	31.845	19.695
Synthetic Datasets (resolution)		Execution time (ms)		
Bulldozer (800x800)	21.465	15.878	7.544	5.953
Ficus (800x800)	142.950	24.305	237.401	46.480
Hotdog (800x800)	43.052	18.623	33.295	13.242
Drums (800x800)	113.925	61.438	11.174	5.576
Ship (800x800)	118.386	51.575	83.639	14.181
Materials (800x800)	203.372	92.228	33.939	15.002
Mic (800x800)	349.115	53.287	335.457	69.639
Chair (800x800)	21.617	12.810	11.082	6.430