

Floating Radiance Networks

Krzysztof Byrski^{1*†}, Rafał Tobiasz^{1,3*}, Grzegorz Wilczyński^{1,3}, Mikołaj Zieliński², Dawid Baran,
Dominik Belter², Jacek Tabor¹, Przemysław Spurek^{1,3}

¹ Jagiellonian University, Faculty of Mathematics and Computer Science

² Poznań University of Technology, Institute of Robotics and Machine Intelligence

³ IDEAS Research Institute

krzysiek.byrski@uj.edu.pl

Abstract

Recent advances in neural scene representations enable photorealistic novel-view synthesis, yet most methods remain tightly coupled to a single rendering paradigm, limiting their versatility and integration with conventional graphics workflows. We introduce Floating Radiance Networks (FlaRe), a neural scene representation combining explicit ray-traceable geometry with continuous neural radiance functions. A scene is represented by floating planar generalized Gaussian primitives, each carrying a compact latent descriptor of a local radiance field. A lightweight decoder shared across the scene maps this descriptor, local surface coordinates, and viewing direction to color and opacity. This formulation preserves the expressiveness of neural fields while providing an explicitly addressable structure that can be efficiently queried and manipulated. Hardware-accelerated primitive intersections enable interactive rendering and recursive ray-tracing, including reflections, refractions, transparency, and shadows. The same representation further supports primitive-level deformation, mesh extraction, and appearance stylization directly in its learned descriptor space. Experiments across standard reconstruction benchmarks demonstrate competitive rendering quality while using a compact set of primitives. Together, these results establish FlaRe as a versatile representation that brings high-fidelity neural rendering, ray-tracing, geometric manipulation, and appearance editing into a unified scene model. Source code is available online. Source code can be found at: <https://github.com/KByrski/FlaRe>

Introduction

Neural scene representations have transformed photorealistic 3D reconstruction and novel-view synthesis. Neural Radiance Fields (NeRFs) (Mildenhall et al. 2020) achieve high visual fidelity by modeling scenes as continuous volumetric functions, but dense ray marching and repeated network evaluations make rendering costly and complicate integration with conventional graphics pipelines. In contrast, 3D Gaussian Splatting (3DGS) (Kerbl et al. 2023) enables real-time rendering through explicit anisotropic primitives and rasterization. However, its finite per-primitive appearance coefficients do not define a continuous spatial radiance function over each primitive footprint. This design space exposes a persistent trade-off between the expressiveness of continuous neural fields and the efficiency, addressability, and graphics compatibility of explicit geometry. Recent methods combine neural features with explicit structures or support direct

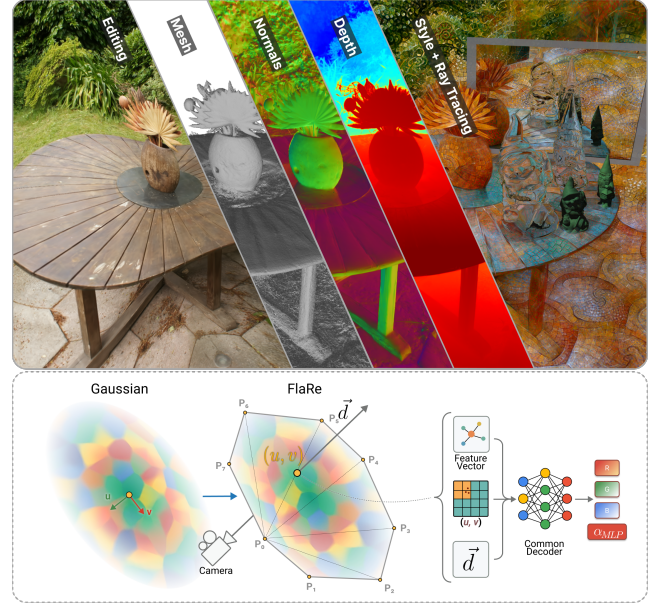


Figure 1: FlaRe represents a scene with floating planar primitives, each carrying a compact local radiance descriptor. At a ray-primitive intersection (u, v) , a shared decoder maps the descriptor, local coordinates, and viewing direction to color and opacity. Hardware-accelerated intersections replace volumetric sampling, so the capabilities above share one explicit, ray-traceable model.

ray queries (Govindarajan et al. 2025; Zieliński et al. 2026; Wilczyński et al. 2026; Govindarajan et al. 2026). Nevertheless, they are often tied to a particular rendering procedure, aggregate features across scene elements, or require task-specific extensions for editing, ray-tracing, or geometry extraction. Combining locally continuous radiance with independently addressable, ray-traceable geometry would provide a more direct bridge between neural rendering and conventional graphics.

We introduce Floating Radiance Networks (FlaRe), a neural scene representation built from floating planar generalized Gaussian primitives carrying local neural radiance functions (Figure 1). Each primitive stores a compact trainable descriptor evaluated by a lightweight decoder shared across

*These authors contributed equally.

†Corresponding author.

Method	Features	Local Neural Appearance	Ray Tracing	Secondary Rays	Geometry Editing	Mesh Extraction	Primitive Reduction
3DGS	✗	✗	✗	✗	✗	✗	✗
Scaffold-GS	●	✗	✗	✗	✗	✗	✗
2DGS	✗	✗	✗	✗	●	✓	✗
Nexels	✗	✓	✗	✗	✗	✗	✓
3DGRT	✗	✗	✓	✓	✗	✗	✗
IRIS	✓	●	✗	✗	✓	✗	✗
Power Foam	✗	✓	✓	●	✗	✗	✗
FlaRe (ours)	✓	✓	✓	✓	✓	✓	✓

Table 1: Capabilities demonstrated by the original formulations. A circle denotes partial or indirect support. Features denote learned latent descriptors stored with explicit scene elements.

the scene. Conditioned on the descriptor, local surface coordinates, and viewing direction, the decoder predicts spatially varying and view-dependent color and opacity. This avoids storing an independent network per primitive while retaining a continuous appearance function over its local support. The explicit primitive structure makes FlaRe directly compatible with hardware-accelerated ray-tracing. Generalized Gaussian primitives use shared canonical mesh proxies, enabling efficient collection of ordered ray-primitive intersections and front-to-back compositing. The representation supports arbitrary and recursive ray queries, including reflections, refractions, transparency, shadows, and composition with mesh geometry. Because primitives remain individually addressable, their geometry can be directly deformed or manipulated. With geometry-aware regularization, the planar representation also supports mesh extraction, while its descriptor space provides a suitable parameterization for appearance stylization and modeling under restricted primitive budgets.

Unlike NeRF-Casting, TraM-NeRF, and Mirror-NeRF (Verbin et al. 2024; Holland et al. 2024; Zeng et al. 2023), which introduce secondary rays within specialized volumetric rendering formulations, FlaRe makes the representation itself directly ray-queryable. Thus, rendering, recursive ray-tracing, geometry manipulation, mesh extraction, and appearance editing use the same underlying representation rather than separate task-specific scene models (as shown in Table 1). Experiments on standard benchmarks demonstrate strong novel-view synthesis quality at interactive rates with a compact set of primitives.

Our contributions are summarized as follows:

- We introduce Floating Radiance Networks, a neural scene representation in which explicit ray-traceable planar primitives carry continuous local radiance functions.
- We design a compact formulation combining per-primitive latent descriptors, a shared lightweight auto-decoder, collision-free multiresolution LUT-Encoding, generalized Gaussian kernels, and hardware-compatible mesh proxies.
- We demonstrate recursive ray-tracing, primitive deformation, mesh extraction, appearance stylization, and reconstruction under restricted primitive budgets while retain-



Figure 2: Recursive ray-tracing on FlaRe reconstructions. Seamless integration with conventional meshes allows for material-dependent reflections, refractions, and nested bounces. This hybrid rendering natively generalizes to the scene after appearance stylization (bottom-right).

ing strong quality and interactive performance.

Related Work

Neural Radiance Fields represent scenes as continuous functions evaluated and integrated along camera rays (Mildenhall et al. 2020), while later methods improve quality and efficiency through scene contraction, sparse grids, hash encodings, and tensor factorizations (Barron et al. 2022, 2023; Fridovich-Keil et al. 2022; Müller et al. 2022; Chen et al. 2022). In contrast, 3D Gaussian Splatting uses explicit anisotropic primitives with rasterized per-primitive appearance (Kerbl et al. 2023), and 2DGS introduces planar surfels for improved geometric consistency (Huang et al. 2024). Several approaches enrich explicit structures with learned appearance: DNMP attaches latent features to mesh elements (Lu et al. 2023), Scaffold-GS predicts Gaussian attributes from anchor features (Lu et al. 2024), Gaussian Billboards use spatially varying textures (Weiss and Bradley 2024), and EKS and IRIS condition neural radiance on Gaussian-carried or ray-intersection features (Zieliński et al. 2026; Wilczyński et al. 2026). Nexels similarly augments planar primitives with spatially varying neural appearance (Rong et al. 2025). Unlike these methods, FlaRe associates each primitive with a compact descriptor of a continuous local radiance function evaluated directly from local coordinates and viewing direction using one shared decoder.

Recent work also develops scene representations compatible with direct ray queries. 3DGRT and 3DGUT adapt Gaussian primitives to hardware-accelerated ray-tracing and secondary rays (Moenne-Loccoz et al. 2024; Wu et al. 2025), while RaySplats performs direct ray-Gaussian intersection (Byrski et al. 2025a). REDiSplats, MeshSplats, and LinPrim expose polygonal or polyhedral structures compatible with conventional graphics pipelines (Byrski et al. 2025b; Tobiasz et al. 2026; von Lützwow and Nießner 2025). Beyond Gaussian representations, Radiant Foam, Power Foam,

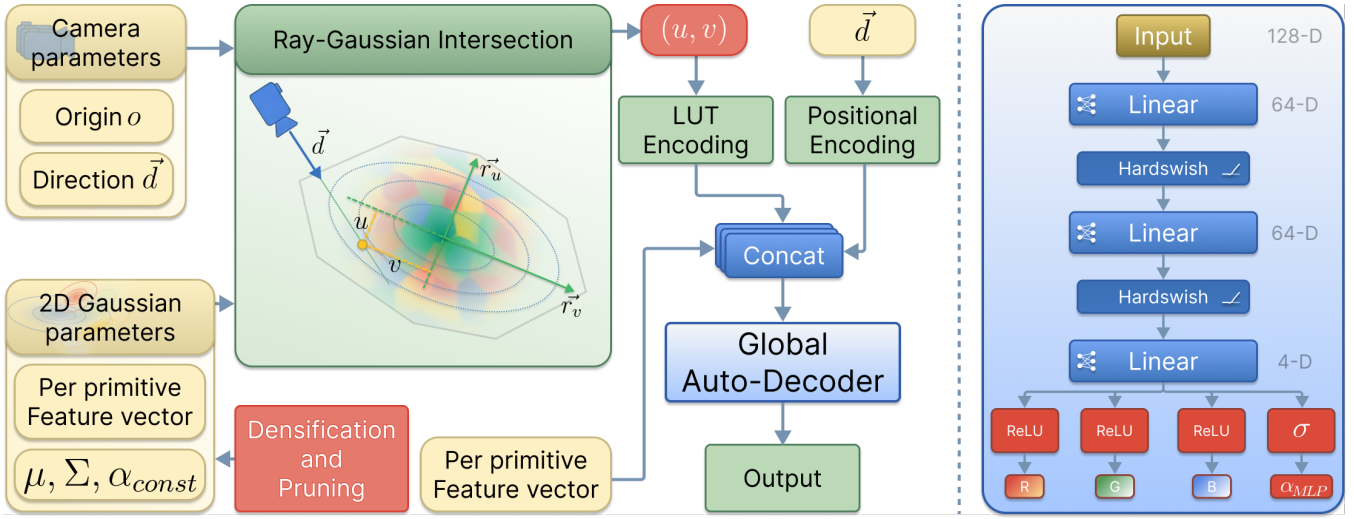


Figure 3: **Left:** Overview of the rendering pipeline. Every intersection along a camera ray contributes local coordinates (u, v) and a viewing direction d , which are encoded and concatenated with the per-primitive feature vector before decoding into color and opacity. Densification and pruning update the primitive set during optimization. **Right:** The auto-decoder architecture comprises two linear hidden layers each followed by a Hardswish activation, and a linear output layer, which branches into four channels processed by three ReLU and one σ activation functions, respectively.

SVRaster and Radiance Meshes employ structured cells or meshes for efficient differentiable ray-tracing and, in some cases, rasterization (Govindarajan et al. 2025, 2026; Sun et al. 2025; Mai et al. 2025). Explicit structure can additionally facilitate mesh extraction, deformation, and appearance manipulation, as demonstrated by 2DGS, SuGaR, Gaussian Frosting, GaMeS, and Gaussian stylization methods (Huang et al. 2024; Guédon and Lepetit 2024b,a; Waczyńska et al. 2024; Liu et al. 2024a; Howil et al. 2025). In contrast to task-specific extensions, FlaRe combines local neural appearance, direct ray-tracing, geometry editing, mesh extraction, and stylization within the same underlying primitive representation.

FlaRe: Floating Radiance Networks

Given posed RGB images, FlaRe optimizes a scene representation $\mathcal{S}$ composed of floating planar Gaussian primitives with learned local radiance descriptors. For each camera ray r , we intersect the ray with $\mathcal{S}$, decode color and opacity at the encountered primitives, and alpha-composite the resulting samples to obtain the rendered pixel c .

Gaussian Primitive Scene Representation FlaRe represents a scene using N anisotropic planar generalized Gaussian primitives, initialized with equal in-plane scales:

$$\mathcal{G} = \{(\mathcal{N}(\mu_i, \Sigma_i, \kappa_i), c_{\text{const},i}, \alpha_{\text{const},i}, z_i)\}_{i=1}^N.$$

Each primitive is parameterized by a trainable center $\mu_i \in \mathbb{R}^3$, singular covariance matrix $\Sigma_i = R_i S_i S_i^\top R_i^\top$, shape parameter $\kappa_i = 1 + \text{softplus}(\tilde{\kappa}_i)$, view-independent constant color parameter in the RGB space $c_{i,\text{const}} \in \mathbb{R}^3$, constant opacity $\alpha_{\text{const},i} = \sigma(\tilde{\alpha}_{\text{const},i})$, and a trainable latent descriptor z_i . Here, R_i is obtained from the trainable quaternion q_i , while $S_i = \text{diag}([\exp(\tilde{s}_i), 0])$ with $\tilde{s}_i \in \mathbb{R}^2$ con-

trols the primitive scale in its local tangent plane. The scalar $\tilde{\kappa}_i \in \mathbb{R}$ determines the generalized Gaussian shape, whereas $\tilde{\alpha}_{\text{const},i} \in \mathbb{R}$ controls its base opacity. Finally, $z_i \in \mathbb{R}^{96}$ encodes the local 2D neural radiance field associated with the primitive.

Isotropic Planar Generalized Gaussian Kernel Inspired by Nexels (Rong et al. 2025), FlaRe employs opacity modulation via an unnormalized generalized Gaussian Kernel in the underlying 2D representation primitives. This formulation enables more precise modeling of sharp boundaries and high-curvature surfaces, effectively mitigating approximation errors inherent in classical Gaussian kernels characterized by a fixed quadratic fall-off profile. We adopt a definition of the Isotropic Planar Generalized Gaussian Kernel, given by the following formula:

$$\mathcal{N}(\mu, \Sigma, \kappa)(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{P}\mathbf{S}^{-1}\mathbf{R}^\top(\mathbf{x} - \mu)\|_2^{2\kappa}}{2\kappa}\right),$$

where $\mathbf{P} = [\mathbf{I}_2 \ \mathbf{0}] \in \mathbb{R}^{2 \times 3}$ selects the two tangent-plane coordinates. Thus, for a point expressed in the primitive’s local frame as (u, v) , the kernel simplifies to

$$\mathcal{N}(\mu, \Sigma, \kappa)(u, v) = \exp\left(-\frac{(u^2 + v^2)^\kappa}{2\kappa}\right),$$

with final opacity $\alpha = \alpha_{\text{const}} \alpha_{\text{MLP}} \mathcal{N}(\mu, \Sigma, \kappa)(u, v)$, where α_{MLP} is predicted by the auto-decoder.

This approach allows us to utilize a single canonical polygon for each Gaussian, regardless of the κ parameter, since the kernel has circular isocontours in normalized canonical coordinates, independent of the shape parameter κ . In contrast, for the density function proposed by (Rong et al. 2025), the polygon’s geometry does not align with the isocontours,

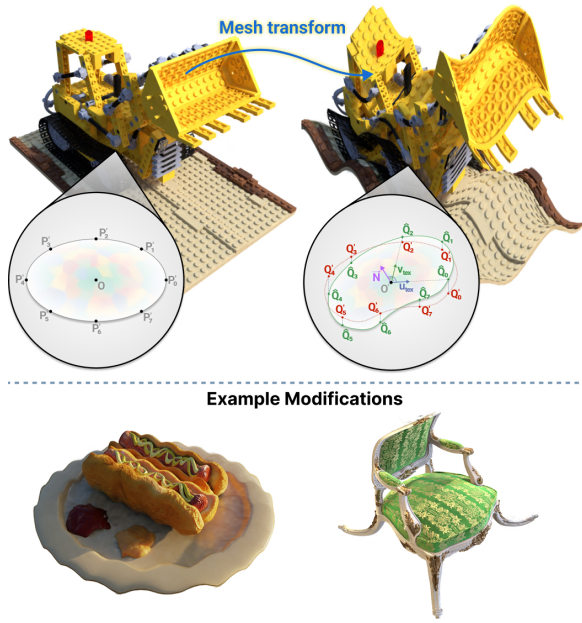


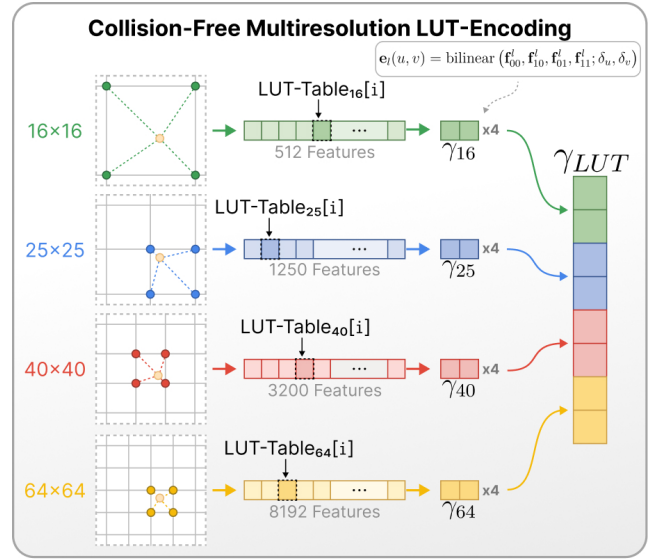
Figure 4: Primitive-level mesh modification: deforming the proxy meshes transfers the deformation to the underlying primitives, whose parameters are recovered directly from the deformed geometry without retraining.

preventing the density values from vanishing at the boundary and necessitating the computation of a unique proxy polygon for every individual value of the shape parameter.

Global auto-decoder For clarity, we omit the ray index superscript (r) throughout this subsection whenever the dependence on the current ray is unambiguous. In contrast to conventional Gaussian-based representations (e.g., 3DGS (Kerbl et al. 2023) or 3DGRT (Moenne-Loccoz et al. 2024)), which rely on static, per-primitive color and opacity attributes, typically optimized as trainable constants modulated by a spatial Gaussian kernel, FlaRe adopts a more flexible approach. Specifically, both the color and opacity of samples within each 2D Gaussian footprint are predicted by a dedicated 2D neural radiance field, uniquely assigned to each individual primitive and operating within its specific local coordinate frame, and encoded individually for each Gaussian primitive as a trainable latent descriptor vector. The color and opacity are then decoded using a single global auto-decoder MLP:

$$\Phi\left((u_i, v_i), \vec{d}, z_i; \mathcal{G}, \mathcal{F}, \Theta\right) = (c_{\text{MLP},i}, \alpha_{\text{MLP},i})$$

where $(c_{\text{MLP},i}, \alpha_{\text{MLP},i}) \in \mathbb{R}^3 \times \mathbb{R}$ denotes the color and opacity predicted by the MLP at the intersection point P_i of ray r and the i -th primitive encountered by the ray, given the direction vector $\vec{d}$ and per-primitive local 2D neural radiance field descriptor z_i with P_i having the coordinates: (u_i, v_i) within the primitive’s local reference frame. The network is conditioned on the set of all Gaussian primitives $\mathcal{G}$, the LUT-Table features $\mathcal{F}$ and MLP parameters Θ . Our global auto-decoder architecture is visible in Figure 3.



Learned Primitive Descriptor Field

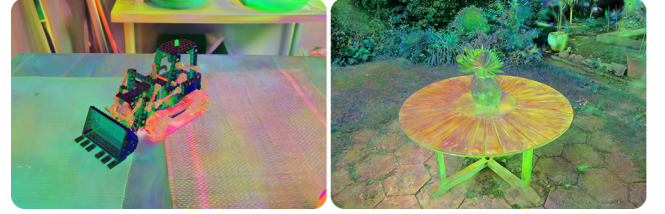


Figure 5: Top: Collision-free multiresolution LUT encoding. Local coordinates (u, v) are bilinearly interpolated on four compact 2D grids and concatenated into γ_{LUT} . Bottom: the resulting encoding visualized on reconstructed scenes.

LUT-Encoding Prior to entering the network, input data is processed through specific encoding schemes: the spatial coordinates (u_i, v_i) are mapped using lookup-table encoding (LUT-Encoding), while the view direction vector $\vec{d}$ is encoded via positional encoding. These encoded vectors are then concatenated with the individual per-primitive neural radiance field descriptor to form a latent vector, which serves as the input to our MLP (see Figure 3).

Our LUT-Encoding mechanism generates latent vectors functionally equivalent to the hash-based encodings proposed in InstantNGP (Müller et al. 2022). However, unlike their approach, which relies on high-dimensional hash-grid lookups, we leverage the fact that our floating primitives already carry a significant portion of the geometric representation. Consequently, we can reduce the encoding overhead by employing noticeably a smaller multiresolution structure. The compact size of the 2D grids enables a direct, collision-free one-to-one mapping between grid vertices and feature vectors, effectively replacing the hashing mechanism with a straightforward lookup table (LUT) (see Figure 15).

Mesh representation of a 2D Gaussian primitive Following (Moenne-Loccoz et al. 2024), we represent each 2D Gaussian by a triangulated stretched regular polygon that serves as a mesh proxy for hardware-accelerated ray-triangle

		Mip-NeRF360					Tanks and Temples			Deep Blending			
Method		SSIM ↑	PSNR ↑	LPIPS ↓	Train ↓	FPS ↑	MEM ↓	SSIM ↑	PSNR ↑	LPIPS ↓	SSIM ↑	PSNR ↑	LPIPS ↓
RF	Plenoxels	0.670	23.63	0.440	25min 45s	7	2.1GB	0.379	21.08	0.795	0.510	23.06	0.510
	INGP	0.725	26.43	-	5m 37s	11.7	13MB	0.723	21.72	0.330	0.797	23.62	0.423
	M-NeRF360	0.844	29.23	-	48h	0.06	8.6MB	0.759	22.22	0.257	0.901	29.40	0.245
GS	Scaffold-GS	0.848	28.84	0.220	24m 22s	102	156MB	0.853	23.96	0.177	0.906	30.21	0.254
	3DGS	0.870	28.69	0.220	41m 33s	134	734MB	0.841	23.14	0.183	0.903	29.41	0.243
	3DGRT	0.854	28.71	0.250	47m 49s	178	383MB	0.830	23.20	0.222	0.900	29.23	0.315
	2DGS	0.854	28.28	0.217	33m 53s	62	428MB	0.880	24.07	0.124	0.903	29.53	0.257
	Nexels	0.860	28.89	0.174	1h 6m	50	160MB	0.839	23.51	0.157	0.911	30.47	0.204
ENF	PowerFoam	0.840	28.91	0.210	-	174	-	0.833	22.53	0.177	0.884	28.59	0.285
	SVRaster	0.822	27.34	0.186	15m 24s	117	1.8GB	0.851	23.04	0.144	0.896	29.84	0.228
	RadianceMeshes	0.867	27.99	0.230	4h	294	2.4GB	0.854	23.21	0.207	0.904	29.58	0.353
	IRIS	0.820	28.54	0.163	43m 43s	0.32	9.6GB	0.843	22.74	0.226	0.896	29.83	0.323
	FlaRe (our)	0.836	28.93	0.219	11h 34m	16.53	207MB	0.804	22.84	0.221	0.896	29.90	0.264

Table 2: Quantitative comparison on Mip-NeRF360 (Barron et al. 2022), Tanks and Temples (Knapitsch et al. 2017), and Deep Blending (Hedman et al. 2018). Methods are grouped by scene representation: RF denotes radiance fields, GS denotes Gaussian-based methods, ENF denotes explicit neural fields.

intersection. To reduce memory usage, all primitives share a single canonical polygon and are instantiated through per-primitive affine transformations. For a canonical vertex

$$\mathbf{P}_j = [\cos(\frac{2\pi j}{k}), \sin(\frac{2\pi j}{k}), 0]^T,$$

the corresponding world-space proxy is obtained using the Gaussian center, rotation, and planar scale. We scale the proxy such that the Gaussian opacity reaches a predefined threshold $\alpha_{\min}$ at its boundary, yielding

$$\Delta s = \left(-2\kappa \ln \frac{\alpha_{\min}}{\alpha_{\text{const}}} \right)^{\frac{1}{2\kappa}}, \quad \alpha_{\min} \leq \alpha_{\text{const}}.$$

The resulting proxy therefore tightly bounds the effective support of the generalized Gaussian.

Mesh modification Given a deformed proxy polygon with world-space vertices $\mathbf{Q}_i$, we recover the corresponding 2D Gaussian parameters directly from the transformed geometry, without retraining (see Figure 4). Our procedure is designed to remain stable under non-rigid deformations that may break local coplanarity or alter the primitive orientation. We center the transformed proxy vertices, compute their covariance matrix, and use its SVD to recover the principal tangent directions of the deformed primitive. Since the proxy is planar, we obtain the surface normal as the cross product of these directions and construct the updated local frame from the resulting orthonormal basis. The Gaussian center is set to the transformed polygon centroid, while its planar scales are recovered from the extent of the projected vertices in this local frame, accounting for the clamping factor Δs .

For purely rigid or axis-aligned deformations, this reconstruction is sufficient. However, under shear, the original orthogonal texture coordinates (u, v) become non-orthogonal, causing a mismatch between the deformed geometry and the primitive’s local neural radiance field. We therefore additionally estimate a linear transformation $\mathbf{A}$ that maps the local

coordinates into the deformed texture space before they are passed to the MLP.

Color aggregation along the ray Following (Kopanas et al. 2021, 2022), we render each pixel by collecting the ordered intersections of its camera ray with the Gaussian proxy meshes and compositing the corresponding samples front-to-back. For ray r , the final color is

$$\mathbf{c} = \sum_{i=1}^{|\mathcal{G}^{(r)}|} \mathbf{c}_{\text{const},i} \mathbf{c}_{\text{MLP},i} \alpha_i \prod_{j < i} (1 - \alpha_j),$$

where the effective opacity of the i -th intersection is

$$\alpha_i = \alpha_{\text{const},i} \alpha_{\text{MLP},i} \mathcal{N}(u_i, v_i).$$

Thus, the learned color and opacity are modulated by the primitive’s generalized Gaussian kernel value before standard alpha compositing.

Loss function To stabilize geometry optimization before the local neural radiance fields become expressive, we introduce a warmup stage that smoothly transitions from a simplified base renderer to the full FlaRe model. The base renderer uses only view-independent per-primitive RGB color and opacity, while the full model additionally employs the learned neural radiance descriptors alongside MLP-predicted color and opacity. We optimize

$$\mathcal{L} = (1 - \lambda) \mathcal{L}_{\text{base}} + \lambda \mathcal{L}_{\text{FlaRe}} + \lambda_s \mathcal{L}_s,$$

where

$$\mathcal{L}_m = \lambda_{\text{RGB},m} \mathcal{L}_{\text{RGB},m}, m \in \{\text{base}, \text{FlaRe}\},$$

and $\lambda \in [0, 1]$ is linearly increased during the warmup.

$\mathcal{L}_{\text{RGB}}$ is the standard $\mathcal{L}_2$ photometric loss. Finally,

$$\mathcal{L}_s = \frac{1}{N} \sum_{i=1}^N \|\mathbf{s}_i\|_2,$$



Figure 6: Qualitative comparison on Mip-NeRF 360. Enlarged crops highlight view-dependent reflections and specularities, which FlaRe reconstructs more faithfully than the competing methods.

regularizes primitive scales and discourages excessively large Gaussians.

Additional implementation details, derivations, and an extended description of the FlaRe pipeline are provided in the Supplementary Material.

Experiments

We evaluate FlaRe in terms of novel-view synthesis, graphics capabilities, and representation compactness. In addition to standard reconstruction benchmarks, we study recursive ray-tracing, mesh extraction, appearance stylization, primitive reduction, and LUT-Encoding. Experimental details are provided in the Supplementary Material.

Novel-View Synthesis

Quantitative comparison Across radiance fields (RF), Gaussian-based methods (GS), and explicit neural fields (ENF), FlaRe achieves a favorable balance of novel-view synthesis quality, training cost, rendering speed, and memory, as summarized in Table 2. On Mip-NeRF 360, FlaRe achieves the second-highest PSNR overall, outperforming 3DGS, 3DGRT, and the remaining ENF baselines. On Deep Blending, FlaRe leads the ENF group in PSNR. Compared with other ray-queryable neural representations, it offers a stronger quality–efficiency trade-off: FlaRe surpasses IRIS while using substantially less memory and rendering interactively, and improves over Radiance Meshes on both benchmarks at roughly $12\times$ lower memory. Rasterized Gaussian methods remain faster, but FlaRe retains competitive reconstruction quality within a compact, explicitly ray-traceable representation.

Qualitative comparison Compared with 3DGS, 3DGRT, and Radiance Meshes, FlaRe more faithfully reconstructs reflections and specular highlights, as shown in Figure 6. Rather than assigning each primitive a fixed or low-order directional color, FlaRe predicts radiance at every ray-primitive intersection from the primitive descriptor, local coordinates, and viewing direction. This allows a single primitive to model continuous spatial and angular appearance

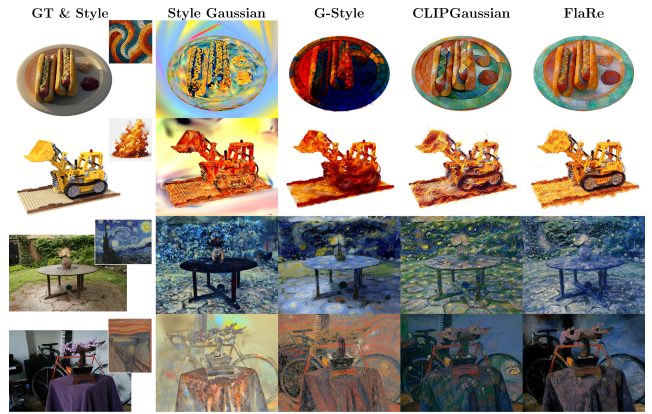


Figure 7: Image-guided 3D style transfer comparison. FlaRe reproduces the target style while better preserving scene structure and shading, showing that its learned primitive descriptors provide a directly optimizable appearance representation.

variation, producing sharper and more accurately positioned highlights.

The structure of the learned appearance space is further illustrated in Figure 15. We project the per-primitive descriptors onto three principal components and map them to RGB. Primitives with similar appearance form coherent regions, whereas distinct materials and patterns separate more clearly. This indicates that the descriptors capture meaningful local appearance information rather than serving merely as primitive identifiers.

Graphics Capabilities

Ray-Tracing One of our contributions is direct compatibility with recursive ray-tracing. Primary and secondary rays query the same learned primitives through hardware-accelerated intersections, without converting the scene to another representation or retraining it for a specific effect. This enables reflections, refractions, transparency, shadows, and hybrid composition with conventional mesh geometry within a single rendering pipeline. As shown in Figure 2, mesh objects with different BRDFs interact directly with the neural scene: metallic cars reflect the outdoor Garden environment, while glass, mirror, and metal spheres in Dr. Johnson exhibit material-dependent reflection and refraction. The Bonsai example further demonstrates multiple recursive interactions, where a refractive dragon and opposing mirrors produce nested reflections containing both mesh objects and reconstructed scene content. We additionally apply the same setup after appearance stylization (see Figure 1), showing that secondary rays continue to query the modified primitive descriptors without ray-tracing-specific adaptation. These results demonstrate that FlaRe supports general ray-based graphics effects within a single editable neural scene representation.

Style Transfer We evaluate whether FlaRe’s per-primitive descriptors form a controllable appearance space by adapting the image-guided objective of CLIPGaussian (Howil

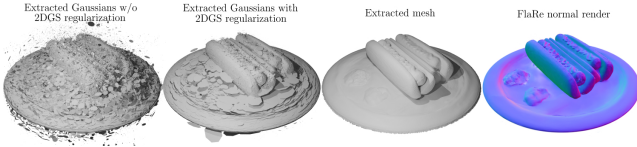


Figure 8: Mesh extraction with FlaRe. From left to right: proxy meshes of the learned primitives without and with the 2DGS regularizers, the TSDF-extracted triangle mesh, and the corresponding normal render produced by FlaRe.

et al. 2025) and comparing against StyleGaussian (Liu et al. 2024b) and G-Style (Kovács, Hermosilla, and Raidou 2024). Rather than directly optimizing Gaussian appearance parameters, FlaRe performs stylization in the descriptor space while fine-tuning the shared decoder. As shown in Figure 7, FlaRe transfers the reference palette and visual character while better preserving object structure, shading, and scene-specific color relationships. This suggests that the descriptor space provides a coherent appearance prior across primitives.

Mesh Extraction The explicit planar structure of FlaRe allows us to directly adopt the mesh-extraction pipeline of 2DGS (Huang et al. 2024). For mesh-oriented training, we additionally use its depth-distortion and normal-consistency regularizers, activated after 3k and 7k iterations, respectively. As shown in Figure 8, these losses substantially improve the geometric alignment of the primitives and enable TSDF-based surface reconstruction. We render depth maps from the training views, fuse them into a TSDF volume, and extract its zero level set as a triangle mesh. During depth rendering, we set $\alpha_{\text{MLP}} = 1$ so that the recovered geometry depends only on the explicit primitives rather than appearance-dependent opacity.

Representation Analysis

Primitive reduction. We evaluate how well FlaRe preserves reconstruction quality when the number of primitives is restricted, comparing against PUP-3DGS (Hanson et al. 2025) and LP-3DGS (Zhang et al. 2024) on *Lego* and *Bonsai*. As shown in Figure 9, FlaRe matches or surpasses both baselines across comparable primitive budgets, with the largest advantage under aggressive reduction on *Bonsai*. This indicates that its local neural radiance descriptors retain strong appearance capacity even with substantially fewer primitives, while maintaining stable reconstruction quality. Full protocols and results are provided in the Supplementary Material.

LUT-Encoding. We isolate the encoding component by training a small coordinate-based network to reconstruct five continuous RGB-valued 2D signals from their spatial coordinates (u, v) . The signals include both smoothly varying patterns and localized high-frequency details, allowing us to compare reconstruction fidelity and lookup efficiency. We evaluate our collision-free LUT against hash-grid encodings with matched resolutions or parameter budgets. As shown in Table 5, LUT-Encoding achieves the highest mean PSNR and the fastest forward evaluation. Its advantage over increasingly constrained hash tables also grows with the collision rate, in-

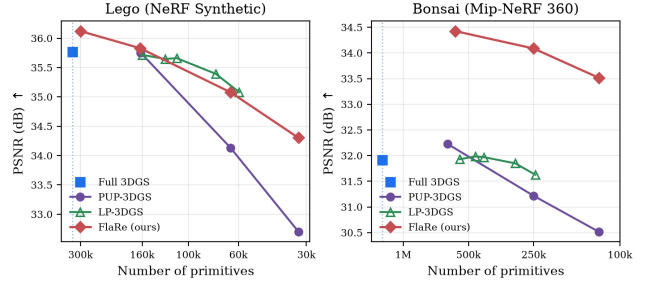


Figure 9: Compression study: test PSNR versus number of primitives on *Lego* and *Bonsai*. PUP-3DGS (Hanson et al. 2025) prunes a densified 3DGS model to fixed remaining ratios; LP-3DGS (Zhang et al. 2024) varies the mask sparsity weight λ_{mask} ; FlaRe is trained from scratch under matched primitive budgets. Vertical dotted lines mark the full densified 3DGS counts.

Encoding	Params (K)	Collisions (%)	Mean PSNR (dB) $\uparrow$	Time (ms) $\downarrow$	
				Forward	Train step
LUT (ours)	13.15	0.0	42.93 ± 0.61	1.123	3.917
Hash ($T=2048$)	9.06	31.1	41.06 ± 0.34	1.208	4.018
Hash ($T=1024$)	5.86	56.6	38.92 ± 0.31	1.295	4.130
Hash ($T=512$)	3.58	74.3	33.98 ± 0.26	1.379	4.250
Hash (equal budget)	13.15	91.6	41.48 ± 0.27	1.293	4.105

Table 3: Controlled coordinate-based reconstruction of five continuous RGB-valued 2D signals. Mean PSNR is averaged over the signals and random seeds. All encodings use the same training samples, bilinear interpolation, feature dimensionality, decoder, and optimization protocol. “Train step” includes both forward and backward passes.

dicating that direct lookup is better suited to the compact local coordinate domains used by FlaRe.

Conclusions

We introduced Floating Radiance Networks (FlaRe), a neural scene representation that combines explicit ray-traceable planar primitives with continuous local neural radiance functions. By coupling compact per-primitive descriptors with a shared auto-decoder, collision-free LUT-Encoding, and hardware-compatible proxy geometry, FlaRe retains the expressiveness of neural fields while exposing a directly queryable and manipulable scene structure. Our experiments demonstrate competitive reconstruction quality together with interactive rendering, recursive ray-tracing, descriptor-space style transfer, mesh extraction, and favorable quality under aggressive primitive reduction. These results show that neural scene representations need not be specialized for a single rendering paradigm, and position FlaRe as a step toward unified representations that can support both high-quality neural rendering and conventional graphics operations within the same scene model.

Limitations. The finite set of planar primitives can limit far-field geometric accuracy in large, unbounded outdoor scenes, while training and interactive rendering require a

modern GPU with hardware-accelerated ray-tracing.

Societal Impact As with other photorealistic 3D reconstruction and editing methods, FlaRe can support beneficial applications in visualization and content creation but may also facilitate the creation or manipulation of realistic synthetic visual content.

Acknowledgments

M. Zieliński and D. Belter were supported by the National Science Centre, Poland, under research project no UMO-2023/51/B/ST6/01646. The scientific work was carried out using the infrastructure of the Poznan Supercomputing and Networking Center. The work of P. Spurek was supported by the National Centre of Science (Poland) Grant No. 2023/50/E/ST6/00068.

References

- Barron, J. T.; Mildenhall, B.; Tancik, M.; Hedman, P.; Martin-Brualla, R.; and Srinivasan, P. P. 2021. Mip-NeRF: A Multiscale Representation for Anti-Aliasing Neural Radiance Fields. *ICCV*.
- Barron, J. T.; Mildenhall, B.; Verbin, D.; Srinivasan, P. P.; and Hedman, P. 2022. Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields. *CVPR*.
- Barron, J. T.; Mildenhall, B.; Verbin, D.; Srinivasan, P. P.; and Hedman, P. 2023. Zip-NeRF: Anti-Aliased Grid-Based Neural Radiance Fields. *ICCV*.
- Byrski, K.; Mazur, M.; Tabor, J.; Dziarmaga, T.; Kądziołka, M.; Baran, D.; and Spurek, P. 2025a. Raysplats: Ray tracing based gaussian splatting. *arXiv preprint arXiv:2501.19196*.
- Byrski, K.; Wilczyński, G.; Smolak-Dyżewska, W.; Borycki, P.; Baran, D.; Tadeja, S.; and Spurek, P. 2025b. REdiSplats: Ray Tracing for Editable Gaussian Splatting.
- Chen, A.; Xu, Z.; Geiger, A.; Yu, J.; and Su, H. 2022. TensorRF: Tensorial Radiance Fields. In *European Conference on Computer Vision (ECCV)*.
- Chen, Y.; Chen, Z.; Zhang, C.; Wang, F.; Yang, X.; Wang, Y.; Cai, Z.; Yang, L.; Liu, H.; and Lin, G. 2024. GaussianEditor: Swift and Controllable 3D Editing with Gaussian Splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 21476–21485.
- Fridovich-Keil, S.; Yu, A.; Tancik, M.; Chen, Q.; Recht, B.; and Kanazawa, A. 2022. Plenoxels: Radiance Fields Without Neural Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 5501–5510.
- Govindarajan, S.; Rebain, D.; Verbin, D.; Yi, K. M.; Prabhu, A.; and Tagliasacchi, A. 2026. Power Foam: Unifying Real-Time Differentiable Ray Tracing and Rasterization. *arXiv*.
- Govindarajan, S.; Rebain, D.; Yi, K. M.; and Tagliasacchi, A. 2025. Radiant Foam: Real-Time Differentiable Ray Tracing. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 4135–4145.
- Guédon, A.; and Lepetit, V. 2024a. Gaussian Frosting: Editable Complex Radiance Fields with Real-Time Rendering. *ECCV*.
- Guédon, A.; and Lepetit, V. 2024b. Sugar: Surface-aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 5354–5363.
- Hanson, A.; Tu, A.; Singla, V.; Jayawardhana, M.; Zwicker, M.; and Goldstein, T. 2025. Pup 3d-gs: Principled uncertainty pruning for 3d gaussian splatting. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, 5949–5958.
- Hedman, P.; Philip, J.; Price, T.; Frahm, J.-M.; Drettakis, G.; and Brostow, G. 2018. Deep blending for free-viewpoint image-based rendering. *ACM Transactions on Graphics (ToG)*, 37(6): 1–15.
- Holland, L. V.; Bliersbach, R.; Müller, J. U.; Stotko, P.; and Klein, R. 2024. TraM-NeRF: Tracing Mirror and Near-Perfect Specular Reflections Through Neural Radiance Fields. *Computer Graphics Forum*, e15163.
- Howil, K.; Waczynska, J.; Borycki, P.; Dziarmaga, T.; Mazur, M.; and Spurek, P. 2025. CLIPGaussian: Universal and Multimodal Style Transfer Based on Gaussian Splatting. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.
- Huang, B.; Yu, Z.; Chen, A.; Geiger, A.; and Gao, S. 2024. 2D Gaussian Splatting for Geometrically Accurate Radiance Fields. In *SIGGRAPH 2024 Conference Papers*. Association for Computing Machinery.
- Kerbl, B.; Kopanas, G.; Leimkühler, T.; and Drettakis, G. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics*, 42(4).
- Knapitsch, A.; Park, J.; Zhou, Q.-Y.; and Koltun, V. 2017. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics (ToG)*, 36(4): 1–13.
- Kopanas, G.; Leimkühler, T.; Rainer, G.; Jambon, C.; and Drettakis, G. 2022. Neural Point Catacaustics for Novel-View Synthesis of Reflections. *ACM Transactions on Graphics*, 41(6): Article–201.
- Kopanas, G.; Philip, J.; Leimkühler, T.; and Drettakis, G. 2021. Point-Based Neural Rendering with Per-View Optimization. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering)*, 40(4).
- Kovács, Á. S.; Hermosilla, P.; and Raidou, R. G. 2024. G-Style: Stylized Gaussian Splatting. In *Computer Graphics Forum*, volume 43, e15259. Wiley Online Library.
- Liu, K.; Zhan, F.; Xu, M.; Theobalt, C.; Shao, L.; and Lu, S. 2024a. StyleGaussian: Instant 3D Style Transfer with Gaussian Splatting. *arXiv preprint arXiv:2403.07807*.
- Liu, K.; Zhan, F.; Xu, M.; Theobalt, C.; Shao, L.; and Lu, S. 2024b. Stylegaussian: Instant 3d style transfer with gaussian splatting. In *SIGGRAPH Asia 2024 Technical Communications*, 1–4.
- Lu, F.; Xu, Y.; Chen, G.; Li, H.; Lin, K.-Y.; and Jiang, C. 2023. Urban Radiance Field Representation with Deformable Neural Mesh Primitives. *ICCV*.

- Lu, T.; Yu, M.; Xu, L.; Xiangli, Y.; Wang, L.; Lin, D.; and Dai, B. 2024. Scaffold-gs: Structured 3d gaussians for view-adaptive rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 20654–20664.
- Mai, A.; Hedstrom, T.; Kopanas, G.; Kontkanen, J.; Kuester, F.; and Barron, J. T. 2025. Radiance Meshes for Volumetric Reconstruction. *arXiv:2512.04076*.
- Mildenhall, B.; Srinivasan, P. P.; Tancik, M.; Barron, J. T.; Ramamoorthi, R.; and Ng, R. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *ECCV*.
- Moenne-Loccoz, N.; Mirzaei, A.; Perel, O.; de Lutio, R.; Esturo, J. M.; State, G.; Fidler, S.; Sharp, N.; and Gojcic, Z. 2024. 3D Gaussian Ray Tracing: Fast Tracing of Particle Scenes. *ACM Transactions on Graphics and SIGGRAPH Asia*.
- Müller, T.; Evans, A.; Schied, C.; and Keller, A. 2022. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM Trans. Graph.*, 41(4): 102:1–102:15.
- Rong, V.; Held, J.; Chu, V.; Rebain, D.; Van Droogenbroeck, M.; Kutulakos, K. N.; Tagliasacchi, A.; and Lindell, D. B. 2025. Nexels: Neurally-textured surfels for real-time novel view synthesis with sparse geometries. *arXiv preprint arXiv:2512.13796*.
- Sun, C.; Choe, J.; Loop, C.; Ma, W.-C.; and Wang, Y.-C. F. 2025. Sparse voxels rasterization: Real-time high-fidelity radiance field rendering. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 16187–16196.
- Thies, J.; Zollhöfer, M.; and Nießner, M. 2019. Deferred neural rendering: image synthesis using neural textures. *ACM Trans. Graph.*, 38(4).
- Tobiasz, R.; Wilczyński, G.; Mazur, M.; Tadeja, S.; Smolak-Dyżewska, W.; and Spurek, P. 2026. MeshSplats: Mesh-Based Rendering with Gaussian Splatting Initialization. In Neumann, P.; Puma, M. J.; Lees, M. H.; Groen, D.; Dongarra, J. J.; and Sloot, P. M. A., eds., *Computational Science – ICCS 2026*, 386–394. Cham: Springer Nature Switzerland. ISBN 978-3-032-29924-6.
- Verbin, D.; Srinivasan, P. P.; Hedman, P.; Mildenhall, B.; Atal, B.; Szeliski, R.; and Barron, J. T. 2024. NeRF-Casting: Improved View-Dependent Appearance with Consistent Reflections. In *SIGGRAPH Asia 2024 Conference Papers*, SA '24. New York, NY, USA: Association for Computing Machinery. ISBN 9798400711312.
- von Lützwow, N.; and Nießner, M. 2025. LinPrim: Linear Primitives for Differentiable Volumetric Rendering. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.
- Waczyńska, J.; Borycki, P.; Tadeja, S.; Tabor, J.; and Spurek, P. 2024. Games: Mesh-based adapting and modification of gaussian splatting. *arXiv preprint arXiv:2402.01459*.
- Weiss, S.; and Bradley, D. 2024. Gaussian billboards: Expressive 2d gaussian splatting with textures. *arXiv preprint arXiv:2412.12734*.
- Wilczyński, G.; Zieliński, M.; Byrski, K.; Waczyńska, J.; Belter, D.; and Spurek, P. 2026. IRIS: Intersection-aware Ray-based Implicit Editable Scenes.
- Wu, Q.; Martinez Esturo, J.; Mirzaei, A.; Moenne-Loccoz, N.; and Gojcic, Z. 2025. 3DGUT: Enabling Distorted Cameras and Secondary Rays in Gaussian Splatting. *Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Ye, M.; Danelljan, M.; Yu, F.; and Ke, L. 2024. Gaussian Grouping: Segment and Edit Anything in 3D Scenes. In *ECCV*.
- Zeng, J.; Bao, C.; Chen, R.; Dong, Z.; Zhang, G.; Bao, H.; and Cui, Z. 2023. Mirror-NeRF: Learning Neural Radiance Fields for Mirrors with Whitted-Style Ray Tracing. In *Proceedings of the 31st ACM International Conference on Multimedia*, 4606–4615.
- Zhang, Z.; Song, T.; Lee, Y.; Yang, L.; Peng, C.; Chellappa, R.; and Fan, D. 2024. Lp-3dgs: Learning to prune 3d gaussian splatting. *Advances in Neural Information Processing Systems*, 37: 122434–122457.
- Zieliński, M.; Byrski, K.; Szczepanik, T.; Belter, D.; and Spurek, P. 2026. Affine-Equivariant Kernel Space Encoding for NeRF Editing. *arXiv:2508.02831*.

Supplementary Materials

Floating Radiance Networks

This supplementary material provides additional technical details, experimental results, and implementation information for Floating Radiance Networks (FlaRe). We first extend the discussion of related work and report supplementary evaluations of primitive reduction, qualitative reconstruction quality, and LUT-Encoding. We then present a more detailed description of the FlaRe representation, including its planar generalized Gaussian primitives, shared auto-decoder, collision-free multiresolution LUT-Encoding, polygonal mesh proxies, rendering formulation, optimization procedure, and primitive-level deformation. Finally, we provide reproducibility details and additional analyses that complement the results presented in the main paper.

Extended Related Works

Radiance fields and learned appearance on explicit primitives. Neural Radiance Fields (NeRFs) represent scenes as continuous functions of position and viewing direction and synthesize novel views through differentiable volume rendering (Mildenhall et al. 2020). Subsequent work improved anti-aliasing, unbounded-scene reconstruction, and rendering efficiency through integrated positional encodings, scene contraction, sparse grids, multiresolution hash encodings, and tensor factorizations (Barron et al. 2021, 2022, 2023; Fridovich-Keil et al. 2022; Müller et al. 2022; Chen et al. 2022). Despite these advances, NeRF-style methods still reconstruct images by evaluating and integrating samples along rays. In contrast, 3D Gaussian Splatting (3DGS) (Kerbl et al. 2023) achieves real-time rendering using explicit anisotropic primitives and visibility-aware rasterization, while 2DGS (Huang et al. 2024) replaces volumetric Gaussians with planar primitives to improve geometric consistency. Several methods further combine explicit geometry with learned appearance. Deferred Neural Rendering (Thies, Zollhöfer, and Nießner 2019) associates conventional geometry with neural textures, DNMP (Lu et al. 2023) attaches latent features to local mesh elements, and Scaffold-GS (Lu et al. 2024) predicts view-adaptive Gaussian attributes from anchor features. Gaussian Billboards (Weiss and Bradley 2024) introduce spatially varying textures over planar Gaussians, whereas EKS (Zieliński et al. 2026) uses transformation-aware Gaussian feature carriers to condition an editable neural radiance field. IRIS (Wilczyński et al. 2026) instead aggregates features at ordered ray-primitive intersections. These approaches demonstrate the benefits of learned features on explicit structures, but are not designed around continuous local radiance functions evaluated directly on independently ray-traceable primitives. FlaRe assigns each floating planar primitive a compact latent descriptor and decodes spatially varying, view-dependent color and opacity at the corresponding ray-primitive intersection using one shared network.

Ray-queryable neural scene representations. A growing body of work adapts Gaussian and explicit radiance representations to arbitrary and secondary rays. 3D Gaussian Ray Tracing (3DGRT) (Moenne-Loccoz et al. 2024) encloses Gaussian particles in proxy geometry and uses hardware-accelerated ray intersections, while 3DGUT (Wu et al. 2025) extends Gaussian rendering to distorted cameras and secondary-ray applications. RaySplats (Byrski et al. 2025a) directly intersects rays with Gaussian confidence ellipses, and REdiSplats (Byrski et al. 2025b) combines polygonal Gaussian proxies with ray tracing and editable scene structure. MeshSplats (Tobiasz et al. 2026) converts a pretrained Gaussian representation into mesh faces, whereas LinPrim (von Lützw and Nießner 2025) directly optimizes polyhedral primitives compatible with mesh-based rendering tools. Beyond Gaussian-derived methods, Radiant Foam (Govindarajan et al. 2025), Power Foam (Govindarajan et al. 2026), and Radiance Meshes (Mai et al. 2025) introduce structured volumetric or mesh-based representations for efficient differentiable ray tracing and, in some cases, rasterization. These methods primarily focus on efficient traversal, graphics-compatible geometry, or conversion from existing representations. In contrast, FlaRe makes a shared decoder of continuous local radiance a fundamental component of the ray-traceable representation: every primitive independently carries a compact descriptor that is evaluated using its local coordinates and the ray direction.

Editable and graphics-compatible neural representations. Explicit scene structure is also valuable for downstream graphics operations. SuGaR (Guédon and Lepetit 2024b), Gaussian Frosting (Guédon and Lepetit 2024a), and 2DGS (Huang et al. 2024) introduce surface-aware Gaussian formulations that facilitate mesh extraction. GaMeS (Waczyńska et al. 2024) binds Gaussians to mesh faces to support deformation and animation, while REdiSplats (Byrski et al. 2025b) exposes editable ray-traceable proxy geometry. GaussianEditor (Chen et al. 2024) and Gaussian Grouping (Ye et al. 2024) enable local manipulation through generative guidance or semantic grouping. Appearance-oriented methods such as StyleGaussian (Liu et al. 2024a) and CLIPGaussian (Howil et al. 2025) perform image- or text-guided stylization by optimizing Gaussian appearance and geometry. These capabilities are commonly introduced through specialized representations or task-specific extensions. In FlaRe, explicit geometry, learned local appearance, and direct ray queries are properties of the same scene model, allowing primitive-level editing, deformation, mesh extraction, recursive ray tracing, and descriptor-space stylization without replacing the underlying representation.

Extended Results

Primitive Reduction

We expand the primitive-reduction study from the main paper by comparing FlaRe against two recent primitive-reduction baselines for 3DGS on *Lego* (NeRF Synthetic) and *Bonsai* (Mip-NeRF 360, indoor, half-resolution): PUP-3DGS (Hanson et al. 2025), which post-hoc prunes a pretrained densified model using a Fisher sensitivity score followed by short fine-tuning, and LP-3DGS (Zhang et al. 2024), which learns a differentiable pruning mask jointly with densification.

Protocol. For PUP-3DGS we first train a full 3DGS model for 30k iterations and then prune to approximately 50%, 20%, and 10% of the remaining Gaussians (PUP 50, 80, and cascaded 80→50), following the authors’ prune-refine recipe. For LP-3DGS we sweep the mask sparsity weight $\lambda_{\text{mask}} \in \{10^{-2}, 5 \cdot 10^{-3}, 10^{-3}, 5 \cdot 10^{-4}, 10^{-4}\}$, which induces a continuum of operating points rather than fixed target counts. For FlaRe we train from scratch for 30k iterations under a hard upper bound on the primitive count, matched to the PUP remaining counts on each scene (and a high reference cap of 300k on *Lego* / 2M on *Bonsai*). Mid-training evaluation is disabled so that reported optimization time reflects training only; metrics are measured once from the final checkpoint.

Results. The quantitative comparison in Table 4 shows that FlaRe consistently provides the best PSNR-primitive-count trade-off on *Bonsai*, with the largest advantage at aggressive budgets. Even with a loose upper bound, its primitive count saturates well below the imposed cap while its reconstruction quality remains above that of the full 3DGS model. On *Lego*, the gains are smaller but remain consistent at low primitive counts; at moderate budgets, FlaRe matches LP-3DGS and outperforms PUP-3DGS. Near the full model size, all three methods approach the quality of densified 3DGS, with FlaRe slightly exceeding the full-model baseline. LP-3DGS preserves quality better than PUP-3DGS at moderate reduction levels, but its learned mask does not reach the most aggressive counts attained by PUP-3DGS and by our budgeted runs. Overall, the shared neural decoder of FlaRe provides a more compact appearance model per primitive, so a given reconstruction fidelity requires substantially fewer primitives than rasterized 3DGS under either post-hoc or learned pruning.

Scene	Method	#Prim.	PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$
Lego	Full 3DGS	324,826	35.77	0.983	0.016
		32,482	32.70	0.969	0.039
	PUP-3DGS	64,965	34.13	0.977	0.024
		162,413	35.75	0.983	0.016
	LP-3DGS	59,531	35.08	0.981	0.019
		159,430	35.72	0.983	0.016
	FlaRe	32,548	34.30	0.973	0.025
		64,967	35.08	0.977	0.020
		163,179	35.83	0.980	0.017
		300,328	36.12	0.981	0.015
Bonsai	Full 3DGS	1,250,561	31.91	0.940	0.206
		125,056	30.52	0.919	0.251
	PUP-3DGS	250,112	31.22	0.931	0.226
		625,280	32.23	0.941	0.205
	LP-3DGS	245,755	31.63	0.935	0.219
		548,477	31.93	0.939	0.210
	FlaRe	125,057	33.51	0.929	0.237
		250,128	34.09	0.937	0.223
		575,455	34.43	0.942	0.212

Table 4: Quantitative primitive-reduction comparison on *Lego* and *Bonsai*. PUP-3DGS rows correspond to $\approx 10\%$ / 20% / 50% remaining Gaussians after prune-refine. LP-3DGS rows are representative sparsity weights ($\lambda_{\text{mask}} = 10^{-2}$ and 10^{-4}); the complete PSNR-versus-count sweep is shown in the primitive-reduction figure in the main paper. FlaRe rows report runs with hard primitive-count caps selected to match the PUP operating points, together with high-cap reference runs (300k on *Lego* and 2M on *Bonsai*).

Qualitative Results

Figure 10 shows cleaner walls and ceilings with fewer streaks and splat-shaped artifacts. Here, continuous prediction over each primitive footprint reduces piecewise color discontinuities, while the shared decoder gives all primitives a common appearance model. Together with depth-distortion and scale regularization, which discourage inconsistent depth contributions and oversized primitives, this yields a more coherent reconstruction of broad planar surfaces.

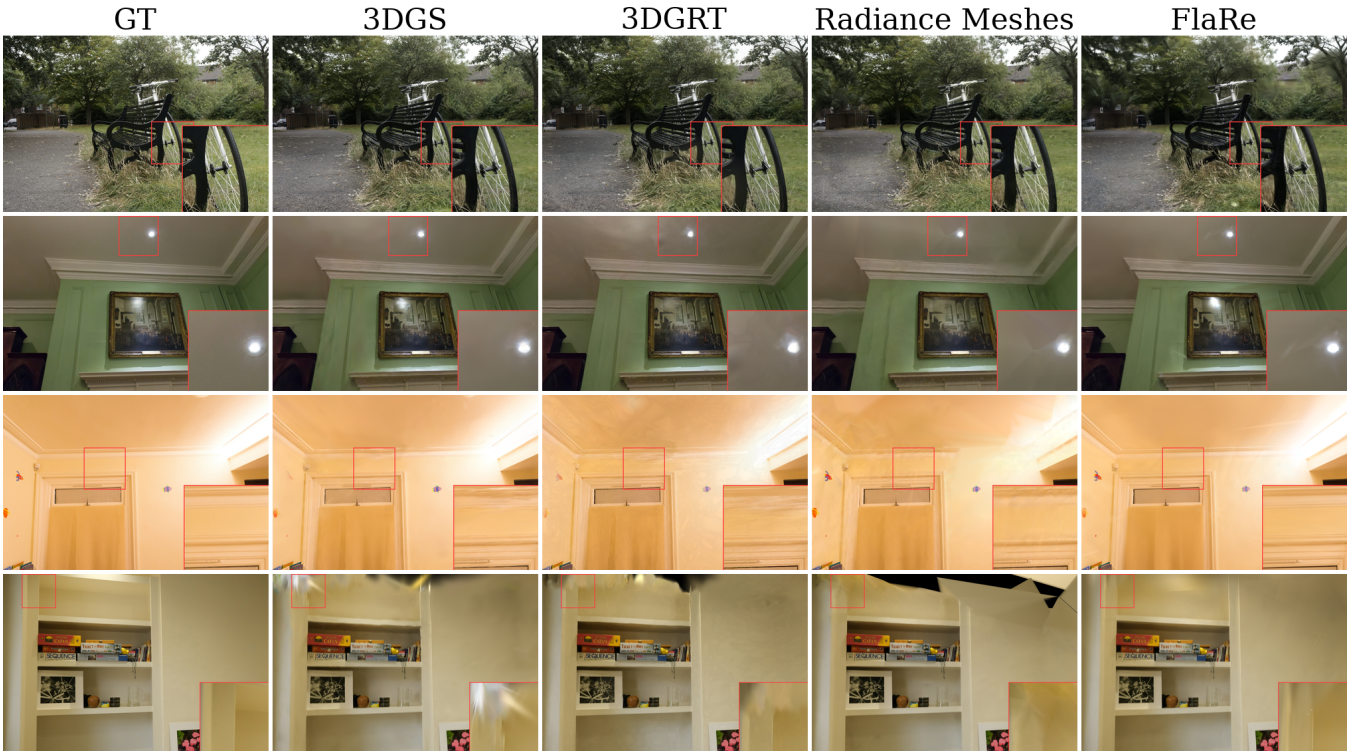


Figure 10: Qualitative comparison on scenes from Mip-NeRF 360 and Deep Blending. FlaRe yields cleaner reconstructions of broad, weakly textured surfaces such as walls and ceilings, with fewer visible rendering artifacts, while preserving fine-scale structures and high-frequency appearance details, as highlighted by the bicycle wheel and its spokes.

LUT-Encoding

Controlled 2D encoding experiment. We isolate the spatial encoding from rendering and geometry by fitting five continuous RGB functions on $[0, 1]^2$: smooth, multiscale Fourier, spatially varying chirp, localized high-frequency, and discontinuous checkerboard signals. At every iteration, we sample 16,384 continuous coordinates uniformly and optimize an identical 8–64–64–3 ReLU MLP for 5,000 iterations using Adam ($\text{lr} = 10^{-3}$, $\beta = (0.9, 0.99)$). All encodings use four bilinearly interpolated levels with two features per level, identical initialization and training samples, and are evaluated on a fixed 512^2 grid over five seeds. We report mean PSNR and encoding-only forward and backward time for batches of 262,144 coordinates. Exact target definitions and configurations are provided in our released evaluation script.

Encoding	Enc. params (K)	Coll. (%)	PSNR (dB) $\uparrow$					Mean $\uparrow$	Time (ms) $\downarrow$	
			Smooth	Multi.	Chirp	Local	Checker		Fwd.	Fwd.+Bwd.
LUT (ours)	13.15	0.0	63.36 ± 3.07	53.24 ± 0.35	27.67 ± 0.14	43.05 ± 0.38	27.33 ± 0.13	42.93 ± 0.61	1.123	3.917
Hash ($T=2048$)	9.06	31.1	<u>61.00</u> ± 1.92	<u>51.33</u> ± 0.13	<u>23.70</u> ± 0.12	<u>42.67</u> ± 1.18	<u>26.62</u> ± 0.16	<u>41.06</u> ± 0.34	<u>1.208</u>	<u>4.018</u>
Hash ($T=1024$)	5.86	56.6	59.25 ± 1.21	48.72 ± 0.27	19.11 ± 0.30	42.82 ± 0.14	24.68 ± 0.15	38.92 ± 0.31	1.295	4.130
Hash ($T=512$)	3.58	74.3	48.78 ± 0.46	45.14 ± 0.35	16.12 ± 0.22	39.51 ± 0.65	20.36 ± 0.38	33.98 ± 0.26	1.379	4.250
Hash (equal budget)	13.15	91.6	59.12 ± 0.62	49.47 ± 0.48	20.05 ± 0.08	51.72 ± 1.28	<u>27.06</u> ± 0.21	<u>41.48</u> ± 0.27	1.293	4.105

Table 5: Controlled 2D encoding comparison. All methods use identical bilinear interpolation, feature dimensionality, decoder, training samples, and optimization settings. Mean $\pm$ standard deviation over 5 seeds. Collision rate is measured over all virtual grid vertices. Best and second-best values are shown in **bold** and underline, respectively.

Baselines. Our LUT uses collision-free grids of resolutions $[16, 25, 40, 64]$, containing 13,154 trainable scalars. We compare it with hash grids using the same virtual resolutions and per-level capacities $T \in \{2048, 1024, 512\}$, resulting in 9,058, 5,858, and 3,584 parameters, respectively. Dense levels are addressed directly, whereas levels exceeding T use Instant-NGP-style hashing. We additionally construct an equal-budget hash grid with resolutions $[16, 40, 101, 256]$ and $T = 2360$. It contains 13,152 parameters, approximately matching the LUT, but exchanges collision-free storage for higher virtual resolution.

Results. LUT-Encoding achieves the highest average PSNR (42.93 dB) and the lowest forward time (1.123 ms). For the fixed-resolution hash grids, increasing collision rates from 31.1% to 74.3% reduces average PSNR from 41.06 to 33.98 dB, while hashing adds 7.5–22.8% forward overhead. The equal-budget hash grid reaches 41.48 dB, remaining 1.45 dB below LUT on average. LUT performs best on smooth, multiscale, chirp, and discontinuous signals because their domains are densely sampled: one-to-one addressing preserves independent vertex features, whereas collisions couple unrelated locations and their gradients. The only exception is localized detail, where the equal-budget hash grid obtains 51.72 dB compared with 43.05 dB for LUT. This target contains a localized 34-cycle component, exceeding the approximately 31.5-cycle Nyquist limit of the LUT’s 64-vertex level; the hash grid’s virtual resolution of 256 resolves this sparse detail, while the collision cost remains spatially limited.

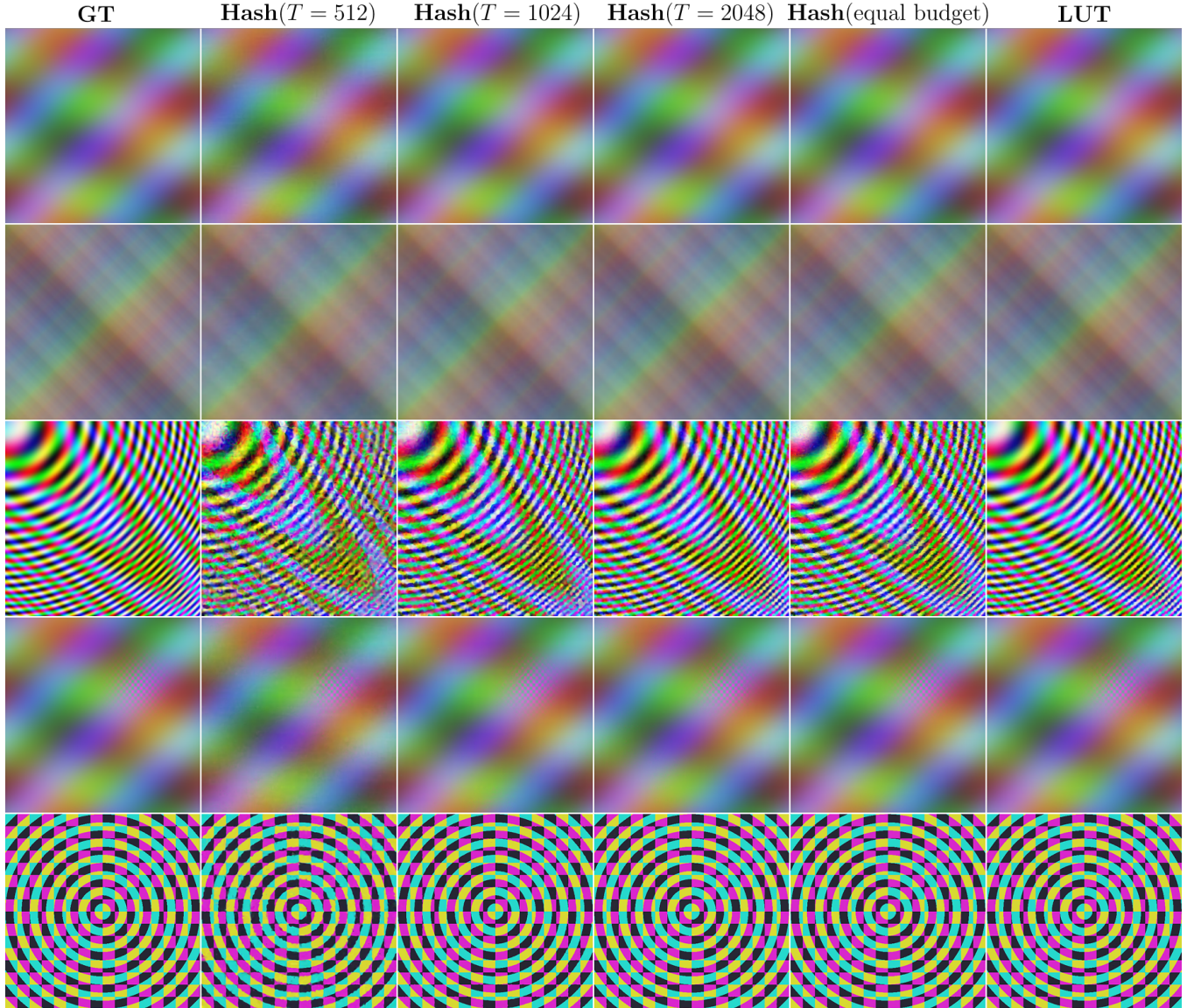


Figure 11: Qualitative comparison on the controlled 2D approximation tasks. Columns show the ground truth followed by hash-grid variants of increasing capacity, the equal-budget hash grid, and our collision-free LUT-Encoding. Rows correspond to the smooth, multiscale, chirp, localized-detail, and checkerboard signals.

Reproducibility

Datasets. We evaluate on Mip-NeRF 360 (Barron et al. 2022), Tanks and Temples, Deep Blending, and NeRF Synthetic (Mildenhall et al. 2020). These datasets are industry-standard benchmarks for novel-view synthesis and enable direct comparison with prior radiance-field and Gaussian-based methods under commonly used train/test splits.

Metrics. We report PSNR, SSIM, and LPIPS, the standard metrics for this task. They quantify pixel fidelity, structural similarity, and perceptual quality, respectively, and are the common reporting suite in the literature we compare against. Hyperparameter selection during tuning is guided by validation/test PSNR.

Data preprocessing. No dataset-specific image preprocessing is required (e.g., no custom rescaling or color normalization beyond the standard loaders used for these benchmarks). Posed RGB images and camera parameters are consumed in their usual released form. Operations such as nearest-neighbor distance estimation for initializing primitive scales are part of model initialization, not an external data-preprocessing stage.

Randomness. Training uses fixed configuration defaults. A subset of primitive parameters (e.g., latent descriptors and network weights) is randomly initialized at the start of optimization; afterward, densification/pruning and optimization are deterministic given that initialization and the data order. At the beginning of every training run seeds are set. Residual nondeterminism may arise from GPU/CUDA kernels and hardware-accelerated ray tracing.

Hyperparameter search. We tune hyperparameters by coordinate descent: for a selected macro-parameter we evaluate a small set of candidates in the range $[\text{selected}/100, 100 \cdot \text{selected}]$, where selected is the current best value (or a functionally analogous value from a previous model), keep the candidate with the best PSNR, and move to the next parameter. Final reported numbers use the configuration available in the code supplement.

Compute setup. All main experiments were run on a single NVIDIA RTX 3090 GPU with CUDA/OptiX hardware-accelerated ray-triangle intersection.

Extended Description of Floating Radiance Networks

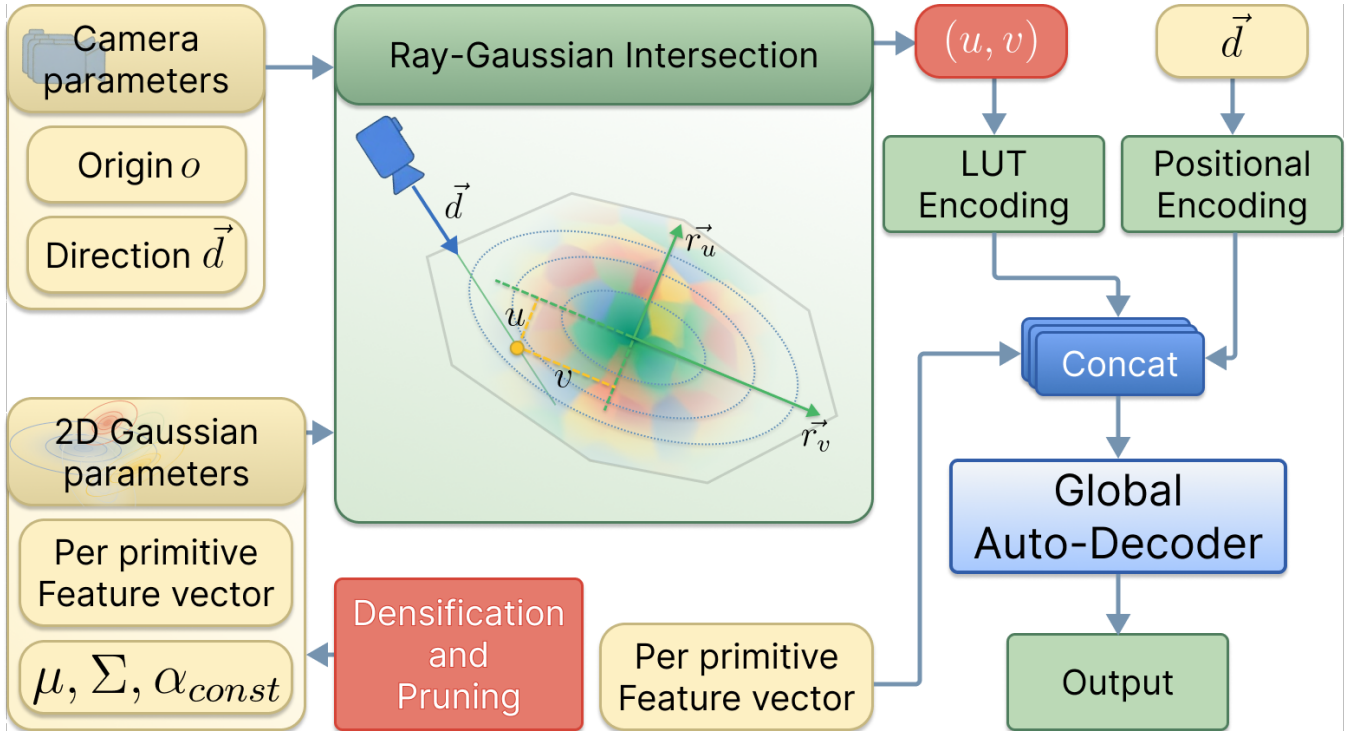


Figure 12: The Floating Radiance Networks (FlaRe) pipeline

Gaussian Primitive Scene Representation

FlaRe represents the scene as a collection of anisotropic planar generalized Gaussian primitives with an isotropic kernel defined in normalized local coordinates.:

$$\mathcal{G} = \{(\mathcal{N}(\mu_i, \Sigma_i, \kappa_i), c_{\text{const},i}, \alpha_{\text{const},i}, z_i)\}_{i=1}^N.$$

where:

- $\mathcal{N}(\mu_i, \Sigma_i, \kappa_i)$ denotes the Isotropic Planar Generalized Gaussian Kernel parametrized by:
 - $\mu_i \in \mathbb{R}^3$: the trainable mean of the Gaussian;

- $\Sigma_i \in \mathbb{R}^{3 \times 3}$: the singular covariance matrix defined via the rotation matrix R_i (derived from the trainable quaternion q_i) and the scaling matrix $S_i = \text{diag}([s_i, 0]) = \text{diag}([\exp(\tilde{s}_i), 0])$ for the trainable parameter $\tilde{s}_i \in \mathbb{R}^2$ controlling the primitive scale in its local tangent plane, admitting the following factorization:

$$\Sigma_i = R_i \cdot S_i \cdot S_i^T \cdot R_i^T$$

- $\kappa_i = 1 + \text{softplus}(\tilde{\kappa}_i)$: the shape parameter of the Gaussian for the trainable parameter $\tilde{\kappa}_i \in \mathbb{R}$;
- $c_{\text{const},i} \in \mathbb{R}^3$ represents the trainable view-independent constant color parameter of the i -th Gaussian in the RGB space;
- $\alpha_{\text{const},i} = \sigma(\tilde{\alpha}_{\text{const},i})$ defines the constant opacity of the i -th Gaussian for the trainable parameter $\tilde{\alpha}_{\text{const},i} \in \mathbb{R}$;
- $z_i \in \mathbb{R}^{96}$ serves as the trainable per-primitive local 2D neural radiance field descriptor;
- N is the total number of the primitives;

Isotropic Planar Generalized Gaussian Kernel

Following the approach presented in Nexels (Rong et al. 2025), FlaRe employs opacity modulation via an unnormalized generalized Gaussian Kernel in the underlying 2D representation primitives. This formulation enables more precise modeling of sharp boundaries and high-curvature surfaces, effectively mitigating approximation errors inherent in classical Gaussian kernels characterized by a fixed quadratic fall-off profile.

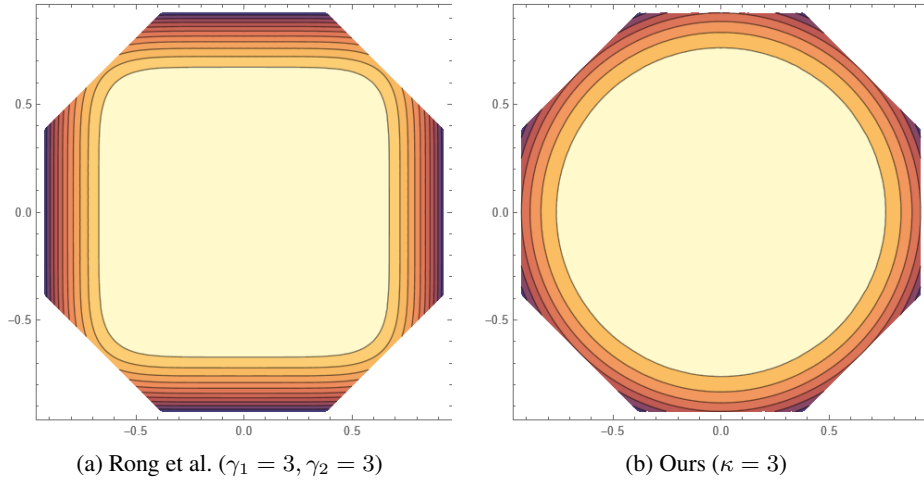


Figure 13: Comparison of kernel value plots for two distinct Generalized Gaussian Kernel formulations: Rong et al. (left) vs. our Isotropic Planar Generalized Gaussian Kernel (right).

Unlike the approach by (Rong et al. 2025), where the value of the final opacity α is computed using the following formula:

$$\alpha = o \exp\left(-\frac{|u|^{2\gamma_1}}{2}\right) \exp\left(-\frac{|v|^{2\gamma_2}}{2}\right)$$

we adopt a modified definition of the generalized Gaussian Kernel resulting in our Isotropic Planar Generalized Gaussian Kernel, given by the following formula:

$$\mathcal{N}(\mu, \Sigma, \kappa)(x) = \exp\left(-\frac{\|PS^{-1}R^T(x - \mu)\|_2^{2\kappa}}{2\kappa}\right)$$

where P is the projection matrix, defined as:

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Note that for (u, v) expressed in the Gaussian local coordinate frame, the above-mentioned formula simplifies to:

$$\mathcal{N}(\mu, \Sigma, \kappa)(u, v) = \exp\left(-\frac{(u^2 + v^2)^\kappa}{2\kappa}\right)$$

yielding the following value of the final modulated opacity α :

$$\alpha = \alpha_{\text{const}} \cdot \alpha_{\text{MLP}} \cdot \exp \left(-\frac{(u^2 + v^2)^\kappa}{2\kappa} \right)$$

where α_{MLP} denotes the opacity predicted by our auto-decoder MLP as detailed in the Subsection **Global auto-decoder**.

This approach allows us to utilize a single canonical polygon for each Gaussian, regardless of the κ parameter, since the density function’s isocontours remain isotropic. In contrast, for the density function proposed by (Rong et al. 2025), the polygon’s geometry does not align with the isocontours, preventing the density values from vanishing at the boundary and necessitating the computation of a unique proxy polygon for every individual value of the shape parameter.

Global auto-decoder architecture

In contrast to conventional Gaussian-based representations (e.g., 3DGS (Kerbl et al. 2023) or 3DGRT (Moenne-Loccoz et al. 2024)), which rely on static, per-primitive color and opacity attributes — typically optimized as trainable constants modulated by a spatial Gaussian kernel — FlaRe adopts a more flexible approach. Specifically, both the color and opacity of samples within each 2D Gaussian footprint are predicted by a dedicated 2D neural radiance field, uniquely assigned to each individual primitive and operating within its specific local coordinate frame, and encoded individually for each Gaussian primitive as a 96-element FP16 trainable latent descriptor vector. The color and opacity are then decoded using a single global auto-decoder MLP:

$$\Phi \left(\left(u_i^{(r)}, v_i^{(r)} \right), \vec{d}^{(r)}, z_i^{(r)}; \mathcal{G}, \mathcal{F}, \Theta \right) = \left(c_{\text{MLP},i}^{(r)}, \alpha_{\text{MLP},i}^{(r)} \right)$$

where $\left(c_{\text{MLP},i}^{(r)}, \alpha_{\text{MLP},i}^{(r)} \right) \in \mathbb{R}^3 \times \mathbb{R}$ denotes the color and opacity predicted by the MLP at the intersection point $P_i^{(r)}$ of ray r and the i -th primitive encountered by the ray, given the direction vector $\vec{d}^{(r)}$ and per-primitive local 2D neural radiance field descriptor $z_i^{(r)}$ with $P_i^{(r)}$ having the coordinates: $\left(u_i^{(r)}, v_i^{(r)} \right)$ within the primitive’s local reference frame. The network is conditioned on the following entities:

- $\mathcal{G}$ representing the set of all Gaussian primitives;
- $\mathcal{F}$ defining the set of the LUT-Table features;
- Θ denoting the MLP parameters.

Our global auto-decoder MLP adopts a lightweight architecture, as detailed in Figure 14:

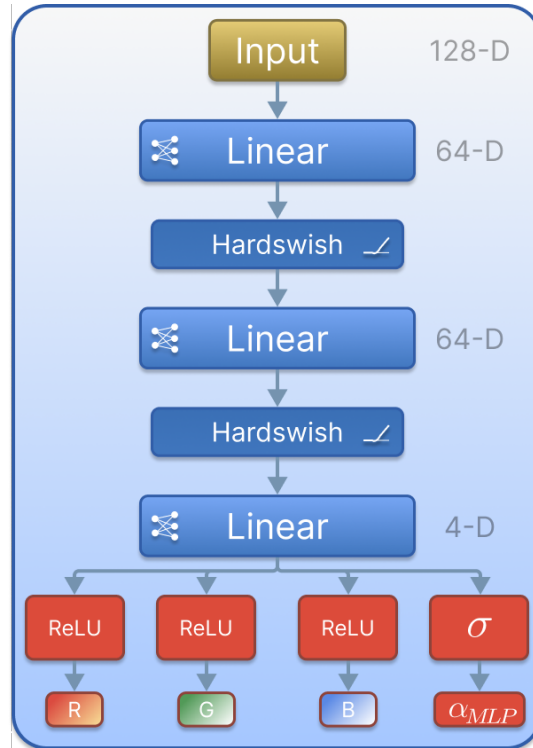


Figure 14: FlaRe MLP

The architecture comprises a 128-wide input layer, two 64-neuron hidden layers each followed by a Hardswish activation, and a 4-neuron output layer, which branches into four channels processed by three ReLU and one σ activation functions, respectively.

Prior to entering the network, input data are processed through specific encoding schemes: the spatial coordinates $(u_i^{(r)}, v_i^{(r)})$ are mapped using LUT-Encoding to produce an 8-dimensional vector (4 levels $\times$ 2 feature dimensions), while the view direction vector $\vec{d}^{(r)}$ is encoded via positional encoding across 4 frequency bands yielding a 24-dimensional vector (3 coordinates $\times$ 4 bands $\times$ 2 trigonometric functions). These encoded vectors are then concatenated with the 96-dimensional individual per-primitive neural radiance field descriptor to form a 128-dimensional latent vector, which serves as the input to our MLP (see Figure 12). The forward pass of the neural network Φ can be expressed by the following formula:

$$\Phi\left(\left(u_i^{(r)}, v_i^{(r)}\right), \vec{d}^{(r)}, z_i^{(r)}; \mathcal{G}, \mathcal{F}, \Theta\right) = W_3 \varphi_3 \left(W_2 \varphi_2 \left(W_1 \varphi_1 \left(\begin{bmatrix} E_{\text{LUT}}\left(u_i^{(r)}, v_i^{(r)}\right) \\ \gamma\left(\vec{d}^{(r)}\right) \\ z_i^{(r)} \end{bmatrix} \right) + b_1 \right) + b_2 \right) + b_3$$

where:

$$\varphi_1(x) = \begin{bmatrix} \text{hardswish}(x_1) \\ \vdots \\ \text{hardswish}(x_{128}) \end{bmatrix}$$

$$\varphi_2(x) = \begin{bmatrix} \text{hardswish}(x_1) \\ \vdots \\ \text{hardswish}(x_{64}) \end{bmatrix}$$

$$\varphi_3(x) = \begin{bmatrix} \text{ReLU}(x_1) \\ \text{ReLU}(x_2) \\ \text{ReLU}(x_3) \\ \sigma(x_4) \end{bmatrix}$$

Our LUT-Encoding mechanism generates latent vectors functionally equivalent to the hash-based encodings proposed in Instant-NGP (Müller et al. 2022). However, unlike their approach, which relies on high-dimensional hash-grid lookups, we leverage the fact that our floating primitives already carry a significant portion of the geometric representation. Consequently, we can reduce the encoding overhead by employing a 4-level multiresolution structure with grid resolutions of: 16×16 , 25×25 , 40×40 , and 64×64 . The compact size of these 2D grids enables a direct, collision-free one-to-one mapping between grid vertices and feature vectors, effectively replacing the hashing mechanism with a straightforward lookup table (LUT) (see Figure 15). The two-dimensional feature vectors are serialized in row-major order within fast on-chip memory across the four consecutive levels.

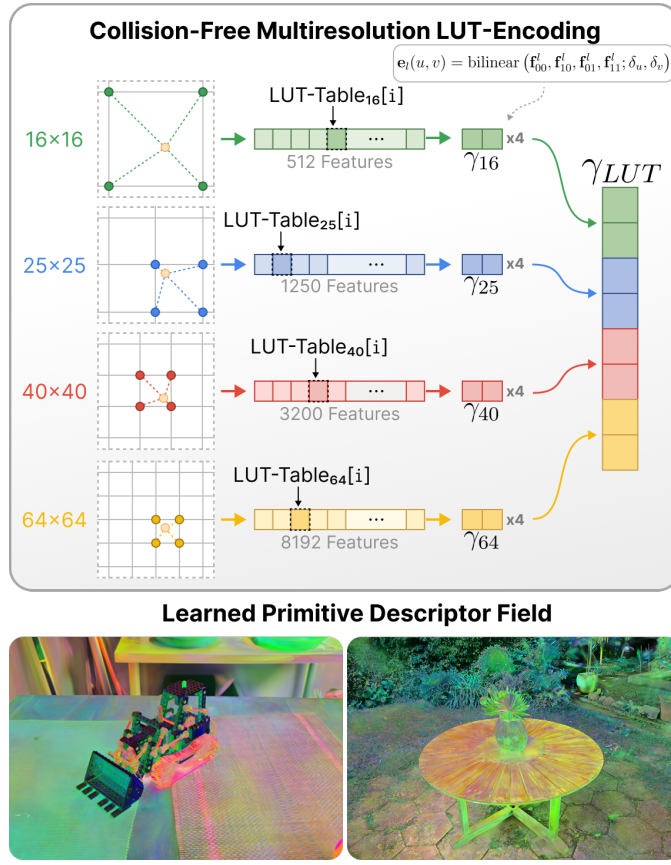


Figure 15: LUT Encoding

The calculations below illustrate how the indices of the four adjacent 2D LUT-grid entries are determined for a given level i . For brevity, we have omitted both the superscript (r) (where $r \in \mathcal{B}$ specifies the particular ray from the batch $\mathcal{B}$) and the subscript i (denoting, before notation changes, the index of the Gaussian primitive hit by ray r), for which the (u, v) coordinates are computed. First, let us compute:

$$\begin{pmatrix} u^{(i)} \\ v^{(i)} \end{pmatrix} = \begin{pmatrix} \frac{u}{R_*} + 1}{2} \cdot (N^{(i)} - 1) \\ \frac{v}{R_*} + 1}{2} \cdot (N^{(i)} - 1) \end{pmatrix}^T$$

Then, we can determine the coordinates of the top-left corner LUT feature map entry: $(u_{\text{floor}}^{(i)}, v_{\text{floor}}^{(i)}) \in (\mathbb{N} \cup \{0\}) \times (\mathbb{N} \cup \{0\})$ used in the bilinear interpolation, as well as the interpolation coefficients: $(u_{\text{frac}}^{(i)}, v_{\text{frac}}^{(i)}) \in \mathbb{R}^2$, using the following formula:

$$\begin{pmatrix} u_{\text{floor}}^{(i)} \\ v_{\text{floor}}^{(i)} \end{pmatrix} = \begin{pmatrix} \min \left\{ \left\lfloor \frac{u}{R_*} + 1}{2} \cdot (N^{(i)} - 1) \right\rfloor, N^{(i)} - 2 \right\} \\ \min \left\{ \left\lfloor \frac{v}{R_*} + 1}{2} \cdot (N^{(i)} - 1) \right\rfloor, N^{(i)} - 2 \right\} \end{pmatrix}^T$$

$$\begin{pmatrix} u_{\text{frac}}^{(i)} \\ v_{\text{frac}}^{(i)} \end{pmatrix} = \begin{pmatrix} u^{(i)} - u_{\text{floor}}^{(i)} \\ v^{(i)} - v_{\text{floor}}^{(i)} \end{pmatrix}^T$$

where $i \in \{1, \dots, 4\}$ denotes the LUT resolution level, $N^{(i)}$ defines the LUT feature map resolution at the i -th level and R_* is the maximum radius (over all κ values) of the Gaussian footprint in the reference frame with the identity Gaussian's covariance matrix acting as a normalization constant that ensures the coordinates of each of the four adjacent 2D LUT-grid entries remain within their bounds.

Note that although using a single normalization constant R_* for all possible values of the parameter κ may leave a significant number of LUT-grid entries unutilized, introducing individual normalization for each κ value could lead to training instability (as varying κ values would cause the same coordinates in the local primitives' reference frame to be mapped to different LUT entries across training iterations) and would significantly complicate the gradient formulation.

In order to derive the formula for the normalization constant R_* , let us take $\kappa \in (1, \infty)$. Then the Gaussian footprint radius $R(\kappa)$ for the given κ and the fixed configurable $\alpha_{\min}$ parameter is given by the following expression:

$$R(\kappa) = (-2\kappa \cdot \ln(\alpha_{\min}))^{\frac{1}{2\kappa}}$$

Now it can be shown using basic calculus that:

$$R_* = \sup_{\kappa \in (1, \infty)} R(\kappa) = \begin{cases} \sqrt{-2 \ln(\alpha_{\min})} & , \alpha_{\min} < e^{-\frac{5}{2}} \\ (\alpha_{\min})^{-\frac{1}{e}} & , \text{otherwise} \end{cases}$$

Let us define:

$$\text{offsets} = \begin{bmatrix} 0 \\ 0 + (16 \times 16 \times 2) \\ 0 + (16 \times 16 \times 2) + (25 \times 25 \times 2) \\ 0 + (16 \times 16 \times 2) + (25 \times 25 \times 2) + (40 \times 40 \times 2) \end{bmatrix}^T = [0, \quad 512, \quad 1762, \quad 4962]$$

Then, for a given LUT resolution level $i \in \{1, \dots, 4\}$, one can compute the base memory addresses: $\text{ind}_{00}^{(i)}$, $\text{ind}_{01}^{(i)}$, $\text{ind}_{10}^{(i)}$ and $\text{ind}_{11}^{(i)}$ for the feature vectors: $\mathbf{f}_{00}^{(i)}, \mathbf{f}_{01}^{(i)}, \mathbf{f}_{10}^{(i)}, \mathbf{f}_{11}^{(i)} \in \mathbb{R}^2$ using the following formula:

$$(\text{ind}_{00}^{(i)}, \text{ind}_{01}^{(i)}, \text{ind}_{10}^{(i)}, \text{ind}_{11}^{(i)}) = \begin{pmatrix} \text{offsets}_i + \left(\left(\left(v_{\text{floor}}^{(i)} \cdot N^{(i)} \right) + u_{\text{floor}}^{(i)} \right) \cdot 2 \right) \\ \text{offsets}_i + \left(\left(\left(v_{\text{floor}}^{(i)} \cdot N^{(i)} \right) + (u_{\text{floor}}^{(i)} + 1) \right) \cdot 2 \right) \\ \text{offsets}_i + \left(\left(\left((v_{\text{floor}}^{(i)} + 1) \cdot N^{(i)} \right) + u_{\text{floor}}^{(i)} \right) \cdot 2 \right) \\ \text{offsets}_i + \left(\left(\left((v_{\text{floor}}^{(i)} + 1) \cdot N^{(i)} \right) + (u_{\text{floor}}^{(i)} + 1) \right) \cdot 2 \right) \end{pmatrix}^T$$

Consequently, as $\mathcal{F} = \{\mathbf{f}_1, \dots, \mathbf{f}_{6577}\}$ we have:

$$\begin{aligned} (\mathbf{f}_{00}^{(i)}, \mathbf{f}_{01}^{(i)}, \mathbf{f}_{10}^{(i)}, \mathbf{f}_{11}^{(i)}) &= \left(\mathbf{f}_{\frac{\text{ind}_{00}^{(i)}}{2} + 1}, \mathbf{f}_{\frac{\text{ind}_{01}^{(i)}}{2} + 1}, \mathbf{f}_{\frac{\text{ind}_{10}^{(i)}}{2} + 1}, \mathbf{f}_{\frac{\text{ind}_{11}^{(i)}}{2} + 1} \right) = \\ &= \left(\begin{pmatrix} \text{LUT}[\text{ind}_{00}^{(i)}] \\ \text{LUT}[\text{ind}_{00}^{(i)} + 1] \end{pmatrix}^T, \begin{pmatrix} \text{LUT}[\text{ind}_{01}^{(i)}] \\ \text{LUT}[\text{ind}_{01}^{(i)} + 1] \end{pmatrix}^T, \begin{pmatrix} \text{LUT}[\text{ind}_{10}^{(i)}] \\ \text{LUT}[\text{ind}_{10}^{(i)} + 1] \end{pmatrix}^T, \begin{pmatrix} \text{LUT}[\text{ind}_{11}^{(i)}] \\ \text{LUT}[\text{ind}_{11}^{(i)} + 1] \end{pmatrix}^T \right) \end{aligned}$$

The feature vectors $\mathbf{f}_{00}^{(i)}$, $\mathbf{f}_{01}^{(i)}$, $\mathbf{f}_{10}^{(i)}$, and $\mathbf{f}_{11}^{(i)}$ are then bilinearly interpolated analogously to the approach used in Instant-NGP:

$$\begin{aligned} \hat{\mathbf{f}}^{(i)} &= \left(1 - u_{\text{frac}}^{(i)} \right) \cdot \left(1 - v_{\text{frac}}^{(i)} \right) \cdot \mathbf{f}_{00}^{(i)} + \\ &+ u_{\text{frac}}^{(i)} \cdot \left(1 - v_{\text{frac}}^{(i)} \right) \cdot \mathbf{f}_{01}^{(i)} + \\ &+ \left(1 - u_{\text{frac}}^{(i)} \right) \cdot v_{\text{frac}}^{(i)} \cdot \mathbf{f}_{10}^{(i)} + \\ &+ u_{\text{frac}}^{(i)} \cdot v_{\text{frac}}^{(i)} \cdot \mathbf{f}_{11}^{(i)} \end{aligned}$$

Finally, we obtain:

$$E_{\text{LUT}}(u, v) = [\hat{\mathbf{f}}^{(0)}, \hat{\mathbf{f}}^{(1)}, \hat{\mathbf{f}}^{(2)}, \hat{\mathbf{f}}^{(3)}]$$

Mesh representation of 2D Gaussian primitive

In the spirit of the technique proposed in (Moenne-Loccoz et al. 2024), we use mesh proxies, defined as triangulated stretched regular polygons, as spatial containers to encapsulate our 2D Gaussian primitives. This allows us to leverage efficient built-in hardware-accelerated ray-triangle intersection tests, which, unlike custom `INTERSECTION` shaders, are computed on dedicated RTX cores.

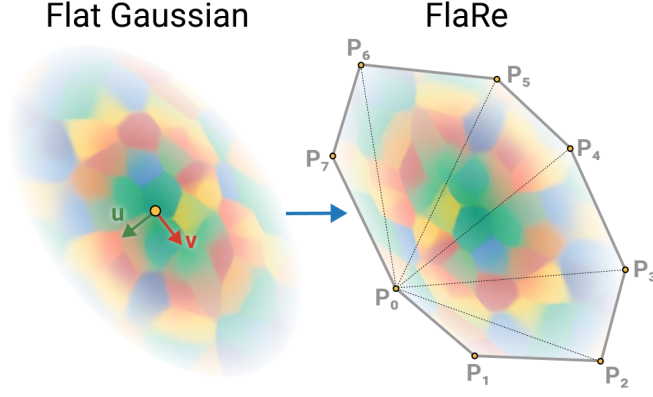


Figure 16: Computing the mesh representation of the Gaussian primitive

To minimize memory overhead, we utilize geometry instancing, where each proxy is represented as an affine transformation of a canonical mesh whose vertices are defined by the following formula:

$$P_i = \begin{bmatrix} \cos\left(\frac{2\pi}{k} \cdot i\right) \\ \sin\left(\frac{2\pi}{k} \cdot i\right) \\ 0 \end{bmatrix}$$

where k denotes the number of sides of the regular polygon. The per-Gaussian transformation matrix is given by:

$$M = \begin{bmatrix} R \cdot S & \mu^T \\ 0 & 1 \end{bmatrix}$$

where:

$$R = \begin{bmatrix} 1 - cc - dd & bc - ad & bd + ac \\ bc + ad & 1 - bb - dd & cd - ab \\ bd - ac & cd + ab & 1 - bb - cc \end{bmatrix}$$

for:

$$s = \frac{2}{q_1^2 + q_2^2 + q_3^2 + q_4^2}$$

$$\begin{aligned} bs &= q_2 \cdot s, & cs &= q_3 \cdot s, & ds &= q_4 \cdot s \\ ab &= q_1 \cdot bs, & ac &= q_1 \cdot cs, & ad &= q_1 \cdot ds \\ bb &= q_2 \cdot bs, & bc &= q_2 \cdot cs, & bd &= q_2 \cdot ds \\ cc &= q_3 \cdot cs, & cd &= q_3 \cdot ds, & dd &= q_4 \cdot ds \end{aligned}$$

and:

$$S = \text{diag}(s_1 \cdot \Delta s, s_2 \cdot \Delta s)$$

with the factor Δs defined as follows, accounting for adaptive clamping to the target opacity $\alpha_{\min}$ at the border of the Gaussian primitive:

$$\Delta s = \begin{cases} \left(-2\kappa \cdot \ln\left(\frac{\alpha_{\min}}{\alpha_{\text{const}}}\right)\right)^{\frac{1}{2\kappa}}, & \alpha_{\min} \leq \alpha_{\text{const}} \\ 0, & \text{otherwise} \end{cases}$$

Mesh modification

Given points $Q_i = M \cdot P_i$ derived from an arbitrary transformation of the original Gaussian proxy polygon vertices P'_i in the world space, we will show how to adapt the 2D Gaussian parameters to the transformed scene without retraining, by directly computing the updated parameters from the base model's state and the deformed geometry Q_i . We present a robust strategy that accounts for cases where the intrinsic structure of the proxy polygon is lost — for instance, due to non-rigid deformations that violate local coplanarity or induce orientation shifts. After computing the local centroid:

$$O' = \frac{1}{k} \sum_{i=1}^k Q_i$$

of the k vertices defining the transformed proxy polygon, we center the point set by translating the centroid to the origin, yielding normalized points:

$$\mathbf{Q}'_i = \mathbf{Q}_i - \mathbf{O}'$$

We then estimate the covariance matrix:

$$\Sigma = \frac{1}{k} \sum_{i=1}^k \mathbf{Q}'_i \mathbf{Q}'_i{}^T$$

of these centered points and perform SVD to extract the principal axes. This procedure identifies the best-fitting plane and the directions of maximum variance, which define the local coordinate frame of the transformed Gaussian. Since the vertices of the canonical proxy polygon are coplanar, the covariance matrix Σ is singular or ill-conditioned. Consequently, the eigenvector corresponding to the smallest eigenvalue—nominally the surface normal—is highly sensitive to numerical instability. To ensure robustness, we compute the surface normal:

$$\vec{\mathbf{N}} = \vec{\mathbf{U}} \times \vec{\mathbf{V}}$$

via the cross product of the two principal eigenvectors $\vec{\mathbf{U}}$ and $\vec{\mathbf{V}}$, which span the tangent plane. This approach provides a numerically stable normal estimate, effectively mitigating the noise associated with the smallest singular vector in degenerate cases. With the basis vectors: $\vec{\mathbf{U}}, \vec{\mathbf{V}}$ spanning the tangent plane and the robustly computed surface normal $\vec{\mathbf{N}}$, we construct the rotation matrix:

$$\mathbf{R} = [\vec{\mathbf{U}} \mid \vec{\mathbf{V}} \mid \vec{\mathbf{N}}]$$

To derive the corresponding rotation quaternion q_i , we employ the algorithm proposed by Bar-Itzhack et al., which ensures numerical stability even for near-singular configurations. To determine the Gaussian scale parameters, we project the normalized vertices $\mathbf{Q}'_i$ onto the local tangent plane defined by the basis vectors $\vec{\mathbf{U}}$ and $\vec{\mathbf{V}}$ obtaining:

$$\hat{\mathbf{Q}}'_i = \text{proj}(\mathbf{Q}'_i, \vec{\mathbf{N}}) = \mathbf{Q}'_i - \vec{\mathbf{N}} \langle \vec{\mathbf{N}}, \mathbf{Q}'_i \rangle$$

and compute the axis-aligned bounding box (AABB) of these projections in the local frame. The scale parameters along each axis are set proportional to the respective AABB dimensions, adjusted by a factor Δs that incorporates adaptive clamping as described in the Subsection **Mesh representation of 2D Gaussian primitive**.

$$s_1 = \log \left(\frac{1}{\Delta s} \cdot \max_{i=1}^k \left| \langle \vec{\mathbf{U}}, \hat{\mathbf{Q}}'_i \rangle \right| \right)$$

$$s_2 = \log \left(\frac{1}{\Delta s} \cdot \max_{i=1}^k \left| \langle \vec{\mathbf{V}}, \hat{\mathbf{Q}}'_i \rangle \right| \right)$$

Finally, we obtain the Gaussian means μ using the formula:

$$\mu = \mathbf{O}'$$

While the aforementioned approach suffices for radially symmetric primitives, it becomes suboptimal for our FlaRe model, where color and opacity are functions of (u, v) coordinates. Specifically, when the local transformation involves shearing, the original orthogonal texture axes are mapped to non-orthogonal vectors. Consequently, the vectors $\vec{\mathbf{U}}$ and $\vec{\mathbf{V}}$ obtained via SVD, which serve as the axes of the primitive’s local reference frame and define the transformed neural texture coordinates (u, v) , fail to coincide with the originally transformed and subsequently projected vectors $\vec{\mathbf{U}}_{\text{tex}}$ and $\vec{\mathbf{V}}_{\text{tex}}$. This misalignment leads to artifacts in the fine details modeled by our per-primitive 2D neural radiance field, despite the perfect geometric alignment of the resulting Gaussian parameters themselves. To ensure consistency between the color representation and the geometric deformation, we subsequently derive a linear transformation matrix $\mathbf{A}$, which maps the texture coordinates (u, v) into the deformed space before they are fed into our MLP.

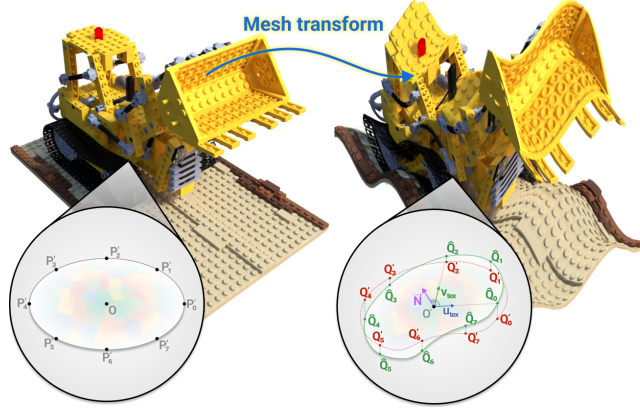


Figure 17: Mesh representation transformation and subsequent retrieval of Gaussian parameters from the transformed mesh without model retraining

If $4 \mid k$ one can simply assume that:

$$\begin{aligned}\vec{U}_{\text{tex}} &= \vec{Q}'_0 \\ \vec{V}_{\text{tex}} &= \vec{Q}'_{\frac{k}{4}}\end{aligned}$$

Otherwise, leveraging the fact that the vector $\vec{P}'_0$ is parallel to the axis $\vec{e}_1$ of the Gaussian's local reference frame, along with the known formulas (see **Mesh representation of 2D Gaussian primitive**) for $\vec{P}'_i$ in terms of the (u, v) coordinates with respect to the $\vec{e}_1$ and $\vec{e}_2$ axes, one can express $\vec{e}_2$ as a linear combination of $\vec{e}_1$ and an arbitrary vector $\vec{P}'_i$ for $i \neq 0$, and then compute the corresponding linear combination of $\vec{Q}'_i$ and $\vec{U}_{\text{tex}} = \vec{Q}'_0$ using these same coefficients to obtain $\vec{V}_{\text{tex}}$.

Let us assume, therefore, that we have determined the vectors: $\vec{U}_{\text{tex}}$ and $\vec{V}_{\text{tex}}$. Then, expressing the above vectors in the reference frame determined by the SVD and normalizing the coordinate frame so that the covariance matrix of the Gaussian is the identity, we obtain:

$$\begin{aligned}\vec{U}'_{\text{tex}} &= \left[\frac{\langle \vec{U}, \vec{U}_{\text{tex}} \rangle}{s_1}, \frac{\langle \vec{V}, \vec{U}_{\text{tex}} \rangle}{s_2} \right] \\ \vec{V}'_{\text{tex}} &= \left[\frac{\langle \vec{U}, \vec{V}_{\text{tex}} \rangle}{s_1}, \frac{\langle \vec{V}, \vec{V}_{\text{tex}} \rangle}{s_2} \right]\end{aligned}$$

Then, the vectors orthogonal to them are given by:

$$\begin{aligned}\vec{U}'_{\perp, \text{tex}} &= \left[-\frac{\langle \vec{V}, \vec{U}_{\text{tex}} \rangle}{s_2}, \frac{\langle \vec{U}, \vec{U}_{\text{tex}} \rangle}{s_1} \right] \\ \vec{V}'_{\perp, \text{tex}} &= \left[-\frac{\langle \vec{V}, \vec{V}_{\text{tex}} \rangle}{s_2}, \frac{\langle \vec{U}, \vec{V}_{\text{tex}} \rangle}{s_1} \right]\end{aligned}$$

We want to determine the scaling factors: k_1 and k_2 that will allow us to compute the target vectors: $\vec{U}''_{\text{tex}}$ and $\vec{V}''_{\text{tex}}$ spanning the Gaussian's tight-fitting parallelogram with sides parallel to: $\vec{U}_{\text{tex}}$ and $\vec{V}_{\text{tex}}$. Note that these factors satisfy the following system of two independent equations:

$$\begin{cases} \left\langle \frac{\vec{U}'_{\perp, \text{tex}}}{\|\vec{U}'_{\perp, \text{tex}}\|}, \vec{V}'_{\text{tex}} k_2 \right\rangle = 1 \\ \left\langle \frac{\vec{V}'_{\perp, \text{tex}}}{\|\vec{V}'_{\perp, \text{tex}}\|}, \vec{U}'_{\text{tex}} k_1 \right\rangle = 1 \end{cases}$$

By transforming the above system of equations, we successively obtain:

$$\begin{cases} \frac{k_2}{\|\vec{U}'_{\perp, \text{tex}}\|} \langle \vec{U}'_{\perp, \text{tex}}, \vec{V}'_{\text{tex}} \rangle = 1 \\ \frac{k_1}{\|\vec{V}'_{\perp, \text{tex}}\|} \langle \vec{V}'_{\perp, \text{tex}}, \vec{U}'_{\text{tex}} \rangle = 1 \end{cases}$$

$$\begin{cases} k_2 \frac{\langle \vec{U}'_{\perp, \text{tex}}, \vec{V}'_{\text{tex}} \rangle}{\|\vec{U}'_{\perp, \text{tex}}\|} = 1 \\ k_1 \frac{\langle \vec{V}'_{\perp, \text{tex}}, \vec{U}'_{\text{tex}} \rangle}{\|\vec{V}'_{\perp, \text{tex}}\|} = 1 \end{cases}$$

$$\begin{cases} k_1 = \frac{\|\vec{V}'_{\perp, \text{tex}}\|}{\langle \vec{V}'_{\perp, \text{tex}}, \vec{U}'_{\text{tex}} \rangle} \\ k_2 = \frac{\|\vec{U}'_{\perp, \text{tex}}\|}{\langle \vec{U}'_{\perp, \text{tex}}, \vec{V}'_{\text{tex}} \rangle} \end{cases}$$

Hence, after transforming back to the original coordinate system, we obtain:

$$\begin{aligned} \vec{U}''_{\text{tex}} &= k_1 \cdot \text{diag} \cdot (s_1, s_2) \cdot \vec{U}'_{\text{tex}} = \begin{bmatrix} k_1 \langle \vec{U}, \vec{U}_{\text{tex}} \rangle, & k_1 \langle \vec{V}, \vec{U}_{\text{tex}} \rangle \end{bmatrix} \\ \vec{V}''_{\text{tex}} &= k_2 \cdot \text{diag} \cdot (s_1, s_2) \cdot \vec{V}'_{\text{tex}} = \begin{bmatrix} k_2 \langle \vec{U}, \vec{V}_{\text{tex}} \rangle, & k_2 \langle \vec{V}, \vec{V}_{\text{tex}} \rangle \end{bmatrix} \end{aligned}$$

Now, since:

$$(u, v)^T = \begin{bmatrix} \vec{U}''_{\text{tex}} & \vec{V}''_{\text{tex}} \end{bmatrix} \cdot (u', v')^T$$

we finally obtain:

$$(u', v')^T = A \cdot (u, v)^T$$

for:

$$A = \begin{bmatrix} \vec{U}''_{\text{tex}} & \vec{V}''_{\text{tex}} \end{bmatrix}^{-1}$$

Finally, let us note that since the vectors: $\vec{U}''_{\text{tex}}$ and $\vec{V}''_{\text{tex}}$ span the Gaussian's tight-fitting parallelogram, and the Gaussian footprint is contained within this parallelogram, the (u', v') coordinates will fall within the valid range for the encoding function implementing the LUT-encoding.

Color aggregation along the ray

Following the approach described in (Kopanas et al. 2021, 2022), the FlaRe method maps 2D planar Gaussian primitives onto the image plane by blending colors for individual pixels. This is achieved by collecting samples at the intersection points of a ray, traced from the camera through the pixel center, with the Gaussian primitives' footprints, which are defined by the boundary of their proxy meshes. The final unclamped pixel color $\tilde{I}^{(r)}$ for the ray $r \in \mathcal{B}$, where by $r \in \mathcal{B}$ we denote the batch of rays, is given by the following formula:

$$\tilde{I}^{(r)} = \sum_{i=1}^{|\mathcal{G}^{(r)}|} c^{(r)} \alpha_i^{(r)} T_i^{(r)} = \sum_{i=1}^{|\mathcal{G}^{(r)}|} c_{\text{const}, i}^{(r)} c_{\text{MLP}, i}^{(r)} \alpha_i^{(r)} \prod_{j=1}^{i-1} (1 - \alpha_j^{(r)})$$

where:

$$\alpha_i^{(r)} = \alpha_{\text{const}, i}^{(r)} \cdot \alpha_{\text{MLP}, i}^{(r)} \cdot \exp \left(- \frac{\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}}}{2 \kappa_i^{(r)}} \right)$$

Loss function

In our differentiable rendering framework, we define a composite loss function to guide the optimization of the scene representation. The total loss $\mathcal{L}$ is formulated as the sum of a convex combination of $\mathcal{L}_{\text{RGB}, \text{FlaRe}}$ and $\mathcal{L}_{\text{RGB}, \text{base}}$ —representing the $\mathcal{L}_2$ photometric losses in our target FlaRe model and the simplified base model, respectively—and the common $\mathcal{L}_s$ term. The base model relies solely on the view-independent constant color parameters expressed in the RGB space, $c_{\text{const}, i}$, and opacity parameters $\bar{\alpha}_{\text{const}, i}$. To prevent overfitting - which occurs in certain scenes when the FlaRe model tends to capture fine details via the 2D neural radiance field rather than the underlying geometry — we introduce a warmup phase. During this phase, the loss function smoothly transitions between the base and target FlaRe models. This warmup allows for the preliminary optimization of the geometry before the 2D neural radiance field is optimized in the target model. The loss function $\mathcal{L}$ itself is expressed as:

$$\mathcal{L} = \lambda_1 \cdot ((1 - \lambda) \cdot \mathcal{L}_{\text{RGB}, \text{base}} + \lambda \cdot \mathcal{L}_{\text{RGB}, \text{FlaRe}}) + \lambda_2 \cdot \mathcal{L}_s$$

where:

$$\lambda = \text{clamp} \left(\frac{\text{iter} - \text{warmup_start_iter}}{\text{warmup_end_iter} - \text{warmup_start_iter}}, 0, 1 \right)$$

and $\lambda_1, \lambda_2 \geq 0$ are non-negative hyperparameters that balance the contribution of each objective term. Both $\mathcal{L}_{\text{RGB}, \text{FlaRe}}$ and $\mathcal{L}_{\text{RGB}, \text{base}}$ are given by the same formula:

$$\mathcal{L}_{\text{RGB}, \text{FlaRe}} = \frac{1}{3 |\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{i=1}^3 \left(I_i^{(r)} - \hat{I}_i^{(r)} \right)^2$$

$$\mathcal{L}_{\text{RGB,base}} = \frac{1}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{i=1}^3 \left(I_i^{(r)} - \hat{I}_i^{(r)} \right)^2$$

where $I_i^{(r)}$ denotes the i -th color channel value of the rendered pixel corresponding to ray r clamped to the interval $[0, 1]$ (the value before clamping is denoted by $\tilde{I}_i^{(r)}$), and $\hat{I}_i^{(r)}$ is the ground-truth value of the corresponding pixel in the reference image. Lastly, the common regularization component $\mathcal{L}_s$ defined as:

$$\mathcal{L}_s = \frac{1}{N} \sum_{i=1}^N \|s_i\|$$

serves as a regularization term on the Gaussians' scale parameters $\tilde{s}_i$, preventing them from growing excessively. This constraint promotes compactness, which effectively improves both training and inference efficiency.

We intentionally omitted the depth distortion and normal consistency regularization component from 2DGS (Huang et al. 2024). In the case of the former, we obtained worse results in terms of PSNR on test poses in the scene reconstruction task, whereas the latter is inherently incompatible with our batch training paradigm. Specifically, batch sampling does not guarantee that the rays corresponding to the direct spatial neighbors of a ray for the pixel (i, j) in a training image will be present simultaneously, which are required for estimating surface normals.

Implementation details

Since the `OPTIXCOOPVEC` abstraction introduced in OptiX 9.0 is architecture-agnostic and does not allow the developer to manually manipulate the matrix layout within shader code — with the matrix remaining static throughout the entire `OPTIXLAUNCH` execution once packed via the host-side `OPTIXCOOPVEC_MATRIX_CONVERT` function — and because matrix multiplication currently supports only half-precision FP16 accumulation, which (due to accumulated numerical errors and resulting rendering artifacts) precludes its application even to small networks with 64–128 neurons and 2 hidden layers, we adopted a hybrid approach in our method. In an interleaved fashion, we perform `OPTIXLAUNCH` calls — used solely to determine the indices of consecutive planar Gaussian primitives hit by the ray within the `HIT_BUFFER_SIZE` limit and their corresponding ray t -parameters — along with a custom CUDA kernel designed to prepare the input matrix for our auto-decoder MLP by manually manipulating the data layout in the registers and evaluating it through the network for both the forward and backward passes. Following the 3DGRT (Moenne-Loccoz et al. 2024) approach, we set the hit buffer size `HIT_BUFFER_SIZE` to 16 and execute these two operations iteratively within a loop, followed by stream compaction to remove inactive rays until all rays are fully processed.

Derivation of Gradient Formulas

Computing upstream gradients

Computing $\frac{d}{dc_{\text{const},i}^{(r)}} \mathcal{L}_{\text{RGB,Flare}}$

$$\begin{aligned} \frac{d}{dc_{\text{const},i}^{(r)}} \mathcal{L}_{\text{RGB,Flare}} &= \frac{d}{dc_{\text{const},i}^{(r)}} \left(\frac{1}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(I_j^{(r)} - \hat{I}_j^{(r)} \right)^2 \right) = \\ &= \frac{1}{3|\mathcal{B}|} \cdot \frac{d}{dc_{\text{const},i}^{(r)}} \left(\sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(I_j^{(r)} - \hat{I}_j^{(r)} \right)^2 \right) = \\ &= \frac{1}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \frac{d}{dc_{\text{const},i}^{(r)}} \left(\sum_{j=1}^3 \left(I_j^{(r)} - \hat{I}_j^{(r)} \right)^2 \right) = \\ &= \frac{1}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \frac{d}{dc_{\text{const},i}^{(r)}} \left(\left(I_j^{(r)} - \hat{I}_j^{(r)} \right)^2 \right) = \\ &= \frac{1}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 2 \left(I_j^{(r)} - \hat{I}_j^{(r)} \right) \frac{d}{dc_{\text{const},i}^{(r)}} I_j^{(r)} = \\ &= \frac{2}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(I_j^{(r)} - \hat{I}_j^{(r)} \right) \frac{d}{dc_{\text{const},i}^{(r)}} I_j^{(r)} = \end{aligned}$$

$$\begin{aligned}
&= \frac{2}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(\mathbf{I}_j^{(r)} - \hat{\mathbf{I}}_j^{(r)} \right) \frac{\mathbf{d}}{\mathbf{d}\mathbf{c}_{\text{const},i}^{(r)}} \text{clamp} \left(\tilde{\mathbf{I}}_j^{(r)}, 0, 1 \right) = \\
&= \frac{2}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(\mathbf{I}_j^{(r)} - \hat{\mathbf{I}}_j^{(r)} \right) \cdot \mathbb{1}_{\tilde{\mathbf{I}}_j^{(r)} \in [0,1]} \cdot \frac{\mathbf{d}}{\mathbf{d}\mathbf{c}_{\text{const},i}^{(r)}} \tilde{\mathbf{I}}_j^{(r)} = \\
&= \frac{2}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(\mathbf{I}_j^{(r)} - \hat{\mathbf{I}}_j^{(r)} \right) \cdot \mathbb{1}_{\tilde{\mathbf{I}}_j^{(r)} \in [0,1]} \cdot \frac{\mathbf{d}}{\mathbf{d}\mathbf{c}_{\text{const},i}^{(r)}} \left(\sum_{k=1}^{|\mathcal{G}^{(r)}|} \mathbf{c}_{\text{const},k,j}^{(r)} \mathbf{c}_{\text{MLP},k,j}^{(r)} \alpha_k^{(r)} T_k^{(r)} \right) = \\
&= \frac{2}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(\mathbf{I}_j^{(r)} - \hat{\mathbf{I}}_j^{(r)} \right) \cdot \mathbb{1}_{\tilde{\mathbf{I}}_j^{(r)} \in [0,1]} \cdot \left(\sum_{k=1}^{|\mathcal{G}^{(r)}|} \frac{\mathbf{d}}{\mathbf{d}\mathbf{c}_{\text{const},i}^{(r)}} \left(\mathbf{c}_{\text{const},k,j}^{(r)} \mathbf{c}_{\text{MLP},k,j}^{(r)} \alpha_k^{(r)} T_k^{(r)} \right) \right) = \\
&= \frac{2}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(\mathbf{I}_j^{(r)} - \hat{\mathbf{I}}_j^{(r)} \right) \cdot \mathbb{1}_{\tilde{\mathbf{I}}_j^{(r)} \in [0,1]} \cdot \left(\sum_{k=1}^{|\mathcal{G}^{(r)}|} \mathbf{c}_{\text{MLP},k,j}^{(r)} \alpha_k^{(r)} T_k^{(r)} \cdot \frac{\mathbf{d}\mathbf{c}_{\text{const},k,j}^{(r)}}{\mathbf{d}\mathbf{c}_{\text{const},i}^{(r)}} \right) = \\
&= \frac{2}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(\mathbf{I}_j^{(r)} - \hat{\mathbf{I}}_j^{(r)} \right) \cdot \mathbb{1}_{\tilde{\mathbf{I}}_j^{(r)} \in [0,1]} \cdot \left(\sum_{k=1}^{|\mathcal{G}^{(r)}|} \mathbf{c}_{\text{MLP},k,j}^{(r)} \alpha_k^{(r)} T_k^{(r)} \cdot \delta_{ik} \cdot \vec{\epsilon}_j \right) = \\
&= \frac{2}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(\mathbf{I}_j^{(r)} - \hat{\mathbf{I}}_j^{(r)} \right) \cdot \mathbb{1}_{\tilde{\mathbf{I}}_j^{(r)} \in [0,1]} \cdot \mathbf{c}_{\text{MLP},i,j}^{(r)} \alpha_i^{(r)} T_i^{(r)} \cdot \vec{\epsilon}_j = \\
&= \frac{2}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \left(\mathbf{I}^{(r)} - \hat{\mathbf{I}}^{(r)} \right) \cdot \mathbb{1}_{\tilde{\mathbf{I}}^{(r)} \in [0,1]} \cdot \mathbf{c}_{\text{MLP},i}^{(r)} \alpha_i^{(r)} T_i^{(r)}
\end{aligned}$$

Computing $\frac{\mathbf{d}}{\mathbf{d}\tilde{\alpha}_{\text{const},i}^{(r)}} \mathcal{L}_{\text{RGB,Flare}}$

$$\begin{aligned}
\frac{\mathbf{d}}{\mathbf{d}\tilde{\alpha}_{\text{const},i}^{(r)}} \mathcal{L}_{\text{RGB,Flare}} &= \frac{\mathbf{d}}{\mathbf{d}\tilde{\alpha}_{\text{const},i}^{(r)}} \left(\frac{1}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(\mathbf{I}_j^{(r)} - \hat{\mathbf{I}}_j^{(r)} \right)^2 \right) = \\
&= \frac{1}{3|\mathcal{B}|} \cdot \frac{\mathbf{d}}{\mathbf{d}\tilde{\alpha}_{\text{const},i}^{(r)}} \left(\sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(\mathbf{I}_j^{(r)} - \hat{\mathbf{I}}_j^{(r)} \right)^2 \right) = \\
&= \frac{1}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \frac{\mathbf{d}}{\mathbf{d}\tilde{\alpha}_{\text{const},i}^{(r)}} \left(\sum_{j=1}^3 \left(\mathbf{I}_j^{(r)} - \hat{\mathbf{I}}_j^{(r)} \right)^2 \right) = \\
&= \frac{1}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \frac{\mathbf{d}}{\mathbf{d}\tilde{\alpha}_{\text{const},i}^{(r)}} \left(\left(\mathbf{I}_j^{(r)} - \hat{\mathbf{I}}_j^{(r)} \right)^2 \right) = \\
&= \frac{1}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 2 \left(\mathbf{I}_j^{(r)} - \hat{\mathbf{I}}_j^{(r)} \right) \frac{\mathbf{d}}{\mathbf{d}\tilde{\alpha}_{\text{const},i}^{(r)}} \mathbf{I}_j^{(r)} = \\
&= \frac{2}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(\mathbf{I}_j^{(r)} - \hat{\mathbf{I}}_j^{(r)} \right) \frac{\mathbf{d}}{\mathbf{d}\tilde{\alpha}_{\text{const},i}^{(r)}} \mathbf{I}_j^{(r)} = \\
&= \frac{2}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(\mathbf{I}_j^{(r)} - \hat{\mathbf{I}}_j^{(r)} \right) \frac{\mathbf{d}}{\mathbf{d}\tilde{\alpha}_{\text{const},i}^{(r)}} \text{clamp} \left(\tilde{\mathbf{I}}_j^{(r)}, 0, 1 \right) = \\
&= \frac{2}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(\mathbf{I}_j^{(r)} - \hat{\mathbf{I}}_j^{(r)} \right) \cdot \mathbb{1}_{\tilde{\mathbf{I}}_j^{(r)} \in [0,1]} \cdot \frac{\mathbf{d}\tilde{\mathbf{I}}_j^{(r)}}{\mathbf{d}\tilde{\alpha}_{\text{const},i}^{(r)}} =
\end{aligned}$$

$$= \frac{2}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(I_j^{(r)} - \hat{I}_j^{(r)} \right) \cdot \mathbb{1}_{\tilde{I}_j^{(r)} \in [0,1]} \cdot \frac{d\tilde{I}_j^{(r)}}{d\alpha_i^{(r)}} \cdot \frac{d\alpha_i^{(r)}}{d\tilde{\alpha}_{\text{const},i}^{(r)}}$$

Computing $\frac{d}{d\tilde{\kappa}_i^{(r)}} \mathcal{L}_{\text{RGB,FlaRe}}$

$$\begin{aligned} \frac{d}{d\tilde{\kappa}_i^{(r)}} \mathcal{L}_{\text{RGB,FlaRe}} &= \frac{d}{d\tilde{\kappa}_i^{(r)}} \left(\frac{1}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(I_j^{(r)} - \hat{I}_j^{(r)} \right)^2 \right) = \\ &= \frac{1}{3|\mathcal{B}|} \cdot \frac{d}{d\tilde{\kappa}_i^{(r)}} \left(\sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(I_j^{(r)} - \hat{I}_j^{(r)} \right)^2 \right) = \\ &= \frac{1}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \frac{d}{d\tilde{\kappa}_i^{(r)}} \left(\sum_{j=1}^3 \left(I_j^{(r)} - \hat{I}_j^{(r)} \right)^2 \right) = \\ &= \frac{1}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \frac{d}{d\tilde{\kappa}_i^{(r)}} \left(\left(I_j^{(r)} - \hat{I}_j^{(r)} \right)^2 \right) = \\ &= \frac{1}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 2 \left(I_j^{(r)} - \hat{I}_j^{(r)} \right) \frac{d}{d\tilde{\kappa}_i^{(r)}} I_j^{(r)} = \\ &= \frac{2}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(I_j^{(r)} - \hat{I}_j^{(r)} \right) \frac{d}{d\tilde{\kappa}_i^{(r)}} I_j^{(r)} = \\ &= \frac{2}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(I_j^{(r)} - \hat{I}_j^{(r)} \right) \frac{d}{d\tilde{\kappa}_i^{(r)}} \text{clamp} \left(\tilde{I}_j^{(r)}, 0, 1 \right) = \\ &= \frac{2}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(I_j^{(r)} - \hat{I}_j^{(r)} \right) \cdot \mathbb{1}_{\tilde{I}_j^{(r)} \in [0,1]} \cdot \frac{d\tilde{I}_j^{(r)}}{d\tilde{\kappa}_i^{(r)}} = \\ &= \frac{2}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(I_j^{(r)} - \hat{I}_j^{(r)} \right) \cdot \mathbb{1}_{\tilde{I}_j^{(r)} \in [0,1]} \cdot \frac{d\tilde{I}_j^{(r)}}{d\alpha_i^{(r)}} \cdot \frac{d\alpha_i^{(r)}}{d\tilde{\kappa}_i^{(r)}} \end{aligned}$$

Computing $\frac{d}{dp} \mathcal{L}_{\text{RGB,FlaRe}}$ **for** $p \in \pi_z(\mathcal{G}) \cup \mathcal{F} \cup \Theta \cup \left\{ u_i^{(r)} \right\}_{\substack{r \in \mathcal{B} \\ i \in \{1, \dots, |\mathcal{G}^{(r)}|\}}} \cup \left\{ v_i^{(r)} \right\}_{\substack{r \in \mathcal{B} \\ i \in \{1, \dots, |\mathcal{G}^{(r)}|\}}}$

Let $p \in \pi_z(\mathcal{G}) \cup \mathcal{F} \cup \Theta \cup \left\{ u_i^{(r)} \right\}_{\substack{r \in \mathcal{B} \\ i \in \{1, \dots, |\mathcal{G}^{(r)}|\}}} \cup \left\{ v_i^{(r)} \right\}_{\substack{r \in \mathcal{B} \\ i \in \{1, \dots, |\mathcal{G}^{(r)}|\}}}$. **Then:**

$$\begin{aligned} \frac{d}{dp} \mathcal{L}_{\text{RGB,FlaRe}} &= \frac{d}{dp} \left(\frac{1}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(I_j^{(r)} - \hat{I}_j^{(r)} \right)^2 \right) = \\ &= \frac{1}{3|\mathcal{B}|} \cdot \frac{d}{dp} \left(\sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(I_j^{(r)} - \hat{I}_j^{(r)} \right)^2 \right) = \\ &= \frac{1}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \frac{d}{dp} \left(\sum_{j=1}^3 \left(I_j^{(r)} - \hat{I}_j^{(r)} \right)^2 \right) = \\ &= \frac{1}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \frac{d}{dp} \left(\left(I_j^{(r)} - \hat{I}_j^{(r)} \right)^2 \right) = \end{aligned}$$

$$\begin{aligned}
&= \frac{1}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 2 \left(I_j^{(r)} - \hat{I}_j^{(r)} \right) \frac{d}{dp} I_j^{(r)} = \\
&= \frac{2}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(I_j^{(r)} - \hat{I}_j^{(r)} \right) \frac{d}{dp} I_j^{(r)} = \\
&= \frac{2}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(I_j^{(r)} - \hat{I}_j^{(r)} \right) \frac{d}{dp} \text{clamp} \left(\tilde{I}_j^{(r)}, 0, 1 \right) = \\
&= \frac{2}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(I_j^{(r)} - \hat{I}_j^{(r)} \right) \cdot \mathbb{1}_{\tilde{I}_j^{(r)} \in [0,1]} \cdot \frac{d\tilde{I}_j^{(r)}}{dp}
\end{aligned}$$

Let us put:

$$\delta_{4,i}^{(r)} = \left[\begin{aligned} &\frac{2}{3|\mathcal{B}|} \left(I_1^{(r)} - \hat{I}_1^{(r)} \right) \cdot \mathbb{1}_{\tilde{I}_1^{(r)} \in [0,1]} \cdot c_{\text{const},i,1}^{(r)} \alpha_i^{(r)} T_i^{(r)} \cdot \frac{d}{dp} \Phi_1 \left(\left(u_i^{(r)}, v_i^{(r)} \right), \vec{d}^{(r)}, z_i^{(r)}; \mathcal{G}, \mathcal{F}, \Theta \right) \\ &\frac{2}{3|\mathcal{B}|} \left(I_2^{(r)} - \hat{I}_2^{(r)} \right) \cdot \mathbb{1}_{\tilde{I}_2^{(r)} \in [0,1]} \cdot c_{\text{const},i,2}^{(r)} \alpha_i^{(r)} T_i^{(r)} \cdot \frac{d}{dp} \Phi_2 \left(\left(u_i^{(r)}, v_i^{(r)} \right), \vec{d}^{(r)}, z_i^{(r)}; \mathcal{G}, \mathcal{F}, \Theta \right) \\ &\frac{2}{3|\mathcal{B}|} \left(I_3^{(r)} - \hat{I}_3^{(r)} \right) \cdot \mathbb{1}_{\tilde{I}_3^{(r)} \in [0,1]} \cdot c_{\text{const},i,3}^{(r)} \alpha_i^{(r)} T_i^{(r)} \cdot \frac{d}{dp} \Phi_3 \left(\left(u_i^{(r)}, v_i^{(r)} \right), \vec{d}^{(r)}, z_i^{(r)}; \mathcal{G}, \mathcal{F}, \Theta \right) \\ &\frac{2}{3|\mathcal{B}|} \left(\sum_{j=1}^3 \left(I_j^{(r)} - \hat{I}_j^{(r)} \right) \cdot \mathbb{1}_{\tilde{I}_j^{(r)} \in [0,1]} \cdot \frac{d\tilde{I}_j^{(r)}}{d\alpha_i^{(r)}} \right) \cdot \alpha_{\text{const},i}^{(r)} \cdot \exp \left(- \frac{\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) \cdot \frac{d}{dp} \Phi_4 \left(\left(u_i^{(r)}, v_i^{(r)} \right), \vec{d}^{(r)}, z_i^{(r)}; \mathcal{G}, \mathcal{F}, \Theta \right) \end{aligned} \right]$$

Since:

$$\begin{aligned}
\frac{d\alpha_i^{(r)}}{dp} &= \frac{d}{dp} \left(\alpha_{\text{const},i}^{(r)} \cdot \alpha_{\text{MLP},i}^{(r)} \cdot \exp \left(- \frac{\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) \right) = \\
&= \alpha_{\text{const},i}^{(r)} \cdot \exp \left(- \frac{\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) \cdot \frac{d}{dp} \alpha_{\text{MLP},i}^{(r)} = \\
&= \alpha_{\text{const},i}^{(r)} \cdot \exp \left(- \frac{\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) \cdot \frac{d}{dp} \Phi_4 \left(\left(u_i^{(r)}, v_i^{(r)} \right), \vec{d}^{(r)}, z_i^{(r)}; \mathcal{G}, \mathcal{F}, \Theta \right)
\end{aligned}$$

and:

$$\begin{aligned}
\frac{d\tilde{I}_j^{(r)}}{dc_{\text{MLP},i,j}^{(r)}} &= \frac{d}{dc_{\text{MLP},i,j}^{(r)}} \left(\sum_{k=1}^{|\mathcal{G}^{(r)}|} c_{\text{const},k,j}^{(r)} c_{\text{MLP},k,j}^{(r)} \alpha_k^{(r)} T_k^{(r)} \right) = \\
&= \sum_{k=1}^{|\mathcal{G}^{(r)}|} \frac{d}{dc_{\text{MLP},i,j}^{(r)}} \left(c_{\text{const},k,j}^{(r)} c_{\text{MLP},k,j}^{(r)} \alpha_k^{(r)} T_k^{(r)} \right) = \\
&= \sum_{k=1}^{|\mathcal{G}^{(r)}|} c_{\text{const},k,j}^{(r)} \alpha_k^{(r)} T_k^{(r)} \cdot \frac{dc_{\text{MLP},k,j}^{(r)}}{dc_{\text{MLP},i,j}^{(r)}} = \\
&= \sum_{k=1}^{|\mathcal{G}^{(r)}|} c_{\text{const},k,j}^{(r)} \alpha_k^{(r)} T_k^{(r)} \cdot \delta_{ik} =
\end{aligned}$$

$$= c_{\text{const},i,j}^{(r)} \alpha_i^{(r)} T_i^{(r)}$$

we have:

$$\begin{aligned} \frac{d}{dp} \mathcal{L}_{\text{RGB,Flare}} &= \frac{2}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(I_j^{(r)} - \hat{I}_j^{(r)} \right) \cdot \mathbb{1}_{\tilde{I}_j^{(r)} \in [0,1]} \cdot \frac{d\tilde{I}_j^{(r)}}{dp} = \\ &= \frac{2}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(I_j^{(r)} - \hat{I}_j^{(r)} \right) \cdot \mathbb{1}_{\tilde{I}_j^{(r)} \in [0,1]} \cdot \sum_{k=1}^{|\mathcal{G}^{(r)}|} \left(\frac{d\tilde{I}_j^{(r)}}{d\alpha_k^{(r)}} \cdot \frac{d\alpha_k^{(r)}}{dp} + \frac{d\tilde{I}_j^{(r)}}{dc_{\text{MLP},k,j}^{(r)}} \cdot \frac{dc_{\text{MLP},k,j}^{(r)}}{dp} \right) = \\ &= \sum_{r \in \mathcal{B}} \sum_{k=1}^{|\mathcal{G}^{(r)}|} \left\langle \delta_{4,k}^{(r)}, \frac{d\Phi \left(\left(u_k^{(r)}, v_k^{(r)} \right), \vec{d}^{(r)}, z_k^{(r)}; \mathcal{G}, \mathcal{F}, \Theta \right)}{dp} \right\rangle \end{aligned}$$

Computing $\frac{d}{dp} \mathcal{L}_{\text{RGB,Flare}}$ for $p \in \pi_{\mu}(\mathcal{G}) \cup \pi_{\tilde{s}}(\mathcal{G}) \cup \pi_{\text{q}}(\mathcal{G})$

Let $p \in \pi_{\mu}(\mathcal{G}) \cup \pi_{\tilde{s}}(\mathcal{G}) \cup \pi_{\text{q}}(\mathcal{G})$. Then:

$$\begin{aligned} \frac{d}{dp} \mathcal{L}_{\text{RGB,Flare}} &= \frac{d}{dp} \left(\frac{1}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(I_j^{(r)} - \hat{I}_j^{(r)} \right)^2 \right) = \\ &= \frac{1}{3|\mathcal{B}|} \cdot \frac{d}{dp} \left(\sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(I_j^{(r)} - \hat{I}_j^{(r)} \right)^2 \right) = \\ &= \frac{1}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \frac{d}{dp} \left(\sum_{j=1}^3 \left(I_j^{(r)} - \hat{I}_j^{(r)} \right)^2 \right) = \\ &= \frac{1}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \frac{d}{dp} \left(\left(I_j^{(r)} - \hat{I}_j^{(r)} \right)^2 \right) = \\ &= \frac{1}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 2 \left(I_j^{(r)} - \hat{I}_j^{(r)} \right) \frac{d}{dp} I_j^{(r)} = \\ &= \frac{2}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(I_j^{(r)} - \hat{I}_j^{(r)} \right) \frac{d}{dp} I_j^{(r)} = \\ &= \frac{2}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(I_j^{(r)} - \hat{I}_j^{(r)} \right) \frac{d}{dp} \text{clamp} \left(\tilde{I}_j^{(r)}, 0, 1 \right) = \\ &= \frac{2}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(I_j^{(r)} - \hat{I}_j^{(r)} \right) \cdot \mathbb{1}_{\tilde{I}_j^{(r)} \in [0,1]} \cdot \frac{d\tilde{I}_j^{(r)}}{dp} \end{aligned}$$

Since:

$$\frac{d\alpha_i^{(r)}}{du_i^{(r)}} = \frac{d}{du_i^{(r)}} \left(\alpha_{\text{const},i}^{(r)} \cdot \alpha_{\text{MLP},i}^{(r)} \cdot \exp \left(- \frac{\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) \right) =$$

$$\begin{aligned}
&= \alpha_{\text{const},i}^{(r)} \cdot \alpha_{\text{MLP},i}^{(r)} \cdot \frac{d}{du_i^{(r)}} \exp \left(-\frac{\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) + \\
&+ \alpha_{\text{const},i}^{(r)} \cdot \exp \left(-\frac{\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) \cdot \frac{d}{du_i^{(r)}} \alpha_{\text{MLP},i}^{(r)} =
\end{aligned}$$

$$\begin{aligned}
&= \alpha_{\text{const},i}^{(r)} \cdot \alpha_{\text{MLP},i}^{(r)} \cdot \exp \left(-\frac{\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) \cdot \left(-\frac{1}{2\kappa_i^{(r)}} \right) \cdot \kappa_i^{(r)} \cdot \left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}-1} \cdot 2u_i^{(r)} + \\
&+ \alpha_{\text{const},i}^{(r)} \cdot \exp \left(-\frac{\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) \cdot \frac{d}{du_i^{(r)}} \Phi_4 \left(\left(u_i^{(r)}, v_i^{(r)} \right), \vec{d}^{(r)}, z_i^{(r)} ; \mathcal{G}, \mathcal{F}, \Theta \right) =
\end{aligned}$$

$$\begin{aligned}
&= \alpha_{\text{const},i}^{(r)} \cdot \alpha_{\text{MLP},i}^{(r)} \cdot \exp \left(-\frac{\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) \cdot \left(-\frac{2\kappa_i^{(r)}}{2\kappa_i^{(r)}} \right) \cdot \left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}-1} \cdot u_i^{(r)} + \\
&+ \alpha_{\text{const},i}^{(r)} \cdot \exp \left(-\frac{\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) \cdot \frac{d}{du_i^{(r)}} \Phi_4 \left(\left(u_i^{(r)}, v_i^{(r)} \right), \vec{d}^{(r)}, z_i^{(r)} ; \mathcal{G}, \mathcal{F}, \Theta \right) =
\end{aligned}$$

$$\begin{aligned}
&= -\alpha_{\text{const},i}^{(r)} \cdot \alpha_{\text{MLP},i}^{(r)} \cdot \exp \left(-\frac{\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) \cdot \left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}-1} \cdot u_i^{(r)} + \\
&+ \alpha_{\text{const},i}^{(r)} \cdot \exp \left(-\frac{\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) \cdot \frac{d}{du_i^{(r)}} \Phi_4 \left(\left(u_i^{(r)}, v_i^{(r)} \right), \vec{d}^{(r)}, z_i^{(r)} ; \mathcal{G}, \mathcal{F}, \Theta \right) =
\end{aligned}$$

$$\begin{aligned}
&= -\alpha_i^{(r)} \cdot \left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}-1} \cdot u_i^{(r)} + \\
&+ \alpha_{\text{const},i}^{(r)} \cdot \exp \left(-\frac{\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) \cdot \frac{d}{du_i^{(r)}} \Phi_4 \left(\left(u_i^{(r)}, v_i^{(r)} \right), \vec{d}^{(r)}, z_i^{(r)} ; \mathcal{G}, \mathcal{F}, \Theta \right) =
\end{aligned}$$

and (the derivation is analogous to the above):

$$\begin{aligned}
\frac{d\alpha_i^{(r)}}{dv_i^{(r)}} &= \frac{d}{dv_i^{(r)}} \left(\alpha_{\text{const},i}^{(r)} \cdot \alpha_{\text{MLP},i}^{(r)} \cdot \exp \left(-\frac{\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) \right) = \\
&= -\alpha_i^{(r)} \cdot \left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}-1} \cdot v_i^{(r)} + \\
&+ \alpha_{\text{const},i}^{(r)} \cdot \exp \left(-\frac{\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) \cdot \frac{d}{dv_i^{(r)}} \Phi_4 \left(\left(u_i^{(r)}, v_i^{(r)} \right), \vec{d}^{(r)}, z_i^{(r)} ; \mathcal{G}, \mathcal{F}, \Theta \right) =
\end{aligned}$$

given that:

$$\begin{aligned}
\frac{d\tilde{\Gamma}_j^{(r)}}{dp} &= \sum_{k=1}^{|\mathcal{G}^{(r)}|} \left(\frac{d\tilde{\Gamma}_j^{(r)}}{d\alpha_k^{(r)}} \cdot \left(\frac{d\alpha_k^{(r)}}{du_i^{(r)}} \cdot \frac{du_i^{(r)}}{dp} + \frac{d\alpha_k^{(r)}}{dv_i^{(r)}} \cdot \frac{dv_i^{(r)}}{dp} \right) + \frac{d\tilde{\Gamma}_j^{(r)}}{dc_{\text{MLP},k,j}^{(r)}} \cdot \left(\frac{dc_{\text{MLP},k,j}^{(r)}}{du_i^{(r)}} \cdot \frac{du_i^{(r)}}{dp} + \frac{dc_{\text{MLP},k,j}^{(r)}}{dv_i^{(r)}} \cdot \frac{dv_i^{(r)}}{dp} \right) \right) = \\
&= \sum_{k=1}^{|\mathcal{G}^{(r)}|} \left(\left(\frac{d\tilde{\Gamma}_j^{(r)}}{d\alpha_k^{(r)}} \cdot \frac{d\alpha_k^{(r)}}{du_i^{(r)}} + \frac{d\tilde{\Gamma}_j^{(r)}}{dc_{\text{MLP},k,j}^{(r)}} \cdot \frac{dc_{\text{MLP},k,j}^{(r)}}{du_i^{(r)}} \right) \cdot \frac{du_i^{(r)}}{dp} + \left(\frac{d\tilde{\Gamma}_j^{(r)}}{d\alpha_k^{(r)}} \cdot \frac{d\alpha_k^{(r)}}{dv_i^{(r)}} + \frac{d\tilde{\Gamma}_j^{(r)}}{dc_{\text{MLP},k,j}^{(r)}} \cdot \frac{dc_{\text{MLP},k,j}^{(r)}}{dv_i^{(r)}} \right) \cdot \frac{dv_i^{(r)}}{dp} \right)
\end{aligned}$$

after a few algebraic steps, we obtain:

$$\begin{aligned}
&\frac{d}{dp} \mathcal{L}_{\text{RGB,Flare}} = \\
&= \sum_{r \in \mathcal{B}} \sum_{i=1}^{|\mathcal{G}^{(r)}|} \left\langle \delta_{4,i}^{(r)}, \frac{d\Phi \left(\left(u_i^{(r)}, v_i^{(r)} \right), \vec{d}^{(r)}, z_i^{(r)} ; \mathcal{G}, \mathcal{F}, \Theta \right)}{du_i^{(r)}} \right\rangle \cdot \frac{du_i^{(r)}}{dp} + \\
&+ \sum_{r \in \mathcal{B}} \sum_{i=1}^{|\mathcal{G}^{(r)}|} \left\langle \delta_{4,i}^{(r)}, \frac{d\Phi \left(\left(u_i^{(r)}, v_i^{(r)} \right), \vec{d}^{(r)}, z_i^{(r)} ; \mathcal{G}, \mathcal{F}, \Theta \right)}{dv_i^{(r)}} \right\rangle \cdot \frac{dv_i^{(r)}}{dp} - \\
&- \frac{2}{3|\mathcal{B}|} \sum_{r \in \mathcal{B}} \sum_{j=1}^3 \left(\Gamma_j^{(r)} - \hat{\Gamma}_j^{(r)} \right) \cdot \mathbb{1}_{\tilde{\Gamma}_j^{(r)} \in [0,1]} \cdot \sum_{i=1}^{|\mathcal{G}^{(r)}|} \frac{d\tilde{\Gamma}_j^{(r)}}{d\alpha_i^{(r)}} \cdot \alpha_i^{(r)} \cdot \left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}-1} \cdot \left(u_i^{(r)} \cdot \frac{du_i^{(r)}}{dp} + v_i^{(r)} \cdot \frac{dv_i^{(r)}}{dp} \right)
\end{aligned}$$

Computing $\frac{d\mathcal{L}_s}{d\tilde{s}}$

$$\begin{aligned}
\frac{d\mathcal{L}_s}{d\tilde{s}_{i,j}} &= \frac{d}{d\tilde{s}_{i,j}} \left(\frac{1}{N} \sum_{k=1}^N \|s_k\| \right) = \\
&= \frac{1}{N} \cdot \frac{d}{d\tilde{s}_{i,j}} \left(\sum_{k=1}^N \|s_k\| \right) = \\
&= \frac{1}{N} \cdot \sum_{k=1}^N \frac{d}{d\tilde{s}_{i,j}} \|s_k\| = \\
&= \frac{1}{N} \cdot \sum_{k=1}^N \frac{d}{d\tilde{s}_{i,j}} \sqrt{\langle s_k, s_k \rangle} = \\
&= \frac{1}{N} \cdot \sum_{k=1}^N \frac{1}{2\sqrt{\langle s_k, s_k \rangle}} \cdot \frac{d}{d\tilde{s}_{i,j}} \langle s_k, s_k \rangle = \\
&= \frac{1}{N} \cdot \sum_{k=1}^N \frac{1}{2\sqrt{\langle s_k, s_k \rangle}} \cdot \frac{d}{ds_{i,j}} \langle s_k, s_k \rangle \cdot \frac{ds_{i,j}}{d\tilde{s}_{i,j}} = \\
&= \frac{1}{N} \cdot \sum_{k=1}^N \frac{1}{2\sqrt{\langle s_k, s_k \rangle}} \cdot 2 \left\langle s_k, \frac{ds_k}{ds_{i,j}} \right\rangle \cdot \frac{ds_{i,j}}{d\tilde{s}_{i,j}} = \\
&= \frac{1}{N} \cdot \sum_{k=1}^N \frac{1}{2\sqrt{\langle s_k, s_k \rangle}} \cdot 2 \langle s_k, \delta_{ik} \cdot \vec{\epsilon}_j \rangle \cdot \frac{ds_{i,j}}{d\tilde{s}_{i,j}} = \\
&= \frac{1}{N} \cdot \sum_{k=1}^N \frac{1}{2\sqrt{\langle s_k, s_k \rangle}} \cdot 2 \langle s_k, \vec{\epsilon}_j \rangle \cdot \delta_{ik} \cdot \frac{ds_{i,j}}{d\tilde{s}_{i,j}} = \\
&= \frac{1}{N} \cdot \frac{1}{2\sqrt{\langle s_i, s_i \rangle}} \cdot 2 \langle s_i, \vec{\epsilon}_j \rangle \cdot \frac{ds_{i,j}}{d\tilde{s}_{i,j}} = \\
&= \frac{1}{N} \cdot \frac{1}{2\sqrt{\langle s_i, s_i \rangle}} \cdot 2 \cdot s_{i,j} \cdot \frac{ds_{i,j}}{d\tilde{s}_{i,j}} = \\
&= \frac{1}{N} \cdot \frac{s_{i,j}}{\sqrt{\langle s_i, s_i \rangle}} \cdot \frac{ds_{i,j}}{d\tilde{s}_{i,j}} = \\
&= \frac{1}{N} \cdot \frac{s_{i,j}}{\sqrt{\langle s_i, s_i \rangle}} \cdot \frac{d}{d\tilde{s}_{i,j}} \exp(\tilde{s}_{i,j}) = \\
&= \frac{1}{N} \cdot \frac{s_{i,j}}{\sqrt{\langle s_i, s_i \rangle}} \cdot \exp(\tilde{s}_{i,j}) \cdot 1 = \\
&= \frac{1}{N} \cdot \frac{s_{i,j}}{\sqrt{\langle s_i, s_i \rangle}} \cdot s_{i,j} = \\
&= \frac{1}{N} \cdot \frac{s_{i,j}^2}{\sqrt{\langle s_i, s_i \rangle}}
\end{aligned}$$

Hence:

$$\frac{d\mathcal{L}_s}{d\tilde{s}_i} = \frac{1}{N} \cdot \frac{s_i^2}{\sqrt{\langle s_i, s_i \rangle}}$$

Computing $\frac{d\tilde{l}^{(r)}}{d\alpha_i^{(r)}}$

As for $\frac{d\tilde{l}^{(r)}}{d\alpha_i^{(r)}}$, it can be efficiently computed making use of the following three lemmas:

Lemma 1. Let $r \in \mathcal{B}$. Then:

$$\frac{d\tilde{I}^{(r)}}{d\alpha_1^{(r)}} = \frac{-\tilde{I}^{(r)} + c_1^{(r)}}{1 - \alpha_1^{(r)}}$$

Proof. Since:

$$\tilde{I}^{(r)} = \sum_{i=1}^{|\mathcal{G}^{(r)}|} c_i^{(r)} \alpha_i^{(r)} \prod_{j=1}^{i-1} (1 - \alpha_j^{(r)})$$

we have:

$$\frac{d\tilde{I}^{(r)}}{d\alpha_1^{(r)}} = c_1^{(r)} - \sum_{i=2}^{|\mathcal{G}^{(r)}|} c_i^{(r)} \alpha_i^{(r)} \prod_{\substack{j=1 \\ j \neq 1}}^{i-1} (1 - \alpha_j^{(r)})$$

Let us multiply both sides of the equation by $1 - \alpha_1^{(r)}$. Then:

$$\begin{aligned} \frac{d\tilde{I}^{(r)}}{d\alpha_1^{(r)}} (1 - \alpha_1^{(r)}) &= c_1^{(r)} (1 - \alpha_1^{(r)}) - \sum_{i=2}^{|\mathcal{G}^{(r)}|} c_i^{(r)} \alpha_i^{(r)} \prod_{j=1}^{i-1} (1 - \alpha_j^{(r)}) = \\ &= c_1^{(r)} (1 - \alpha_1^{(r)}) - \left(0 + \sum_{i=2}^{|\mathcal{G}^{(r)}|} c_i^{(r)} \alpha_i^{(r)} \prod_{j=1}^{i-1} (1 - \alpha_j^{(r)}) \right) = \\ &= c_1^{(r)} (1 - \alpha_1^{(r)}) - \left((c_1^{(r)} \alpha_1^{(r)} - c_1^{(r)} \alpha_1^{(r)}) + \sum_{i=2}^{|\mathcal{G}^{(r)}|} c_i^{(r)} \alpha_i^{(r)} \prod_{j=1}^{i-1} (1 - \alpha_j^{(r)}) \right) = \\ &= c_1^{(r)} \left((1 - \alpha_1^{(r)}) + \alpha_1^{(r)} \right) - \sum_{i=1}^{|\mathcal{G}^{(r)}|} c_i^{(r)} \alpha_i^{(r)} \prod_{j=1}^{i-1} (1 - \alpha_j^{(r)}) = \\ &= c_1^{(r)} \cdot 1 - \tilde{I}^{(r)} = \\ &= c_1^{(r)} - \tilde{I}^{(r)} = \\ &= -\tilde{I}^{(r)} + c_1^{(r)} \end{aligned}$$

Hence:

$$\frac{d\tilde{I}^{(r)}}{d\alpha_1^{(r)}} = \frac{-\tilde{I}^{(r)} + c_1^{(r)}}{1 - \alpha_1^{(r)}}$$

□

Lemma 2. Let $r \in \mathcal{B}$ and $1 \leq i < |\mathcal{G}(r)|$. Then:

$$\frac{d\tilde{I}^{(r)}}{d\alpha_{i+1}^{(r)}} = \frac{\frac{d\tilde{I}^{(r)}}{d\alpha_i^{(r)}} \cdot (1 - \alpha_i^{(r)}) + (c_{i+1}^{(r)} - c_i^{(r)}) \prod_{k=1}^i (1 - \alpha_k^{(r)})}{1 - \alpha_{i+1}^{(r)}}$$

Proof. Since:

$$\frac{d\tilde{I}^{(r)}}{d\alpha_i^{(r)}} = c_i^{(r)} \prod_{j=1}^{i-1} (1 - \alpha_j^{(r)}) - \sum_{j=i+1}^{|\mathcal{G}(r)|} c_j^{(r)} \alpha_j^{(r)} \prod_{\substack{k=1 \\ k \neq i}}^{j-1} (1 - \alpha_k^{(r)})$$

after multiplying both sides of the equation by $1 - \alpha_i^{(r)}$ we obtain:

$$\frac{d\tilde{I}^{(r)}}{d\alpha_i^{(r)}} (1 - \alpha_i^{(r)}) = c_i^{(r)} (1 - \alpha_i^{(r)}) \prod_{j=1}^{i-1} (1 - \alpha_j^{(r)}) - \sum_{j=i+1}^{|\mathcal{G}(r)|} c_j^{(r)} \alpha_j^{(r)} \prod_{k=1}^{j-1} (1 - \alpha_k^{(r)}) =$$

$$\begin{aligned}
&= c_i^{(r)} \left(1 - \alpha_i^{(r)}\right) \prod_{j=1}^{i-1} \left(1 - \alpha_j^{(r)}\right) + 0 - \sum_{j=i+1}^{|\mathcal{G}(r)|} c_j^{(r)} \alpha_j^{(r)} \prod_{k=1}^{j-1} \left(1 - \alpha_k^{(r)}\right) = \\
&= c_i^{(r)} \left(1 - \alpha_i^{(r)}\right) \prod_{j=1}^{i-1} \left(1 - \alpha_j^{(r)}\right) + 0 \cdot \prod_{j=1}^{i-1} \left(1 - \alpha_j^{(r)}\right) - \sum_{j=i+1}^{|\mathcal{G}(r)|} c_j^{(r)} \alpha_j^{(r)} \prod_{k=1}^{j-1} \left(1 - \alpha_k^{(r)}\right) = \\
&= c_i^{(r)} \left(1 - \alpha_i^{(r)}\right) \prod_{j=1}^{i-1} \left(1 - \alpha_j^{(r)}\right) + \left(c_i^{(r)} \alpha_i^{(r)} - c_i^{(r)} \alpha_i^{(r)}\right) \prod_{j=1}^{i-1} \left(1 - \alpha_j^{(r)}\right) - \sum_{j=i+1}^{|\mathcal{G}(r)|} c_j^{(r)} \alpha_j^{(r)} \prod_{k=1}^{j-1} \left(1 - \alpha_k^{(r)}\right) = \\
&= c_i^{(r)} \left(\left(1 - \alpha_i^{(r)}\right) + \alpha_i^{(r)}\right) \prod_{j=1}^{i-1} \left(1 - \alpha_j^{(r)}\right) - \sum_{j=i}^{|\mathcal{G}(r)|} c_j^{(r)} \alpha_j^{(r)} \prod_{k=1}^{j-1} \left(1 - \alpha_k^{(r)}\right) = \\
&= c_i^{(r)} \cdot 1 \cdot \prod_{j=1}^{i-1} \left(1 - \alpha_j^{(r)}\right) - \sum_{j=i}^{|\mathcal{G}(r)|} c_j^{(r)} \alpha_j^{(r)} \prod_{k=1}^{j-1} \left(1 - \alpha_k^{(r)}\right) = \\
&= c_i^{(r)} \prod_{j=1}^{i-1} \left(1 - \alpha_j^{(r)}\right) + 0 - \sum_{j=i}^{|\mathcal{G}(r)|} c_j^{(r)} \alpha_j^{(r)} \prod_{k=1}^{j-1} \left(1 - \alpha_k^{(r)}\right) = \\
&= c_i^{(r)} \prod_{j=1}^{i-1} \left(1 - \alpha_j^{(r)}\right) + 0 \cdot \prod_{j=1}^{i-1} \left(1 - \alpha_j^{(r)}\right) - \sum_{j=i}^{|\mathcal{G}(r)|} c_j^{(r)} \alpha_j^{(r)} \prod_{k=1}^{j-1} \left(1 - \alpha_k^{(r)}\right) = \\
&= c_i^{(r)} \prod_{j=1}^{i-1} \left(1 - \alpha_j^{(r)}\right) + \left(c_{i-1}^{(r)} - c_{i-1}^{(r)}\right) \prod_{j=1}^{i-1} \left(1 - \alpha_j^{(r)}\right) - \sum_{j=i}^{|\mathcal{G}(r)|} c_j^{(r)} \alpha_j^{(r)} \prod_{k=1}^{j-1} \left(1 - \alpha_k^{(r)}\right) = \\
&= \left(c_i^{(r)} - c_{i-1}^{(r)}\right) \prod_{j=1}^{i-1} \left(1 - \alpha_j^{(r)}\right) + c_{i-1}^{(r)} \prod_{j=1}^{i-1} \left(1 - \alpha_j^{(r)}\right) - \sum_{j=i}^{|\mathcal{G}(r)|} c_j^{(r)} \alpha_j^{(r)} \prod_{k=1}^{j-1} \left(1 - \alpha_k^{(r)}\right) = \\
&= \left(c_i^{(r)} - c_{i-1}^{(r)}\right) \prod_{j=1}^{i-1} \left(1 - \alpha_j^{(r)}\right) + \left(1 - \alpha_{i-1}^{(r)}\right) \left(c_{i-1}^{(r)} \prod_{j=1}^{i-2} \left(1 - \alpha_j^{(r)}\right) - \sum_{j=i}^{|\mathcal{G}(r)|} c_j^{(r)} \alpha_j^{(r)} \prod_{\substack{k=1 \\ k \neq i-1}}^{j-1} \left(1 - \alpha_k^{(r)}\right)\right) = \\
&= \left(c_i^{(r)} - c_{i-1}^{(r)}\right) \prod_{j=1}^{i-1} \left(1 - \alpha_j^{(r)}\right) + \left(1 - \alpha_{i-1}^{(r)}\right) \frac{d\tilde{I}^{(r)}}{d\alpha_{i-1}^{(r)}} = \\
&= \left(1 - \alpha_{i-1}^{(r)}\right) \frac{d\tilde{I}^{(r)}}{d\alpha_{i-1}^{(r)}} + \left(c_i^{(r)} - c_{i-1}^{(r)}\right) \prod_{j=1}^{i-1} \left(1 - \alpha_j^{(r)}\right) = \\
&= \frac{d\tilde{I}^{(r)}}{d\alpha_{i-1}^{(r)}} \left(1 - \alpha_{i-1}^{(r)}\right) + \left(c_i^{(r)} - c_{i-1}^{(r)}\right) \prod_{j=1}^{i-1} \left(1 - \alpha_j^{(r)}\right)
\end{aligned}$$

Hence:

$$\frac{d\tilde{I}^{(r)}}{d\alpha_i^{(r)}} = \frac{\frac{d\tilde{I}^{(r)}}{d\alpha_{i-1}^{(r)}} \left(1 - \alpha_{i-1}^{(r)}\right) + \left(c_i^{(r)} - c_{i-1}^{(r)}\right) \prod_{j=1}^{i-1} \left(1 - \alpha_j^{(r)}\right)}{1 - \alpha_i^{(r)}}$$

□

Lemma 3. Let $r \in \mathcal{B}$ and $1 \leq i \leq |\mathcal{G}(r)|$. Then:

$$\frac{d\tilde{I}^{(r)}}{d\alpha_i^{(r)}} = \frac{-\tilde{I}^{(r)} + \sum_{j=1}^{i-1} c_j^{(r)} \alpha_j^{(r)} \prod_{k=1}^{j-1} \left(1 - \alpha_k^{(r)}\right) + c_i^{(r)} \prod_{k=1}^{i-1} \left(1 - \alpha_k^{(r)}\right)}{1 - \alpha_i^{(r)}}$$

Proof. We will prove the lemma by induction. As for the base case it immediately follows from Lemma 1. Let us assume then that for some fixed $i \geq 1$ we have:

$$\frac{d\tilde{I}^{(r)}}{d\alpha_i^{(r)}} = \frac{-\tilde{I}^{(r)} + \sum_{j=1}^{i-1} c_j^{(r)} \alpha_j^{(r)} \prod_{k=1}^{j-1} (1 - \alpha_k^{(r)}) + c_i^{(r)} \prod_{k=1}^{i-1} (1 - \alpha_k^{(r)})}{1 - \alpha_i^{(r)}}$$

We will show that:

$$\frac{d\tilde{I}^{(r)}}{d\alpha_{i+1}^{(r)}} = \frac{-\tilde{I}^{(r)} + \sum_{j=1}^i c_j^{(r)} \alpha_j^{(r)} \prod_{k=1}^{j-1} (1 - \alpha_k^{(r)}) + c_{i+1}^{(r)} \prod_{k=1}^i (1 - \alpha_k^{(r)})}{1 - \alpha_{i+1}^{(r)}}$$

Rewriting the RHS yields:

$$\begin{aligned} & \frac{-\tilde{I}^{(r)} + \sum_{j=1}^i c_j^{(r)} \alpha_j^{(r)} \prod_{k=1}^{j-1} (1 - \alpha_k^{(r)}) + c_{i+1}^{(r)} \prod_{k=1}^i (1 - \alpha_k^{(r)})}{1 - \alpha_{i+1}^{(r)}} = \\ & = \frac{-\tilde{I}^{(r)} + \left(\sum_{j=1}^{i-1} c_j^{(r)} \alpha_j^{(r)} \prod_{k=1}^{j-1} (1 - \alpha_k^{(r)}) + c_i^{(r)} \alpha_i^{(r)} \prod_{k=1}^{i-1} (1 - \alpha_k^{(r)}) \right) + c_{i+1}^{(r)} \prod_{k=1}^i (1 - \alpha_k^{(r)}) + 0}{1 - \alpha_{i+1}^{(r)}} = \\ & = \frac{-\tilde{I}^{(r)} + \sum_{j=1}^{i-1} c_j^{(r)} \alpha_j^{(r)} \prod_{k=1}^{j-1} (1 - \alpha_k^{(r)}) + c_i^{(r)} \alpha_i^{(r)} \prod_{k=1}^{i-1} (1 - \alpha_k^{(r)}) + c_{i+1}^{(r)} \prod_{k=1}^i (1 - \alpha_k^{(r)}) + 0 \prod_{k=1}^{i-1} (1 - \alpha_k^{(r)})}{1 - \alpha_{i+1}^{(r)}} = \\ & = \frac{-\tilde{I}^{(r)} + \sum_{j=1}^{i-1} c_j^{(r)} \alpha_j^{(r)} \prod_{k=1}^{j-1} (1 - \alpha_k^{(r)}) + c_i^{(r)} \alpha_i^{(r)} \prod_{k=1}^{i-1} (1 - \alpha_k^{(r)}) + c_{i+1}^{(r)} \prod_{k=1}^i (1 - \alpha_k^{(r)}) + (c_i^{(r)} - c_i^{(r)}) \prod_{k=1}^{i-1} (1 - \alpha_k^{(r)})}{1 - \alpha_{i+1}^{(r)}} = \\ & = \frac{-\tilde{I}^{(r)} + \sum_{j=1}^{i-1} c_j^{(r)} \alpha_j^{(r)} \prod_{k=1}^{j-1} (1 - \alpha_k^{(r)}) + c_i^{(r)} \prod_{k=1}^{i-1} (1 - \alpha_k^{(r)}) + (c_i^{(r)} \alpha_i^{(r)} - c_i^{(r)}) \prod_{k=1}^{i-1} (1 - \alpha_k^{(r)}) + c_{i+1}^{(r)} \prod_{k=1}^i (1 - \alpha_k^{(r)})}{1 - \alpha_{i+1}^{(r)}} = \\ & = \frac{-\tilde{I}^{(r)} + \sum_{j=1}^{i-1} c_j^{(r)} \alpha_j^{(r)} \prod_{k=1}^{j-1} (1 - \alpha_k^{(r)}) + c_i^{(r)} \prod_{k=1}^{i-1} (1 - \alpha_k^{(r)}) - c_i^{(r)} (1 - \alpha_i^{(r)}) \prod_{k=1}^{i-1} (1 - \alpha_k^{(r)}) + c_{i+1}^{(r)} (1 - \alpha_i^{(r)}) \prod_{k=1}^{i-1} (1 - \alpha_k^{(r)})}{1 - \alpha_{i+1}^{(r)}} = \\ & = \frac{-\tilde{I}^{(r)} + \sum_{j=1}^{i-1} c_j^{(r)} \alpha_j^{(r)} \prod_{k=1}^{j-1} (1 - \alpha_k^{(r)}) + c_i^{(r)} \prod_{k=1}^{i-1} (1 - \alpha_k^{(r)}) + (-c_i^{(r)} (1 - \alpha_i^{(r)}) + c_{i+1}^{(r)} (1 - \alpha_i^{(r)})) \prod_{k=1}^{i-1} (1 - \alpha_k^{(r)})}{1 - \alpha_{i+1}^{(r)}} = \\ & = \frac{-\tilde{I}^{(r)} + \sum_{j=1}^{i-1} c_j^{(r)} \alpha_j^{(r)} \prod_{k=1}^{j-1} (1 - \alpha_k^{(r)}) + c_i^{(r)} \prod_{k=1}^{i-1} (1 - \alpha_k^{(r)})}{1 - \alpha_{i+1}^{(r)}} + \frac{(c_{i+1}^{(r)} - c_i^{(r)}) (1 - \alpha_i^{(r)}) \prod_{k=1}^{i-1} (1 - \alpha_k^{(r)})}{1 - \alpha_{i+1}^{(r)}} = \\ & = \frac{-\tilde{I}^{(r)} + \sum_{j=1}^{i-1} c_j^{(r)} \alpha_j^{(r)} \prod_{k=1}^{j-1} (1 - \alpha_k^{(r)}) + c_i^{(r)} \prod_{k=1}^{i-1} (1 - \alpha_k^{(r)})}{1 - \alpha_{i+1}^{(r)}} \cdot 1 + \frac{(c_{i+1}^{(r)} - c_i^{(r)}) \prod_{k=1}^i (1 - \alpha_k^{(r)})}{1 - \alpha_{i+1}^{(r)}} = \\ & = \frac{-\tilde{I}^{(r)} + \sum_{j=1}^{i-1} c_j^{(r)} \alpha_j^{(r)} \prod_{k=1}^{j-1} (1 - \alpha_k^{(r)}) + c_i^{(r)} \prod_{k=1}^{i-1} (1 - \alpha_k^{(r)})}{1 - \alpha_{i+1}^{(r)}} \cdot \frac{1 - \alpha_i^{(r)}}{1 - \alpha_i^{(r)}} + \frac{(c_{i+1}^{(r)} - c_i^{(r)}) \prod_{k=1}^i (1 - \alpha_k^{(r)})}{1 - \alpha_{i+1}^{(r)}} = \end{aligned}$$

$$= \frac{-\tilde{I}^{(r)} + \sum_{j=1}^{i-1} c_j^{(r)} \alpha_j^{(r)} \prod_{k=1}^{j-1} (1 - \alpha_k^{(r)}) + c_i^{(r)} \prod_{k=1}^{i-1} (1 - \alpha_k^{(r)})}{1 - \alpha_i^{(r)}} \cdot \frac{1 - \alpha_i^{(r)}}{1 - \alpha_{i+1}^{(r)}} + \frac{(c_{i+1}^{(r)} - c_i^{(r)}) \prod_{k=1}^i (1 - \alpha_k^{(r)})}{1 - \alpha_{i+1}^{(r)}}$$

By inductive hypothesis we get:

$$\frac{-\tilde{I}^{(r)} + \sum_{j=1}^i c_j^{(r)} \alpha_j^{(r)} \prod_{k=1}^{j-1} (1 - \alpha_k^{(r)}) + c_{i+1}^{(r)} \prod_{k=1}^i (1 - \alpha_k^{(r)})}{1 - \alpha_i^{(r)}} = \frac{d\tilde{I}^{(r)}}{d\alpha_i^{(r)}}$$

Therefore:

$$\begin{aligned} & \frac{-\tilde{I}^{(r)} + \sum_{j=1}^i c_j^{(r)} \alpha_j^{(r)} \prod_{k=1}^{j-1} (1 - \alpha_k^{(r)}) + c_{i+1}^{(r)} \prod_{k=1}^i (1 - \alpha_k^{(r)})}{1 - \alpha_{i+1}^{(r)}} = \\ &= \frac{d\tilde{I}^{(r)}}{d\alpha_i^{(r)}} \cdot \frac{1 - \alpha_i^{(r)}}{1 - \alpha_{i+1}^{(r)}} + \frac{(c_{i+1}^{(r)} - c_i^{(r)}) \prod_{k=1}^i (1 - \alpha_k^{(r)})}{1 - \alpha_{i+1}^{(r)}} = \\ &= \frac{\frac{d\tilde{I}^{(r)}}{d\alpha_i^{(r)}} \cdot (1 - \alpha_i^{(r)}) + (c_{i+1}^{(r)} - c_i^{(r)}) \prod_{k=1}^i (1 - \alpha_k^{(r)})}{1 - \alpha_{i+1}^{(r)}} \end{aligned}$$

Thus by Lemma 2:

$$\frac{-\tilde{I}^{(r)} + \sum_{j=1}^i c_j^{(r)} \alpha_j^{(r)} \prod_{k=1}^{j-1} (1 - \alpha_k^{(r)}) + c_{i+1}^{(r)} \prod_{k=1}^i (1 - \alpha_k^{(r)})}{1 - \alpha_{i+1}^{(r)}} = \frac{d\tilde{I}^{(r)}}{d\alpha_{i+1}^{(r)}}$$

Consequently by mathematical induction, the Lemma 3 holds for all $i \leq |\mathcal{G}(r)|$. $\square$

Finally, let us note that while computing $\frac{d\tilde{I}^{(r)}}{d\alpha_i^{(r)}}$ for each $i \in \{1, \dots, |\mathcal{G}(r)|\}$ in the naïve manner directly from the gradient formula:

$$\frac{d\tilde{I}^{(r)}}{d\alpha_i^{(r)}} = c_i^{(r)} \prod_{j=1}^{i-1} (1 - \alpha_j^{(r)}) - \sum_{j=i+1}^{|\mathcal{G}(r)|} c_j^{(r)} \alpha_j^{(r)} \prod_{\substack{k=1 \\ k \neq i}}^{j-1} (1 - \alpha_k^{(r)})$$

requires $\mathcal{O}(N)$ operations for single $i \in \{1, \dots, |\mathcal{G}(r)|\}$, by employing the above lemmas along with a single prefix sum and a single prefix product, we can perform each step in constant time $\mathcal{O}(1)$, yielding a total of $\mathcal{O}(N)$ steps for the entire ray. Finally, let us note that the formula given by Lemma 3 is undefined when $\alpha_i^{(r)} = 1$. However, since in our algorithm we terminate the ray whenever the transmittance $T_i^{(r)}$ falls below a certain configurable threshold ε , we assume that the i -th intersected planar Gaussian primitive for which $T_i^{(r)} < \varepsilon$ is the last primitive intersected by the ray and apply to it the regular formula above, which does not require division by $1 - \alpha_i^{(r)}$, thereby protecting us against division by zero.

Computing $\frac{d\alpha_i^{(r)}}{d\tilde{\alpha}_{\text{const},i}^{(r)}}$

$$\begin{aligned} \frac{d\alpha_i^{(r)}}{d\tilde{\alpha}_{\text{const},i}^{(r)}} &= \frac{d}{d\tilde{\alpha}_{\text{const},i}^{(r)}} \left(\alpha_{\text{const},i}^{(r)} \cdot \alpha_{\text{MLP},i}^{(r)} \cdot \exp \left(- \frac{\left((u_i^{(r)})^2 + (v_i^{(r)})^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) \right) = \\ &= \alpha_{\text{MLP},i}^{(r)} \cdot \exp \left(- \frac{\left((u_i^{(r)})^2 + (v_i^{(r)})^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) \cdot \frac{d\alpha_{\text{const},i}^{(r)}}{d\tilde{\alpha}_{\text{const},i}^{(r)}} = \end{aligned}$$

$$\begin{aligned}
&= \alpha_{\text{MLP},i}^{(r)} \cdot \exp \left(-\frac{\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) \cdot \frac{d}{d\tilde{\alpha}_{\text{const},i}^{(r)}} \sigma \left(\tilde{\alpha}_{\text{const},i}^{(r)} \right) = \\
&= \alpha_{\text{MLP},i}^{(r)} \cdot \exp \left(-\frac{\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) \cdot \frac{d}{d\tilde{\alpha}_{\text{const},i}^{(r)}} \left(\frac{1}{1 + e^{-\tilde{\alpha}_{\text{const},i}^{(r)}}} \right) = \\
&= \alpha_{\text{MLP},i}^{(r)} \cdot \exp \left(-\frac{\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) \cdot \left(-\frac{1}{\left(1 + e^{-\tilde{\alpha}_{\text{const},i}^{(r)}} \right)^2} \right) \cdot \left(-e^{-\tilde{\alpha}_{\text{const},i}^{(r)}} \right) = \\
&= \alpha_{\text{MLP},i}^{(r)} \cdot \exp \left(-\frac{\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) \cdot \frac{e^{-\tilde{\alpha}_{\text{const},i}^{(r)}}}{\left(1 + e^{-\tilde{\alpha}_{\text{const},i}^{(r)}} \right)^2} = \\
&= \alpha_{\text{MLP},i}^{(r)} \cdot \exp \left(-\frac{\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) \cdot \frac{e^{-\tilde{\alpha}_{\text{const},i}^{(r)}} + 0}{\left(1 + e^{-\tilde{\alpha}_{\text{const},i}^{(r)}} \right)^2} = \\
&= \alpha_{\text{MLP},i}^{(r)} \cdot \exp \left(-\frac{\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) \cdot \frac{e^{-\tilde{\alpha}_{\text{const},i}^{(r)}} + (1 - 1)}{\left(1 + e^{-\tilde{\alpha}_{\text{const},i}^{(r)}} \right)^2} = \\
&= \alpha_{\text{MLP},i}^{(r)} \cdot \exp \left(-\frac{\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) \cdot \frac{\left(1 + e^{-\tilde{\alpha}_{\text{const},i}^{(r)}} \right) - 1}{\left(1 + e^{-\tilde{\alpha}_{\text{const},i}^{(r)}} \right)^2} = \\
&= \alpha_{\text{MLP},i}^{(r)} \cdot \exp \left(-\frac{\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) \cdot \left(\frac{1}{1 + e^{-\tilde{\alpha}_{\text{const},i}^{(r)}}} - \frac{1}{\left(1 + e^{-\tilde{\alpha}_{\text{const},i}^{(r)}} \right)^2} \right) = \\
&= \alpha_{\text{MLP},i}^{(r)} \cdot \exp \left(-\frac{\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) \cdot \frac{1}{1 + e^{-\tilde{\alpha}_{\text{const},i}^{(r)}}} \cdot \left(1 - \frac{1}{1 + e^{-\tilde{\alpha}_{\text{const},i}^{(r)}}} \right) = \\
&= \alpha_{\text{MLP},i}^{(r)} \cdot \exp \left(-\frac{\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) \cdot \sigma \left(\tilde{\alpha}_{\text{const},i}^{(r)} \right) \cdot \left(1 - \sigma \left(\tilde{\alpha}_{\text{const},i}^{(r)} \right) \right) =
\end{aligned}$$

$$\begin{aligned}
&= \alpha_{\text{MLP},i}^{(r)} \cdot \exp \left(-\frac{\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) \cdot \alpha_{\text{const},i}^{(r)} \cdot \left(1 - \alpha_{\text{const},i}^{(r)} \right) = \\
&= \alpha_{\text{const},i}^{(r)} \cdot \alpha_{\text{MLP},i}^{(r)} \cdot \exp \left(-\frac{\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) \cdot \left(1 - \alpha_{\text{const},i}^{(r)} \right) = \\
&= \alpha_i^{(r)} \cdot \left(1 - \alpha_{\text{const},i}^{(r)} \right)
\end{aligned}$$

Computing $\frac{d\alpha_i^{(r)}}{d\tilde{\kappa}_i^{(r)}}$

$$\begin{aligned}
\frac{d\alpha_i^{(r)}}{d\tilde{\kappa}_i^{(r)}} &= \\
&= \frac{d}{d\tilde{\kappa}_i^{(r)}} \left(\alpha_{\text{const},i}^{(r)} \cdot \alpha_{\text{MLP},i}^{(r)} \cdot \exp \left(-\frac{\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) \right) = \\
&= \alpha_{\text{const},i}^{(r)} \cdot \alpha_{\text{MLP},i}^{(r)} \cdot \frac{d}{d\tilde{\kappa}_i^{(r)}} \exp \left(-\frac{\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) = \\
&= \alpha_{\text{const},i}^{(r)} \cdot \alpha_{\text{MLP},i}^{(r)} \cdot \exp \left(-\frac{\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) \cdot \frac{d}{d\tilde{\kappa}_i^{(r)}} \left(-\frac{\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) = \\
&= -\alpha_i^{(r)} \cdot \frac{d}{d\tilde{\kappa}_i^{(r)}} \left(\frac{\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \right) = \\
&= -\alpha_i^{(r)} \cdot \left(\frac{\frac{d}{d\tilde{\kappa}_i^{(r)}} \left(\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}} \right) \cdot \left(2\kappa_i^{(r)} \right) - \left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}} \cdot \frac{d}{d\tilde{\kappa}_i^{(r)}} \left(2\kappa_i^{(r)} \right)}{\left(2\kappa_i^{(r)} \right)^2} \right) = \\
&= -\alpha_i^{(r)} \cdot \frac{\left(\left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}} \cdot \ln \left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right) \cdot \frac{d\kappa_i^{(r)}}{d\tilde{\kappa}_i^{(r)}} \right) \cdot \left(2\kappa_i^{(r)} \right) - \left(2 \cdot \left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}} \cdot \frac{d\kappa_i^{(r)}}{d\tilde{\kappa}_i^{(r)}} \right)}{4 \left(\kappa_i^{(r)} \right)^2} = \\
&= -\alpha_i^{(r)} \cdot \frac{2\kappa_i^{(r)} \cdot \left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}} \cdot \ln \left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right) \cdot \frac{d\kappa_i^{(r)}}{d\tilde{\kappa}_i^{(r)}} - 2 \cdot \left(\left(u_i^{(r)} \right)^2 + \left(v_i^{(r)} \right)^2 \right)^{\kappa_i^{(r)}} \cdot \frac{d\kappa_i^{(r)}}{d\tilde{\kappa}_i^{(r)}}}{4 \left(\kappa_i^{(r)} \right)^2} =
\end{aligned}$$

$$\begin{aligned}
&= -\alpha_i^{(r)} \cdot \frac{\kappa_i^{(r)} \cdot \left((u_i^{(r)})^2 + (v_i^{(r)})^2 \right)^{\kappa_i^{(r)}} \cdot \ln \left((u_i^{(r)})^2 + (v_i^{(r)})^2 \right) \cdot \frac{d\kappa_i^{(r)}}{d\tilde{\kappa}_i^{(r)}} - \left((u_i^{(r)})^2 + (v_i^{(r)})^2 \right)^{\kappa_i^{(r)}} \cdot \frac{d\kappa_i^{(r)}}{d\tilde{\kappa}_i^{(r)}}}{2 \cdot \left(\kappa_i^{(r)} \right)^2} = \\
&= \alpha_i^{(r)} \cdot \frac{\left((u_i^{(r)})^2 + (v_i^{(r)})^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \cdot \left(\frac{1}{\kappa_i^{(r)}} - \ln \left((u_i^{(r)})^2 + (v_i^{(r)})^2 \right) \right) \cdot \frac{d\kappa_i^{(r)}}{d\tilde{\kappa}_i^{(r)}} = \\
&= \alpha_i^{(r)} \cdot \frac{\left((u_i^{(r)})^2 + (v_i^{(r)})^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \cdot \left(\frac{1}{\kappa_i^{(r)}} - \ln \left((u_i^{(r)})^2 + (v_i^{(r)})^2 \right) \right) \cdot \frac{d}{d\tilde{\kappa}_i^{(r)}} \left(\ln \left(1 + \exp \left(\tilde{\kappa}_i^{(r)} \right) \right) \right) = \\
&= \alpha_i^{(r)} \cdot \frac{\left((u_i^{(r)})^2 + (v_i^{(r)})^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \cdot \left(\frac{1}{\kappa_i^{(r)}} - \ln \left((u_i^{(r)})^2 + (v_i^{(r)})^2 \right) \right) \cdot \frac{\exp \left(\tilde{\kappa}_i^{(r)} \right)}{1 + \exp \left(\tilde{\kappa}_i^{(r)} \right)} = \\
&= \alpha_i^{(r)} \cdot \frac{\left((u_i^{(r)})^2 + (v_i^{(r)})^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \cdot \left(\frac{1}{\kappa_i^{(r)}} - \ln \left((u_i^{(r)})^2 + (v_i^{(r)})^2 \right) \right) \cdot \frac{\exp \left(\tilde{\kappa}_i^{(r)} \right)}{1 + \exp \left(\tilde{\kappa}_i^{(r)} \right)} \cdot 1 = \\
&= \alpha_i^{(r)} \cdot \frac{\left((u_i^{(r)})^2 + (v_i^{(r)})^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \cdot \left(\frac{1}{\kappa_i^{(r)}} - \ln \left((u_i^{(r)})^2 + (v_i^{(r)})^2 \right) \right) \cdot \frac{\exp \left(\tilde{\kappa}_i^{(r)} \right)}{1 + \exp \left(\tilde{\kappa}_i^{(r)} \right)} \cdot \frac{\frac{1}{\exp \left(\tilde{\kappa}_i^{(r)} \right)}}{\frac{1}{\exp \left(\tilde{\kappa}_i^{(r)} \right)}} = \\
&= \alpha_i^{(r)} \cdot \frac{\left((u_i^{(r)})^2 + (v_i^{(r)})^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \cdot \left(\frac{1}{\kappa_i^{(r)}} - \ln \left((u_i^{(r)})^2 + (v_i^{(r)})^2 \right) \right) \cdot \frac{1}{\frac{1}{\exp \left(\tilde{\kappa}_i^{(r)} \right)} + 1} = \\
&= \alpha_i^{(r)} \cdot \frac{\left((u_i^{(r)})^2 + (v_i^{(r)})^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \cdot \left(\frac{1}{\kappa_i^{(r)}} - \ln \left((u_i^{(r)})^2 + (v_i^{(r)})^2 \right) \right) \cdot \frac{1}{\exp \left(-\tilde{\kappa}_i^{(r)} \right) + 1} = \\
&= \alpha_i^{(r)} \cdot \frac{\left((u_i^{(r)})^2 + (v_i^{(r)})^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \cdot \left(\frac{1}{\kappa_i^{(r)}} - \ln \left((u_i^{(r)})^2 + (v_i^{(r)})^2 \right) \right) \cdot \frac{1}{1 + \exp \left(-\tilde{\kappa}_i^{(r)} \right)} = \\
&= \alpha_i^{(r)} \cdot \frac{\left((u_i^{(r)})^2 + (v_i^{(r)})^2 \right)^{\kappa_i^{(r)}}}{2\kappa_i^{(r)}} \cdot \left(\frac{1}{\kappa_i^{(r)}} - \ln \left((u_i^{(r)})^2 + (v_i^{(r)})^2 \right) \right) \cdot \sigma \left(\tilde{\kappa}_i^{(r)} \right) =
\end{aligned}$$

Computing MLP backpropagation gradients

As detailed in the Subsection **Global auto-decoder**:

$$\Phi \left((u_i^{(r)}, v_i^{(r)}), \vec{d}^{(r)}, z_i^{(r)}; \mathcal{G}, \mathcal{F}, \Theta \right) = W_3 \varphi_3 \left(W_2 \varphi_2 \left(W_1 \varphi_1 \left(\begin{bmatrix} E_{\text{LUT}}(u_i^{(r)}, v_i^{(r)}) \\ \gamma(\vec{d}^{(r)}) \\ z_i^{(r)} \end{bmatrix} \right) + b_1 \right) + b_2 \right) + b_3$$

For brevity, we will omit both the superscript (r) (where $r \in \mathcal{B}$ specifies the particular ray from the batch $\mathcal{B}$) and the subscript i (denoting the index of the Gaussian primitive hit by ray r). Let us put:

$$\mathbf{x}_0 := \begin{bmatrix} E_{\text{LUT}}(u, v) \\ \gamma(\vec{d}) \\ z \end{bmatrix}$$

$$\begin{aligned}
h_1 &:= W_1 \varphi_1(x_0) + b_1, \quad x_1 := \varphi_1(h_1) \\
h_2 &:= W_2 \varphi_2(x_1) + b_2, \quad x_2 := \varphi_2(h_2) \\
h_3 &:= W_3 \varphi_3(x_2) + b_3, \quad x_3 := \varphi_3(h_3)
\end{aligned}$$

and:

$$\begin{aligned}
\delta_3 &:= W_3^T (\delta_4 \odot \varphi'_3(h_3)) \\
\delta_2 &:= W_2^T (\delta_3 \odot \varphi'_2(h_2)) \\
\delta_1 &:= W_1^T (\delta_2 \odot \varphi'_1(h_1))
\end{aligned}$$

Before deriving formulas for the inner products: $\left\langle \delta_4, \frac{d\Phi}{dW_3} \right\rangle$, $\left\langle \delta_4, \frac{d\Phi}{db_3} \right\rangle$, $\left\langle \delta_4, \frac{d\Phi}{dW_2} \right\rangle$, $\left\langle \delta_4, \frac{d\Phi}{db_2} \right\rangle$, $\left\langle \delta_4, \frac{d\Phi}{dW_1} \right\rangle$, $\left\langle \delta_4, \frac{d\Phi}{db_1} \right\rangle$, $\left\langle \delta_4, \frac{d\Phi}{dz} \right\rangle$, $\left\langle \delta_4, \frac{d\Phi}{d\mathcal{F}} \right\rangle$ that define the gradients of the loss function $\mathcal{L}$ with respect the parameters: $W_3, b_3, W_2, b_2, W_1, b_1, z, \mathcal{F}$ we will show how to compute the auxiliary identities: $\left\langle \delta_4, \frac{d\Phi}{dx_3} \right\rangle$, $\left\langle \delta_4, \frac{d\Phi}{dx_2} \right\rangle$, $\left\langle \delta_4, \frac{d\Phi}{dx_1} \right\rangle$ and $\left\langle \delta_4, \frac{d\Phi}{dx_0} \right\rangle$. Before doing so, however, we will state and prove a lemma that, as we shall see later, will prove useful in deriving the relevant formulas.

Lemma 4. *Let $a, b, c \in \mathbb{R}^n$. Then the mixed Hadamard product: $\langle a, b \odot c \rangle$ is invariant under any permutation of the arguments: a, b , and c .*

Proof.

$$\begin{aligned}
\langle a, b \odot c \rangle &= \sum_{i=1}^n a_i (b \odot c)_i = \sum_{i=1}^n a_i (b_i \cdot c_i) = \sum_{i=1}^n a_i \cdot b_i \cdot c_i = \sum_{i=1}^n b_i \cdot c_i \cdot a_i = \sum_{i=1}^n b_i \cdot (c_i \cdot a_i) = \\
&= \sum_{i=1}^n b_i \cdot (c \odot a)_i = \langle b, c \odot a \rangle
\end{aligned}$$

On the other hand:

$$\begin{aligned}
\langle a, b \odot c \rangle &= \sum_{i=1}^n a_i (b \odot c)_i = \sum_{i=1}^n a_i (b_i \cdot c_i) = \sum_{i=1}^n a_i \cdot b_i \cdot c_i = \sum_{i=1}^n c_i \cdot a_i \cdot b_i = \sum_{i=1}^n c_i \cdot (a_i \cdot b_i) = \\
&= \sum_{i=1}^n c_i \cdot (a \odot b)_i = \langle c, a \odot b \rangle
\end{aligned}$$

The proof of invariance under the remaining three possible permutations is trivial and follows immediately from the commutativity of the Hadamard product. $\square$

Computing $\left\langle \delta_4, \frac{d\Phi}{dx_3} \right\rangle$

$$\left\langle \delta_4, \frac{d\Phi}{dx_{3,i}} \right\rangle = \left\langle \delta_4, \frac{dx_3}{dx_{3,i}} \right\rangle = \langle \delta_4, \vec{e}_i \rangle = \delta_{4,i}$$

Hence:

$$\left\langle \delta_4, \frac{d\Phi}{dx_3} \right\rangle = \delta_4$$

Computing $\left\langle \delta_4, \frac{d\Phi}{dx_2} \right\rangle$

$$\begin{aligned}
\left\langle \delta_4, \frac{d\Phi}{dx_{2,i}} \right\rangle &= \left\langle \delta_4, \frac{dx_3}{dx_{2,i}} \right\rangle = \\
&= \left\langle \delta_4, \frac{d}{dx_{2,i}} (\varphi_3(h_3)) \right\rangle = \\
&= \left\langle \delta_4, \varphi'_3(h_3) \odot \frac{dh_3}{dx_{2,i}} \right\rangle = \\
&= \left\langle \delta_4, \varphi'_3(h_3) \odot \frac{d}{dx_{2,i}} (W_3 x_2 + b_3) \right\rangle =
\end{aligned}$$

$$\begin{aligned}
&= \left\langle \delta_4, \varphi'_3(h_3) \odot \left(W_3 \frac{dx_2}{dx_{2,i}} \right) \right\rangle = \\
&= \langle \delta_4, \varphi'_3(h_3) \odot (W_3 \vec{\epsilon}_i) \rangle
\end{aligned}$$

By Lemma 4 we get:

$$\begin{aligned}
\left\langle \delta_4, \frac{d\Phi}{dx_{2,i}} \right\rangle &= \langle \delta_4 \odot \varphi'_3(h_3), W_3 \vec{\epsilon}_i \rangle = \\
&= (W_3 \vec{\epsilon}_i)^T (\delta_4 \odot \varphi'_3(h_3)) = \\
&= \vec{\epsilon}_i^T W_3^T (\delta_4 \odot \varphi'_3(h_3)) = \\
&= \vec{\epsilon}_i^T \delta_3 = \\
&= \delta_{3,i}
\end{aligned}$$

Hence:

$$\left\langle \delta_4, \frac{d\Phi}{dx_2} \right\rangle = \delta_3$$

Computing $\left\langle \delta_4, \frac{d\Phi}{dx_1} \right\rangle$

$$\begin{aligned}
\left\langle \delta_4, \frac{d\Phi}{dx_{1,j}} \right\rangle &= \left\langle \delta_4, \sum_{i=1}^{64} \frac{d\Phi}{dx_{2,i}} \cdot \frac{dx_{2,i}}{dx_{1,j}} \right\rangle = \\
&= \sum_{i=1}^{64} \left\langle \delta_4, \frac{d\Phi}{dx_{2,i}} \cdot \frac{dx_{2,i}}{dx_{1,j}} \right\rangle = \\
&= \sum_{i=1}^{64} \left\langle \delta_4, \frac{d\Phi}{dx_{2,i}} \right\rangle \cdot \frac{dx_{2,i}}{dx_{1,j}} = \\
&= \sum_{i=1}^{64} \delta_{3,i} \cdot \frac{dx_{2,i}}{dx_{1,j}} = \\
&= \sum_{i=1}^{64} \delta_{3,i} \cdot \frac{d}{dx_{1,j}} (\varphi_2(h_2))_i = \\
&= \sum_{i=1}^{64} \delta_{3,i} \cdot \left(\varphi'_2(h_2) \odot \frac{dh_2}{dx_{1,j}} \right)_i = \\
&= \sum_{i=1}^{64} \delta_{3,i} \cdot \left(\varphi'_2(h_2) \odot \frac{d}{dx_{1,j}} (W_2 x_1 + b_2) \right)_i = \\
&= \sum_{i=1}^{64} \delta_{3,i} \cdot \left(\varphi'_2(h_2) \odot \left(W_2 \frac{dx_1}{dx_{1,j}} \right) \right)_i = \\
&= \sum_{i=1}^{64} \delta_{3,i} \cdot (\varphi'_2(h_2) \odot (W_2 \vec{\epsilon}_j))_i = \\
&= \langle \delta_3, \varphi'_2(h_2) \odot (W_2 \vec{\epsilon}_j) \rangle
\end{aligned}$$

By Lemma 4 we get:

$$\begin{aligned}
\left\langle \delta_4, \frac{d\Phi}{dx_{1,j}} \right\rangle &= \langle \delta_3 \odot \varphi'_2(h_2), W_2 \vec{\epsilon}_j \rangle = \\
&= (W_2 \vec{\epsilon}_j)^T (\delta_3 \odot \varphi'_2(h_2)) = \\
&= \vec{\epsilon}_j^T W_2^T (\delta_3 \odot \varphi'_2(h_2)) = \\
&= \vec{\epsilon}_j^T \delta_2 =
\end{aligned}$$

$$= \delta_{2,j}$$

Hence:

$$\left\langle \delta_4, \frac{d\Phi}{dx_1} \right\rangle = \delta_2$$

Computing $\left\langle \delta_4, \frac{d\Phi}{dx_0} \right\rangle$

$$\begin{aligned} \left\langle \delta_4, \frac{d\Phi}{dx_{0,j}} \right\rangle &= \left\langle \delta_4, \sum_{i=1}^{64} \frac{d\Phi}{dx_{1,i}} \cdot \frac{dx_{1,i}}{dx_{0,j}} \right\rangle = \\ &= \sum_{i=1}^{64} \left\langle \delta_4, \frac{d\Phi}{dx_{1,i}} \cdot \frac{dx_{1,i}}{dx_{0,j}} \right\rangle = \\ &= \sum_{i=1}^{64} \left\langle \delta_4, \frac{d\Phi}{dx_{1,i}} \right\rangle \cdot \frac{dx_{1,i}}{dx_{0,j}} = \\ &= \sum_{i=1}^{64} \delta_{2,i} \cdot \frac{dx_{1,i}}{dx_{0,j}} = \\ &= \sum_{i=1}^{64} \delta_{2,i} \cdot \frac{d}{dx_{0,j}} (\varphi_1(h_1))_i = \\ &= \sum_{i=1}^{64} \delta_{2,i} \cdot \left(\varphi'_1(h_1) \odot \frac{dh_1}{dx_{0,j}} \right)_i = \\ &= \sum_{i=1}^{64} \delta_{2,i} \cdot \left(\varphi'_1(h_1) \odot \frac{d}{dx_{0,j}} (W_1 x_0 + b_1) \right)_i = \\ &= \sum_{i=1}^{64} \delta_{2,i} \cdot \left(\varphi'_1(h_1) \odot \left(W_1 \frac{dx_0}{dx_{0,j}} \right) \right)_i = \\ &= \sum_{i=1}^{64} \delta_{2,i} \cdot (\varphi'_1(h_1) \odot (W_1 \vec{e}_j))_i = \\ &= \langle \delta_2, \varphi'_1(h_1) \odot (W_1 \vec{e}_j) \rangle \end{aligned}$$

By Lemma 4 we get:

$$\begin{aligned} \left\langle \delta_4, \frac{d\Phi}{dx_{0,j}} \right\rangle &= \langle \delta_2 \odot \varphi'_1(h_1), W_1 \vec{e}_j \rangle = \\ &= (W_1 \vec{e}_j)^T (\delta_2 \odot \varphi'_1(h_1)) = \\ &= \vec{e}_j^T W_1^T (\delta_2 \odot \varphi'_1(h_1)) = \\ &= \vec{e}_j^T \delta_1 = \\ &= \delta_{1,j} \end{aligned}$$

Hence:

$$\left\langle \delta_4, \frac{d\Phi}{dx_0} \right\rangle = \delta_1$$

Now, we are ready to evaluate the inner products: $\left\langle \delta_4, \frac{d\Phi}{dW_3} \right\rangle, \left\langle \delta_4, \frac{d\Phi}{db_3} \right\rangle, \left\langle \delta_4, \frac{d\Phi}{dW_2} \right\rangle, \left\langle \delta_4, \frac{d\Phi}{db_2} \right\rangle, \left\langle \delta_4, \frac{d\Phi}{dW_1} \right\rangle, \left\langle \delta_4, \frac{d\Phi}{db_1} \right\rangle, \left\langle \delta_4, \frac{d\Phi}{dz} \right\rangle, \left\langle \delta_4, \frac{d\Phi}{df} \right\rangle.$

Computing $\left\langle \delta_4, \frac{d\Phi}{dW_3} \right\rangle$

$$\left\langle \delta_4, \frac{d\Phi}{dW_{3,j,k}} \right\rangle = \left\langle \delta_4, \sum_{i=1}^4 \frac{d\Phi}{dx_{3,i}} \cdot \frac{dx_{3,i}}{dW_{3,j,k}} \right\rangle =$$

$$\begin{aligned}
&= \sum_{i=1}^4 \left\langle \delta_4, \frac{d\Phi}{dx_{3,i}} \cdot \frac{dx_{3,i}}{dW_{3,j,k}} \right\rangle = \\
&= \sum_{i=1}^4 \left\langle \delta_4, \frac{d\Phi}{dx_{3,i}} \right\rangle \cdot \frac{dx_{3,i}}{dW_{3,j,k}} = \\
&= \sum_{i=1}^4 \delta_{4,i} \cdot \frac{dx_{3,i}}{dW_{3,j,k}} = \\
&= \sum_{i=1}^4 \delta_{4,i} \cdot \frac{d}{dW_{3,j,k}} (\varphi_3(h_3))_i = \\
&= \sum_{i=1}^4 \delta_{4,i} \cdot \left(\varphi'_3(h_3) \odot \frac{dh_3}{dW_{3,j,k}} \right)_i = \\
&= \sum_{i=1}^4 \delta_{4,i} \cdot \left(\varphi'_3(h_3) \odot \frac{d}{dW_{3,j,k}} (W_3 x_2 + b_3) \right)_i = \\
&= \sum_{i=1}^4 \delta_{4,i} \cdot \left(\varphi'_3(h_3) \odot \left(\frac{dW_3}{dW_{3,j,k}} x_2 \right) \right)_i = \\
&= \sum_{i=1}^4 \delta_{4,i} \cdot \left(\varphi'_3(h_3) \odot (\tilde{\epsilon}_j \tilde{\epsilon}_k^T x_2) \right)_i = \\
&= \langle \delta_4, \varphi'_3(h_3) \odot (\tilde{\epsilon}_j \tilde{\epsilon}_k^T x_2) \rangle = \\
&= \langle \delta_4, \varphi'_3(h_3) \odot (\tilde{\epsilon}_j x_{2,k}) \rangle
\end{aligned}$$

By Lemma 4 we get:

$$\begin{aligned}
\left\langle \delta_4, \frac{d\Phi}{dW_{3,j,k}} \right\rangle &= \langle \delta_4 \odot \varphi'_3(h_3), \tilde{\epsilon}_j x_{2,k} \rangle = \\
&= (\tilde{\epsilon}_j x_{2,k})^T (\delta_4 \odot \varphi'_3(h_3)) = \\
&= x_{2,k} \tilde{\epsilon}_j^T (\delta_4 \odot \varphi'_3(h_3)) = \\
&= x_{2,k} (\delta_4 \odot \varphi'_3(h_3))_j = \\
&= (\delta_4 \odot \varphi'_3(h_3))_j x_{2,k}
\end{aligned}$$

Hence:

$$\left\langle \delta_4, \frac{d\Phi}{dW_3} \right\rangle = (\delta_4 \odot \varphi'_3(h_3)) x_2^T$$

Computing $\left\langle \delta_4, \frac{d\Phi}{db_3} \right\rangle$

$$\begin{aligned}
\left\langle \delta_4, \frac{d\Phi}{db_{3,j}} \right\rangle &= \left\langle \delta_4, \sum_{i=1}^4 \frac{d\Phi}{dx_{3,i}} \cdot \frac{dx_{3,i}}{db_{3,j}} \right\rangle = \\
&= \sum_{i=1}^4 \left\langle \delta_4, \frac{d\Phi}{dx_{3,i}} \cdot \frac{dx_{3,i}}{db_{3,j}} \right\rangle = \\
&= \sum_{i=1}^4 \left\langle \delta_4, \frac{d\Phi}{dx_{3,i}} \right\rangle \cdot \frac{dx_{3,i}}{db_{3,j}} = \\
&= \sum_{i=1}^4 \delta_{4,i} \cdot \frac{dx_{3,i}}{db_{3,j}} = \\
&= \sum_{i=1}^4 \delta_{4,i} \cdot \frac{d}{db_{3,j}} (\varphi_3(h_3))_i =
\end{aligned}$$

$$\begin{aligned}
&= \sum_{i=1}^4 \delta_{4,i} \cdot \left(\varphi'_3(h_3) \frac{dh_3}{db_{3,j}} \right)_i = \\
&= \sum_{i=1}^4 \delta_{4,i} \cdot \left(\varphi'_3(h_3) \frac{d}{db_{3,j}} (W_3 x_2 + b_3) \right)_i = \\
&= \sum_{i=1}^4 \delta_{4,i} \cdot \left(\varphi'_3(h_3) \frac{db_3}{db_{3,j}} \right)_i = \\
&= \sum_{i=1}^4 \delta_{4,i} \cdot (\varphi'_3(h_3) \vec{e}_j)_i = \\
&= \sum_{i=1}^4 \delta_{4,i} \cdot \varphi'_{3,i}(h_3) \cdot \delta_{ij} = \\
&= \delta_{4,j} \cdot \varphi'_{3,j}(h_3)
\end{aligned}$$

Hence:

$$\left\langle \delta_4, \frac{d\Phi}{db_3} \right\rangle = \delta_4 \cdot \varphi'_3(h_3)$$

Computing $\left\langle \delta_4, \frac{d\Phi}{dW_2} \right\rangle$

$$\begin{aligned}
\left\langle \delta_4, \frac{d\Phi}{dW_{2,j,k}} \right\rangle &= \left\langle \delta_4, \sum_{i=1}^{64} \frac{d\Phi}{dx_{2,i}} \cdot \frac{dx_{2,i}}{dW_{2,j,k}} \right\rangle = \\
&= \sum_{i=1}^{64} \left\langle \delta_4, \frac{d\Phi}{dx_{2,i}} \cdot \frac{dx_{2,i}}{dW_{2,j,k}} \right\rangle = \\
&= \sum_{i=1}^{64} \left\langle \delta_4, \frac{d\Phi}{dx_{2,i}} \right\rangle \cdot \frac{dx_{2,i}}{dW_{2,j,k}} = \\
&= \sum_{i=1}^{64} \delta_{3,i} \cdot \frac{dx_{2,i}}{dW_{2,j,k}} = \\
&= \sum_{i=1}^{64} \delta_{3,i} \cdot \frac{d}{dW_{2,j,k}} (\varphi_2(h_2))_i = \\
&= \sum_{i=1}^{64} \delta_{3,i} \cdot \left(\varphi'_2(h_2) \odot \frac{dh_2}{dW_{2,j,k}} \right)_i = \\
&= \sum_{i=1}^{64} \delta_{3,i} \cdot \left(\varphi'_2(h_2) \odot \frac{d}{dW_{2,j,k}} (W_2 x_1 + b_2) \right)_i = \\
&= \sum_{i=1}^{64} \delta_{3,i} \cdot \left(\varphi'_2(h_2) \odot \left(\frac{dW_2}{dW_{2,j,k}} x_1 \right) \right)_i = \\
&= \sum_{i=1}^{64} \delta_{3,i} \cdot (\varphi'_2(h_2) \odot (\vec{e}_j \vec{e}_k^T x_1))_i = \\
&= \langle \delta_3, \varphi'_2(h_2) \odot (\vec{e}_j \vec{e}_k^T x_1) \rangle = \\
&= \langle \delta_3, \varphi'_2(h_2) \odot (\vec{e}_j x_{1,k}) \rangle
\end{aligned}$$

By Lemma 4 we get:

$$\left\langle \delta_4, \frac{d\Phi}{dW_{2,j,k}} \right\rangle = \langle \delta_3 \odot \varphi'_2(h_2), \vec{e}_j x_{1,k} \rangle =$$

$$\begin{aligned}
&= (\vec{\varepsilon}_j \mathbf{x}_{1,k})^T (\delta_3 \odot \varphi'_2(\mathbf{h}_2)) = \\
&= \mathbf{x}_{1,k} \vec{\varepsilon}_j^T (\delta_3 \odot \varphi'_2(\mathbf{h}_2)) = \\
&= \mathbf{x}_{1,k} (\delta_3 \odot \varphi'_2(\mathbf{h}_2))_j = \\
&= (\delta_3 \odot \varphi'_2(\mathbf{h}_2))_j \mathbf{x}_{1,k}
\end{aligned}$$

Hence:

$$\left\langle \delta_4, \frac{d\Phi}{d\mathbf{W}_2} \right\rangle = (\delta_3 \odot \varphi'_2(\mathbf{h}_2)) \mathbf{x}_1^T$$

Computing $\left\langle \delta_4, \frac{d\Phi}{d\mathbf{b}_2} \right\rangle$

$$\begin{aligned}
\left\langle \delta_4, \frac{d\Phi}{d\mathbf{b}_{2,j}} \right\rangle &= \left\langle \delta_4, \sum_{i=1}^{64} \frac{d\Phi}{d\mathbf{x}_{2,i}} \cdot \frac{d\mathbf{x}_{2,i}}{d\mathbf{b}_{2,j}} \right\rangle = \\
&= \sum_{i=1}^{64} \left\langle \delta_4, \frac{d\Phi}{d\mathbf{x}_{2,i}} \cdot \frac{d\mathbf{x}_{2,i}}{d\mathbf{b}_{2,j}} \right\rangle = \\
&= \sum_{i=1}^{64} \left\langle \delta_4, \frac{d\Phi}{d\mathbf{x}_{2,i}} \right\rangle \cdot \frac{d\mathbf{x}_{2,i}}{d\mathbf{b}_{2,j}} = \\
&= \sum_{i=1}^{64} \delta_{3,i} \cdot \frac{d\mathbf{x}_{2,i}}{d\mathbf{b}_{2,j}} = \\
&= \sum_{i=1}^{64} \delta_{3,i} \cdot \frac{d}{d\mathbf{b}_{2,j}} (\varphi_2(\mathbf{h}_2))_i = \\
&= \sum_{i=1}^{64} \delta_{3,i} \cdot \left(\varphi'_2(\mathbf{h}_2) \frac{d\mathbf{h}_2}{d\mathbf{b}_{2,j}} \right)_i = \\
&= \sum_{i=1}^{64} \delta_{3,i} \cdot \left(\varphi'_2(\mathbf{h}_2) \frac{d}{d\mathbf{b}_{2,j}} (\mathbf{W}_2 \mathbf{x}_1 + \mathbf{b}_2) \right)_i = \\
&= \sum_{i=1}^{64} \delta_{3,i} \cdot \left(\varphi'_2(\mathbf{h}_2) \frac{d\mathbf{b}_2}{d\mathbf{b}_{2,j}} \right)_i = \\
&= \sum_{i=1}^{64} \delta_{3,i} \cdot (\varphi'_2(\mathbf{h}_2) \vec{\varepsilon}_j)_i = \\
&= \sum_{i=1}^{64} \delta_{3,i} \cdot \varphi'_{2,i}(\mathbf{h}_2) \cdot \delta_{ij} = \\
&= \delta_{3,j} \cdot \varphi'_{2,j}(\mathbf{h}_2)
\end{aligned}$$

Hence:

$$\left\langle \delta_4, \frac{d\Phi}{d\mathbf{b}_2} \right\rangle = \delta_3 \cdot \varphi'_2(\mathbf{h}_2)$$

Computing $\left\langle \delta_4, \frac{d\Phi}{d\mathbf{W}_1} \right\rangle$

$$\begin{aligned}
\left\langle \delta_4, \frac{d\Phi}{d\mathbf{W}_{1,j,k}} \right\rangle &= \left\langle \delta_4, \sum_{i=1}^{64} \frac{d\Phi}{d\mathbf{x}_{1,i}} \cdot \frac{d\mathbf{x}_{1,i}}{d\mathbf{W}_{1,j,k}} \right\rangle = \\
&= \sum_{i=1}^{64} \left\langle \delta_4, \frac{d\Phi}{d\mathbf{x}_{1,i}} \cdot \frac{d\mathbf{x}_{1,i}}{d\mathbf{W}_{1,j,k}} \right\rangle =
\end{aligned}$$

$$\begin{aligned}
&= \sum_{i=1}^{64} \left\langle \delta_4, \frac{d\Phi}{dx_{1,i}} \right\rangle \cdot \frac{dx_{1,i}}{dW_{1,j,k}} = \\
&= \sum_{i=1}^{64} \delta_{2,i} \cdot \frac{dx_{1,i}}{dW_{1,j,k}} = \\
&= \sum_{i=1}^{64} \delta_{2,i} \cdot \frac{d}{dW_{1,j,k}} (\varphi_1(h_1))_i = \\
&= \sum_{i=1}^{64} \delta_{2,i} \cdot \left(\varphi'_1(h_1) \odot \frac{dh_1}{dW_{1,j,k}} \right)_i = \\
&= \sum_{i=1}^{64} \delta_{2,i} \cdot \left(\varphi'_1(h_1) \odot \frac{d}{dW_{1,j,k}} (W_1 x_0 + b_1) \right)_i = \\
&= \sum_{i=1}^{64} \delta_{2,i} \cdot \left(\varphi'_1(h_1) \odot \left(\frac{dW_1}{dW_{1,j,k}} x_0 \right) \right)_i = \\
&= \sum_{i=1}^{64} \delta_{2,i} \cdot (\varphi'_1(h_1) \odot (\vec{\epsilon}_j \vec{\epsilon}_k^T x_0))_i = \\
&= \langle \delta_2, \varphi'_1(h_1) \odot (\vec{\epsilon}_j \vec{\epsilon}_k^T x_0) \rangle = \\
&= \langle \delta_2, \varphi'_1(h_1) \odot (\vec{\epsilon}_j x_{0,k}) \rangle
\end{aligned}$$

By Lemma 4 we get:

$$\begin{aligned}
\left\langle \delta_4, \frac{d\Phi}{dW_{1,j,k}} \right\rangle &= \langle \delta_2 \odot \varphi'_1(h_1), \vec{\epsilon}_j x_{0,k} \rangle = \\
&= (\vec{\epsilon}_j x_{0,k})^T (\delta_2 \odot \varphi'_1(h_1)) = \\
&= x_{0,k} \vec{\epsilon}_j^T (\delta_2 \odot \varphi'_1(h_1)) = \\
&= x_{0,k} (\delta_2 \odot \varphi'_1(h_1))_j = \\
&= (\delta_2 \odot \varphi'_1(h_1))_j x_{0,k}
\end{aligned}$$

Hence:

$$\left\langle \delta_4, \frac{d\Phi}{dW_1} \right\rangle = (\delta_2 \odot \varphi'_1(h_1)) x_0^T$$

Computing $\left\langle \delta_4, \frac{d\Phi}{db_1} \right\rangle$

$$\begin{aligned}
\left\langle \delta_4, \frac{d\Phi}{db_{1,j}} \right\rangle &= \left\langle \delta_4, \sum_{i=1}^{64} \frac{d\Phi}{dx_{1,i}} \cdot \frac{dx_{1,i}}{db_{1,j}} \right\rangle = \\
&= \sum_{i=1}^{64} \left\langle \delta_4, \frac{d\Phi}{dx_{1,i}} \cdot \frac{dx_{1,i}}{db_{1,j}} \right\rangle = \\
&= \sum_{i=1}^{64} \left\langle \delta_4, \frac{d\Phi}{dx_{1,i}} \right\rangle \cdot \frac{dx_{1,i}}{db_{1,j}} = \\
&= \sum_{i=1}^{64} \delta_{2,i} \cdot \frac{dx_{1,i}}{db_{1,j}} = \\
&= \sum_{i=1}^{64} \delta_{2,i} \cdot \frac{d}{db_{1,j}} (\varphi_1(h_1))_i = \\
&= \sum_{i=1}^{64} \delta_{2,i} \cdot \left(\varphi'_1(h_1) \frac{dh_1}{db_{1,j}} \right)_i =
\end{aligned}$$

$$\begin{aligned}
&= \sum_{i=1}^{64} \delta_{2,i} \cdot \left(\varphi'_1(h_1) \frac{d}{db_{1,j}} (W_1 x_0 + b_1) \right)_i = \\
&= \sum_{i=1}^{64} \delta_{2,i} \cdot \left(\varphi'_1(h_1) \frac{db_1}{db_{1,j}} \right)_i = \\
&= \sum_{i=1}^{64} \delta_{2,i} \cdot (\varphi'_1(h_1) \vec{\varepsilon}_j)_i = \\
&= \sum_{i=1}^{64} \delta_{2,i} \cdot \varphi'_{1,i}(h_1) \cdot \delta_{ij} = \\
&= \delta_{2,j} \cdot \varphi'_{1,j}(h_1)
\end{aligned}$$

Hence:

$$\left\langle \delta_4, \frac{d\Phi}{db_1} \right\rangle = \delta_2 \cdot \varphi'_1(h_1)$$

Computing $\left\langle \delta_4, \frac{d\Phi}{dz} \right\rangle$ Since

$$z = \begin{bmatrix} x_{0,8+24+1} \\ \vdots \\ x_{0,8+24+96} \end{bmatrix} = \begin{bmatrix} x_{0,32+1} \\ \vdots \\ x_{0,32+96} \end{bmatrix}$$

we have:

$$\left\langle \delta_4, \frac{d\Phi}{dz_i} \right\rangle = \left\langle \delta_4, \frac{d\Phi}{dx_{0,32+i}} \right\rangle = \delta_{1,32+i}$$

Consequently:

$$\left\langle \delta_4, \frac{d\Phi}{dz} \right\rangle = \begin{bmatrix} \delta_{1,32+1} \\ \vdots \\ \delta_{1,32+96} \end{bmatrix}$$

Computing $\left\langle \delta_4, \frac{d\Phi}{d\mathcal{F}} \right\rangle$

Since $\mathcal{F} = \{f_1, \dots, f_{6577}\}$, let us fix $i \in \{1, \dots, 6577\}$ and $j \in \{1, 2\}$. Then:

$$\begin{aligned}
\left\langle \delta_4, \frac{d\Phi}{df_{i,j}} \right\rangle &= \left\langle \delta_4, \sum_{k=1}^4 \frac{d\Phi}{dx_{0,2(k-1)+j}} \cdot \frac{dx_{0,2(k-1)+j}}{df_{i,j}} \right\rangle = \\
&= \sum_{k=1}^4 \left\langle \delta_4, \frac{d\Phi}{dx_{0,2(k-1)+j}} \cdot \frac{dx_{0,2(k-1)+j}}{df_{i,j}} \right\rangle = \\
&= \sum_{k=1}^4 \left\langle \delta_4, \frac{d\Phi}{dx_{0,2(k-1)+j}} \right\rangle \cdot \frac{dx_{0,2(k-1)+j}}{df_{i,j}} = \\
&= \sum_{k=1}^4 \delta_{1,2(k-1)+j} \cdot \frac{dx_{0,2(k-1)+j}}{df_{i,j}} = \\
&= \sum_{k=1}^4 \delta_{1,2(k-1)+j} \cdot \frac{d\hat{f}_j^{(k)}}{df_{i,j}}
\end{aligned}$$

It remains, therefore, to show how to compute: $\frac{d\hat{f}_j^{(k)}}{df_{i,j}}$. As can be easily verified:

$$\begin{aligned}
\frac{d\hat{f}_j^{(k)}}{df_{i,j}} &= \left(1 - u_{\text{frac}}^{(k)}\right) \cdot \left(1 - v_{\text{frac}}^{(k)}\right) \cdot \delta_{\frac{\text{ind}_{00}^{(k)}}{2} + 1, i} + \\
&\quad + u_{\text{frac}}^{(k)} \cdot \left(1 - v_{\text{frac}}^{(k)}\right) \cdot \delta_{\frac{\text{ind}_{01}^{(k)}}{2} + 1, i} +
\end{aligned}$$

$$\begin{aligned}
& + \left(1 - u_{\text{frac}}^{(k)}\right) \cdot v_{\text{frac}}^{(k)} \cdot \delta_{\frac{\text{ind}_{10}}{2} + 1, i} + \\
& + u_{\text{frac}}^{(k)} \cdot v_{\text{frac}}^{(k)} \cdot \delta_{\frac{\text{ind}_{11}}{2} + 1, i}
\end{aligned}$$

where temporarily, due to a notation conflict, δ_{ij} denotes the Kronecker delta.

To complete the derivation, we provide the closed-form formulas for the derivatives of the hardswish, ReLU and sigmoid activation function σ .

Computing: $\frac{d}{dx}(\text{hardswish}(x))$, $\frac{d}{dx}(\text{ReLU}(x))$ and $\frac{d}{dx}(\sigma(x))$

$$\begin{aligned}
\frac{d}{dx}(\text{hardswish}(x)) &= \frac{d}{dx} \left(\begin{cases} 0 & , x \leq -3 \\ x & , x \geq 3 \\ \frac{x(x+3)}{6} & , \text{otherwise} \end{cases} \right) = \\
&= \frac{d}{dx}(x \cdot \text{clamp}(x+3, 0, 6)) = \\
&= (1 \cdot \text{clamp}(x+3, 0, 6)) + \left(x \cdot \frac{d}{dx}(\text{clamp}(x+3, 0, 6)) \right) = \\
&= (\text{clamp}(x+3, 0, 6)) + (x \cdot \mathbb{1}_{(x+3) \in [0, 6]}) = \\
&= (\text{clamp}(x+3, 0, 6)) + (x \cdot \mathbb{1}_{x \in [-3, 3]}) =
\end{aligned}$$

$$\begin{aligned}
\frac{d}{dx}(\text{ReLU}(x)) &= \frac{d}{dx}(\max\{x, 0\}) = \\
&= \mathbb{1}_{x \geq 0}
\end{aligned}$$

$$\begin{aligned}
\frac{d}{dx}(\sigma(x)) &= \frac{d}{dx} \left(\frac{1}{1 + e^{-x}} \right) = \\
&= -\frac{1}{(1 + e^{-x})^2} \cdot (-e^{-x}) = \\
&= \frac{e^{-x}}{(1 + e^{-x})^2} = \\
&= \frac{e^{-x} + 0}{(1 + e^{-x})^2} = \\
&= \frac{e^{-x} + (1 - 1)}{(1 + e^{-x})^2} = \\
&= \frac{(1 + e^{-x}) - 1}{(1 + e^{-x})^2} = \\
&= \frac{1}{1 + e^{-x}} - \frac{1}{(1 + e^{-x})^2} = \\
&= \frac{1}{1 + e^{-x}} \left(1 - \frac{1}{1 + e^{-x}} \right) = \\
&= \sigma(x)(1 - \sigma(x))
\end{aligned}$$

We will now show how to compute the inner products: $\langle \delta_4, \frac{d\Phi}{du} \rangle$ and $\langle \delta_4, \frac{d\Phi}{dv} \rangle$. These quantities will be used to deduce the derivatives of $\frac{d\mathcal{L}}{d\mu_i^{(r)}}$, $\frac{d\mathcal{L}}{ds_i^{(r)}}$ and $\frac{d\mathcal{L}}{dq_i^{(r)}}$, as they comprise one of their key components.

Computing $\langle \delta_4, \frac{d\Phi}{du} \rangle$ and $\langle \delta_4, \frac{d\Phi}{dv} \rangle$

$$\left\langle \delta_4, \frac{d\Phi}{du} \right\rangle = \left\langle \delta_4, \sum_{i=1}^8 \frac{d\Phi}{dx_{0,i}} \cdot \frac{dx_{0,i}}{du} \right\rangle =$$

$$\begin{aligned}
&= \left\langle \delta_4, \sum_{i=1}^4 \sum_{j=1}^2 \frac{d\Phi}{dx_{0,2(i-1)+j}} \cdot \frac{dx_{0,2(i-1)+j}}{du} \right\rangle = \\
&= \sum_{i=1}^4 \sum_{j=1}^2 \left\langle \delta_4, \frac{d\Phi}{dx_{0,2(i-1)+j}} \cdot \frac{dx_{0,2(i-1)+j}}{du} \right\rangle = \\
&= \sum_{i=1}^4 \sum_{j=1}^2 \left\langle \delta_4, \frac{d\Phi}{dx_{0,2(i-1)+j}} \right\rangle \cdot \frac{dx_{0,2(i-1)+j}}{du} = \\
&= \sum_{i=1}^4 \sum_{j=1}^2 \delta_{1,2(i-1)+j} \cdot \frac{dx_{0,2(i-1)+j}}{du} = \\
&= \sum_{i=1}^4 \sum_{j=1}^2 \delta_{1,2(i-1)+j} \cdot \frac{d\hat{f}_j^{(i)}}{du}
\end{aligned}$$

$$\begin{aligned}
\left\langle \delta_4, \frac{d\Phi}{dv} \right\rangle &= \left\langle \delta_4, \sum_{i=1}^8 \frac{d\Phi}{dx_{0,i}} \cdot \frac{dx_{0,i}}{dv} \right\rangle = \\
&= \left\langle \delta_4, \sum_{i=1}^4 \sum_{j=1}^2 \frac{d\Phi}{dx_{0,2(i-1)+j}} \cdot \frac{dx_{0,2(i-1)+j}}{dv} \right\rangle = \\
&= \sum_{i=1}^4 \sum_{j=1}^2 \left\langle \delta_4, \frac{d\Phi}{dx_{0,2(i-1)+j}} \cdot \frac{dx_{0,2(i-1)+j}}{dv} \right\rangle = \\
&= \sum_{i=1}^4 \sum_{j=1}^2 \left\langle \delta_4, \frac{d\Phi}{dx_{0,2(i-1)+j}} \right\rangle \cdot \frac{dx_{0,2(i-1)+j}}{dv} = \\
&= \sum_{i=1}^4 \sum_{j=1}^2 \delta_{1,2(i-1)+j} \cdot \frac{dx_{0,2(i-1)+j}}{dv} = \\
&= \sum_{i=1}^4 \sum_{j=1}^2 \delta_{1,2(i-1)+j} \cdot \frac{d\hat{f}_j^{(i)}}{dv}
\end{aligned}$$

It remains, therefore, to show how to compute: $\frac{d\hat{f}^{(i)}}{du}$ and $\frac{d\hat{f}^{(i)}}{dv}$. As can be easily verified:

$$\begin{aligned}
\frac{d\hat{f}^{(i)}}{du} &= \frac{d\hat{f}^{(i)}}{du_{\text{frac}}^{(i)}} \cdot \frac{du_{\text{frac}}^{(i)}}{du} = \\
&= \left(\left(1 - v_{\text{frac}}^{(i)}\right) \left(f_{01}^{(i)} - f_{00}^{(i)}\right) + v_{\text{frac}}^{(i)} \left(f_{11}^{(i)} - f_{10}^{(i)}\right) \right) \cdot \frac{du_{\text{frac}}^{(i)}}{du} = \\
&= \left(\left(1 - v_{\text{frac}}^{(i)}\right) \left(f_{01}^{(i)} - f_{00}^{(i)}\right) + v_{\text{frac}}^{(i)} \left(f_{11}^{(i)} - f_{10}^{(i)}\right) \right) \cdot \frac{N^{(i)} - 1}{2R_*}
\end{aligned}$$

$$\begin{aligned}
\frac{d\hat{f}^{(i)}}{dv} &= \frac{d\hat{f}^{(i)}}{dv_{\text{frac}}^{(i)}} \cdot \frac{dv_{\text{frac}}^{(i)}}{dv} = \\
&= \left(\left(1 - u_{\text{frac}}^{(i)}\right) \left(f_{10}^{(i)} - f_{00}^{(i)}\right) + u_{\text{frac}}^{(i)} \left(f_{11}^{(i)} - f_{01}^{(i)}\right) \right) \cdot \frac{dv_{\text{frac}}^{(i)}}{dv} = \\
&= \left(\left(1 - u_{\text{frac}}^{(i)}\right) \left(f_{10}^{(i)} - f_{00}^{(i)}\right) + u_{\text{frac}}^{(i)} \left(f_{11}^{(i)} - f_{01}^{(i)}\right) \right) \cdot \frac{N^{(i)} - 1}{2R_*}
\end{aligned}$$

where the straight-through estimator was employed in the final equalities of both derivative formulas, neglecting the discontinuities associated with the floor function.

Computing leaf gradients

Since the vector: $P'_i{}^{(r)} - \mu_i^{(r)}$ connecting the mean $\mu_i^{(r)}$ of the i -th Gaussian intersected by a ray r to the intersection point $P'_i{}^{(r)}$ is orthogonal to the Gaussians' normal vector $\vec{N}_i^{(r)}$, we have:

$$\langle \vec{N}_i^{(r)}, P'_i{}^{(r)} - \mu_i^{(r)} \rangle = 0$$

Since:

$$\begin{aligned} \langle \vec{N}_i^{(r)}, P'_i{}^{(r)} - \mu_i^{(r)} \rangle &= \langle \vec{N}_i^{(r)}, P'_i{}^{(r)} \rangle - \langle \vec{N}_i^{(r)}, \mu_i^{(r)} \rangle = \\ &= \langle \vec{N}_i^{(r)}, o^{(r)} + t_i^{(r)} \vec{d}^{(r)} \rangle - \langle \vec{N}_i^{(r)}, \mu_i^{(r)} \rangle = \\ &= \langle \vec{N}_i^{(r)}, o^{(r)} \rangle + \langle \vec{N}_i^{(r)}, t_i^{(r)} \vec{d}^{(r)} \rangle - \langle \vec{N}_i^{(r)}, \mu_i^{(r)} \rangle = \\ &= \langle \vec{N}_i^{(r)}, o^{(r)} \rangle + \langle \vec{N}_i^{(r)}, \vec{d}^{(r)} \rangle \cdot t_i^{(r)} - \langle \vec{N}_i^{(r)}, \mu_i^{(r)} \rangle = \\ &= \langle \vec{N}_i^{(r)}, o^{(r)} - \mu_i^{(r)} \rangle + \langle \vec{N}_i^{(r)}, \vec{d}^{(r)} \rangle \cdot t_i^{(r)} \end{aligned}$$

for some $t_i^{(r)} \in \mathbb{R}_+ \cup \{0\}$, it must hold that:

$$\langle \vec{N}_i^{(r)}, \vec{d}^{(r)} \rangle \cdot t_i^{(r)} = -\langle \vec{N}_i^{(r)}, o^{(r)} - \mu_i^{(r)} \rangle = \langle \vec{N}_i^{(r)}, \mu_i^{(r)} - o^{(r)} \rangle$$

Hence:

$$t_i^{(r)} = \frac{\langle \vec{N}_i^{(r)}, \mu_i^{(r)} - o^{(r)} \rangle}{\langle \vec{N}_i^{(r)}, \vec{d}^{(r)} \rangle}$$

and the hit point $P'_i{}^{(r)}$ is given by the following formula:

$$P'_i{}^{(r)} = o^{(r)} + t_i^{(r)} \vec{d}^{(r)}$$

The vectors: $\vec{U}_i^{(r)}$, $\vec{V}_i^{(r)}$, and $\vec{N}_i^{(r)}$ form an orthogonal coordinate frame, therefore the intersection point local frame coordinates: $u_i^{(r)}$ and $v_i^{(r)}$ can be determined by:

$$\begin{aligned} u_i^{(r)} &= \frac{\langle \vec{U}_i^{(r)}, P'_i{}^{(r)} - \mu_i^{(r)} \rangle}{s_{i,1}^{(r)}} \\ v_i^{(r)} &= \frac{\langle \vec{V}_i^{(r)}, P'_i{}^{(r)} - \mu_i^{(r)} \rangle}{s_{i,2}^{(r)}} \end{aligned}$$

Thus:

$$\frac{dt_i^{(r)}}{d\mu_{i,j}^{(r)}} = \frac{\left\langle \vec{N}_i^{(r)}, \frac{\partial}{\partial \mu_{i,j}^{(r)}} (\mu_i^{(r)} - o^{(r)}) \right\rangle}{\langle \vec{N}_i^{(r)}, \vec{d}^{(r)} \rangle} = \frac{\langle \vec{N}_i^{(r)}, \vec{\epsilon}_j \rangle}{\langle \vec{N}_i^{(r)}, \vec{d}^{(r)} \rangle} = \frac{\vec{N}_{i,j}^{(r)}}{\langle \vec{N}_i^{(r)}, \vec{d}^{(r)} \rangle}$$

$$\begin{aligned} \frac{du_i^{(r)}}{d\mu_{i,j}^{(r)}} &= \frac{\left\langle \vec{U}_i^{(r)}, \frac{\partial}{\partial \mu_{i,j}^{(r)}} (P'_i{}^{(r)} - \mu_i^{(r)}) \right\rangle}{s_{i,1}^{(r)}} = \\ &= \frac{\left\langle \vec{U}_i^{(r)}, \frac{\partial}{\partial \mu_{i,j}^{(r)}} (P'_i{}^{(r)} - \mu_i^{(r)}) \right\rangle}{s_{i,1}^{(r)}} = \\ &= \frac{\left\langle \vec{U}_i^{(r)}, \frac{\partial}{\partial \mu_{i,j}^{(r)}} P'_i{}^{(r)} - \frac{\partial}{\partial \mu_{i,j}^{(r)}} \mu_i^{(r)} \right\rangle}{s_{i,1}^{(r)}} = \end{aligned}$$

$$\begin{aligned}
&= \frac{\left\langle \vec{U}_i^{(r)}, \frac{\partial}{\partial \mu_{i,j}^{(r)}} \left(o^{(r)} + t_i^{(r)} \vec{d}^{(r)} \right) - \vec{\epsilon}_j \right\rangle}{s_{i,1}^{(r)}} = \\
&= \frac{\left\langle \vec{U}_i^{(r)}, \frac{\partial}{\partial \mu_{i,j}^{(r)}} \left(t_i^{(r)} \right) \cdot \vec{d}^{(r)} - \vec{\epsilon}_j \right\rangle}{s_{i,1}^{(r)}} = \\
&= \frac{\left\langle \vec{U}_i^{(r)}, \frac{\partial}{\partial \mu_{i,j}^{(r)}} \left(t_i^{(r)} \right) \cdot \vec{d}^{(r)} \right\rangle}{s_{i,1}^{(r)}} - \frac{\left\langle \vec{U}_i^{(r)}, \vec{\epsilon}_j \right\rangle}{s_{i,1}^{(r)}} = \\
&= \frac{\left\langle \vec{U}_i^{(r)}, \vec{d}^{(r)} \right\rangle}{s_{i,1}^{(r)}} \cdot \frac{\partial}{\partial \mu_{i,j}^{(r)}} t_i^{(r)} - \frac{\vec{U}_{i,j}^{(r)}}{s_{i,1}^{(r)}} = \\
&= \frac{\left\langle \vec{U}_i^{(r)}, \vec{d}^{(r)} \right\rangle \cdot \frac{\partial}{\partial \mu_{i,j}^{(r)}} t_i^{(r)} - \vec{U}_{i,j}^{(r)}}{s_{i,1}^{(r)}}
\end{aligned}$$

$$\begin{aligned}
\frac{dv_i^{(r)}}{d\mu_{i,j}^{(r)}} &= \frac{\left\langle \vec{V}_i^{(r)}, \frac{\partial}{\partial \mu_{i,j}^{(r)}} \left(P_i'^{(r)} - \mu_i^{(r)} \right) \right\rangle}{s_{i,2}^{(r)}} \\
&= \frac{\left\langle \vec{V}_i^{(r)}, \frac{\partial}{\partial \mu_{i,j}^{(r)}} \left(P_i'^{(r)} - \mu_i^{(r)} \right) \right\rangle}{s_{i,2}^{(r)}} \\
&= \frac{\left\langle \vec{V}_i^{(r)}, \frac{\partial}{\partial \mu_{i,j}^{(r)}} P_i'^{(r)} - \frac{\partial}{\partial \mu_{i,j}^{(r)}} \mu_i^{(r)} \right\rangle}{s_{i,2}^{(r)}} \\
&= \frac{\left\langle \vec{V}_i^{(r)}, \frac{\partial}{\partial \mu_{i,j}^{(r)}} \left(o^{(r)} + t_i^{(r)} \vec{d}^{(r)} \right) - \vec{\epsilon}_j \right\rangle}{s_{i,2}^{(r)}} \\
&= \frac{\left\langle \vec{V}_i^{(r)}, \frac{\partial}{\partial \mu_{i,j}^{(r)}} \left(t_i^{(r)} \right) \cdot \vec{d}^{(r)} - \vec{\epsilon}_j \right\rangle}{s_{i,2}^{(r)}} \\
&= \frac{\left\langle \vec{V}_i^{(r)}, \frac{\partial}{\partial \mu_{i,j}^{(r)}} \left(t_i^{(r)} \right) \cdot \vec{d}^{(r)} \right\rangle}{s_{i,2}^{(r)}} - \frac{\left\langle \vec{V}_i^{(r)}, \vec{\epsilon}_j \right\rangle}{s_{i,2}^{(r)}} \\
&= \frac{\left\langle \vec{V}_i^{(r)}, \vec{d}^{(r)} \right\rangle}{s_{i,2}^{(r)}} \cdot \frac{\partial}{\partial \mu_{i,j}^{(r)}} t_i^{(r)} - \frac{\vec{V}_{i,j}^{(r)}}{s_{i,2}^{(r)}} \\
&= \frac{\left\langle \vec{V}_i^{(r)}, \vec{d}^{(r)} \right\rangle \cdot \frac{\partial}{\partial \mu_{i,j}^{(r)}} t_i^{(r)} - \vec{V}_{i,j}^{(r)}}{s_{i,2}^{(r)}}
\end{aligned}$$

$$\frac{dt_i^{(r)}}{d\tilde{s}_{i,j}^{(r)}} = 0$$

$$\frac{du_i^{(r)}}{d\tilde{s}_{i,j}^{(r)}} = \frac{d}{d\tilde{s}_{i,j}^{(r)}} \frac{\left\langle \vec{U}_i^{(r)}, P_i'^{(r)} - \mu_i^{(r)} \right\rangle}{\exp\left(\tilde{s}_{i,1}^{(r)}\right)} =$$

$$\begin{aligned}
&= -\frac{\left\langle \vec{U}_i^{(r)}, P_i'^{(r)} - \mu_i^{(r)} \right\rangle}{\exp^2(\tilde{s}_{i,1}^{(r)})} \cdot \frac{d}{d\tilde{s}_{i,j}^{(r)}} \left(\exp(\tilde{s}_{i,1}^{(r)}) \right) \cdot \delta_{j1} = \\
&= -\frac{\left\langle \vec{U}_i^{(r)}, P_i'^{(r)} - \mu_i^{(r)} \right\rangle}{\exp^2(\tilde{s}_{i,1}^{(r)})} \cdot \exp(\tilde{s}_{i,1}^{(r)}) \cdot \delta_{j1} = \\
&= -\frac{\left\langle \vec{U}_i^{(r)}, P_i'^{(r)} - \mu_i^{(r)} \right\rangle}{\exp(\tilde{s}_{i,1}^{(r)})} \cdot \delta_{j1} = \\
&= -\frac{\left\langle \vec{U}_i^{(r)}, P_i'^{(r)} - \mu_i^{(r)} \right\rangle}{\exp(\tilde{s}_{i,1}^{(r)})} \cdot \delta_{j1} = \\
&= -u_i^{(r)} \cdot \delta_{j1}
\end{aligned}$$

$$\begin{aligned}
\frac{dv_i^{(r)}}{d\tilde{s}_{i,j}^{(r)}} &= \frac{d}{d\tilde{s}_{i,j}^{(r)}} \frac{\left\langle \vec{V}_i^{(r)}, P_i'^{(r)} - \mu_i^{(r)} \right\rangle}{\exp(\tilde{s}_{i,2}^{(r)})} = \\
&= -\frac{\left\langle \vec{V}_i^{(r)}, P_i'^{(r)} - \mu_i^{(r)} \right\rangle}{\exp^2(\tilde{s}_{i,2}^{(r)})} \cdot \frac{d}{d\tilde{s}_{i,j}^{(r)}} \left(\exp(\tilde{s}_{i,2}^{(r)}) \right) \cdot \delta_{j2} = \\
&= -\frac{\left\langle \vec{V}_i^{(r)}, P_i'^{(r)} - \mu_i^{(r)} \right\rangle}{\exp^2(\tilde{s}_{i,2}^{(r)})} \cdot \exp(\tilde{s}_{i,2}^{(r)}) \cdot \delta_{j2} = \\
&= -\frac{\left\langle \vec{V}_i^{(r)}, P_i'^{(r)} - \mu_i^{(r)} \right\rangle}{\exp(\tilde{s}_{i,2}^{(r)})} \cdot \delta_{j2} = \\
&= -\frac{\left\langle \vec{V}_i^{(r)}, P_i'^{(r)} - \mu_i^{(r)} \right\rangle}{\exp(\tilde{s}_{i,2}^{(r)})} \cdot \delta_{j2} = \\
&= -v_i^{(r)} \cdot \delta_{j2}
\end{aligned}$$

$$\begin{aligned}
\frac{dt_i^{(r)}}{dq_{i,j}^{(r)}} &= \frac{\frac{\partial}{\partial q_{i,j}^{(r)}} \left(\left\langle \vec{N}_i^{(r)}, \mu_i^{(r)} - o^{(r)} \right\rangle \right) \cdot \left\langle \vec{N}_i^{(r)}, \vec{d}^{(r)} \right\rangle - \left\langle \vec{N}_i^{(r)}, \mu_i^{(r)} - o^{(r)} \right\rangle \cdot \frac{\partial}{\partial q_{i,j}^{(r)}} \left(\left\langle \vec{N}_i^{(r)}, \vec{d}^{(r)} \right\rangle \right)}{\left\langle \vec{N}_i^{(r)}, \vec{d}^{(r)} \right\rangle^2} = \\
&= \frac{\left\langle \frac{\partial}{\partial q_{i,j}^{(r)}} \vec{N}_i^{(r)}, \mu_i^{(r)} - o^{(r)} \right\rangle \cdot \left\langle \vec{N}_i^{(r)}, \vec{d}^{(r)} \right\rangle - \left\langle \vec{N}_i^{(r)}, \mu_i^{(r)} - o^{(r)} \right\rangle \cdot \left\langle \frac{\partial}{\partial q_{i,j}^{(r)}} \vec{N}_i^{(r)}, \vec{d}^{(r)} \right\rangle}{\left\langle \vec{N}_i^{(r)}, \vec{d}^{(r)} \right\rangle^2}
\end{aligned}$$

$$\begin{aligned}
\frac{du_i^{(r)}}{dq_{i,j}^{(r)}} &= \frac{\left\langle \frac{\partial}{\partial q_{i,j}^{(r)}} \vec{U}_i^{(r)}, P_i'^{(r)} - \mu_i^{(r)} \right\rangle + \left\langle \vec{U}_i^{(r)}, \frac{\partial}{\partial q_{i,j}^{(r)}} (P_i'^{(r)} - \mu_i^{(r)}) \right\rangle}{s_{i,1}^{(r)}} = \\
&= \frac{\left\langle \frac{\partial}{\partial q_{i,j}^{(r)}} \vec{U}_i^{(r)}, P_i'^{(r)} - \mu_i^{(r)} \right\rangle + \left\langle \vec{U}_i^{(r)}, \frac{\partial}{\partial q_{i,j}^{(r)}} P_i'^{(r)} \right\rangle}{s_{i,1}^{(r)}} = \\
&= \frac{\left\langle \frac{\partial}{\partial q_{i,j}^{(r)}} \vec{U}_i^{(r)}, P_i'^{(r)} - \mu_i^{(r)} \right\rangle + \left\langle \vec{U}_i^{(r)}, \frac{\partial}{\partial q_{i,j}^{(r)}} (o^{(r)} + t_i^{(r)} \vec{d}^{(r)}) \right\rangle}{s_{i,1}^{(r)}} =
\end{aligned}$$

$$\begin{aligned}
&= \frac{\left\langle \frac{\partial}{\partial \mathbf{q}_{i,j}^{(r)}} \vec{\mathbf{U}}_i^{(r)}, \mathbf{P}'_i^{(r)} - \boldsymbol{\mu}_i^{(r)} \right\rangle + \left\langle \vec{\mathbf{U}}_i^{(r)}, \frac{\partial}{\partial \mathbf{q}_{i,j}^{(r)}} \left(t_i^{(r)} \vec{\mathbf{d}}^{(r)} \right) \right\rangle}{s_{i,1}^{(r)}} = \\
&= \frac{\left\langle \frac{\partial}{\partial \mathbf{q}_{i,j}^{(r)}} \vec{\mathbf{U}}_i^{(r)}, \mathbf{P}'_i^{(r)} - \boldsymbol{\mu}_i^{(r)} \right\rangle + \left\langle \vec{\mathbf{U}}_i^{(r)}, \frac{\partial}{\partial \mathbf{q}_{i,j}^{(r)}} \left(t_i^{(r)} \right) \cdot \vec{\mathbf{d}}^{(r)} \right\rangle}{s_{i,1}^{(r)}} = \\
&= \frac{\left\langle \frac{\partial}{\partial \mathbf{q}_{i,j}^{(r)}} \vec{\mathbf{U}}_i^{(r)}, \mathbf{P}'_i^{(r)} - \boldsymbol{\mu}_i^{(r)} \right\rangle + \left\langle \vec{\mathbf{U}}_i^{(r)}, \vec{\mathbf{d}}^{(r)} \right\rangle \cdot \frac{\partial}{\partial \mathbf{q}_{i,j}^{(r)}} t_i^{(r)}}{s_{i,1}^{(r)}} =
\end{aligned}$$

$$\begin{aligned}
\frac{dv_i^{(r)}}{dq_{i,j}^{(r)}} &= \frac{\left\langle \frac{\partial}{\partial \mathbf{q}_{i,j}^{(r)}} \vec{\mathbf{V}}_i^{(r)}, \mathbf{P}'_i^{(r)} - \boldsymbol{\mu}_i^{(r)} \right\rangle + \left\langle \vec{\mathbf{V}}_i^{(r)}, \frac{\partial}{\partial \mathbf{q}_{i,j}^{(r)}} \left(\mathbf{P}'_i^{(r)} - \boldsymbol{\mu}_i^{(r)} \right) \right\rangle}{s_{i,2}^{(r)}} = \\
&= \frac{\left\langle \frac{\partial}{\partial \mathbf{q}_{i,j}^{(r)}} \vec{\mathbf{V}}_i^{(r)}, \mathbf{P}'_i^{(r)} - \boldsymbol{\mu}_i^{(r)} \right\rangle + \left\langle \vec{\mathbf{V}}_i^{(r)}, \frac{\partial}{\partial \mathbf{q}_{i,j}^{(r)}} \mathbf{P}'_i^{(r)} \right\rangle}{s_{i,2}^{(r)}} = \\
&= \frac{\left\langle \frac{\partial}{\partial \mathbf{q}_{i,j}^{(r)}} \vec{\mathbf{V}}_i^{(r)}, \mathbf{P}'_i^{(r)} - \boldsymbol{\mu}_i^{(r)} \right\rangle + \left\langle \vec{\mathbf{V}}_i^{(r)}, \frac{\partial}{\partial \mathbf{q}_{i,j}^{(r)}} \left(\mathbf{o}^{(r)} + t_i^{(r)} \vec{\mathbf{d}}^{(r)} \right) \right\rangle}{s_{i,2}^{(r)}} = \\
&= \frac{\left\langle \frac{\partial}{\partial \mathbf{q}_{i,j}^{(r)}} \vec{\mathbf{V}}_i^{(r)}, \mathbf{P}'_i^{(r)} - \boldsymbol{\mu}_i^{(r)} \right\rangle + \left\langle \vec{\mathbf{V}}_i^{(r)}, \frac{\partial}{\partial \mathbf{q}_{i,j}^{(r)}} \left(t_i^{(r)} \vec{\mathbf{d}}^{(r)} \right) \right\rangle}{s_{i,2}^{(r)}} = \\
&= \frac{\left\langle \frac{\partial}{\partial \mathbf{q}_{i,j}^{(r)}} \vec{\mathbf{V}}_i^{(r)}, \mathbf{P}'_i^{(r)} - \boldsymbol{\mu}_i^{(r)} \right\rangle + \left\langle \vec{\mathbf{V}}_i^{(r)}, \frac{\partial}{\partial \mathbf{q}_{i,j}^{(r)}} \left(t_i^{(r)} \right) \cdot \vec{\mathbf{d}}^{(r)} \right\rangle}{s_{i,2}^{(r)}} = \\
&= \frac{\left\langle \frac{\partial}{\partial \mathbf{q}_{i,j}^{(r)}} \vec{\mathbf{V}}_i^{(r)}, \mathbf{P}'_i^{(r)} - \boldsymbol{\mu}_i^{(r)} \right\rangle + \left\langle \vec{\mathbf{V}}_i^{(r)}, \vec{\mathbf{d}}^{(r)} \right\rangle \cdot \frac{\partial}{\partial \mathbf{q}_{i,j}^{(r)}} t_i^{(r)}}{s_{i,2}^{(r)}} =
\end{aligned}$$

Recall that:

$$\vec{\mathbf{U}}_i^{(r)} = \left[\mathbf{R}_{i,1,1}^{(r)}, \mathbf{R}_{i,2,1}^{(r)}, \mathbf{R}_{i,3,1}^{(r)} \right]^T$$

$$\vec{\mathbf{V}}_i^{(r)} = \left[\mathbf{R}_{i,1,2}^{(r)}, \mathbf{R}_{i,2,2}^{(r)}, \mathbf{R}_{i,3,2}^{(r)} \right]^T$$

$$\vec{\mathbf{N}}_i^{(r)} = \left[\mathbf{R}_{i,1,3}^{(r)}, \mathbf{R}_{i,2,3}^{(r)}, \mathbf{R}_{i,3,3}^{(r)} \right]^T$$

where:

$$\mathbf{R}_i^{(r)} = \begin{bmatrix} 1 - cc_i^{(r)} - dd_i^{(r)} & bc_i^{(r)} - ad_i^{(r)} & bd_i^{(r)} + ac_i^{(r)} \\ bc_i^{(r)} + ad_i^{(r)} & 1 - bb_i^{(r)} - dd_i^{(r)} & cd_i^{(r)} - ab_i^{(r)} \\ bd_i^{(r)} - ac_i^{(r)} & cd_i^{(r)} + ab_i^{(r)} & 1 - bb_i^{(r)} - cc_i^{(r)} \end{bmatrix}$$

for:

$$s_i^{(r)} = \frac{2}{\left(\mathbf{q}_{i,1}^{(r)} \right)^2 + \left(\mathbf{q}_{i,2}^{(r)} \right)^2 + \left(\mathbf{q}_{i,3}^{(r)} \right)^2 + \left(\mathbf{q}_{i,4}^{(r)} \right)^2}$$

$$\begin{aligned}
bs_i^{(r)} &= q_{i,2}^{(r)} \cdot s_i^{(r)}, & cs_i^{(r)} &= q_{i,3}^{(r)} \cdot s_i^{(r)}, & ds_i^{(r)} &= q_{i,4}^{(r)} \cdot s_i^{(r)} \\
ab_i^{(r)} &= q_{i,1}^{(r)} \cdot bs_i^{(r)}, & ac_i^{(r)} &= q_{i,1}^{(r)} \cdot cs_i^{(r)}, & ad_i^{(r)} &= q_{i,1}^{(r)} \cdot ds_i^{(r)} \\
bb_i^{(r)} &= q_{i,2}^{(r)} \cdot bs_i^{(r)}, & bc_i^{(r)} &= q_{i,2}^{(r)} \cdot cs_i^{(r)}, & bd_i^{(r)} &= q_{i,2}^{(r)} \cdot ds_i^{(r)} \\
cc_i^{(r)} &= q_{i,3}^{(r)} \cdot cs_i^{(r)}, & cd_i^{(r)} &= q_{i,3}^{(r)} \cdot ds_i^{(r)}, & dd_i^{(r)} &= q_{i,4}^{(r)} \cdot ds_i^{(r)}
\end{aligned}$$

Let us define $aa := \left(q_{i,1}^{(r)}\right)^2 \cdot s_i^{(r)}$, $a_i^{(r)} := q_{i,1}^{(r)}$, $b_i^{(r)} := q_{i,2}^{(r)}$, $c_i^{(r)} := q_{i,3}^{(r)}$, and $d_i^{(r)} := q_{i,4}^{(r)}$. Then, after straightforward yet tedious calculations, we finally obtain:

$$\begin{aligned}
\frac{dR_i^{(r)}}{dq_{i,1}^{(r)}} &= \begin{bmatrix} s_i^{(r)} a_i^{(r)} \left((1 - aa_i^{(r)}) + (1 - bb_i^{(r)}) \right) & s_i^{(r)} \left(-d_i^{(r)} (1 - aa_i^{(r)}) - ab_i^{(r)} \cdot c_i^{(r)} \right) & s_i^{(r)} \left(c_i^{(r)} (1 - aa_i^{(r)}) - ab_i^{(r)} \cdot d_i^{(r)} \right) \\ s_i^{(r)} \left(d_i^{(r)} (1 - aa_i^{(r)}) - ab_i^{(r)} \cdot c_i^{(r)} \right) & s_i^{(r)} a_i^{(r)} \left((1 - aa_i^{(r)}) + (1 - cc_i^{(r)}) \right) & s_i^{(r)} \left(-b_i^{(r)} (1 - aa_i^{(r)}) - a_i^{(r)} \cdot cd_i^{(r)} \right) \\ s_i^{(r)} \left(-c_i^{(r)} (1 - aa_i^{(r)}) - ab_i^{(r)} \cdot d_i^{(r)} \right) & s_i^{(r)} \left(b_i^{(r)} (1 - aa_i^{(r)}) - a_i^{(r)} \cdot cd_i^{(r)} \right) & s_i^{(r)} a_i^{(r)} \left((1 - aa_i^{(r)}) + (1 - dd_i^{(r)}) \right) \end{bmatrix} \\
\frac{dR_i^{(r)}}{dq_{i,2}^{(r)}} &= \begin{bmatrix} s_i^{(r)} b_i^{(r)} \left((1 - bb_i^{(r)}) + (1 - aa_i^{(r)}) \right) & s_i^{(r)} \left(c_i^{(r)} (1 - bb_i^{(r)}) - (-ab_i^{(r)} \cdot d_i^{(r)}) \right) & s_i^{(r)} \left(d_i^{(r)} (1 - bb_i^{(r)}) - ab_i^{(r)} \cdot c_i^{(r)} \right) \\ s_i^{(r)} \left(c_i^{(r)} (1 - bb_i^{(r)}) - ab_i^{(r)} \cdot d_i^{(r)} \right) & -s_i^{(r)} b_i^{(r)} \left((1 - bb_i^{(r)}) + (1 - dd_i^{(r)}) \right) & s_i^{(r)} \left(-a_i^{(r)} (1 - bb_i^{(r)}) - b_i^{(r)} \cdot cd_i^{(r)} \right) \\ s_i^{(r)} \left(d_i^{(r)} (1 - bb_i^{(r)}) - (-ab_i^{(r)} \cdot c_i^{(r)}) \right) & s_i^{(r)} \left(a_i^{(r)} (1 - bb_i^{(r)}) - b_i^{(r)} \cdot cd_i^{(r)} \right) & -s_i^{(r)} b_i^{(r)} \left((1 - bb_i^{(r)}) + (1 - cc_i^{(r)}) \right) \end{bmatrix} \\
\frac{dR_i^{(r)}}{dq_{i,3}^{(r)}} &= \begin{bmatrix} -s_i^{(r)} c_i^{(r)} \left((1 - cc_i^{(r)}) + (1 - dd_i^{(r)}) \right) & s_i^{(r)} \left(b_i^{(r)} (1 - cc_i^{(r)}) - (-a_i^{(r)} \cdot cd_i^{(r)}) \right) & s_i^{(r)} \left(a_i^{(r)} (1 - cc_i^{(r)}) - b_i^{(r)} \cdot cd_i^{(r)} \right) \\ s_i^{(r)} \left(b_i^{(r)} (1 - cc_i^{(r)}) - a_i^{(r)} \cdot cd_i^{(r)} \right) & s_i^{(r)} c_i^{(r)} \left((1 - cc_i^{(r)}) + (1 - aa_i^{(r)}) \right) & s_i^{(r)} \left(d_i^{(r)} (1 - cc_i^{(r)}) - (-ab_i^{(r)} \cdot c_i^{(r)}) \right) \\ s_i^{(r)} \left(-a_i^{(r)} (1 - cc_i^{(r)}) - b_i^{(r)} \cdot cd_i^{(r)} \right) & s_i^{(r)} \left(d_i^{(r)} (1 - cc_i^{(r)}) - ab_i^{(r)} \cdot c_i^{(r)} \right) & -s_i^{(r)} c_i^{(r)} \left((1 - cc_i^{(r)}) + (1 - bb_i^{(r)}) \right) \end{bmatrix} \\
\frac{dR_i^{(r)}}{dq_{i,4}^{(r)}} &= \begin{bmatrix} -s_i^{(r)} d_i^{(r)} \left((1 - dd_i^{(r)}) + (1 - cc_i^{(r)}) \right) & s_i^{(r)} \left(-a_i^{(r)} (1 - dd_i^{(r)}) - b_i^{(r)} \cdot cd_i^{(r)} \right) & s_i^{(r)} \left(b_i^{(r)} (1 - dd_i^{(r)}) - a_i^{(r)} \cdot cd_i^{(r)} \right) \\ s_i^{(r)} \left(a_i^{(r)} (1 - dd_i^{(r)}) - b_i^{(r)} \cdot cd_i^{(r)} \right) & -s_i^{(r)} d_i^{(r)} \left((1 - dd_i^{(r)}) + (1 - bb_i^{(r)}) \right) & s_i^{(r)} \left(c_i^{(r)} (1 - dd_i^{(r)}) - (-ab_i^{(r)} \cdot d_i^{(r)}) \right) \\ s_i^{(r)} \left(b_i^{(r)} (1 - dd_i^{(r)}) - (-a_i^{(r)} \cdot cd_i^{(r)}) \right) & s_i^{(r)} \left(c_i^{(r)} (1 - dd_i^{(r)}) - ab_i^{(r)} \cdot d_i^{(r)} \right) & s_i^{(r)} d_i^{(r)} \left((1 - dd_i^{(r)}) + (1 - aa_i^{(r)}) \right) \end{bmatrix}
\end{aligned}$$