Relighting Gaussian Splats: An overview

A Thesis

submitted to the designated
by the Assembly
of the Department of Computer Science and Engineering
Examination Committee

by

Aristeidis Panagiotidis Diktampanis

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN DATA AND COMPUTER SYSTEMS ENGINEERING

WITH SPECIALIZATION
IN ADVANCED COMPUTER SYSTEMS

University of Ioannina
School of Engineering
Ioannina 2025

Examining Committee:

- **Ioannis Fudos**, Professor, Department of Computer Science and Engineering, University of Ioannina (Advisor)
- **Vassilios Dimakopoulos**, Professor, Department of Computer Science and Engineering, University of Ioannina
- **Christophoros Nikou**, Professor, Department of Computer Science and Engineering, University of Ioannina

Acknowledgements

Areti. The Computer Graphics Research Group; my supervisor, professor Ioannis Fudos, PhD candidates Christos Georgiou Mousses and Grigoris Tsopouridis. The faculty of the Department of Computer Science & Engineering. My colleagues from the program of graduate studies throughout the years. The Georges, Voudiotis and Zisopoulos. My team, Polyxeni Mantzou, Xenofon Bitsikas. My favorite colleague, my friend, Anastasis. My dearest, Alexandros, Eytyxia, Konstantinos, Yakinthi. Dimitri, Giouli, Ilias, Foteini. My beloved parents; Despoina, Panagiotis. My grandparents; Elisavet, Giorgos. All of you on the 15th. Areti.

My deepest appreciation. My most heartfelt gratitude.
Even when I was alone, I wasn't. For that, and a million reasons more; thank you.

Table of Contents

List of Figures

List of Tables

Abstract

Aristeidis Panagiotidis Diktampanis,
M.Sc. in Data and Computer Systems Engineering, Department of Computer Science and Engineering, School of Engineering, University of Ioannina, Greece, 2025
Thesis title: Relighting Gaussian Splats: An overview
Advisor: Ioannis Fudos, Professor

The emergence of 3D Gaussian Splatting in 3D scene reconstruction has led to a wealth of research with diverse applications in both computer vision and graphics. A less explored area within this field is the relighting of reconstructed scenes under various lighting conditions, such as moving, adding, removing, or altering light sources. In this thesis, we aim to explore different approaches to relighting within the Gaussian Splatting framework from a computer graphics perspective. By focusing on five distinct projects that propose various solutions to this problem, we intend to provide an overview of the suggested techniques and an evaluation of their results. We begin by analyzing the concept of relighting in 3D Gaussian Splatting, breaking it down into its fundamental components. This analysis includes examining different methods of scene representation and novel view synthesis, as well as addressing the mathematical formulation of light transport, visibility computation, and shading pipelines. Our goal is to provide the necessary theoretical background to understand and evaluate the different approaches. Next, we delve into the specifics and peculiarities of each project's method by carefully examining their pipelines and noting their characteristics. Finally, we apply these methods to five diverse datasets that encompass a wide range of materials, reflective properties, and scene structures. We present our qualitative and quantitative results, along with a comparison of the strengths and weaknesses of each approach.

Εκτεταμενη Περιληψη

Αριστείδης Παναγιωτίδης Δικταμπάνης, Δ.Μ.Σ. στη Μηχανική Δεδομένων και Υπολογιστικών Συστημάτων, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πολυτεχνική Σχολή, Πανεπιστήμιο Ιωαννίνων, 2025.
Τίτλος διατριβής: Επαναφωτισμός με Gaussian Splats: Μια επισκόπηση
Επιβλέπων: Ιωάννης Φούντος, Καθηγητής

Η έλευση του 3D Gaussian Splatting στην ανακατασκευή τρισδιάστατων σκηνών έχει οδηγήσει σε μια πληθώρα έργων με ποικίλες εφαρμογές τόσο στην υπολογιστική όραση όσο και στα γραφικά υπολογιστών. Ένα λιγότερο μελετημένο θέμα είναι ο επαναφωτισμός μιας ανακατασκευασμένης σκηνής υπό διαφορετικές συνθήκες φωτισμού (μετακίνηση, προσθήκη/αφαίρεση ή τροποποίηση πηγών φωτός). Με αυτή τη διατριβή στοχεύουμε να διερευνήσουμε τις διαφορετικές προσεγγίσεις στον επαναφωτισμό με Gaussian Splatting από την οπτική των γραφικών και των δικών τους κριτηρίων και προεκτάσεων. Εστιάζοντας σε πέντε ξεχωριστά project που προτείνουν διαφορετικές λύσεις στο πρόβλημα του επαναφωτισμού, σκοπεύουμε να παρέχουμε μια επισκόπηση των προτεινόμενων τεχνικών και μια αξιολόγηση των αποτελεσμάτων τους. Ξεκινάμε αναλύοντας αυτό το γενικό πρόβλημα επαναφωτισμού με 3DGS στα δομικά του συστατικά. Από τις διαφορετικές μεθόδους αναπαράστασης σκηνών και το Novel View Synthesis έως τη μαθηματική διατύπωση της μεταφοράς φωτός, τον υπολογισμό ορατότητας και τις μεθόδους σκίασης, στοχεύουμε να παρέχουμε το απαραίτητο θεωρητικό υπόβαθρο που θα μας βοηθήσει να κατανοήσουμε και να αξιολογήσουμε τις διαφορετικές αυτές προσεγγίσεις. Στη συνέχεια, αναλύουμε κάθε project στα συστατικά του, εξετάζοντας προσεκτικά τη μεθοδολογία του και σημειώνοντας τα χαρακτηριστικά του. Τέλος, εφαρμόζουμε σε πέντε διαφορετικά σύνολα δεδομένων που περιέχουν ένα ευρύ φάσμα υλικών, ανακλαστικών ιδιοτήτων και δομών σκηνής. Αναφέρουμε τα ποιοτικά και ποσοτικά μας αποτελέσματα και τα συγκρίνουμε μεταξύ τους αναφέροντας τις δυνατότητες και τις αδυναμίες τους.

CHAPTER 1

BACKGROUND

1.1 Scene Representation

3D representations define how geometry and appearance are stored, accessed, and transformed across different stages of a computer graphics pipeline. They are, in other words, different ways of describing a scene so that we can represent or work with it digitally. Each representation can be better or worse in terms of memory efficiency, rendering speed, expressiveness, and suitability for downstream tasks in learning and optimization environments. Over the years, the different approaches that have been introduced tend to fall in one of two categories. Explicit and implicit representations. There have been alternative methods[1], [2] that fuse characteristics from both categories, they are however expanding on already established methods and are beyond our scope and necessity. In this chapter, we will introduce the two main categories and the most common methods that comprise them. We will also introduce the 3D Gaussian representation and the incentive behind it.

1.1.1 Explicit representations

Explicit representations store geometric information explicitly. That means that once the data used to describe a scene, like for example the 3D coordinates of edges, are loaded from memory, no extra computation is needed to understand its structure. They are among the most widely used formats in 3D geometry and in this section, we will discuss the three most common methods: point clouds, polygonal meshes and voxel grids. We will come across their many advantages but also their pain points that alternative representation methods are trying to relieve.

1.1.1.1 Point Cloud

Point clouds represent geometry as a collection of discrete points in three-dimensional space. Each point is defined by its coordinates (x, y, z) and may include additional attributes like color or surface normals[3]. They are typically produced by 3D scanning methods like LiDAR, depth cameras or Structure-from-Motion (SfM) reconstruction pipelines.

Unlike volumetric or mesh-based formats, point clouds are inherently unstructured, meaning they do not encode topological relationships between points. This makes them particularly simple to generate and interpret, especially in applications where raw geometric data is sufficient. However, processing tasks like segmentation, recognition, and reconstruction pose significant challenges[4] due to their unstructured nature in combination with the fact that point clouds can be sparce or contain noise.

1.1.1.2 Mesh

A mesh models the surface of a 3D object using a set of vertices, edges, and faces. The vertices define points in 3D space. Edges connect pairs of vertices. Faces, usually triangles or quadrilaterals, span those edges to form flat surfaces. Unlike point clouds, which consist of unconnected points, meshes encode relationships between points. They carry geometric and topological information that enables the efficient computation of surface properties like normal and curvature which are essential for rendering and for evaluating light interactions. They are expressive, as in, they can approximate complex geometries with arbitrary precision simply by adjusting the number of vertices and faces. Moreover, meshes are well-supported by graphics hardware making them popular for rendering and simulation tasks[3].

Despite their strengths, meshes have limitations. They require clean, watertight geometry and well-defined topology to function reliably in lighting and rendering tasks, which can be difficult to obtain from real-world data. They are also inherently surface-based, making it hard to represent translucent materials or volumetric effects.

1.1.1.3 Voxel

A voxel, short for "volume element," is the 3D analogue of a pixel[3]. Just as an image is divided into a grid of rectangles in two dimensions, a volume can be subdivided into a regular grid, with each one representing a small portion of space. Each voxel represents a fixed location, and is typically visualized as a small cube, although voxels don’t have to be cubic[5]. They can be tetrahedral[6], prisms or whatever shape your marching algorithm can support[7]. Each voxel stores information about its position on the grid and may carry one or more values depending on the context. In some applications, a single scalar, for opacity or density is enough. In others, it may include color, surface normals, or multiple data channels describing material properties or physical measurements. Strictly speaking, voxels mark a single point on the grid rather than the volume between points. In a voxel-based dataset, the space between each voxel is not represented. Common ways to reconstruct the volume are interpolation or Signed Distance Functions (SDF) which we will discuss

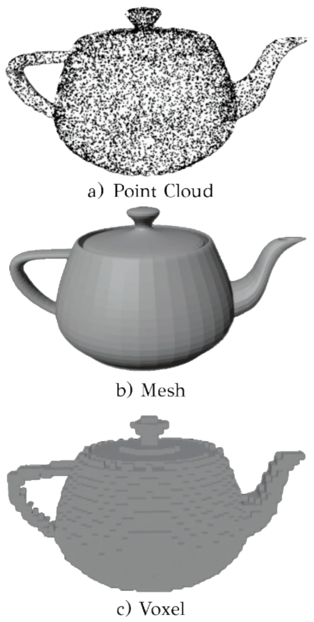

Figure 1.1 Explicit 3D representations. 3D model from [40].

a bit more in implicit representations. The likes of Fig 1.1 or Minecraft though prove that voxels can be expressive, or even imply aesthetic, simply as cubic explicit representations.

1.1.2 Implicit representations

Implicit representations describe shape and appearance through continuous functions. Rather than storing surfaces directly, they use a function that tells whether a point in space lies inside or outside an object, or what color and density are present at a location.

1.1.2.1 Signed Distance Function

A common implementation of an implicit representation is provided by a Signed Distance Function (SDF). SDF is a function $f(p): R^3 \rightarrow R$ that tells us how far a point p is from the surface of some solid and which side it is on. Mathematically, it can be expressed as:

$$f(p) = min_{x \in S} \|p - x\|, \qquad with\ sign(f(p)) = \begin{cases} > 0, & p\ outside\ S \\ < 0, & p\ inside\ S \end{cases} \tag{1.1}$$

where S is the object surface. Because f measures the shortest Euclidean distance to that surface, the shape itself is its zero-level set $f(p) = 0$. SDF satisfies the Eikonal equation[8] $\|\nabla f(p)\| = 1$ almost everywhere, meaning the gradient vector at any point points exactly toward the nearest surface. This property makes SDF suitable for gradient-based optimization and differentiable rendering pipelines.

In practice, these functions are rarely hand-crafted for complex shapes, instead they are typically learned with a multilayer perceptron (MLP)[3]. DeepSDF[9] first demonstrated that a compact MLP conditioned on a latent vector can learn a continuous, watertight distance field directly from partial or noisy scans. Then, it recovers crisp meshes via Marching Cubes with far lower memory requirements than voxels or dense point clouds. Building on this, Articulated SDFs[10] (A-SDFs) disentangle shape and pose in separate latent codes, enabling on-the-fly deformation of learned characters without retraining. AutoSDF[11] incorporates an autoregressive shape prior that unifies 3D completion, reconstruction, and unconditional generation within a single model. SDF-StyleGAN[12] adapts StyleGAN2's hierarchical latents to an implicit distance field. It uses specialized discriminators targeting both SDF values and gradients to significantly boost the visual fidelity and geometric accuracy of generated shapes. LAS-Diffusion[13] brings view-aware local attention into a diffusion framework over SDFs, offering sketch-driven, controllable 3D synthesis that respects fine local details. Finally, Shap-E[14] from OpenAI jointly generates SDF parameters and NeRF appearance fields, achieving rapid convergence and multi-representation outputs that bridge geometry and view synthesis in one network.

SDFs aren't without compromises. They demand watertight geometry since holes or non-manifold edges can flip the sign convention. Querying a learned SDF via a deep MLP can be computationally heavy if we need dense meshes, prompting research into hybrid grids or spatial hash encodings for faster evaluation. Moreover, the medial axis, i.e., points equidistant to multiple patches, can introduce gradient discontinuities that trip up naïve optimizers.

1.1.2.2 Neural Radiance Fields

Another, newer example of implicit representations is the Neural Radiance Field (NeRF)[15]. Neural Radiance Fields (NeRF) propose different approach to novel-view synthesis by directly learning a continuous volumetric representation of a scene, rather than relying on explicit geometry or densely sampled light fields. NeRF models a static scene as a 5D vector-valued function:

$$F_{\Theta} = (x, y, z, \theta, \varphi) \mapsto (\sigma, c) \tag{1.2}$$

where (x, y, z) are the spatial 3D coordinates of a point, (φ, θ) is the 2D viewing direction, $c = (r, g, b)$ is the view-dependent emitted color and σ is the volume density (opacity). This function is approximated by a multi-layer perceptron (MLP) whose weights Θ are optimized so that, when combined with classical volume rendering, the rendered images match a sparse set of RGB images/photographs.

At rendering time, each pixel's color $C(r)$ is computed by casting a camera ray $r(t) = o + td$ through the scene near and far bounds $[t_n, t_f]$ and sampling N points $\{t_i\}$ along the ray. For each sample, the MLP is queried to produce (σ_i, c_i). These values are then composited via the volume rendering integral that is approximated by:

$$\hat{C}(r) = \sum_{i=1}^{N} T_i(1 - e^{-\sigma_i\delta_i})\, c_i, \qquad where\ T_i = \exp\,(\sum_{j=1}^{i-1} \sigma_j\delta_j) \tag{1.3}$$

with $\delta_i = t_{i+1} - t_i$ being the distance between adjacent samples.

This process is fully differentiable, so by minimizing the photometric error between $\hat{C}(r)$ and the ground truth pixel colors across training rays we can adjust the MLP parameters so that its density and radiance outputs reproduce the real scene's geometry and appearance.

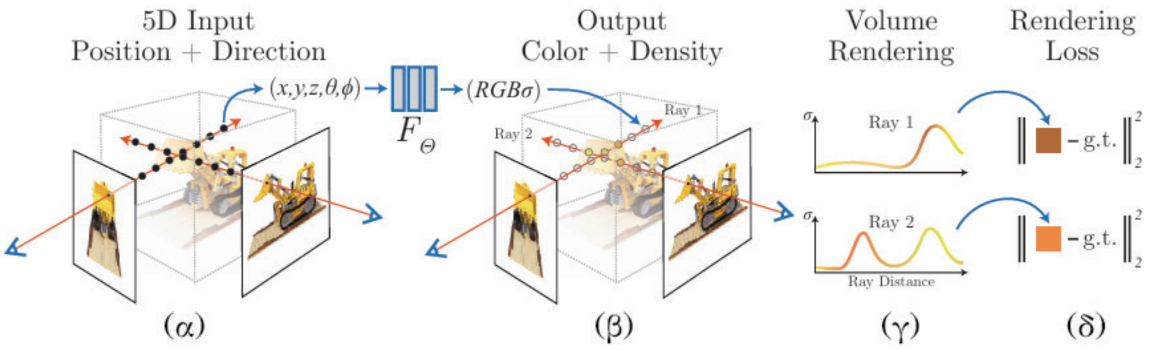

Figure 1.2 NeRF pipeline. Figure from [15].

NeRF proposes two optimizations to lessen the number of samples. First, each scalar input coordinate is lifted via a sinusoidal positional encoding which mitigates the MLP's low-frequency bias and enables sharp geometry and view-dependent effects. Second, NeRF employs a two-stage hierarchical sampling: a "coarse" pass uses stratified sampling to identify regions of interest along each ray, and a "fine" pass draws additional samples from a probability density function derived from the coarse output, concentrating evaluations where they matter most.

In practice, training NeRF requires only a set of RGB images, associated camera poses and intrinsics, and scene bounds that are often estimated via structure-from-motion for real scenes. At each iteration, batches of several thousand rays are sampled; for each ray, the coarse network is queried at N_c locations and the fine network at an additional N_f locations. The total loss is the sum of squared errors between both coarse and fine renderings and the observed pixel colors, optimized via Adam over hundreds of thousands of iterations.

Vanilla NeRFs do come with limitations. Training a separate MLP per scene typically takes one to two days on a modern GPU, restricting it from interactive or real-time applications. The method assumes static scenes with known camera poses; handling dynamic content, unbounded outdoor environments, or unknown poses requires substantial extensions. Moreover, because appearance and illumination remain entangled in the learned radiance field, NeRF does not support relighting or explicit material editing. Lastly, the uniform treatment of space makes scaling to very large or highly detailed scenes challenging, prompting a rich body of follow-on work on multi-resolution grids, sparse-voxel and hash-based encodings, and hardware-accelerated inference to address these bottlenecks.

Each representation comes with specific trade-offs. Meshes are typically memory-efficient and render quickly using standard graphics pipelines, but they require clean geometry and are less suited for representing fuzzy or transparent regions. Voxel grids are easy to work with and support volumetric effects, but they use large amounts of memory, especially at high resolutions. Point clouds are lightweight and simple to capture, but their lack of structure makes them harder to use in tasks like shading or surface reconstruction. Implicit representations, like neural radiance fields, are compact and highly expressive, capturing fine detail and view-dependent effects, but they are slow to render and difficult to edit directly. Representations like 3D Gaussians aim to strike a balance by supporting fast, parallel rendering and integrating well with gradient-based optimizations, making them suitable for learning-based pipelines.

1.2 Novel View Synthesis

Novel view synthesis addresses the problem of generating photorealistic images of a scene from previously unseen camera positions, given only a sparse set of posed input photographs. The essential challenge is to infer both the unseen 3D structure and the appearance handling occlusions, view-dependent lighting, and fine detail so that the rendered views closely match what a real camera would see. Below, we survey the principal paradigms[16], explaining each approach, its core concepts, and its practical trade-offs.

1.2.1 Active 3D Reconstruction Methods

Active techniques project controlled signals into the scene and recover depth directly from their interaction i.e., how these signals are "bouncing" back to the sensor. Computed tomography[17] (CT) acquires many X-ray projections and reconstructs volumetric density via inverse Radon transforms, yielding highly accurate internal and external models that are widely used in medical imaging and industrial inspection, but impractical for everyday environments due to radiation and equipment constraints. Structured light[18] systems cast known patterns (e.g., stripes or dots) onto the scene and depth is inferred by triangulating the distortion of these patterns in the captured image. This delivers real-time performance and sub-millimeter accuracy in controlled lighting but fails under strong ambient illumination or on featureless surfaces. Time-of-flight (ToF)[19] cameras send out infrared pulses and measure the return time per pixel, giving direct depth maps with real-time performance. Their trade-off is lower spatial resolution and multipath interference in complex scenes. LiDAR/laser scanning[20], [21] sweeps a laser beam and times its reflection, producing very precise point clouds even over large distances, but typically requires larger, more expensive hardware and suffers from occlusion shadows in cluttered environments.

1.2.2 Passive 3D Reconstruction Methods

Passive approaches rely solely on multiple photographs to estimate depth and geometry. Mimicking human depth perception, binocular stereo[22] uses two cameras: it finds matching features in both images and triangulates each match into 3D points; it is straightforward but degrades in textureless or repetitive regions. Structure-from-motion (SfM)[23] extends this to many unstructured images, recovering both camera poses and a sparse point cloud via feature matching and bundle adjustment; multi-view stereo (MVS)[24] then densifies these into depth maps or meshes. While these pipelines work with consumer cameras and can handle large scenes, they often produce noisy reconstructions with holes and require careful post-processing to fuse depth maps into watertight meshes.

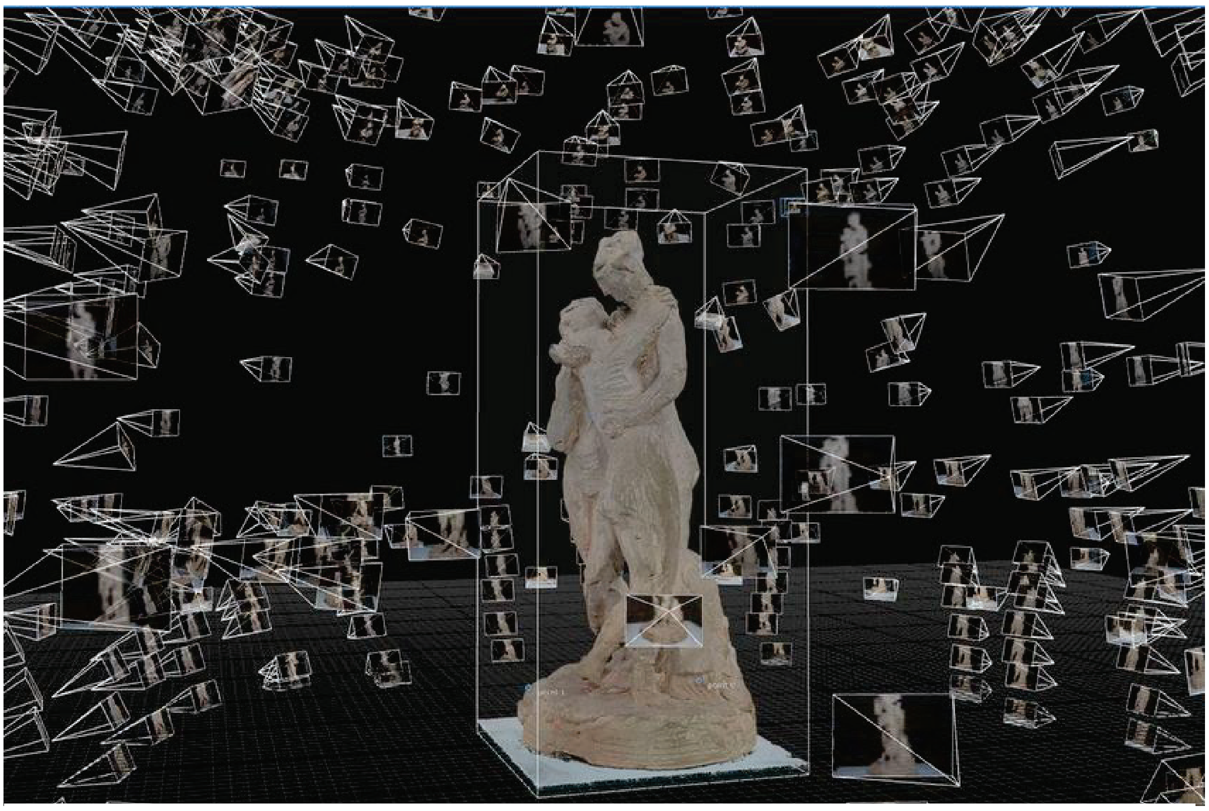
Figure 1.3 Point cloud of Three Graces by Antonio Canova. Each triangle represents a camera position. Image by Factum Arte[28].

1.2.3 3D Reconstruction Methods Based on Machine Learning

Instead of building meshes or voxel grids, implicit-neural methods learn a continuous function that tells us, for any point in space (and optionally a viewing direction), whether there's matter there and what color it emits. Occupancy networks[25] train a small neural network to answer, "inside or outside?" for any 3D point, yielding smooth shapes that naturally handle holes and topology changes. Signed Distance Functions (SDFs) go a step further by predicting the exact distance to the nearest surface (negative if inside), which is useful for collision, Boolean operations, and clean mesh extraction. Neural Radiance Fields (NeRF)[15] combine density (how much light something blocks) and radiance (color emitted along a direction) into one function of position plus direction; using differentiable volume rendering, NeRF learns directly from posed color images and then synthesizes novel views with rich detail and soft shadows. The trade-off is training time (often hours per scene) and slow ray-marching at render time.

1.2.4 Hybrid and Real-Time Accelerations

To bring neural quality toward real-time, hybrid methods merge explicit primitives with learned models. 3D Gaussian Splatting[26] represents the scene as a cloud of tiny anisotropic Gaussians each carrying position, shape (covariance), color, and opacity and splats them onto the screen with GPU shaders. This delivers interactive frame rates with smooth interpolation and view-dependence, though one must manage how to add, merge, or remove Gaussians as the scene changes. Hash-grid encodings (e.g. InstantNGP[27]) store learned feature vectors in a multi-resolution hash table indexed by 3D position, replacing expensive positional encodings and yielding sub-second training without sacrificing fine detail; however, large scenes can inflate memory requirements.

1.2.5 Dynamic and Large-Scale Extensions

Real-world applications often involve moving objects or expansive environments. Dynamic NeRFs (like D-NeRF[28] or Nerfies[29]) introduce time or deformation parameters into the radiance field, letting the model learn how geometry and appearance evolve; these capture non-rigid motion but need dense temporal sampling and additional regularization to prevent flicker. SLAM-based pipelines (e.g., KinectFusion[30] variants) fuse live RGB-D streams into voxel or Gaussian maps for interactive mapping and tracking, trading off global consistency for immediate feedback. Block-wise NeRFs partition a large scene into spatial tiles or octrees, training and rendering each subregion separately; this scales to city-scale capture but requires strategies to blend block boundaries seamlessly and manage loading latency.

Each paradigm involves trade-offs among capture complexity, reconstruction fidelity, training/inference cost, and rendering speed. Active sensors offer precise depth but at higher hardware and setup cost; passive multi-view methods work with simple cameras but struggle under sparse sampling or complex materials; implicit-neural models yield unmatched visual quality yet demand significant compute; and hybrids aim for real-time performance at the expense of algorithmic and representational complexity. Selecting the right pipeline therefore depends on the specific application's requirements for accuracy, speed, cost, and scene dynamics.

1.3 Gaussian Splatting

Gaussian Splatting[26] is a real-time scene representation and rendering method that models a 3D scene as a set of anisotropic Gaussian distributions. Each Gaussian is parameterized by its position, orientation, scale (in the form of a covariance matrix), opacity, and spherical harmonics coefficients for view-dependent color.

The representation is initialized from sparse point clouds, often obtained from Structure-from-Motion (SfM), and optimized through gradient descent for geometry reconstruction. Gaussian Splatting makes use of a tile-based rasterizer that efficiently blends the Gaussians respecting visibility and depth order while maintaining differentiability. This way, we can achieve real-time rendering contrasting traditional NeRF-style methods that rely on expensive volumetric ray-marching.

In this chapter we will formally introduce the mathematical definition of 3D Gaussian splats, describe the process of projecting them into screen space, and discuss their blending and optimization within a differentiable framework. We will also present some extensions to the base approach and introduce the fundamental principles of View Synthesis.

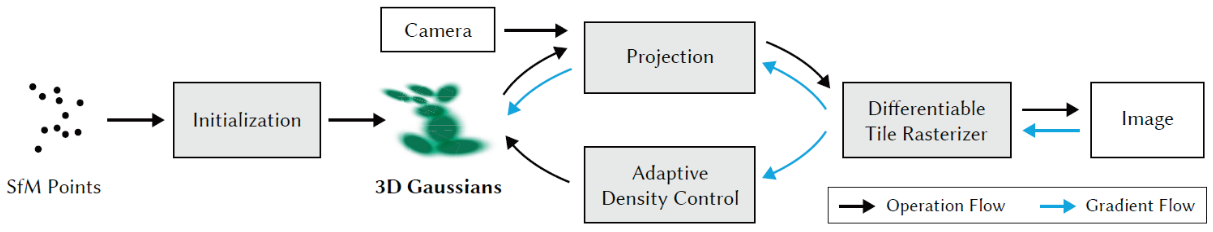

Figure 1.4 3D Gaussian Splatting pipeline. Figure from [26].

1.3.1 Definition and Properties of 3D Gaussians

A 3D Gaussian function is mathematically defined as:

$$G(x) = e^{\frac{1}{2}(x)^T \Sigma^{-1}(x)} \tag{1.4}$$

where $x \in R^3$ represents a point in world coordinates, and $\Sigma \in R^{3\times3}$ is the covariance matrix controlling the spread and orientation of the Gaussian.

Σ is required to be symmetric and positive semi-definite, i.e., all its eigenvalues must be non-negative, to define valid ellipsoidal Gaussians. It is factorized as:

$$\Sigma = RSS^TR^T \tag{1.5}$$

where R is a rotation matrix derived from a unit quaternion q, and S is a diagonal scaling matrix constructed from a scale vector s. This way, we avoid directly optimizing the covariance matrix Σ, which is difficult to constrain during gradient descent. Instead, we split Σ into separate scale and rotation components, each with natural constraints: scaling is limited to positive values while rotation is kept valid through quaternion normalization.

Each Gaussian is parameterized by:

- Position (μ): the 3D center of the Gaussian.
- Scale (s): a 3D vector specifying the scaling along the principal axes.
- Rotation (q): a unit quaternion representing orientation, converted into R for constructing S.
- Opacity (a): a scalar representing the Gaussian's transparency, used during blending.

The directional appearance (color) of each Gaussian is encoded via spherical harmonics (SH) coefficients c. These SH coefficients are optimized alongside the geometric parameters to model the view-dependent radiance properties of the scene.

1.3.2 Differentiable Rendering Pipeline

1.3.2.1 Projection from 3D to 2D Space

The covariance matrix Σ is transformed into the camera's image space using:

$$\Sigma' = JW\Sigma W^TJ^T \tag{1.6}$$

where W is the camera projection matrix, incorporating the transformation from world space to the image plane and J is the Jacobian matrix of the affine approximation of the projection function W[31]. The Jacobian approximates how the projection matrix transforms points near μ, ensuring that anisotropic scaling and rotation are accurately handled in the image plane.

After applying this transformation, the projection simplifies by removing the third row and column of Σ', resulting in a 2×2 covariance matrix that defines the shape and orientation of the Gaussian's elliptical splat in screen space. The mean position μ of each Gaussian is projected into 2D using the projection matrix W.

1.3.2.2 Rasterization and Blending

Point-based rendering provides a fast way to render point clouds but struggles with holes, aliasing, and discontinuities. To overcome these, splatting approaches use primitives like ellipsoids or surfels, which reduce aliasing and improve continuity. However, these methods still depend on Multi-View-Stereo (MVS) generated geometry which introduces reconstruction errors in complex or featureless regions. Neural point-based renderers have improved realism and performance but inherit these limitations and often suffer from instability.

Gaussian Splatting builds on a shared idea found in both point-based rendering and volumetric methods like NeRF: the way images are formed by accumulating color along a viewing ray. The final color C of a pixel is calculated by adding the contributions of multiple samples along the ray:

$$C = \sum_{i=1}^{N} T_i \alpha_i c_i \tag{1.7}$$

where c_i represents the color of sample i, α_i is its opacity and T_i is the transmittance accounting for how much light is transmitted from sample i to the camera. Opacity α_i is given by the equation:

$$\alpha_i = 1 - \exp(-\sigma_i \delta_i) \tag{1.8}$$

where σ_i is the sample's density and δ_i the distance between samples. Transmittance T_i represents the accumulated light transmission from previous samples along the ray and is given by:

$$T_i = \prod_{j=1}^{i-1} (1 - \alpha_j) \tag{1.9}$$

This method expresses how light from different parts of a scene contributes to the pixel color, with closer or denser samples blocking light from farther ones. This compositing model describes the blending of Gaussians. The blending process is differentiable, allowing gradients to propagate through rasterization and compositing for optimizing all Gaussian parameters.

To implement this, Gaussian Splatting introduces a tile-based rasterization approach combined with fast sorting and blending. The screen is divided into 16×16 pixel tiles, and for each tile, projected Gaussians overlapping the tile are identified. A culling process eliminates Gaussians whose 99% confidence interval does not intersect the view frustum, and a guard band excludes those whose means are near the near plane or far outside the view frustum. Each "surviving" Gaussian is instantiated per overlapping tile and assigned a composite key combining its view-space depth and tile ID. This enables a fast GPU Radix sort of all Gaussians across the screen, avoiding the need for per-pixel sorting.

After sorting, each tile receives a depth-sorted list of Gaussians. A thread block processes each tile, loading Gaussian packets into shared memory. For each pixel, the threads accumulate color and opacity from front-to-back in the sorted list. Processing for a pixel stops when its opacity reaches saturation ($\alpha = 1$), and a tile finishes when all pixels saturate.

For backpropagation, the system reconstructs the forward blending sequence by re-traversing the sorted list, without needing to store long per-pixel lists. In the backward pass, traversal is back-to-front, with processing starting only if the Gaussian's depth is less than or equal to the last contributing splat's depth. The final accumulated opacity of each pixel, stored during the forward pass, is used to recover the intermediate opacities needed for gradient computation by dividing by the individual splat's opacity.

1.3.2.3 Differentiability and Optimization

The optimization process uses Stochastic Gradient Descent (SGD). It proceeds iteratively, alternating between rendering and comparing the synthesized image to ground-truth views in the training dataset. Since initial geometry may contain inaccuracies due to ambiguities in 3D-to-2D projection, the optimization allows the creation, removal, and repositioning of Gaussians to refine the scene representation.

To maintain smooth gradients and stability during optimization, a sigmoid activation function constrains opacity α within the range $[0,1)$, and an exponential activation function is used for scaling the covariance matrix Σ, ensuring stable updates of the Gaussian shape. The initial covariance is estimated as an isotropic Gaussian, with axes scaled to the mean distance to the nearest three neighboring points.

The loss function driving the optimization combines an L_1 photometric loss with a D-SSIM (Structural Similarity) loss to balance pixel-wise accuracy with perceptual quality:

$$L = (1 - \lambda)L_1 + \lambda L_{D-SSIM} \tag{1.10}$$

where λ is a weighting factor controlling the trade-off between the two terms and is set to $\lambda = 0.2$. L_1 loss penalizes pixel-wise differences, while the D-SSIM enhances perceptual alignment with the ground truth.

Another optimization step is – what the authors call – adaptive control of the Gaussians. After an initial optimization warm-up phase, Gaussian densification is performed every 100 iterations, and Gaussians with opacity α below a threshold τ_α are culled as essentially transparent. Densification targets regions with high positional gradients – indicative of under or over-reconstruction – by detecting Gaussians with average positional gradient magnitudes above a threshold τ_{pos} that is set to 0.0002.

For under-reconstructed regions, small Gaussians are cloned and displaced along the direction of the positional gradient. For over-reconstructed regions, large Gaussians are split into two smaller ones, scaling their covariance by a factor $\varphi = 1.6$ and sampling positions from the original Gaussian's distribution. To prevent excessive Gaussian proliferation, every 3000 iterations, the opacity of all Gaussians is reset close to zero, allowing the culling mechanism to remove redundant or insignificant splats while letting necessary ones reestablish their opacity.

1.3.3 Extensions and Variants of 3D Gaussian Splatting

Subsequent work on 3DGS proposes improvements, extensions and pain-point mitigation. We can divide these works in seven categories, as proposed by [32]. Note however that in this chapter, we reference works that build on 3DGS on a series of issues apart from relighting which is the topic of this thesis. We conduct our own review of these cases in chapters 2 and 3. Here, we present a state-of-the-art of the 3DGS paradigm to provide the reader with a compact view on current trends and approaches.

1.3.3.1 3DGS for Sparse Input

Sparse-view capture often causes holes or collapsed geometry. Depth regularization methods mitigate this by adding external depth cues: DNGaussian[33] applies global–local depth normalization to preserve shape consistency, while MVSplat[34] constructs a classical multi-view stereo cost volume to guide Gaussian placement. Both improve completeness but depend on the quality of the auxiliary depth information. Learned-prior approaches such as PixelSplat[35] and Splatter Image[36] train a feed-forward network to propose initial Gaussian distributions from as few as two images, offering rapid initialization and generalization, though fine detail may suffer if training data lack diversity.

1.3.3.2 Memory-Efficient 3DGS

Handling millions of Gaussians in large scenes challenges GPU memory. Pruning and clustering methods remove low-opacity Gaussians or merge nearby ones into "super-Gaussians," cutting primitive counts by up to 10 times with minimal fidelity loss. Complementary attribute compression schemes like LightGaussian[37] quantize positions, covariances, and colors into codebooks, achieving ~15 times storage reduction while still rendering above 200 fps; however, aggressive quantization can introduce banding or color artifacts under complex lighting.

1.3.3.3 Photorealistic 3DGS

Basic splatting handles smooth shading but struggles with aliasing and reflections. Mip-splatting[38] applies multi-scale Gaussian kernels (or image-space MIP filtering) to avoid jagged edges at varying zoom levels. Spec-Gaussian[39] attaches per-primitive BRDF lobes, enabling rudimentary specular highlights. Analytic-splatting[40] performs closed-form integration of Gaussians over pixel footprints, producing anti-aliased edges without extra blur passes. StopThePop[41] ensures consistent visibility ordering by sorting per-tile Gaussians, eliminating popping artifacts during camera motion.

1.3.3.4 Improved Optimization Algorithms

Faster, more stable fitting accelerates pre-processing. COLMAP-Free 3DGS[42] jointly estimates camera poses and Gaussians, removing reliance on external SfM and reducing pose-error cascades. RelaxInit[43] introduces coarse-to-fine covariance control, relaxing precise initialization requirements. GsDF[44] hybridizes splatting with SDF regularization to recover sharper geometry. FRegs[45] progressively increases frequency content, guiding convergence from smooth to fine detail. GS++[46] uses per-Gaussian importance metrics to adaptively split or prune primitives, automatically tuning density for efficiency and quality.

1.3.3.5 3D Gaussians with Additional Properties

Enriching Gaussians enables semantics and interaction. Language-embedded Gaussians[47], [48] attach CLIP-style text embeddings for open-vocabulary scene querying and editing. Foundation-model GS[49] distills 2D features (e.g., DINO, SAM) into each primitive, yielding explicit 3D feature fields for segmentation or object-level edits. 4D GS[50] extends Gaussians with temporal harmonics to capture non-rigid dynamics in a single optimization, supporting real-time dynamic rendering.

1.3.3.6 Hybrid Representations

Combining Gaussians with structured models affords interpretability and control. Scaffold-GS[51] anchors primitives to a coarse mesh or skeletal rig, enabling real-time avatar retargeting and pose editing with Gaussian detail. Relightable Codec Avatars[52] embed per-Gaussian BRDF parameters for on-the-fly relighting under novel illumination. DarkGS[53] incorporates learned illumination estimates for robust splatting in low-light or high-contrast robotics scenarios.

1.3.3.7 Hybrid Representations

Moving beyond rasterization-style splatting, ray-traced Gaussian primitives enable physically grounded effects. GaussianTracer[54] treats each primitive as an ellipsoidal volume and performs exact per-ray integration for correct occlusion and reduced popping. EVER (Exact Volumetric Ellipsoid Rendering)[55] derives closed-form light transport through Gaussians, facilitating real-time soft shadows and volumetric scattering without shadow maps. Though these methods incur higher per-primitive cost, they unlock richer lighting effects for applications requiring physical accuracy.

1.4 Shading

Shading is the computational process by which a renderer determines the color and brightness of a point visible in an image. It operates after visibility has been resolved and assigns radiance values based on how surface elements interact with incoming light and how that interaction varies with view direction. The purpose of shading is to compute how light interacts with surface geometry and material at a given point. The result is the color and brightness values that convey shape, depth, and surface characteristics in the rendered image[56]. In classical pipelines, shading is performed at discrete points on surface geometry, using predefined models of reflection[57]. In more recent approaches, including differentiable and point-based methods, shading may be embedded within learned functions or distributed volumetrically.

1.4.1 Shading Pipeline Architecture

The architecture of a shading pipeline determines how and when shading calculations occur during rendering. Depending on the rendering strategy, shading computations can be coupled with geometry processing, or delayed until after geometry has been entirely processed. Two primary pipeline architectures are forward and deferred shading.

Forward shading is the conventional method used in real-time rendering pipelines, where shading and lighting computations are performed directly as geometry is rasterized. Once geometry reaches the graphics card, it is projected, broken down into vertices, and then further divided into fragments (the potential pixels), each of which receives individual lighting calculations. This method is linear and straightforward, making it efficient for simple scenes with a limited number of lights. However, forward shading quickly becomes computationally expensive as the number of dynamic lights increases. For multiple lights, each fragment on screen must calculate lighting contributions from every single one of these sources, regardless of whether the fragment is ultimately visible or hidden behind other geometry. In practice, many of these fragments never even reach the screen due to depth testing, meaning a lot of lighting calculations are wasted. Culling distant lights, combining static lights into precomputed lightmaps, pushing lighting calculations to vertex shaders which reduces the load on fragment shaders are all techniques that try to work with these restrictions. When dealing with many dynamic lights though, the limit of forward shading is still apparent. As an alternative, deferred shading separates geometry and lighting computations managing large-scale lighting scenarios way more efficiently.

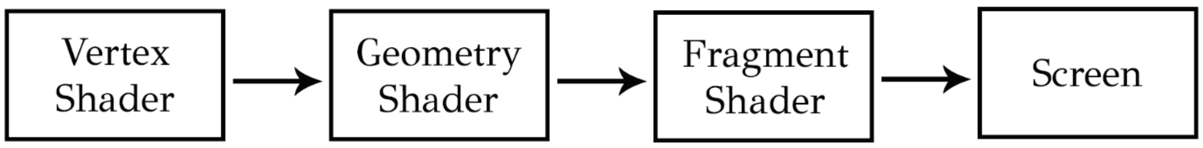

Figure 1.5 A typical forward shading pipeline

Deferred shading was first introduced in a hardware design in 1988[58], with a more general purpose method using full screen Geometry Buffers (G-Buffers) following in 1990[59]. Deferred shading decouples geometry and light processing, making it relatively simple to manage large numbers of light sources[60]. Rather than shading fragments when each object passes through the pipeline, it first performs a geometry pass storing relative attributes into intermediate buffers known as Geometry Buffers (G-buffers). These can include color, normal, depth, position, material (for more than one we can use a derivative method called deferred lighting[61]) and are stored per pixel. After that, it performs a shading pass where lighting calculations are performed only on visible pixels. This way, redundant computations are avoided making deferred shading preferable for scenes with dynamic or complex lights. It comes however with its own drawbacks. For one, in order to handle transparency, we need to combine deferred with forward rendering for these transparent objects. Another solution could be depth peeling in order to achieve order-independent transparency but at the cost of G-buffer size. As mentioned, multiple materials need a different technique called deferred lighting but at the cost of an extra pass. Finally, hardware anti-aliasing cannot produce correct results. To overcome this, a traditional way is edge detection with more alternative approaches being MLAA[62] and FXAA[63].

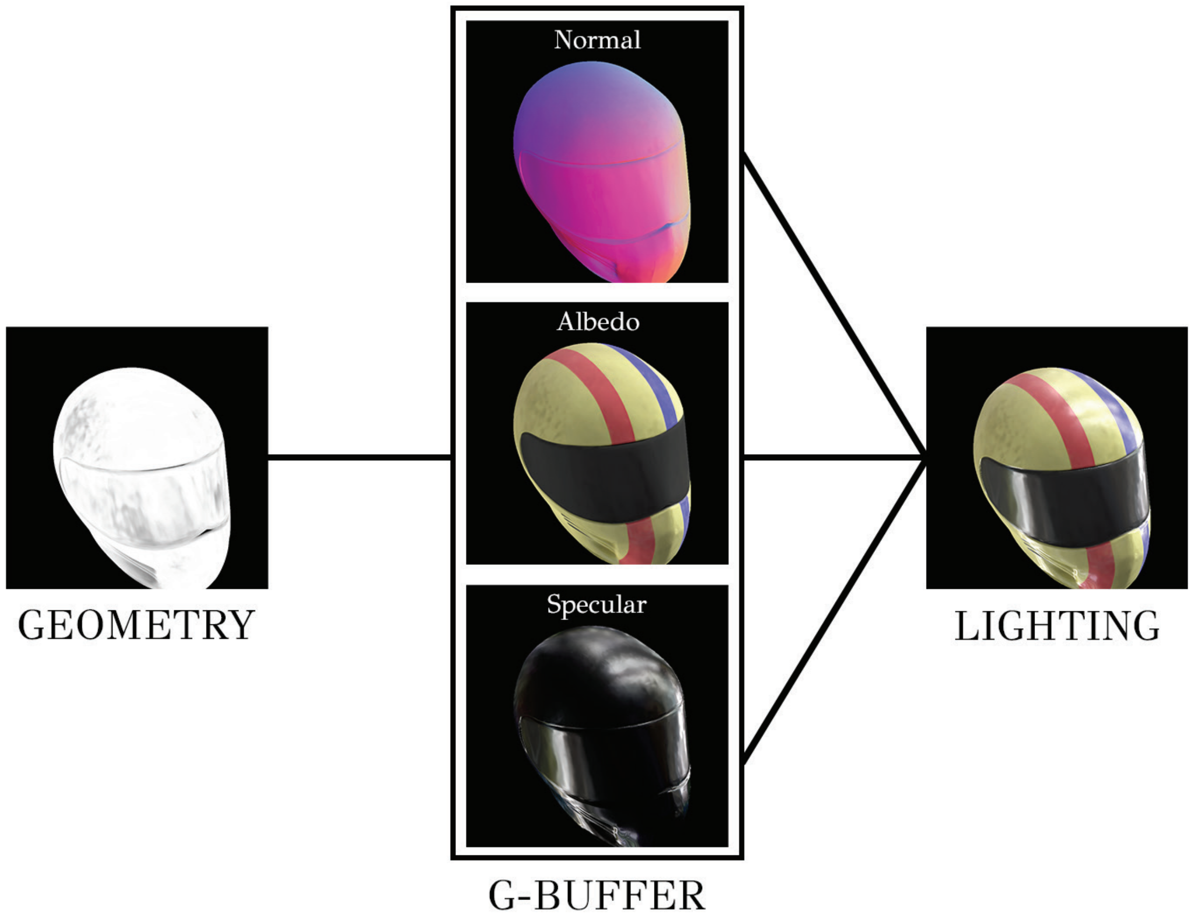

Figure 1.6 A common deferred shading pipeline.

1.4.2 Shading in Differentiable Rendering

Differentiable rendering (DR) is a technique that allows a 3D scene to be adjusted by comparing a rendered image to a reference image. It does this by computing how changes in scene parameters like geometry, material, lighting, or camera affect the final image. Rendering can be treated as a function that maps these 3D scene parameters to the 2D image pixels' intensities. In DR, this function is differentiable, meaning it can calculate the rate of change of the image with respect to changes in the scene parameters. This makes it possible to optimize scenes using gradient descent; same way neural networks are trained. DR methods can be grouped into three broad categories based on how the scene is represented and rendered: physics-based, NeRF-based, and point-based[64].

Physics-based DR follows the light-transport equation and handles reflections, shadows, and global illumination aiming for physical accuracy. Its core challenge is the visibility discontinuity that happens when a tiny change in geometry or camera makes a surface suddenly appear or disappear. Then, the rendered pixel values jump, so ordinary derivatives break down. There are two proposed solutions: boundary sampling, which explicitly integrates contributions near silhouette edges, and re-parameterization, which remaps the integral so those edges stay fixed. NeRF-based DR takes a different route. It learns a continuous volume, via a neural network, that relates space-and-view direction to color and density, then renders it with smooth volume integration. This side steps discontinuities at the cost of ignoring complex light bounces but offers fast, stable gradients. Point-based DR, exemplified by 3D Gaussian Splatting, represents the scene as many Gaussian spheres. Each sphere projects to the image and blends smoothly with its neighbors, giving real-time speed and simple gradients, though at the price of higher memory use and limited geometric editing. Together, these three approaches define the current landscape: physics-based aims for realism, NeRF-based balances detail and learning ease, and point-based targets speed.

1.5 Light and Material Models

1.5.1 Illumination Models and BRDF

1.5.1.1 Bidirectional Reflectance Distribution Function (BRDF)

The Bidirectional Reflectance Distribution Function (BRDF) defines how light is reflected off an opaque surface. It relates the amount of light arriving from an incident direction (incoming) to the amount reflected in an outgoing direction at a point on a surface, taking into account the surface's material properties. Formally, the BRDF $f_r(P, \omega_i, \omega_o)$ specifies the ratio of reflected radiance in direction ω_o to the incident irradiance from direction ω_i with regard to the surface normal at point P. BRDF uses sr^{-1} as it's units with sr being the steradians, a unit of solid angle. In rendering, BRDF is integrated into the rendering equation[65] like:

$$L_o(P, \omega_o) = L_e(P, \omega_o) + \int_{\Omega} f_r(P, \omega_i, \omega_o) L_i(P, \omega_i)(\omega_i, n) d\omega_i \qquad (1.11)$$

where L_o is the outgoing radiance at point P, L_e represents any emitted light, L_i is the incoming radiance from direction ω_i and n is the surface normal. The integral runs over the hemisphere Ω above the surface.

The BRDF operates independent of the overall scene geometry or light path allowing for materials to be modeled separately from light transport methods. Moreover, the BRDF does not distinguish between direct and indirect illumination. It applies equally to light coming directly from sources and light reflected from other surfaces. Finally, it does not account for subsurface scattering or transmission of light through the surface. These are handled by other functions, namely Bidirectional Transmittance Distribution Function (BTDF) or subsurface scattering models.

1.5.1.2 Common BRDF Models

BRDFs can be split into three broad categories. Empirical, physically based and measured models. Empirical models like Lambertian[66] and Phong[67] are designed to approximate observed light behavior. They do not necessarily adhere to physical laws, but they are computationally efficient since they are based on simplified mathematical functions.

Physically based models like Cook-Torrance[68] integrate physical principles to better capture the real-world behavior of materials. These models are constrained by three properties: positivity, reciprocity and energy conservation.

Positivity requires that the BRDF returns non-negative values for all inputs. Formally:

$$f_r(\omega_i, \omega_o) \geq 0 \tag{1.12}$$

Reciprocity (or commonly referred to as Helmholtz reciprocity) means that the reflection behaves identically if the incoming and outgoing directions are swapped. Reversible light paths are a requirement for physically-based or inverse rendering:

$$f_r(\omega_i, \omega_o) = f_r(\omega_o, \omega_i) \tag{1.13}$$

Energy conservation constraint states that the BRDF must not reflect more energy than it receives, formally, the total energy from any incoming direction must not exceed the incoming energy:

$$\forall \omega_i, \qquad \int_{\Omega} f_r(\omega_i, \omega_o) \cos(\theta_o)\, d\omega_o \leq 1 \tag{1.14}$$

where θ_o is the angle between the surface normal n and the outgoing direction. The integral is taken over the hemisphere Ω.

Finally, measured models rely on empirical data collected from physical surfaces to directly describe reflectance behavior of certain materials.

Following, we present some of the principal BRDF models.

Lambertian is one of the simplest BRDF functions. It is used for diffuse term which assumes that incident light is scattered in all possible directions equally (within the hemisphere around the surface normal). It is a good approximation for behavior of many real-world materials and is very fast to evaluate.

$$f_r(\omega_i, \omega_o) = \frac{\rho}{\pi} \tag{1.15}$$

where ρ represents the diffuse reflectance or albedo. The division by π ensures that total reflectance remains below or equal to one[69], satisfying energy conservation. Being an empirical model, Lambertian BRDF lacks view-dependence and surface roughness effects limiting its physical realism.

The Phong model extends Lambertian with a specular highlight component:

$$f_r(\omega_i, \omega_o) = k_s(R \cdot V)^n \tag{1.16}$$

where k_s is the specular reflectance, R is the reflection of the incoming light, V is the view direction and n controls the sharpness of the specular highlights. In and of itself, Phong in neither energy conserving nor reciprocal. However, many attempts have been made to rectify this[70], [71], [72].

The Cook-Torrance model is a physically based BRDF that models surfaces as collections of tiny surfaces or, microfacets[73] with varying orientations. It satisfies positivity, reciprocity, and energy conservation. It is defined as[74]:

$$f_r(\omega_i, \omega_o) = \frac{D(H)F(\omega_i, H)G(\omega_i, \omega_o, H)}{4(n \cdot \omega_i)(n \cdot \omega_o)} \tag{1.17}$$

where H is the half-vector, defined as the normalized sum of the incoming and outgoing directions. It serves as the local normal of a microfacet responsible for reflection at a given point. $D(H)$ is the microfacet distribution function that is often implemented using the GGX distribution and describes how much and in what way microfacets vary. It is typically controlled by the roughness parameter:

$$D_{GGX}(H) = \frac{\alpha^2}{\pi[(n \cdot H)^2(\alpha^2 - 1) + 1]^2} \quad (1.18)$$

$F(\omega_i, H)$ is the Fresnel term, approximated by Schlick's formula[75]:

$$F(\omega_i, H) = F_0 + (1 - F_0)(1 - (\omega_i \cdot H)^5) \quad (1.19)$$

where F_0 is the reflectance at normal incidence. Fresnel term evaluates how much light is reflected off the surface under given angle of incidence. $G(\omega_i, \omega_o, H)$ is the geometry term that is responsible for masking and shadowing effects between microfacets. The denominator[76] comes from derivation of the microfacet model using perfect mirrors as microfacets.

Disney's[77] empirical model modifies the basic Lambertian reflection by adding a grazing-angle adjustment. This helps capture the increase in reflectance observed when light hits a surface at a shallow angle. It's given by the equation:

$$f_r(\omega_i, \omega_o) = \frac{\rho}{\pi}[1 + (F_{D90} - 1)(1 - cos\theta_i)^5][1 + (F_{D90} - 1)(1 - cos\theta_o)^5] \quad (1.20)$$

where ρ is the diffuse reflectance (albedo) of the surface, θ_i, θ_o are the angles between the surface normal n and the incoming and outgoing directions respectively and F_{D90} is the parameter that controls the increase in reflectance at shallow angles. It is given by the equation:

$$F_{D90} = 0.5 + 2 \cdot roughness \cdot cos\theta \quad (1.21)$$

1.5.2 Light Interactions

Accurately simulating light–surface interaction is a central challenge in computer graphics. Complete physical modeling of light paths is too computationally intensive for real-time rendering. To work around this, lighting can be based on approximations of reality using simplified models that are much easier to process and look relatively similar. A common approach, most notably used by the Phong illumination model[78] and its derivatives, is the separation of light interaction into three components: ambient, diffuse and specular light.

1.5.2.1 Ambient, Diffuse and Specular Light

In any scene, light rarely comes from a single, direct source. Instead, surfaces receive some level of illumination from light scattered by other surfaces or diffused through the environment. A simplification of this behavior can be modeled via the ambient light, ensuring that even surfaces not directly illuminated by a light source maintain some form of brightness. Ambient light is uniformly applied to all surfaces in a scene and does not depend on their orientation or position relative to light sources. A typical way of calculating it is:

$$L_{ambient} = k_a I_a \tag{1.22}$$

where k_a is the ambient reflectivity of the material and I_a is the ambient light intensity.

Diffuse lighting approximates the scattering of light after hitting rough surfaces. It is the most visually significant component of the lighting model. As opposed to ambient light, diffuse depends on the angle between the incoming light and the surface normal. Based on Lambert's cosine law[79] which states that brightness is proportional to the cosine of the angle between the light direction and the surface normal, we can deduct that as the angle increases, the surface will get darker. A simplified calculation of diffuse light can be described by the following equation:

$$L_{diffuse} = k_d I_d \max(n \cdot l, 0) \tag{1.23}$$

where k_d is the diffuse reflectivity of the material, I_d is the light intensity, n is the normalized surface normal and l is the normalized vector pointing to the light source. The dot product $n \cdot l$ returns the cosine of the angle between the surface and the light. Clamping it at or above zero avoids incorrect shading on surfaces facing away from the light source.

Specular light models the highlights on reflective objects, appearing as bright spots on the points where light directly hits their surface. In contrast with diffuse light that is scattered in all directions, specular is more focused and depends on the angle between the viewer and the reflected light. It can be described by the equation:

$$L_{specular} = k_s I_s \max(r \cdot v, 0)^{\alpha} \tag{1.24}$$

where k_s is the specular reflectivity, I_s is the light intensity, r is the reflection vector of the light direction reflected across the surface normal, v is the normalized vector pointing toward the viewer, and α is the shininess factor that controls the sharpness of the highlight. Higher values for α result in a smaller (more concentrated) highlight. The dot product $r \cdot v$ calculates how closely the reflected light aligns with the viewer's direction.

By combining ambient, diffuse and specular lights we get a complete representation of how light interacts with an object's surface.

1.5.3 Intrinsic Image Decomposition

Image decomposition refers to the mathematical process of transforming an image into a new set of images, with each new image representing a different aspect (or component) of the pictured scene[80]. Strictly speaking, image decomposition is concerned with transformations on the signal level with broad applications like denoising, scaling etc. There are, however, different kinds of decompositions relative to the nature of the components. In the context of lighting in computer graphics, decompositions like intrinsic[81] and structural[82] present a more useful tool. Intrinsic image decomposition is focused on the content of the scene. Specifically, intrinsic scene characteristics like reflectance (diffuse, albedo) and shading. Structural decomposition, on the other hand, is focused on geometry.

Intrinsic image decomposition factors each pixel value $I(x, y)$ of an observed image into reflectance $R(x, y)$ and shading $S(x, y)$ terms so that $I = R \cdot S$. Here, R represents the true color or albedo of surfaces (what we would see under neutral, even lighting) and S captures how light direction, surface orientation, and shadows darken or brighten those colors. This relationship however is fundamentally ill-posed since infinitely many (R, S) pairs satisfy it making reliable algorithm design challenging[83]. Early methods use the Retinex theory[84] stating that sudden changes in intensity usually signal a change in surface color. Modern approaches learn statistics of real materials and lighting from synthetic or labeled datasets. Once R and S are estimated we are able to relight a scene by altering S, swap the material of an object by editing R, or perform color-consistent edits without altering the lighting.

Structural decomposition focuses on recovering the shape-related aspects of an image rather than its color. From a single photograph we can compute auxiliary "structure" images such as surface normals (the direction each point faces), depth maps (distance from the camera), curvature or height fields. Traditional shape-from-shading methods infer normals by assuming a simple light source and smooth surfaces; depth-from-focus measures sharpness across different focus settings to estimate distance. More recently, convolutional neural networks trained on paired color-and-depth data can predict normals and depth directly from one RGB image. These structural maps feed into 3D reconstruction pipelines building meshes or point clouds and enable novel-view synthesis, collision detection in simulations, and physically based rendering where an accurate geometry model is essential.

By treating intrinsic and structural decomposition as complementary tasks or solving them together in one network, modern systems[85], [86] achieve more consistent outputs. For example, knowing the shading can help disambiguate surface orientation, and accurate normals can guide better separation of reflectance and shading, resulting in both more realistic relighting and more precise geometry recovery.

1.6 Global Illumination

Global illumination is a set of techniques that aim to represent how light interacts with surfaces in a scene, capturing not only the direct illumination from light sources but also the indirect illumination that accounts for rays reflected by other surfaces in the scene[87]. While shadows and reflections can be thought of as a part of global illumination, typically, we are concerned with what is called *diffuse interreflection* or, the way light is reflected from objects that are not shiny or specular. If the diffuse surface is colored, the reflected light is also colored resulting in similar coloration of the surrounding objects.

These techniques are numerical approximations of the rendering equation[65] which describes how outgoing light at any point depends on both emission and the incoming light reflected by the surface. The approximations are needed since directly evaluating this equation is prohibitively expensive for interactive applications because it requires tracing a very large number of light paths and resolving complex visibility relationships. Many families of techniques have been proposed throughout the years: finite-element radiosity[88], photon mapping[89], multipass hybrids that combine a radiosity stage with ray tracing[90], precomputed radiance transfer for static scenes under distant lighting[91] and sparse-sampling schemes ranging from image-space render caches[92] to line-space radiance caches[93]. However, given our context of relighting with 3DGS, we will focus our attention on three main techniques that approximate the rendering equation and are commonly found in 3DGS relighting implementations. These are Monte Carlo ray tracing[94], Image-based lighting[95] and Spherical Harmonics[96].

1.6.1 Monte Carlo Ray Tracing

Monte Carlo ray tracing (often called path tracing) evolved directly from Whitted's recursive ray caster[97], which added perfect specular reflections and refractions to visibility rays, and was generalized to stochastic ray scattering for effects like glossy reflections and depth of field[98] before being applied to the full rendering equation.

Monte Carlo ray tracing begins by treating the rendering equation not as an impossible high-dimensional integral to solve analytically but as a problem of statistical estimation, in which the true illumination at each pixel is approximated by averaging the results of many randomized light paths. To render a single pixel, the algorithm casts a primary ray from the camera through the pixel into the scene, where it intersects a surface. At that intersection point, the path tracing engine evaluates the surface's bidirectional reflectance distribution function (BRDF) to determine how light is reflected, then stochastically selects a new outgoing direction over the hemisphere above the surface according to a probability density function. This choice is made so that the expectation of each sample exactly matches the integral of the rendering equation, guaranteeing that the estimator is unbiased that is, the expected value of the Monte Carlo estimate equals the true value of the lighting integral, with no systematic error remaining even as the number of samples grows[94]. As the ray continues, it may encounter additional surfaces where the process repeats: at each bounce the path accumulates the product of the BRDF, the cosine of the incident angle, and any emitted radiance, before terminating when it reaches a light source or when a predetermined maximum depth or probabilistic cutoff (Russian roulette[99]) criterion is met. Once N such paths have been traced for the pixel, their accumulated radiance values are averaged to produce the final pixel color.

Monte Carlo path tracing faces a fundamental challenge: each path sample is a random variable whose variance contributes to visible image noise. The variance of the Monte Carlo estimator over N samples scales inversely with N, so that the standard deviation and thus the perceptual noise level decreases only as $1/\sqrt{N}$. In practical terms, reducing noise by half mandates four times more samples per.

To address this, several variance-reduction strategies are layered on top of the basic algorithm. Importance sampling[100] tailors the sampling density $p(\omega_i)$ to match the shape of the integrand, often choosing directions proportional to the BRDF's main lobe or the distribution of light source contributions, so that high-contribution paths are sampled more frequently, yielding lower variance for the same sample count. Stratified sampling[101] subdivides the integration domain, such as the image plane or the hemisphere of directions, into regions (strata) and ensures at least one sample per region, smoothing out clustering artifacts and further reducing variance. Multiple importance sampling then combines complementary sampling strategies, such as BRDF-based and emitter-based proposals, into a single estimator using carefully chosen weights that retain unbiasedness while minimizing variance across disparate integrand components[87].

Efficient implementation of Monte Carlo path tracing requires handling millions or even billions of rays per scene. Acceleration structures, including bounding-volume hierarchies (BVHs)[102] and kd-trees[103], cut the cost of ray–geometry intersection tests from linear time in the number of primitives to logarithmic time per ray, enabling heavy ray-marching workloads without prohibitive performance penalties.

1.6.2 Image-based Lighting (HDRs)

Image-based lighting (IBL) captures the real illumination in a scene and reuses it to light virtual objects so they blend naturally without rebuilding full 3D geometry. Rather than tracing rays for every bounce, IBL records how light arrives from every direction in a single map, then looks up those values at render time to drive both diffuse and specular shading[104].

A key form of IBL is the high-dynamic-range (HDR) environment map[95]. To create one, we photograph a mirrored sphere at several exposure settings and merge those shots into a floating-point panorama that preserves both very bright highlights and deep shadows. This panorama is typically stored in an "angle map" projection, where each texel corresponds to a specific direction on the sphere. At render time, a surface's normal or reflection vector is converted into those sphere coordinates, and the corresponding texel's RGB value gives the incident radiance without any ray tracing.

For faster diffuse lighting, the HDR map can be projected once into a low-order spherical-harmonic basis. By sampling the angle map at uniformly distributed directions and accumulating against the SH basis functions, we compute a small set of coefficients that approximate the full lighting function. By that, shading any point only needs a single dot product between the probe's SH coefficients and that point's precomputed transfer terms.

Because there is no fixed "white" level in an HDR image, renderers must include exposure controls to scale the raw radiance values appropriately before lookup or projection. While HDR-based IBL delivers realistic lighting with minimal per-frame cost, it requires a mirror probe, careful camera calibration, a static scene (no moving objects), and a multi-step capture workflow. These factors can limit its use in dynamic or mobile settings.

1.6.3 Spherical Harmonics

Spherical harmonics can be thought of as the spherical equivalent of the sine and cosine waves used in Fourier series: they form a set of smooth "patterns" on the surface of a sphere that, when added together in the right proportions, can recreate any smoothly varying function of direction—like the way light arrives at a point from all around[96]. In plain terms, instead of storing a full environment map (which can be large and detailed), we store just a handful of numbers (the SH coefficients) that capture the overall "shape" of the lighting, smoothing out sharp features but preserving gradual changes. This makes it much faster to rotate, interpolate, and combine lighting for applications like real-time relighting, where we need good-looking results without the cost of high-resolution data.

CHAPTER 2

CASE STUDIES

2.1 GaussianShader: 3D Gaussian Splatting with Shading Functions for Reflective Surfaces

GaussianShader[105] adds a learnable shading model to the 3D Gaussian Splatting method by giving each Gaussian extra attributes like base color, view-dependent color variation, specular tint, roughness, and an estimated normal. Normals are derived from each splat's shortest axis and are then aligned with depth-derived normals. Both direct and indirect reflections are accounted for by querying a mip-mapped environment map. GaussianShader is better suited for reflective surfaces where it achieves great results on both Novel View Synthesis and relighting. On the other hand, it seems to struggle on mat surfaces or real scenes.

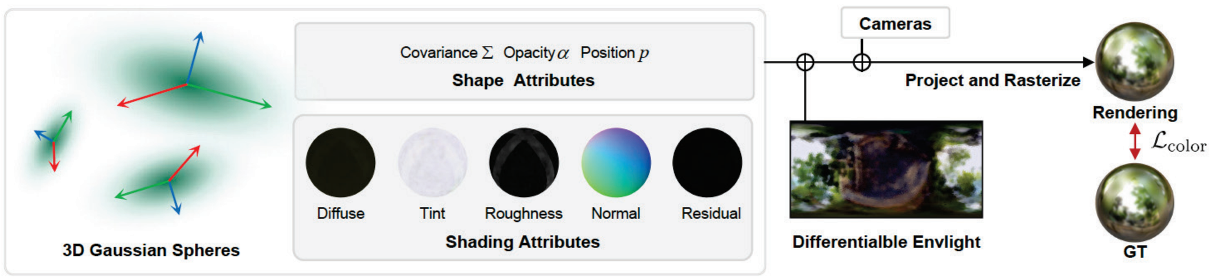

Figure 2.1 GaussianShader Pipeline. Figure from [105].

2.1.1 Illumination Model

2.1.1.1 Shading Function

The shading function used is a simplified version of the rendering function[106]. For each Gaussian, the rendered color c, for a viewing angle ω_o is given by the equation:

$$c(\omega_o) = \gamma(c_d + s \odot L_s(\omega_o, n, \rho) + c_r(\omega_o)) \tag{2.1}$$

where $c_d \in [0,1]^3$ is the diffuse color, $s \in [0,1]^3$ is the specular color, $\rho \in [0,1]$ is the roughness factor which controls the dispersion of the reflection and n is the normal vector. Residual color $c_r: R3 \rightarrow R3$ refers to the additional effects (scattering and reflections) of indirect lighting. c_r is comprised of 3^{rd} order Spherical Harmonics (SH) coefficients and depends on viewing angle.

The term $s \odot L_s(\omega_o, n, \rho)$ determines the color of direct light reflections. Specular color s is multiplied element-wise with direct specular light L_s. Along with $c_r(\omega_o)$, this term accounts for all reflection colors. γ is a gamma correction function.

2.1.1.2 Specular Light

To calculate the specular light L_s, they integrate the incoming radiance $L(\omega_i)$ with the GGX Normal Distribution Function[107]. This procedure is visualized in *Figure 2.2* and is described by the following formula:

$$L_s(\omega_o, n, \rho) = \int_{\Omega} L(\omega_i) D(r, \rho)(\omega_i \cdot n) d\omega_i \tag{2.1}$$

where Ω is the hemisphere above each Gaussian. The integral sums the contribution of radiation from all directions ω_i within this hemisphere. $L(\omega_i)$ is the environment light from direction ω_i. D is the normal elliptical distribution (GGX NDF). It's the circular sector that defines the reflected radiation. It models how reflections are spread based on two parameters. r, which is the direction of the outgoing reflection and ρ which is the surface roughness. $(\omega_i \cdot n)$ is Lambert's cosine law, weighting incoming light based on the angle of incidence.

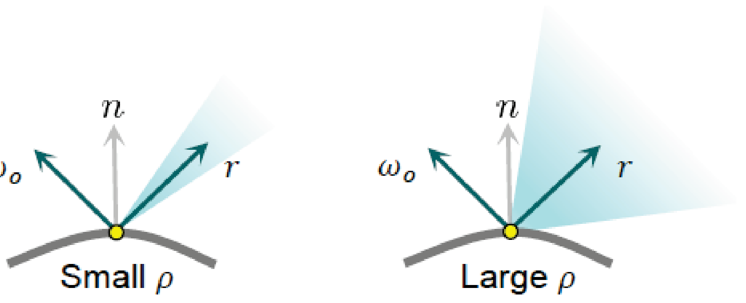

Figure 2.2 Normal Distribution Function D in Eq. 2.2 is determined by roughness ρ and reflective direction r. The surface's specular lobe grows with the value of ρ. Figure from [105].

2.1.1.3 Environment Light

Environment light refers to the incoming radiance from all directions of the scene. To represent it, they use a $6 \times 64 \times 64$ cube map. That is, 6 two-dimensional textures of 64×64 pixel resolution. The cube map stores precomputed radiance from different directions. To further speed up the calculations they use pre-filtered versions of this cube map, the so-called mipmaps. Mipmaps are a hierarchy of versions of a texture with progressively lower resolution.

In case of low roughness ($\rho \approx 0$), there is a sharp reflection. So, sample from the highest resolution mip level (detailed environment map). In case of high roughness ($\rho \approx 1$), there is a diffused reflection. So, sample from lower resolution mip level (blurred version of the environment).

The general pipeline to retrieve the appropriate part of the environment light is:

- The environment light is stored as a latitude-longitude HDRI texture file. This file is converted to a cube map. The mipmaps are generated based on that cube map.
- Compute the reflection direction $r = 2(\omega_o \cdot n)n - \omega_o$. r is used to determine which texel (color value) to sample from the cube map.
- Depending on roughness value ρ, select the appropriate mipmap level i.e. how blurred that color should be. If ρ lies between two mipmap levels, they take an average of the two levels (they refer to this process as interpolation).

2.1.2 Normal Estimation

The authors' observation is that during optimization, Gaussians tend to approach a planar shape (*Figure 2.2*). That being so, they chose the shortest axis as the normal vector of this "flattened" Gaussian as a normal estimate (denoted by v).

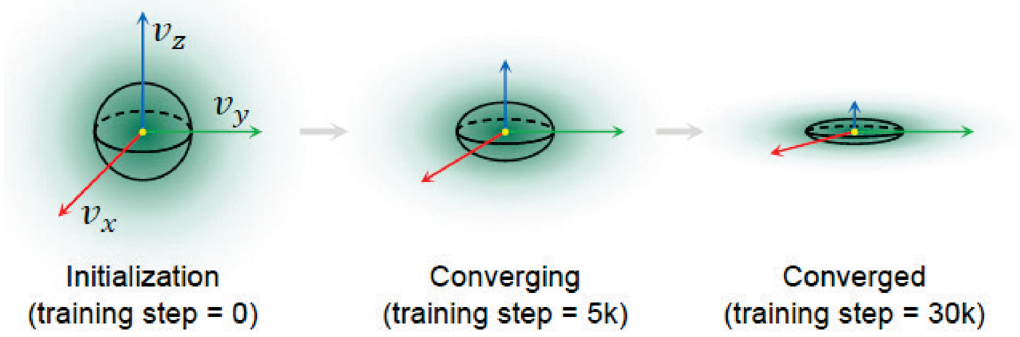

Figure 2.3 Gaussians tend to become planar (flat) as they converge.
Figure from [105].

This vector however presents ambiguities. Specifically, the direction of the shortest axis might point outward or inward from the surface. To tackle this, the authors introduce two trainable normal "residuals" (Δn_1, Δn_2) which are optimized during training.

Initially, they choose the normal direction that coincides with the viewing direction ω_o. Then, they apply the corresponding normal residual based on the following cases:

$$\begin{aligned} n &= v + \Delta n_1, && \omega_o \cdot v > 0 \\ n &= -(v + \Delta n_2), && otherwise \end{aligned} \tag{2.3}$$

Moreover, they add a penalty towards normal residual making sure it doesn't deviate much from the shortest axis. This penalty is described by $L_{reg} = ||\Delta n||^2$.

The above computation is performed per single Gaussian. To guarantee consistency among the local geometry (neighboring Gaussians), the authors introduce a new loss $L_{normal} = ||\bar{n} - \hat{n}||^2$. $\bar{n}$ is the rendered normal map containing information about the separately defined normal of each Gaussian. $\hat{n}$ is computed by applying a Sobel-like operator on the rendered depth map and contains information on the local geometry formulated by multiple Gaussians. The consistency between the local geometry and the estimated normals is enforced by minimizing the difference between $\bar{n}$ and $\hat{n}$.

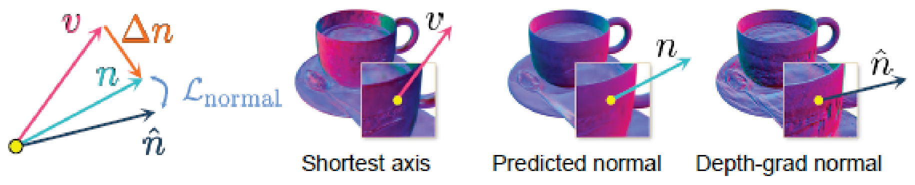

Figure 2.4 Relationship between shortest axis v, predicted normal n and depth-derived normal $\hat{n}$. $\mathcal{L}_{normal}$ is the consistency loss. Figure from [105].

2.1.3 Loss Functions

The total training loss L is:

$$L = L_{color} + \lambda_n L_{normal} + \lambda_s L_{sparce} + \lambda_r L_{reg} \tag{2.4}$$

where $\lambda_n = 0.01$, $\lambda_s = 0.001$, $\lambda_r = 0.001$.

2.1.3.1 Color Loss

$$L_{color} = ||C - C_{gt}||^2 \tag{2.5}$$

Color loss minimizes the difference between the blended color C of all Gaussians and the corresponding pixel in the ground-truth image.

$$C = \sum_{i \in N} c_i a_i \prod_{j=1}^{i-1} (1 - a_j) \tag{2.6}$$

where c_i is the color of the i-th Gaussian and a_i is the opacity of the i-th Gaussian. $\prod_{j=1}^{i-1}(1 - a_j)$ represents the transmittance, i.e., how much of the background is still visible after blending previous Gaussians. This is a front-to-back compositing method that ensures that closer Gaussians contribute more, while farther Gaussians are attenuated (weakened in visibility/made more transparent) based on the opacity of the ones in the front.

2.1.3.2 Normal Residual Loss

$$L_{reg} = ||\Delta n||^2 \tag{2.7}$$

Normal residual loss prevents normal residuals from deviating too much from the Gaussian's shortest axis.

2.1.3.3 Normal loss

$$L_{normal} = \|\bar{n} - \hat{n}\|^2 \tag{2.8}$$

Normal loss enforces consistency between the local geometry and the estimated normals.

2.1.3.4 Sparsity Loss

$$L_{sparce} = \frac{1}{|\alpha|}\sum_{a_i} |log\ (\alpha_i) + \log(1 - a_i)| \tag{2.9}$$

Sparsity loss "pushes" opacity towards 0 (fully transparent) or 1 (fully opaque). This way, Gaussians that do not contribute significantly to the rendering are not affecting blending.

2.1.4 Parameters

We split the parameters of GaussianShader into three categories: trainable, non-trainable and user-defined.

The trainable parameters are those optimized during the training process. They include Gaussian shape attributes, shading attributes, normals and environment light representations. Gaussian shape attributes consist of position p, covariance matrix Σ – divided into a scaling matrix S and a rotation matrix R – and opacity α. Shading attributes are diffuse colorc_d, specular color s, roughness factor ρ, and residual color c_r. Normal estimation is refined using two normal residuals Δn_1, Δn_2. The environment light is represented by a cube map, along with its pre-filtered mipmap levels.

The non-trainable parameters remain fixed throughout training and evaluation. These include background color, which is either black or white, and camera viewpoints that are used both during training and testing. Camera viewpoints share common attributes such as extrinsics (position and rotation), intrinsics (focal length and field of view) and ground truth images.

The user-defined parameters configure the model and the optimization process. Model parameters include the Spherical Harmonics degree, BRDF mode (which determines whether the environment map is used), environment map resolution and the checkpoint saving. Optimization parameters include the number of training steps, loss weights (λ_n, λ_s, λ_r), densification settings (interval, opacity reset) and regularization factors (normal and lambda).

2.1.5 Architecture and Information Flow

2.1.5.1 Training

During training, the input consists of dataset images and the COLMAP sparse model with information on cameras, images and points. When it comes to shading, an additional input is required, that of a lookup table in the form of a BIN file. The lookup table represents an integration of the specular BSDF of the GGX model, which is the same specular BSDF that is used in the NVDIFFREC[108] shading model. The scene's initial environment map is produced via a random initialization. This means that no environment map is required as input.

The output of the training phase includes the optimized 3D point cloud, the camera parameters (intrinsics and extrinsics) used for training, the training configuration arguments and the initial point cloud. These outputs are in accordance with the seminal implementation of Gaussian Splats. Where GaussianShader differs is the inclusion of brdf_mlp. As mentioned, the initial environment map is produced via a random initialization. The reflectance properties of the scene are trained and refined over training iterations and are then aggregated into the final environment map. That is, an HDR file created from the cube map representing the environment light and stored in a latitude-longitude format.

2.1.5.2 Rendering

Rendering with GaussianShader assumes the existence of the brdf_mlp HDR file and the optimized 3D point cloud produced by the training process. The output contains the rendered RGB images of the scene from all camera views. Also included, for each view, are the alpha, depth, diffuse, normal, roughness, and specular maps. Finally, the 360° lighting for each Gaussian is computed and rendered. These are then combined in both an HDR and a PNG file.

2.1.5.3 Relighting

To perform the relighting of a scene one has to render with a different environment map. This process can be inferred i.e., no retraining is required. We can train the model once in order to acquire the optimized 3D point cloud and HDR environment map. Then, we can simply change the brdf_mlp.hdr file with a different environment map and re-render. Note, the GaussianShader implementation requires an HDR file with that exact name. When relighting, new RGB images are produced along with new specular color maps. All other maps (alpha, depth etc.) remain the same regardless of which environment map is used.

2.2 Relightable 3D Gaussians: Realistic Point Cloud Relighting with BRDF Decomposition and Ray Tracing

Relightable 3D Gaussians[109] begins by extending each Gaussian with a surface normal and simplified Disney BRDF parameters for albedo and roughness. It then constructs a Bounding-Volume Hierarchy over all Gaussians and performs point-based ray tracing to determine which Gaussians are visible from each shading sample, producing per-point visibility weights that capture shadows and occlusion. Lighting is decomposed into a fixed environment map providing direct incident radiance and a learned indirect visibility field that models interreflections. During shading, the physically based rendering equation is calculated for each 3D Gaussian to determine its outgoing radiance, denoted as c'. Each point's color is obtained by combining diffuse and specular terms from the Disney BRDF using the normal and roughness to filter the environment map. The indirect contribution, and all points are composited via alpha blending to obtain vanilla color map C, PBR color map C', depth map D, normal map N, etc. Relightable 3D Gaussians, are optimized using the ground truth image C_{gt} and the pseudo normal map derived from D for supervision.

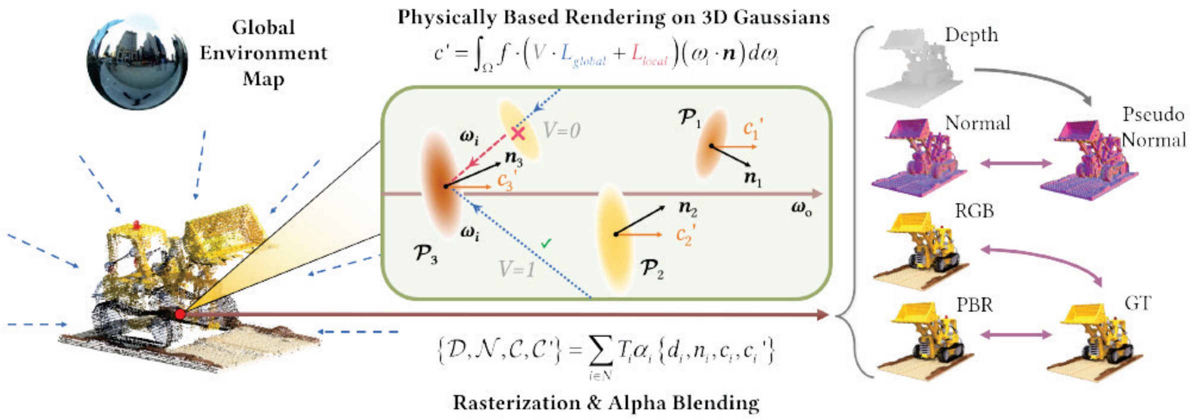
Global Environment Map
Physically Based Rendering on 3D Gaussians
$c' = \int_{\Omega} f \cdot (V \cdot L_{global} + L_{local})(\omega_i \cdot n) d\omega_i$
$V=0$
$V=1$
ω_i
n_3
c_3'
$\mathcal{P}_1$
c_1'
n_1
n_2
c_2'
ω_o
$\mathcal{P}_3$
$\mathcal{P}_2$
$\{\mathcal{D}, \mathcal{N}, \mathcal{C}, \mathcal{C}'\} = \sum_{i \in N} T_i \alpha_i \{d_i, n_i, c_i, c_i'\}$
Rasterization & Alpha Blending
Depth
Normal
Pseudo Normal
RGB
PBR
GT

Figure 2.5 Relightable3DG Differential Rendering Pipeline. Figure from [109].

2.2.1 Illumination Model

2.2.1.1 Incident Light

Incident light is split between local and global components. The sampled incident light at a Gaussian from viewing direction ω_i is represented as:

$$L_i(\omega_i) = V(\omega_i) \cdot L_{global}(\omega_i) + L_{local}(\omega_i) \tag{2.10}$$

where L_{global} or L_{direct} is the globally shared direct component parameterized as a 16×32 environment map and L_{local} or $L_{indirect}$ is the per Gaussian indirect light term that is parameterized as 3rd degree SH. $V(\omega_i) \in [0,1]$ is the visibility term. $V = 1$ if the Gaussian is fully visible to the global light source, $V = 0$ if fully shadowed or $0 < V < 1$ if partially occluded. V is computed via the ray tracing method that is further discussed in a later point.

2.2.1.2 Rendering Equation

$$L_o(\omega_o, x) = \int_{\Omega} f(\omega_o, \omega_i, x) L_i(\omega_i, x)(\omega_i \cdot n) d\omega_i \tag{2.11}$$

where x and n are the surface point and its normal vector, f is the BRDF (material), and L_i and L_o denote the incoming (incident) and outgoing radiance in directions ω_i and ω_o. Ω signifies the hemispherical domain above the surface.

BRDF properties – including base (albedo) $b \in [0,1]^3$ and roughness $r \in [0,1]$ – are assigned to each Gaussian. The model adopted is a simplified version of Disney BRDF[77] divided into a diffuse and a specular term given by the following equations:

$$f_d = \frac{b}{\pi}, \qquad f_s(\omega_o, \omega_i) = \frac{D(h; r) \cdot F(\omega_o, h) \cdot G(\omega_i, \omega_o, h; r)}{(n \cdot \omega_i) \cdot (n \cdot \omega_o)} \tag{2.12}$$

where $h = \frac{(\omega_i + \omega_o)}{2}$ is the half vector (i.e. the normalized vector that bisects the angle between the incoming light and the viewing direction), D, F, and G are the normal distribution function, Frensel and Geometry terms respectively.

2.2.1.3 Physical Based Rendering (PBR)

The general approach is to compute the PBR color $\{c_i'\}_{i=0}^{N}$ for each Gaussian and then obtain the color map C' via alpha compositing. To evenly distribute the sampling points on the hemisphere, the authors use Fibonacci sampling for N_s incident directions. PBR color c' is given by the equation:

$$c'(\omega_o) = \sum_{i=0}^{N_s} \big(f_d + f_s(\omega_o, \omega_i)\big) L_i(\omega_i)(\omega_i \cdot n)\Delta\omega_i \tag{2.13}$$

and color map C' by:

$$C' = \sum\nolimits_{i \in N} T_i \alpha_i c_i' \tag{2.14}$$

2.2.2 Point-based Ray Tracing

The authors propose a ray tracing technique based on the Bounding Volume Hierarchy (BVH). Specifically, they construct a binary radix tree from a given set of 3D Gaussians, where each leaf node represents the tight bounding box of a Gaussian, and each internal node denotes the bounding box of its two children. The process of ray tracing on 3D Gaussian points can be described as follows:

1. Traverse BVH recursively to find Gaussians along the ray. Starting from the root node of the binary radix tree, intersection tests are recursively performed between the ray and the bounding volumes of each node's children. Upon reaching a leaf node, the associated Gaussian is identified.
2. For each Gaussian:
    a. Compute t_j (parametric location on the ray where it contributes most).
    b. Use t_j to get r_x, the estimated intersection point.
    c. Compute a_j (opacity at r_x).
    d. Multiply a_j into the transmittance T_i.
3. Repeat for all Gaussians, progressively accumulating transmittance T_i.

$$T_i = (1 - a_{i-1})T_{i-1}, \qquad for\ i = 1, \dots, j-1\ with\ T_1 = 1 \tag{2.15}$$

4. Stop if transmittance drops below a threshold T_{min}.

The final value of T_i is independent of the order in which Gaussians are encountered.

2.2.2.1 Intersection Points

Unlike polygons, Gaussians do not have a sharp boundary. This complicates the definition of an exact intersection. The authors approximate the intersection of a ray with a 3D Gaussian as a point where the 3D Gaussian's contribution peaks. The intersection point is defined as:

$$r_x = r_o + t_j r_d \tag{2.16}$$

where r_o is the ray origin (camera position), r_d is the ray direction (unit vector) and t_j is a scalar representing the position along the ray where the Gaussian is most likely to contribute. Its computed as:

$$t_j = \frac{(\mu - r_o)^T \sum r_d}{{r_d}^T \sum r_d} \tag{2.17}$$

2.2.2.2 Visibility Term

The visibility term V for incident direction ω_i is obtained from the computed transmittance T_i. $V(\omega_i)$ is pre-computed across the hemispherical domain determined by n for each Gaussian and is subsequently integrated into the rendering equation. This is feasible since Relightable 3DGS solely focuses on static scenes. This pre-computation happens right after the geometry optimization training phase and right before the material and lighting optimization.

2.2.3 Normal Estimation

They start by assigning a normal attribute (vector) n for each 3D Gaussian. n is randomly initialized from the unoptimized point cloud and is optimized via backpropagation. For every camera pose and each Gaussian in it, they compute the depth d_i. Using the "over" operator to perform alpha compositing, the authors compute the depth and normal maps D and N. The process is similar to that performed for color map calculation and is described by the following function:

$$\{D, N\} = \sum_{i \in N} w_i T_i a_i \{d_i, n_i\} \tag{2.18}$$

where $w_i = \frac{T_i a_i}{\sum_{i \in N} T_i a_i}$ is the weight for each Gaussian along a viewing ray and $T_i = \prod_{j=1}^{i-1}(1 - a_j)$ is the transmittance of the ray up to the i-th Gaussian.

2.2.3.1 Normal Densification

The core idea is to add more Gaussians in areas where the normal vectors change rapidly. This usually happens at edges, corners, or detailed regions. The authors introduce an additional criterion to the densification strategy of vanilla 3DGS, one that involves the gradients of normals. Specifically, they choose to densify Gaussians whose normal gradient exceeds a threshold T_n.

2.2.4 Loss Functions

2.2.4.1 Total losses

The training process is divided into two stages. The loss in the first stage is given by:

$$L = \lambda_1 L_1 + \lambda_{ssim} L_{ssim} + \lambda_n L_n + \lambda_{s,n} L_{s,n} + \lambda_O L_O + \lambda_u L_u \tag{2.19}$$

where $\lambda_1 = 0.8, \lambda_{ssim} = 0.2, \lambda_n = 0.01, \lambda_{s,n} = 0.01, \lambda_O = 0.01, \lambda_u = 0.01$.

The loss in the second stage is given by:

$$L = \lambda_1 L_1 + \lambda_{ssim} L_{ssim} + \lambda_l L_l + \lambda_{s,b} L_{s,b} + \lambda_{s,r} L_{s,r} \tag{2.20}$$

where $\lambda_1 = 0.8, \lambda_{ssim} = 0.2, \lambda_l = 0.0001, \lambda_{s,b} = 0.01, \lambda_{s,r} = 0.01$.

Apart from the Mean Absolute Error L_1 and the Structural Similarity Loss L_{ssim}, the following constraints are applied:

2.2.4.2 Normal

A pseudo-normal map $\widetilde{N}$ is computed from the rendered depth map D under the local planarity assumption. They encourage the consistency between the rendered normal map N and the pseudo-normal map $\widetilde{N}$ with the loss function:

$$L_n = \left\| N - \widetilde{N} \right\|_2 \tag{2.21}$$

2.2.4.3 Depth Distribution

Depth uncertainty causes Gaussians to be too spread out, leading to inaccurate surfaces. The authors introduce a depth distribution constraint to distribute Gaussians tightly around an object's surface. The expected pixel depth is given by the equation

$D = \sum_{i \in N} w_i d_i$ while the expected squared depth by $D_{sq} = \sum_{i \in N} w_i d_i^2$. They minimize the uncertainty by minimizing the following loss:

$$L_u = D_{sq} - D^2 \tag{2.22}$$

2.2.4.4 Object Mask

In the case of an object mask the authors further constrain the optimization by a cross entropy loss[110]:

$$L_O = -M \log O - (1 - M) \log(1 - O) \tag{2.23}$$

where M is the object mask and $O = \sum_{i \in N} T_i a_i$ the contribution of the Gaussian to the rendered pixel.

2.2.4.5 Light

The authors apply a light regularization penalizing deviations from neutral white lighting (i.e. balanced light intensity across all color channels). L_c represents the color intensity in each color channel and $\frac{1}{3}\sum_c L_c$ computes the average light intensity across all channels.

$$L_{light} = \sum_c \left(L_c - \frac{1}{3} \sum_c L_c \right), c \in \{R, G, B\} \tag{2.24}$$

2.2.4.6 Smoothness

Moreover, the authors apply smoothness constraints $L_{s,r}$, $L_{s,n}$, $L_{s,b}$ on roughness, normal and albedo maps respectively. For reference, the roughness constraint is described as follows:

$$L_{s,r} = \|\nabla R\| \exp(-\|\nabla C_{gt}\|) \tag{2.25}$$

2.2.5 Architecture and Information Flow

2.2.5.1 Training

The optimization process is divided into two stages. The first stage involves the geometry reconstruction and the visibility term precomputation. The second stage addresses the material and lighting optimization. The input consists of dataset images along with a JSON file containing information on them and camera intrinsics/extrinsics. Furthermore, the authors have performed a series of processing steps to extract depth maps, normal maps and object masks for the dataset images. They propose the usage of Vis-MVSNet[111] to retrieve the photometrically and geometrically consistent depth maps. They use a differentiable computer vision library named Kornia to convert the depth maps to normal maps which are stored in Portable FloatMap (PFM) format. Finally, they propose the usage of NSVF[112] or IDR[113] for the creation of the object masks (PNG). We note that these maps and masks are expected from the pipeline meaning that if they are not part of the input dataset, training fails to complete. No environment map is supplied to the pipeline during training. This procedure is reserved for relighting.

After training for 30000 iterations, the output of the first part of training includes, in keeping with 3DGS, the optimized 3D point cloud, the camera parameters used for training (intrinsics and extrinsics), the training configuration arguments and the initial point cloud. Furthermore, training checkpoints are saved (PTH format), the interval of which is specified via an argument.

Rendering occurs during evaluation. It's output includes ground truth, rendered normals and RGB images. The metrics address PSNR, SSIM, LPIPS and L1 loss. Finally, a "collage" of rendered image, ground truth, depth map, opacity, normal and pseudo-normal is constructed for a sample of views.

The output of the second part adds to that of the first part in base color, cumulative light along with it broken down into global and local components, roughness and visibility renders for every dataset view. Since the second part of training adds another 10000 to 20000 iterations depending on dataset, the further optimized point cloud is included along with new training checkpoints. Finally, training checkpoints for the PBR light are saved in intervals specified via an argument.

2.2.5.2 Relighting

The authors provide a relighting script that takes an input of a trained model and an environment map and renders the model under the new lighting. The trained model must have gone through both parts of training. The environment map can be in HDR, EXR or PNG format although I have found that a high resolution tone-mapped PNG produces the best results. We also need to provide the script with the appropriate transformation matrices. Under the `config/` directory, we need to create a folder for each dataset. Inside it we need three JSON files: *transform* which concerns the trained model, *trajectory* for the camera intrinsics/extrinsics and *light_transform* for lighting. Each file contains one transformation matrix per model viewpoint (e.g., for a dataset with 200 images, we need 200 transformation matrices).

When running the script, we have to specify what should be included in the output. We can choose among normal, roughness, diffuse, specular, lights, local and global lights, visibility, base color, PBR and PBR with the environment map. We can do this via the `--capture_list` flag of the script. Moreover, we can produce a video of the relight object via the `--video` flag. For the purpose of our testing, we limit the capture list to PBR which includes the RGB renders of the relit object.

2.3 Reflective Gaussian Splatting

As the name suggests, Reflective Gaussian Splatting[114] proposes a framework for real-time rendering of reflective surfaces with inter-reflections in 3D space. First, it splats Gaussians into pixel-level maps of albedo, normals, roughness, metallicity, and opacity. Second, it ray traces on a lightweight, periodically extracted mesh to determine occlusion. Third, it uses a physically based deferred renderer that applies BRDF terms to the rendering equation via a split-sum approximation to handle both specular and diffuse components. The approximation alongside a per-Gaussian, spherical-harmonic indirect term are blended back through the splats to produce the final image. To further improve geometry quality, the method adopts 2D Gaussian primitives, runs an initial per-Gaussian shading pass to guide convergence, and propagates normals from high-metallic, low-roughness regions.

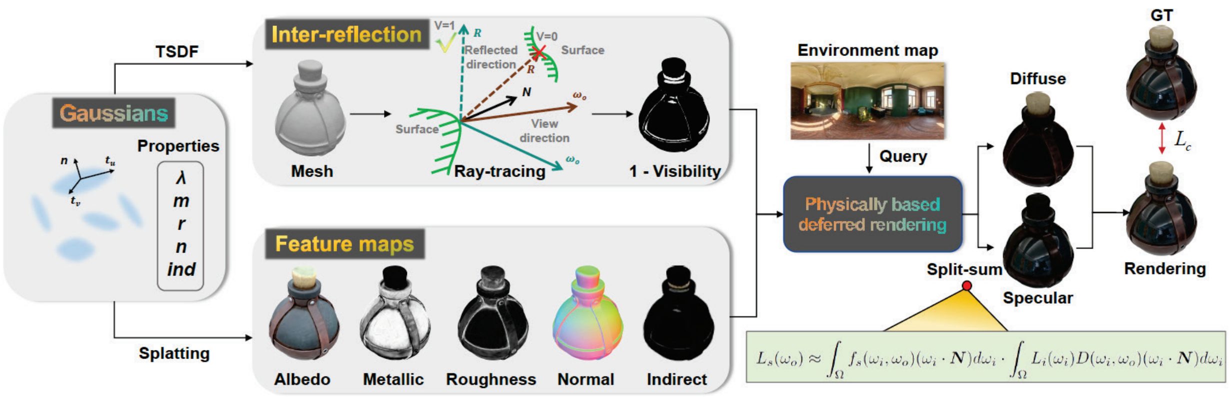

Figure 2.6 Reflective Gaussian pipeline. Figure from [114].

2.3.1 2D Gaussian Primitive

View-consistent, 2D[115] oriented, planar Gaussian disks are used as rendering primitives. Each 2D Gaussian is defined by its position p, tangential vectors t_u and t_v, and scaling factors (s_u, s_v) that control its shape and size. Its influence is given by:

$$G(u,v) = \exp\left(-\frac{u^2+v^2}{2}\right), \tag{2.26}$$

where (u,v) are coordinates in the local tangent space. The ray-splat intersection (u,v) in this space is obtained via:

$$x = (xz, yz, z, 1)^T = WH(u, v, 1,1)^T, \tag{2.27}$$

where $H = \begin{bmatrix} s_u t_u & s_v t_v & 0 & p \\ 0 & 0 & 0 & 1 \end{bmatrix} \in R^{4\times4}$. x represents the homogeneous ray passing through pixel (x, y) and intersecting the 2D Gaussian disk at depth z.

2.3.2 Illumination Model

2.3.2.1 Deferred PBR

In standard PBR, we typically apply the rendering equation at the shading stage per pixel, using material properties and scene lighting to evaluate how each surface point reflects light toward the camera. In naive Gaussian Splatting approaches, shading is done per primitive (i.e., directly on the Gaussians). However, this can cause instability during optimization (noisy gradients) because alpha blending of overlapping Gaussians creates ambiguity since multiple Gaussians can contribute to the same pixel, making it hard to optimize individual ones when shading happens too early.

To address this, Ref-Gaussian adopts Deferred PBR, which shifts shading after alpha blending. By doing so, noisy per-Gaussian gradients are turned into smooth per-pixel ones. It also allows shading to operate on coherent surfaces making material estimation and light transport more physically grounded.

Each Gaussian i is associated with a set of material related properties like albedo $\lambda \in [0,1]^3$, metallic $m \in [0, 1]$, and roughness $r \in [0, 1]$. Instead of shading each Gaussian independently, Ref-Gaussian first aggregates all of these per-pixel by blending them using the classic front-to-back alpha compositing rule. For each attribute x_i, the pixel-level aggregated quantity X is given by:

$$X = \sum_{i=1}^{N} x_i a_i \prod_{j=1}^{i-1} (1 - a_j), \tag{2.28}$$

where $X = \begin{bmatrix} \Lambda \\ M \\ R \\ N \end{bmatrix}, x_i = \begin{bmatrix} \lambda_i \\ m_i \\ r_i \\ n_i \end{bmatrix}$.

This way, each pixel is assigned a composited material “signature” X making the optimization more stable since the combined contribution of many Gaussians is accounted for.

Having obtained the material maps, the rendering function is applied directly at the pixel. For the outgoing radiance $L(\omega_o)$ in direction ω_o the rendering function is:

$$L(\omega_o) = \int_{\Omega} L_i(\omega_i) f(\omega_i, \omega_o)(\omega_i \cdot N) d\omega_i \tag{2.29}$$

where $L_i(\omega_i)$ is incoming radiance from direction ω_i and $f(\omega_i, \omega_o)$ is a simplified version of the Disney BRDF[77] model representing the integral of reflected incident light over the hemisphere Ω. N is the surface normal.

2.3.2.2 BRDF

Since BRDF consists of both diffuse and specular terms, the integral can also be divided into two parts:

$$f(\omega_i, \omega_i) = f_d + f_s \tag{2.30}$$

The diffuse term is computed by querying the pre-integrated environment map using the normal N and multiplying with the material terms. The specular term of the BRDF is given by:

$$f_s(\omega_i, \omega_o) = \frac{DGF}{4(\omega_o \cdot N)(\omega_i, N)} \tag{2.31}$$

where D, F, and G represent the GGX normal distribution function, the Fresnel term, and the shadowing-masking term respectively.

A typical – but expensive – way to solve the above integral term is Monte Carlo sampling. The authors propose the use of split-sum approximation which “breaks” the specular computation into two precomputable parts:

$$L_s(\omega_o) \approx \int_{\Omega} f_s(\omega_i, \omega_o)(\omega_i \cdot N) d\omega_i \cdot \int_{\Omega} L_i(\omega_i) D(\omega_i, \omega_o)(\omega_i \cdot N) d\omega_i \tag{2.32}$$

which simplifies into:

$$L_s(\omega_o) \approx BRDF\_Integration \cdot Lighting\_Integration \tag{2.33}$$

The first term depends solely on surface roughness R and the dot product of normal and view direction $(\omega_i \cdot N)$, which accounts for the cosine falloff of incoming light relative to the surface normal. This allows the results to be precomputed and stored in a 2D lookup texture map.

The second term represents the integral of incident radiance over the specular lobe – or, how light is distributed across the environment – and can be pre-integrated at the beginning of each training iteration with different roughness levels. That allows to efficiently employ a series of cube maps to represent the environment lighting by performing trilinear interpolation using the reflected direction and the roughness as parameters.

2.3.2.3 Gaussian Inter-Reflection

This split-sum formulation assumes that the environment is fully visible in the reflected direction. In reflective scenes with complex geometry, nearby surfaces can occlude reflection paths, meaning the environment contribution is partially or completely blocked. This phenomenon is called inter-reflection. In traditional PBR, modeling this requires expensive global illumination techniques.

Ref-Gaussian addresses this by introducing a visibility factor $V \in \{0,1\}$, which determines whether the reflected direction $R = 2(\omega_o \cdot N)N - \omega_o$ is occluded. This vector R is the perfect specular reflection direction, derived from the view vector ω_o and surface normal N.

To determine visibility in this direction, Ref-Gaussian ray traces from the surface along R against a dynamically extracted mesh that is created by using the Truncated Signed Distance Function (TSDF) fusion. To accelerate the ray tracing process for visibility checks, the authors employ a Bounding Volume Hierarchy (BVH). If the ray hits the mesh, $V = 0$; otherwise, $V = 1$.

This allows the specular component to be rewritten conditionally as:

$$L_s'(\omega_o) \approx (\int_\Omega f_s(\omega_i, \omega_o)(\omega_i \cdot N) d\omega_i) \cdot [L_{dir} \cdot V + L_{ind} \cdot (1 - V)] \qquad (2.34)$$

where L_{dir} is the environment lighting sampled in direction R, used when visibility is unobstructed ($V = 1$). L_{ind} approximates indirect specular light, used when reflection is blocked ($V = 0$).

L_{dir} corresponds to the second term of the $L_s(\omega_o)$ integral. The indirect component L_{ind} is not computed from light bounces directly (which would be too expensive). Instead, it's learned as a per-Gaussian, view-dependent radiance field, parameterized using spherical harmonics. Each Gaussian stores a set of coefficients l_{ind} encoding how much light it emits in different directions after being hit indirectly.

To produce a per-pixel indirect lighting estimate, the spherical harmonics values from all visible Gaussians are composited using the same alpha blending that governs Gaussian Splatting:

$$L_{ind} = \sum_{i=1}^{N} l_{ind} a_i \prod_{j=1}^{i-1} (1 - a_j) \tag{2.35}$$

This equation means that each Gaussian contributes to the final indirect light seen at a pixel in a depth-aware, front-to-back manner. Instead of explicitly modeling inter-Gaussian radiance transfer, the authors "gate" the environment map with binary visibility and fill in the occluded areas with a learned light field.

In a nutshell, ray tracing is used only for binary visibility tests (on a lightweight mesh), and light transport is handled by learned spherical functions instead of physically simulated bounces.

2.3.3 Normal Estimation

The normal vector $n \in [0,1]^3$ is derived from the tangential vectors of each 2D Gaussian using $n = t_u \times t_v$.

2.3.4 Geometry Optimizations

2.3.4.1 Initial Stage with per-Gaussian Shading

When deferred shading is used from the beginning, the model aggregates material features first, then shades the composited surface at each pixel. While this is stable during rendering, it's problematic for early-stage optimization, because it smooths out the gradient signals that are essential for adjusting the Gaussians' positions and shapes.

To address this, the authors introduce an initialization phase in which the shading is done per-Gaussian, not per-pixel. In this phase:

- Each Gaussian computes its own outgoing radiance using its own material and normal properties.
- This radiance is then alpha-blended into the image plane.

In this initial stage, instead of applying the rendering equation on the composite vector X, it is applied on each x_i directly. So, instead of:

$$X = \sum_{i=1}^{N} x_i a_i \prod_{j=1}^{i-1} (1 - a_j), followed\ by\ L = shade(X) \tag{2.36}$$

they perform:

$$L = \sum_{i=1}^{N} shade(x_i) a_i \prod_{j=1}^{i-1} (1 - a_j) \tag{2.37}$$

Once the geometry is reasonably established, the training transitions to deferred shading.

2.3.4.2 Material-Aware Normal Propagation

In PBR, specular highlights are highly sensitive to the surface normal. A small error in the normal can shift the reflection direction significantly. During early training, normals are unstable, particularly for smooth or reflective materials where there's no texture variation to help constrain the shape. Ref-Gaussian addresses this by periodically propagating normal information from well-estimated Gaussians to nearby ones—guided by their material attributes.

Specifically, Gaussians with:

- High metallicity (close to 1), and
- Low roughness (close to 0)

are observed to be more likely to produce accurate gradients. The system temporarily increases the scale of these Gaussians, which causes their normals to influence a broader region of the image.

2.3.5 Loss Functions

The total training loss L is:

$$L = L_c + \lambda_n L_n + \lambda_{smooth} L_{smooth} \tag{2.38}$$

where $\lambda_n = 0.05$ and $\lambda_{smooth} = 1.0$.

2.3.5.1 Color Reconstruction Loss

This term encourages the rendered image to match the ground truth, combining an L1 loss with a D-SSIM (Differentiable Structural Similarity Index):

$$L_c = (1 - \lambda) \cdot L_1 + \lambda \cdot D - SSIM \tag{2.39}$$

Here, $\lambda = 0.2$ balances pixel-wise accuracy (L1) with structural similarity (D-SSIM), which is perceptually aligned.

2.3.5.2 Normal Consistency Loss

To ensure that the rendered normal map matches the geometry implied by the depth, they define:

$$L_n = 1 - \tilde{N}^T N \tag{2.40}$$

where N is the rendered normal from the composited material map and $\tilde{N}$ is the ground truth or depth-derived surface normal. The loss penalizes angular deviation via cosine similarity.

2.3.5.3 Edge-Aware Normal Smoothness

This term encourages smooth normals, but only in regions without strong image gradients (i.e., uniform surfaces):

$$L_{smooth} = \|\nabla N\| \cdot \exp\left(-\left\|\nabla C_{gt}\right\|\right) \tag{2.41}$$

where ∇N is the spatial gradient (change) of the normal field and ∇C_{gt} is the gradient of the ground truth color image.

2.3.6 Architecture and Information Flow

2.3.6.1 Training

Training consists of two stages. An initial stage of 18.000 steps for the per-Gaussian rendering and a final stage of 40.000 steps for the deferred rendering. The final stage takes an input of only the geometry of the Gaussians. All colors and materials are reset before it.

The learning rates of the trainable material attributes are all set to 0.005. The learning rate of the environment map is set to 0.01. Normal propagation is conducted to those Gaussians with $metallic \geq 0.02$ and $roughness \leq 0.1$. The initial roughness value is set to 0.1.

The input consists of train and test views of the applicable dataset along with their respective transformation matrices plus, the sparse point cloud resulting from SfM. The output contains the optimized point cloud (saved on set intervals), renders of normal maps and reconstructed views along their ground truth images, “collaged” iterations of ground truth, render, base color map, specular map, reflection strength map, roughness map, rendered alpha, surface depth, rendered normal and surface normal. Finally, the learned environment maps are instantiated as PNG files.

We note that the actual rendering process (i.e., the production of maps and images) takes place during evaluation that also results in PSNR, SSIM, LPIPS metrics for the trained scene.

2.3.6.2 Relighting

To perform relighting with Ref-Gaussian, first we have to specify the new environment map. To do so, we place our HDR file in the `ref-gaussian/output/model_name/point_cloud/iteration_50000` directory and rename it as `point_cloud.hdr`. Then, we can simply re-render the views through the evaluation script, this time passing the `--relight` flag.

2.4 IRGS: Inter-Reflective Gaussian Splatting with 2D Gaussian Ray Tracing

IRGS: Inter-Reflective Gaussian Splatting with 2D Gaussian Ray Tracing[116] uses inverse rendering to recover a scene's geometry, materials, and lighting (including indirect bounces) from a set of images by embedding the full rendering equation into a Gaussian-splatting framework. It makes use of a fully differentiable 2D Gaussian ray tracer that computes visibility and incident radiance on the fly, and a two-stage Monte Carlo sampling and optimization scheme for material and lighting estimation. Starting from a pretrained 2D Gaussian model, IRGS first rasterizes Gaussians into pixel-level G-buffers (depth, normals, albedo, roughness) and initializes material parameters. It then traces secondary rays through the Gaussian field to gather direct lighting from an environment map and single-bounce indirect lighting via its ray tracer. Then, it applies a physically based BRDF to combine these terms and optimizes all scene and lighting parameters end-to-end using reconstruction and perceptual losses. This pipeline results In accurate relighting and material recovery.

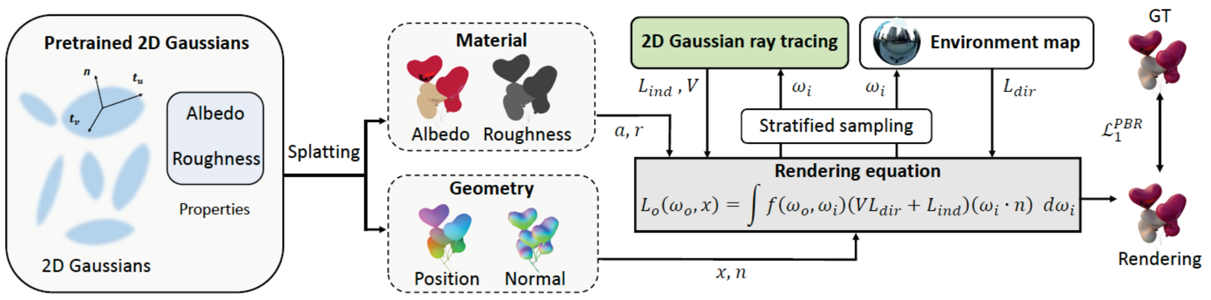

Figure 2.7 IRGS pipeline. Figure from [116].

2.4.1 2D Gaussian Primitive

Each 2D Gaussian is defined by a center point $\mu \in R^3$, an opacity parameter $o \in [0,1]$, two principal tangential vectors $t_u \in R^3$ and $t_v \in R^3$ – which are orthogonal to the disk's normal – and a scaling vector $s = (s_u, s_v) \in R^2$ that defines the elliptical spread along t_u and t_v. The Gaussian disk lies in the tangent plane at μ, with its spatial influence described via a standard anisotropic Gaussian:

$$G(p) = \exp\left(-\frac{u^2 + v^2}{2}\right) \tag{2.42}$$

where $u = \frac{1}{s_u}(p - \mu)^T t_u$ and $v = \frac{1}{s_v}(p - \mu)^T t_v$. It defines the Gaussian weight of a point p relative to the center μ, using the local 2D basis vectors.

2.4.2 2D Gaussian Ray Tracing

While 3D Gaussian Splatting (3DGS) offers fast and high-quality rendering via rasterization, it fundamentally lacks a principled mechanism to compute indirect illumination, as rasterization cannot capture secondary rays or visibility for arbitrary directions. A possible solution to this is ray tracing 3D Gaussians.

The authors examine the case of 3DGRT[117]. There, the intersection of a ray with a 3D Gaussian is defined as the point along the ray where the Gaussian density reaches its maximum. This point is not necessarily where the Gaussian would contribute most to the rendered image in a rasterization context, because 3DGS projects Gaussians onto the image plane and accumulates their influence in screen space, not in 3D along rays.

This discrepancy means that applying 3DGRT on a pretrained 3DGS model – which was trained under the screen-space projection and blending assumptions – results in a misalignment between the intended influence of the splats and the actual light transport computed via ray tracing. The effect is a visible degradation in rendering quality, as shown by lower PSNR and perceptual artifacts.

IRGS proposes 2DGRT which operates directly on the 2D Gaussian primitives. These are defined in the image plane with explicit projection from the start. Because the geometry and radiance accumulation are already screen-space aligned, ray-splat intersections are unambiguous and consistent with the pretrained model. All steps of 2DGRT are differentiable allowing backpropagation of gradients throughout the ray tracing process.

2.4.2.1 Ray/Gaussian Intersection

To perform ray tracing, the intersection point p between a ray and a Gaussian must be calculated. A ray is defined as:

$$r(\tau) = r_o + \tau r_d \tag{2.43}$$

where r_o is the origin, r_d is the direction, and τ is the scalar distance along the ray. Each 2D Gaussian disk lies on a plane with normal $n = t_u \times t_v$. The intersection p of the ray with this plane occurs at:

$$p = r_o + \tau r_d, \qquad where \quad \tau = \frac{n^T(\mu - r_o)}{n^T r_d}. \tag{2.44}$$

2.4.2.2 Bounding Proxy

Because ray/Gaussian intersection needs to be efficient (especially for many Gaussians), each Gaussian is approximated with a bounding mesh – specifically, an icosahedron.

To construct this, the authors follow 3DGRT's strategy but adapted for 2D disks. The key is to scale the icosahedron to fully contain the Gaussian's spatial support above a minimum influence threshold a_{min}. The vertex positions of the bounding mesh are given by:

$$v \leftarrow \sqrt{2\log(o/a_{min})}(s_u t_u \;\; s_v t_v \;\; \epsilon 1)v + \mu \tag{2.45}$$

where ϵ denotes a small positive number. Each icosahedron mesh consists of 20 triangular faces, resulting in a total of $20N$ triangles for N 2D Gaussian primitives.

The logarithmic term ensures the boundary encloses the region of significant influence (i.e., where $G(p) \geq a_{min}$), while the basis vectors and scaling shape the bounding volume to match the Gaussian's anisotropy. These bounding meshes are inserted into a Bounding Volume Hierarchy (BVH) for GPU-accelerated ray tracing via NVIDIA OptiX. Finally, by applying the k −buffer per-ray sorting algorithm, they obtain the exact ordering of intersected Gaussians for each ray. For this, k is set to 16.

2.4.2.3 Accumulating Radiance via Ray Tracing

Once the intersections between rays and the 2D Gaussians are found, the contributions are blended using standard front-to-back alpha compositing:

$$(c_{rt}, o_{rt}) \leftarrow Trace(r_o, r_d) \tag{2.46}$$

providing the accumulated color and opacity along a given ray. Each intersected Gaussian contributes:

$$C = \sum_{i=1}^{N} a_i c_i \prod_{j=1}^{i-1} (1 - a_j) \tag{2.47}$$

2.4.3 Illumination Model

IRGS estimates geometry, material, and lighting from posed RGB images by integrating a full physically-based rendering equation into the Gaussian Splatting framework using differentiable 2DGRT. The method has two stages.

2.4.3.1 Stage I: 2D Gaussian Training

First, they train a standard 2D Gaussian splatting model to produce the geometry (depth and normals) and view-dependent color. RGB rendering is performed via alpha blending. Normalized depth and normal maps are rendered via:

$$\{D, N\} = \sum_{i=1}^{N} w_i \{d_i, n_i\}, \qquad where\ w_i = \frac{T_i a_i}{\sum_{i=1}^{N} T_i a_i} \tag{2.48}$$

From the rendered depth and normal maps, the authors determine the surface position x and the corresponding surface normal vector n for each pixel coordinate.

2.4.3.2 Stage II: Inverse Rendering

Performing shading on each Gaussian can lead to inaccurate and blurred results since the rendered normal map is supervised at the pixel level after rasterization. Instead, IRGS proposes applying the rendering equation after rasterization – at the pixel level with:

$$L_o(\omega_o, x) = \int_{\Omega} f(\omega_o, \omega_i, x) L_i(\omega_i, x)(\omega_i \cdot n) d\omega_i \tag{2.49}$$

where ω_o is the outgoing direction, ω_i is the incident/incoming direction over the hemisphere Ω defined by the surface normal n. f is bidirectional reflectance distribution function (BRDF) and L_i is the radiance of the incident light.

2.4.3.3 Stage II: BRDF

The authors propose a simplified Disney BRDF model[77] which assumes dielectric material and is parameterized with only albedo α and roughness r. The pixel-level albedo map A and the roughness map R are obtained via alpha blending during rasterization:

$$\{A, R\} = \sum_{i=1}^{N} w_i\{a_i, r_i\} \tag{2.50}$$

The BRDF is divided into a diffuse term $f_d = \frac{\alpha}{\pi}$ and a specular termf_s:

$$f_s(\omega_i, \omega_o, x) = \frac{DFG}{4(\omega_i \cdot n)(\omega_o \cdot n)} \tag{2.51}$$

where D, F, and G represent the normal distribution function, the Fresnel term and the geometry term respectively.

2.4.3.4 Stage II: Incident Light

The incident radiance L_i at surface point x with direction ω_i is split into direct and indirect radiance:

$$L_i(\omega_i, x) = V(\omega_i, x)L_{dir}(\omega_i) + L_{ind}(\omega_i, x) \tag{2.52}$$

where $L_{dir}(\omega_i)$ is modeled by an environment cube map. $V(\omega_i, x)$ and $L_{ind}(\omega_i)$ are queried using 2D Gaussian raytracing:

$$(L_{ind}(\omega_i, x), 1 - V(\omega_i, x)) \leftarrow Trace(x, \omega_i) \tag{2.53}$$

from surface point x in direction ω_i, retrieving occlusion and bounced radiance. The RGB values used in ray tracing correspond to the view-dependent color c from the first stage.

2.4.3.5 Stage II: Rendering

The authors evaluate the rendering equation via Monte Carlo integration over the hemisphere of incoming directions. To reduce variance in the Monte Carlo estimate, they employ stratified sampling over the hemisphere. Rather than randomly picking

N_r directions over the whole hemisphere, they partition its domain into N_r equal-area bins and sample one direction per bin.

Each pixel corresponds to a unique surface point x. To compute outgoing radiance L_o for that pixel, the rendering equation must be integrated at that location. Therefore, N_r refers to the number of sampled incoming directions used to evaluate the rendering equation at that pixel. So, each pixel independently uses N_r samples.

Since evaluating 2DGRT per ray is expensive, the authors cap the total number of rays per training iteration to N_{rays}. So, they only evaluate the rendering equation at:

$$\left\lceil \frac{N_{rays}}{N_r} \right\rceil \textit{ randomly selected pixels per iteration} \tag{2.54}$$

The final rendering result is computed as:

$$c_{pbr} = \frac{2\pi}{N_r} \sum_{i=1}^{N_r} (f_d + f_s) L_i(\omega_i, x)(\omega_i \cdot n) \tag{2.55}$$

2.4.3.6 Relighting

Given a sampled incident direction, we can find the intersecting Gaussians by performing 2D ray tracing. Then, IRGS aggregates albedo, roughness and normal by alpha blending $\{A_r, R_r, N_r\} = \sum_i \omega_i(a_i, r_i, n_i)$ where $\omega_i = \frac{T_i a_i}{\sum_i T_i a_i}$. These are then shaded using a pre-filtered environment map (avoiding the expensive Monte Carlo sampling). L_{ind} can be split into diffuse and specular terms i.e., $L_{ind} = L_d + L_s$. By applying the split sum approximation to the specular term, we get

$$L_s \approx \int_{\Omega} f_s(\omega_i, \omega_o)(\omega_i \cdot N_r) d\omega_i \cdot \int_{\Omega} L_i(\omega_i) D(\omega_i, \omega_o)(\omega_i \cdot N_r) d\omega_i \tag{2.56}$$

The left term can be pre-integrated into a 2D lookup texture map since it only depends on $(\omega_i \cdot N_r)$ and roughness. The left part represents the integral of incident radiance and can also be pre-integrated. This way, L_d and L_s can be obtained by a single query.

2.4.4 Loss Functions

2.4.4.1 Stage I

$$L^1 = L_c + \lambda_n L_n + \lambda_d L_d + \lambda_{s,n} L_{s,n} + \lambda_O L_O \tag{2.57}$$

with $\lambda_n = 0.05$, $\lambda_d = 1000$, $\lambda_{s,n} = 0.02$ and $\lambda_O = 0.01$.

L_c is the RGB reconstruction loss, L_n is the normal consistency loss and L_d is the depth distortion loss. The authors introduce two additional loss functions. An edge aware smoothness loss $L_{s,n}$ on the normal map:

$$L_{s,n} = \|\nabla N\| \exp\left(-\left\|\nabla C_{gt}\right\|\right) \tag{2.58}$$

and a binary cross entropy loss L_O using an object mask M:

$$L_O = -M \log O - (1 - M) \log(1 - O) \tag{2.59}$$

where $O = \sum_{i=1}^{N} T_i a_i$ is the accumulated opacity map.

2.4.4.2 Stage II

$$L^2 = L^1 + \lambda_1^{pbr} L_1^{pbr} + \lambda_{light} L_{light} + \lambda_{s,a} L_{s,a} + \lambda_{s,r} L_{s,r} \tag{2.60}$$

where $\lambda_1^{pbr} = 1.0$, $\lambda_{light} = 0.01$, $\lambda_{s,a} = 2.0$ and $\lambda_{s,r} = 2.0$.

L^1 is the training loss of the first stage and L_1^{pbr} is an L1 loss used to supervise the final color c_{cbr}.

Operating under the assumption that the diffuse component of indirect illumination is approximately white-balanced, the authors introduce a white light prior to "push" the RGB values toward their mean. L_{light} is given by the equation:

$$L_{light} = \sum_c (L_{diffuse} - \frac{1}{3} \sum_c L_{diffuse}), \qquad c \in \{R, G, B\} \tag{2.61}$$

where $L_{diffuse} = \frac{1}{N_r} \sum_{i=1}^{N_r} L(\omega_i, x)$ is the average incident radiance across the sampled directions ω_i, focusing on the diffuse component (excluding specular reflection).

$L_{s,a}$ and $L_{s,r}$ are the edge-aware smoothness regularization terms applied on the rendered albedo and roughness maps respectively.

2.4.5 Architecture and Information Flow

The training process consists of two stages. The first stage that is responsible for the geometry reconstruction trains for 40.000 iterations. The seconds stage adds another 20.000 to perform the material decomposition. During optimization, the environment map used for environmental lighting is represented as a 32×32 cube map. The rendering equation is evaluated with $N_r = 256$ rays. At each training iteration, BVH is updated and $N_{rays} = 2^{18}$ rays are sampled. Ray tracing is terminated when transmittance falls below 0.03.

IRGS receives the typical assortment of datasets as inputs (Blender, LLFF, COLMAP etc.). Training output consists of the regular suspects, optimized point cloud, visualizations of the different maps and learned environments and evaluation metrics. Rendering this output results in images of base color, diffuse, roughness, specular, visibility, normals, alpha, along with the ground truth and renders of the final reconstructed model with and without it's environment map.

2.4.5.1 Relighting

To perform relighting, we place our EXR environment maps in `irgs/assets/env_map` directory. From there, we can simply run the evaluation scripts for relighting after we modify them to include the new environment map paths. Two issues; first, to perform this operation our dataset needs to include ground truth views for every environment map used. Second, we need to calculate the albedo scaling before relighting. Hopefully, the authors include the appropriate script for some dataset formats.

Results are stored in `output/model_name/irgs/test_rli_light/env_map_name` directory. These include the ground truth images of the model with and without the environment map and the rendered images again, with and without the environments. Metric results are also produced by the above procedure.

2.5 ReCap: Better Gaussian Relighting with Cross-Environment Captures

ReCap: Better Gaussian Relighting with Cross-Environment Captures[118] uses multiple captures of the same object under different, unknown lighting conditions to untangle surface albedo (material) from lighting. Given as input, k sets of object appearances from unknown lighting conditions, k learnable environment maps are created. It treats each capture as a separate task with its own learnable environment map while sharing a single set of Gaussian-based material attributes (base color, roughness and specular tint). The shading function is based on the split-sum microfacet approximation. Together with post-processing, these environment maps are tied to the shared material model in a joint optimization. 2D images are rasterized with standard Gaussian splatting and used for loss computation. The use of multiple lighting conditions during training helps the model separate material and illumination resulting in physically meaningful learned lighting representations, better material estimates and more accurate relighting.

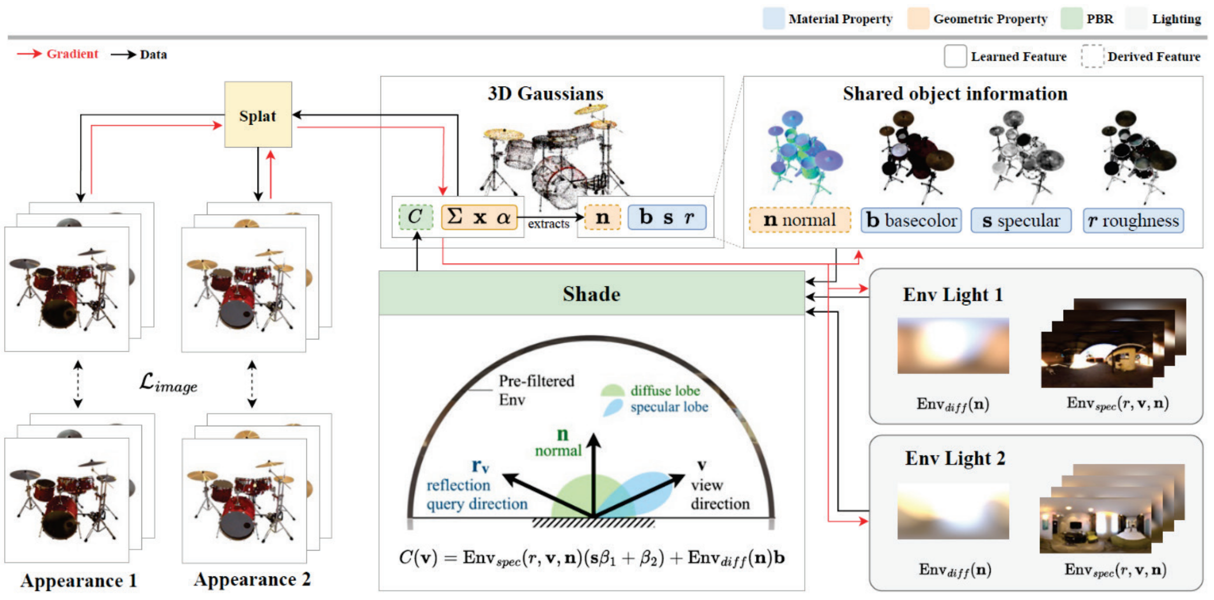
Material Property
Geometric Property
PBR
Lighting
Gradient
Data
Learned Feature
Derived Feature
Splat
3D Gaussians
Shared object information
C Σ $\mathbf{x}$ α extracts $\mathbf{n}$ $\mathbf{b}$ $\mathbf{s}$ r
$\mathbf{n}$ normal
$\mathbf{b}$ basecolor
$\mathbf{s}$ specular
r roughness
Shade
Env Light 1
$\mathrm{Env}_{diff}(\mathbf{n})$
$\mathrm{Env}_{spec}(r, \mathbf{v}, \mathbf{n})$
Env Light 2
$\mathrm{Env}_{diff}(\mathbf{n})$
$\mathrm{Env}_{spec}(r, \mathbf{v}, \mathbf{n})$
$\mathcal{L}_{image}$
Pre-filtered Env
diffuse lobe
specular lobe
$\mathbf{n}$ normal
$\mathbf{r_v}$ reflection query direction
$\mathbf{v}$ view direction
$C(\mathbf{v}) = \mathrm{Env}_{spec}(r, \mathbf{v}, \mathbf{n})(\mathbf{s}\beta_1 + \beta_2) + \mathrm{Env}_{diff}(\mathbf{n})\mathbf{b}$
Appearance 1
Appearance 2

Figure 2.8 ReCap pipeline. Figure from [118].

2.5.1 Gaussian Primitive

ReCap uses the explicit point cloud representation of 3D objects as proposed in the seminal paper of 3DGS[26].

2.5.2 Illumination Model

When it comes to view-dependent color $c(v)$, 3DGS uses spherical harmonics. This representation is not well suited for relighting since it abstracts the interactions between material, lighting and geometry. ReCap proposes an alternative that factors out the influence of light, namely, a shading function. The authors start with the classic rendering equation. The outgoing radiance at Gaussian point x, viewed from direction v is given by the classic rendering equation:

$$L_{out}(x,v) = \int_{\Omega} f_r(x,v,l)L_{in}(x,l)(n \cdot l)dl \tag{2.62}$$

where Ω is the hemisphere above the surface, f_r is the bidirectional reflectance distribution function (BRDF), l is the incident light direction and L_{in} is the incoming radiance.

2.5.2.1 BRDF

ReCap adapts an Epic Games simplification[119] that is based on the Disney BRDF[77]. The parameters it uses are base color $b \in [0,1]^3$, roughness $r \in [0,1]$, metallic $m \in [0,1]$ and specular $s = 0.04$ (constant for non-metals). It is described by:

$$f_r(x,v,l) = (1-m)\frac{b}{\pi} + \frac{D(r,n,l,v)F(b,m,s,l,v)G(r,n,l,v)}{4(l \cdot n)(v \cdot n)} \tag{2.63}$$

where D, F and G are the normal distribution function, the Fresnel term and the geometry term respectively.

2.5.2.2 Shading Function

When it comes to implementing the rendering equation, it is commonly simplified as shading functions. By substituting Eq. (2.62) to the rendering equation (2.61) and applying the split sum approximation[119] we get:

$$L_{out}(x,v) = \underbrace{E_d(n)(1-m)b}_{diffuse} + \underbrace{E_s(n,v)[F_0\beta_1(r,n,v) + \beta_2(r,n,v)]}_{specular} \tag{2.64}$$

where E_d and E_s are prefiltered environment maps for diffuse and specular reflectance. E_d stores the cosine-weighted irradiance, while E_s approximates specular reflection by blurring the environment according to surface roughness. β_1 and β_2 are precalculated BRDF lookup tables that store precomputed scalar terms used to approximate the specular BRDF response, depending on view angle and roughness.

$F_0 = mb + (1 - m)s$ is the normal incidence from Schlick's approximation[75] of the Fresnel term. It is the reflection coefficient for light incoming parallel to the normal (i.e., perpendicular to the surface). Here it is parameterized by base color and metallic although the equation provided by the authors appears to be reversed[1]. For metals, $m = 1$ and $F_0 = b$. For non-metals, $m = 0$ and $F_0 = s$.

Eq. (2.63) represents a linear blend of two different models controlled by the metallic parameter. Continuing with the above mentioned categorization of metals/non-metals, and given the authors' calculation of the F_0 normal incidence, we can discretize (2.63) like:

$$\begin{aligned} L_{metal} &= E_s b \beta_1 + E_s \beta_2 \\ L_{non-metal} &= E_s s \beta_1 + E_s \beta_2 + E_d b \end{aligned} \tag{2.65}$$

however, optimizing the metallic parameter is troublesome since there are ambiguities both between s and b, and $E_s\beta_1$ and E_d. For that reason, the authors drop the metallic parameter while expanding the specular parameter:

$$L_{out} = E_s s \beta_1 + E_s \beta_2 + E_d b \tag{2.66}$$

where $s \in [0,1]^3$ is a vector representing the specular tint. For $s = [s, s, s]^T$, we get $L_{non-metal}$. For $s = b_{metal}$ and $b = 0$, we get L_{metal}.

2.5.2.3 Post-shading Processing

After shading, the resulting radiance L_{out} is clipped to remove out-of-range values, then gamma corrected to convert it from physical (linear) space into perceptual (non-linear) space. This post-processing happens per Gaussian, before the colors are used in alpha blending.

Applying gamma correction at this stage is necessary because the shading model is based on linear light transport. Without gamma correction, relighting quality degrades, even though novel view synthesis (NVS) might still look acceptable. Previous

[1] Note that normal incidence is more commonly found as $F_0 = (1 - m)b + ms$ (i.e., $F_0 = s$ for metals and $F_0 = b$ for dielectrics)[74] or set at a constant value, typically 0.04[120].

works often skipped gamma or used tone-mappers like Reinhard or ACES, but these introduced non-linearities that hurt optimization. ReCap uses simple clipping and gamma because this keeps gradients stable and training effective.

By structuring the pipeline this way, learned maps in linear HDR, shading in linear space, and post-processing before compositing, ReCap ensures that external HDR maps can be used directly for relighting. No normalization or scaling is needed. This also allows the model to absorb residual lighting effects into the Gaussians themselves, which act as pseudo light sinks. These sinks are shaped by HDR-aware shading and processing, helping them approximate ambient light patterns in a way that is consistent across environments.

2.5.2.4 Lighting Representation

To represent light, ReCap adopts an image based lighting model[121]. Each environment map is represented by a $6 \times 256 \times 256$ learnable cube map and is pre-filtered into a diffuse map E_d and a set of specular mipmaps across different roughness levels E_s. In practice, they use the approximation provided by NVDIFFRAST[122] which performs differentiable pre-filtering and querying of the learnable environment maps in every forward pass during shading.

2.5.3 Normal Estimation

ReCap adopt the shortest axis of the converged Gaussians as an approximation of the normal vectors. This method follows the observation that during training, Gaussians tend to flatten and align with the object surface.

The authors use just a depth-derived normal to constrain the approximated normal. Prior methods that use learned normals tend to overfit to specific lighting, baking specular highlights into the geometry. This leads to visually good reconstructions under the training light but fails under relighting. ReCap's cross-lighting supervision discourages this overfitting, separating lighting effects from geometry and producing more accurate normals and relit highlight shapes.

2.5.4 Loss Functions

Apart from the standard $\mathcal{L}_1$ reconstruction loss, ReCap makes use of the following loss functions.

2.5.4.1 Specular Tint

The expanded range of s in the proposed general expression of the shading function introduces unnatural specular colors. To avoid over-saturation of the specular tint, the authors propose a saturation penalty on s:

$$\mathcal{L}_{sat} = \lambda_{sat} \cdot \|s - s_{mean}\| \tag{2.67}$$

2.5.4.2 Energy Conservation

To account for energy conservation, the authors include an extra regularizer $\mathcal{L}_{energy}$ to encourage:

$$\|s\| + \|b\| \leq 1 \tag{2.68}$$

2.5.4.3 Normal Loss

When it comes to normal calculation, ReCap makes use of a single constraint; that of a depth-derived normal $\hat{n}$. Depth images are rendered from the Gaussian opacities. The depth-normal consistency loss is given by:

$$\mathcal{L}_{dn} = \lambda_{dn} \cdot \|n - \hat{n}\|^2 \tag{2.69}$$

2.5.5 Architecture and Information Flow

ReCap trains for the standard 30.000 iterations. Where it differs, however, is its ability to train on more than one environment maps with the authors noting that the more they increase the number of environment maps (up to five), the better results they yield on relighting performance.

As shown in Figure 2.9, the same object position can have a different color depending on viewing direction and lighting. These variations can be explained by multitude of material/lighting combinations forming what the authors call, the albedo-lighting ambiguity. By training with more environment captures, ReCap is able to attribute these variations to the corresponding learnable environment achieving better decoupling between geometry and albedo.

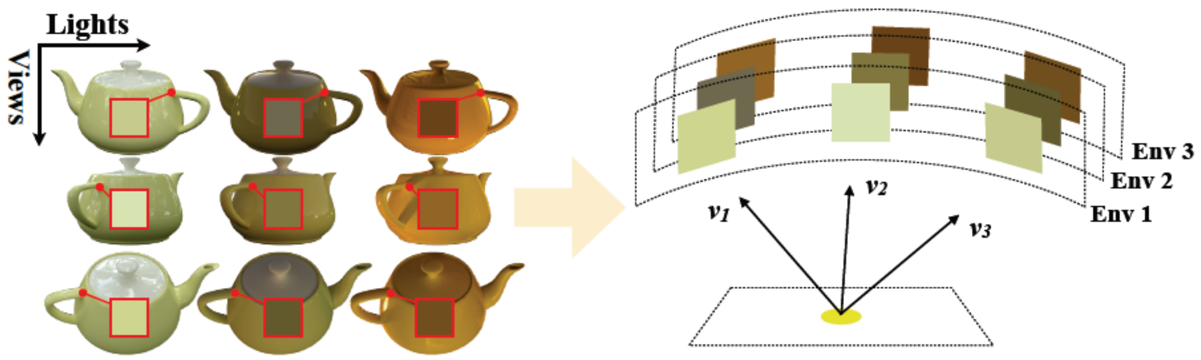

Figure 2.9 Visualization of pixel color's dependance on viewing angle and scene lighting. Figure from [118].

This differentiation however poses a new challenge. That of the appropriate dataset format. Specifically, in order to train with ReCap, we must possess ground truth images of the relit objects on the different environment maps that we will use. In other words, we need to pre-render the different datasets with the environment maps of our choice and supply them as input.

The output of ReCap includes renders of albedo, depth, diffuse, normal, reflectance, roughness and specular maps for a selection of viewing angles, the learned base and diffuse environment maps plus, the rendered images in all viewing angles for every environment map. Relighting is embedded to the training procedure, in the sense that the training script includes this procedure. We simply have to specify the model path and the new environment maps for training and/or relighting.

Chapter 3

Results and Comparative Analysis

3.1 Experiments setup

In this chapter we present the quantitative and qualitative results of novel view synthesis and relighting tasks for each Gaussian project under examination. All experiments were conducted on a single NVIDIA RTX 4080 16GB.

3.1.1.1 Datasets

For the datasets, we wanted to capture the performance on both synthetic and real objects. For synthetic, we chose *drums* from NeRF Synthetic[15], *coffee* and *helmet* from Shiny Blender[123], *train* from Tanks and Temples[124] and *gnome* from Stanford-Orb[125]. NeRF Synthetic included objects with complex geometries and non-Lambertian materials. Shiny Blender further expands our selection with glossy synthetic data with complex material properties. Tanks and Temples includes real-world scenes with larger-scale environments. Stanford-Orb includes real-world objects and was created for evaluating inverse rendering tasks.

3.1.1.2 Evaluation metrics

The quantitative metrics used are Peak Signal-to-Noise Ratio (PSNR)[126], Structural Similarity Index Measure (SSIM)[127] and Learned Perceptual Image Patch Similarity (LPIPS)[128]. PSNR evaluates on a color-wise basis and is widely used to measure the quality of a reconstructed image. SSIM is used to measure the similarities between two images with one of them being a distortion-free reference (ground truth). It considers changes in structural information, perceived luminance, and contrast. LPIPS is a perceptual metric that measures the human-perceived similarity between two images. Unlike PSNR and SSIM that calculate differences based on raw pixel values, LPIPS uses the distance between features extracted by a convolutional neural network (CNN) pretrained on an image classification task. For PSNR and SSIM, higher values are better. For LPIPS the lower the better. Furthermore, we include FLIP comparisons between ground truth and rendered image for Novel View Synthesis evaluation. This produces error map images helping us visualize the problematic regions.

3.1.1.3 Environment maps

Relighting was tested with five different environment maps, namely Adams Place Bridge[129], Courtyard[130], Shady Patch[131], Hotel Room[132] and Moonless Golf[133]. The incentive behind this selection was to capture a variety of lighting conditions like interior, exterior, night, day, shaded etc. Depending on implementation, HDR, EXR or PNG files might be necessary for the environment map representation. The website where we retrieved the maps from (referenced for each map) provides all of them, however, the 8k PNGs had to be scaled down to 1k. The resolution we settled on for the provided map is 1024x512 pixels.

3.2 GaussianShader

3.2.1 Novel View Synthesis

Table 3.1 Quantitative results in Novel View Synthesis with GaussianShader

Novel View Synthesis			
	PSNR↑	SSIM↑	LPIPS↓
drums	25.65	0.945	0.044
coffee	24.39	0.949	0.124
helmet	19.188	0.895	0.132
train	20.23	0.764	0.271
gnome	14.47	0.421	0.523

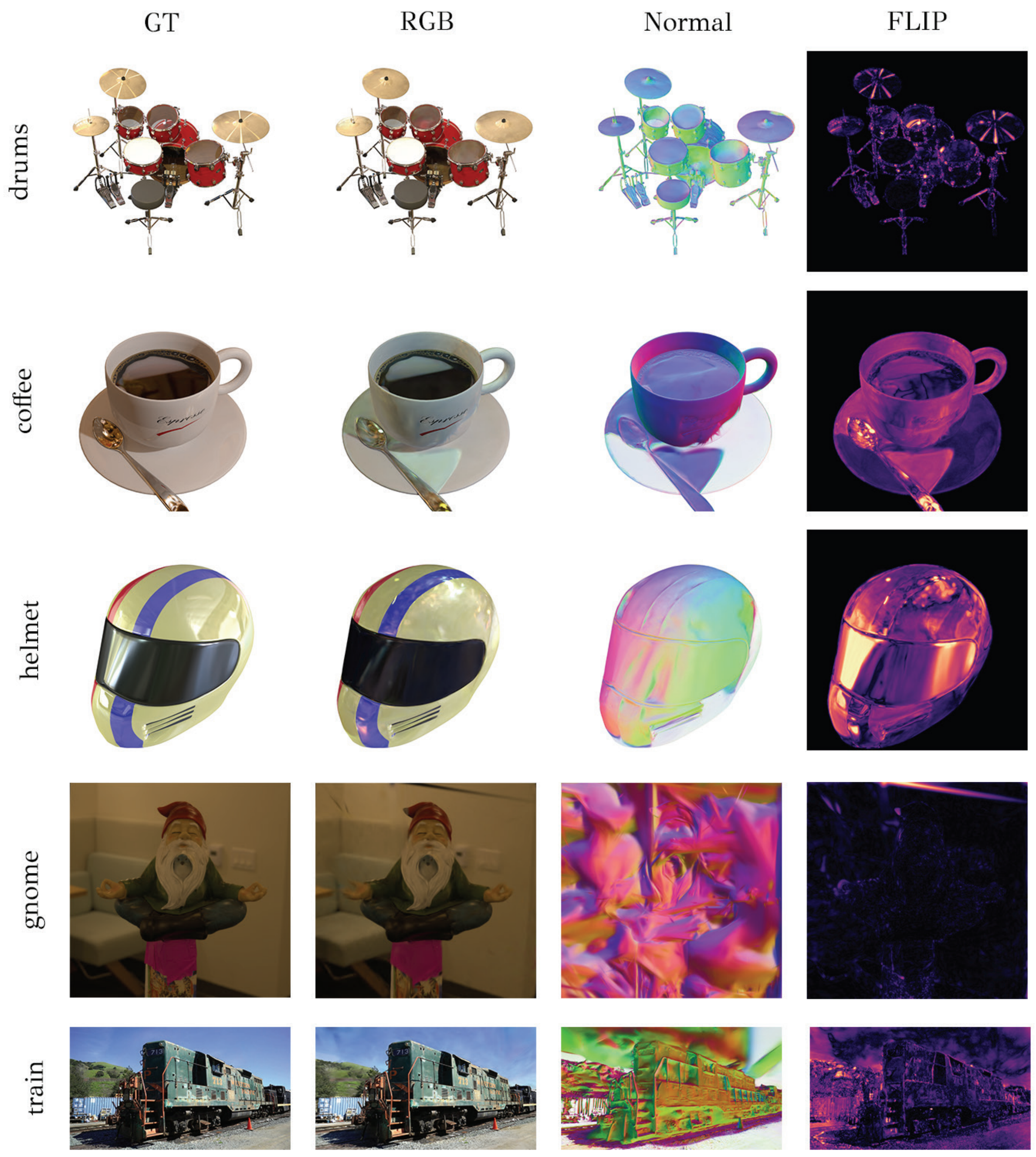

Figure 3.1 Qualitative results in Novel View Synthesis with GaussianShader

3.2.2 Relighting

Table 3.2 Quantitative results in relighting with GaussianShader

Relighting					
PSNR↑					
	Adams Place Bridge	Courtyard	Shady Patch	Hotel Room	Moonless Golf
drums	19.59	16.18	19.25	14.93	21.34
coffee	15.79	12.92	15.84	12.22	24.39
helmet	19.18	15.44	15.51	13.96	19.18
train	13.29	8.22	12.83	7.36	16.76
gnome	10.10	6.02	9.93	5.036	13.45
SSIM↑					
	Adams Place Bridge	Courtyard	Shady Patch	Hotel Room	Moonless Golf
drums	0.888	0.853	0.882	0.843	0.914
coffee	0.893	0.869	0.892	0.864	0.949
helmet	0.895	0.860	0.851	0.856	0.895
train	0.595	0.437	0.557	0.423	0.668
gnome	0.379	0.288	0.356	0.273	0.444
LPIPS↓					
	Adams Place Bridge	Courtyard	Shady Patch	Hotel Room	Moonless Golf
drums	0.076	0.106	0.079	0.124	0.062
coffee	0.157	0.191	0.170	0.201	0.124
helmet	0.132	0.210	0.180	0.216	0.132
train	0.377	0.530	0.403	0.546	0.342
gnome	0.613	0.720	0.621	0.717	0.568

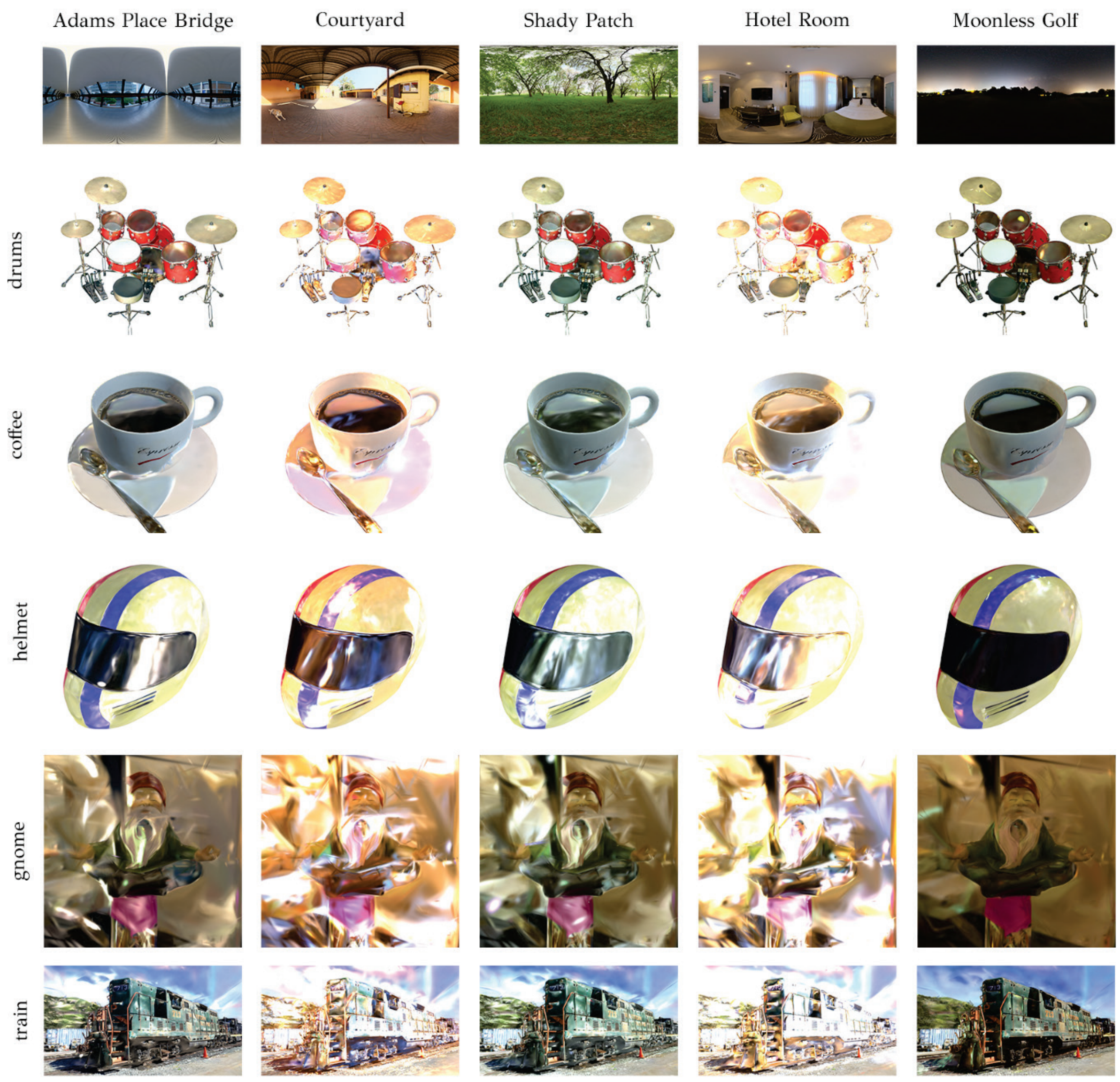

Figure 3.2 Qualitative results in relighting with GaussianShader

3.3 Relightable 3D Gaussians

3.3.1 Novel View Synthesis

Table 3.3 Quantitative results in Novel View Synthesis with R3DG

	Novel View Synthesis		
	PSNR↑	SSIM↑	LPIPS↓
drums	25.19	0.948	0.049
coffee	30.09	0.957	0.095
helmet	26.29	0.955	0.102
train	-	-	-
gnome	40.08	0.988	0.014

We note that in our setup, Relightable 3D Gaussians did not play well with the Tanks and Temples dataset. Specifically, due to the large images of the dataset and the very large number of points in the resulting from training point cloud, memory tends to become an issue. Despite trying different sample size for visibility calculation (authors propose 384, we went as far as 8 which is really low), all scenes of Tanks and Temples produced a CUDA Out Of Memory (OOM) error during memory allocation.

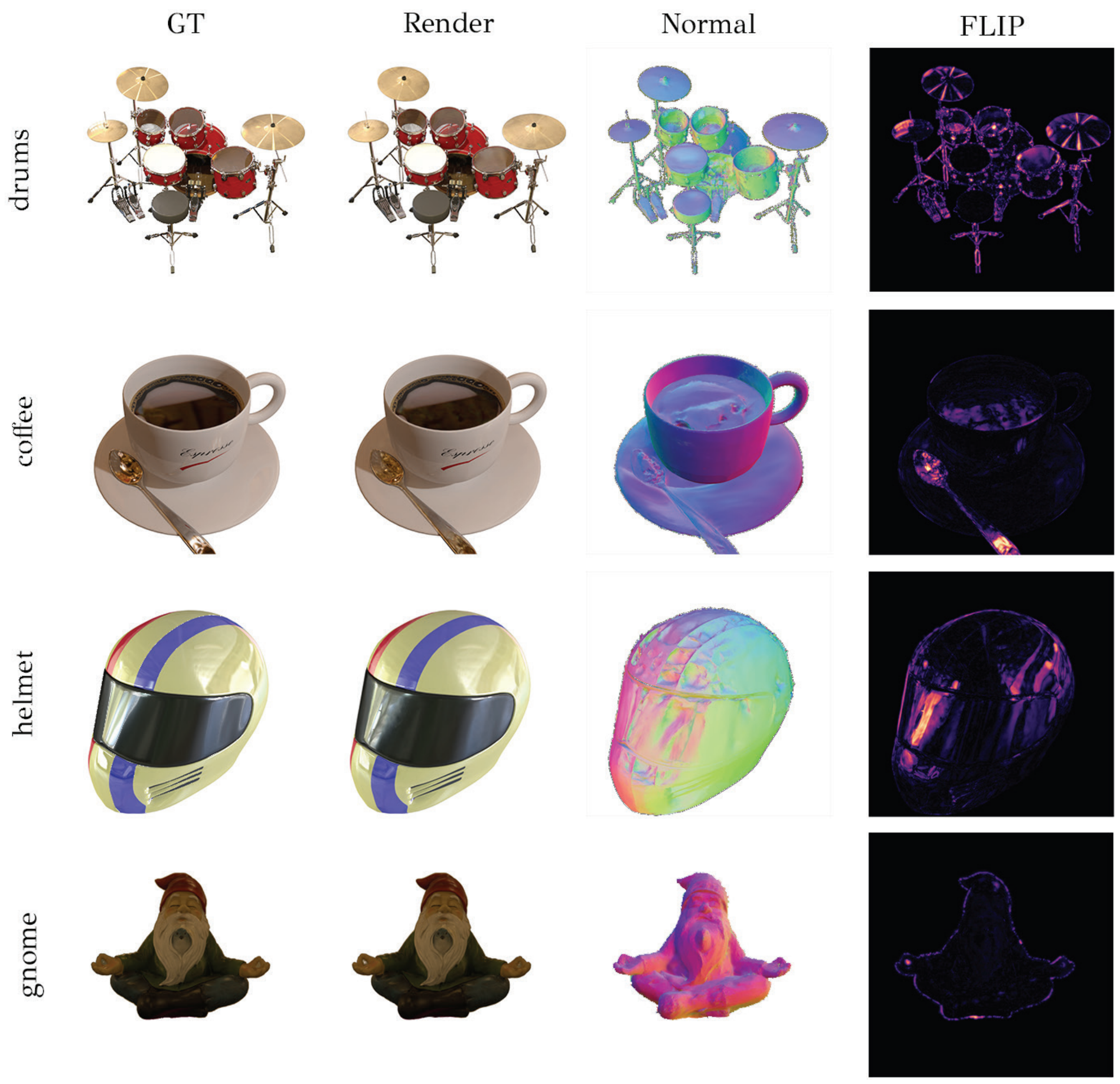

Figure 3.3 Qualitative results in Novel View Synthesis with R3DG

3.3.2 Relighting

Table 3.4 Quantitative results in relighting with R3DG

Relighting					
PSNR↑					
	Adams Place Bridge	Courtyard	Shady Patch	Hotel Room	Moonless Golf
drums	20.17	17.86	19.85	17.07	22.80
coffee	10.97	13.26	11.77	14.07	17.31
helmet	15.07	12.91	14.84	12.12	19.46
train	-	-	-	-	-
gnome	-	-	-	-	-
SSIM↑					
	Adams Place Bridge	Courtyard	Shady Patch	Hotel Room	Moonless Golf
drums	0.809	0.802	0.807	0.801	0.811
coffee	0.755	0.766	0.753	0.797	0.807
helmet	0.789	0.774	0.743	0.788	0.803
train	-	-	-	-	-
gnome	-	-	-	-	-
LPIPS↓					
	Adams Place Bridge	Courtyard	Shady Patch	Hotel Room	Moonless Golf
drums	0.183	0.192	0.187	0.195	0.176
coffee	0.269	0.253	0.277	0.249	0.209
helmet	0.272	0.285	0.280	0.306	0.238
train	-	-	-	-	-
gnome	-	-	-	-	-

Since Tanks and Temples' train scene failed to produce a reconstruction, we weren't able to test relighting with it. Moreover, while we were able to relight the gnome scene, we couldn't produce quantitative measurements for the different environment maps. That is because evaluating relit scenes requires ground truth images of the scene with that specific environment map. I.e., we needed to pre-render the scene with each environment map. Synthetic datasets are produced from 3D models. This enables us to create those pre-renders. Stanford-orb dataset, on the other hand, provides only the files necessary for reconstruction (images, camera files and ground truths). Pre-rendering with only ground truth point cloud proved unsuccessful, since critical geometry information is missing. Thus, we could only retrieve qualitative results on gnome relighting.

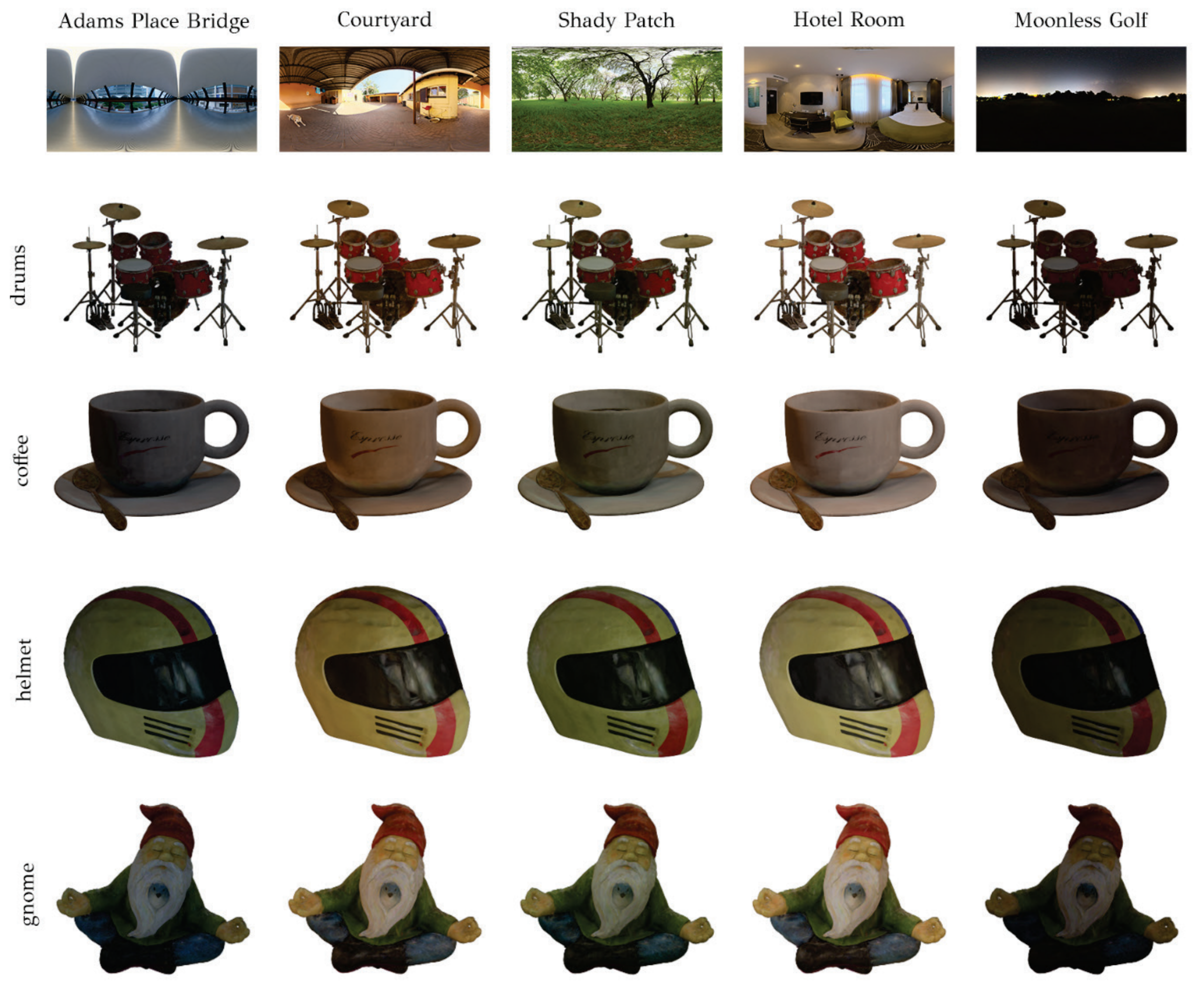

Figure 3.4 Qualitative results in relighting with R3DG

3.4 Reflective Gaussian

3.4.1 Novel View Synthesis

Table 3.5 Quantitative results in Novel View Synthesis with Ref-Gaussian

	Novel View Synthesis		
	PSNR↑	SSIM↑	LPIPS↓
drums	26.38	0.952	0.042
coffee	34.30	0.975	0.077
helmet	29.80	0.965	0.060
train	20.57	0.751	0.305
gnome	16.02	0.571	0.505

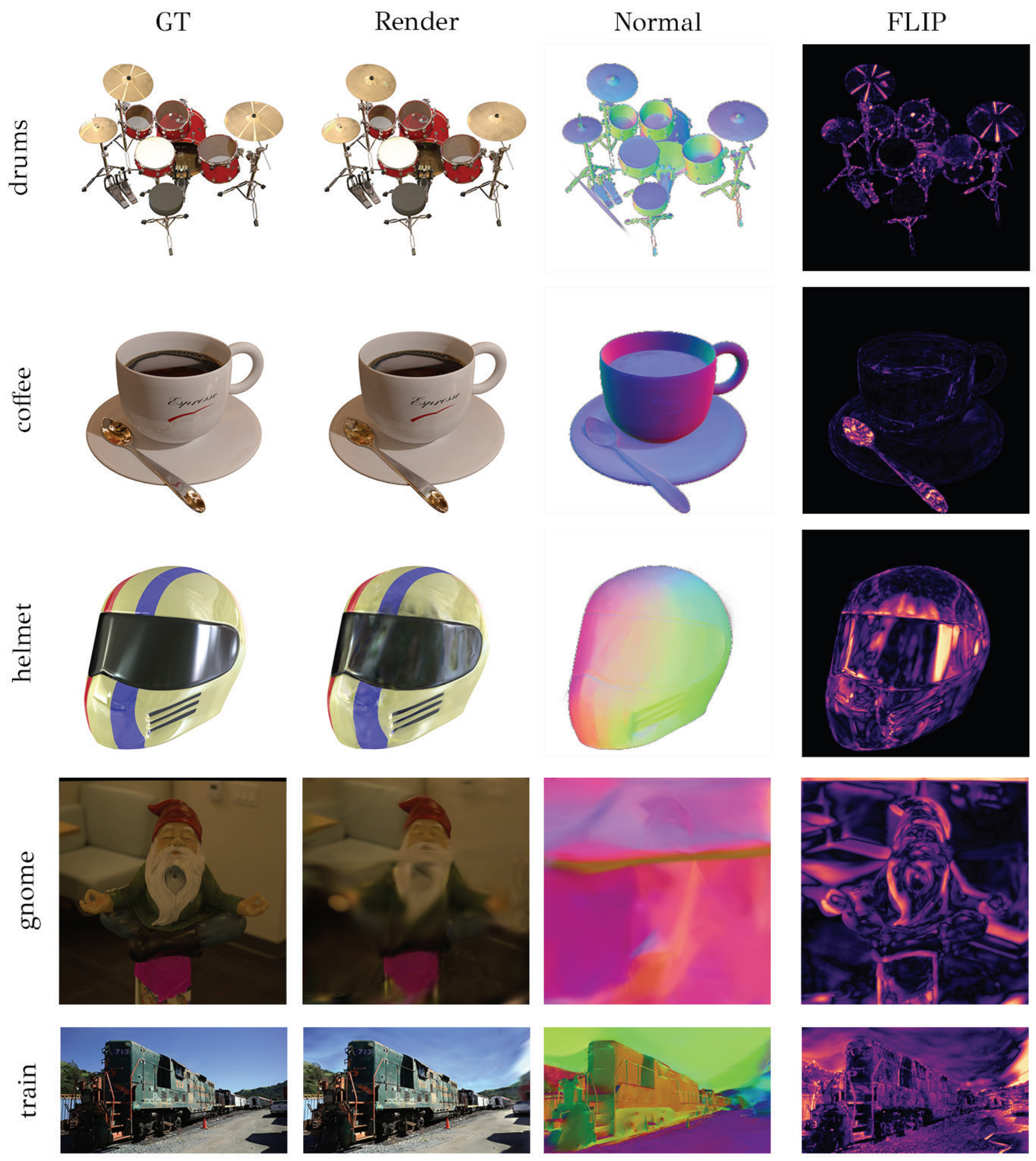

Figure 3.5 Qualitative results on Novel View Synthesis with Reflective Gaussian

3.4.2 Relighting

Table 3.6 Quantitative results in inferred relighting with Ref-Gaussian

Relighting					
PSNR↑					
	Adams Place Bridge	Courtyard	Shady Patch	Hotel Room	Moonless Golf
drums	25.44	25.01	24.89	24.71	25.39
coffee	29.62	29.16	28.75	28.06	30.41
helmet	23.15	21.02	21.08	20.84	22.44
train	20.22	19.51	19.92	19.81	19.98
gnome	15.92	15.77	15.91	15.75	15.94
SSIM↑					
	Adams Place Bridge	Courtyard	Shady Patch	Hotel Room	Moonless Golf
drums	0.947	0.943	0.942	0.941	0.948
coffee	0.964	0.964	0.962	0.960	0.968
helmet	0.927	0.917	0.904	0.912	0.930
train	0.744	0.728	0.738	0.735	0.741
gnome	0.555	0.553	0.541	0.561	0.559
LPIPS↓					
	Adams Place Bridge	Courtyard	Shady Patch	Hotel Room	Moonless Golf
drums	0.045	0.047	0.049	0.048	0.044
coffee	0.087	0.088	0.092	0.091	0.084
helmet	0.112	0.127	0.131	0.124	0.107
train	0.311	0.321	0.317	0.316	0.312
gnome	0.513	0.517	0.524	0.517	0.501

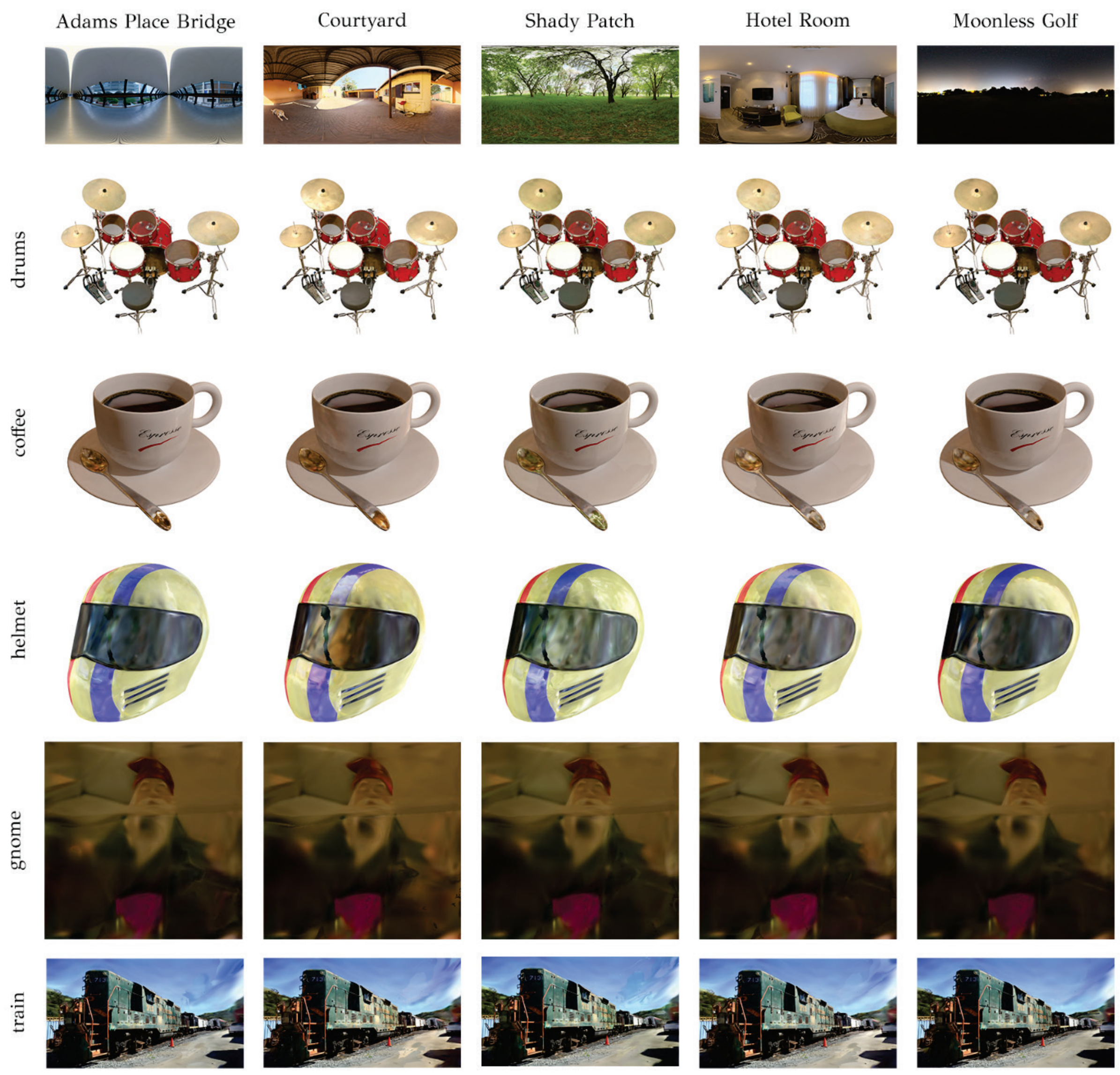

Figure 3.6 Qualitative results in relighting with Reflective Gaussian

3.5 IRGS

3.5.1 Novel View Synthesis

Table 3.7 Quantitative results in Novel View Synthesis with IRGS

Novel View Synthesis			
	PSNR↑	SSIM↑	LPIPS↓
drums	27.15	0.953	0.044
coffee	26.57	0.956	0.104
helmet	17.75	0.904	0.132
train	15.82	0.516	0.468
gnome	17.11	0.610	0.488

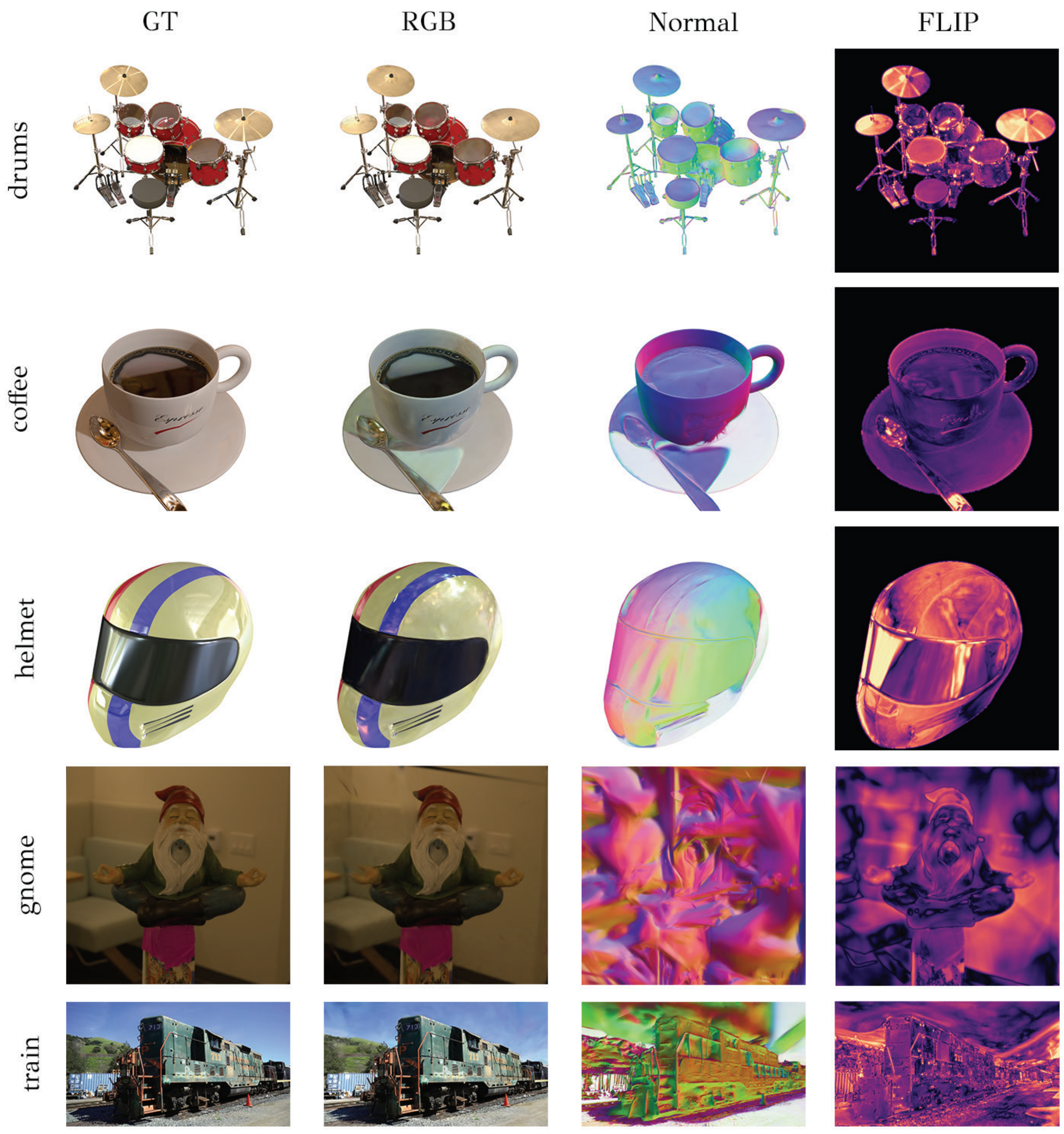

Figure 3.7 Qualitative results in Novel View Synthesis with IRGS

3.5.2 Relighting

Table 3.8 Quantitative results in inferred relighting with IRGS

Relighting					
			PSNR↑		
	Adams Place Bridge	Courtyard	Shady Patch	Hotel Room	Moonless Golf
drums	17.85	15.80	17.71	15.17	17.76
coffee	12.78	13.41	13.20	12.78	17.72
helmet	14.25	11.86	14.05	12.31	14.08
train	-	-	-	-	-
gnome	-	-	-	-	-
			SSIM↑		
	Adams Place Bridge	Courtyard	Shady Patch	Hotel Room	Moonless Golf
drums	0.797	0.786	0.792	0.789	0.794
coffee	0.766	0.761	0.767	0.768	0.813
helmet	0.785	0.752	0.729	0.786	0.769
train	-	-	-	-	-
gnome	-	-	-	-	-
			LPIPS↓		
	Adams Place Bridge	Courtyard	Shady Patch	Hotel Room	Moonless Golf
drums	0.196	0.205	0.202	0.210	0.199
coffee	0.259	0.265	0.259	0.284	0.220
helmet	0.333	0.344	0.336	0.333	0.331
train	-	-	-	-	-
gnome	-	-	-	-	-

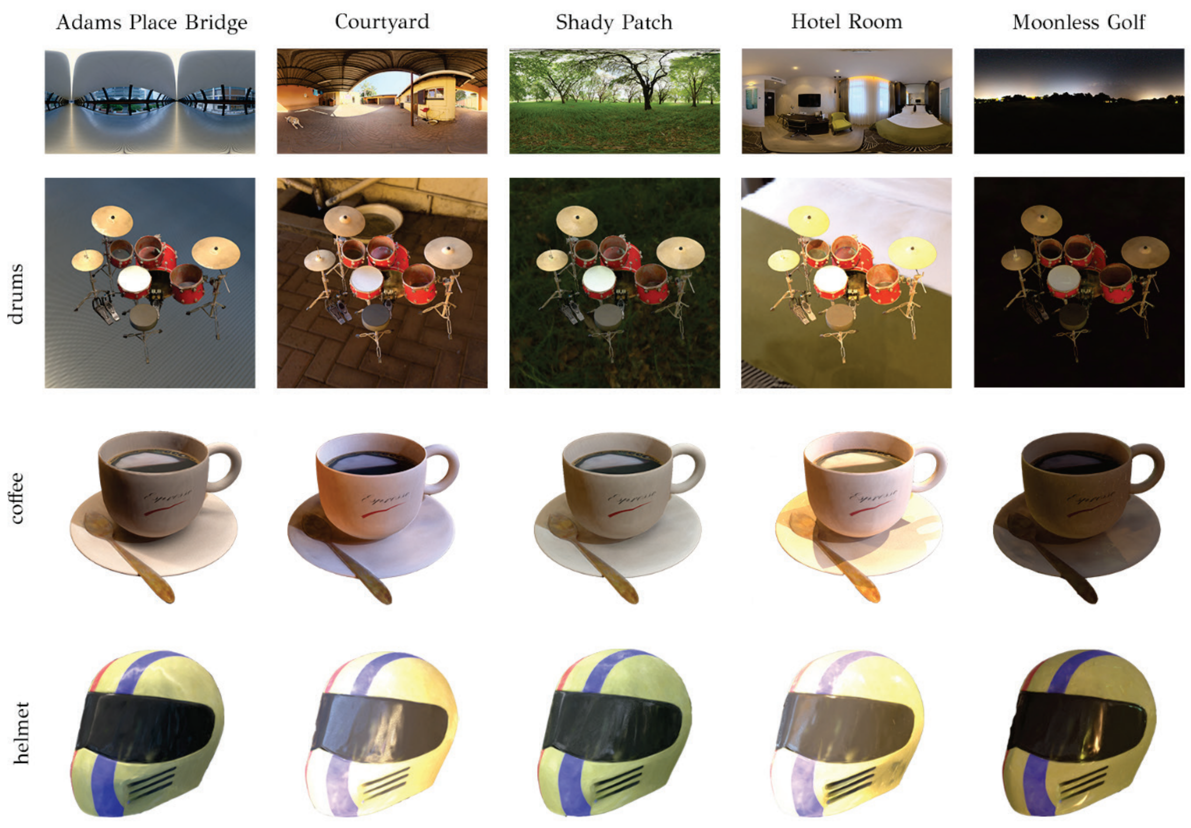

Figure 3.8 Qualitative results in relighting with IRGS

As we can observe in Figure 3.8, drums model carries it's background i.e., the environment map that it is rendered with. That is because IRGS failed to produce a clean reconstruction with a white background. Moreover, we failed to produce relighting results for the real-object datasets. The reason being that IRGS expects ground truth images for each environment map. Both Tanks and Temples and Stanford Orb dataset include no pre-renders and while we tried to render them ourselves with Blender and the scripts from [48], we were unable to.

3.6 ReCap

3.6.1 Novel View Synthesis

Table 3.9 Quantitative results in Novel View Synthesis with ReCap

Novel View Synthesis			
Adams Place Bridge			
	PSNR↑	SSIM↑	LPIPS↓
drums	29.63	0.967	0.028
coffee	31.46	0.962	0.127
helmet	29.49	0.963	0.069
train	-	-	-
gnome	-	-	-
Courtyard			
	PSNR↑	SSIM↑	LPIPS↓
drums	26.10	0.958	0.035
coffee	29.65	0.956	0.121
helmet	28.49	0.956	0.078
train	-	-	-
gnome	-	-	-

Although ReCap mentions that the authors had successfully trained and relit objects from the Stanford-Orb dataset, we had a hard time reproducing their results. The provided by the authors dataset did not include renders for this dataset under different environment maps (as it did for Nerf Synthetic and Shiny Blender). They propose the use of the LLFF dataset structure (provided by Stanford-Orb) that contains the sparse cross-environment alignment pairs and the COLMAP estimated poses for each environment map, both prerequisites for running ReCap. This format however is incompatible with the Blender dataset structure that is required by the implementation. Our attempt at rendering *gnome* with different environment maps using Blender also proved unsuccessful since the geometry reconstruction of the scene is not available in advance. The same thing applied for *train* scene of the Tanks and Temples dataset although here, we only had the COLMAP formatted dataset without environment information. Thus, we present our NVS and relighting results with only synthetic scenes, namely, *drums*, *coffee* and *helmet*.

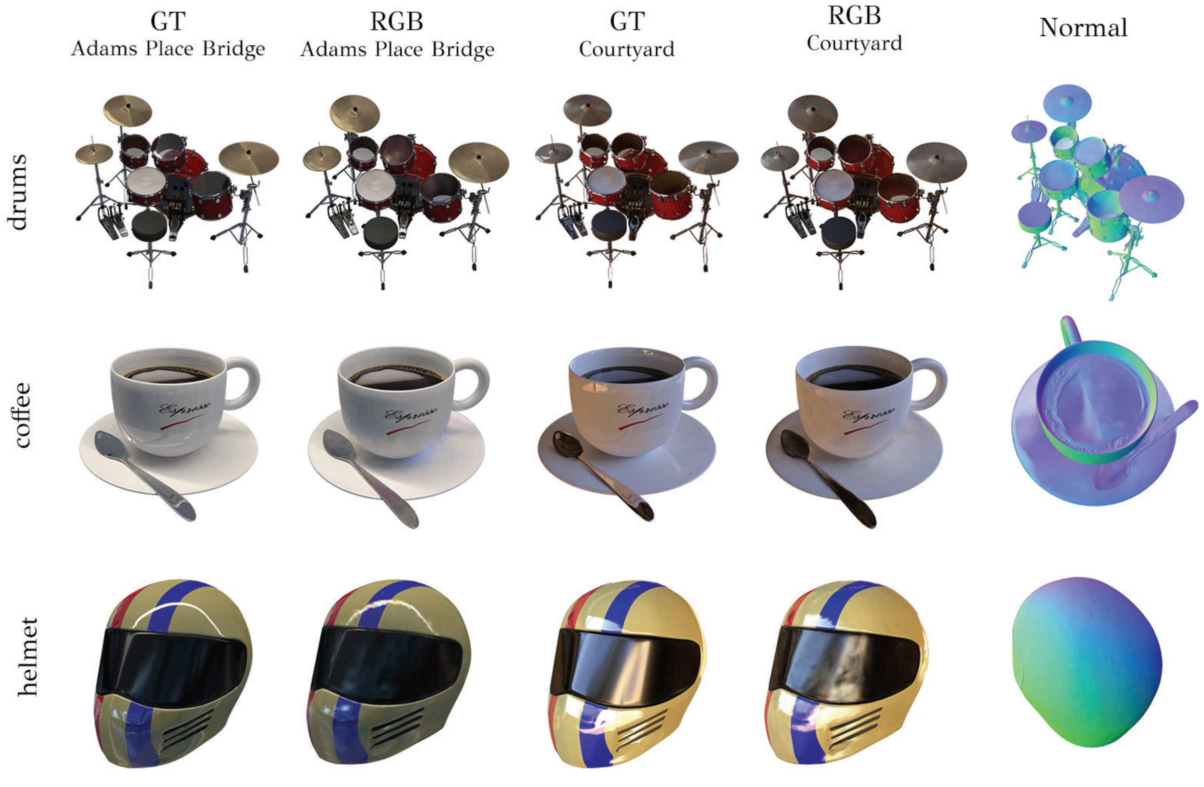

Figure 3.9 Qualitative results in Novel View Synthesis with ReCap

3.6.2 Relighting

Table 3.10 Quantitative results in relighting with ReCap

Relighting					
PSNR↑					
	Adams Place Bridge	Courtyard	Shady Patch	Hotel Room	Moonless Golf
drums	26.37	24.37	24.46	23.39	27.19
coffee	21.59	22.57	20.03	21.81	23.05
helmet	23.18	22.06	22.16	22.58	25.51
train	-	-	-	-	-
gnome	-	-	-	-	-
SSIM↑					
	Adams Place Bridge	Courtyard	Shady Patch	Hotel Room	Moonless Golf
drums	0.957	0.949	0.942	0.935	0.938
coffee	0.943	0.939	0.918	0.941	0.949
helmet	0.946	0.934	0.875	0.943	0.927
train	-	-	-	-	-
gnome	-	-	-	-	-
LPIPS↓					
	Adams Place Bridge	Courtyard	Shady Patch	Hotel Room	Moonless Golf
drums	0.031	0.039	0.038	0.051	0.033
coffee	0.141	0.132	0.163	0.133	0.099
helmet	0.075	0.089	0.097	0.096	0.056
train	-	-	-	-	-
gnome	-	-	-	-	-

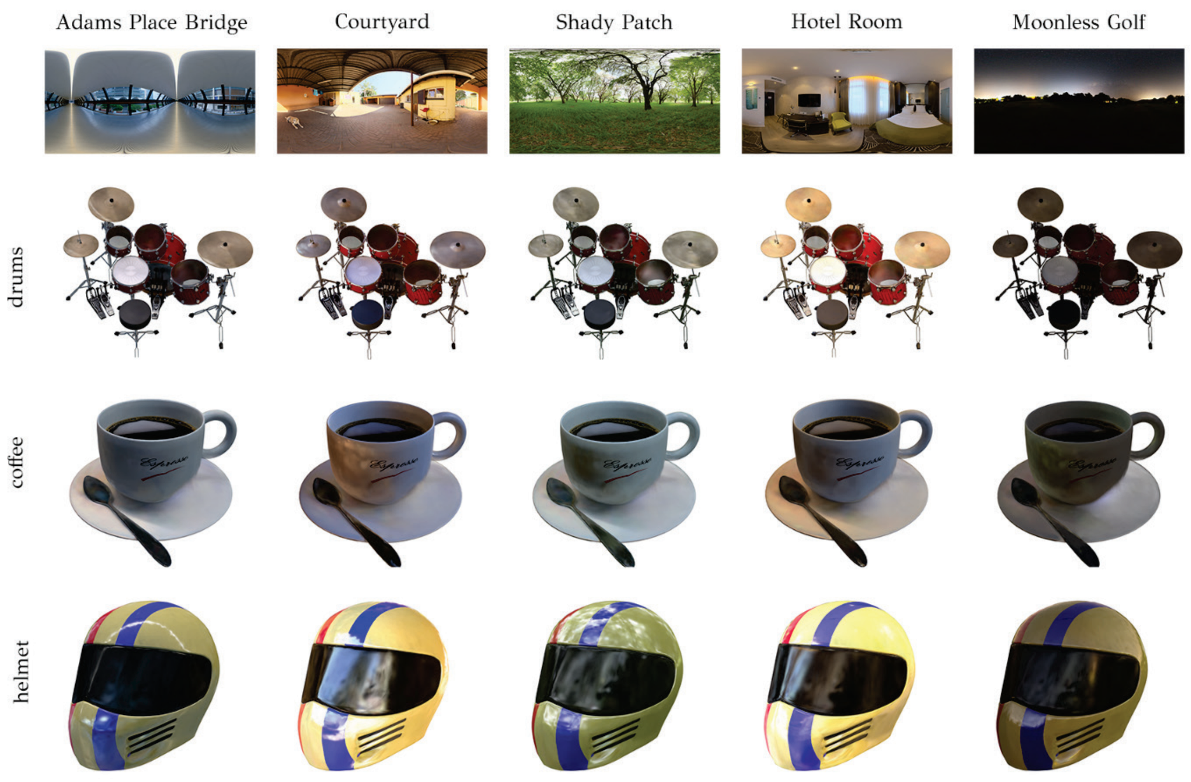

Figure 3.10 Qualitative results in relighting with ReCap

As already mentioned, NVS and relighting tasks proved unsuccessful for real-world scenes.

3.7 Comparison

Having recorded and presented both quantitative and qualitative results for each implementation, it's time to see how they stack up against each other. First, we compile a table with their general characteristics. We include the number of training iterations, the median size on disk of the resulting reconstruction and relighting, the ability of the implementation to handle diverse scenes (synthetic, real captures), the ability of the implementation to relight using unknown environment maps, the requirement of ground truth images for relighting (i.e., if our dataset must contain test views for every environment map) and finally, if the implementation is using ray tracing for global illumination.

Table 3.11 Characteristics comparison between examined implementations.

	Gaussian Shader	R3DG	Reflective Gaussian	IRGS	ReCap
Training Iterations	30.000	40.000	58.000	60.000	30.000
Median result size on disk	1.4GB	9.4GB	1.5GB	2.8GB	603,6MB
Scene Generalization	✓	✓	✓	✓	✓
Env map generalization	✓	✓	✓	✓	✓
Requires GT for relight	×	✓	×	✓	✓
Ray tracing	×	✓	✓	✓	×

Following we present quantitative and qualitative comparisons on NVS and relighting among the examined implementations.

3.7.1 Novel View Synthesis

Table 3.12 Quantitative comparison in Novel View Synthesis. Highlighted are the best and second best scores for each dataset.

Novel View Synthesis					
PSNR↑					
	Gaussian Shader	Relightable 3DGS	Reflective Gaussian	IRGS	ReCap
drums	25.65	25.19	26.38	27.15	29.63
coffee	24.39	30.09	34.30	26.57	31.46
helmet	19.188	26.29	29.80	17.75	29.49
train	20.23	-	20.57	15.82	-
gnome	14.47	40.08	16.02	17.11	-
SSIM↑					
	Gaussian Shader	Relightable 3DGS	Reflective Gaussian	IRGS	ReCap
drums	0.945	0.948	0.952	0.953	0.967
coffee	0.949	0.957	0.975	0.956	0.962
helmet	0.895	0.955	0.965	0.904	0.963
train	0.764	-	0.751	0.516	-
gnome	0.421	0.988	0.571	0.610	-
LPIPS↓					
	Gaussian Shader	Relightable 3DGS	Reflective Gaussian	IRGS	ReCap
drums	0.044	0.049	0.042	0.044	0.028
coffee	0.124	0.095	0.077	0.104	0.127
helmet	0.132	0.102	0.060	0.132	0.069
train	0.271	-	0.305	0.468	-
gnome	0.523	0.014	0.505	0.488	-

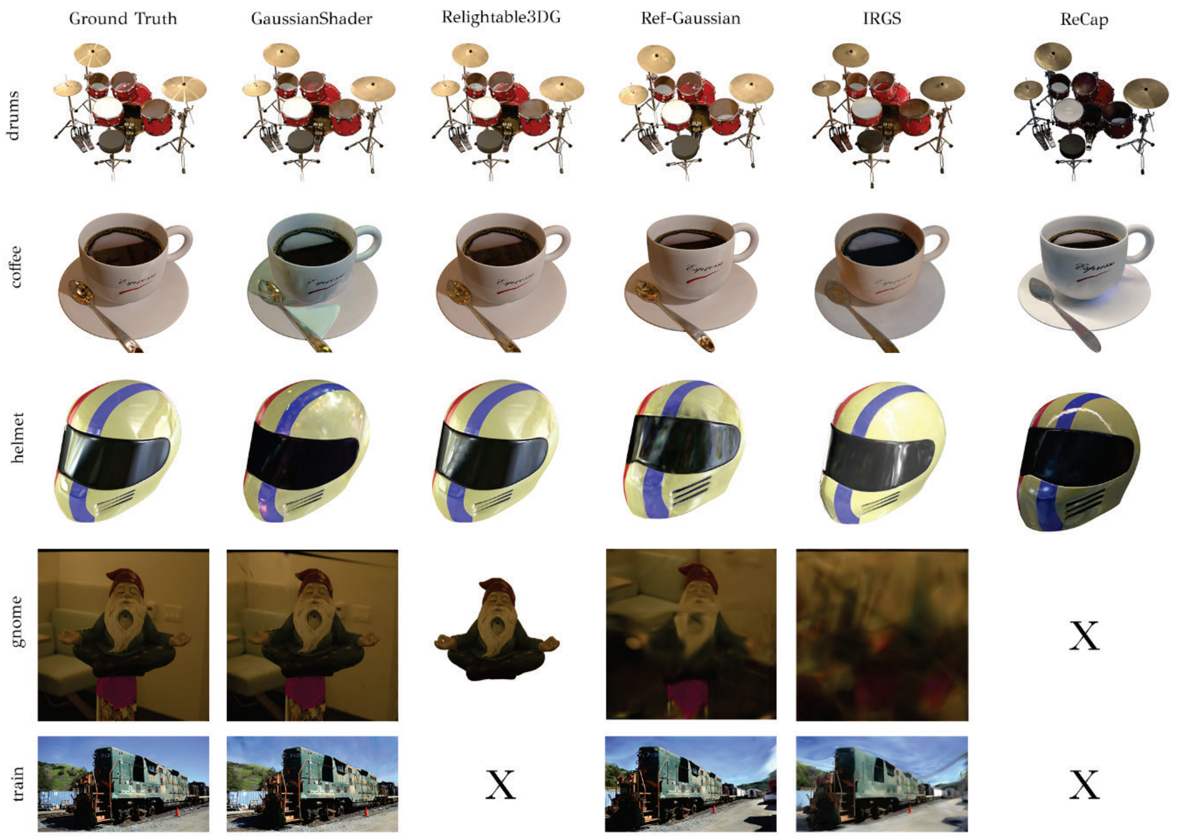

Figure 3.11 Qualitative comparison in Novel View Synthesis

3.7.2 Relighting

Table 3.13 Quantitative comparison in relighting drums dataset. Highlighted are the best and second best scores for each environment map.

Relighting Drums					
PSNR↑					
	Gaussian Shader	R3DG	Ref-Gaussian	IRGS	ReCap
Adams Place Bridge	19.59	20.17	25.44	17.85	26.37
Courtyard	16.18	17.86	25.01	15.80	24.37
Shady Patch	19.25	19.85	24.89	17.71	24.46
Hotel Room	14.93	17.07	24.71	15.17	23.39
Moonless Golf	21.34	22.80	25.39	17.76	27.19
SSIM↑					
	Gaussian Shader	R3DG	Ref-Gaussian	IRGS	ReCap
Adams Place Bridge	0.888	0.809	0.947	0.797	0.957
Courtyard	0.853	0.802	0.943	0.786	0.949
Shady Patch	0.882	0.807	0.942	0.792	0.942
Hotel Room	0.843	0.801	0.941	0.789	0.935
Moonless Golf	0.914	0.811	0.948	0.794	0.938
LPIPS↓					
	Gaussian Shader	R3DG	Ref-Gaussian	IRGS	ReCap
Adams Place Bridge	0.076	0.183	0.045	0.196	0.031
Courtyard	0.106	0.192	0.047	0.205	0.039
Shady Patch	0.079	0.187	0.049	0.202	0.038
Hotel Room	0.124	0.195	0.048	0.210	0.051
Moonless Golf	0.062	0.176	0.044	0.199	0.033

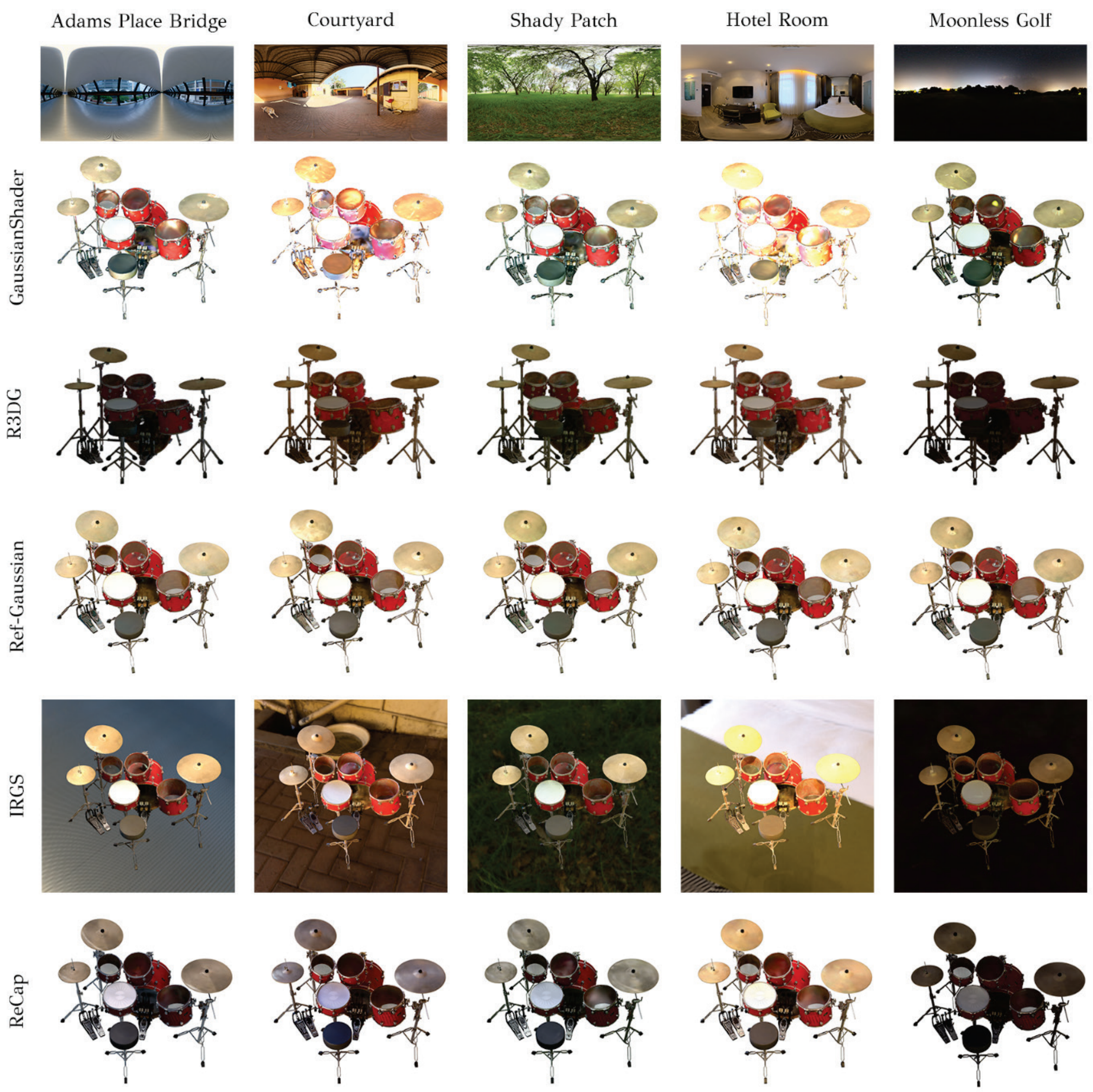

Figure 3.12 Qualitative comparison in relighting the drums dataset

Table 3.14 Quantitative comparison in relighting coffee dataset. Highlighted are the best and second best scores for each environment map.

Relighting Coffee					
PSNR↑					
	Gaussian Shader	R3DG	Ref-Gaussian	IRGS	ReCap
Adams Place Bridge	15.79	10.97	29.62	12.78	21.59
Courtyard	12.92	13.26	29.16	13.41	22.57
Shady Patch	15.84	11.77	28.75	13.20	20.03
Hotel Room	12.22	14.07	28.06	12.78	21.81
Moonless Golf	24.39	17.31	30.41	17.72	23.05
SSIM↑					
	Gaussian Shader	R3DG	Ref-Gaussian	IRGS	ReCap
Adams Place Bridge	0.893	0.755	0.964	0.766	0.943
Courtyard	0.869	0.766	0.964	0.761	0.939
Shady Patch	0.892	0.753	0.962	0.767	0.918
Hotel Room	0.864	0.797	0.960	0.768	0.941
Moonless Golf	0.949	0.807	0.968	0.813	0.949
LPIPS↓					
	Gaussian Shader	R3DG	Ref-Gaussian	IRGS	ReCap
Adams Place Bridge	0.157	0.269	0.087	0.259	0.141
Courtyard	0.191	0.253	0.088	0.265	0.132
Shady Patch	0.170	0.277	0.092	0.259	0.163
Hotel Room	0.201	0.249	0.091	0.284	0.133
Moonless Golf	0.124	0.209	0.084	0.220	0.099

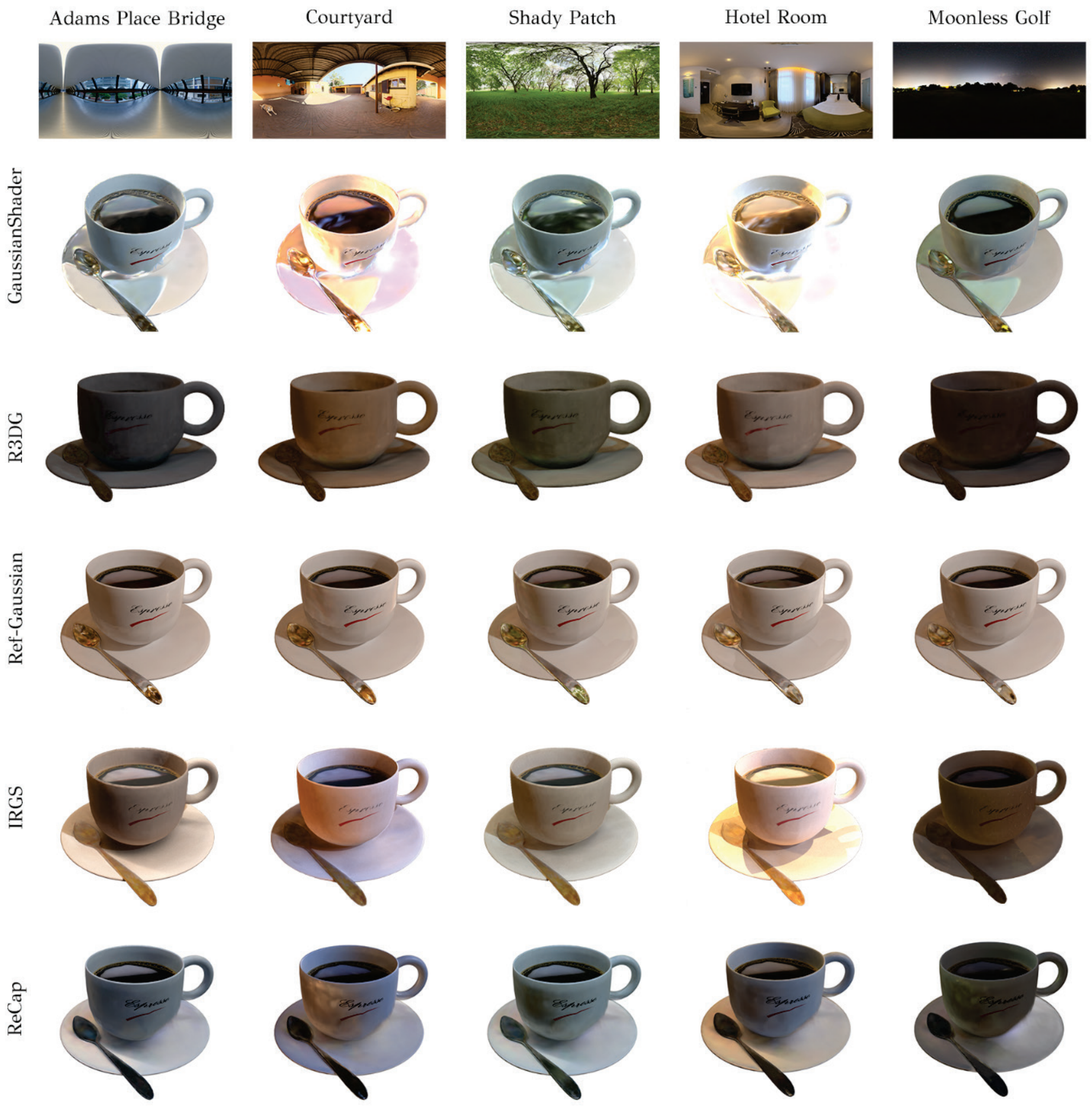

Figure 3.13 Qualitative comparison in relighting coffee dataset

Table 3.15 Quantitative comparison in relighting helmet dataset. Highlighted are the best and second best scores for each environment map.

Relighting Helmet					
PSNR↑					
	Gaussian Shader	R3DG	Ref-Gaussian	IRGS	ReCap
Adams Place Bridge	19.18	15.07	23.15	14.25	23.18
Courtyard	15.44	12.91	21.02	11.86	22.06
Shady Patch	15.51	14.84	21.08	14.05	22.16
Hotel Room	13.96	12.12	20.84	12.31	22.58
Moonless Golf	19.18	19.46	22.44	14.08	25.51
SSIM↑					
	Gaussian Shader	R3DG	Ref-Gaussian	IRGS	ReCap
Adams Place Bridge	0.895	0.789	0.927	0.785	0.946
Courtyard	0.860	0.774	0.917	0.752	0.934
Shady Patch	0.851	0.743	0.904	0.729	0.875
Hotel Room	0.856	0.788	0.912	0.786	0.943
Moonless Golf	0.895	0.803	0.930	0.769	0.927
LPIPS↓					
	Gaussian Shader	R3DG	Ref-Gaussian	IRGS	ReCap
Adams Place Bridge	0.132	0.272	0.112	0.333	0.075
Courtyard	0.210	0.285	0.127	0.344	0.089
Shady Patch	0.180	0.280	0.131	0.336	0.097
Hotel Room	0.216	0.306	0.124	0.333	0.096
Moonless Golf	0.132	0.238	0.107	0.331	0.056

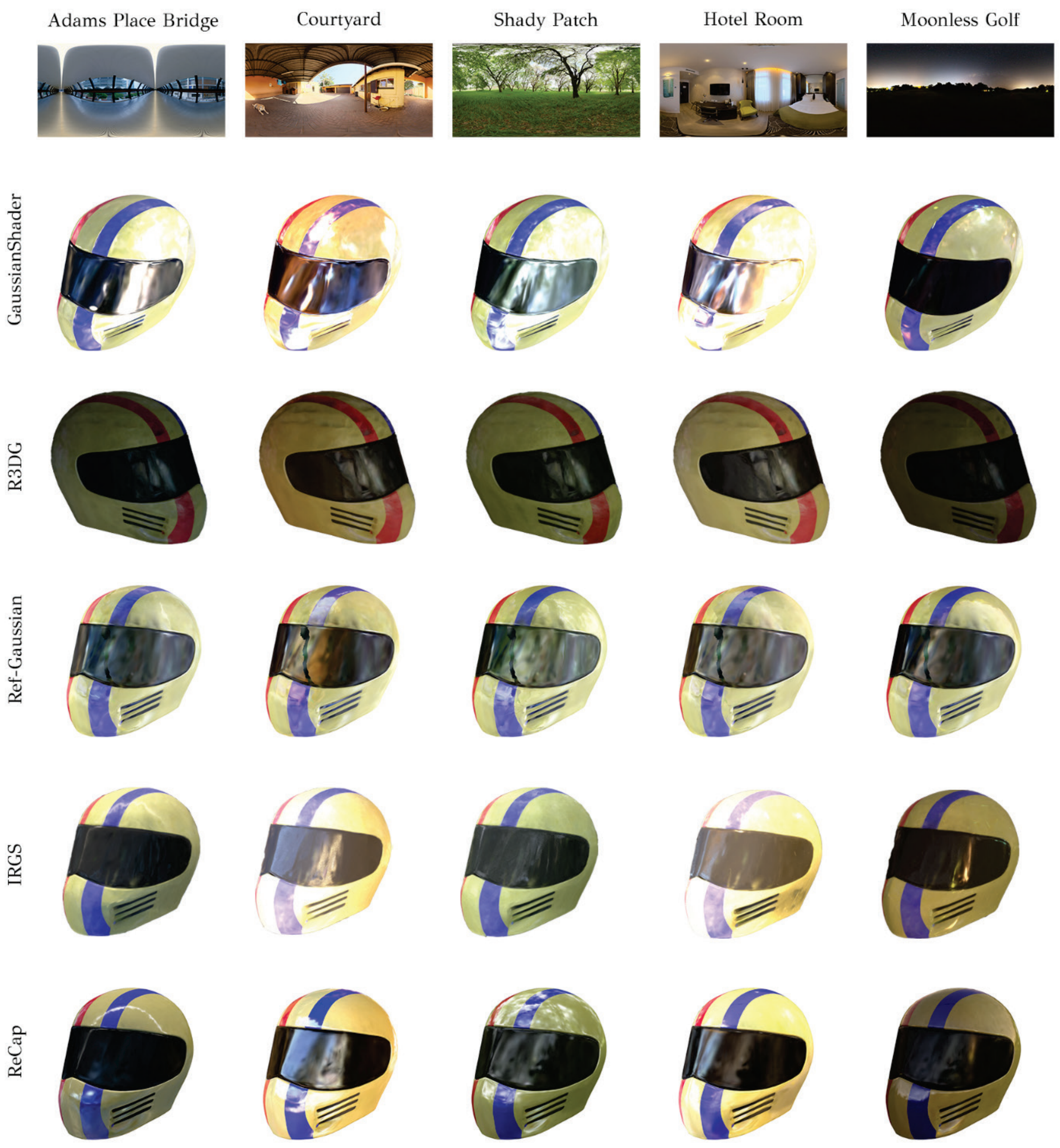

Figure 3.14 Qualitative comparison in relighting drums dataset

Table 3.16 Quantitative comparison in relighting gnome dataset. Highlighted are the best scores for each environment map.

Relighting Gnome					
PSNR↑					
	Gaussian Shader	R3DG	Ref-Gaussian	IRGS	ReCap
Adams Place Bridge	10.10	-	15.92	-	-
Courtyard	6.02	-	15.77	-	-
Shady Patch	9.93	-	15.91	-	-
Hotel Room	5.036	-	15.75	-	--
Moonless Golf	13.45	-	15.94	-	
SSIM↑					
	Gaussian Shader	R3DG	Ref-Gaussian	IRGS	ReCap
Adams Place Bridge	0.379	-	0.555	-	-
Courtyard	0.288	-	0.553	-	-
Shady Patch	0.356	-	0.541	-	-
Hotel Room	0.273	-	0.561	-	-
Moonless Golf	0.444	-	0.559	-	-
LPIPS↓					
	Gaussian Shader	R3DG	Ref-Gaussian	IRGS	ReCap
Adams Place Bridge	0.613	-	0.513	-	-
Courtyard	0.720	-	0.517	-	-
Shady Patch	0.621	-	0.524	-	-
Hotel Room	0.717	-	0.517	-	-
Moonless Golf	0.568	-	0.501	-	-

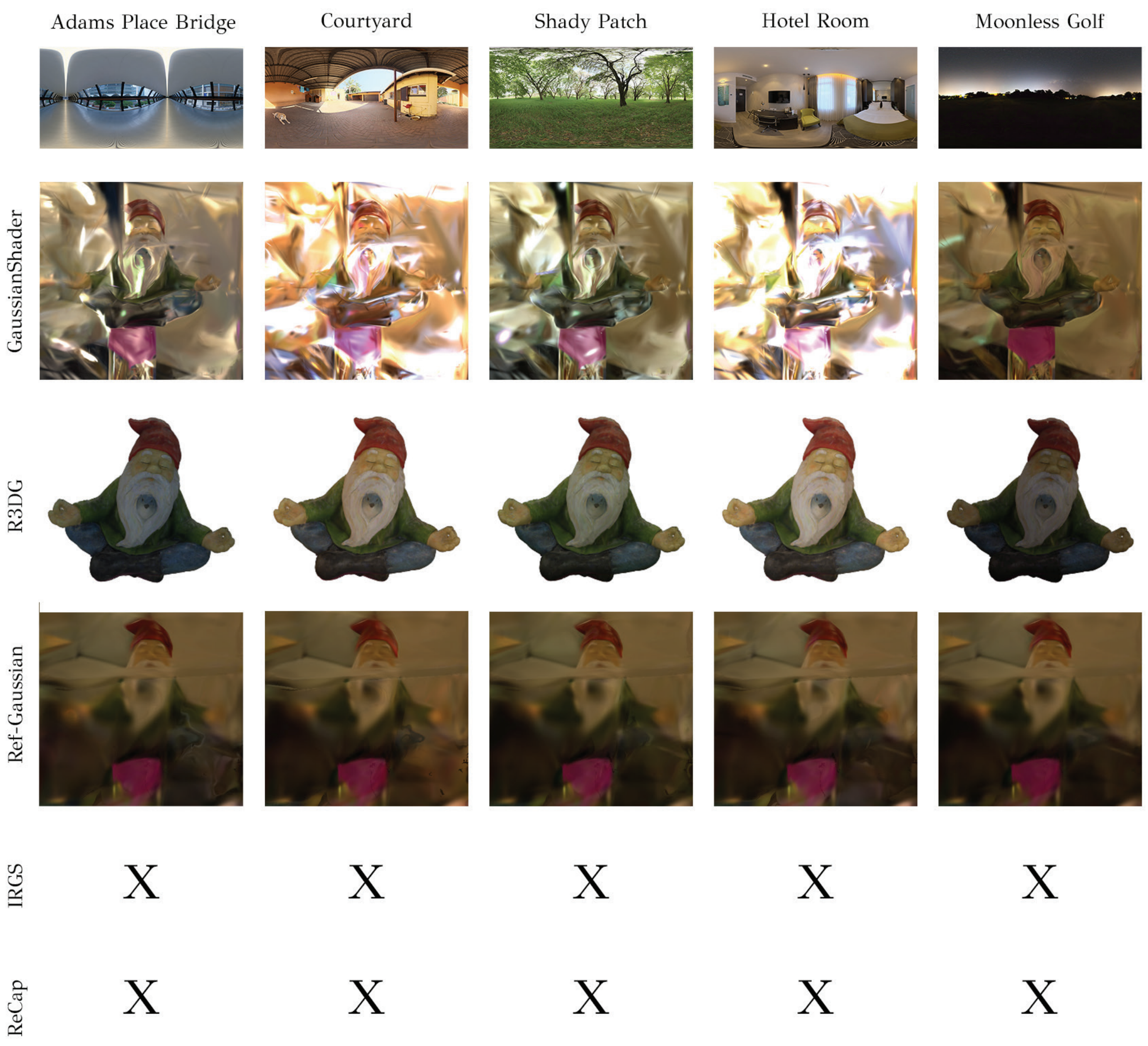

Figure 3.15 Qualitative comparison in relighting gnome dataset

Table 3.17 Quantitative comparison in relighting train dataset. Highlighted are the best scores for each environment map.

Relighting Train					
PSNR↑					
	Gaussian Shader	R3DG	Ref-Gaussian	IRGS	ReCap
Adams Place Bridge	13.29	-	20.22	-	-
Courtyard	8.22	-	19.51	-	-
Shady Patch	12.83	-	19.92	-	-
Hotel Room	7.36	-	19.81	-	-
Moonless Golf	16.76	-	19.98	-	-
SSIM↑					
	Gaussian Shader	R3DG	Ref-Gaussian	IRGS	ReCap
Adams Place Bridge	0.595	-	0.744	-	-
Courtyard	0.437	-	0.728	-	-
Shady Patch	0.557	-	0.738	-	-
Hotel Room	0.423	-	0.735	-	-
Moonless Golf	0.668	-	0.741	-	-
LPIPS↓					
	Gaussian Shader	R3DG	Ref-Gaussian	IRGS	ReCap
Adams Place Bridge	0.377	-	0.311	-	-
Courtyard	0.530	-	0.321	-	-
Shady Patch	0.403	-	0.317	-	-
Hotel Room	0.546	-	0.316	-	-
Moonless Golf	0.342	-	0.312	-	-

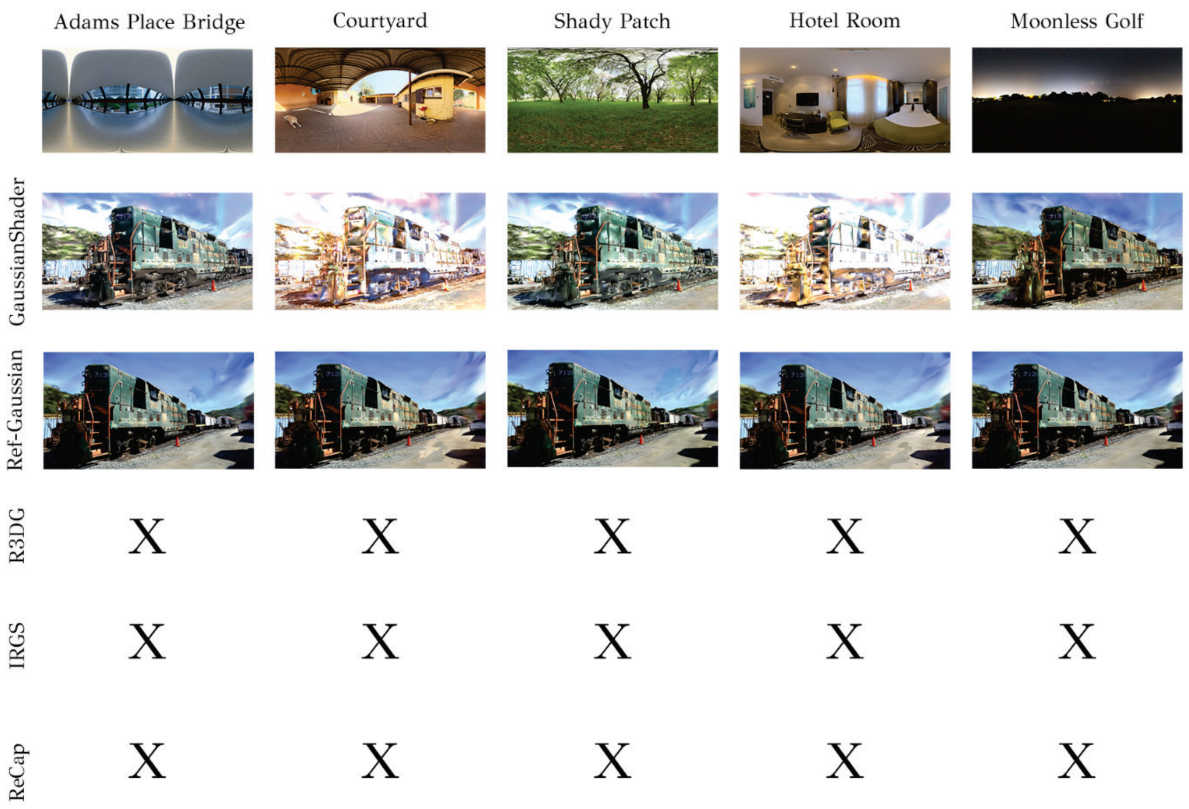

Figure 3.16 Qualitative comparison in relighting train dataset

References

[1] T. Shen, J. Gao, K. Yin, M.-Y. Liu, and S. Fidler, "Deep marching tetrahedra: a hybrid representation for high-resolution 3D shape synthesis," in *Proceedings of the 35th International Conference on Neural Information Processing Systems*, in NIPS '21. Red Hook, NY, USA: Curran Associates Inc., Dec. 2021, pp. 6087–6101.

[2] E. R. Chan *et al.*, "Efficient Geometry-aware 3D Generative Adversarial Networks," *Proc. IEEECVF Conf. Comput. Vis. Pattern Recognit.*, pp. 16123–16133, 2022.

[3] Z. Wang, "3D Representation Methods: A Survey," Oct. 09, 2024, *arXiv*: arXiv:2410.06475. doi: 10.48550/arXiv.2410.06475.

[4] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," Apr. 12, 2017, *arXiv*: arXiv:1612.00593. doi: 10.48550/arXiv.1612.00593.

[5] D. Cavagnino and M. Gribaudo, "Discretization of 3D models using voxel elements of different shapes," in *Proceedings of the Sixth international conference on Computational Aesthetics in Graphics, Visualization and Imaging*, in Computational Aesthetics'10. Goslar, DEU: Eurographics Association, Jun. 2010, pp. 91–98.

[6] P. Schwaha, R. Heinzl, T. E. Simos, G. Psihoyios, and Ch. Tsitouras, "Marching Simplices," presented at the ICNAAM 2010: International Conference of Numerical Analysis and Applied Mathematics 2010, Rhodes (Greece), 2010, pp. 1651–1654. doi: 10.1063/1.3498149.

[7] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," *SIGGRAPH Comput Graph*, vol. 21, no. 4, pp. 163–169, Aug. 1987, doi: 10.1145/37402.37422.

[8] D. Adalsteinsson and J. A. Sethian, "A Fast Level Set Method for Propagating Interfaces," *J Comput Phys*, vol. 118, no. 2, pp. 269–277, May 1995, doi: 10.1006/jcph.1995.1098.

[9] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, "DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation," Jan. 16, 2019, *arXiv*: arXiv:1901.05103. doi: 10.48550/arXiv.1901.05103.

[10] J. Mu, W. Qiu, A. Kortylewski, A. Yuille, N. Vasconcelos, and X. Wang, "A-SDF: Learning Disentangled Signed Distance Functions for Articulated Shape

Representation," Apr. 15, 2021, *arXiv*: arXiv:2104.07645. doi: 10.48550/arXiv.2104.07645.

[11] "AutoSDF: Shape Priors for 3D Completion, Reconstruction and Generation." Accessed: Jul. 03, 2025. [Online]. Available: https://www.computer.org/csdl/proceedings-article/cvpr/2022/694600a306/1H1lTkhvKyQ

[12] X.-Y. Zheng, Y. Liu, P.-S. Wang, and X. Tong, "SDF-StyleGAN: Implicit SDF-Based StyleGAN for 3D Shape Generation," Jun. 24, 2022, *arXiv*: arXiv:2206.12055. doi: 10.48550/arXiv.2206.12055.

[13] "Locally Attentional SDF Diffusion for Controllable 3D Shape Generation | ACM Transactions on Graphics." Accessed: Jul. 03, 2025. [Online]. Available: https://dl.acm.org/doi/10.1145/3592103

[14] H. Jun and A. Nichol, "Shap-E: Generating Conditional 3D Implicit Functions," May 03, 2023, *arXiv*: arXiv:2305.02463. doi: 10.48550/arXiv.2305.02463.

[15] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "NeRF: representing scenes as neural radiance fields for view synthesis," *Commun ACM*, vol. 65, no. 1, pp. 99–106, Dec. 2021, doi: 10.1145/3503250.

[16] L. Zhou, G. Wu, Y. Zuo, X. Chen, and H. Hu, "A Comprehensive Review of Vision-Based 3D Reconstruction Methods," *Sensors*, vol. 24, no. 7, Art. no. 7, Jan. 2024, doi: 10.3390/s24072314.

[17] A. Flisch *et al.*, "Industrial Computed Tomography in Reverse Engineering Application." Accessed: Jul. 05, 2025. [Online]. Available: https://www.researchgate.net/publication/200018530_Industrial_Computed_Tomography_in_Reverse_Engineering_Application

[18] C. Rocchini, P. Cignoni, C. Montani, P. Pingi, and R. Scopigno, "A low cost 3D scanner based on structured light," *Comput. Graph. Forum*, vol. 20, no. 3, pp. 299–308, Sep. 2001, doi: 10.1111/1467-8659.00522.

[19] J. Park, H. Kim, T. Yu-Wing Tai, M. S. Brown, and I. Kweon, "High quality depth map upsampling for 3D-TOF cameras: 2011 IEEE International Conference on Computer Vision, ICCV 2011," *2011 Int. Conf. Comput. Vis. ICCV 2011*, pp. 1623–1630, 2011, doi: 10.1109/ICCV.2011.6126423.

[20] B. Schwarz, "Mapping the world in 3D," *Nat. Photonics*, vol. 4, no. 7, pp. 429–430, Jul. 2010, doi: 10.1038/nphoton.2010.148.

[21] K. Kraus and N. Pfeifer, "Determination of terrain models in wooded areas with airborne laser scanner data," *ISPRS J. Photogramm. Remote Sens.*, vol. 53, no. 4, pp. 193–203, Aug. 1998, doi: 10.1016/S0924-2716(98)00009-4.

[22] N. Matthews, X. Meng, P. Xu, and N. Qian, "A physiological theory of depth perception from vertical disparity," *Vision Res.*, vol. 43, no. 1, pp. 85–99, Jan. 2003, doi: 10.1016/S0042-6989(02)00401-7.

[23] S. K. Karmacharya, N. Ruther, U. Shrestha, and M. B. Bishwakarma, "Evaluating the Structure from Motion Technique for Measurement of Bed Morphology in Physical Model Studies," *Water*, vol. 13, no. 7, Art. no. 7, Jan. 2021, doi: 10.3390/w13070998.

[24] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski, "A Comparison and Evaluation of Multi-View Stereo Reconstruction Algorithms," in *2006 IEEE*

Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06), Jun. 2006, pp. 519–528. doi: 10.1109/CVPR.2006.19.

[25] Y. Huang, W. Zheng, Y. Zhang, J. Zhou, and J. Lu, "Tri-Perspective View for Vision-Based 3D Semantic Occupancy Prediction," Mar. 02, 2023, *arXiv*: arXiv:2302.07817. doi: 10.48550/arXiv.2302.07817.

[26] B. Kerbl, G. Kopanas, T. Leimkuehler, and G. Drettakis, "3D Gaussian Splatting for Real-Time Radiance Field Rendering," *ACM Trans Graph*, vol. 42, no. 4, p. 139:1-139:14, Jul. 2023, doi: 10.1145/3592433.

[27] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant Neural Graphics Primitives with a Multiresolution Hash Encoding," *ACM Trans. Graph.*, vol. 41, no. 4, pp. 1–15, Jul. 2022, doi: 10.1145/3528223.3530127.

[28] A. Pumarola, E. Corona, G. Pons-Moll, and F. Moreno-Noguer, "D-NeRF: Neural Radiance Fields for Dynamic Scenes," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2021, pp. 10313–10322. doi: 10.1109/CVPR46437.2021.01018.

[29] K. Park *et al.*, "Nerfies: Deformable Neural Radiance Fields," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2021, pp. 5845–5854. doi: 10.1109/ICCV48922.2021.00581.

[30] S. Izadi *et al.*, "KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera," in *Proceedings of the 24th annual ACM symposium on User interface software and technology*, in UIST '11. New York, NY, USA: Association for Computing Machinery, Oct. 2011, pp. 559–568. doi: 10.1145/2047196.2047270.

[31] M. Zwicker, H. Pfister, J. Van Baar, and M. Gross, "EWA volume splatting," in *Proceedings Visualization, 2001. VIS '01.*, San Diego, CA, USA: IEEE, 2001, pp. 29–538. doi: 10.1109/VISUAL.2001.964490.

[32] G. Chen and W. Wang, "A Survey on 3D Gaussian Splatting," Mar. 07, 2025, *arXiv*: arXiv:2401.03890. doi: 10.48550/arXiv.2401.03890.

[33] J. Li *et al.*, "DNGaussian: Optimizing Sparse-View 3D Gaussian Radiance Fields with Global-Local Depth Normalization," Mar. 24, 2024, *arXiv*: arXiv:2403.06912. doi: 10.48550/arXiv.2403.06912.

[34] Y. Chen *et al.*, "MVSplat: Efficient 3D Gaussian Splatting from Sparse Multi-View Images," vol. 15079, 2025, pp. 370–386. doi: 10.1007/978-3-031-72664-4_21.

[35] D. Charatan, S. Li, A. Tagliasacchi, and V. Sitzmann, "pixelSplat: 3D Gaussian Splats from Image Pairs for Scalable Generalizable 3D Reconstruction," Apr. 04, 2024, *arXiv*: arXiv:2312.12337. doi: 10.48550/arXiv.2312.12337.

[36] S. Szymanowicz, C. Rupprecht, and A. Vedaldi, "Splatter Image: Ultra-Fast Single-View 3D Reconstruction," Apr. 16, 2024, *arXiv*: arXiv:2312.13150. doi: 10.48550/arXiv.2312.13150.

[37] Z. Fan, K. Wang, K. Wen, Z. Zhu, D. Xu, and Z. Wang, "LightGaussian: Unbounded 3D Gaussian Compression with 15x Reduction and 200+ FPS," Nov. 12, 2024, *arXiv*: arXiv:2311.17245. doi: 10.48550/arXiv.2311.17245.

[38] Z. Yu, A. Chen, B. Huang, T. Sattler, and A. Geiger, "Mip-Splatting: Alias-Free 3D Gaussian Splatting," in *2024 IEEE/CVF Conference on Computer Vision*

and Pattern Recognition (CVPR), Jun. 2024, pp. 19447–19456. doi: 10.1109/CVPR52733.2024.01839.

[39] Z. Yang *et al.*, "Spec-Gaussian: anisotropic view-dependent appearance for 3D Gaussian splatting," in *Proceedings of the 38th International Conference on Neural Information Processing Systems*, in NIPS '24, vol. 37. Red Hook, NY, USA: Curran Associates Inc., Jun. 2025, pp. 61192–61216.

[40] Z. Liang, Q. Zhang, W. Hu, L. Zhu, Y. Feng, and K. Jia, "Analytic-Splatting: Anti-Aliased 3D Gaussian Splatting via Analytic Integration," in *Computer Vision – ECCV 2024*, A. Leonardis, E. Ricci, S. Roth, O. Russakovsky, T. Sattler, and G. Varol, Eds., Cham: Springer Nature Switzerland, 2025, pp. 281–297. doi: 10.1007/978-3-031-72643-9_17.

[41] "StopThePop: Sorted Gaussian Splatting for View-Consistent Real-time Rendering | ACM Transactions on Graphics." Accessed: Jul. 05, 2025. [Online]. Available: https://dl.acm.org/doi/10.1145/3658187

[42] Y. Fu, X. Wang, S. Liu, A. Kulkarni, J. Kautz, and A. A. Efros, "COLMAP-Free 3D Gaussian Splatting," in *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2024, pp. 20796–20805. doi: 10.1109/CVPR52733.2024.01965.

[43] J. Jung, J. Han, H. An, J. Kang, S. Park, and S. Kim, "Relaxing Accurate Initialization Constraint for 3D Gaussian Splatting," May 28, 2024, *arXiv*: arXiv:2403.09413. doi: 10.48550/arXiv.2403.09413.

[44] M. Yu, T. Lu, L. Xu, L. Jiang, Y. Xiangli, and B. Dai, "GSDF: 3DGS Meets SDF for Improved Neural Rendering and Reconstruction," *Adv. Neural Inf. Process. Syst.*, vol. 37, pp. 129507–129530, Dec. 2024.

[45] J. Zhang, F. Zhan, M. Xu, S. Lu, and E. Xing, "FreGS: 3D Gaussian Splatting with Progressive Frequency Regularization," presented at the 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society, Jun. 2024, pp. 21424–21433. doi: 10.1109/CVPR52733.2024.02024.

[46] L. Huang, J. Bai, J. Guo, and Y. Guo, "GS++: Error Analyzing and Optimal Gaussian Splatting," Feb. 01, 2024, *arXiv*: arXiv:2402.00752. doi: 10.48550/arXiv.2402.00752.

[47] J.-C. Shi, M. Wang, H.-B. Duan, and S.-H. Guan, "Language Embedded 3D Gaussians for Open-Vocabulary Scene Understanding," Nov. 30, 2023, *arXiv*: arXiv:2311.18482. doi: 10.48550/arXiv.2311.18482.

[48] "LangSplat: 3D Language Gaussian Splatting." Accessed: Jul. 05, 2025. [Online]. Available: https://www.computer.org/csdl/proceedings-article/cvpr/2024/530000u051/20hLO2hrcR2

[49] X. Zuo, P. Samangouei, Y. Zhou, Y. Di, and M. Li, "FMGS: Foundation Model Embedded 3D Gaussian Splatting for Holistic 3D Scene Understanding," *Int. J. Comput. Vis.*, vol. 133, no. 2, pp. 611–627, Feb. 2025, doi: 10.1007/s11263-024-02183-8.

[50] S. Zhou *et al.*, "Feature 3DGS: Supercharging 3D Gaussian Splatting to Enable Distilled Feature Fields," in *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2024, pp. 21676–21685. doi: 10.1109/CVPR52733.2024.02048.

[51] T. Lu *et al.*, “Scaffold-GS: Structured 3D Gaussians for View-Adaptive Rendering,” in *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2024, pp. 20654–20664. doi: 10.1109/CVPR52733.2024.01952.

[52] “Relightable Gaussian Codec Avatars.” Accessed: Jul. 05, 2025. [Online]. Available: https://www.computer.org/csdl/proceedings-article/cvpr/2024/530000a130/20hRV5btKXm

[53] T. Zhang, K. Huang, W. Zhi, and M. Johnson-Roberson, “DarkGS: Learning Neural Illumination and 3D Gaussians Relighting for Robotic Exploration in the Dark,” in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2024, pp. 12864–12871. doi: 10.1109/IROS58592.2024.10802684.

[54] N. Moenne-Loccoz *et al.*, “3D Gaussian Ray Tracing: Fast Tracing of Particle Scenes,” Oct. 10, 2024, *arXiv*: arXiv:2407.07090. doi: 10.48550/arXiv.2407.07090.

[55] A. Mai *et al.*, “EVER: Exact Volumetric Ellipsoid Rendering for Real-time View Synthesis,” Oct. 29, 2024, *arXiv*: arXiv:2410.01804. doi: 10.48550/arXiv.2410.01804.

[56] R. L. Cook, “Shade trees,” *SIGGRAPH Comput Graph*, vol. 18, no. 3, pp. 223–231, Jan. 1984, doi: 10.1145/964965.808602.

[57] C. Schlick, “A Survey of Shading and Reflectance Models,” *Comput. Graph. Forum*, vol. 13, no. 2, pp. 121–131, 1994, doi: 10.1111/1467-8659.1320121.

[58] M. Deering, S. Winner, B. Schediwy, C. Duffy, and N. Hunt, “The triangle processor and normal vector shader: a VLSI system for high performance graphics,” *SIGGRAPH Comput Graph*, vol. 22, no. 4, pp. 21–30, Jun. 1988, doi: 10.1145/378456.378468.

[59] T. Saito and T. Takahashi, “Comprehensible rendering of 3-D shapes,” *SIGGRAPH Comput Graph*, vol. 24, no. 4, pp. 197–206, Sep. 1990, doi: 10.1145/97880.97901.

[60] O. Olsson, M. Billeter, and U. Assarsson, “Clustered Deferred and Forward Shading,” *High Perform. Graph.*, pp. 1–10, 2012.

[61] “Deferred lighting approaches | Real-Time Rendering.” Accessed: Jun. 22, 2025. [Online]. Available: https://www.realtimerendering.com/blog/deferred-lighting-approaches/

[62] A. De Pereyra, “MLAA: Efficiently Moving Antialiasing from the GPU to the CPU.” Accessed: Jun. 22, 2025. [Online]. Available: https://gamedev.net/tutorials/programming/graphics/mlaa-efficiently-moving-antialiasing-from-the-gpu-to-the-cpu-r2809

[63] T. Lottes, “Fast Approximate Anti-Aliasing (FXAA)”.

[64] R. Gao and Y. Qi, “A Brief Review on Differentiable Rendering: Recent Advances and Challenges,” *Electronics*, vol. 13, no. 17, Art. no. 17, Jan. 2024, doi: 10.3390/electronics13173546.

[65] J. T. Kajiya, “The rendering equation,” *SIGGRAPH Comput Graph*, vol. 20, no. 4, pp. 143–150, Aug. 1986, doi: 10.1145/15886.15902.

[66] S. J. Koppal, “Lambertian Reflectance,” in *Computer Vision*, Springer, Cham, 2021, pp. 729–731. doi: 10.1007/978-3-030-63416-2_534.
[67] B. T. Phong, “Illumination for computer generated pictures,” *Commun ACM*, vol. 18, no. 6, pp. 311–317, Jun. 1975, doi: 10.1145/360825.360839.
[68] R. L. Cook and K. E. Torrance, “A Reflectance Model for Computer Graphics,” *ACM Trans Graph*, vol. 1, no. 1, pp. 7–24, Jan. 1982, doi: 10.1145/357290.357293.
[69] S. Saikia, “Deriving Lambertian BRDF from first principles.” Accessed: Jun. 01, 2025. [Online]. Available: https://sakibsaikia.github.io/graphics/2019/09/10/Deriving-Lambertian-BRDF-From-First-Principles.html
[70] J. F. Blinn, “Models of light reflection for computer synthesized pictures,” *SIGGRAPH Comput Graph*, vol. 11, no. 2, pp. 192–198, Jul. 1977, doi: 10.1145/965141.563893.
[71] M. Ashikhmin and P. Shirley, “An Anisotropic Phong BRDF Model,” *ResearchGate*, Aug. 2000, Accessed: Jun. 02, 2025. [Online]. Available: https://www.researchgate.net/publication/2523875_An_anisotropic_phong_BRDF_model
[72] J. Lawrence, “Importance Sampling of the Phong Reflectance Model,” 2008.
[73] B. Walter, S. R. Marschner, H. Li, and K. E. Torrance, “Microfacet Models for Refraction through Rough Surfaces”.
[74] J. Boksansky, “Crash Course in BRDF Implementation”.
[75] C. Schlick, “An Inexpensive BRDF Model for Physically-based Rendering,” *Comput. Graph. Forum*, vol. 13, no. 3, pp. 233–246, 1994, doi: 10.1111/1467-8659.1330233.
[76] B. Walter, S. R. Marschner, H. Li, and K. E. Torrance, “Microfacet Models for Refraction through Rough Surfaces”.
[77] B. Burley, “Physically Based Shading at Disney”.
[78] B. T. Phong, “Illumination for Computer Generated Pictures,” vol. 18, no. 6, 1975.
[79] F. L. Pedrotti and L. S. Pedrotti, *Introduction to optics*. Englewood Cliffs, N.J. : Prentice Hall, 1993. Accessed: Jun. 02, 2025. [Online]. Available: http://archive.org/details/introductiontoop00pedr
[80] M. F. Tappen, “Image Decomposition: Traditional Approaches,” in *Computer Vision*, Springer, Cham, 2021, pp. 605–607. doi: 10.1007/978-3-030-63416-2_549.
[81] D. Ulucan, O. Ulucan, and M. Ebner, “Challenges and Applications of Intrinsic Image Decomposition: A Short Review,” *SN Comput. Sci.*, vol. 6, no. 2, p. 125, Jan. 2025, doi: 10.1007/s42979-025-03659-1.
[82] D. Frerichs, A. Vidler, and C. Gatzidis, “A survey on object deformation and decomposition in computer graphics,” *Comput. Graph.*, vol. 52, pp. 18–32, Nov. 2015, doi: 10.1016/j.cag.2015.06.004.
[83] D. Ulucan, O. Ulucan, and M. Ebner, “IID-NORD: A Comprehensive Intrinsic Image Decomposition Dataset,” in *2022 IEEE International Conference on Image Processing (ICIP)*, Bordeaux, France: IEEE, Oct. 2022, pp. 2831–2835. doi: 10.1109/ICIP46576.2022.9897456.

[84] E. H. Land, “The Retinex,” *Am. Sci.*, vol. 52, no. 2, pp. 247–264, 1964.

[85] X. Xing, K. Groh, S. Karaoglu, and T. Gevers, “Intrinsic Image Decomposition Using Point Cloud Representation,” Mar. 28, 2024, *arXiv*: arXiv:2307.10924. doi: 10.48550/arXiv.2307.10924.

[86] C. Careaga and Y. Aksoy, “Intrinsic Image Decomposition via Ordinal Shading,” *ACM Trans Graph*, vol. 43, no. 1, p. 12:1-12:24, Nov. 2023, doi: 10.1145/3630750.

[87] P. Dutré, K. Bala, and P. Bekaert, *Advanced global illumination*, 2nd ed. Wellesley, Mass: AK Peters, 2006.

[88] C. M. Goral, K. E. Torrance, D. P. Greenberg, and B. Battaile, “Modeling the interaction of light between diffuse surfaces,” *SIGGRAPH Comput Graph*, vol. 18, no. 3, pp. 213–222, Jan. 1984, doi: 10.1145/964965.808601.

[89] H. W. Jensen and N. J. Christensen, “Photon maps in bidirectional Monte Carlo ray tracing of complex objects,” *Comput. Graph.*, vol. 19, no. 2, pp. 215–224, Mar. 1995, doi: 10.1016/0097-8493(94)00145-O.

[90] S. E. Chen, H. E. Rushmeier, G. Miller, and D. Turner, “A progressive multi-pass method for global illumination,” *SIGGRAPH Comput Graph*, vol. 25, no. 4, pp. 165–174, Jul. 1991, doi: 10.1145/127719.122737.

[91] P.-P. Sloan, J. Kautz, and J. Snyder, “Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments,” *ACM Trans Graph*, vol. 21, no. 3, pp. 527–536, Jul. 2002, doi: 10.1145/566654.566612.

[92] G. Bishop, H. Fuchs, L. McMillan, and E. J. S. Zagier, “Frameless rendering: double buffering considered harmful,” in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, in SIGGRAPH ’94. New York, NY, USA: Association for Computing Machinery, Jul. 1994, pp. 175–176. doi: 10.1145/192161.192195.

[93] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen, “The lumigraph,” in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, in SIGGRAPH ’96. New York, NY, USA: Association for Computing Machinery, Aug. 1996, pp. 43–54. doi: 10.1145/237170.237200.

[94] M. H. Kalos and P. A. Whitlock, *Monte Carlo methods*. New York: J. Wiley & Sons, 1986.

[95] H. Ren, H. Qiu, F. He, and K. Leng, “A Survey on Image-Based Approaches of Synthesizing Objects,” in *2016 International Conference on Virtual Reality and Visualization (ICVRV)*, Sep. 2016, pp. 264–269. doi: 10.1109/ICVRV.2016.50.

[96] R. Green, “Spherical Harmonic Lighting: The Gritty Details”.

[97] “An improved illumination model for shaded display | Communications of the ACM.” Accessed: Jul. 06, 2025. [Online]. Available: https://dl.acm.org/doi/10.1145/358876.358882

[98] R. L. Cook, T. Porter, and L. Carpenter, “Distributed ray tracing,” *SIGGRAPH Comput Graph*, vol. 18, no. 3, pp. 137–145, Jan. 1984, doi: 10.1145/964965.808590.

[99] A. Rath, P. Grittmann, S. Herholz, P. Weier, and P. Slusallek, “EARS: efficiency-aware russian roulette and splitting,” *ACM Trans Graph*, vol. 41, no. 4, p. 81:1-81:14, Jul. 2022, doi: 10.1145/3528223.3530168.

[100] T. Kloek and H. K. van Dijk, “Bayesian Estimates of Equation System Parameters: An Application of Integration by Monte Carlo,” *Econometrica*, vol. 46, no. 1, pp. 1–19, 1978, doi: 10.2307/1913641.

[101] M. D. Shields, K. Teferra, A. Hapij, and R. P. Daddazio, “Refined Stratified Sampling for efficient Monte Carlo based uncertainty quantification,” *Reliab. Eng. Syst. Saf.*, vol. 142, pp. 310–325, Oct. 2015, doi: 10.1016/j.ress.2015.05.023.

[102] D. Meister, S. Ogaki, C. Benthin, M. J. Doyle, M. Guthe, and J. Bittner, “A Survey on Bounding Volume Hierarchies for Ray Tracing,” 2021, Accessed: Jul. 06, 2025. [Online]. Available: https://doi.org/10.1111/cgf.142662

[103] Q. P. Chen, B. Xue, and J. I. Siepmann, “Using the k-d Tree Data Structure to Accelerate Monte Carlo Simulations,” *J. Chem. Theory Comput.*, vol. 13, no. 4, pp. 1556–1565, Apr. 2017, doi: 10.1021/acs.jctc.6b01222.

[104] P. Debevec, “Rendering synthetic objects into real scenes: bridging traditional and image-based graphics with global illumination and high dynamic range photography,” in *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, in SIGGRAPH ’98. New York, NY, USA: Association for Computing Machinery, Jul. 1998, pp. 189–198. doi: 10.1145/280814.280864.

[105] Y. Jiang *et al.*, “GaussianShader: 3D Gaussian Splatting with Shading Functions for Reflective Surfaces,” in *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Seattle, WA, USA: IEEE, Jun. 2024, pp. 5322–5332. doi: 10.1109/CVPR52733.2024.00509.

[106] J. T. Kajiya, “The rendering equation,” in *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, in SIGGRAPH ’86. New York, NY, USA: Association for Computing Machinery, Aug. 1986, pp. 143–150. doi: 10.1145/15922.15902.

[107] B. Walter, S. R. Marschner, H. Li, and K. E. Torrance, “Microfacet models for refraction through rough surfaces,” in *Proceedings of the 18th Eurographics conference on Rendering Techniques*, in EGSR’07. Goslar, DEU: Eurographics Association, Jun. 2007, pp. 195–206.

[108] J. Munkberg *et al.*, “Extracting Triangular 3D Models, Materials, and Lighting From Images,” Apr. 11, 2023, *arXiv*: arXiv:2111.12503. doi: 10.48550/arXiv.2111.12503.

[109] J. Gao *et al.*, “Relightable 3D Gaussians: Realistic Point Cloud Relighting with BRDF Decomposition and Ray Tracing,” in *Computer Vision – ECCV 2024*, vol. 15103, A. Leonardis, E. Ricci, S. Roth, O. Russakovsky, T. Sattler, and G. Varol, Eds., in Lecture Notes in Computer Science, vol. 15103. , Cham: Springer Nature Switzerland, 2025, pp. 73–89. doi: 10.1007/978-3-031-72995-9_5.

[110] P. Wang, L. Liu, Y. Liu, C. Theobalt, T. Komura, and W. Wang, “NeuS: learning neural implicit surfaces by volume rendering for multi-view reconstruction,” in *Proceedings of the 35th International Conference on Neural Information Processing Systems*, in NIPS ’21. Red Hook, NY, USA: Curran Associates Inc., Dec. 2021, pp. 27171–27183.

[111] J. Zhang, Y. Yao, S. Li, Z. Luo, and T. Fang, “Visibility-aware Multi-view Stereo Network,” Aug. 19, 2020, *arXiv*: arXiv:2008.07928. doi: 10.48550/arXiv.2008.07928.

[112] L. Liu, J. Gu, K. Z. Lin, T.-S. Chua, and C. Theobalt, "Neural sparse voxel fields," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, in NIPS '20. Red Hook, NY, USA: Curran Associates Inc., Dec. 2020, pp. 15651–15663.

[113] L. Yariv *et al.*, "Multiview neural surface reconstruction by disentangling geometry and appearance," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, in NIPS '20. Red Hook, NY, USA: Curran Associates Inc., Dec. 2020, pp. 2492–2502.

[114] Y. Yao, Z. Zeng, C. Gu, X. Zhu, and L. Zhang, "Reflective Gaussian Splatting," Feb. 03, 2025, *arXiv*: arXiv:2412.19282. doi: 10.48550/arXiv.2412.19282.

[115] B. Huang, Z. Yu, A. Chen, A. Geiger, and S. Gao, "2D Gaussian Splatting for Geometrically Accurate Radiance Fields," in *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers '24*, Denver CO USA: ACM, Jul. 2024, pp. 1–11. doi: 10.1145/3641519.3657428.

[116] C. Gu, X. Wei, Z. Zeng, Y. Yao, and L. Zhang, "IRGS: Inter-Reflective Gaussian Splatting with 2D Gaussian Ray Tracing," Mar. 24, 2025, *arXiv*: arXiv:2412.15867. doi: 10.48550/arXiv.2412.15867.

[117] N. Moenne-Loccoz *et al.*, "3D Gaussian Ray Tracing: Fast Tracing of Particle Scenes," Oct. 10, 2024, *arXiv*: arXiv:2407.07090. doi: 10.48550/arXiv.2407.07090.

[118] J. Li, Z. Wu, E. Zamfir, and R. Timofte, "ReCap: Better Gaussian Relighting with Cross-Environment Captures," *Proc. Comput. Vis. Pattern Recognit. Conf.*, pp. 21307–21316, 2025.

[119] B. Karis, "Real Shading in Unreal Engine 4," *Proc Phys. Based Shading Theory Pract.*, p. 4(3):1.

[120] N. Hoffman, "Crafting Physically Motivated Shading Models for Game Development".

[121] P. Debevec, "Rendering with natural light," in *ACM SIGGRAPH 98 Electronic art and animation catalog*, in SIGGRAPH '98. New York, NY, USA: Association for Computing Machinery, Jan. 1998, p. 166. doi: 10.1145/281388.281983.

[122] J. Munkberg *et al.*, "Extracting Triangular 3D Models, Materials, and Lighting From Images," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2022, pp. 8270–8280. doi: 10.1109/CVPR52688.2022.00810.

[123] D. Verbin, P. Hedman, B. Mildenhall, T. Zickler, J. T. Barron, and P. P. Srinivasan, "Ref-NeRF: Structured View-Dependent Appearance for Neural Radiance Fields," Dec. 07, 2021, *arXiv*: arXiv:2112.03907. doi: 10.48550/arXiv.2112.03907.

[124] "Tanks and temples: benchmarking large-scale scene reconstruction: ACM Transactions on Graphics: Vol 36, No 4." Accessed: Jun. 29, 2025. [Online]. Available: https://dl.acm.org/doi/10.1145/3072959.3073599

[125] Z. Kuang, Y. Zhang, H.-X. Yu, S. Agarwala, S. Wu, and J. Wu, "Stanford-ORB: A Real-World 3D Object Inverse Rendering Benchmark," Jan. 17, 2024, *arXiv*: arXiv:2310.16044. doi: 10.48550/arXiv.2310.16044.

[126] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, “Image quality assessment: from error visibility to structural similarity,” *IEEE Trans. Image Process.*, vol. 13, no. 4, pp. 600–612, Apr. 2004, doi: 10.1109/TIP.2003.819861.

[127] J. Nilsson and T. Akenine-Möller, “Understanding SSIM,” Jun. 29, 2020, *arXiv*: arXiv:2006.13846. doi: 10.48550/arXiv.2006.13846.

[128] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, “The Unreasonable Effectiveness of Deep Features as a Perceptual Metric,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun. 2018, pp. 586–595. doi: 10.1109/CVPR.2018.00068.

[129] A. Mischok, “Adams Place Bridge HDRI • Poly Haven,” Poly Haven. Accessed: Jun. 29, 2025. [Online]. Available: https://polyhaven.com/a/adams_place_bridge

[130] “Courtyard HDRI • Poly Haven.” Accessed: Jun. 29, 2025. [Online]. Available: https://polyhaven.com/a/courtyard

[131] G. Zaal, “Shady Patch HDRI • Poly Haven,” Poly Haven. Accessed: Jun. 29, 2025. [Online]. Available: https://polyhaven.com/a/shady_patch

[132] G. Zaal, “Hotel Room HDRI • Poly Haven,” Poly Haven. Accessed: Jun. 29, 2025. [Online]. Available: https://polyhaven.com/a/hotel_room

[133] G. Zaal, “Moonless Golf HDRI • Poly Haven,” Poly Haven. Accessed: Jun. 29, 2025. [Online]. Available: https://polyhaven.com/a/moonless_golf