Universität des Saarlandes
Max-Planck-Institut für Informatik

Novel View Synthesis of Structural Color Objects Created by Laser Markings

Masterarbeit im Fach Medieninformatik
Master's Thesis in Media Informatics
von / by

Prakash Naikade

angefertigt unter der Leitung von / supervised by
Dr. Vahid Babaei

betreut von / advised by
Dr. Vahid Babaei

begutachtet von / reviewers
Dr. Vahid Babaei
Prof. Dr. Eddy Ilg

Saarbrücken, August, 2024

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement in Lieu of an Oath

I hereby confirm under oath that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass die vorliegende Arbeit mit der elektronischen Version übereinstimmt.

Statement in Lieu of an Oath

I hereby confirm the congruence of the contents of the printed data and the electronic version of the thesis.

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

_____ _____

(Datum / Date) (Unterschrift / Signature)

Acknowledgments

I am profoundly thankful to everyone who contributed to shaping this thesis into its current state. The interactions with these individuals not only improved my research skills but also enriched my personal growth.

First and foremost, I want to acknowledge Dr. Vahid Babaei for entrusting me with the opportunity to engage in this fascinating project. His insights and feedback played a pivotal role in advancing the research consistently. Special thanks go to Siddhartha, Balaji Venkatesan, Emiliano Luci, Sebastian Cucerca, and other colleagues in the research group for their valuable discussions and assistance throughout my thesis period. I also want to express my sincere gratitude to Prof. Dr. Eddy Ilg for dedicating the time to review the thesis.

Lastly, a heartfelt thanks to all my friends and family in Saarbrücken and back home in India. Their shared joy in good times and unwavering support during tough times made this entire journey much more manageable.

Novel View Synthesis of Structural Color Objects Created by Laser Markings

by

Prakash Naikade

Submitted to the Department of Computer Science
in August, 2024, in partial fulfillment of the
requirements for the degree of
Master of Science in Media Informatics

Abstract

Transforming physical object into its high quality 3D digital twin using novel view synthesis is crucial for researchers in the domain of automatic laser marking of any color image on different metal substrates. Current Radiance Field methods have significantly advanced novel view synthesis of scenes captured with multiple photos or videos. But, they struggle to represent the scene with shiny objects. Moreover, multiview reconstruction of reflective objects with structural colors is extremely challenging because specular reflections are view-dependent and thus violate the multiview consistency, which is the cornerstone for most multiview reconstruction methods. However, there is a general lack of synthetic datasets for objects with structural colors and a literature review on state-of-the-art (SOTA) novel view synthesis methods for this kind of materials. Addressing these issues, we introduce a novel synthetic dataset that is used to conduct quantitative and qualitative analysis on a SOTA novel view synthesis methods. We demonstrate different techniques to improve the scene representation of laser printed planar structural color objects, focusing on the 3D Gaussian Splatting (3D-GS) method, which performs exceptionally well on our synthetic dataset. Our techniques, such as using geometric prior of planar structural color objects while initializing scene with sparse structure-from-motion (SfM) point cloud and the Anisotropy Regularizer, significantly improves the visual quality of view synthesis. We design different capture setups to acquire images of objects and evaluate the visual quality of the scene with different capture setups. Additionally, we present comprehensive experimentation to demonstrate methods to simulate structural color objects using just captured images of laser-printed primaries. This comprehensive research aims to contribute to the advancement of novel view synthesis methods for scenes involving reflective objects with structural colors.

Thesis Supervisor: Dr. Vahid Babaei
Title: Senior Researcher [Research Group Leader]

Contents

Chapter 1

Introduction

1.1 Motivation

In recent years, there has been a growing desire among consumers and researchers for immersive and realistic experiences while interactively visualizing product objects they are interested in. This has captured the attention of both academia and industry in transforming physical objects into their digital twins using novel view synthesis.

Traditional view synthesis is extensively applied in both computer vision and computer graphics to simulate a scenes or objects from various perspectives and under diverse conditions. However, this method necessitates the explicit input of all physical parameters related to the scenes, encompassing camera parameters, lighting conditions, and material properties of the objects. When attempting to create controllable images of real-world scenes, traditional view synthesis approaches encounter considerable difficulties due to the absence of explicit physical parameters. The limited availability of comprehensive physical parameter information restricts the development of numerous practical applications such as product visualization and simulating or creating digital twin of physical objects.

Contrary to this, current Radiance Field methods have recently revolutionized novel-view synthesis of scenes captured with multiple photos or videos, mostly as a result of advances in machine learning. These radiance field methods works well for scenes with diffuse surfaces. But they struggle to represent the scene with shiny objects with reflective surfaces. Multiview reconstruction of reflective objects with structural colors is extremely challenging because specular reflections are view-dependent and thus violate the multiview consistency, which is the cornerstone for most multiview reconstruction methods, as structural colors are based on the physical structure of the object rather than the chemical composition. Structural colors arises from the interference, diffraction, or scattering of light by the microscopic structure of the material.

However, there is a general lack of synthetic datasets for objects with structural colors, and a literature review on state-of-the-art (SOTA) novel view synthesis methods for such datasets. Thus, there is lack of benchmarking of these SOTA methods for scenes involving objects with structural colors.

Addressing the aforementioned issues, in this work, we introduce synthetic dataset with scene involving object which has surface with structural colors. We further present a evaluation of a SOTA novel view synthesis methods on this synthetic dataset.

Furthermore, We demonstrate different techniques to improve the visual quality of scene representation of laser-printed planar objects with structural colors i.e. images or paintings laser-printed on different metal substrates, focusing on the 3D Gaussian Splatting (3D-GS) [32] method, which outperforms the other novel view synthesis methods on our synthetic dataset with respect to visual quality metrics, training time, and inference speed. We show that using geometric prior of planar object of structural color object while initializing scene with sparse structure-from-motion (SfM) point cloud and Anisotropy Regularizer significantly improves the visual quality of view-synthesis. We design different capture setups to acquire images of object with structural colors and evaluate the visual quality of the scene with different capture setups.

We further present comprehensive experimentation to demonstrate methods to simulate structural color objects with just captured images of laser-printed primaries using Radiance Field method and Optical Flow method i.e. 3D-GS [32] and CoTracker [31] respectively. These primaries are color patches generated by laser-marking on metal substrates. This comprehensive research aims to draw attention of researchers and contribute to the advancement of novel view synthesis methods for scenes involving reflective objects with structural colors.

1.2 Contribution

In this work, we present a new synthetic dataset named Structural Color Blender dataset containing StructColorToaster scene representing object with structural colors. Exploring related different research directions, we discuss unexplored realm of novel view synthesis of scenes involving objects with structural colors.

Additionally, we also conduct extensive experiments to showcase techniques for simulating structural color objects solely based on images captured from laser-printed primaries on metal substrates.

Our key contributions can be summarized as follows:

- We present StructColorToaster scene, in a synthetic dataset called Structural Color Blender, for novel view synthesis of scenes containing objects with structural colors. It provides a significant addition to materials not tested before for SOTA novel view synthesis methods.
- We further present a quantitative and qualitative evaluation of a SOTA novel view synthesis methods on this synthetic dataset.
- We further show that the different techniques to improve the visual quality of scene representation of laser-printed planar structural color objects i.e. images or paintings laser-printed on different metal substrates, focusing on the 3D Gaussian Splatting (3D-GS) [32] method, which outperforms the other novel view synthesis methods on our synthetic dataset with respect to visual quality metrics, training time, and inference speed. We show that using geometric prior of planar structural-color object while initializing scene with sparse structure-from-motion (SfM) point cloud and Anisotropy Regularizer significantly improves the visual quality of view-synthesis by reducing floaters and regularizing the size of Gaussians.
- We design different capture setups to acquire images of object and evaluate the visual quality of the scene with different capture setups, demonstrating best practices for collecting real datasets with a hand-held camera.
- We further present comprehensive experimentation to demonstrate methods to simulate structural color objects using just captured images of laser-printed primaries employing the Radiance Field method and Optical Flow method, i.e. 3D-GS [32] and CoTracker [31] respectively.

1.3 Outline of the Thesis

The rest of this thesis is structured as follows:

- Chapter 2 provides the required background and other preliminaries for easy understanding of the later chapters.
- Chapter 3 discusses the most relevant works within the scope of this thesis.
- Chapter 4 discusses the generated synthetic dataset in detail.
- In Chapter 5, we discuss the quantitative and qualitative results of the experiments conducted with the generated synthetic datasets.
- We then move onto the problem of capturing real world dataset i.e. laser-printed planar structural color object demonstrating best practices for collecting real dataset with a hand-held camera in chapter 6.

- In the subsequent chapter 7, we discuss evaluation of vanilla 3D-GS [32] on real dataset and different techniques to improve the visual quality of scene representation using 3D-GS.
- In this Chapter 8, we will try to simulate object of structural color painting using only primary patches corresponding to the primaries intended for use in the laser printing process.
- Chapter 9 summarizes the work and discusses possible extensions to the proposed work.

Chapter 2

Background

In this chapter, we discuss some of the key concepts that are important in the context of this thesis. We start by giving an introduction to novel view synthesis. We then discuss Traditional View Synthesis Methods, Learning-Based View Synthesis Methods, Evaluation Metrics used in radiance field methods, Differential Volume Rendering, Structure from Motion, Spherical Harmonics, and Perspective Transformation.

2.1 Novel View Synthesis

In computer graphics and computer vision, novel view synthesis is a task referring to rendering images or views of a specific object or scene from unseen or unobserved viewpoints, given a sparse set of images and respective camera poses, as shown in Figure 2-1). It has a wide range of applications in product visualization, scene editing, scene relighting, augmented reality (AR), virtual reality (VR), etc. As a result, it has attracted considerable interest from the computer vision and graphics communities, resulting in numerous advances and practical applications [82]. Primary challenges associated with novel view synthesis are inferring the scene's 3D structure given sparse observations, as well as inpainting of occluded and unseen parts of the scene.

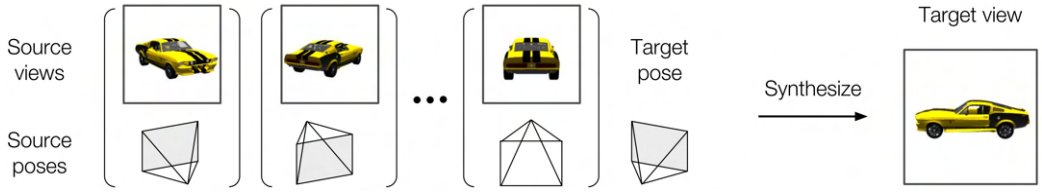


Figure 2-1: This figure depicts the goal of novel view synthesis. It refers to generating image for the scene from new viewpoint not in the training data, given training data includes sparse set of (image, viewpoint) pairs for scene. Images from [65].

Classic novel view synthesis techniques like Image-Based Rendering (IBR) [30], Light Field Rendering [44], Multi-View Stereo (MVS) [14], etc has formed the foundation

for subsequent research in novel view synthesis and provided valuable insights into the field.

While these classical approaches have contributed substantially to novel view synthesis, they frequently faced challenges in dealing with complex scenes, occlusions, and substantial differences between view. However, current advances in machine learning and deep learning have led to the development of more sophisticated data-driven methods.

In past years, the application of deep learning has demonstrated promising outcomes in producing high-quality novel views. This includes the use of Convolutional Neural Networks (CNNs) [71], Variational Autoencoders (VAEs) [34], and Generative Adversarial Networks (GANs) [18]. Initially, the early phases of deep learning-based novel view synthesis commonly employed 2D convolutional encoder-decoder architectures. These architectures are tasked to map sparse input to the target image with respect to the desired view. Some techniques have investigated various representations, such as multiplane images [75], depth-layered images [61], light fields [47], depth-based warping [52] [95], etc. Those approaches typically employ geometric modeling, optimization, and image warping to render novel views. Certain techniques predicted 2D flow fields [61] and few methods directly predicted colors [81]. However, these methods were surpassed by 3D-aware convolutional methods. These 3D-aware convolutional methods utilized techniques such as volumetric rendering [39], rasterization [80], or learnable neural rendering [12] to encode and render explicit 3D latent features.

Lately, there has been a notable increase in interest surrounding implicit neural shape and appearance representations, as evidenced by recent work such as Scene Representation Networks [63], Neural Radiance Field (NeRF) [48], Mip-NeRF [3], Mip-NeRF 360 [4], and InstantNGP [51]. However, these NeRF methods were computationally intensive, often requiring extensive training times and substantial resources for rendering, especially for high-resolution outputs.

Therefore, more recently explicit representations based methods like Plenoxels [13] and 3D-GS [32] provides highly parallelized workflows, facilitates more efficient computation and real-time rendering. As these methods use primitives like voxels and points, which inherit the properties of differentiable volumetric representations while being unstructured and explicit, they allow very fast computation and rendering.

2.2 Traditional View Synthesis Methods

Traditional approaches for novel view synthesis can be divided into two categories: Light Field Rendering and Image-Based Rendering (IBR). Light field rendering involves interpolating directly between densely sampled images, while IBR utilizes geometric information obtained from multi-view stereo (MVS) to generate views.

2.2.1 Light Field Rendering

Early research in novel view synthesis recognizes that the task of generating novel views can be redefined as approximating a light field representation [44]. This representation is characterized as a continuous *5-dimensional Plenoptic function* that provides the incoming radiance L originating from the scene for a specified position V_x, V_y, V_z, viewing direction (θ, ϕ), at given time t, and wavelength λ,

$$\boldsymbol{L} = P(\boldsymbol{\theta}, \boldsymbol{\phi}, \lambda, \mathbf{V_x}, \mathbf{V_y}, \mathbf{V_z}, t) \tag{2.1}$$

The light field records the propagation of light within the scene, enabling the generation of new perspectives by sampling it from selected positions. With a comprehensive set of source views and camera positions covering the parameter space thoroughly, the light field can be divided into a uniform grid and interpolated between points to render new views [7].

2.2.2 Image-Based Rendering (IBR)

Novel view synthesis using IBR techniques leverage a geometric approximation of the scene in conjunction with neighboring views to create new views. The emergence of structure from motion (SfM) and multi-view stereo (MVS) algorithms provided a robust means to estimate both the camera poses and scene geometry from a collection of images, laying the groundwork for IBR. The aim of SfM [60] is to determine the parameters of each camera and the 3D coordinates of the common points in the scene by searching for the correspondences between the images and an incremental reconstruction procedure. MVS [14] extends this SfM approach by associating every pixel in the source views with a globally consistent depth value and then generated dense point cloud transformed into textured mesh using algorithms like Delaunay triangulation [25].

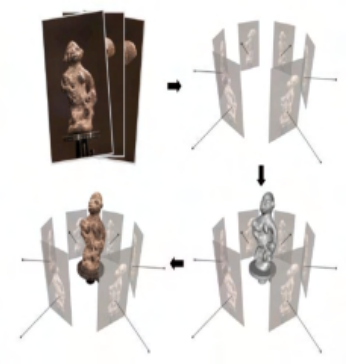


Figure 2-2: Multi-view stereo algorithm steps. We start by capturing the images, then compute the parameters of the camera for each image, follow this by reconstructing 3-D geometry of the object from the series of images using a triangulation process, and then we can optionally reconstruct the materials of the scene. Images from [20].

2.2.3 Limitations of Traditional View Synthesis

While all the methods discussed can generate believable views, but these methods have many limitations. The technique of light field interpolation is only viable when dealing with a sufficiently dense set input of images.

On the other hand, IBR offers greater scalability and can be seamlessly integrated into existing graphics engines. Nonetheless, it encounters its own set of challenges. Primarily, its effectiveness is inherently tied to the accuracy of the reconstructed surface geometry. Scenes featuring intricate geometry and view-dependent materials pose particular difficulties for SfM and MVS algorithms, resulting in noisy or erroneous geometry and areas of omission. Basically, artifacts such as ghosting, blur, holes, or seams can arise due to view-dependent effects, imperfect proxy geometry, or too few source images.

In summary, generating novel views from scenes characterized by sparse views, detailed geometry, and view-dependent materials remains a formidable task for classic view synthesis methods, which lead to the exploration of machine learning and deep learning based approaches to address these challenges.

2.3 Learning-Based View Synthesis Methods

In this section, we will introduce deep learning and some important deep learning architectures used in the learning-based view synthesis methods which can be categorized into two main categories: implicit scene representation and explicit scene representation,

$$L_{\text{implicit}}(x, y, z, \theta, \phi) = \text{NeuralNetwork}(x, y, z, \theta, \phi), \tag{2.2}$$
$$L_{\text{explicit}}(x, y, z, \theta, \phi) = \text{DataStructure}[(x, y, z)] \cdot f(\theta, \phi), \tag{2.3}$$

where radiance field function $L(x, y, z, \theta, \phi)$ maps a point in space (x, y, z), and a direction specified by spherical coordinates (θ, ϕ), to a non-negative radiance value. And, then we can divide these two main categories into many subcategories such as image-based rendering, volumetric representations and neural implicit scene representations, explicit scene representation with and without neural network, etc.

2.3.1 Deep Learning

Deep learning is a subset of machine learning based on artificial neural networks (ANNs) with a goal to solve computational problems by training multi-layer networks to automatically learn feature representations from large data and make predictions [37]. In supervised learning [28], given a dataset consisting of input-output pairs

$(\mathbf{x_i}, \mathbf{y_i})$, the objective is to discover a computational model $\boldsymbol{F_\theta(x)}$ that predicts $\mathbf{y}$ for any given $\mathbf{x}$ by minimizing appropriate loss functions $\mathcal{L}$, such that

$$\boldsymbol{\theta} = \arg\min_{\theta} \sum_{i} \mathcal{L}\left(\boldsymbol{F_\theta}\left(\mathbf{x}_i\right), \mathbf{y}_i\right) \tag{2.4}$$

We represent $\boldsymbol{F_\theta}$ as Deep Neural Networks (DNN) and then the expression 2.4 is minimized using first-order optimization techniques. DNNs are consist of multiple layers representing non-linear functions, such that,

$$\mathbf{f}(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{2.5}$$

where, σ is activation function [28] like *Sigmoid, Tanh, ReLU*; $\mathbf{W}$ and $\mathbf{b}$ are weights and bias of the layer respectively. These layers can be stacked to form a DNN, specifically known as Feed-Forward Neural Network (FFN),

$$F(x) = f_0 \oplus f_1 \oplus \ldots \oplus f_N(x) \tag{2.6}$$

During training of a DNN, the network is optimized by minimizing the loss function using backpropagation [22]. Backpropagation is a method to update the weights in the neural network by taking into account the actual output and the desired output. The derivative with respect to weight $\mathbf{W}$ is computed using chain rule and is of the following form:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_i} = \frac{\partial \mathbf{f}_i}{\partial \mathbf{W}_i} \frac{\partial \mathcal{L}}{\partial \mathbf{f}_i} \tag{2.7}$$

As a result, the weight is updated as follows:

$$\mathbf{W}_i \leftarrow \mathbf{W}_i - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{W}_i} \tag{2.8}$$

This process is also known as gradient descent and more sophisticated schemes like ADAM [33] are available now.

2.3.2 Multi-Layer Perceptron

A multilayer perceptron (MLP) [21] is a form of a feedforward neural network comprised of fully connected neurons with a non-linear activation function. MLPs consist of a sequence of layers where the weight matrix is fully connected, meaning every element in the matrix is non-zero and trainable. Due to the computational demands of a fully connected weight matrix, these networks usually handle low-dimensional data. Its primary purpose is to discriminate between data that lack linear separability.

MLPs have found extensive application across diverse domains, such as image recognition, natural language processing, and speech recognition. Their adaptability in structure and capacity to model continuous scene representations [48] for novel view synthesis under specific circumstances establish them as a fundamental cornerstone

in both deep learning and learning-based novel view synthesis investigations.

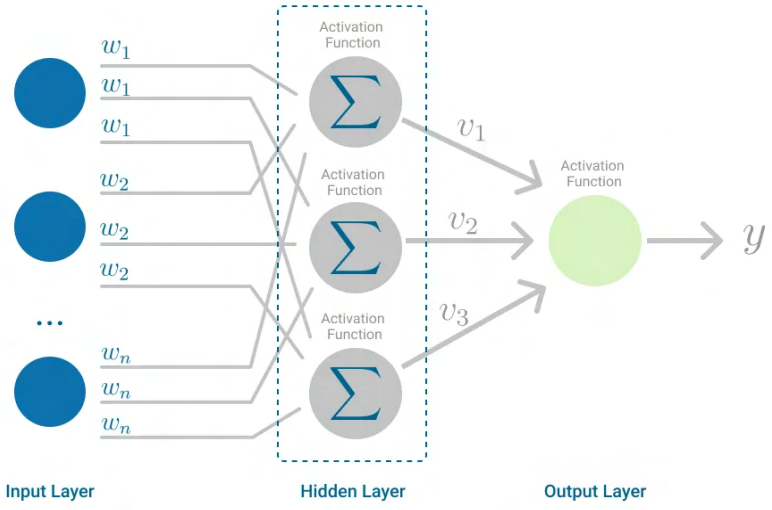


Figure 2-3: Multilayer Perceptron, highlighting input layer, hidden layer, activation function, and output layer. Images taken from [5].

2.3.3 Neural Image-Based Rendering

Neural Image-based Rendering (N-IBR) aims to enhance the processes involved in classical image-based rendering by incorporating or substituting elements of view synthesis with supervised learning models. Basically, N-IBR is a fusion of traditional image-based rendering techniques and deep neural networks, where learned components replace conventional heuristics. N-IBR techniques substitute the typical heuristics employed in classical IBR methods with learned blending functions or corrections. These adjustments consider view-dependent effects to enhance rendering outcomes.

N-IBR techniques substitute the typical heuristics employed in classical IBR methods with learned blending functions or corrections [23, 56, 73]. These adjustments consider view-dependent effects to enhance rendering outcomes. Hedman et al. [23] proposes to use U-Net [58] to predict the blending weights of the projected source images for compositing in the target image space. The image is generated by combining the weighted colors from each reprojected source view. Recently, the focus has shifted towards predicting density rather than depth, and subsequently generating the view through volume rendering techniques [77].

2.3.4 Neural Re-rendering

Neural Re-rendering is inspired by advancements in image-to-image translation [24]. These methods integrates classical 3D representation and deep neural network enabled renderer. In these methods, proxy geometry, such as point cloud or mesh [46]

obtained using 3D reconstruction, is rasterized into a graphics buffer using a traditional rasterizer, and then the graphics buffers are translated into the target view using the rendering network, synthesizing more comprehensive and realistic views.

2.3.5 Multiplane Images

A multiplane image comprises a series of RGBα images positioned within a view frustum at fixed depth intervals, with α representing the pixel's opacity.

Zhou et al. [94] popularized the concept of multiplane image representation (MPI) for synthesizing views for forward-facing scenes. In this approach, for each camera ray intersecting the image at depths ranging from $d = 1, ..., D$, the projected color c is formed by compositing each intersected color $\mathbf{c}_d$ in front-to-back fashion:

$$\mathbf{c} = \sum_{d=1}^{D} \alpha_d \mathbf{c}_d \prod_{i=d+1}^{D} (1 - \alpha_i) \tag{2.9}$$

As shown in figure 2-4, given a sparse set of input views of an object, Xu et al. [86] also address the problem of rendering the object from novel view-points.

(a) Input images

(b) View synthesis result and Ground truth image

Figure 2-4: Xu et al. [86] present a method to synthesize scene appearance from a novel view by interpolating only six wide-baseline images. Images taken from [86].

The primary benefit of multiplane images is their ease of storage in common image formats and integration into graphics engines for fast rendering, eliminating the necessity of neural network evaluation. However, these representations have certain constraints, particularly concerning translational movement within the scene.

2.3.6 Voxel Grids Based Neural Scene Representations

A voxel grid organizes scene information in a grid-like structure similar to Cartesian coordinates. A multiplane image representation is restricted to forward-facing scenes, but voxel grids based methods have the capability to depict various scene types within a bounded volume. Motivated by advancements in geometric deep learning [9], a series of neural rendering methods have emerged, advocating the representation of the scene through a voxel grid, thus ensuring the enforcement of 3D structure.

DeepVoxels [62] proposed to use convolutional neural networks to encode images into a voxel grid of latent codes and then back project the grid representation to a novel viewpoints.

NeuralVolumes [40] proposed a technique in which a convolutional network predicts an $RGB\alpha$ grid, which is subsequently utilized for rendering a novel view using volume rendering [11].

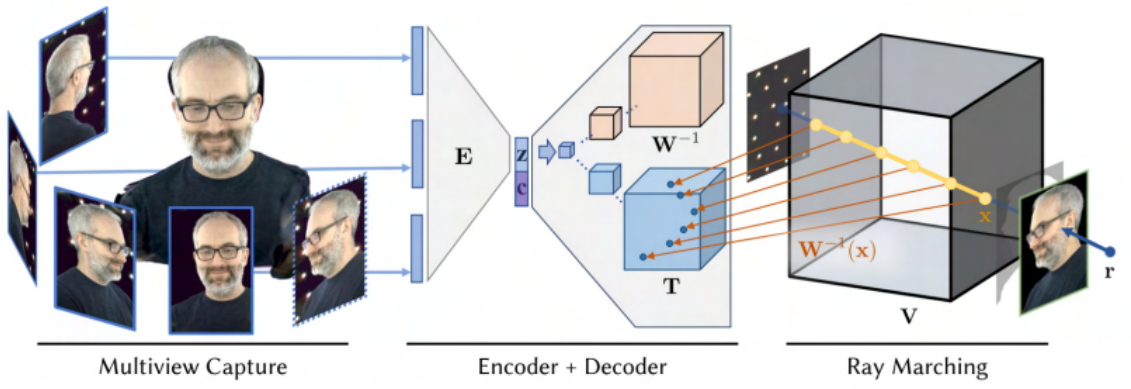


Figure 2-5: Workflow of Neural Volumes [40]. Multi-view capture is input to an encoder to produce a latent code $\mathbf{z}$. $\mathbf{z}$ is decoded to a volume that stores RGB and α values for each point in space, as well as a warp field used to index into the $RGB\alpha$ volume. Differentiable ray marching renders the volume into an image, allowing the system to be trained by minimizing the difference between rendered and target images. Images taken from [40].

2.3.7 Implicit Neural Representations

While 3D voxel grids have demonstrated that a 3D-structured scene representation benefits multiview consistent scene modeling, their memory requirement scales cubically with spatial resolution, recent works [53, 45, 63] alleviated these problems by modeling geometry and appearance (color) using implicit functions as the level set of a neural network. These implicit neural representations store information about the scene in the weights of a coordinate-based neural network. Coordinate-based neural

representations are neural networks that are trained using spatial coordinates as input data such as position, direction, and respective images which output a continuous quantity that implicitly describes the scene.

Early works, *Occupancy Network* [45] and *DeepSDF* [53] were restricted to explicit 3D information such as point clouds or meshes, as a supervisory signal but later *Scene Representation Networks* (SRNs) proposed to utilize differentiable rendering networks to use images as supervisory signal. Lastly, Mildenhall et al. [48] introduced radiance fields method paired with differentiable volume rendering and gave promising direction for novel view synthesis.

A frequently adopted network architecture for such representations consists of multilayer perceptrons (MLPs) with ReLU activation. But MLPs with ReLU as activation function struggle to model high-frequency function using low frequency input such as spatial coordinates. To solve this, Tancik et al. [69] showed positional encoding, which maps low-dimensional data to higher frequency, and can improve the performance of MLPs at scene representation.

The disadvantages of using MLPs to model scenes are their inefficiency and slowness in training and inference.

2.3.8 Explicit Representations with and without Neural Networks

In *NeRF-based* [48] methods, evaluating the MLP predictions for each given 3D coordinate individually is computationally expensive and slow to train. To address this, *InstantNGP* [51] encode coordinates using a hash map and linear interpolation.

Plenoxels [90], utilize a sparse voxel grid to interpolate a continuous density field, eliminating the need for neural networks entirely. Although both *Planoxels* [90] and *InstantNGP* [51] approaches produce remarkable representation of the scene, they may encounter challenges to effectively represent empty space. Moreover, the quality of the image is largely constrained by the selection of structured grids utilized for acceleration, whereas rendering speed is affected by the requirement to query numerous samples for each ray-marching step.

Point-based methods [42] efficiently render disconnected and unstructured geometry samples such as point clouds. There has been recent interest in differentiable point-based rendering techniques [89]. *PointNeRF* [85] uses points to represent a radiance field with a radial basis function approach using volumetric ray marching and, therefore, lacks real-time rendering speed. To address all these issues, Kerbl et al. [32] proposed *3D-GS* without any neural network for real-time rendering of radiance fields.

2.4 Evaluation Metrics

We use the Peak Signal-to-Noise Ratio (PSNR), Structural Similarity Index (SSIM), and Learned Perceptual Image Patch Similarity (LPIPS) proposed by Zhang et al. [93] as our primary quantitative and qualitative metrics to assess the quality of synthesized scenes compared to ground truth images.

2.4.1 PSNR

PSNR, is a commonly used metric to measure the reconstruction quality of images, expressed in terms of the logarithmic decibel scale and is directly related to the mean squared error (MSE). Assuming image resolution be (W, H) and each color lies in [0, 1], PSNR between a ground-truth RGB-image I^{gt} and a synthesized RGB-image I is defined as:

$$\text{PSNR} = 20\log_{10}(\max(I)) - 10\log_{10}(\text{MSE}) = -10\log_{10}(\text{MSE}) \tag{2.10}$$

Where MSE is expressed as:

$$\text{MSE} = \frac{1}{3WH}\sum_{j=1}^{H}\sum_{i=1}^{W}\left\|I_{i,j} - I_{i,j}^{\text{gt}}\right\|_2^2 \tag{2.11}$$

The higher the PSNR, the better the quality of the synthesized image.

2.4.2 SSIM

SSIM enhances perceptual similarity assessment by recognizing the human visual system's sensitivity to changes in structural information. It characterizes image structure, contrast, and luminance through statistical features, particularly focusing on the mean μ, variance σ^2 and covariance σ of both ground-truth RGB-image I^{gt} and a synthesized RGB-image I:

$$\text{SSIM} = \frac{(2\mu_{I^{gt}}\mu_I + C_1)\,(2\sigma_{I^{\text{g}},I} + C_2)}{\left(\mu_{I^{\text{E}}}^2 + \mu_I^2 + C_1\right)(\sigma_{I^{\text{g}}}^2 + \sigma_I^2 + C_2)} \tag{2.12}$$

Where, C_1 and C_2 are small constants to prevent divergence. A value of 1 indicates that the two given images are very similar or the same while a value of 0 indicates the two given images are very different.

2.4.3 LPIPS

Although widely used perceptual metrics are PSNR and SSIM but these metrics are simplistic and superficial, they overlook many nuances of human perception. LPIPS, a relatively effective metric, assesses the distance between image patches obtained from

the layers of a deep neural network. Researchers found that deep neural networks (DNNs) trained on image prediction and modeling tasks possess the emergent ability to learn feature representations that closely align with human perception. LPIPS leverages this characteristic to compare images by inputting both images into such a network, aggregating the output of intermediate layers y^l and subsequently computing their average euclidean distance:

$$\text{LPIPS} = \sum_{l}^{\text{layers}} \frac{1}{W_l H_l} \sum_{(i,j)}^{(W_l,H_l)} \left\| \mathbf{w}_l \odot \left(\mathbf{y}^l_{i,j} - \hat{\mathbf{y}}^l_{i,j} \right) \right\|_2^2 \tag{2.13}$$

Here, weights $\mathbf{w}_l$ are optimized by fine-tuning the network using a smaller network that predicts perceptual judgments based on a dataset annotated by humans. A higher LPIPS value indicates greater dissimilarity between the rendered scene and the original set of input images, while a lower LPIPS score suggests greater similarity between the generated scene and the input image sequence.

2.5 Differential Volume Rendering

NeRF [48] based methods utilize basic volume rendering to address the difficulties associated with differentiating through ray-triangle intersections using a probabilistic notion of visibility. The approach assumes that the scene consists of a collection of light-emitting particles whose density varies in space. In the context of physically-based rendering, this scenario would be characterized as a volume featuring absorption and emission without scattering [67].

Mildenhall et al. [48] takes the density field $\sigma(\mathbf{x})$, with $\mathbf{x}\in\mathbb{R}^3$ indicating the probability of hitting a particle while travelling an infinitesimal distance. Mildenhall et al. [48] reparameterize the density along a given ray $\mathbf{r}=(\mathbf{o},\mathbf{d})$ as a scalar function $\sigma(t)$, since one can express any point $\mathbf{x}$ along the ray can be written as $\mathbf{r}(t)=\mathbf{o}+t\mathbf{d}$. Density is closely associated with the ***Transmittance function*** $\mathcal{T}(t)$, which indicates the probability of a ray traveling over the interval $[0,t)$ without hitting any particles.

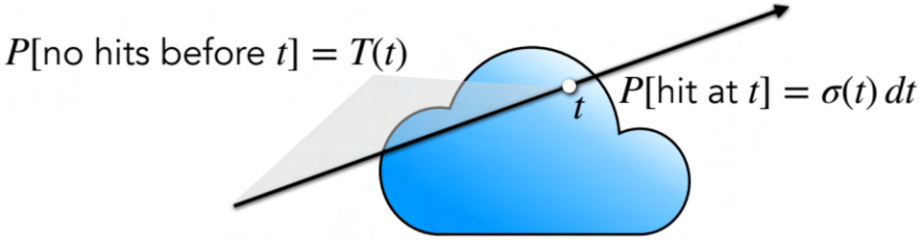


Figure 2-6: While taking a differential step dt probability $\mathcal{T}(t+dt)$ of *not* hitting a particle is equal to the likelihood of the ray reaching t, $\mathcal{T}(t)$, multiplied by the probability of not hitting anything during the step $(1-dt\cdot\sigma(t))$. Images taken from [68].

$$\mathcal{T}(t+dt) = \mathcal{T}(t) \cdot (1 - dt \cdot \sigma(t)) \tag{2.14}$$

$$\mathcal{T}(t) + \mathcal{T}'(t)\, dt = \mathcal{T}(t) \cdot (1 - dt \cdot \sigma(t)) \qquad \text{Taylor expansion for } \mathcal{T} \tag{2.15}$$

Solve this differential equation for $\mathcal{T}$:

$$\frac{\mathcal{T}'(t)}{\mathcal{T}(t)} = -\sigma(t) \qquad \text{Rearrange} \tag{2.16}$$

$$\int_a^b \frac{\mathcal{T}'(t)}{\mathcal{T}(t)}\, dt = -\int_a^b \sigma(t)\, dt \qquad \text{Integrate} \tag{2.17}$$

$$\log \mathcal{T}(t)\big|_a^b = -\int_a^b \sigma(t)\, dt \tag{2.18}$$

$$\mathcal{T}(a \to b) \equiv \frac{\mathcal{T}(b)}{\mathcal{T}(a)} = \exp\left(-\int_a^b \sigma(t)\, dt\right) \qquad \text{Exponentiate} \tag{2.19}$$

where $\mathcal{T}(a \to b)$ is the probability that the ray travels from distance a to b along the ray without hitting a particle, which is same as $\mathcal{T}(t) = \mathcal{T}(0 \to t)$.

We can understand the function $1 - \mathcal{T}(t)$ known as *opacity* indicating the probability that the ray *does* hit a particle sometime before reaching distance t. Then $\mathcal{T}(t) \cdot \sigma(t)$ is the corresponding **Probability Density Function (PDF)**, giving the likelihood that the ray stops precisely at distance t.

Using **Volume rendering**, we can predict the expected value of the light emitted by the particles within the volume while ray travels from t=0 to D, composited on top of a background color. The expected color can be determined using PDF as,

$$\boldsymbol{C} = \int_0^D \mathcal{T}(t) \cdot \sigma(t) \cdot \mathbf{c}(t)\, dt \; + \; \mathcal{T}(D) \cdot \mathbf{c}_{\text{bg}} \tag{2.20}$$

where $\mathbf{c}_{\text{bg}}$ represents a background color that combines with the foreground scene based on the remaining transmittance $\mathcal{T}(D)$.

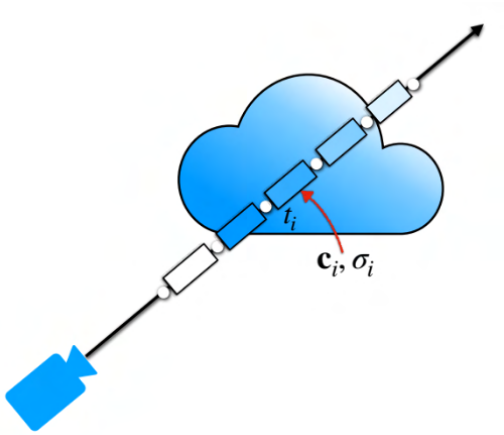


Figure 2-7: We assume **Homogeneous Volumetric Media** with constant color $\mathbf{c}_a$ and density σ_a over a ray segment. Images taken from [68].

We can calculate the expected Color over a ray segment $[a, b]$ by integrating (2.20):

$$\boldsymbol{C}(a \to b) = \int_a^b \mathcal{T}(a \to t) \cdot \sigma(t) \cdot \mathbf{c}(t)\, dt \qquad \text{omit the background term} \tag{2.21}$$

$$= \sigma_a \cdot \mathbf{c}_a \int_a^b \mathcal{T}(a \to t)\, dt \qquad \text{constant density/radiance} \tag{2.22}$$

$$= \sigma_a \cdot \mathbf{c}_a \int_a^b \exp\left(-\int_a^t \sigma(u)\, du\right)\, dt \qquad \text{substituting (2.19)} \tag{2.23}$$

$$= \sigma_a \cdot \mathbf{c}_a \int_a^b \exp\left(-\sigma_a u\big|_a^t\right)\, dt \qquad \text{constant density (again)} \tag{2.24}$$

$$= \sigma_a \cdot \mathbf{c}_a \int_a^b \exp\left(-\sigma_a(t-a)\right)\, dt \tag{2.25}$$

$$= \sigma_a \cdot \mathbf{c}_a \cdot \frac{\exp\left(-\sigma_a(t-a)\right)}{-\sigma_a}\Bigg|_a^b \tag{2.26}$$

$$= \mathbf{c}_a \cdot \left(1 - \exp\left(-\sigma_a(b-a)\right)\right) \tag{2.27}$$

Transmittance is multiplicative. Probability of the ray that does not hit any particles within segment [a, c] is the product of the probabilities of the two independent events that it does not hit any particles within [a, b] or within [b, c].

$$\mathcal{T}(a \to c) = \mathcal{T}(a \to b) \cdot \mathcal{T}(b \to c) \tag{2.28}$$

Given a set of piecewise constant intervals $\{[t_n, t_{n+1}]\}_{n=1}^N$ with constant density σ_n within the n-th segment, and with t_1=0 and δ_n=t_{n+1}−t_n, transmittance is equal to:

$$\mathcal{T}_n = \mathcal{T}(t_n) = \mathcal{T}(0 \to t_n) = \exp\left(-\int_0^{t_n} \sigma(t)\, dt\right) = \exp\left(\sum_{k=1}^{n-1} -\sigma_k \delta_k\right) \tag{2.29}$$

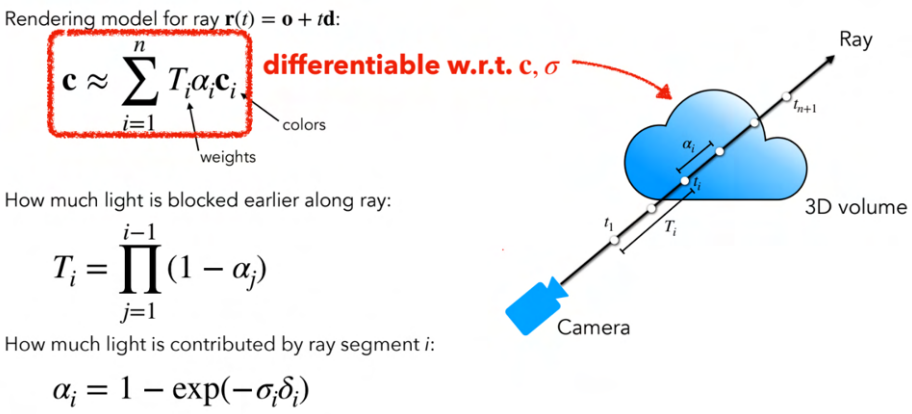


Figure 2-8: Illustration of three important elements from volume rendering equations. Images taken from [68].

Now, we can evaluate the volume rendering integral through a homogeneous medium with piecewise constant color and density:

$$\boldsymbol{C}(t_{N+1}) = \sum_{n=1}^{N} \int_{t_n}^{t_{n+1}} \mathcal{T}(t) \cdot \sigma_n \cdot \mathbf{c}_n \, dt \qquad \text{piecewise constant} \tag{2.30}$$

$$= \sum_{n=1}^{N} \int_{t_n}^{t_{n+1}} \mathcal{T}(0 \to t_n) \cdot \mathcal{T}(t_n \to t) \cdot \sigma_n \cdot \mathbf{c}_n \, dt \qquad \text{from (2.28)} \tag{2.31}$$

$$= \sum_{n=1}^{N} \mathcal{T}(0 \to t_n) \int_{t_n}^{t_{n+1}} \mathcal{T}(t_n \to t) \cdot \sigma_n \cdot \mathbf{c}_n \, dt \qquad \text{constant} \tag{2.32}$$

$$= \sum_{n=1}^{N} \mathcal{T}(0 \to t_n) \cdot (1 - \exp(-\sigma_n (t_{n+1} - t_n))) \cdot \mathbf{c}_n \qquad \text{from (2.27)} \tag{2.33}$$

Now, we get the volume rendering equations from NeRF [48]:

$$\boldsymbol{C}(t_{N+1}) = \sum_{n=1}^{N} \mathcal{T}_n \cdot (1 - \exp(-\sigma_n \delta_n)) \cdot \mathbf{c}_n, \quad \text{where} \quad \mathcal{T}_n = \exp\left(\sum_{k=1}^{n-1} -\sigma_k \delta_k \right) \tag{2.34}$$

Finally, we can express 2.34 in terms of alpha-compositing weights $\alpha_n \equiv 1 - \exp(-\sigma_n \delta_n)$,

and by noting that $\prod_i \exp x_i = \exp\left(\sum_i x_i\right)$ in (2.29):

$$\boldsymbol{C}(t_{N+1}) = \sum_{n=1}^{N} \mathcal{T}_n \cdot \alpha_n \cdot \mathbf{c}_n, \quad \text{where} \quad \mathcal{T}_n = \prod_{n=1}^{N-1} (1 - \alpha_n) \tag{2.35}$$

2.6 Structure from Motion

Before we discuss Structure from Motion, let's start by explaining the basic pinhole camera model, which describes how images are formed.

2.6.1 Pinhole Camera Model

The pinhole camera model provides a simplified representation of how light interacts with a camera to form an image. The following figure shows how a pinhole camera works:

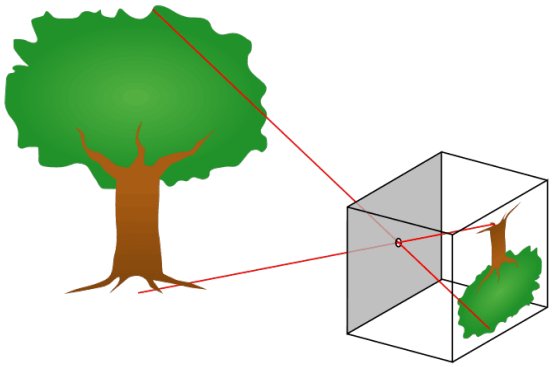


Figure 2-9: Operating principle of the pinhole camera. Images taken from [1].

As shown in Figure 2-9, in this model, the camera is depicted as a basic box with a small hole on one side and a flat image plane on the opposite side. This pinhole camera model is defined by the following fundamental parameters:

- The *center of projection* refers to the point at which the pinhole aperture is positioned.
- The *focal length* is the distance from the center of projection to the image plane.
- The *optical axis* is a line that is perpendicular to the image plane and passes through the center of projection.
- The *principal point* is the intersection where the optical axis meets the image plane.

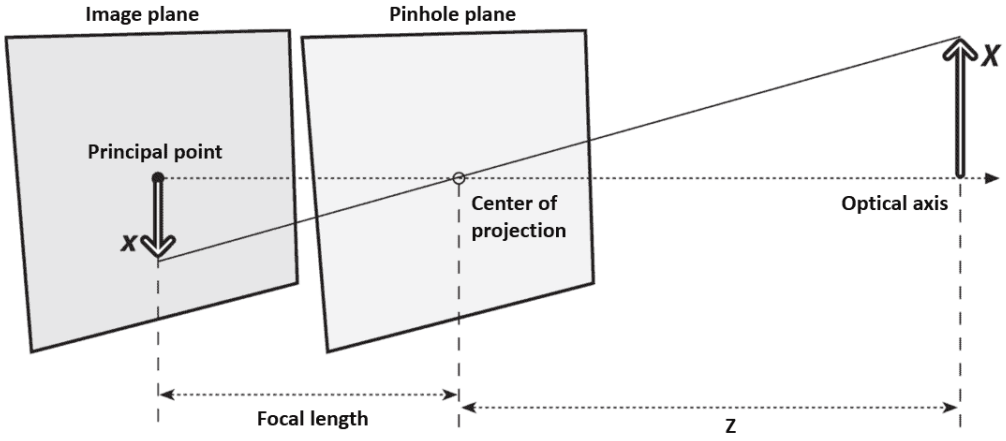


Figure 2-10: The projection geometry and the parameters of the pinhole camera. Images taken from [1].

As shown in Figure 2-10, we notice two similar triangles, which allows us to derive a simple equation that relates the size of an object to the size of its projected image:

$$-x = f\frac{X}{Z} \tag{2.36}$$

where X is the actual size of the object, x is the size of the object on the image plane, f is the focal length, and Z is the distance from the object to the pinhole aperture. The negative sign indicates that the image is inverted. Therefore, it is often preferable to conceptually swap the positions of the pinhole and the image plane, as illustrated in the following figure:

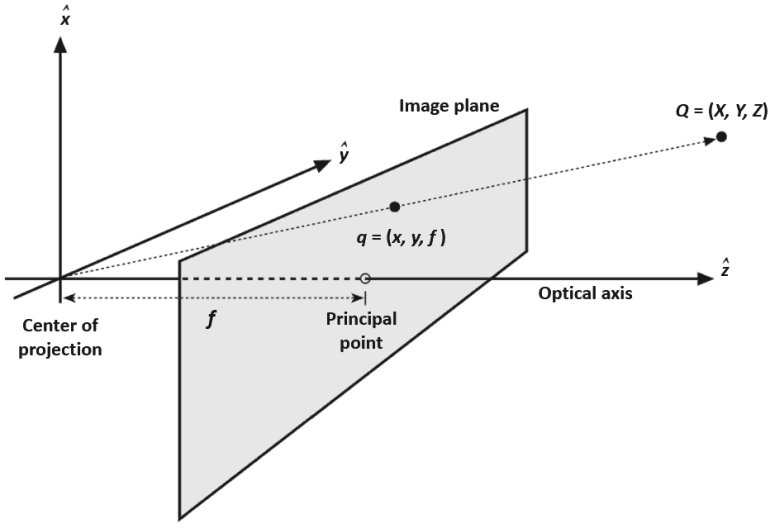


Figure 2-11: Swapping the pinhole and the image plane of the pinhole camera. Images taken from [1].

As shown in the Figure 2-11, image plane is ideally positioned in front of the pinhole. A point in space, $Q = (X, Y, Z)$, is projected onto the image plane by drawing a line through the point Q and the center of projection. The resulting projection of Q on the image plane is $q = (x, y, f)$. In this configuration, the object appears upright, and it also simplifies the mathematical calculations. The relationship between a point Q=(X,Y,Z) in real space and its projection onto the image plane at the pixel location (x_s, y_s) is described by the following equations:

$$x_s = f_x \frac{X}{Z} + c_x \tag{2.37}$$

$$y_s = f_y \frac{Y}{Z} + c_y \tag{2.38}$$

Here, c_x and c_y are parameters that account for any misalignment of the principal point with the image center, and f_x and f_y are the focal lengths expressed in pixels.

2.6.2 SfM

The advancements in lightweight and compact digital cameras, combined with ongoing enhancements in computer vision algorithms, have transformed traditional photogrammetry methods. This has led to the emergence of a new digital photogrammetry approach based on Structure-from-Motion (SfM). This approach offers exceptional spatial resolution while maintaining low computational costs.

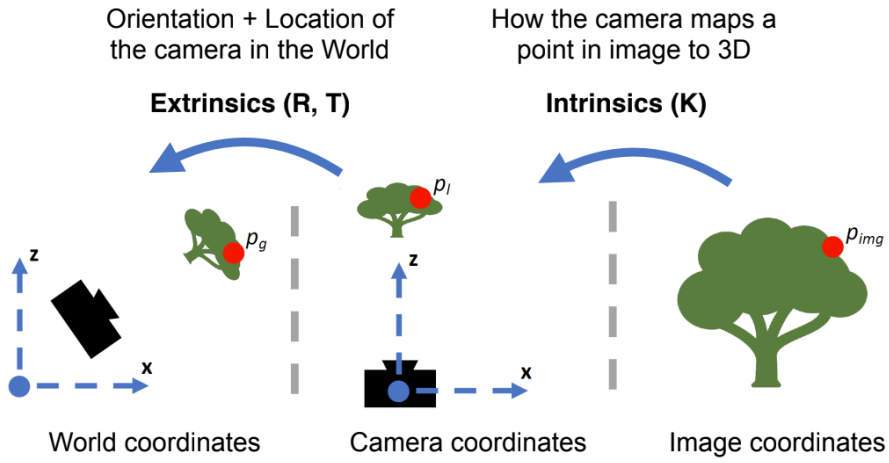


Figure 2-12: Pipeline for going from Camera coordinate system to World coordinate system. Images taken from [68].

Intrinsic orientation parameters encompass the optical properties of the camera, such as focal length, principal point, skew coefficient, and lens distortion coefficients. On the other hand, the extrinsic orientation parameters denote the 3D position and ori-

entation of the camera at the time of image acquisition, as shown in Figure 2-12. We use COLMAP [59] to find camera poses of corresponding images to create a dataset.

SfM is a photogrammetric technique, increasingly utilized for generating high resolution mapping products, such as point clouds and orthoimages, from imagery captured using affordable, consumer-grade cameras with adequate endlap and sidelap. The general workflow for SfM photogrammetry involves several steps, as illustrated in Figure 2-13. Initially, key features are automatically extracted from the acquired images. These features are then characterized using multidimensional descriptors like SIFT [41], which facilitates feature matching based on multidimensional maximum likelihood and outlier rejection criteria. Subsequently, bundle adjustment [74] is performed to simultaneously determine the intrinsic and extrinsic orientation parameters of the camera, resulting in the generation of a sparse point cloud. The sparse point cloud undergoes a densification process to produce depth maps using a technique known as multi-view stereopsis (MVS) [15]. These depth maps can then be utilized for rendering various outputs such as mesh surfaces, point clouds, etc.

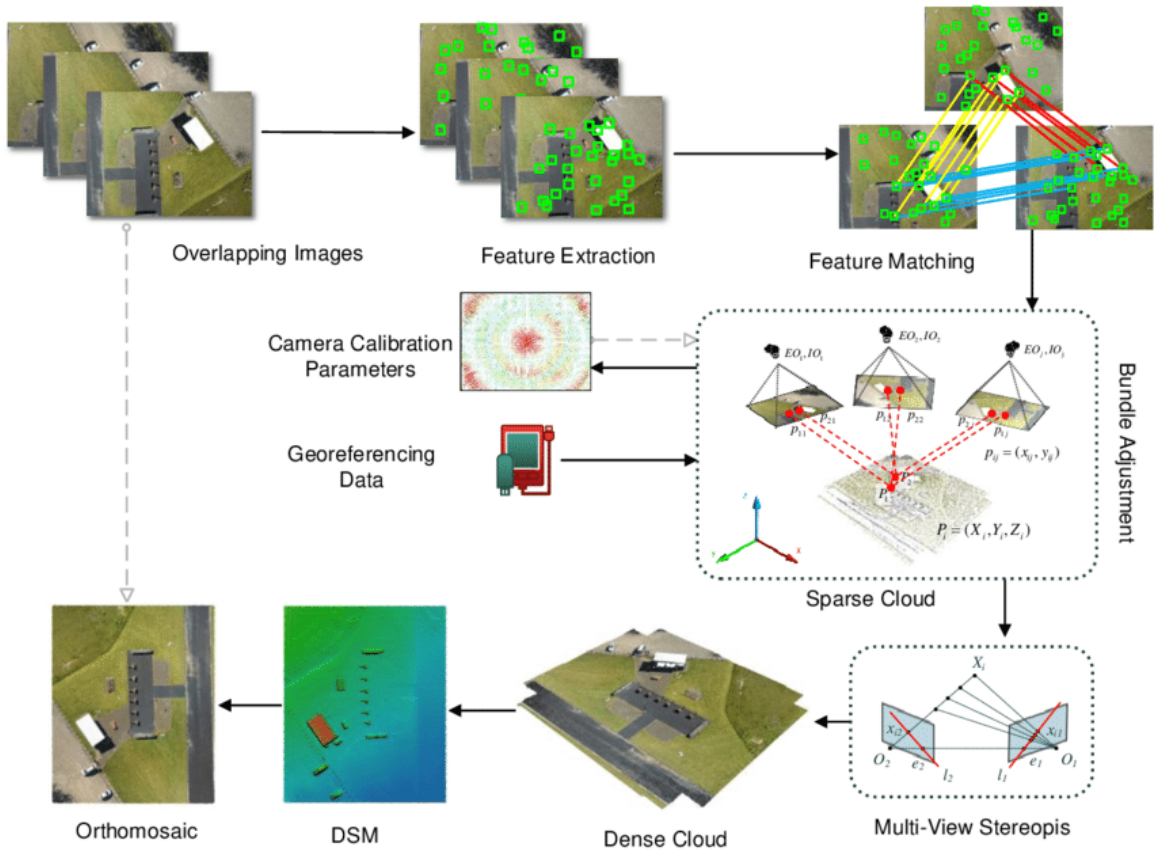


Figure 2-13: Workflow of SfM Photogrammetry. Images taken from [27].

2.6.3 COLMAP

There are many SfM methods such as, incremental [64], hierarchical [17], and global [66] approaches. Incremental SfM is arguably the most commonly used method for reconstruction from sequences of photo collections. A recent study by Schoenberger et al. [59] introduces COLMAP, an incremental SfM technique that incorporates a geometric verification strategy. This enhances the scene graph with additional information, thereby improving the robustness of the initialization and triangulation process.

COLMAP [59] proposes an iterative approach involving bundle adjustment, retriangulation, and outlier filtering to enhance completeness and accuracy by addressing drift effects. Additionally, they introduce a more efficient parameterization for bundle adjustment in dense photo collections through redundant view mining. COLMAP [59] surpasses previous state-of-the-art reconstruction techniques in terms of computational costs, robustness and completeness while maintaining efficiency.

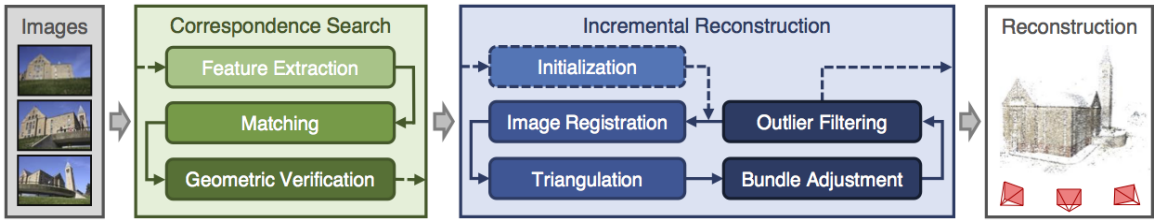


Figure 2-14: Workflow of COLMAP - an increamental SfM technique for efficient reconstruction [59]. Images taken from [59].

2.7 Spherical Harmonics

Now, we will discuss how Spherical Harmonics (SHs) are designed and used in calculating color values and encoding view directions within Radiance Field methods.

A spherical harmonic function, is a function defined on the surface of a sphere, which takes the angle (θ: polar angle, ϕ: azimuth angle) from the spherical coordinate system as shown in Figure 2-15 and outputs the value at the surface position (P) of the sphere.

When we solve the *Laplace equation* in the spherical coordinate system, we get the following formula [19]:

$$
\begin{aligned}
Y_l^m(\theta,\phi) &= \sqrt{\frac{(2l+1)(l-|m|)!}{4\pi(l+|m|)!}} P_l^{|m|}(\cos\theta) e^{im\phi} \\
P_\ell^{(|m|)}(\cos\theta) &= (-1)^m \frac{(\ell+|m|)!}{(\ell-|m|)!} P_\ell^{(-|m|)}(\cos\theta)
\end{aligned}
\tag{2.39}
$$

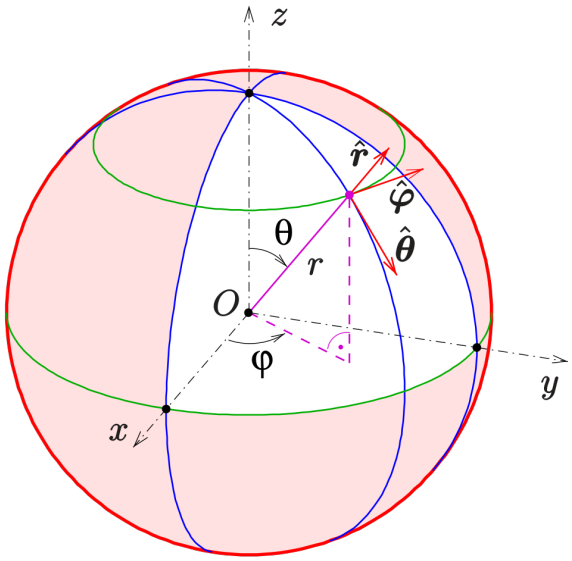


Figure 2-15: Spherical Coordinate System. Images taken from [78].

By changing the values of m and l in this formula 2.39, and $-l \leqslant m \leqslant l$, we get multiple SH functions Y and each Y function takes an angle (θ, ϕ) as input. l bundles functions of the same type, also known as band. If l is 0, it is called $L0$ band, and if l is 2, it is called $L2$ band. Following are the SH functions till $L2$ band 2.40 and graphical representation of SH functions till $L3$ band is shown in Figure 2-16:

$$
\begin{aligned}
Y_0^0(\theta,\phi) &= \frac{1}{2}\sqrt{\frac{1}{\pi}} \\
Y_1^{-1}(\theta,\phi) &= \frac{1}{2}\sqrt{\frac{3}{2\pi}} \cdot e^{-i\phi} \cdot \sin\theta \\
Y_1^0(\theta,\phi) &= \frac{1}{2}\sqrt{\frac{3}{\pi}} \cdot \cos\theta \\
Y_1^1(\theta,\phi) &= -\frac{1}{2}\sqrt{\frac{3}{2\pi}} \cdot e^{i\phi} \cdot \sin\theta \\
Y_2^{-2}(\theta,\phi) &= \frac{1}{4}\sqrt{\frac{15}{2\pi}} \cdot e^{-2i\phi} \cdot \sin^2\theta \\
Y_2^{-1}(\theta,\phi) &= \frac{1}{2}\sqrt{\frac{15}{2\pi}} \cdot e^{-i\phi} \cdot \sin\theta \cdot \cos\theta \\
Y_2^0(\theta,\phi) &= \frac{1}{4}\sqrt{\frac{5}{\pi}} \cdot \left(3\cos^2\theta - 1\right) \\
Y_2^1(\theta,\phi) &= -\frac{1}{2}\sqrt{\frac{15}{2\pi}} \cdot e^{i\phi} \cdot \sin\theta \cdot \cos\theta \\
Y_2^2(\theta,\phi) &= \frac{1}{4}\sqrt{\frac{15}{2\pi}} \cdot e^{2i\phi} \cdot \sin^2\theta
\end{aligned}
\tag{2.40}
$$

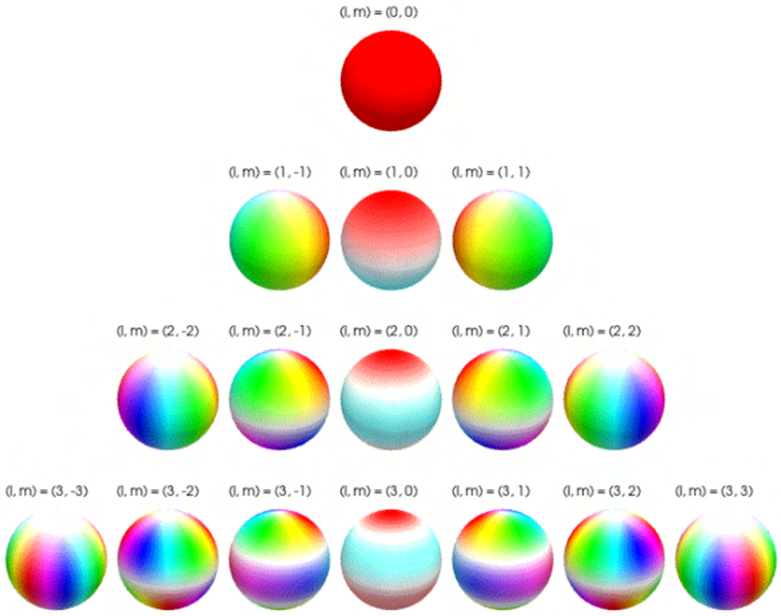


Figure 2-16: Graphical representation of SH functions till $L3$ band. Images taken from [84].

Now, we can represent colors using these coordinates obtained in the form of SH functions Y [19]:

$$C = \sum_{l=0}^{l_{\max}} \sum_{m=-l}^{l} k_l^m Y_l^m(\theta, \phi) \tag{2.41}$$

Where, color C is calculated by weighted sum with a weight k for the function Y. To understand this, we can imagine SH function Y as a color palette, and each palette has a weight. As the input angle varies, the SH function Y generates different color values, which are then blended based on the assigned weights to create a final color output C as shown Figure 2-17.

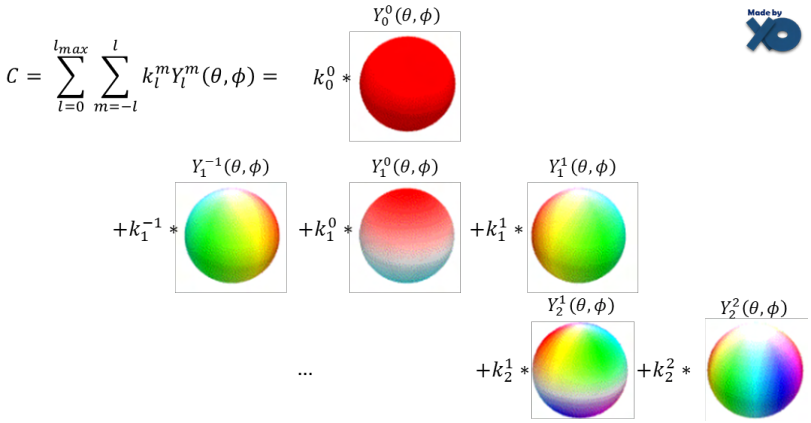


Figure 2-17: View dependent color computation in 3D scene representation. Images taken from [84].

2.8 Perspective Transformation

Here we discuss how to obtain perspective transformation of given image using given rotations and vertical field-of-view, we refer this derivation from here [26]. We want to find the general perspective transform $\mathcal{F}$ of a quadrilateral,

$$M = \mathcal{F}\left(w_p, h_p, \theta, \phi, f_V\right)$$

which produces a 4×4 warp matrix M by which to transform the vertices of an image of width w_p and height h_p pixels, given a camera axis rotation θ, out-of-plane (tilt) rotation ϕ and vertical field-of-view f_V.

More specifically, we seek

$$\boxed{\mathcal{F} = PTR_\phi R_\theta}$$

where R_θ is the rotation matrix around the z-axis, R_ϕ is the rotation matrix around the x-axis, T is the translation matrix that displaces the coordinate system down the z-axis and P is the projection matrix for a square viewport with vertical field-of-view f_V.

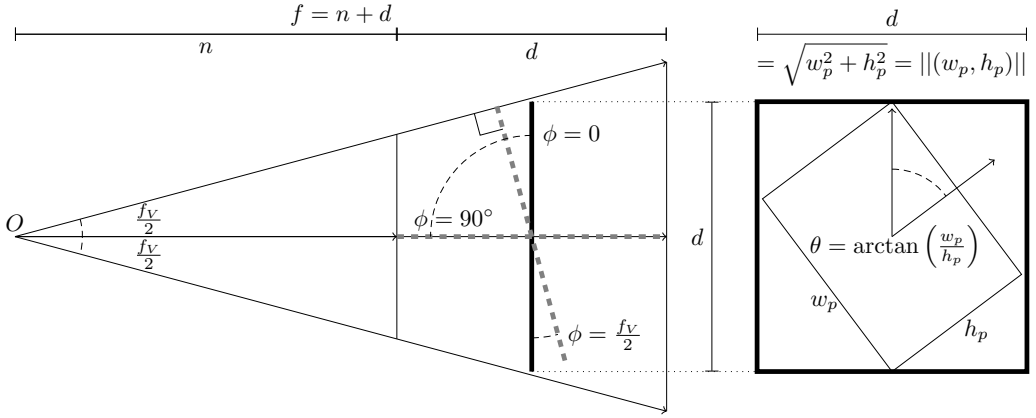


Figure 2-18: Illustration of the relationships between w_p, h_p, θ, ϕ and f_V for the particular configuration (Tilt equal to $\frac{f_V}{2}$, Rotation equal to arctan $\left(\frac{w_p}{h_p}\right)$) which results in the biggest image to process (The extreme case). Image taken from [26]

Let us assign to each corner of the image a coordinate in the object coordinate system as follows, assuming the convention that the observer faces down the negative z-axis, with the positive y-axis pointing up and the positive x-axis pointing to the right:

Corner	Object Coordinate System
Top Left	$\left(-\frac{w_p}{2}, \frac{h_p}{2}, 0\right)$
Top Right	$\left(\frac{w_p}{2}, \frac{h_p}{2}, 0\right)$
Bottom Right	$\left(\frac{w_p}{2}, -\frac{h_p}{2}, 0\right)$
Bottom Left	$\left(-\frac{w_p}{2}, -\frac{h_p}{2}, 0\right)$

R_θ and R_ϕ are trivial to find; they are

$$R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and

$$R_\phi = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We use rotation matrix R calculated using COLMAP from respective pose. To find T, we first define a variable d representing the side length of the square that would wholy contain any rotation of the image. This is plainly

$$d = \sqrt{w_p^2 + h_p^2}$$

Now, given f_V, we solve for the hypotenuse of a right triangle with an opposite side of length $\frac{d}{2}$ subtending an angle $\frac{f_V}{2}$. This measure we will denote h.

$$h = \frac{d}{2\sin\left(\frac{f_V}{2}\right)}$$

It is h that describes by how much to translate the object coordinate system down the z-axis. The matrix T is therefore defined as:

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -h \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -\frac{d}{2\sin\left(\frac{f_V}{2}\right)} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The perspective matrix P, relies on a few more assumptions. Let us define two more values, n and f, based on d and h computed above:

$$n = h - \frac{d}{2}$$

$$f = h + \frac{d}{2}$$

These represent the distances to the near plane and far plane, respectively. Given that the aspect ratio is 1 (we are rotating the square in which the image is embedded), we can now define a perspective matrix P as follows:

$$P = \begin{bmatrix} \frac{n}{n\tan\left(\frac{f_V}{2}\right)} & 0 & 0 & 0 \\ 0 & \frac{n}{n\tan\left(\frac{f_V}{2}\right)} & 0 & 0 \\ 0 & 0 & -\frac{(f+n)}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

$$P = \begin{bmatrix} \cot\left(\frac{f_V}{2}\right) & 0 & 0 & 0 \\ 0 & \cot\left(\frac{f_V}{2}\right) & 0 & 0 \\ 0 & 0 & -\frac{(f+n)}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

One additional aspect requires further explanation: the dimensions of the resulting image. A desirable property of the warp is that, for the special case $\theta = 0, \phi = 0$, it should leave the image unchanged (same size, no distortion), except for centering it. This can be accomplished if the viewport transform maps to a square image of side length l, where l is defined as:

$$l = d\sec\left(\frac{f_V}{2}\right)$$

This follows from the fact that the distance from the center point of the image to the point directly above it on the upper plane of the viewing frustum is $h\tan\left(\frac{f_V}{2}\right)$, and the square side length is twice that distance since it also extends downwards to the lower plane. Hence,

$$l = 2h\tan\left(\frac{f_V}{2}\right) = 2\frac{d}{2\sin\left(\frac{f_V}{2}\right)}\tan\left(\frac{f_V}{2}\right) = d\frac{\tan\left(\frac{f_V}{2}\right)}{\sin\left(\frac{f_V}{2}\right)} = d\sec\left(\frac{f_V}{2}\right)$$

Chapter 3

Related Work

In this section, we discuss some of the well-known SOTA radiance field methods such as NeRF [48], Instant-NGP [51], Mip-NeRF [3], NeRFacto [70], Ref-NeRF [76], and 3D-GS [32]; and we evaluate these radiance field methods on our synthetic dataset called StructColorToaster scene. The reasons for choosing these SOTA radiance field methods are: available related existing work to solve the problem at hand, what they can do, and our need to grasp how radiance field methods work overall.

3.1 NeRF

NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis [48] method comes under the category of Implicit Neural Representations based Novel View Synthesis discussed in Section 2.3.7. Using the original NeRF model might not be the best option for many tasks; however, it is important to grasp the fundamentals of how NeRF functions, as many subsequent models follow a similar framework.

At the heart of NeRF is an implicit MLP-based model. It takes in 5D vectors, which include 3D coordinates along with 2D viewing directions, and gives out values for opacity (α) and color (c). These vectors are determined by training the model using a sparse set of different views of the scene. Once trained, this model can render new views of the scene using the traditional volume rendering technique discussed in Section 2.5.

To sample a 5D coordinates along camera rays, we map from the (camera, pixel) to ray in camera coordinate frame as shown in the Figure 3-1. Following full mapping is used to get 3D coordinates in camera coordinate space for a point (i, j) on the image plane:

$$(i, j) \rightarrow \left(\frac{i - w/2}{f}, \frac{j - h/2}{f}, -1 \right) \tag{3.1}$$

Then simply apply rigid rotation and translation to 3D coordinates obtained in cam-

era coordinate space to get 3D coordinates in world space, as shown in Figure 3-1.

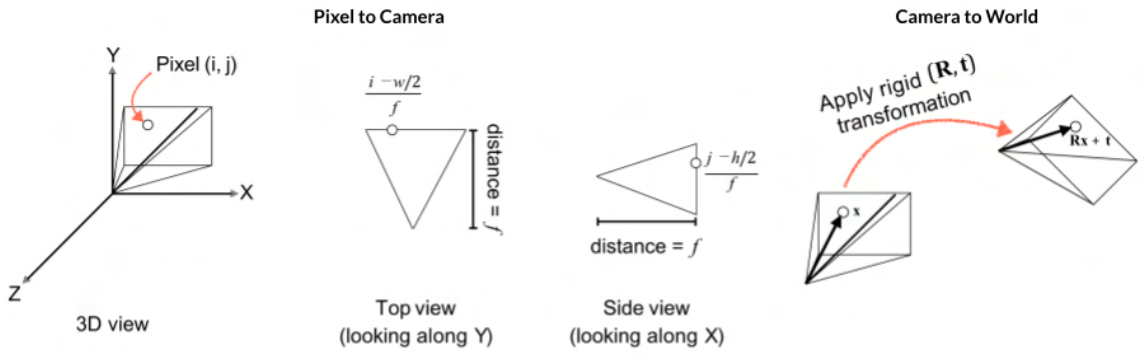


Figure 3-1: Sampling 5D coordinates along camera rays. Images based on [68].

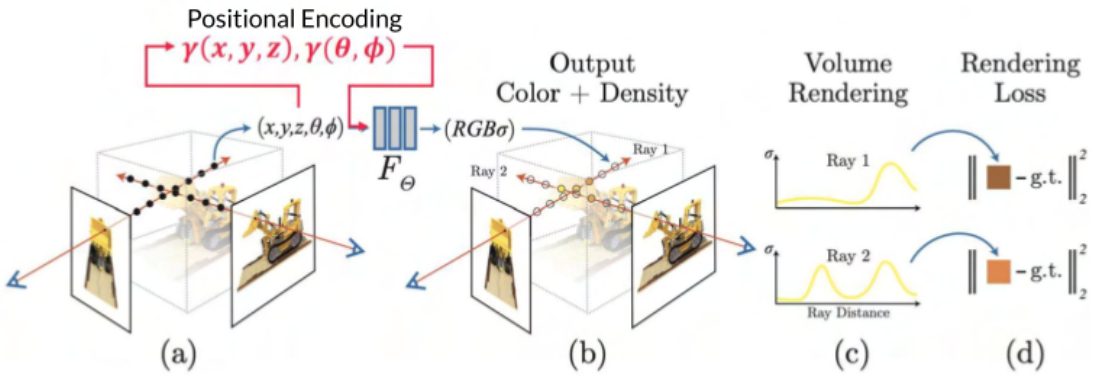


Figure 3-2: Pipeline of NeRF for neural scene representation. (a) Sample 5D coordinates along camera rays, (b) Feed those locations into an MLP to produce a color and volume density, (c) Use volume rendering techniques to composite these values into an image, (d) Optimize our scene representation by minimizing the residual between synthesized and ground-truth observed images. Images based on [48].

As shown in the Figure 3-2, in NeRF, the model approximates the continuous 5D input (x, y, z, θ, ϕ) representing the scene using an MLP network F_θ, we optimize the weights θ of this network to make it map each 5D input coordinate to the corresponding view-dependent color c and density γ of the scene at that point. Then, we composite all these color c and density γ values to make an actual image using volume rendering techniques discussed in Section 2.5:

$$\mathbf{C} \approx \sum_{i=1}^{N} T(t_i).\alpha_i.\mathbf{c}(\mathbf{r}(t_i)) \tag{3.2}$$

where,

$$\alpha_i = 1 - e^{-\sigma(\mathbf{r}(t_i))\delta}$$
$$T(t_i) = \prod_{j=1}^{i} 1 - \alpha_i \tag{3.3}$$

Once the image is created, its rendering error is calculated for each pixel. In this SOTA method, the researchers have employed two primary techniques to minimize this error and create better quality renderings, namely *positional encoding* and *hierarchical volume sampling*.

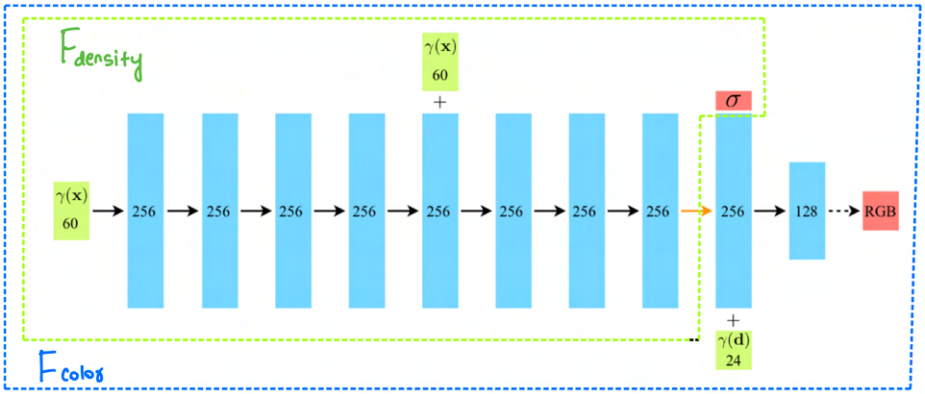


Figure 3-3: MLP used in NeRF to approximate the functions modelling color and density. Image from on [48].

When the network directly process input coordinates, it fails to represent high-frequency details in color and geometry. Tancik et. al. [69] showed that Fourier features let networks learn high frequency functions in low dimensional domains, therefore under positional encoding technique authors used following encoding function to map the 5D input coordinates from $\mathbb{R}$ to higher dimensional space $\mathbb{R}^2 L$:

$$\gamma(p) = (sin(2^0 \pi p), cos(2^0 \pi p), ..., sin(2^{L-1} \pi p), cos(2^{L-1} \pi p)) \tag{3.4}$$

The points t_i are positioned along the camera ray and are selected from a uniform distribution arranged in a structured way to ensure better coverage while training. Additionally, under hierarchical volume sampling technique, authors opt to refine two separate networks: one coarse and one fine. The coarse network gathers data from the radiance field as previously outlined, while the fine network selects t_i points based on earlier weights, $p_i = \frac{T_i \sigma_i}{\sum_i T_i \sigma_i}$. This probability density function (PDF) sampler utilizes these weights to produce a new batch of samples that are inclined towards areas with greater weight, and therefore these areas typically lie close to the object's surface. The model's parameters are optimized using ADAM optimizer with the following loss function:

$$\mathcal{L} = \sum_{\mathbf{r} \in \mathcal{R}} \left[\left\| \hat{C}_c(\mathbf{r}) - C(\mathbf{r}) \right\|_2^2 + \left\| \hat{C}_f(\mathbf{r}) - C(\mathbf{r}) \right\|_2^2 \right] \tag{3.5}$$

where $\mathcal{R}$ is the set of rays in each batch, and $C(r)$, $\hat{C}_c(r)$, and $\hat{C}_f(r)$ are the ground truth, coarse RGB volume prediction, and fine RGB volume prediction for ray r respectively.

To address the limitations of NeRF, researchers are proposing many variants and these improvements could be classified into faster training & inference, with various research topics branching out into deformable & dynamic scene representation, generalization, pose estimation, lighting & material decomposition, editing, multiscale scene representation, and more.

3.2 Instant-NGP

Instant Neural Graphics Primitives with a Multiresolution Hash Encoding [51] method falls into the class of explicit representation with neural network, aiming at faster training and inference. Instant-NGP [51] divides NeRF [48] training into three main components and suggests enhancements for each to make real-time training of NeRFs possible. These components include as shown in Figure 3-4: (1) A smaller, fully-fused neural network, (2) A multi-resolution hash encoding, (3) A faster training and rendering algorithm through a ray marching scheme that utilizes an occupancy grid.

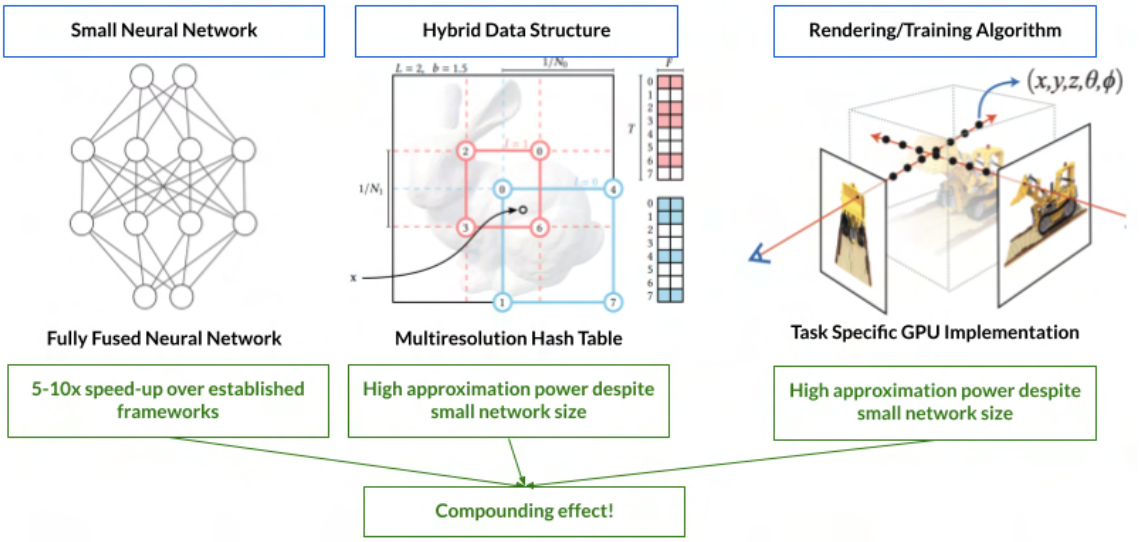


Figure 3-4: Compounding effect of Three pillars of Instant-NGP give speedup of 1000x which enables faster training and inference, in about few seconds. Images from [51].

The main idea behind the enhanced sampling method is to skip sampling in areas with no content and in areas behind high density regions. This is done by keeping track of multi-scale occupancy grids that roughly identify empty and non-empty space. Each space's occupancy is represented by a single bit, and if the occupancy is too low, a sample on a ray is skipped. These grids are kept separate from the trainable encoding and are adjusted during training according to the updated density predictions. The

authors discovered that this approach can speed up sampling by 10-100 times compared to simpler methods.

Another significant obstacle to NeRF's training speed has been the process of querying the neural network. In this study, the authors address this by designing the network to operate entirely within a single CUDA kernel. Additionally, they reduced the network to just four layers, each with 64 neurons. They demonstrate that their fully-fused neural network i.e. tiny-cuda-nn [50] is 5-10 times faster compared to an implementation using Tensorflow.

One important technique introduced by Instant-NGP is the multi-resolution hash encoding. Unlike traditional NeRF pipelines that utilize positional encoding functions such as Equation 3.4 to map input coordinates to higher-dimensional spaces, Instant-NGP suggests employing a trainable hash-based encoding. This method involves mapping the coordinates to learnable feature vectors that are optimized during the regular NeRF training process.

The learnable features that can be trained are $F - dimensional$ vectors, organized into L grids, each grid containing a maximum of T vectors. Here, L signifies the number of resolutions for features, while T represents the quantity of feature vectors within each hash grid. The Figure 3-5 shows the pipeline for hash grid encoding, steps are as follows:
(1) Define L-numbered d-dimensional grids (each grid a single level with N resolution. A geometric progression is used to define intermediate resolutions between the coarsest (N_{min}) to the finest (N_{max}) level:

$$\begin{aligned} N_l &:= \left\lfloor N_{\min} \cdot b^l \right\rfloor, \\ b &:= \exp\left(\frac{\ln N_{\max} - \ln N_{\min}}{L-1}\right) \end{aligned} \tag{3.6}$$

each level l is associated with up to T feature vectors with dimensionality F.
(2) Given an input coordinate, find the surrounding voxels at L resolution levels and hash the vertices of these grids using spatial hash function:

$$h(\mathbf{x}) = \left(\bigoplus_{i=1}^{d} x_i \pi_i\right) \mod T \tag{3.7}$$

where $\bigoplus$ denotes the bit-wise XOR operation and π_i are unique large prime numbers, d is dimension, T is the size of the hash table.
(3) The hashed vertices are used as keys to look up trainable F-dimensional feature vectors. The number of vertices for a given layer is $V = (N_l + 1)^d$ vertices. If $V \leqslant T$, we have a 1:1 mapping between the vertices of the level and the feature vectors. At finer levels, where $V > T$, we use a hash function h to map each d-dimensional vertex to one of the T feature vectors of the respective level.

(4) Based on where the coordinate lies in space, the feature vectors are linearly interpolated to match the input coordinate.
(5) The feature vectors from each grid are concatenated, along with any other parameters such as viewing direction. Instant-NGP encodes the viewing direction using spherical harmonic encodings.
(6) The final vector is fed into the MLP to predict the RGB and density output.

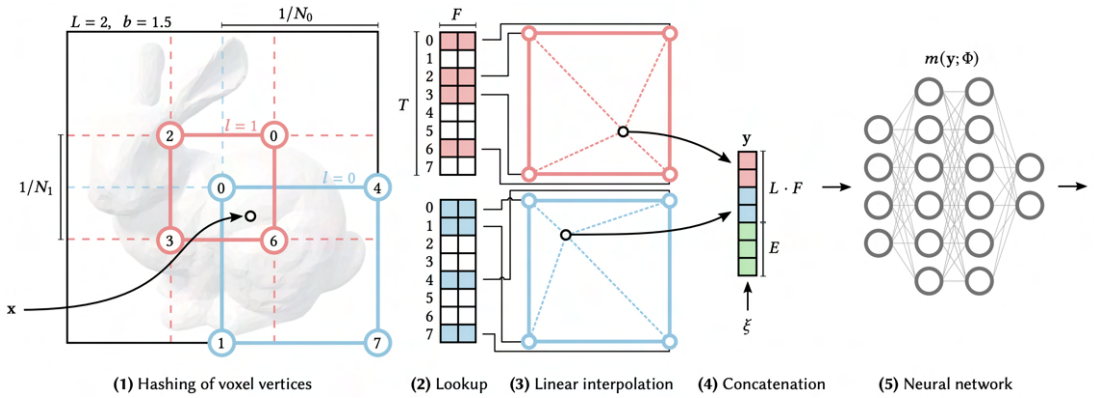


Figure 3-5: Illustration of the multiresolution hash encoding in 2D. (1) For a given input coordinate x, we find the surrounding voxels at the resolution levels L and assign indices to their corners by hashing their integer coordinates. (2) For all resulting corner indices, we look up the corresponding F-dimensional feature vectors from the hash tables θ_l, and (3) Linearly interpolate them according to the relative position of x within the respective l-th voxel. (4) we concatenate the result of each level, as well as auxiliary inputs $\xi \in \mathbb{R}^E$, producing the encoded MLP input $y \in \mathbb{R}^{LF+E}$, which (5) is evaluated at last. To train encoding, the loss gradients are backpropagated through MLP (5), concatenation (4), linear interpolation (3), and then accumulated in the looked-up feature vectors (2). Images from [51].

Steps 1 to 3 are carried out separately at every resolution level. Consequently, because these feature vectors are trainable, during the backward propagation of the loss gradient, the gradients will traverse through the neural network and interpolation function, reaching all the way back to the feature vectors. The feature vectors are interpolated with respect to the coordinates, enabling the network to learn a continuous function smoothly.

It's important to understand that this method doesn't deal with hash collisions directly. In each hash index, there might be several vertices pointing to the same feature vector. However, since these vectors are trainable, the vertices most significant for the particular output will have the strongest influence on optimizing that feature, as they'll have the highest gradient.

This encoding structure creates a tradeoff between quality, memory, and performance. The main parameters which can be adjusted are the hash table size (T), the size of the feature vectors (F), and the number of resolutions (L).

3.3 Mip-NeRF

Mip-NeRF: A Multiscale Representation for Anti-Aliasing Neural Radiance Fields [3] method falls into the category of implicit neural scene representation, with the objective of multiscale scene representation to reduce aliasing artifacts. Mip-NeRF has three main contributions: (1) Cone tracing, (2) Integrated Positional Encoding (IPE), and (3) a single multiscale MLP.

As we know NeRF [48] performs point sampling in the rendering procedure, which ignores the shape and size of the volume seen by each ray. Because of this, two different cameras imaging the same location at different scales can produce the same ambiguous point-sampled features, causing excessively blurred or aliased renderings, resulting in significant performance degradation. A simple way to solve this is to use multiple rays to render a single pixel, but it is takes a lot of time and resources. However, as shown in Figure 3-6 Mip-NeRF resolves this ambiguity by creating cones instead of rays and explicitly modeling the volume of each frustum.

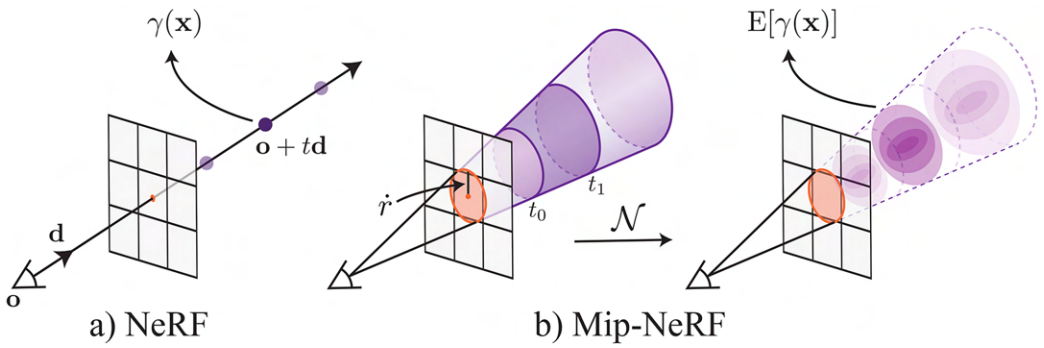


Figure 3-6: Different sampling strategies used in NeRF and Mip-NeRF. Images taken from [4].

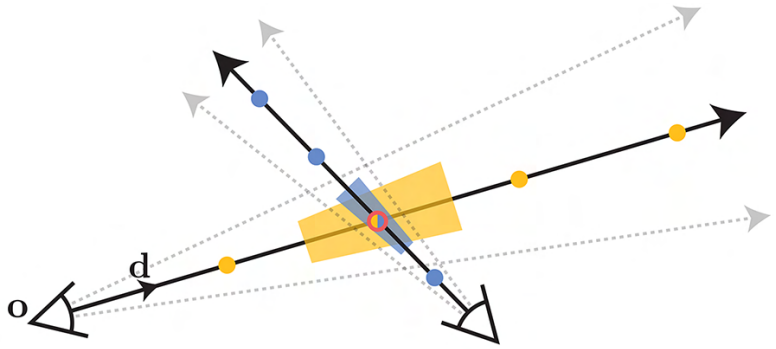


Figure 3-7: Two different cameras viewing the same point in space may result in vastly different conical frustums illustrated in 2D. Images taken from [4].

As shown in Figure 3-7, for each queried point along a ray, Mip-NeRF considers its associated 3D conical frustum. To create the input feature vector to the MLP,

Mip-NeRF fit the multivariate Gaussian to the conical frustum using *integrated positional encoding* (IPE). IPE considers Gaussian regions of space, rather than infinitesimal points. This provides a natural way to input a "region" of space as query to a coordinate-based neural network, allowing the network to reason about sampling and aliasing. This cone was approximated by a multivariate Gaussian, whose mean vector and variance matrix were derived to have the appropriate geometry, resulting in the IPE. The expected value of each positional encoding component has a simple closed form:

$$\begin{aligned}\gamma(\boldsymbol{\mu}, \boldsymbol{\Sigma}) &= \mathbb{E}_{\mathbf{x}\sim\mathcal{N}}(\boldsymbol{\mu}, \boldsymbol{\Sigma})[\gamma(\mathbf{x})] \\ &= \begin{bmatrix}\sin(\boldsymbol{\mu}_\gamma) \odot \exp(-(1/2)\mathrm{diag}(\boldsymbol{\Sigma}_\gamma)) \\ \cos(\boldsymbol{\mu}_\gamma) \odot \exp(-(1/2)\mathrm{diag}(\boldsymbol{\Sigma}_\gamma))\end{bmatrix}\end{aligned} \tag{3.8}$$

where $\boldsymbol{\mu}_\gamma, \boldsymbol{\Sigma}_\gamma$ are the means and variances of the multivariate Gaussian lifted onto the positional encoding basis with N levels.

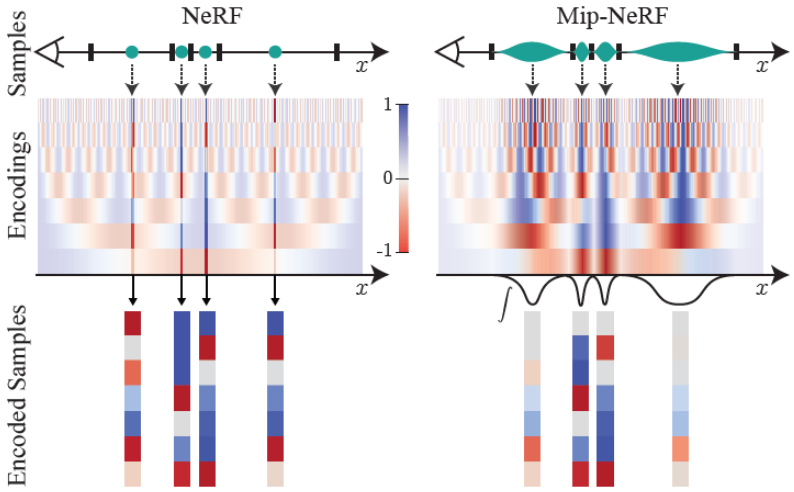


Figure 3-8: Toy 1D visualizations of the positional encoding (PE) used by NeRF (left) and integrated positional encoding (IPE) (right) used by Mip-NeRF. Images taken from [4].

As shown in Figure 3-8, in NeRF, points are sampled along each ray and all frequency components of the positional encoding are treated equally. This means that even high-frequency features are represented with the same importance as low-frequency ones. However, this equal treatment of frequencies can lead to a problem known as aliasing, which results in unwanted patterns or artifacts in the rendered images.

To address this issue, the authors propose an integrated positional encoding (IPE) approach. In this approach, the positional encoding features are integrated or combined over small intervals along the ray. When the frequency of the details in the scene is small compared to the size of the interval being integrated, the high-frequency components of the IPE features gradually diminish towards zero within that interval. This process effectively reduces the prominence of high-frequency details and

helps to smooth out the representation, resulting in anti-aliased features. Additionally, by considering higher dimensions, such as in 3D scenes, this integrated approach implicitly encodes not only the size but also the shape of the interval being integrated.

Following are steps for training and sampling in Mip-NeRF as shown in Figure 3-9:
(1) We start by creating evenly spaced coarse intervals along a ray, similar to dividing a histogram into sections.
(2) Next, we pass Gaussian distributions corresponding to each interval through a neural network, which gives us a colored histogram with weights w and colors c.
(3) These weights and colors are then averaged to generate the final color for that specific pixel using an alpha-compositing technique.
(4) After that, we re-sample these weights to create a new set of intervals, focusing more on areas with content in the scene.
(5) These new intervals are passed through the neural network to obtain weights and colors, which are used to determine the pixel's color.
(6) We optimize the hyperparameters of the MLP by minimizing a reconstruction loss between all rendered pixel values and the actual pixel colors from the input images.

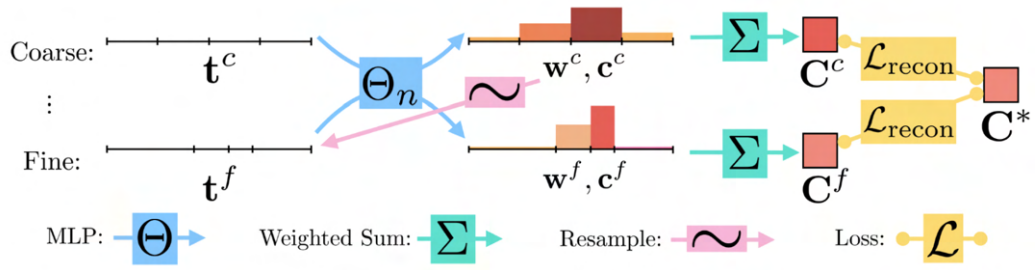


Figure 3-9: Pipeline of Mip-NeRF describing training and sampling procedure. Images taken from [54].

3.4 NeRFacto

NeRFacto [70] method falls into the category of implicit neural scene representation, which combines many techniques developed in different published works into the NeRFacto model. These techniques include *camera pose refinement, per image appearance conditioning, proposal sampling, scene contraction, and hash encoding.*

As shown in the Figure 3-10, the NeRFacto model consist of following main components:Pose Refinement, Piecewise Sampler, Proposal Sampler, Density Field, and NeRFacto Field.

Pose Refinement is used to minimize the error in the camera poses predicted using COLMAP. When poses are not properly aligned, it leads to cloudy artifacts and a loss of sharpness and detail in the scene. Pose Refinement framework refine and optimize

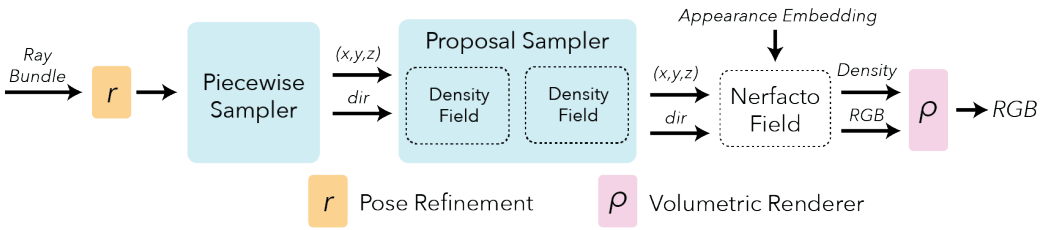


Figure 3-10: Pipeline of NeRFacto Method. Images taken from [70].

the poses by backpropagating loss gradients to the initial pose calculations.

Piecewise Sampler is used to generate the initial set of samples of the scene. This sampler evenly distributes half of the samples within a distance of 1 from the camera. The rest of the samples are distributed in a way that increases the step size with each sample. This step size selection ensures that the viewing frustums are scaled versions of each other. By increasing the step sizes, it is possible to sample distant objects while maintaining a dense set of samples for nearby objects.

Proposal Sampler concentrates sample points in the areas of the scene that have the greatest impact on the final render, usually where surfaces intersect first. This significantly enhances the quality of the reconstruction. To operate, the proposal network sampler needs a density function for the scene. Authors implemented this density function using a small fused-MLP with hash encoding which offers satisfactory accuracy and efficiency. Multiple density functions can be linked together with the proposal network sampler to further refine the sampling process. Through experimentation, authors determined that employing two density functions yields superior results compared to using just one. However, using more than two density functions does not yield significant improvements and can even lead to diminished returns.

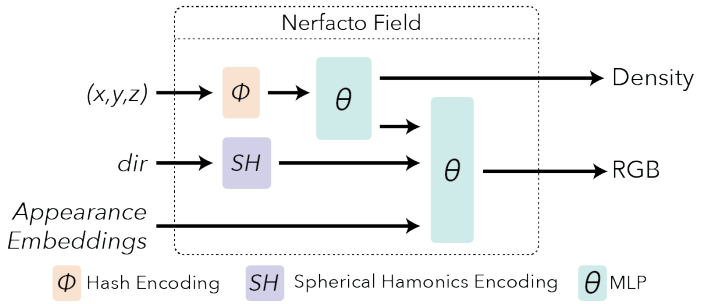


Figure 3-11: NeRFacto Field used in NeRFacto method. Images taken from [70].

Density Field doesn't have to precisely represent every detail of the scene; it just needs to give a rough idea to guide sampling. Using a hash encoding along with a small fused-MLP from tiny-cuda-nn [50] offers a quick method to query the scene. Authors further improve efficiency by reducing the size of the encoding dictionary and

the number of feature levels. These adjustments don't significantly affect the quality of the reconstruction because the density function doesn't need to capture fine details in the initial stages of training.

NeRFacto Field is used to produce final color and density as shown in Figure 3-11. The output sample locations and viewing directions are encoded using hash encoding and spherical harmonics encoding respectively. Two small fused-mlp's from [50] are used, one takes encoded sample locations as input to produce final density and other fed with encoded direction along with appearance embedding [43] to produce final color.

3.5 Ref-NeRF

Ref-NeRF: Structured View-Dependent Appearance for Neural Radiance Fields [76] is built on Mip-NeRF falls in the category of neural implicit representation, designed to better model the reflective surfaces. The main contributions of the Ref-NeRF are *Reflection Direction Parameterization* and *Integrated Directional Encoding*.

NeRF-based methods [48, 3, 4, 51, 70] are good at depicting representing complex geometric structures with smoothly changing view-dependent appearance, but these methods struggle to precisely represent the view-dependent appearance of glossy and reflective surfaces.

To address this, as shown in Figure 3-12 Ref-NeRF reparameterize the outgoing radiance as a function of the reflection of the view direction about the local normal vector:

$$\hat{\boldsymbol{\omega}}_r = 2(\hat{\boldsymbol{\omega}}_o \cdot \hat{\mathbf{n}})\hat{\mathbf{n}} - \hat{\boldsymbol{\omega}}_o \qquad \text{(Reflection Eq.)} \tag{3.9}$$

where $\hat{\boldsymbol{\omega}}_o = \hat{-\boldsymbol{d}}$ is a unit vector pointing from a point in space to the camera, and $\hat{\boldsymbol{n}}$ is the normal vector at that point. Then the view-dependent radiance function of the reflection direction $\hat{\boldsymbol{\omega}}_r$ is,

$$L_{\text{out}}(\hat{\boldsymbol{\omega}}_o) \propto \int L_{\text{in}}(\hat{\boldsymbol{\omega}}_i) p(\hat{\boldsymbol{\omega}}_r \cdot \hat{\boldsymbol{\omega}}_i) d\hat{\boldsymbol{\omega}}_i = F(\hat{\boldsymbol{\omega}}_r) \tag{3.10}$$

As shown in Figure 3-13, authors modified the density MLP into a directionless spatial MLP which outputs density τ and the input feature vector of the directional MLP such as diffuse color $\mathbf{C}_d$, specular tint $\mathbf{s}$, roughness ρ, bottleneck vector $\mathbf{b}$, and surface normal $\hat{\mathbf{n}}$. Then $\hat{\boldsymbol{\omega}}_r$ is computed using Reflection Eq. 3.9 and fed it to the Directional MLP along with roughness ρ to predict specular color $\mathbf{c}_s$. Finally single color value is obtained using,

$$\mathbf{c} = \gamma(\mathbf{c}_d + \mathbf{s} \odot \mathbf{c}_s) \tag{3.11}$$

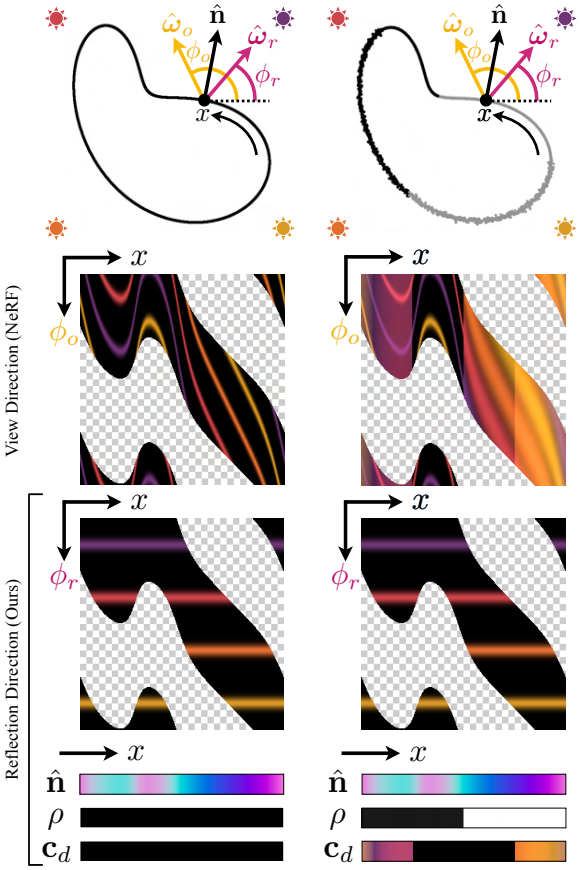


Figure 3-12: When NeRF based methods encounters glossy surfaces or spatially varying materials, then they need to interpolate between very complex function. While Ref-NeRF simplifies radiance function to represent shiny or spatially varying surfaces by reparameterizing radiance using a normal vector $\hat{\mathbf{n}}$ and a reflection angle ϕ_r, and its spatial MLP outputs additional diffuse color $\mathbf{C}_d$ and roughness ρ. Images taken from [76].

Authors propose the IDE which enables the directional MLP to represent reflected radiance functions for any continuously-valued roughness. Every component of the IDE is a spherical harmonic function Y_ℓ^m explained in Section 2.7 convolved with a von Mises-Fisher (vMF) distribution with concentration parameter $k = 1/\rho$ which is predicted by spatial MLP:

$$\text{IDE}(\hat{\boldsymbol{\omega}}_r, \kappa) = \mathbb{E}_{\hat{\boldsymbol{\omega}} \sim \text{vMF}(\hat{\boldsymbol{\omega}}_r, \kappa)}[Y_\ell^m(\hat{\boldsymbol{\omega}})] = A_\ell(\kappa) Y_\ell^m(\hat{\boldsymbol{\omega}}_r) \quad \text{(IDE Eq.)} \tag{3.12}$$

where,

$$\begin{aligned} A_\ell(\kappa) &\approx \exp\left(-\frac{\ell(\ell+1)}{2\kappa}\right) \\ (\ell, m)\colon \ell &= 1, ..., 2^L, m = 0, ..., \ell \end{aligned} \tag{3.13}$$

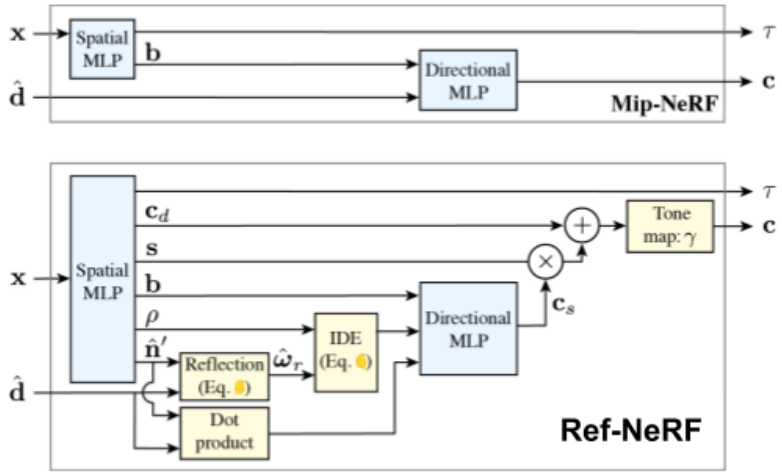


Figure 3-13: A comparison of Mip-NeRF's and Ref-NeRF's architectures. Images taken from [76].

Hence, areas with smoother surfaces are assigned higher-frequency encodings, whereas regions with rougher surfaces are given encodings with reduced high frequencies.

3.6 3D Gaussian Splatting

3D Gaussian Splatting for Real-Time Radiance Field Rendering method [32] falls under the category of explicit representation without neural network specifically point-based method aiming at computational efficiency and controllability. In contrast to neural methods such as Neural Radiance Fields (NeRF) [48, 3, 4, 51, 70, 76], which use neural networks conditioned on position and viewpoint to represent a 3D scene, 3D Gaussian Splatting (3D-GS) [32] employs Gaussian ellipsoids to model the scene. This approach enables real-time rendering by rasterizing these ellipsoids into images.

It remains challenging to train a NeRF-based method quickly ($\leqslant$1 hour) on a standard GPU accessible to consumers, while also rendering a 3D scene at an interactive frame rate of around 30 frames per second (FPS) on common devices such as smartphones and laptops. To address these speed challenges, 3DGS suggests rasterizing a set of Gaussian ellipsoids to approximate the appearance of a 3D scene, which facilitates low-cost 3D content creation and real-time applications.

3D-GS is the fusion of the implicit and explicit approaches and leverages strengths of both by utilizing 3D Gaussians as a flexible and efficient representation. These Gaussians are optimized to accurately represent the scene, combining the benefits of implicit representation [48] like optimization and explicit representation like unstructured data storage. This hybrid approach aims to achieve high-quality rendering capabilities while enhancing training speed and enabling real-time performance, es-

pecially for intricate scenes and high-resolution outputs. The formulation of the 3D Gaussian representation is as follows:

$$L_{\text{3DGS}}(x, y, z, \theta, \phi) = \sum_i G(x, y, z, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \cdot c_i(\theta, \phi), \tag{3.14}$$

where G is the Gaussian function with mean $\boldsymbol{\mu}_i$ and covariance $\boldsymbol{\Sigma}_i$, and c represents the view-dependent color.

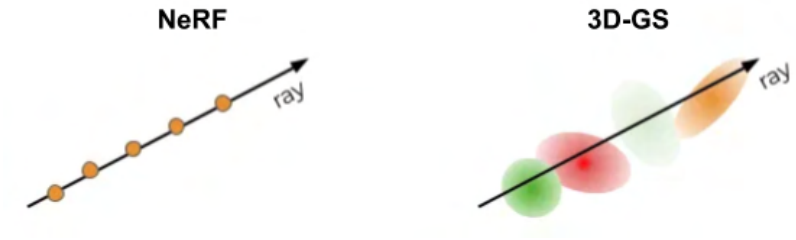


Figure 3-14: Fundamental difference between NeRF and 3D-GS. *NeRF*: Query a continuous MLP along the ray using volumetric ray marchiing, *3D-GS*: Blend a discrete set of Gaussians relevant to the given ray using splatting. Images taken from [92].

Point-based rendering [42] is a method used to depict 3D scenes by employing points instead of conventional polygons. These methods [89, 85, 2, 36] are particularly effective and efficient for rendering complex, unstructured, or sparse geometric data. However, these approaches can suffer from issues like holes in the rendering or aliasing effects. 3D-GS [32] expands on these ideas by incorporating anisotropic Gaussians to achieve a continuous and more unified depiction of the scene.

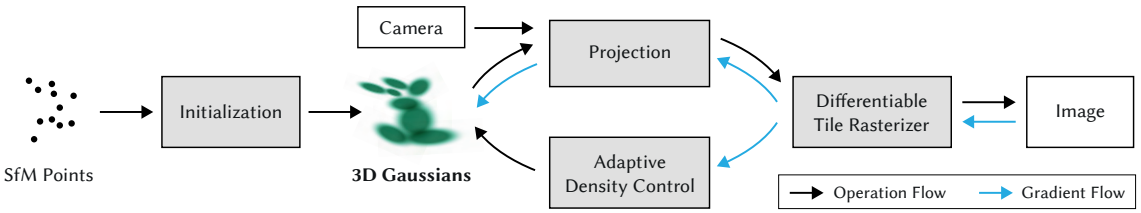


Figure 3-15: Optimization pipeline of 3D-GS. Images taken from [32].

In 3D-GS, a scene is represented by approximately millions of 3D Gaussians which are optimized using the pipeline shown in Figure 3-15. Each 3D Gaussian (Eq. 3.14) is parameterized by:

- Mean $\boldsymbol{\mu}$ interpretable as 3D location (x, y, z);
- Covariance $\boldsymbol{\Sigma}$;
- Opacity α, a sigmoid function is applied to map the parameter α to the [0, 1] interval;

- Color c parameters (R, G, B) are modelled as spherical harmonics (SH) coefficients (Section 2.7).

The covariance is deliberately designed to be *anisotropic covariance*. This implies that a 3D point can resemble an ellipsoid that is rotated and stretched along any direction in space which enables representing complex geometry compactly. Like most of the other parameters of 3D Gaussian, we can not directly optimize covariance matrix, as it will loose its physical interpretation if it no longer remains a positive semi-definite matrix. To address this issue during optimization using gradient descent, 3D-GS chooses to factorize $\boldsymbol{\Sigma}$ into a quaternion $\boldsymbol{q}$ and a 3D vector $\boldsymbol{s}$ representing rotation and scale respectively, as follows:

$$\Sigma = RSS^TR^T, \text{(Eigendecomposition of a covariance matrix)} \tag{3.15}$$

where, S is a diagonal scaling matrix with 3 parameters for scale, R is a 3×3 rotation matrix analytically expressed with 4 quaternions, derived from $\boldsymbol{s}$ and $\boldsymbol{q}$ respectively. The impact of a 3D Gaussian i on an arbitrary 3D point p in 3D can be defined using a probability density function of the multivariate normal distribution as follows:

$$f_i(p) = \sigma(\alpha_i)\exp(-\frac{1}{2}(p-\mu_i)\Sigma_i^{-1}(p-\mu_i)) \tag{3.16}$$

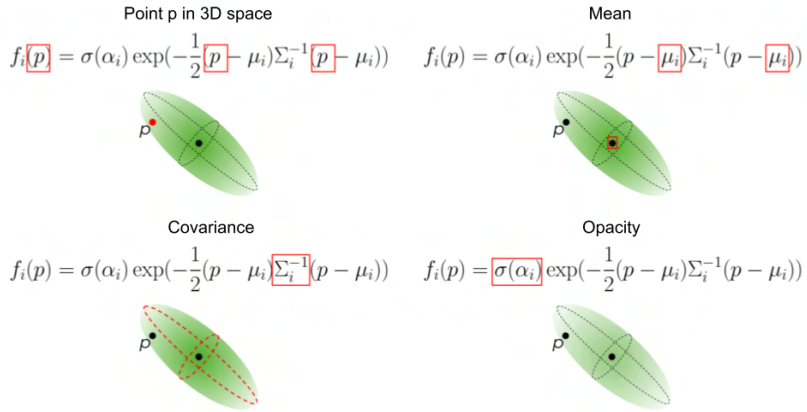


Figure 3-16: An influence of a 3D Gaussian i on a point p in 3D. Images taken from [92].

Image formation model used in 3D-GS and NeRF based methods is the same. In NeRFs, differential volumetric rendering (Section 2.5) is used for point-wise α-blending as follows:

$$C(p) = \sum_{i=1}^{N} c_i\alpha_i \underbrace{\prod_{j=1}^{i-1}(1-\alpha_j)}_{transmittance} \tag{3.17}$$

From above equation, we can interpret the final color as a weighted sum of colors of 3D points sampled along this ray, and down-weighted by transmittance in point-based approach [35] as follows:

$$C(p) = \sum_{i \in N} c_i f_i^{2D}(p) \underbrace{\prod_{j=1}^{i-1}(1 - f_j^{2D}(p))}_{transmittance}, \tag{3.18}$$

where, function $f^{2D}(p)$ is a projection of $f(p)$ (Eq. 3.16) onto 2D image plane of the camera which is being rendered. Both a 3D point and its projection follow multivariate Gaussian distributions. Thus, calculating the effect of a projected 2D Gaussian on a pixel uses the same formula (Eq. 3.18) as the impact of a 3D Gaussian on other points in 3D. But while calculating $f^{2D}(p)$, the projected mean μ and covariance Σ into 2D are used, which are calculated using EWA splatting [96].

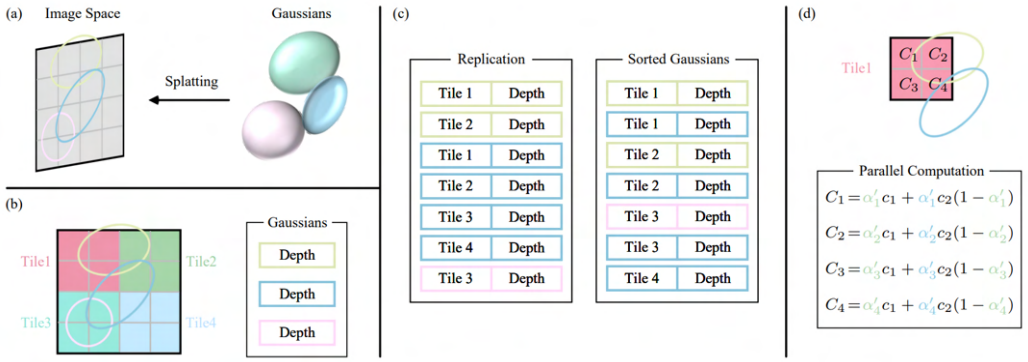


Figure 3-17: An illustration of Forward Pass of 3D-GS pipeline. (a) During the splatting 3D Gaussians are projected into the image space. (b) Differential Tile Rasterizer divides the image into non-overlapping tiles. (c) Sorting algorithm duplicates the Gaussians covering multiple tiles and assigns each copy a tile ID. (d) Rendering the sorted Gaussians enables the retrieval of all pixels within each tile. The computational processes for pixels and tiles are independent and can be executed in parallel. Images taken from [8].

In 2D, you can easily find the projection of a vector μ, represented in homogeneous coordinates (with an additional coordinate of 1), onto an image plane using an intrinsic camera matrix K and an extrinsic camera matrix W=$[R|t]$.

$$\begin{bmatrix} u \\ v \\ z \end{bmatrix} = z \begin{bmatrix} u/z \\ v/z \\ 1 \end{bmatrix} = KW \begin{bmatrix} \mu_x \\ \mu_y \\ \mu_z \\ 1 \end{bmatrix} \tag{3.19}$$

$$\mu^{2D} = \begin{bmatrix} \mu_x^{2D} \\ \mu_y^{2D} \end{bmatrix} = \begin{bmatrix} u/z \\ v/z \end{bmatrix} \tag{3.20}$$

$$\mu^{2D} = K((W\mu)/(W\mu)_z) \tag{3.21}$$

In 2D, Covariance is defined using a Jacobian J of Eq. 3.21:

$$J = \frac{\partial \mu^{2D}(\mu)}{\partial \mu} \tag{3.22}$$

$$\Sigma^{2D} = JW\Sigma W^T J^T \tag{3.23}$$

where J is the Jacobian of the affine approximation of the projective transformation [96].

The *rendering process* with differential rasterization is computationally more efficient than NeRFs due to the following reasons:

- Each 3D point's $f(p)$ for a given camera can be projected into 2D beforehand, eliminating the need for repeated projections when blending Gaussians for nearby pixels.
- There's no need to query MLPs ($Height * Width * Points$) times for a single image; instead, 2D Gaussians are directly blended onto the image.
- There's no ambiguity in selecting which 3D point to evaluate along the ray, as a fixed set of 3D points overlapping the ray of each pixel is determined after optimization. Therefore, no need to have a ray sampling strategy.
- A pre-processing sorting stage is conducted once per frame on the GPU, utilizing custom differentiable CUDA kernels.

As shown in the Figure 3-17, the *Sorting Algorithm* serves to facilitate color rendering with the Eq. 3.18 by arranging 3D points based on their depth determined by their closeness to an image plane and grouping them into tiles. Depth sorting is essential for calculating transmittance, while tile grouping enables the weighted sum for each pixel to be limited to α-blending of relevant 3D points or their 2D projections. Simple 16×16 pixel tiles are utilized for grouping, ensuring that a Gaussian can be distributed across multiple tiles if it spans more than one view frustum. Therefore, rendering of each pixel involves α-blending of preordered points from the corresponding tile.

As shown in Figure 3-15, there are three main components to represent a scene with 3D-GS for good quality renderings: *good initialization, differentiable optimization, and adaptive densification*. 3D-GS starts with the initial set of sparse points from SfM or random initialization. After initialization, a simple Stochastic Gradient Descent is used to fit parameters of 3D Gaussians properly. The scene is optimized using a loss function that is a combination of $\mathcal{L}_1$ and D-SSIM (structural dissimilarity index measure) between the rendered image and ground truth:

$$\mathcal{L} = (1-\lambda)\mathcal{L}_1 + \lambda\mathcal{L}_{D-SSIM}, \tag{3.24}$$

where λ is a weighting factor.

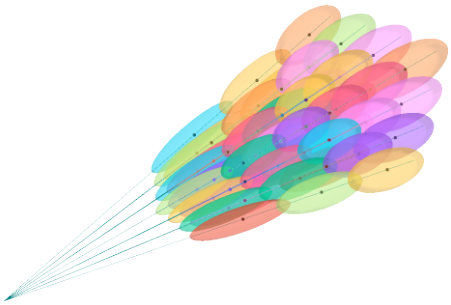


Figure 3-18: An illustration of View Frustums as Gaussians. Images taken from [70].

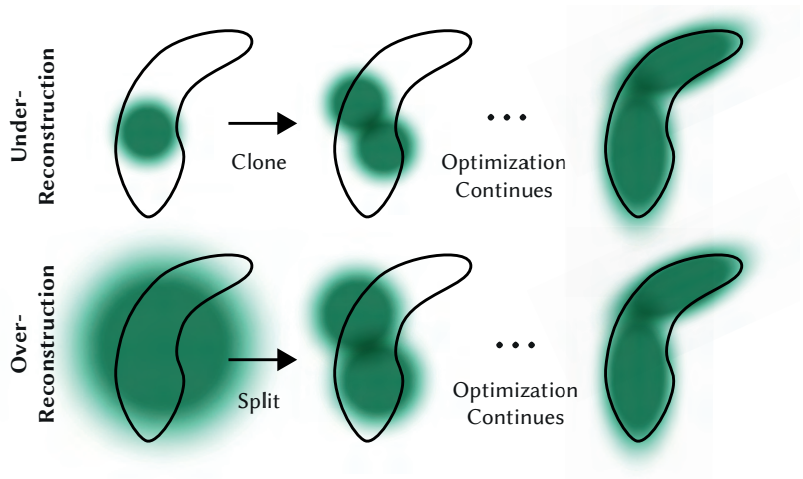


Figure 3-19: Adaptive Gaussian densification. An illustrative example demonstrating the process of fitting a bean-shaped small-scale geometry for rendering using a sparse set of points. Images taken from [32].

As depicted in the Figure 3-19, adaptive densification is a crucial optimization component. It is triggered periodically during training, such as every 100 steps of Stochastic Gradient Descent (SGD, specifically Adam [33]), to tackle under-reconstruction and over-reconstruction issues. But SGD alone can only adjust existing points, making it challenging to optimize areas with sparse or excessive points. Adaptive densification addresses this challenge by splitting points with significant gradients and removing those with very low transparency values. This approach not only saves computational resources but also maintains the accuracy and effectiveness of the Gaussians in accurately representing the scene.

To model the *non-Lambertian effects* like specularities of metallic and glossy surfaces, the Spherical Harmonics (SH) functions are used to learn a view-dependant colors (Section 2.7). SH functions are obtained by selecting positive integers for l and ensuring that $-l \leqslant m \leqslant l$, with one (l, m) pair chosen for each SH:

$$Y_l^m(\theta,\phi) = \frac{(-1)^l}{2^l l!}\sqrt{\frac{(2l+1)(l+m)!}{4\pi(l-m)!}}e^{im\phi}(\sin\theta)^{-m}\frac{d^{l-m}}{d(cos\theta)^{l-m}}(\sin\theta)^{2l} \tag{3.25}$$

We can evaluate this function at any point on the sphere and obtain a corresponding value.

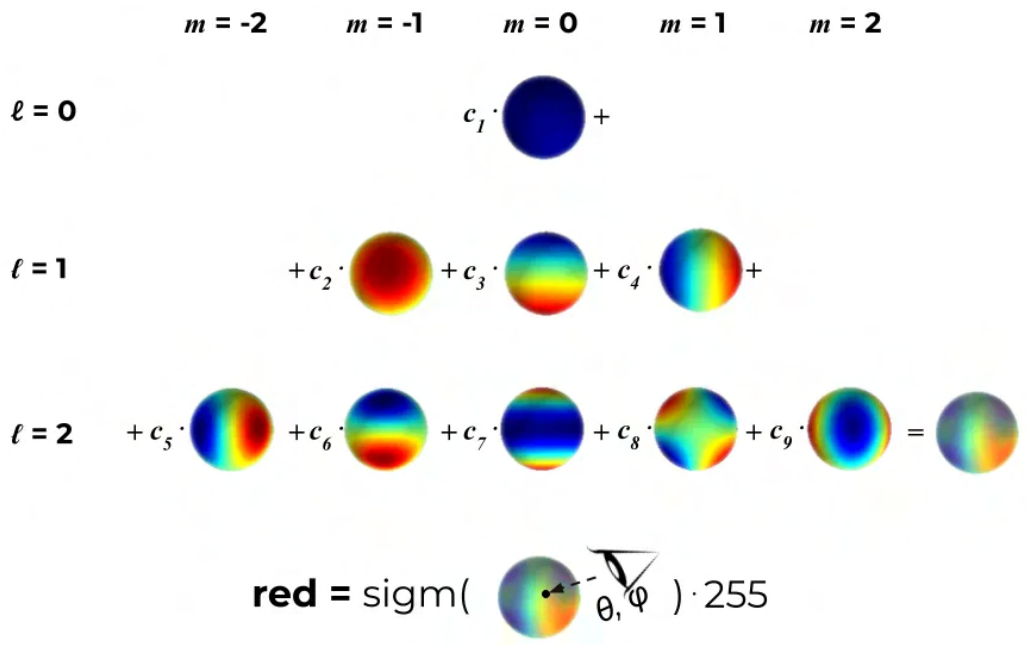


Figure 3-20: Defining view-dependant color using Spherical Harmonics (SH) functions. A toy example illustrates a view-dependant color for red component of a point with $l_{max} = 2$ and 9 learned coefficients. A sigmoid function is used to map the value into the [0, 1] interval. Images taken from [92].

Suppose, each color (red, green, and blue) is a linear combination of the first l_{max} SH functions. As shown in Figure 3-20, we aim to learn the appropriate coefficients for each 3D Gaussian. These coefficients enable the representation of view-dependant color that closely match the true color when viewing the 3D point from a specific direction.

3.6.1 Mathematical Details of Forward and Backward Pass

We discuss forward and backward passes of differential rasterization algorithm in detail, referred from here [29]. As discussed earlier Gaussian has five parameters: Mean Position, Color, Opacity, Scale, and Spherical Harmonic Coefficients described in Table 3.1.

	Variable Name	N Params	Valid Range	Activation
Mean Position	xyz	3	(-inf, inf)	none
Color	rgb	3	[0, 1]	none or sigmoid
Opacity	opacity	1	[0, 1]	sigmoid
Scale	scale	3	(0, inf)	exponential
Spherical Harmonic Coefficients	sh	0, 9, 24, 45		none

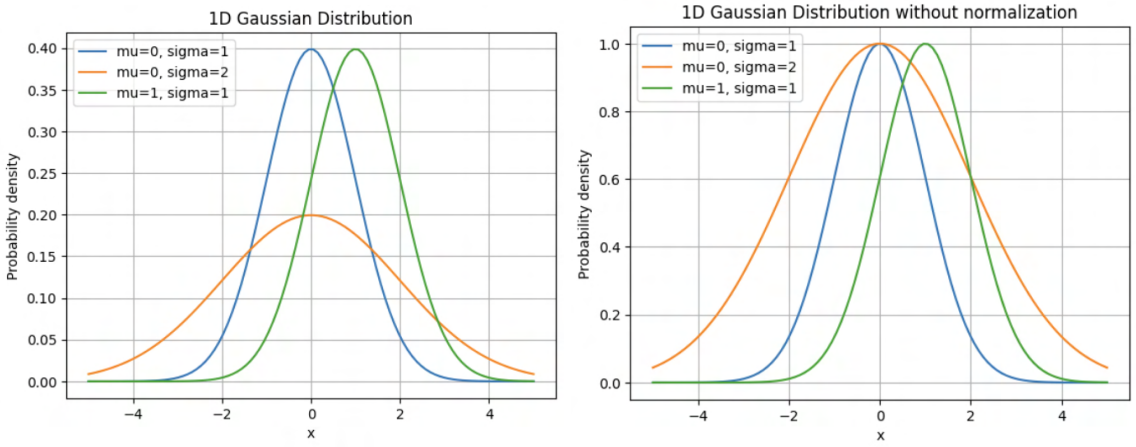
Table 3.1: Description of 3D Gaussian parameters.

Forward Pass

1D Gaussian Distribution. 1D Gaussian probability density function can be defined as,

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} exp\left(\frac{-(x-\mu)}{2\sigma^2}\right) \tag{3.26}$$

where, σ is the standard deviation and μ is the mean.



(a) 1D Gaussian Distribution

(b) 1D Gaussian Distribution without Normalization

Figure 3-21: Illustration of 1D Gaussian probability density function.

As shown in the Figure 3-21, the peak probability changes with due to the normalization term $\frac{1}{\sigma\sqrt{2\pi}}$. In the context of Gaussian splatting, this implies that the opacity of the Gaussian is influenced by its size. When dealing with large gaussians, a considerably high transparency factor is needed to ensure their visibility, while small gaussians might oversaturate the image. Furthermore, this poses a challenge when comparing transparency among gaussians of varying sizes, which is crucial for implementing opacity reset and deletion in the adaptive control algorithm. Removing the normalization factor separates the density from the opacity of the Gaussian distribution, as follows:

$$g(x) = exp\left(\frac{-(x-\mu)}{2\sigma^2}\right) \tag{3.27}$$

Multivariate Gaussian Distribution. The generalized multivariate Gaussian distribution without normalization can be defined as:

$$g(\boldsymbol{x}) = exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right) \tag{3.28}$$

where, μ represent the mean vector, while Σ denotes the covariance matrix. The covariance matrix is a symmetric and positive semi-definite $N\times N$ matrix, with N denoting the number of dimensions in the distribution.

Optimizing 3D Gaussians. The 3D Gaussian distributions are described using a 3×3 covariance matrix σ. To keep the matrix symmetric, it's sufficient to optimize only the 6 parameters in its upper triangular part. However, ensuring the matrix is positive semi-definite is challenging. Rather than directly optimizing the 3D covariance matrix Σ_3, the authors create the matrix representation of an ellipsoid using 3 scale factors and a 3D rotation. This method essentially performs an inverse Principal Component Analysis. The covariance matrix can be deconstructed into its eigenvalues and eigenvectors:

$$\Sigma_3\boldsymbol{v} = \lambda\boldsymbol{v} \tag{3.29}$$

where, $\boldsymbol{v}$ is eigenvector and λ is respective eigenvalue, and can be written as,

$$\Sigma_3\begin{bmatrix}v_0 & v_1 & v_2\end{bmatrix} = \begin{bmatrix}v_0 & v_1 & v_2\end{bmatrix}\begin{bmatrix}\lambda_1 & 0 & 0\\ 0 & \lambda_2 & 0\\ 0 & 0 & \lambda_3\end{bmatrix} \tag{3.30}$$

Considering V as a matrix in which columns represents eigenvectors,

$$\Sigma_3 V = VL \tag{3.31}$$

$$\Sigma_3 = VLV^{-1} \tag{3.32}$$

In Principal Component Analysis (PCA), the eigenvectors indicate the directions with the highest variance, while the eigenvalues represent the magnitude of this variance. The matrix of eigenvectors corresponds to the rotation matrix of the axes of largest variance to the starting reference frame. Since this rotation matrices are orthogonal, their inverse is equivalent to their transpose.

$$\Sigma_3 = RLR^T \tag{3.33}$$

$$\Sigma_3 = R\begin{bmatrix}\lambda_1 & 0 & 0\\ 0 & \lambda_2 & 0\\ 0 & 0 & \lambda_3\end{bmatrix}R^T \tag{3.34}$$

Now, we can define the scale matrix with $s_n = \sqrt{\lambda_n}$,

$$S = \begin{bmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & s_3 \end{bmatrix}$$

and, substituting it back into the above equation,

$$\Sigma_3 = RSS^T R^T = RS(RS)^T \tag{3.35}$$

The 3×3 rotation matrix can be represented using quaternion with 7 parameters to be optimized, namely q_w, q_x, q_y, q_z and s_1, s_2, s_3. The covariance matrix derived from this representation is assured to be positive semi-definite due to $\lambda_n = {s_n}^2$, ensuring that all eigenvalues are non-negative.

Projecting 3D Gaussians. The 3D covariance matrix can be projected into a 2D covariance matrix by:

$$\Sigma_2 = JW\Sigma_3 W^T J^T \tag{3.36}$$

where, W is 3×3 rotation matrix that represents viewing transformation, while J is 3×3 Jacobian matrix of the affine approximation of the projective transformation. When dealing with large focal lengths and small Gaussians, this approximation is expected to perform effectively,

$$J = \begin{bmatrix} f_x/z & 0 & -f_x x/z^2 \\ 0 & f_y/z & -f_y y/z^2 \end{bmatrix}$$

EWA Splatting [96] uses a projection plane at $z = 1$ which is equivalent to $f_x = f_y = 1$.

Creating Images with 2D Gaussians. The mean of a 2D Gaussian can be determined by projecting the 3D mean by a standard pinhole projection,

$$\begin{bmatrix} \bar{u} & \bar{v} \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \end{bmatrix} \begin{bmatrix} \bar{x}/\bar{z} \\ \bar{y}/\bar{z} \\ 1 \end{bmatrix} \tag{3.37}$$

We can calculate the unnormalized probability density for each pixel using,

$$g(\boldsymbol{x}) = exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right) \tag{3.38}$$

where,

$$(x-\mu) = \begin{bmatrix} \bar{u}-u \\ \bar{v}-v \end{bmatrix}$$

$$\Sigma^{-1} = \frac{1}{ad-bc}\begin{bmatrix} d & -b & -c & a \end{bmatrix}$$

Tile Based Rasterization. Computing the probability of every Gaussian at every pixel in the image would be impractical for real-time rendering due to its slow speed. Fortunately, the majority of Gaussians only impact a small section of the rendered image, which is crucial for accurately reconstructing intricate scene details. Consequently, the authors opt to divide the image into 16 by 16 pixel tiles shown in Figure 3-22 and solely focus on rendering the Gaussians that substantially contribute to each tile.

The rendered images are transformed into *uint*8, thereby limiting their resolution to 1/255, approximately resembling the probability of a Gaussian distribution at 3σ. The process of associating Gaussian with tiles involves finding the intersection of the Gaussian distribution at 3σ and the tiles. We can determine the oriented bounding box of an ellipse by using the 2D covariance matrix,

$$\Sigma_2 = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

We can compute radii at 3σ,

$$r_1 = 3\sqrt{\lambda_1}$$
$$r_2 = 3\sqrt{\lambda_2},$$

for which we can compute two eigenvalues for a 2×2 symmetric matrix

$$\lambda = \frac{a + d + \sqrt{(a-d)^2 + 4bc}}{2}$$

The orientation can be computed with:
if $b \neq 0$:

$$\theta = arctan\left(\frac{\lambda_1}{b}\right)$$

if $b = 0$ and $a >= d$:

$$\theta = 0$$

if $b = 0$ and $a < d$:

$$\theta = \pi/2$$

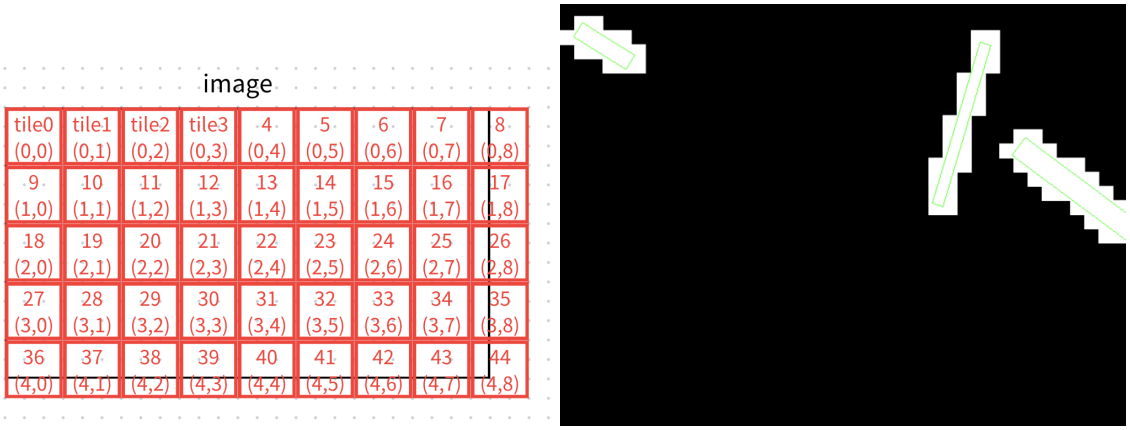
The oriented bounding box's four corner points can be generated from r_1, r_2, θ by initializing with the four corner points of the axis-aligned bounding box and then applying a rotation using a 2D rotation matrix:

$$R = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$$

We can find where the oriented bounding box and the tile overlap by applying the *Separating Axis Theorem* (SAT). In this scenario, we can simplify the SAT because each bounding box has two sets of parallel axes, and we only need to check one set.

Moreover, the axes of the tile are aligned with the axes, which removes two projection steps from the process.

The Figure 3-22 displays three Gaussian splatted onto the image. The green rectangles represent the oriented bounding boxes at 3σ, with intersecting tiles highlighted in white.



(a) Illustration of how the whole image divided into tiles which are 16×16 pixel blocks.

(b) Illustration of tile based rasterization.

Figure 3-22: Illustration of how to compute which tiles each Gaussian covers.

Alpha Compositing. Each pixel's RGB value is calculated by α blending the Gaussian sequentially from the front to the back. The RGB values for each pixel can be determined using the following method:

$$\begin{aligned} C(u,v) &= \sum_{i=1}^{N} \alpha_i c_i w_i \\ w_i &= (1 - \sum_{j=0}^{i-1} \alpha_j w_j) \\ w_0 &= 1 \\ \alpha_i &= o_i g_i(u,v) \end{aligned} \tag{3.39}$$

where, c_i is color and o_i is opacity of i^{th} Gaussian, and $g_i(u,v)$ is the probability of the the i^{th} Gaussian at the pixel coordinates (u,v).

Backward Pass

All derivatives are denoted by ∇.

Camera Projection. Calculating the derivatives for the camera projection is quite

simple. When it comes to the forward projection:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \end{bmatrix} \begin{bmatrix} \frac{x}{z} \\ \frac{y}{z} \\ z \\ 1 \end{bmatrix} \tag{3.40}$$

The Jacobian remains the same, the one utilized for projecting 3D Gaussians onto a 2D plane:

$$J = \begin{bmatrix} f_x/z & 0 & -f_x x/z^2 \\ 0 & f_y/z & -f_y y/z^2 \end{bmatrix} \tag{3.41}$$

vector-Jacobian product is,

$$[\nabla x \quad \nabla y \quad \nabla z] = [\nabla u \quad \nabla v] \begin{bmatrix} f_x/z & 0 & -f_x x/z^2 \\ 0 & f_y/z & -f_y y/z^2 \end{bmatrix} \tag{3.42}$$

Normalizing Quaternion. For a quaternion $q = [w \quad x \quad y \quad z]$, normalized quaternion of q is,

$$\hat{q} = \begin{bmatrix} \frac{w}{\|q\|} & \frac{x}{\|q\|} & \frac{y}{\|q\|} & \frac{z}{\|q\|} \end{bmatrix} \tag{3.43}$$

vector-Jacobian products are:

$$\begin{aligned}
\nabla w &= \nabla \hat{q}^T \begin{bmatrix} -\frac{w^2}{(w^2+x^2+y^2+z^2)^{\frac{3}{2}}} + \frac{1}{\sqrt{w^2+x^2+y^2+z^2}} \\ -\frac{wx}{(w^2+x^2+y^2+z^2)^{\frac{3}{2}}} \\ -\frac{wy}{(w^2+x^2+y^2+z^2)^{\frac{3}{2}}} \\ -\frac{wz}{(w^2+x^2+y^2+z^2)^{\frac{3}{2}}} \end{bmatrix} \\
\nabla x &= \nabla \hat{q}^T \begin{bmatrix} -\frac{wx}{(w^2+x^2+y^2+z^2)^{\frac{3}{2}}} \\ -\frac{x^2}{(w^2+x^2+y^2+z^2)^{\frac{3}{2}}} + \frac{1}{\sqrt{w^2+x^2+y^2+z^2}} \\ -\frac{xy}{(w^2+x^2+y^2+z^2)^{\frac{3}{2}}} \\ -\frac{xz}{(w^2+x^2+y^2+z^2)^{\frac{3}{2}}} \end{bmatrix} \\
\nabla y &= \nabla \hat{q}^T \begin{bmatrix} -\frac{wy}{(w^2+x^2+y^2+z^2)^{\frac{3}{2}}} \\ -\frac{xy}{(w^2+x^2+y^2+z^2)^{\frac{3}{2}}} \\ -\frac{y^2}{(w^2+x^2+y^2+z^2)^{\frac{3}{2}}} + \frac{1}{\sqrt{w^2+x^2+y^2+z^2}} \\ -\frac{yz}{(w^2+x^2+y^2+z^2)^{\frac{3}{2}}} \end{bmatrix} \\
\nabla z &= \nabla \hat{q}^T \begin{bmatrix} -\frac{wz}{(w^2+x^2+y^2+z^2)^{\frac{3}{2}}} \\ -\frac{xz}{(w^2+x^2+y^2+z^2)^{\frac{3}{2}}} \\ -\frac{yz}{(w^2+x^2+y^2+z^2)^{\frac{3}{2}}} \\ -\frac{z^2}{(w^2+x^2+y^2+z^2)^{\frac{3}{2}}} + \frac{1}{\sqrt{w^2+x^2+y^2+z^2}} \end{bmatrix}
\end{aligned} \tag{3.44}$$

Normalized Quaternion to Rotation Matrix. Consider w, x, y, and z are the components of $\hat{q}$. For a normalized quaternion $\hat{q}$,

$$R = \begin{bmatrix} -2y^2 - 2z^2 + 1 & -2wz + 2xy & 2wy + 2xz \\ 2wz + 2xy & -2x^2 - 2z^2 + 1 & -2wx + 2yz \\ -2wy + 2xz & 2wx + 2yz & -2x^2 - 2y^2 + 1 \end{bmatrix} \tag{3.45}$$

vector-Jacobian products are:

$$\begin{aligned} \nabla w &= \nabla R^T \begin{bmatrix} 0 & -2z & 2y \\ 2z & 0 & -2x \\ -2y & 2x & 0 \end{bmatrix} \\ \nabla x &= \nabla R^T \begin{bmatrix} 0 & 2y & 2z \\ 2y & -4x & -2w \\ 2z & 2w & -4x \end{bmatrix} \\ \nabla y &= \nabla R^T \begin{bmatrix} -4y & 2x & 2w \\ 2x & 0 & 2z \\ -2w & 2z & -4y \end{bmatrix} \\ \nabla z &= \nabla R^T \begin{bmatrix} -4z & -2w & 2x \\ 2w & -4z & 2y \\ 2x & 2y & 0 \end{bmatrix} \end{aligned} \tag{3.46}$$

3D Covariance Matrix. 3D covariance matrix $\Sigma_3 = RS(RS)^T$, can be written as,

$$\begin{aligned} M &= RS \\ \Sigma_3 &= MM^T \end{aligned}$$

we can compute gradients using formula: $C = AB, \nabla A = \nabla C B^T, \nabla B = A^T \nabla C$,

$$\begin{aligned} \nabla M &= \nabla \Sigma_3 (M^T)^T = \nabla \Sigma_3 M \\ \nabla M^T &= M^T \nabla \Sigma_3 \end{aligned} \tag{3.47}$$

Gradients of the components of M,

$$\begin{aligned} \nabla R &= \nabla M S^T \\ \nabla S &= R^T \nabla M \end{aligned} \tag{3.48}$$

Gradients of the components of M^T,

$$\begin{aligned} \nabla R^T &= S \nabla M^T \\ \nabla S^T &= \nabla M^T R \end{aligned} \tag{3.49}$$

Combining the gradients of the transposed components:

$$\begin{aligned}\nabla R &= \nabla M S^T + (S\nabla M^T)^T = \nabla M S^T + (\nabla M^T)^T S^T \\ \nabla S &= R^T \nabla M + (\nabla M^T R)^T = R^T \nabla M + R^T (\nabla M^T)^T\end{aligned} \tag{3.50}$$

Substituting in for M,

$$\begin{aligned}\nabla R &= \nabla \Sigma_{3D} R S S^T + (\nabla \Sigma_{3D})^T R S S^T \\ \nabla S &= R^T \nabla \Sigma_{3D} R S + R^T (\nabla \Sigma_{3D})^T R S\end{aligned} \tag{3.51}$$

Simplifying further as $S, \Sigma_3, are \nabla \Sigma_3$ are symmetric,

$$\begin{aligned}\nabla R &= 2\nabla \Sigma_{3D} R S S \\ \nabla S &= 2R^T \nabla \Sigma_{3D} R S\end{aligned} \tag{3.52}$$

Evaluating the Gaussian. The unnormalized probability of the Gaussian function,

$$g(\boldsymbol{x}) = exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right)$$

using Mahalanobis distance, $d_M = \sqrt{(\boldsymbol{x}-\boldsymbol{\mu})^T \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}$,

$$g(\boldsymbol{x}) = exp\left(-\frac{1}{2} d_M^2\right) \tag{3.53}$$

Using the conic representation of the 2D covariance matrix evaluating at the pixel (u, v),

$$d_M^2 = \begin{bmatrix} u-\mu_u \\ v-\mu_v \end{bmatrix}^T \begin{bmatrix} a & b \\ b & c \end{bmatrix}^{-1} \begin{bmatrix} u-\mu_u \\ v-\mu_v \end{bmatrix} \tag{3.54}$$

$$simlifying, d_M^2 = \frac{a\Delta v^2 - 2b\Delta u \Delta v + c\Delta u^2}{(ac-2b)} \tag{3.55}$$

vector-Jacobian products are,

$$\begin{aligned}\nabla \Sigma_{2D} &= \nabla d_M^2 \begin{bmatrix} \frac{\partial d_M^2}{\partial a} & \frac{\partial d_M^2}{\partial b} \\ \frac{\partial d_M^2}{\partial b} & \frac{\partial d_M^2}{\partial c} \end{bmatrix} \\ \nabla a &= \nabla d_M^2 \left(-\frac{c\left(av^2 - 2buv + cu^2\right)}{(ac-2b)^2} + \frac{v^2}{ac-2b}\right) \\ \nabla b &= 2\nabla d_M^2 \left(-\frac{uv}{ac-2b} + \frac{\left(av^2 - buv + cu^2\right)}{(ac-2b)^2}\right) \\ \nabla c &= -\nabla d_M^2 \left(\frac{a\left(av^2 - 2buv + cu^2\right)}{(ac-2b)^2} + \frac{u^2}{ac-2b}\right)\end{aligned} \tag{3.56}$$

The derivative of d_M^2 is calculated using the probability density function at (u, v):

$$\nabla d_M^2 = -\frac{1}{2}\nabla g(u,v)g(u,v) = -\frac{1}{2}\nabla g(u,v)exp\left(-\frac{1}{2}d_M^2\right) \tag{3.57}$$

Alpha Compositing. Color gradient ∇C_n for compositing N gaussians can be computed iteratively starting from $n = 0$,

$$\begin{aligned} \nabla C_n &= \nabla C_{image}\alpha_n w_n \\ w_n &= (1 - \sum_{i=0}^{n-1} \alpha_i w_i) \\ w_0 &= 1 \end{aligned} \tag{3.58}$$

By computing the gradients from back to front the weight and accumulated color can be efficiently computed by saving the final weight in the forward pass. Starting with,

$$\begin{aligned} C_{accum} &= 0 \\ w_i &= w_{final} \end{aligned} \tag{3.59}$$

At each iteration,

$$\begin{aligned} \nabla C_i &= \nabla C_{image}\alpha_i w_i \\ \nabla \alpha_i &= \nabla C_{image}\left(C_i w_i - \frac{C_{accum}}{1-\alpha_i}\right) \end{aligned} \tag{3.60}$$

Updating accumulated color and weight for the next step,

$$\begin{aligned} C_{accum} &= C_{accum} + C_i\alpha_i w_i \\ w_{i-1} &= \frac{w_i}{1-\alpha_i} \end{aligned} \tag{3.61}$$

3.6.2 gsplat implementation of 3D-GS

gsplat [87] is an open-source library for CUDA-accelerated differentiable rasterization of 3D Gaussians [32] with Python bindings, making it easier to interact with the rasterizer for developing additional features. It provides numerous additional features and techniques, such as batch rasterization, N-D feature rendering, depth rendering, sparse gradients, absgrad [88], anti-aliasing [91], and more.

Depth Estimation

In gsplat [87], The estimated per-pixel z-depth values, denoted as $\hat{D}$, are computed using a discrete volume rendering approximation similar to the computation of color

values:

$$\hat{D} = \sum_{i \in N} d_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j) \tag{3.62}$$

where d_i represents the z-depth of the i^{th} Gaussian in view space. Since 3D-GS [32] does not sort the Gaussians individually for each pixel along a viewing ray, but instead performs a single global sort for efficiency, this approach provides only an approximate per-pixel depth estimation.

Anti-aliasing

Mip-Splatting Alias-free 3D-GS [91] identified aliasing artifacts can be resulting from a lack of 3D frequency constraints and the use of a 2D dilation filter. To solve this, the authors introduced a 3D smoothing filter to adjust the size of 3D Gaussian primitives based on the input views' maximum sampling frequency, removing high-frequency artifacts when zooming in. Additionally, replacing the 2D dilation with a 2D Mip-filter, which mimics a 2D box filter, helps reduce aliasing and dilation issues.

Chapter 4

Proposed Synthetic Scene Dataset

In this section, we dive into the concept of structural colors and introduce our synthetic scene referred to as StructColorToaster scene.

4.1 Structural Colors

Traditional color refers to the colors that we perceive due to the selective absorption, reflection, and transmission of light by different materials. It is based on the absorption spectrum of pigments or dyes present in an object. The color we see is the result of certain wavelengths of light being absorbed, and the remaining wavelengths are reflected, giving the object its perceived color.

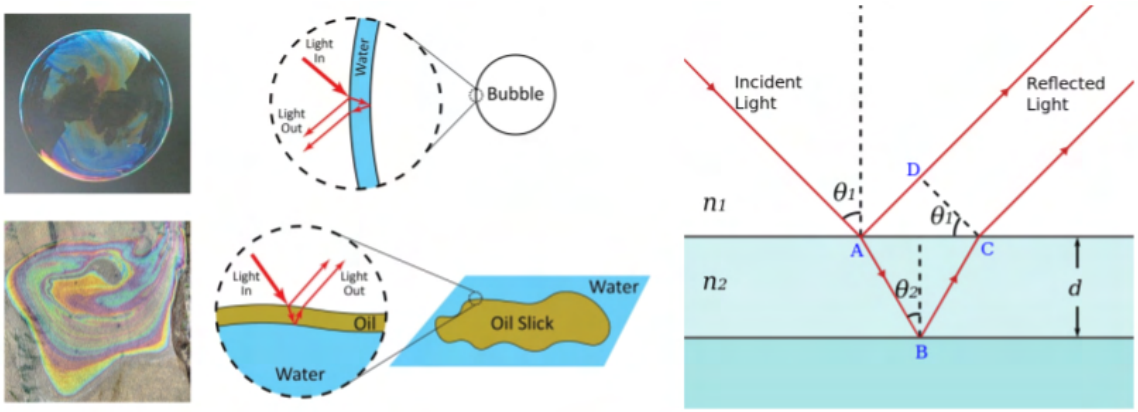


Figure 4-1: An illustration of structural color formation on soap bubble and oil slick (Left). Thin-film interference, when light falls on a thin film, the waves reflected from the upper and lower surfaces travel different distances depending on the angle, so they interfere (Right). Images taken from [10], and [79].

Structural color, on the other hand, is based on the physical structure of the object rather than the chemical composition. It arises from interference, diffraction, or scattering of light by the microscopic structure present on the material. If the material

contains specific shapes or patterns on the surface, it will reflect specific colors of light. In structural colors, not only the color brightness and saturation, but also the color hue varies with viewing angle. In nature, we can find structural colors on the surface of a soap bubble, oil slick, and the wings of butterflies, shown in Figures 4-1 and 4-2.

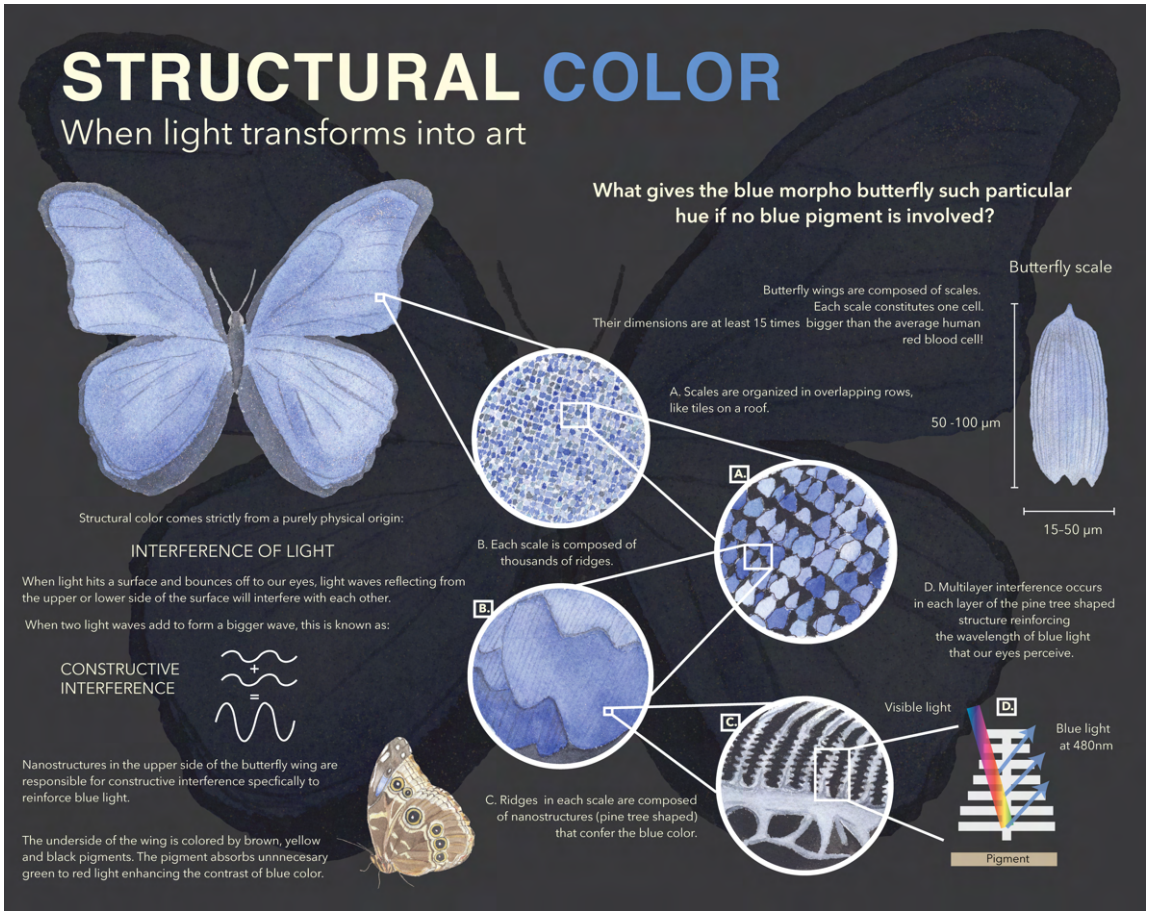


Figure 4-2: An illustration of structural color formation on wings of butterfly. The scales on the butterfly's wings are arranged like tiles on a roof. Each scale comprises numerous ridges, which in turn consist of nanostructures resembling the shape of pine trees. When light interacts with these nanostructures, it undergoes wave interference, reinforcing the wavelength of blue light. Images taken from [57].

4.2 Synthetic Dataset - Structural Color Blender

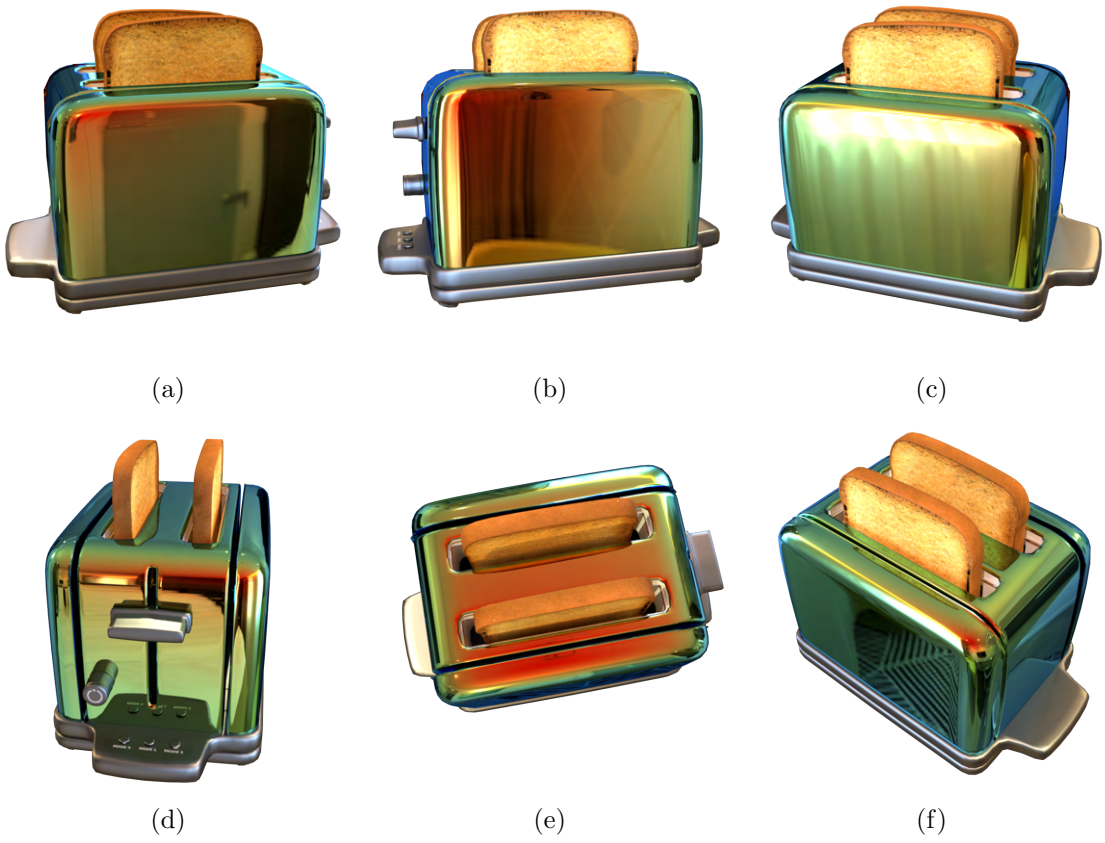
While the dataset referred to as "Blender", which is utilized by NeRF [48] and "Shiny Blender", utilized by Ref-NeRF [76], contains a wide range of objects with intricate geometry, but it notably lacks diversity in terms of materials. In these datasets, most scenes primarily consist of Lambertian objects in the Blender dataset and glossy objects in the Shiny Blender dataset. To explore more demanding material charac-

teristics, we have developed an additional dataset named "Structural Color Blender," which consists of a toaster object scene from Ref-NeRF but with material mimicking structural colors, referred to as the StructColorToaster scene. This scene is rendered in Blender under conditions identical to those of the NeRF and Ref-NeRF datasets.

We are assigning names to the scenes within our "Structural Color Blender" dataset, so that in the future, other researchers may contribute their scenes featuring objects with structural colors to the dataset for benchmarking purposes.

4.3 StructColorToaster Scene

We generate StructColorToaster scene from our dataset by rendering pathtraced images of object that exhibit complicated geometry and realistic non-Lambertian materials specifically material mimicking structural-colors or related phenomenon like Pearlescence and Iridescence. These images are rendered from viewpoints sampled on the upper hemisphere. We render 124 views of the scene as input and 249 for testing, all at 800 × 800 pixels.



(a) (b) (c)

(d) (e) (f)

Figure 4-3: Few samples from StructColorToaster scene rendered using the Blender.

As depicted in Figure 4-3, the color brightness, saturation, and hue of the sample

images undergo significant changes depending on the viewing angle. Additionally, the images exhibit intricate reflections of complex geometries due to the utilization of the environment map (Figure 4-4) during rendering. The *transform_*.json* files contains camera poses. The dataset also contains Depth and Normal maps of rendered views.

Figure 4-4: Environment map used for rendering StructColorToaster scene in Blender. Images taken from [76].

The inclusion of StructColorToaster scene [1], which simulates complex shiny, reflective, and pearlescent effects resembling structural colors on objects surfaces, is certainly steering research towards innovative directions in novel view synthesis involving complex surfaces with structural colors.

[1] If you are interested in our dataset, please refer to: `https://bit.ly/Structural-Color-Blender-Dataset`

Chapter 5

Experiments with StructColorToaster Scene

In this chapter, we describe the experimental settings in detail to perform novel view synthesis experiments using the methods discussed in Chapter 3. We also discuss the findings obtained after experiments on the StructColorToaster scene. Lastly, we briefly discuss how considering these findings is important for future research direction.

5.1 Experiments

We use the open-source NeRFStudio [70] implementation of NeRF [48], instant-NGP [51], Mip-NeRF [3], NeRFacto [70]; multinerf [49] implementation of Ref-NeRF [76]; and orginal 3D-GS [32] implementation for our experiments. We use NVIDIA Quadro RTX 8000 (48GB) and NVIDIA Tesla A16 (64GB) GPUs for training purposes.

We will compare different evaluation metrics such as PSNR, LPIPS, SSIM, training speed, rendering speed, and the visual appearance and geometry of rendered test-set views for the StructColorToaster scene for the above-mentioned methods. Based on this comparison, we chose method to conduct experiments on our real dataset captured using handheld smartphone's camera.

5.1.1 NeRF

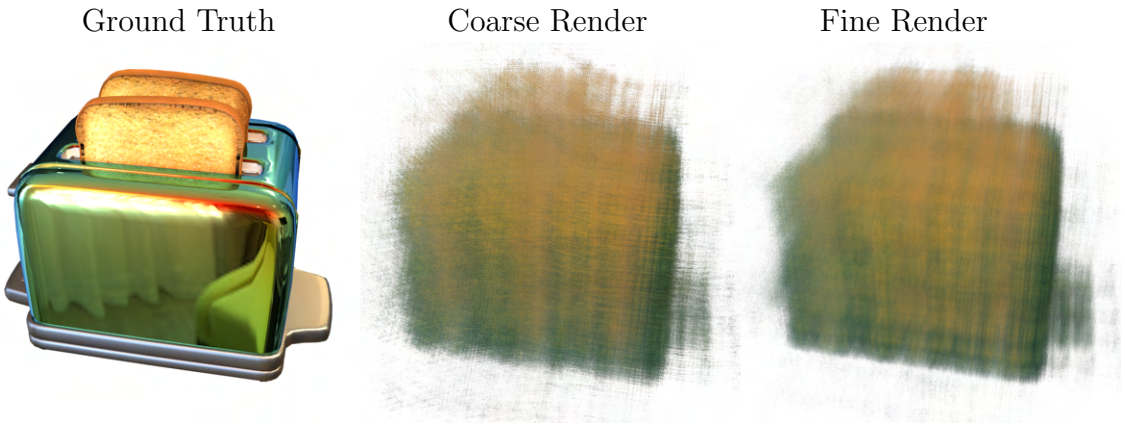
We train NeRF [48] for 50K iterations over approximately one day.

As shown in Figures 5-1 and 5-2, NeRF achieves a PSNR↑ score of 23.359, SSIM↑ score of 0.8427, and LPIPS↓ score of 0.1895. Nevertheless, it struggles to accurately depict the crisp glossiness of the surface and lacks the ability to accurately represent reflections of objects visible on the toaster's surfaces. Figure 5-3 shows the accumulations and depth rendered from the optimized NeRF for StructColorToaster. These visualization shows that NeRF learns the overall surface geometry of the toaster scene decently but fails to reconstruct the appearance. Overall, NeRF is able to represent the geometry but fails to represent complex appearance of surfaces with structural colors.

(a) PSNR↑ (b) SSIM↑

(c) LPIPS↓ (d) $L2$ Loss

Figure 5-1: Evaluation metrics for NeRF on test images from the StructColorToaster scene.

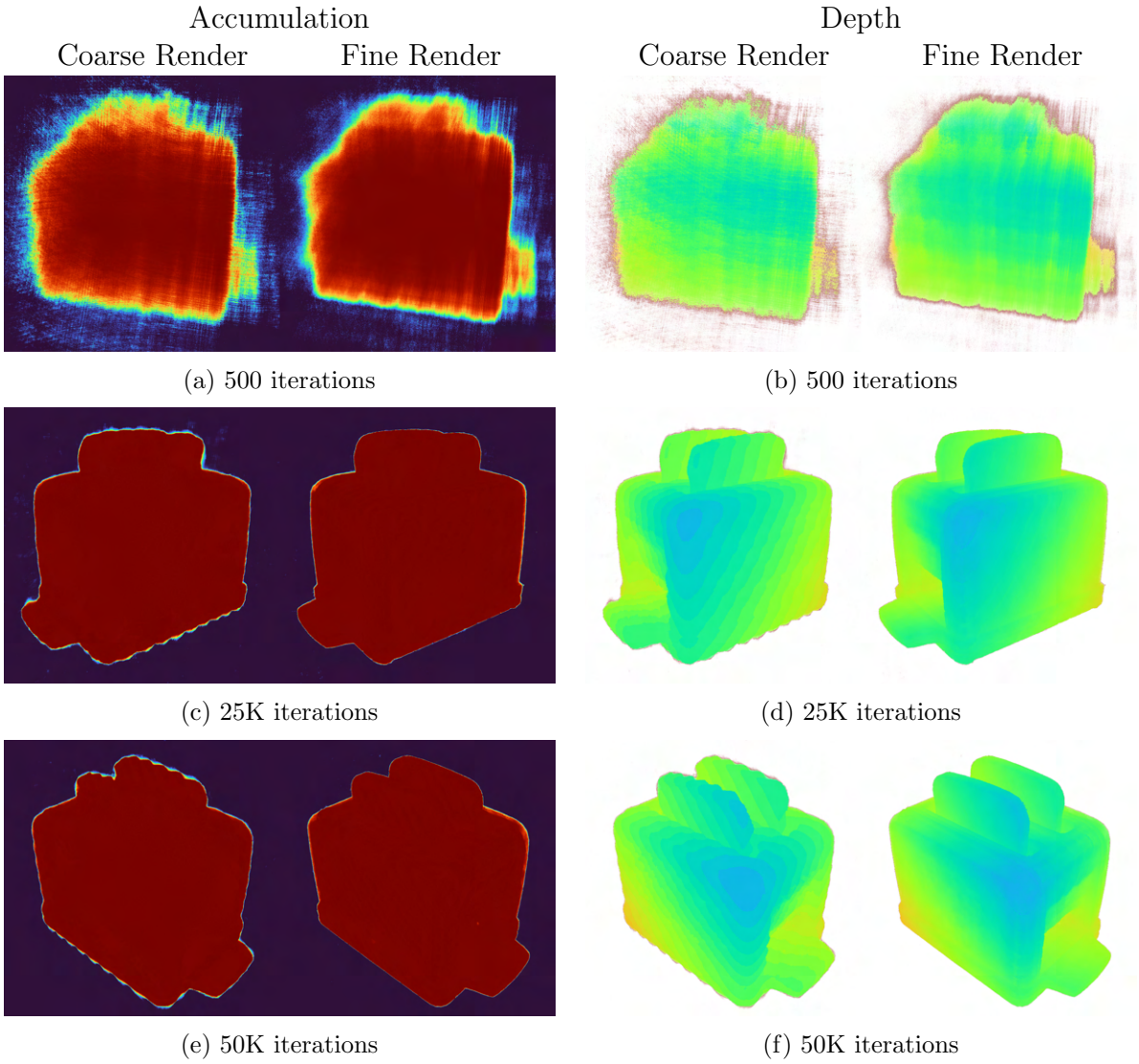


(a) 500 iterations

(b) 25K iterations

(c) 50K iterations

Figure 5-2: Renderings shows NeRF fails to learn appearance and specular reflections with increasing iterations.



(a) 500 iterations

(b) 500 iterations

(c) 25K iterations

(d) 25K iterations

(e) 50K iterations

(f) 50K iterations

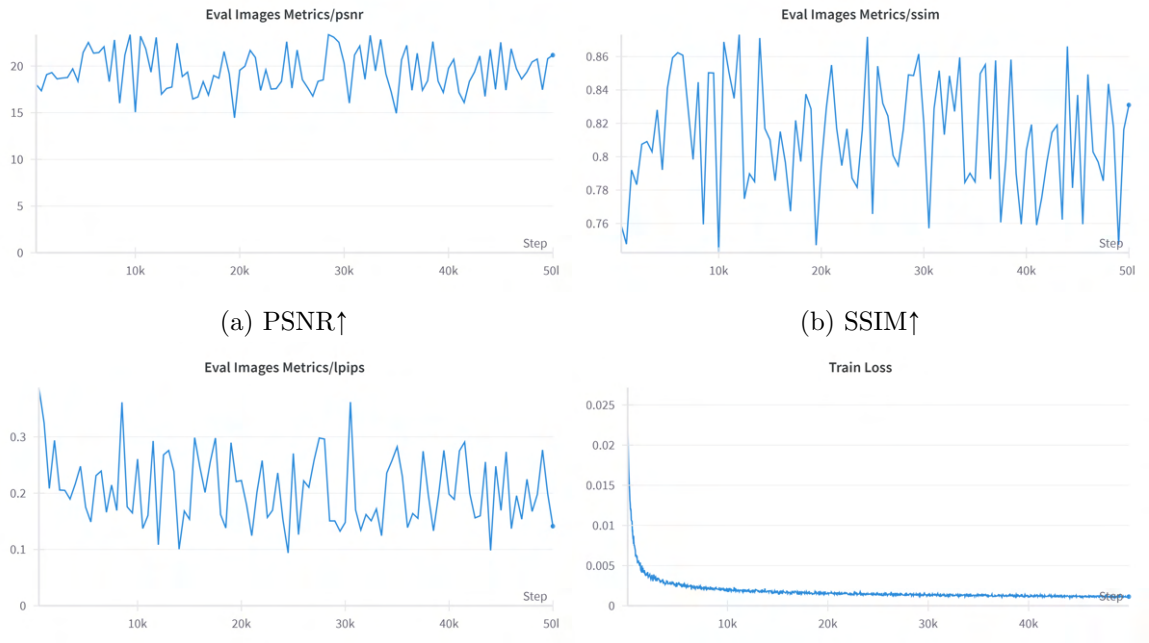
Figure 5-3: Accumulations and Depth visualizations from optimized NeRF.

5.1.2 InstantNGP

We train InstantNGP [51] for 50K iterations, which takes approximately 23 minutes.

Renderings of test views from optimized InstantNGP shown in Figures 5-5 and 5-6 reveal that InstantNGP fails to learn appearance of the scene objects but learns the geometry of toaster decently. Additionally, it fails to render the specular reflections accurately, instead producing blurred reflections.

InstantNGP achieves the PSNR↑ value of 19.659, SSIM↑ value of 0.8109, and LPIPS↓ value of 0.2068 for StructColorToaster scene, which are not up to the mark. In general, InstantNGP successfully captures the geometry of objects but struggles to depict the intricate appearance of surfaces with structural colors.



(a) PSNR↑ (b) SSIM↑

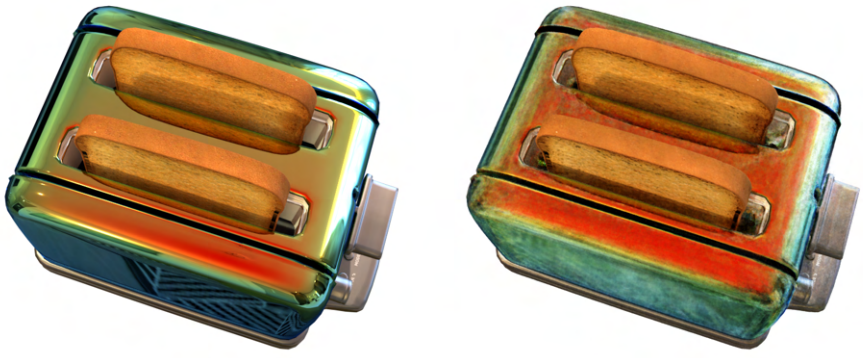
(c) LPIPS↓ (d) $L2$ Loss

Figure 5-4: Evaluation metrics for InstantNGP on test images from the StructColorToaster scene.

Ground Truth InstantNGP

(a) 500 iterations

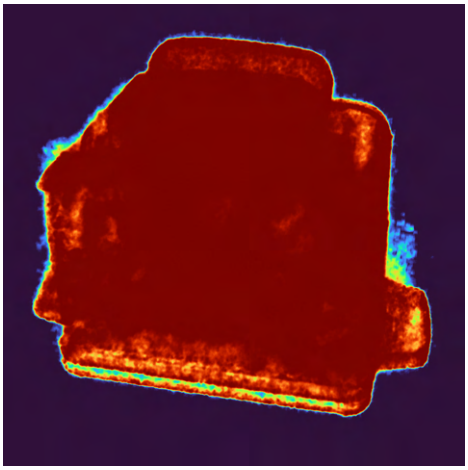
(b) 25K iterations



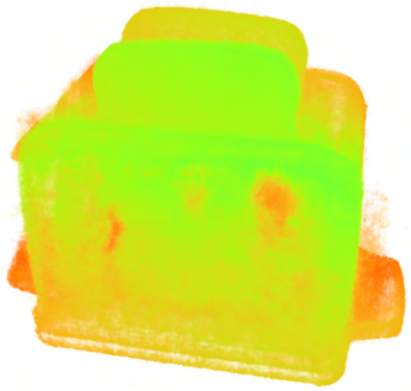
(c) 50K iterations

Figure 5-5: Renderings shows InstantNGP fails to learn appearance with increasing iterations.

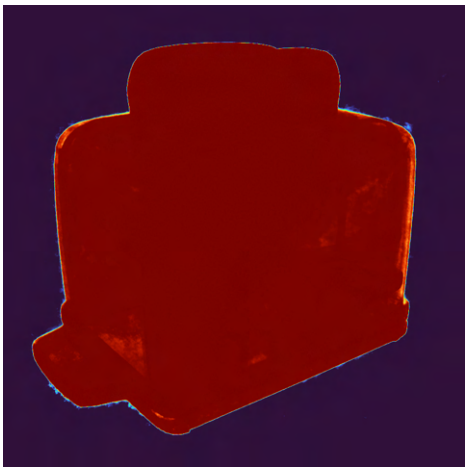
Accumulation | Depth



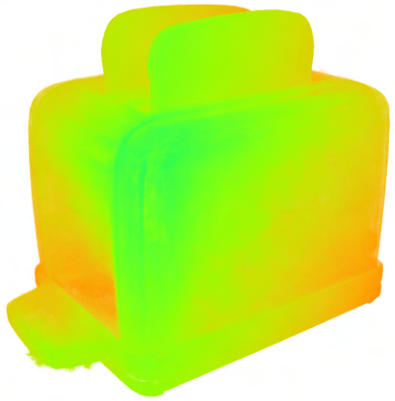
(a) 500 iterations



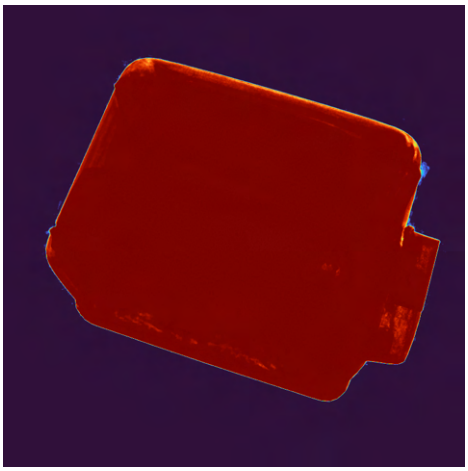
(b) 500 iterations



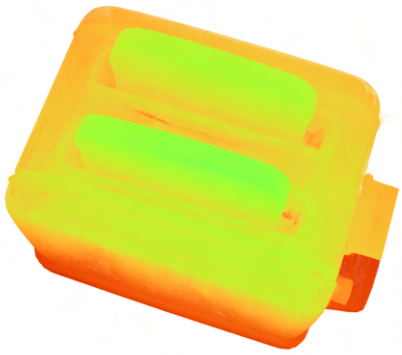
(c) 25K iterations



(d) 25K iterations



(e) 50K iterations



(f) 50K iterations

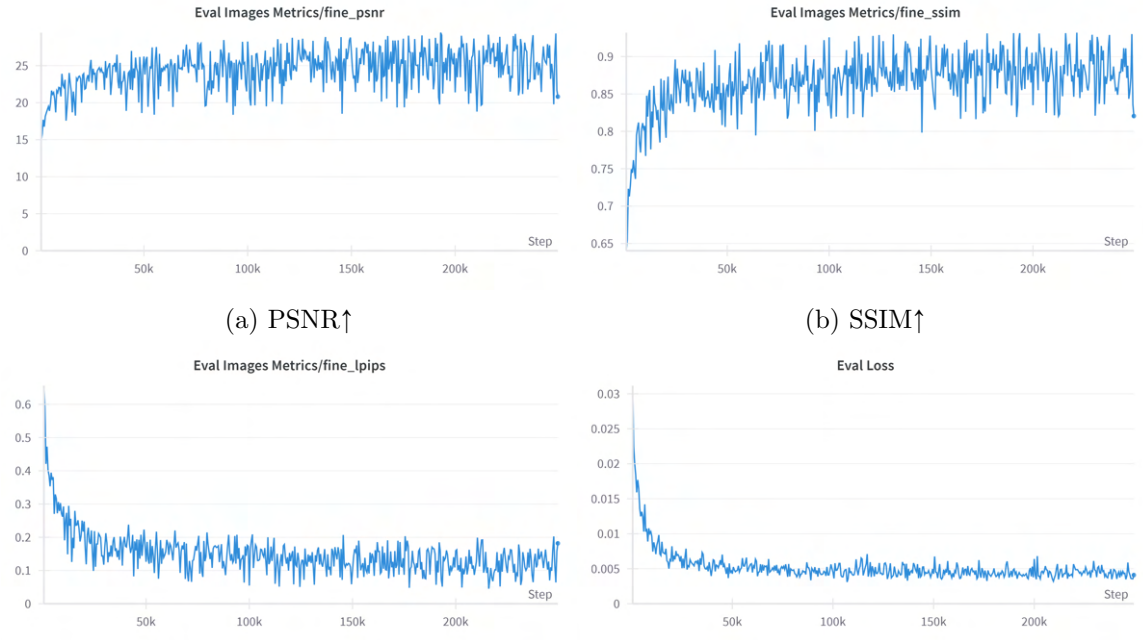
Figure 5-6: Accumulations and Depth visualizations from optimized InstantNGP.

5.1.3 Mip-NeRF

We train for 250K iterations, which takes approximately more than one day.

As shown in Figures 5-7, Mip-NeRF achieves a PSNR↑ score of 22.818, SSIM↑ score of 0.877, and LPIPS↓ score of 0.1263.

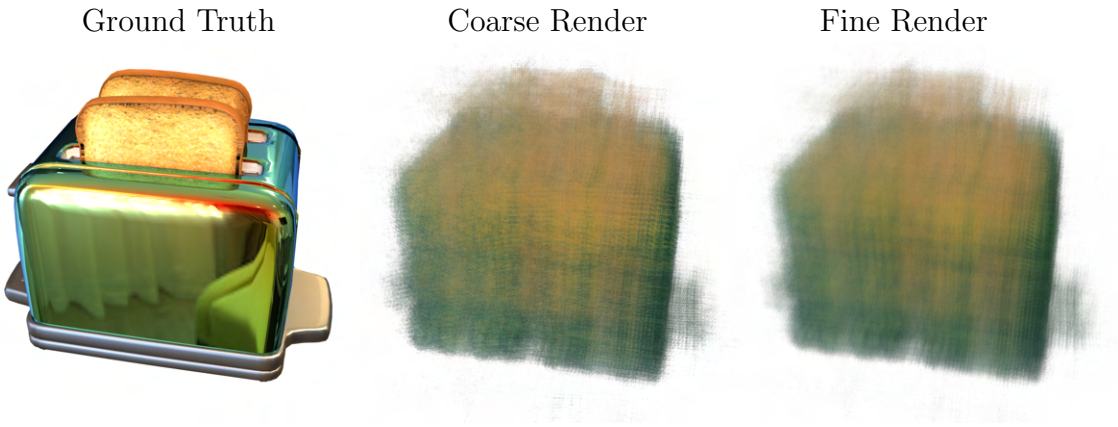
However, as shown in Figure 5-8, it struggles to accurately represents the crisp shininess of the surface. It renders the foggy reflections and lacks the ability to accurately represent reflections of objects visible on the toaster's surfaces. Figure 5-9 shows the accumulations and depth rendered from the optimized Mip-NeRF for StructColorToaster scene. These visualization shows that Mip-NeRF learns the overall surface geometry of the scene decently but fails to reconstruct the appearance. Overall, Mip-NeRF is able to represent the geometry but fails to represent complex appearance of surfaces with structural colors.



(a) PSNR↑ (b) SSIM↑

(c) LPIPS↓ (d) $L2$ Loss

Figure 5-7: Evaluation metrics on test views from the StructColorToaster scene.

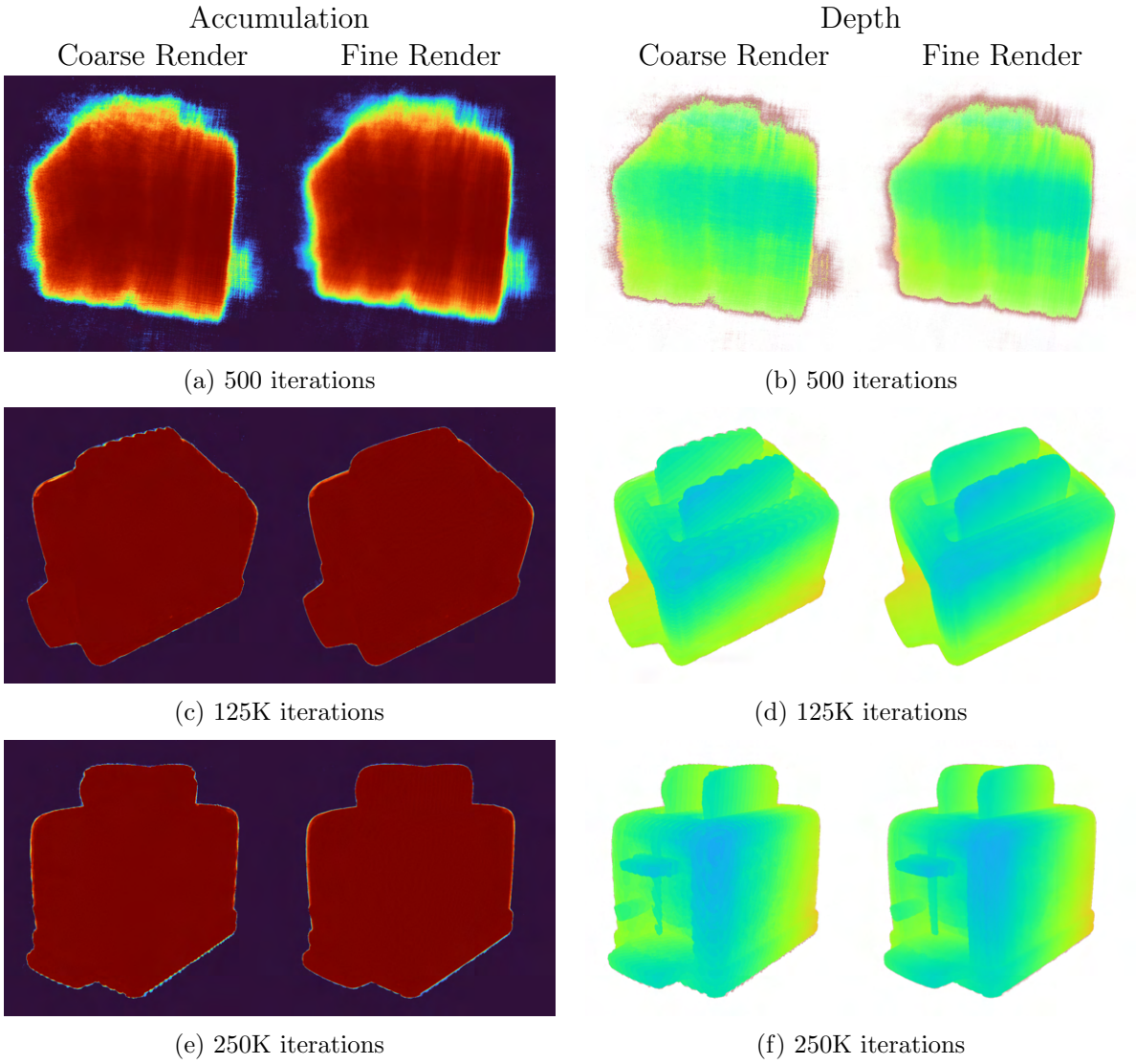


(a) 500 iterations

(b) 125K iterations

(c) 250K iterations

Figure 5-8: Renderings shows Mip-NeRF fails to learn appearance and specular reflections with increasing iterations.



(a) 500 iterations

(b) 500 iterations

(c) 125K iterations

(d) 125K iterations

(e) 250K iterations

(f) 250K iterations

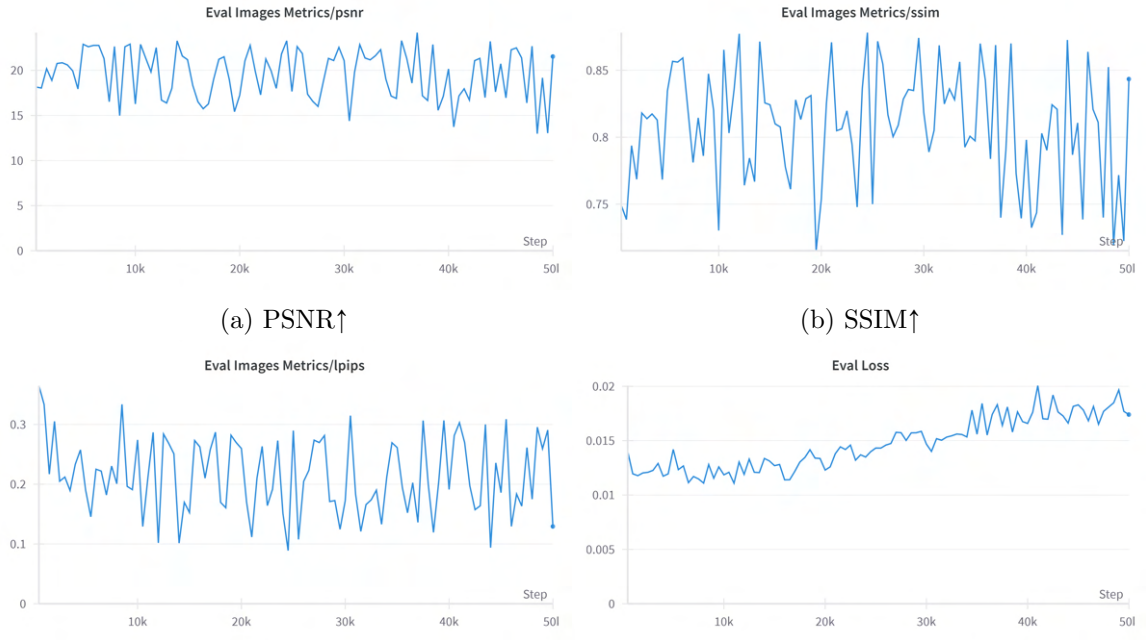
Figure 5-9: Accumulations and Depth visualizations from optimized Mip-NeRF.

5.1.4 NeRFacto

We train Nerfacto [70] for 50K iterations, which takes approximately 35 minutes.

As shown in Figures 5-10, NeRFacto achieves a PSNR↑ score of 19.305, SSIM↑ score of 0.8056, and LPIPS↓ score of 0.2123.

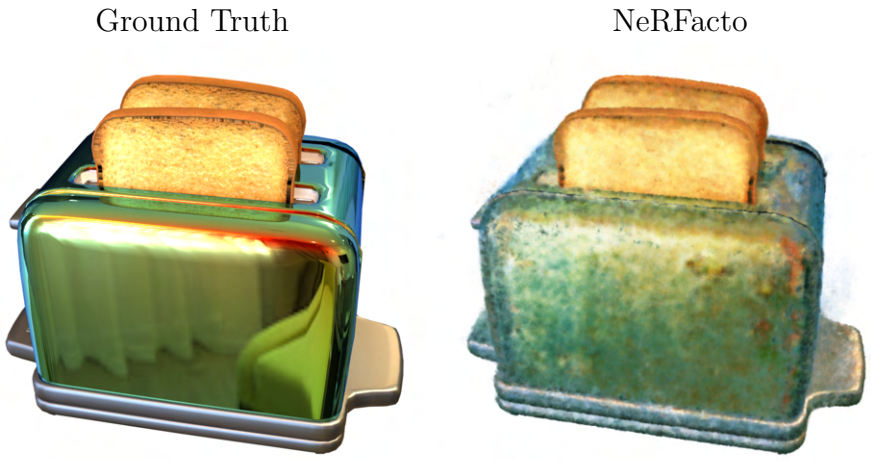
Figure 5-12 shows the accumulations and depth rendered from the optimized Mip-NeRF for StructColorToaster scene. These visualization shows that NeRFacto struggle to learn the surface geometry of the scene decently as there are some foggy surface renderings. And, as shown in Figure 5-11, it also struggles to accurately represents the glossy nature of the surface. It renders the foggy reflections and lacks the ability to accurately represent reflections of objects visible on the toaster's surfaces. Generally, NeRFacto struggle to represent the geometry and also fails to represent complex appearance of surfaces with structural colors.



(a) PSNR↑ (b) SSIM↑

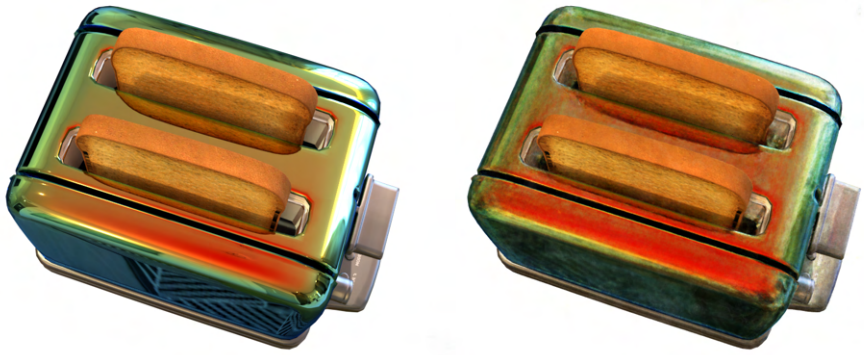
(c) LPIPS↓ (d) $L2$ Loss

Figure 5-10: Evaluation metrics on test images from the StructColorToaster scene.



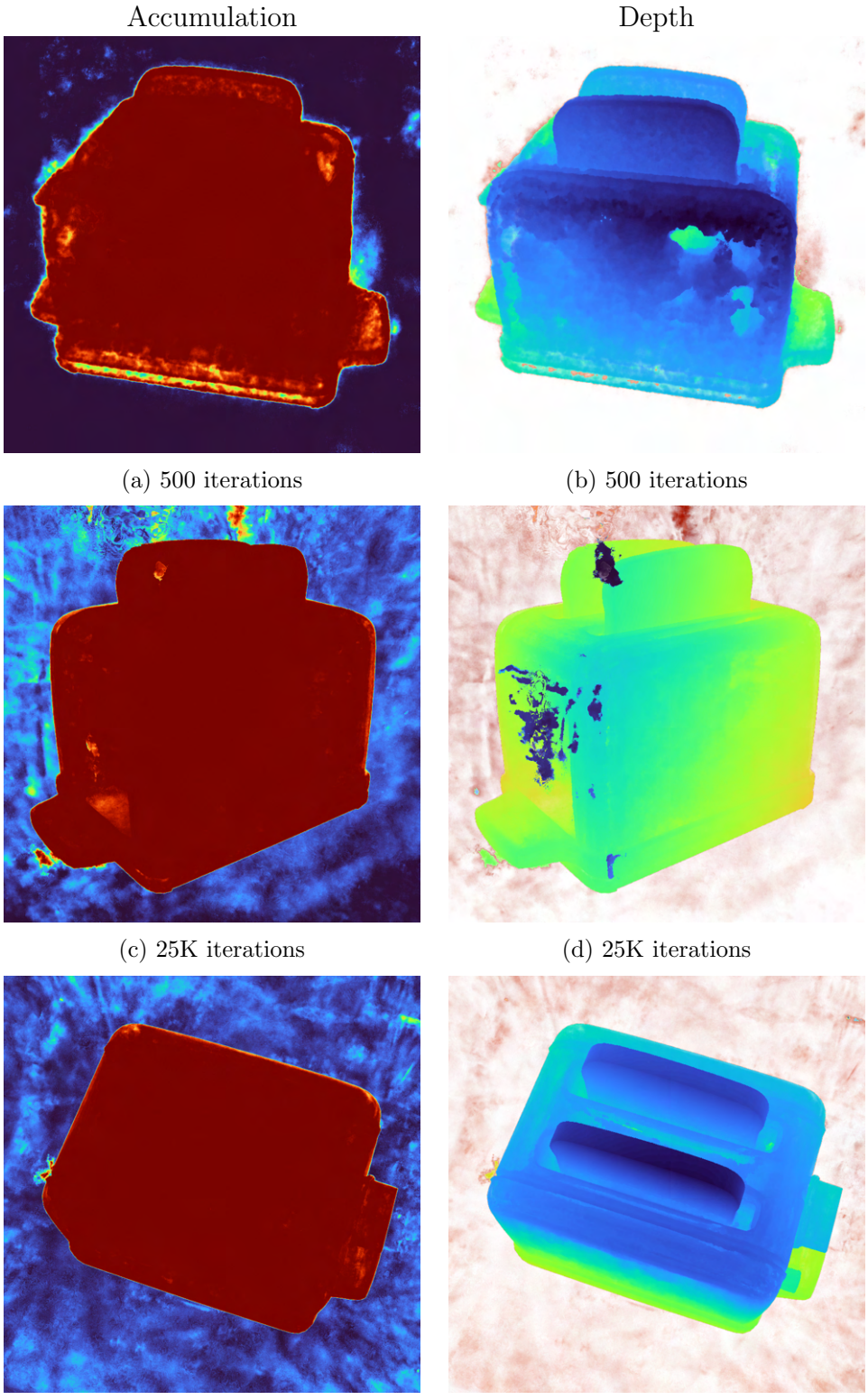
(a) 500 iterations

(b) 25K iterations



(c) 50K iterations

Figure 5-11: Renderings shows NeRFacto fails to learn appearance and specular reflections with increasing iterations.



(a) 500 iterations (b) 500 iterations

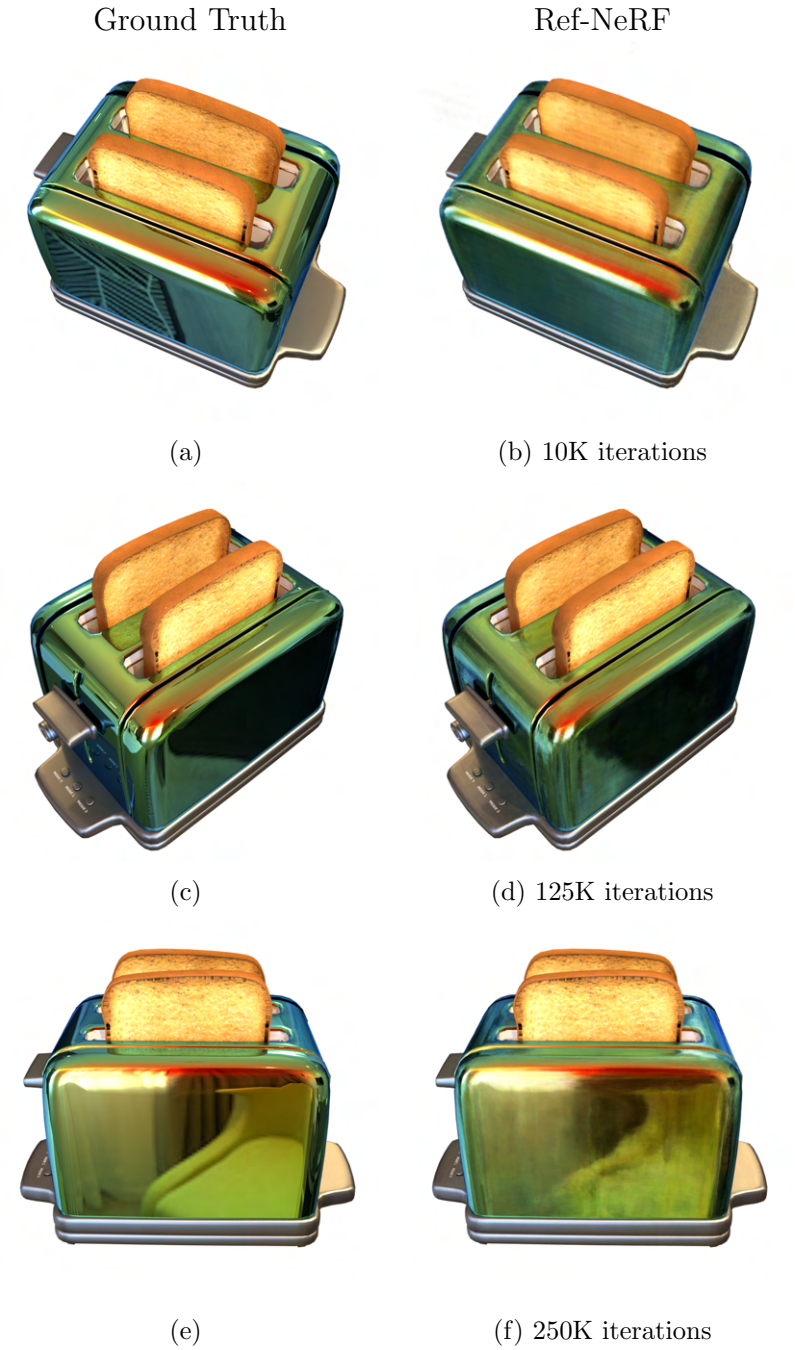
(c) 25K iterations (d) 25K iterations

(e) 50K iterations (f) 50K iterations

Figure 5-12: Accumulations and Depth visualizations from optimized NeRFacto. NeRFacto also fails represent the simple scene geometry efficiently.

5.1.5 Ref-NeRF

We train Ref-NeRF [76] for 250K iterations, which takes approximately more than 2 days. Ref-NeRF achieves a PSNR↑ score of 27.5194, SSIM↑ score of 0.9142, and LPIPS↓ score of 0.1286.



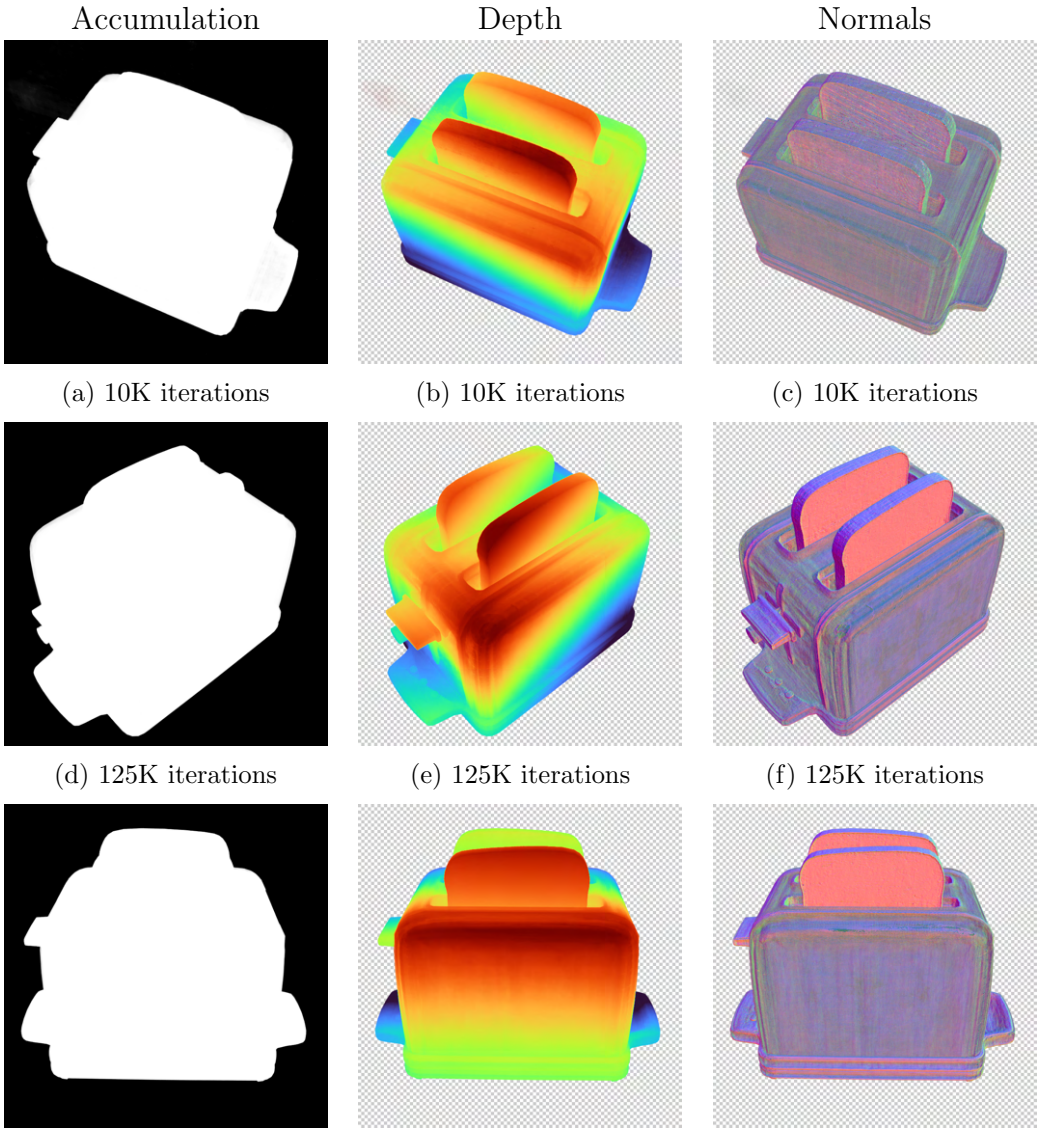
(a) (b) 10K iterations

(c) (d) 125K iterations

(e) (f) 250K iterations

Figure 5-13: The renderings show that Ref-NeRF fails to capture the crisp shiny appearance of the surface. However, it does improve in learning specular reflections with increasing iterations, though it fails to achieve good quality renderings.

As shown in Figure 5-13, Ref-NeRF is able to represent reflections of objects visible on the toaster's surfaces but not up to the mark and it still struggles to accurately represents the crisp glossiness of the surface. Moreover, reflections of intricate geometries with high-frequency details appear blurred in the renderings. Figure 5-14 shows the accumulations, depth, and Normals rendered from the optimized Ref-NeRF for StructColorToaster. These visualization shows that Ref-NeRF learns the overall surface geometry of the toaster scene accurately but not able to recover accurate normal vectors which leads to blurred reflections. Hence, Ref-NeRF is able to represent the geometry and complex appearance of surfaces with structural colors but quality of renderings still not up to the mark.



(a) 10K iterations (b) 10K iterations (c) 10K iterations

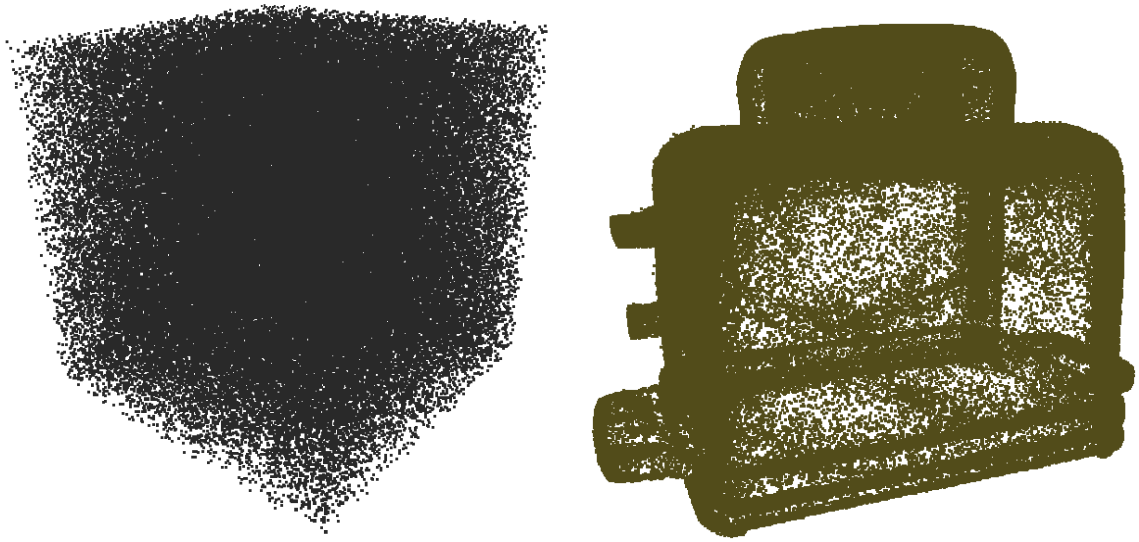
(d) 125K iterations (e) 125K iterations (f) 125K iterations

(g) 250K iterations (h) 250K iterations (i) 250K iterations

Figure 5-14: Accumulation, Depth and Normals visualizations from Ref-NeRF.

5.1.6 3D-GS

We train 3D-GS [32] for 30K iterations over period of approximately 25 minutes. As shown Figure 5-15, we start with 100K uniformly random Gaussians (Figure 5-15a) inside a volume that encloses the scene bounds and 3D-GS prunes them to about 6–10K meaningful Gaussians (Figure 5-15b) after optimization. After 30K iterations, 3D-GS achieves a PSNR↑ score of 24.0435, SSIM↑ score of 0.9065, and LPIPS↓ score of 0.1052.



(a) Initial point cloud

(b) Optimized point cloud

Figure 5-15: Illustration shows the initial point cloud used as input and optimized point cloud after training 3D-GS for StructColorToaster scene.

Optimized point cloud shown in Figure 5-15b shows that 3D-GS able to represent the geometry of object with structural colors correctly.

As shown in Figure 5-16, 3D-GS accurately represents the crisp glossiness of the surface. 3D-GS struggle to represent reflections of complex geometries with high-frequency details visible on the toaster's surfaces due to their view-dependent appearance. One possible reason for this aliasing effect also known as popping artifacts is simple visibility algorithm, which can lead to Gaussians suddenly switching depth or blending order. Therefore, 3D-GS effectively captures the geometry and intricate visual characteristics of surfaces featuring structural colors, yet the rendering quality remains subpar.

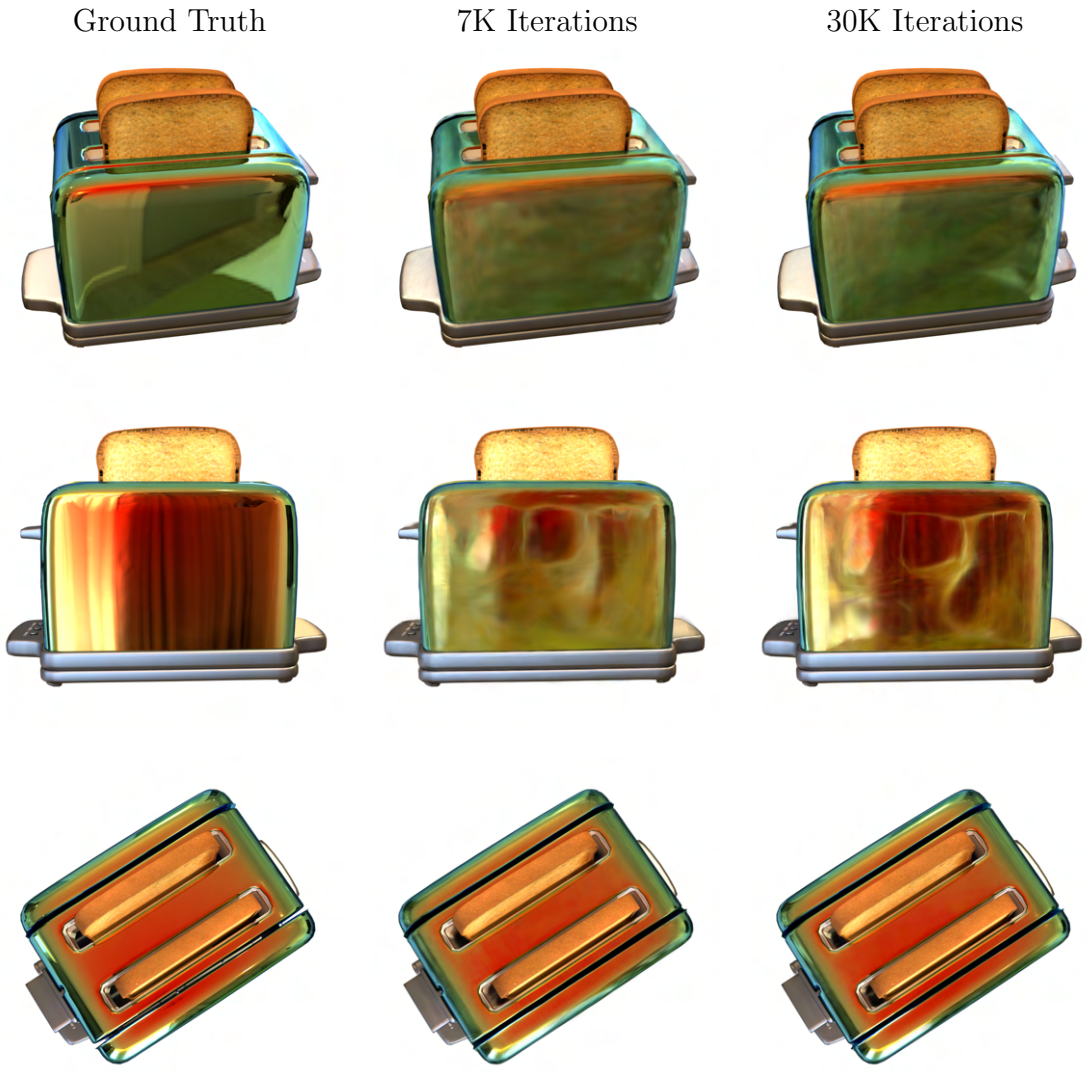


Figure 5-16: The renderings show that 3D-GS able to capture the crisp shininess appearance of the surface. However, it does improve in learning specular reflections with increasing iterations, though it is still not up to the mark.

5.2 Comparison of Results on StructColorToaster Scene

For consistent and meaningful comparisons, we generate Table 5.1 using the metrics most frequently in the literature such as the standard PSNR, L-PIPS, and SSIM metrics, along with training iterations, time, rendering speed and memory used to store optimized parameters.

	Iterations	PSNR ↑	SSIM ↑	LPIPS ↓	Train ↓	FPS ↑	Memory ↓
NeRF [48]	50K	23.359	0.8427	0.1895	1day	0.001	22MB
InstantNGP [51]	50K	19.659	0.8109	0.2068	23min	6	159.2MB
Mip-NeRF [3]	250K	22.818	0.877	0.1263	1day	0.03	15.9MB
NeRFacto [70]	50K	19.305	0.8056	0.2123	35min	0.6	168MB
Ref-NeRF [76]	250K	27.5194	0.9142	0.1286	2day	0.3	8.2MB
3D-GS [32]	30K	24.0435	0.9065	0.1061	20min	140	77MB
3D-GS [32]	60K	24.3117	0.9070	0.1052	30min	140	77MB

Table 5.1: Quantitative evaluation of selected methods' results computed over our StructColorToaster scene. The values for the Train time and FPS are approximate.

As shown in Table 5.1, purely NeRF based methods such as NeRF [48], Mip-NeRF [3] and Ref-NeRF [76] shows very slow training and rendering speed. On the other hand, methods that use explicit representation such as InstantNGP [51], NeRFacto [70], and 3D-GS [32] are very fast in training, and 3D-GS is particularly very fast in rendering than any other method.

In terms of rendering quality, Ref-NeRF and 3D-GS are comparable, but due to its real-time rendering capability, 3D-GS surpasses Ref-NeRF. Also, 3D-GS represents reflections of complex geometries with high-frequency details visible on the toaster's surfaces and their view-dependent appearance better than Ref-NeRF.

Why 3D-GS?. We choose 3D-GS for our further research using our real dataset of laser printed objects with structural colors because it offers the advantages of both implicit and explicit representation-based methods:

- **Faster rendering**, as 3D-GS uses 3D Gaussians, which are unstructured explicit representations like meshes enabling real-time rendering.
- **Faster optimization**, as 3D Gaussians inherits the properties of differentiable volumetric representations, enabling faster optimization and faster splat-based rasterization.
- **Quality rendering**, as 3D-GS renders the crisp glossy/shiny surface, reflections of complex geometries with high-frequency details visible on the object's surfaces, and extreme view-dependent appearance of structural colors.

Chapter 6

Real-World Scene Dataset

In this section, we briefly discuss how the structural color object is created in the lab using laser printing. Then, we delve into the evolution of the capture setup to generate the real-world scene dataset using these objects. Finally, we provide a concise overview of how COLMAP [59] is used to estimate the poses and structure from motion (SfM) point cloud.

6.1 Structural Color Objects

The Artificial Intelligence aided Design and Manufacturing group (AIDAM), develops AI algorithms to automate the laser processing of materials. Research group's Image Reproduction software module uses advanced computer graphics algorithms to laser-mark any color image on a range of substrates, including stainless steel and titanium. Final product of this software module is laser printed structural-color image-painting on the metal substrate plates. We aim to develop a method with which potential users can visualize this class of objects and understand the intricacies of angle dependent structural-color image/painting, thereby transforming these objects into their digital twins for better interactive visualization for research purposes.

As shown in Figure 6-1, the phenomenon of structural colors arises from a laser-induced oxide layer formed on a highly absorbent yet durable ceramic film composed of titanium aluminum nitride ($TiAlN$). The adjacent titanium nitride (TiN) layer functions as a reflector. The process involves large-scale printing using meander scanning techniques. Through the interference of light at the interfaces, there emerges a pronounced resonant absorption, leading to the exhibition of adjustable colors. Here, t_1 denotes the thickness of the laser-induced oxide layer (medium 1), t_2 signifies the thickness of the remaining $TiAlN$ film (medium 2), and t_3 represents the thickness of TiN (medium 3). By simultaneously altering t_1 and t_2 with the laser, a range of reflected colors can be achieved [16].

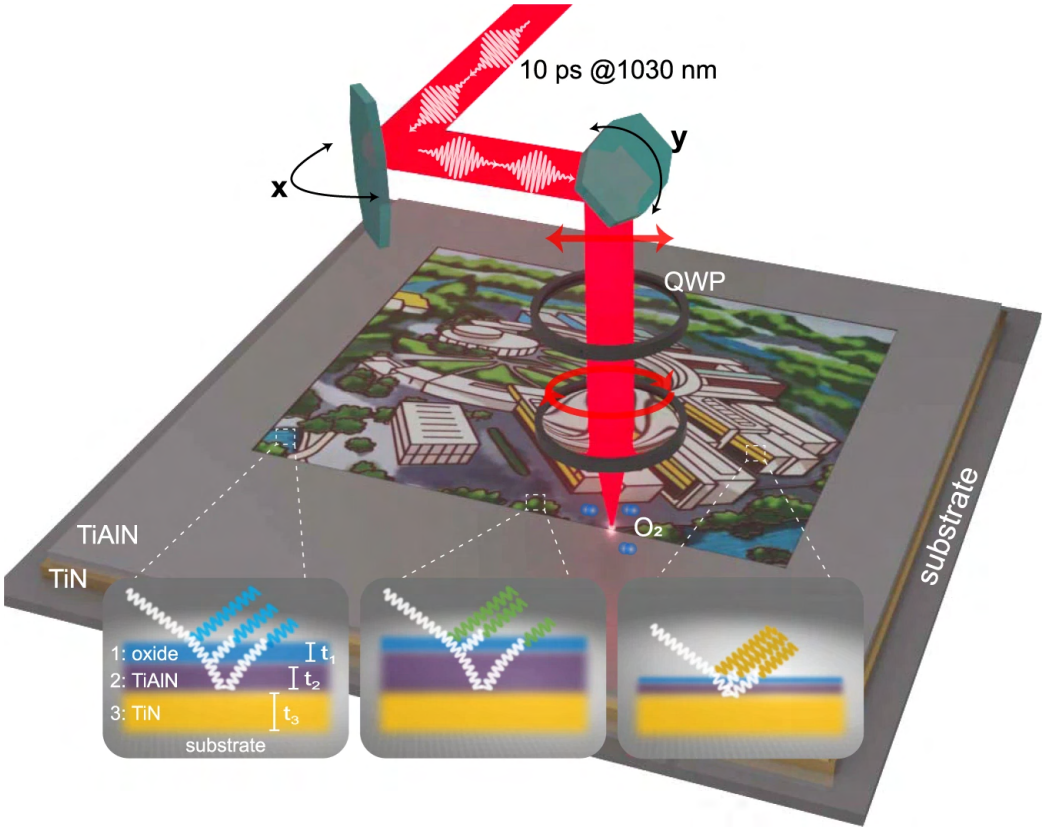


Figure 6-1: An illustration of generating structural color image-painting object using laser printing in lab. Images taken from [16].

Figure 6-2 shows view-dependent nature of structural colors object created using laser printing in the AIDAM lab. Even with the slight change in viewing angle, appearance of the objects changes rapidly. For example, glossy green and black from viewing direction 1 changes to black and white respectively in viewing direction 2.

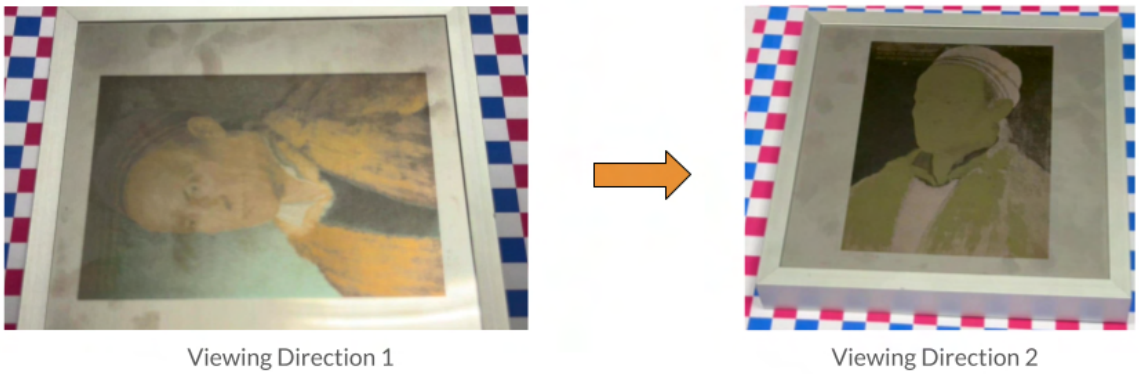


Figure 6-2: An illustration of view-dependent appearance of structural color image-painting object laser printed in lab.

6.2 Evolution of Capture Setup

Capturing structural color objects with a handheld smartphone camera presents numerous challenges due to their highly specular nature and extreme view-dependent appearance. Some of these challenges include:

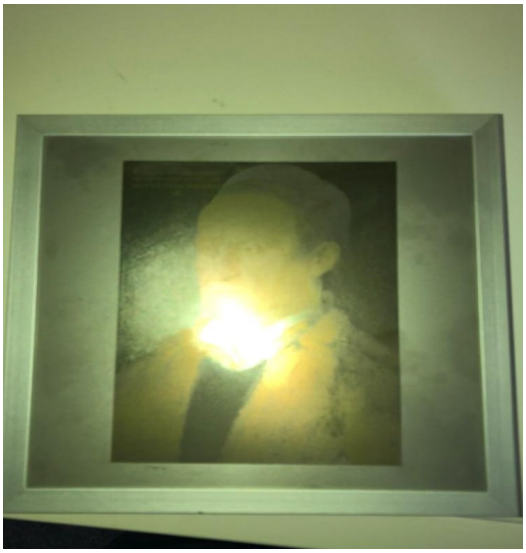
- Lighting and Environmental Changes: Variations in lighting conditions, such as shifts in sunlight and surrounding environment, create difficulties in achieving consistent image capture. Because these objects have metal surfaces that strongly reflect light and their surrounding environment, changes in lighting can significantly impact the captured images.
- Texture-less Surfaces: The reflective metal surfaces of these objects are essentially texture-less surfaces, which poses challenges to Colmap [59], which relies on textures for accurate alignment. Without distinct textures as reference points, Colmap struggles to achieve precise alignment, potentially leading to inaccuracies in the reconstruction process.
- Noise Minimization and Quality: The capture process is essential for achieving high-quality results. This involves managing exposure settings, camera positioning, and the number of images captured. Both an excessive and insufficient number of images can introduce artifacts or distortions in the final reconstruction. Therefore, finding the right balance is crucial for achieving optimal outcomes.

We aim to maintain a simple and affordable capture setup. To achieve this and to obtain higher quality captured images depicting the view-dependent appearance of this class of objects, we experimented with various capture settings. This included testing different smartphone cameras, as handheld cameras are our primary choice for capturing, along with different camera settings, light conditions, and capture environments.

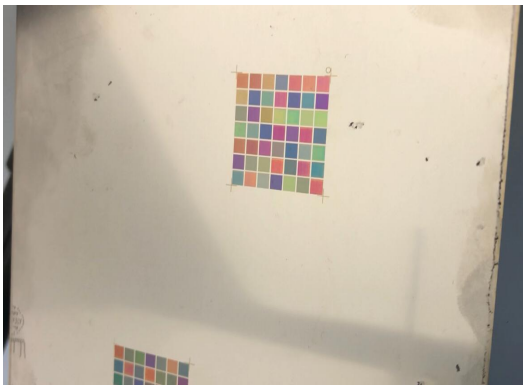
Figure 6-3 illustrates the different challenges posed by the highly reflective surfaces of these objects, such as reflections of lighting, shadows, and objects from the surrounding environment, resulting in highly noisy images.

To address these difficulties, we experimented with using a photobox equipped with a light source positioned both inside and outside the photobox. However, the reflective surfaces of the objects led to pronounced reflections of shadows cast by the person's body parts holding the camera, the handheld smartphone camera, and the light source itself, as depicted in Figure 6-4. Furthermore, capturing the object by handheld smartphone camera from the upper hemisphere to encompass a wider range of viewing angles was not feasible due to space constraints. Consequently, we abandoned this approach and explored alternative strategies.

(a) Reflection of shadows generated by objects from surrounding environment



(b) Reflection of lighting from highly reflective metal surface



(c) Reflection of shadows from highly reflective metal surface

(d) Reflection of lightings, shadows and surrounding environment

Figure 6-3: Few samples of captures without controlled settings.

(a) Photobox with light source to capture objects

(b) Sample captured image of object using photobox and single light source

Figure 6-4: Experiments with photobox capture method under controlled settings.

After experimentation, we developed the capture setup similar to illustrated in Figure 6-5. This setup allows users to rotate around the object effortlessly, capturing various viewing angles from the upper hemisphere in the object's space. Additionally, users can adjust their distance from the object to obtain images with different depths. By positioning four rectangular area light sources on the ceiling and away from the reflective surface of the object, we were able to mitigate strong light reflections, resulting in images with less noise and true view-dependent colors.

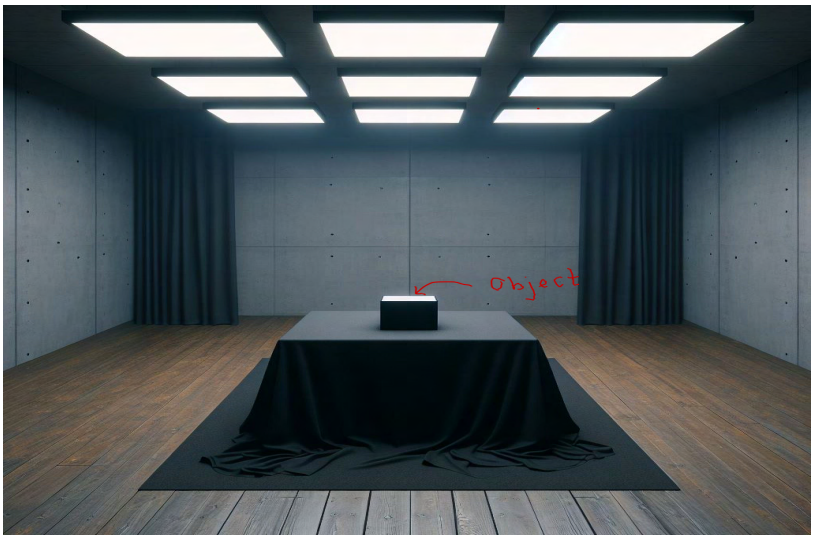


Figure 6-5: The capture setup we used closely resembles to this studio room. (For illustration purposes only.)

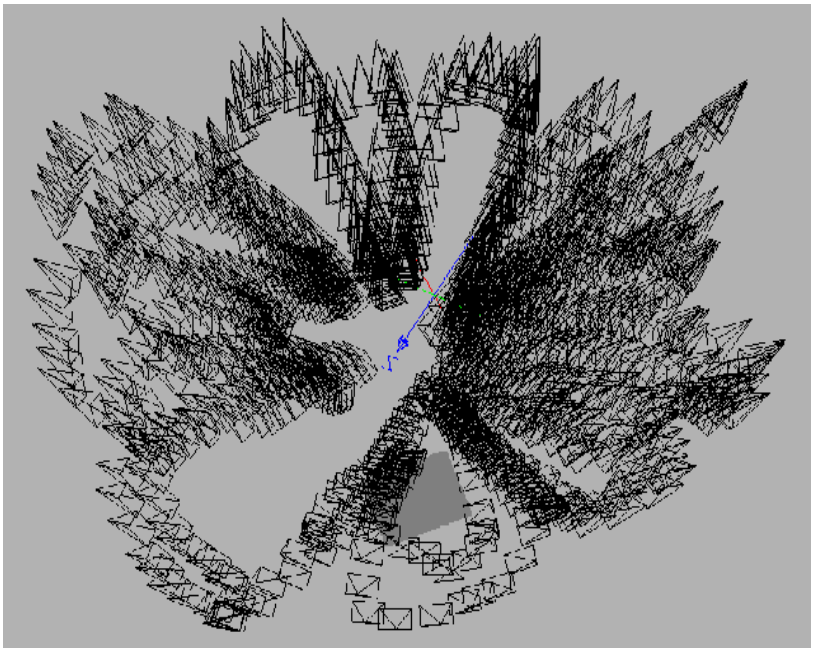


Figure 6-6: Capture path used to capture images of the object.

There are different capture patterns available such as grid, circular, and random pattern. We chose to capture the object with circular pattern, as illustrated in Figure 6-6. This involves circling around the object, approaching it, and then moving away periodically across the upper hemisphere of the object's space. We use the rear camera of an iPhone 15 Pro smartphone to capture video of an object, adjusting the focus and brightness manually while disabling features such as Auto Exposure, Auto Focus, and Auto White Balance. We use FFmpeg to extract images from the captured video.

For COLMAP [59] to work successfully for better pose estimation and 3D-reconstruction, we place checkerboard under the object to compensate for texture-less surface and capture images with sufficient overlap between consecutive images to ensure better correspondences between features. Few sample images from our best performing capture are shown in Figures 6-7 and 6-9. And, Figure 6-8 shows sparse point cloud reconstructed as byproduct while performing camera calibration using COLMAP [59].

Figure 6-7: Few samples from real-world dataset scene.

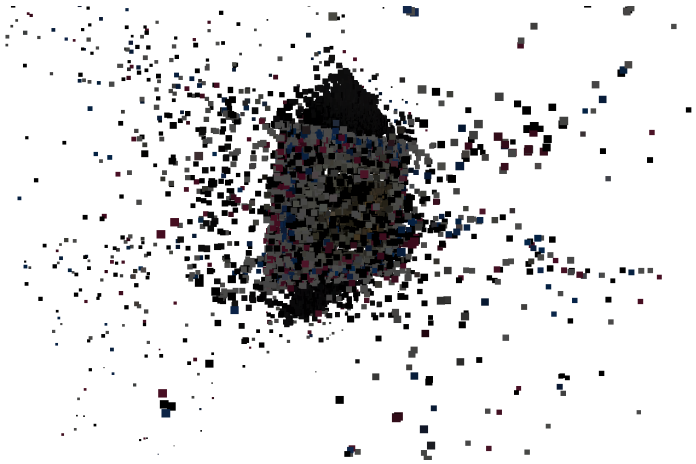


Figure 6-8: Very noisy Sparse Point Cloud reconstructed using COLMAP for our real-world scene.

6.3 StructColorPainting Scene

We call our best performing capture as the StructColorPainting scene, which represents our real dataset. It consists of 891 images of dimensions 1920 × 1080 as shown in Figures 6-7 and 6-9. We perform experiments with different modifications and settings of 3D-GS and StructColorPainting scene.

Figure 6-9: Few more samples from real-world dataset scene.

Chapter 7

Experiments with Real-World Scenes

In this chapter, we discuss experiments performed with the StructColorPainting scene and 3D-GS [32], using different settings and modifications. We also discuss the quantitative and qualitative findings obtained after experiments on the StructColorPainting scene and validate the design choices and parameters of 3D-GS with an extensive ablation study. We show using the geometric prior of planar structural-color object while initializing scene with sparse structure-from-motion (SfM) point cloud significantly improves the visual quality of view-synthesis by reducing floaters. We then show StructColorTaylorSwift scene, captured under simplified lighting conditions along with our techniques, improves the quality of scene optimization and renderings. Finally, we briefly discuss how considering these findings is important for future research direction.

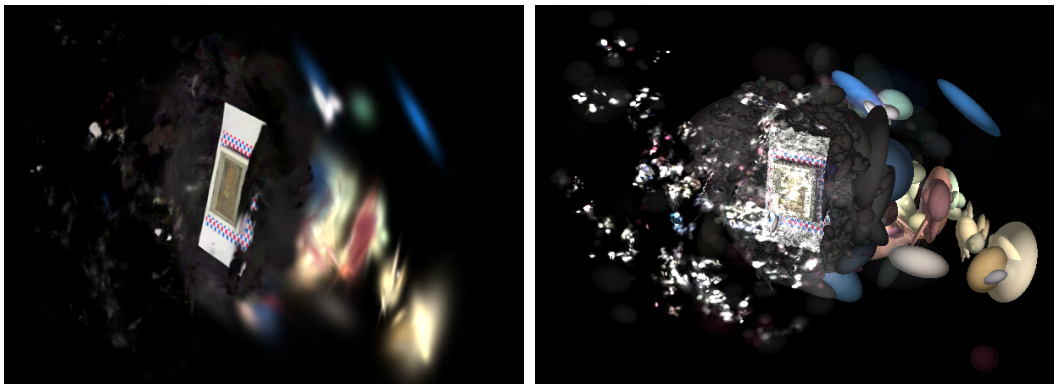
7.1 Experiments

We experiment with initialization SfM point cloud, hyperparameters such as scene background color (black, white, and random), spherical harmonics degree, number of training iterations, iteration where densification stops, how frequently to reset opacity, scaling learning rate, and loss function for a better distribution of points over a surface.

7.1.1 Vanilla 3D-GS

We experiment with vanilla 3D-GS on our StructColorPainting scene and an unmodified SfM point cloud obtained from COLMAP [59]. After training for 30K iterations, the optimized scene representation shows floaters and grainy appearance. As shown in Figure 7-1, the optimized scene has floaters and a grainy appearance possibly related to initialization with noisy SfM points. Also, the optimized Gaussians are not compacted in a small space nearby object's surface, instead spread over a large space

behind the object. Therefore we also experiment with cleaned point cloud shown in the Figure 7-3.



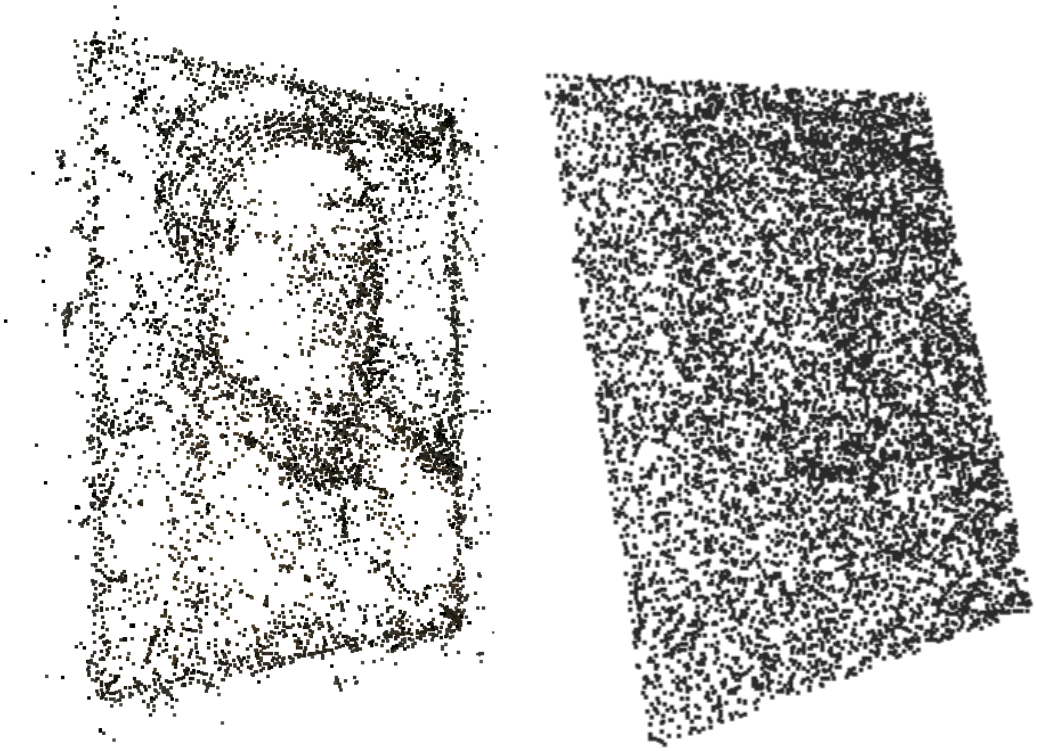
(a) Floaters and grainy artifacts in the scene. (b) Ellipsoidal representation of the scene.

Figure 7-1: Floaters and grainy artifacts appear when the scene is optimized with original SfM point cloud.

7.1.2 Initialization with Cleaned-SfM Point Cloud

Point cloud without normals produced by COLMAP is very noisy, as shown in Figure 6-8. We found cleaned point cloud can generate better renderings without floaters. As our object is planar, we fit the plane using Random Sample Consensus (RANSAC) algorithm [38] along the point cloud, and then we add more points with random placements in the plane to augment these SfM points. We use these augmented SfM points for initialization of 3D-GS. As shown in Figures 7-2 and 7-4, initialization with cleaned SfM pointcloud shows a better optimized scene with almost no floaters. It also achieves faster optimization and better visual quality of our scene representation.

Figure 7-2: Floaters are Reduced but grainy artifacts appear in SIBR viewer.



(a) Manually cleaned SfM points

(b) Automatically cleaned using RANSAC algorithm and augmented SfM points

Figure 7-3: Initializing scene optimization using automatically cleaned points using RANSAC and augmented SfM point cloud for StructColorPainting scene.

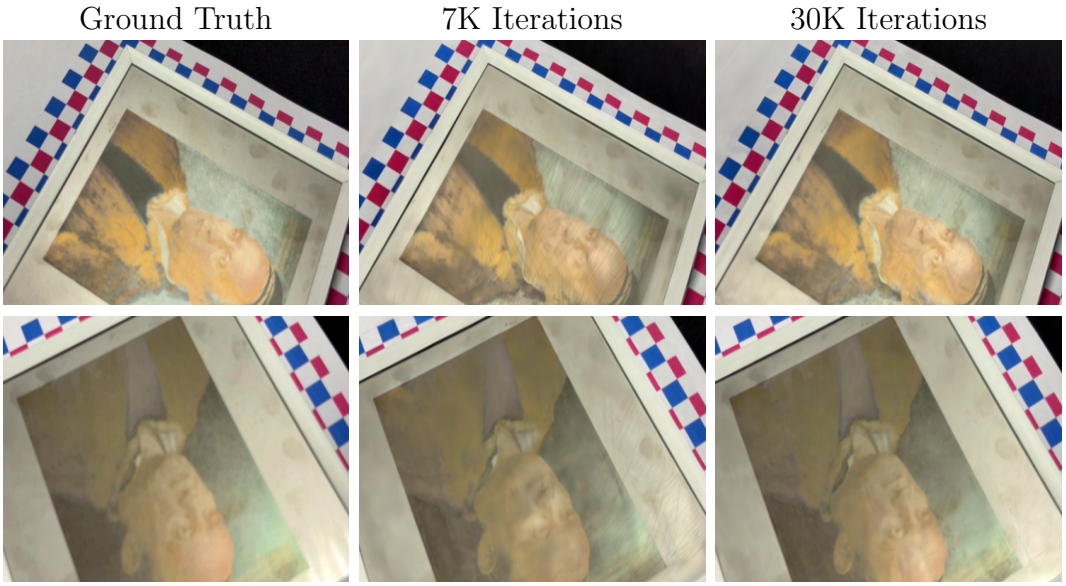


Figure 7-4: The renderings show that 3D-GS able to reduce the floaters and grainy artifacts with clean SfM points initialization.

7.1.3 With Masks

For these experiments we use images little different from StructColorPainting scene as we use different lightnings and background in our capture setup. We call it StructColorPaintingOld scene as it is capture from the initial stages of research. We create masks of these captured images from StructColorPaintingOld scene using Grounded-Segment-Anything [55]. We train the 3D-GS with these masks to represent the scene to include only object we are interested in. While optimizing the scene, we replace the corresponding pixels in the ground truth image with those from the rendered image where the mask is True. This operation effectively updates the ground truth image in regions where the mask is active, potentially refining the training process by focusing on specific areas of interest. However, we do not achieve a quality scene representation due to spiky Gaussians generated during optimization, as shown in the Figure 7-6.

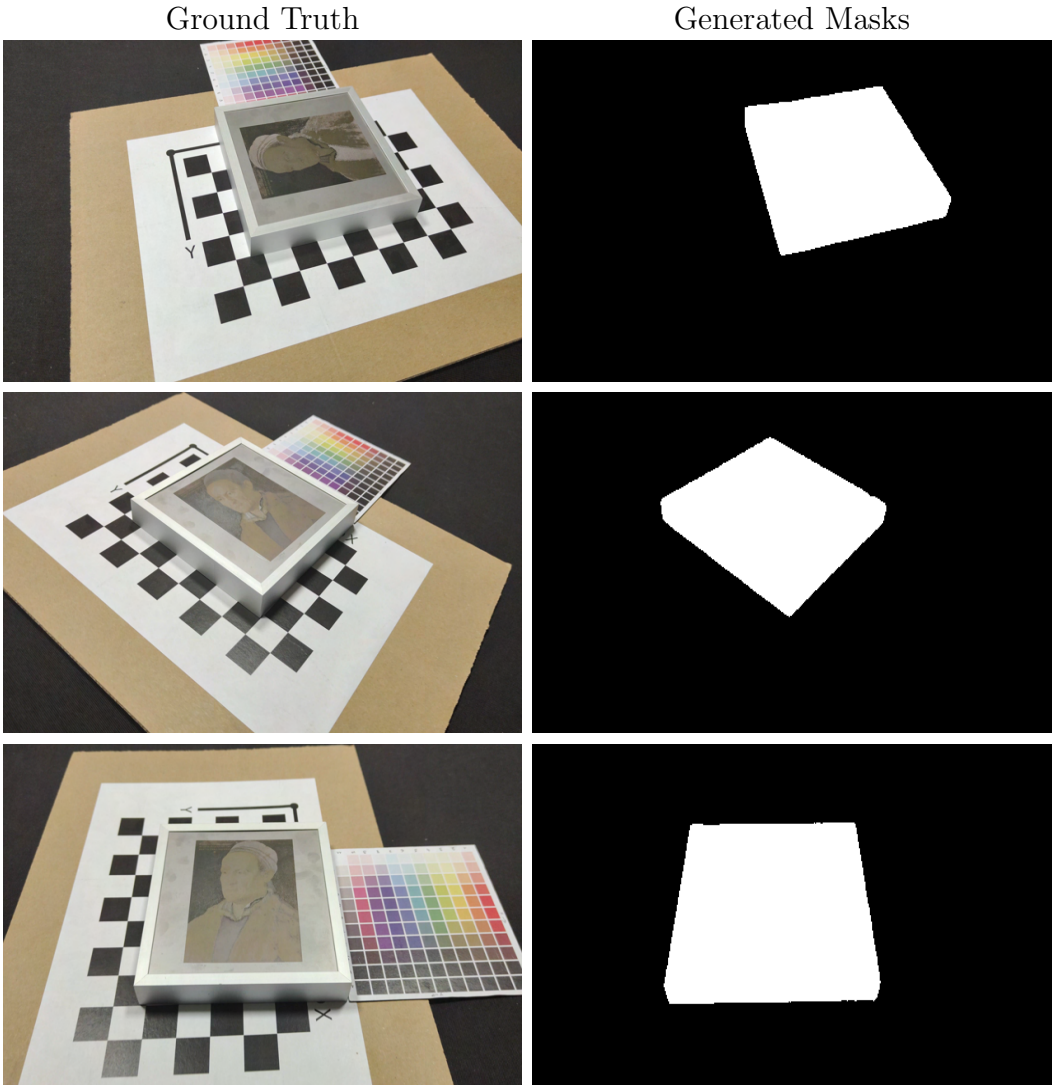


Figure 7-5: Few sample images and respective generated masks from StructColorPaintingOld scene.

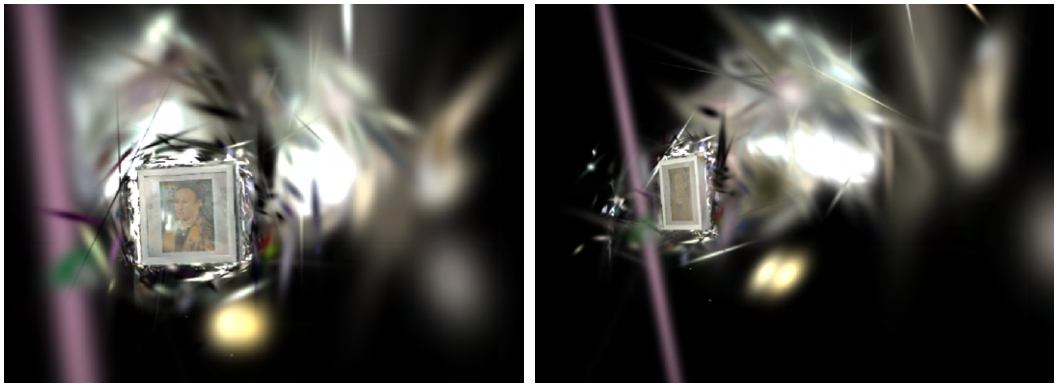


Figure 7-6: Floaters and spiky Gaussian artifacts visualized in SIBR viewer.

7.1.4 With Cropped Images Using Masks

For these experiments, we use images from the StructColorPaintingOld scene. We crop the images to include only object we are interested in, using the masks of these captured images obtained using Grounded-Segment-Anything [55], as shown in the Figure 7-7.

We optimize our StructColorPaintingOld scene using the 3D-GS with these cropped images as ground truths to represent the scene to include only the object in which we are interested. This operation effectively refines the training process by focusing on specific areas of interest. We achieve a decent quality of scene representation but there are some grainy and popping artifacts appear while visualizing the scene in SIBR viewer 7-8. Possible reasons for these popping artifacts are the trivial rejection of Gaussians via a guard band in the rasterizer and a simple visibility algorithm, which can lead to Gaussians suddenly switching depth/blending order.

Also, it is sometimes difficult for Grounded-Segment-Anything [55] correctly predict the mask for structural color object from challenging viewpoints, which can result in masking pixels from the structural color object itself. This, in turn, reduces the crisp shininess nature of view-dependent structural colors, as shown in Figure 7-8.

We also demonstrate that not optimizing the positions of Gaussians and not densifying or pruning Gaussians does not effectively optimize the scene, as shown in the experiment in Figure 7-9.

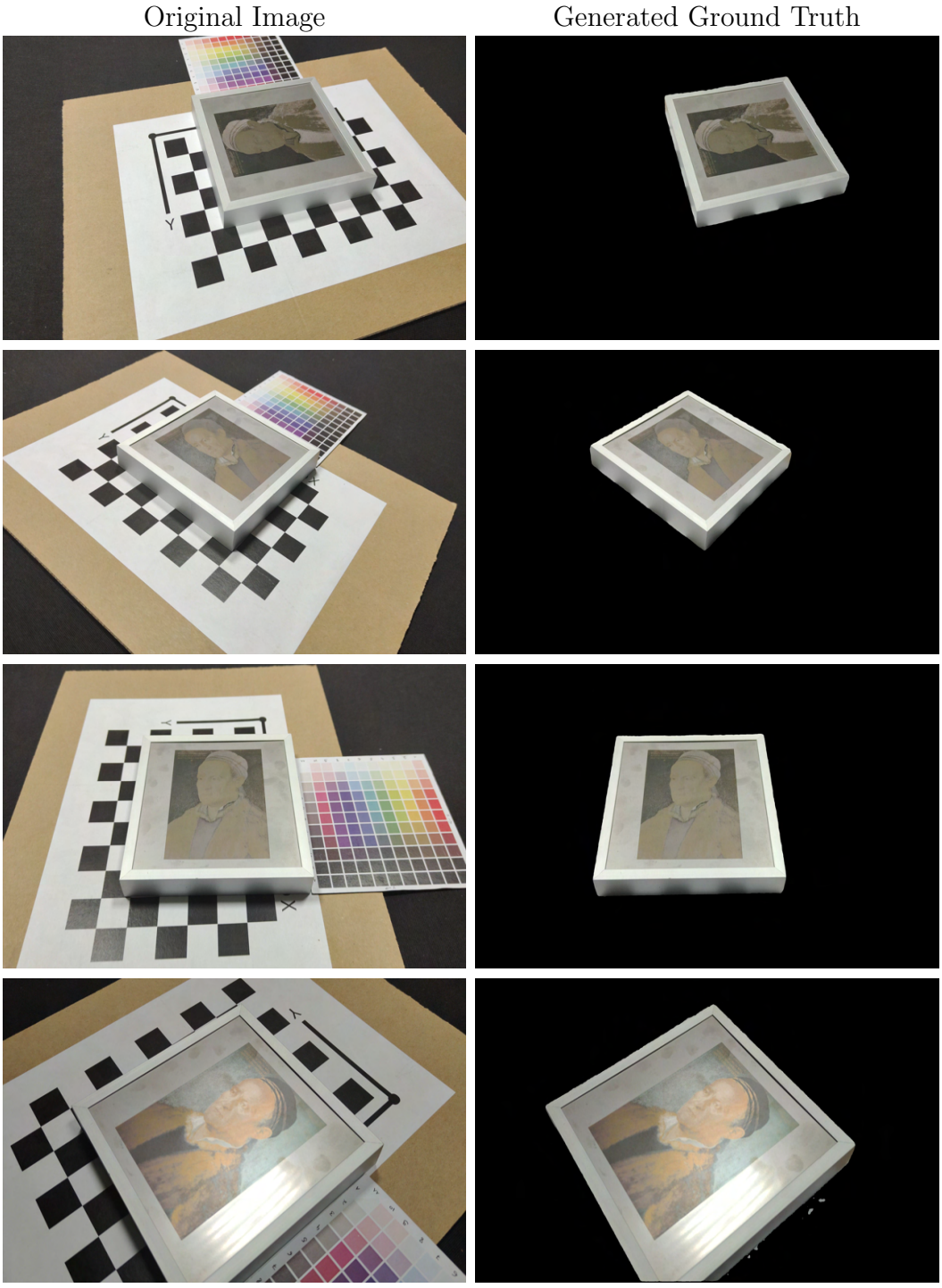


Figure 7-7: Few sample Original Images and respective generated Ground Truths using masks from StructColorPaintingOld scene.

Figure 7-8: Floaters are reduced, but grainy and popping artifacts appear in the SIBR viewer. Also, crisp shininess nature of view dependent structural colors is reduced.

When we don't optimize positions of Gaussians and don't densify Gaussians then we get very blurry scene representation as shown Figure 7-9.

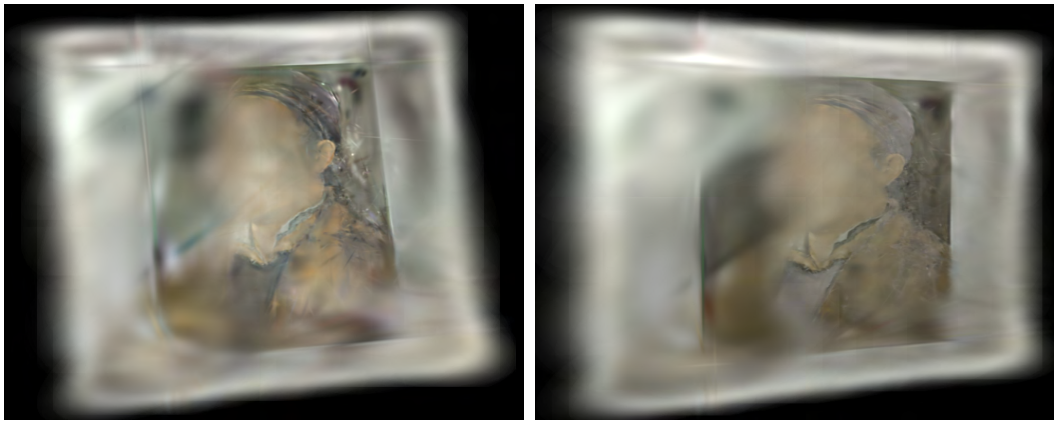


Figure 7-9: Blurry renderings of scene in SIBR viewer with no densification and position optimization of Gaussians.

7.1.5 Hyperparameters

We experiment on our StructColorPainting scene with hyperparameters such as the scene background color (black, white, and random), spherical harmonics degree, number of training iterations, iteration where densification stops, how frequently to reset opacity, and scaling learning rate. Default values for these hyperparameters are: background is black, spherical harmonic degree is 3, total train iterations are 30K, iteration where densification stops is 15K, densify for every 100 iterations, and opacity reset interval is 3000. We found changing hyperparameters to values other than the defaults does not significantly improve the reconstruction quality.

7.1.6 Loss Function - Anisotropy Regularizer

We found in 3D-GS, optimization creates large Gaussians in regions with strong view-dependent appearance like structural colors, and therefore we experiment with *Anisotropy Regularizer* introduced in PhysGaussian [83]. Anisotropy regularizer is a scale regularizer that encourages Gaussians to be more evenly shaped and reduces spikey Gaussians.

$$\mathcal{L}_{aniso} = \frac{1}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} \max\{\max(\boldsymbol{S}_p)/\min(\boldsymbol{S}_p), r\} - r, \tag{7.1}$$

where S_p are the scalings of 3D Gaussians. This loss essentially constrains that the ratio between the major axis length and minor axis length does not exceed r. We experiment with $r = 1.0$ which is a threshold of ratio of Gaussian max to min scale. Figure 7-10 shows little improvement in the renderings.

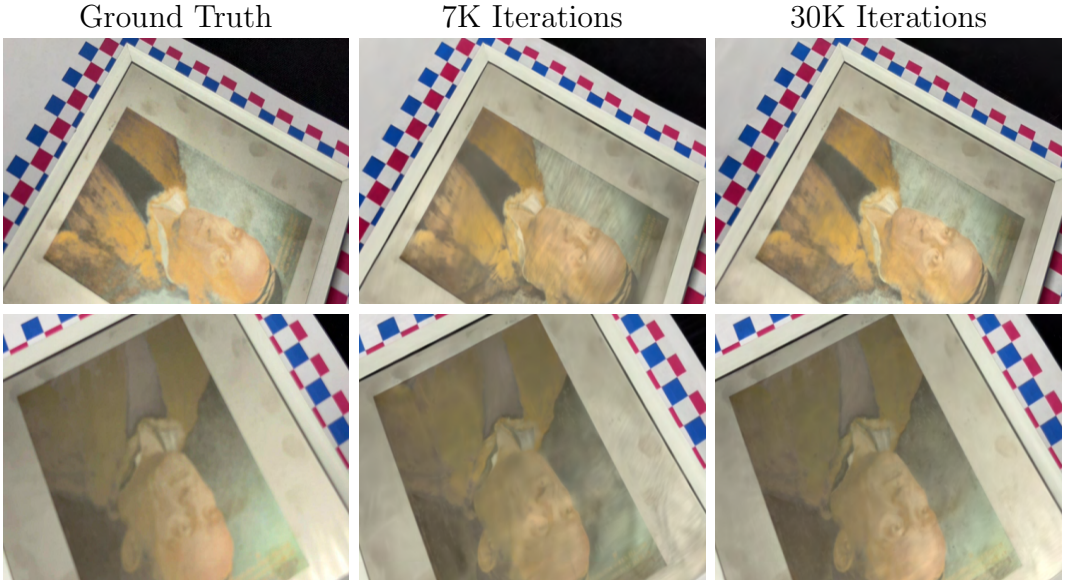


Figure 7-10: Anisotropy Regularizer improves the renderings in SIBR viewer as it keeps the spiky Gaussians in check.

7.2 Findings

The quantitative and qualitative effect of each choice is summarized in Table 7.1. We demonstrate that both Cleaned-SfM and the Anisotropy-Regularizer improve renderings in the SIBR viewer visually, qualitatively, and quantitatively. Changing hyperparameters to values other than the defaults does not significantly improve the rendering quality. We found higher degree spherical harmonic function able to capture view-dependent effects of structural colors and improve PSNR. Also initialization with cleaned-SfM points helps in reducing floaters. Further, we do full evaluation of our best performing techniques such as *Cleaned-SfM*, *Anisotropy-Regularizer*, and *Anisotropy-Regularizer with fewer images*.

	StructColorPainting-7K	StructColorPainting-30K
Vanilla 3D-GS	27.11	28.92
Cleaned-SfM	27.6715	**29.1789**
With-Masks	9.352	9.4603
With-Cropped-Images/RGBA Images	29.1759	30.0007
With-Cropped-Images/RGBA Images: (No-Densification + No-Position Optimization)	24.4612	25.089
Densify Until Iteration = 30000	26.8325	29.4149
Densification Interval = 30	25.2634	28.6454
Densification Interval = 50	26.0189	28.2718
Reset Opacity = Every 1000 Iteration	27.7216	29.2581
Reset Opacity = Every 2000 Iteration	27.6354	29.1786
Reset Opacity = Every 5000 Iteration	27.82	29.2726
SH-Degree 0	26.4699	27.9771
White-Background	27.6579	29.2025
Random-Background	27.7607	29.3303
No-Densification	17.7919	18.3929
Reduced Number of Images = 297	27.3657	28.7223
Anisotropy-Regularizer	27.5747	**29.4191**

Table 7.1: PSNR Score for ablation runs. Quantitative evaluation of results computed over our StructColorPainting scene using different 3D-GS settings.

	Iterations	PSNR ↑	SSIM ↑	LPIPS ↓	Train ↓	FPS ↑	Memory ↓
Cleaned-SfM	7K	27.6452	0.9221	0.1795	5.122min	150	79MB
	30K	29.1789	0.9342	0.1602	27.69min	120	150MB
Anisotropy-Regularizer	7K	27.5476	0.9223	0.1780	5.134min	150	78.6MB
	30K	**29.9631**	**0.9346**	**0.1572**	26.35 min	120	152.4MB
Anisotropy-Regularizer: 297-Images	7K	26.9920	0.9238	0.1833	5.812 min	150	86.1MB
	30K	29.3213	0.9339	0.1648	28.57min	120	176.2MB

Table 7.2: Quantitative evaluation of selected techniques' results computed over our StructColorPainting scene. The values for FPS are approximate.

As shown in the Table 7.2, all of our best-performing techniques, such as Cleaned-

SfM and Anisotropy-Regularizer, produce approximately the same good quality renderings. However, Anisotropy-Regularizer exhibits better PSNR, SSIM, and LPIPS metrics. Furthermore, with just 297 images instead of 891, the Anisotropy-Regularizer technique, along with Cleaned-SfM, performs well and achieves comparable quantitative metrics and visual quality.

As shown in the Figure 7-11, when visualizing a structural color object in the SIBR viewer, view rotation introduces popping and blending artifacts in regions with view-dependent appearance. Possible reasons for these artifacts and aliasing effects are the trivial rejection of Gaussians via a guard band in the rasterizer, which can cause optimization to produce large Gaussians and 3D-GS's simple visibility algorithm, which can cause Gaussians to suddenly switch depth or blending order.

Figure 7-11: Visualizations of popping, blending, and aliasing artifacts in SIBR viewer while navigating StructColorPainting scene trained with 3D-GS.

We also found that renderings using the 3D-GS rasterizer are free from popping and blending artifacts shown in Figures 7-12 and 7-10, compared to the renderings during interactive visualization in the SIBR viewer shown in Figure 7-11.

Figure 7-12: Visualization of rendered view using 3D-GS rasterizer.

Using another dataset called StructColorTaylorSwift, consisting of 514 images of dimensions 1920 × 1080, in which we minimized scene complexity by decreasing the number of reflections with simplified lighting settings, by reducing the area light sources from four to two while capturing the dataset. And then applying our techniques such as Cleaned-SfM and Anisotropy-Regularizer during 3D-GS optimization, we improved the overall structural color effect and PSNR value to 33.6547 shown

in Table 7.3. We believe that having good lighting settings during dataset capture influences the quality of reconstruction and the structural color effect.

	Iterations	PSNR ↑	SSIM ↑	LPIPS ↓	Train ↓	FPS ↑	Memory ↓
Our Techniques	7K	30.2567	0.9338	0.1921	4.123min	150	55MB
	30K	33.6547	0.9458	0.1749	18.5min	120	94MB
gsplat	7K	31.8051	0.9391	0.1880	2.86min	150	64MB
	30K	34.6451	0.9511	0.1658	16.61min	120	119MB
gsplat + Mip-Splatting	7K	31.3528	0.9362	0.1922	2.54min	150	66MB
	30K	34.6654	0.9379	0.1827	17.1min	120	124MB

Table 7.3: Quantitative evaluation of renderings using all selected techniques' computed over our StructColorTaylorSwift scene. The values for FPS are approximate.

As shown in Figures 7-13, 7-14, and 7-15, we found that all the 3D-GS rasterizers - i.e. original implementation, gsplat and gsplat along with mip-splatting technique - perform equally well, with a slightly better PSNR observed with gsplat and gsplat using the mip-splatting technique. However, our techniques with the original 3D-GS rasterizer manage to keep the Gaussians closer to the surface while maintaining specular reflections. In contrast, gsplat and gsplat with the mip-splatting technique have several Gaussians positioned away from the surface to maintain specular reflections and view-dependent structural colors, which results in a less clean interactive visualization in the SIBR viewer.

We found that the original implementation of the 3D-GS rasterizer, along with our techniques, successfully optimizes the StructColorTaylorSwift scene with the same capture setup shown in Figure 6-5 under simplified lightnings, and produces better quality renderings.

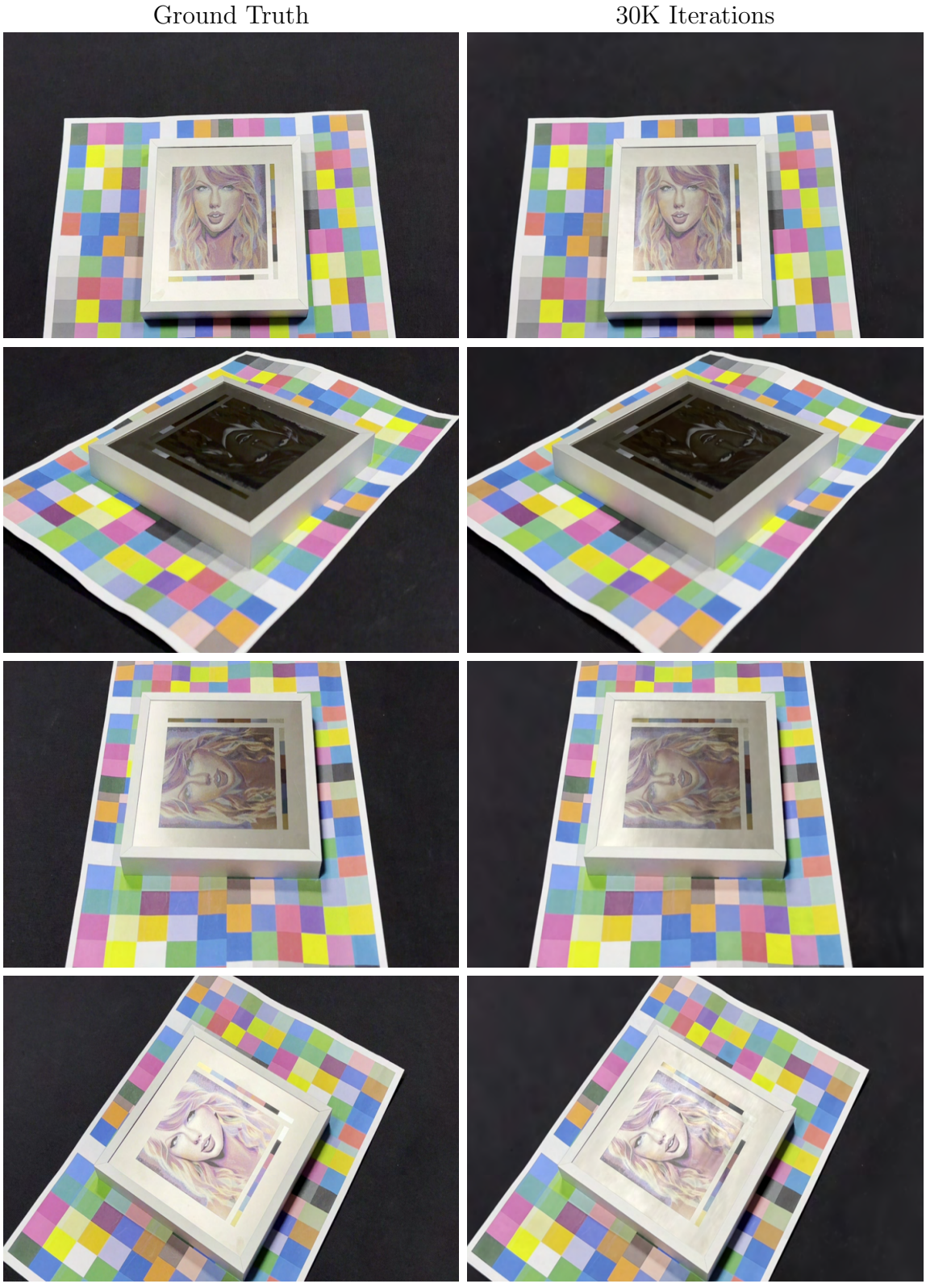


Figure 7-13: Visualization of rendered views of optimized StructColorTaylorSwift scene using 3D-GS rasterizer.

Figure 7-14: Visualization of rendered views of optimized StructColorTaylorSwift scene using gsplat rasterizer.

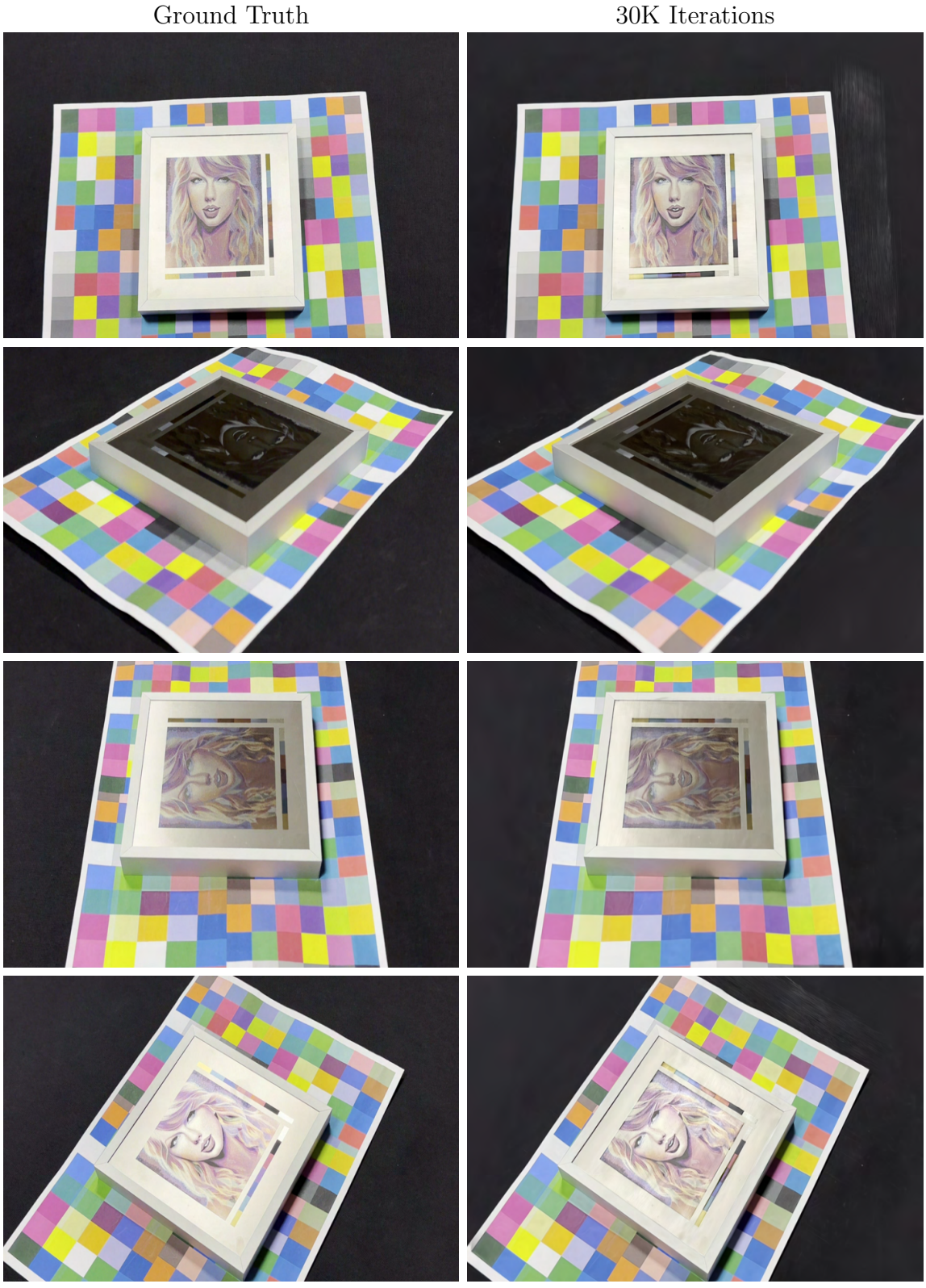


Figure 7-15: Visualization of rendered views of optimized StructColorTaylorSwift scene using gsplat rasterizer along with mip-splatting.

3D-GS + Our Techniques

gsplat

gsplat + mip-splatting

Figure 7-16: Interactive visualization of optimized StructColorTaylorSwift dataset in SIBR viewer.

Chapter 8

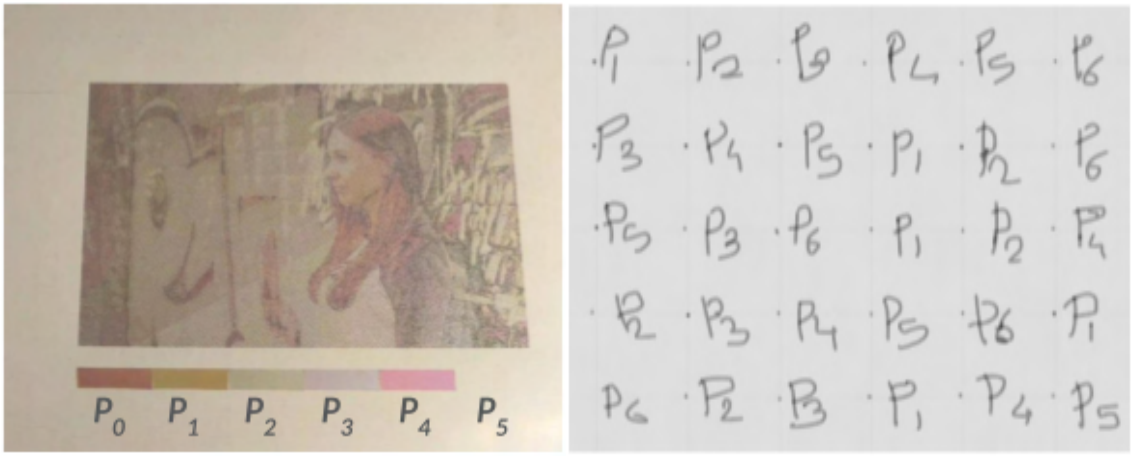
Simulating Structural Color Objects with Just Primaries

In this chapter, we discuss methods that can be used to synthesize arbitrary images of structural color objects for arbitrary view directions using only the patches of primaries laser printed on the metal substrates. These primaries are used in the laser printing of these structural color objects.

Creating a single structural color object using laser printing takes approximately six to seven hours. The goal of the task is to simulate the structural color object using patches of primaries laser printed on metal substrates, as laser printing primary patches takes less time, about an hour. This will enable researchers to visualize a simulated pseudo-structural-color object without actually printing it and find new and vibrant primaries with faster speed to further their research rapidly.

8.1 Primaries and Primary Map

As previously discussed in Section 6.1, we use laser printing technology to create a structural color painting object. We use laser printing technology to generate structural color objects shown in Figures 8-1 and 8-2, using *Primaries* P_0, P_1, P_2, P_3, P_4, and P_5 shown in Figure 8-1a, and *Primary Map* shown in Figure 8-1b. These primaries are structural colors printed on metal substrates at locations defined using a primary map. The primary map is a *Halftone Image* with boolean mask of dimension (height * width * number of primaries), representing at which locations which primaries are used in laser printing process to generate structural color object.



(a) Structural color object laser printed using primaries named P_0, P_1, P_2, P_3, P_4, and P_5.

(b) Structural color object laser printed using *Primary Map* (illustration purpose only).

Figure 8-1: Illustration of structural color object laser printed using *Primaries* and *Primary Map*.

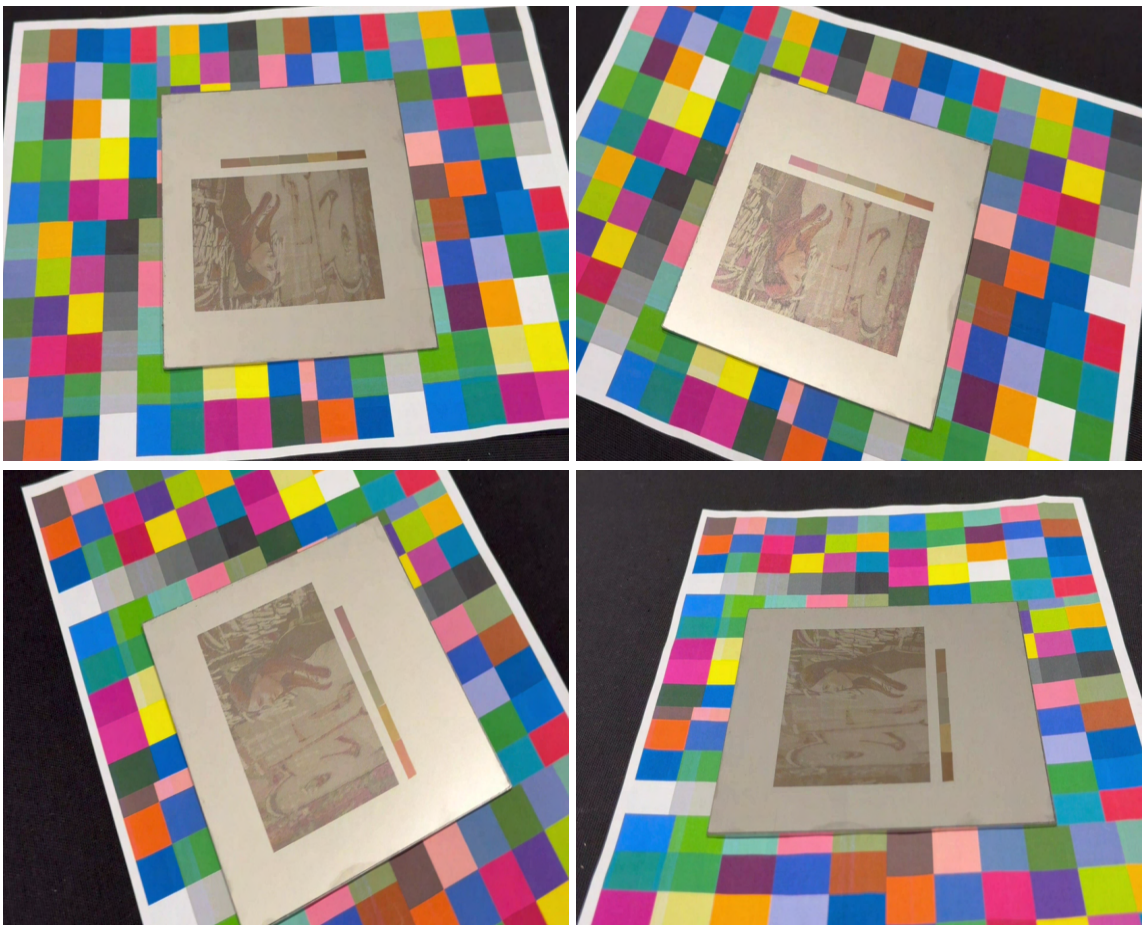


Figure 8-2: Image captures of structural color object laser printed using *Primaries* and *Primary Map*.

Figure 8-2 shows a few sample images captured using the same capture setup used to capture images of real dataset discussed in Section 6.2. We call it StructColorPrimaries scene. Dataset of StructColorPrimaries scene contains scene images and respective camera poses. These images are of the object with structural color painting and six patches of structural color primaries used to create this structural color painting using laser printing. The structural color primaries and structural color painting are on the same metal plate so that we can compare the rendered painting results using just primaries and actual laser printed painting visually and evaluate results. We will use just the primary patches to simulate structural color painting for arbitrary camera poses using the Radiance Field method called 3D-GS [32], gsplat [87], and the Optical Flow method called CoTracker [31]. We will evaluate the visual quality of results from both the methods and discuss our findings.

8.2 Simulation Using 3D-GS

We start by optimizing the radiance field of the StructColorPrimaries scene using 3D-GS. After successful optimization, we get the optimized point cloud shown in Figure 8-3. The Green box contains the radiance field associated with six primaries, and the Orange box contains the radiance field associated with structural color painting.

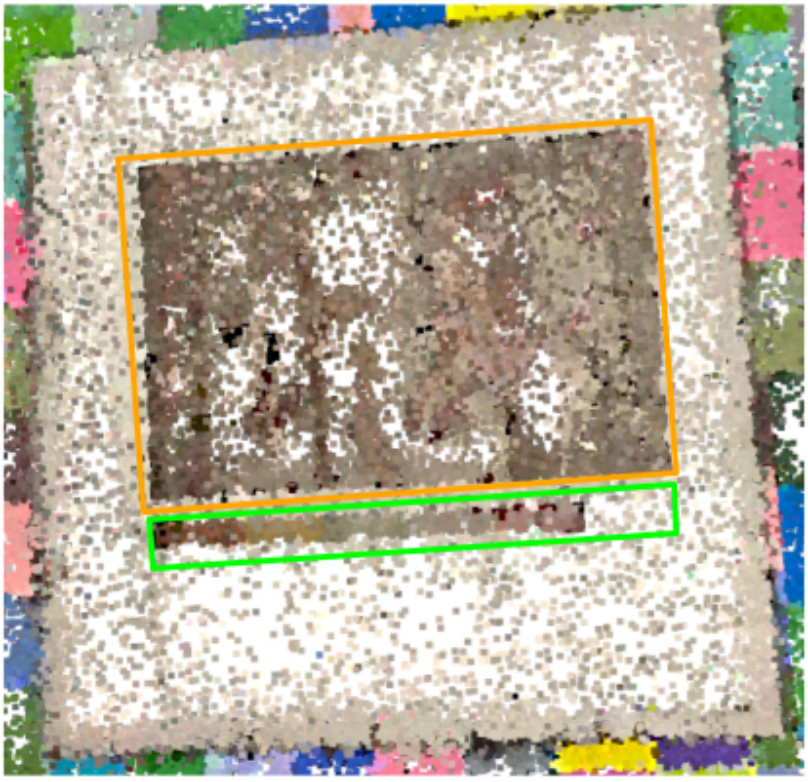


Figure 8-3: Optimized point cloud representing StructColorPrimaries scene. The Green box contains the trained points associated with six primaries, and the Orange box contains the trained points associated with structural color painting.

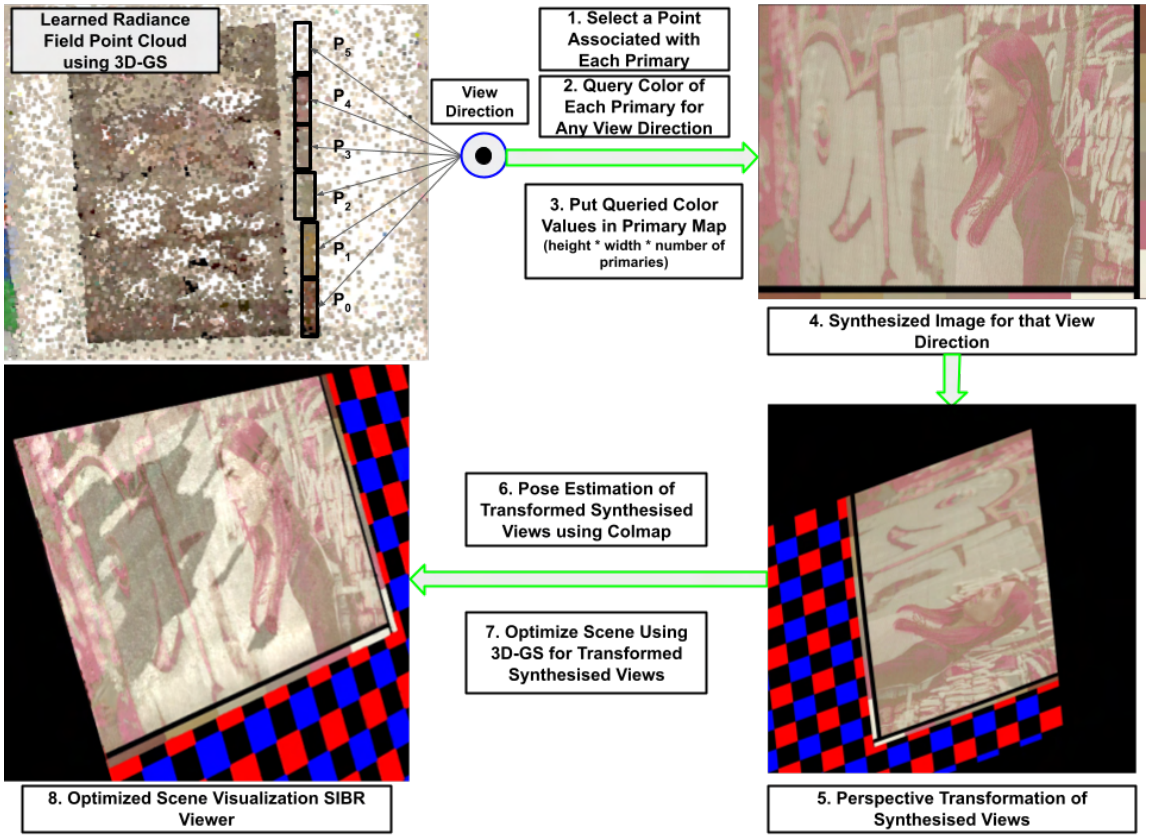


Figure 8-4: Pipeline of Synthesizing Arbitrary Images of Structural Color Object for Arbitrary Viewing Direction: We first learn the Radiance Field of the primaries using 3D-GS. We then query this learned Radiance Fields of primaries according to given primaries-map to render output image of view-dependent structural color object for given view direction. We perform perspective transformation on synthesized views using COLMAP poses from StructColorPrimaries scene. Subsequently, we estimate the poses of these images again using COLMAP and optimize the scene using 3D-GS.

As shown in Figure 8-4, we manually select the Gaussian which are at the front in optimized points approximately representing the radiance field of particular primary. Once we have identified all the Gaussians representing all the primaries used to laser-print structural color objects, we compute the colors from Spherical Harmonics (SHs) associated with each Gaussian for a given viewing direction, as shown in Figure 8-5. These colors associated with primaries are then placed at locations on the image grid defined by our primary map. Figure 8-6 shows some synthesized images of structural color painting using radiance field of primaries. We subsequently perform a perspective transformation of synthesized views according to poses calculated on images from StructColorPrimaries scene. Figure 8-7 shows few perspective projections of synthesized images. However, as we are using the extrinsic and intrinsic parameters of images from StructColorPrimaries scene and therefore perspective projection is approximate and not exactly the same as original images from StructColorPrimaries scene, and also the dimensions of transformed images changes. Therefore, while performing per-

spective transformation we add color checkerboard in the background of synthesized views so that COLMAP can estimate the new poses of transformed images correctly. We then optimize the scene with transformed images and respective COLMAP poses using 3D-GS as shown in the Figure 8-4.

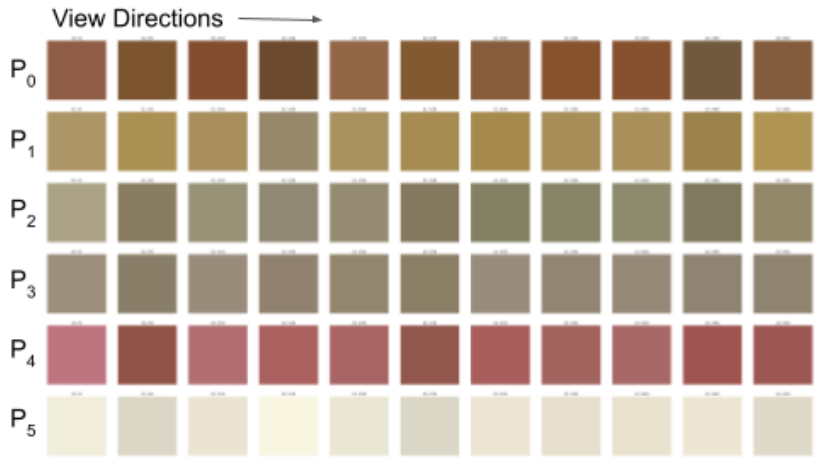


Figure 8-5: Visualization of computed colors of Gaussians representing different primaries with respect to given viewing directions.

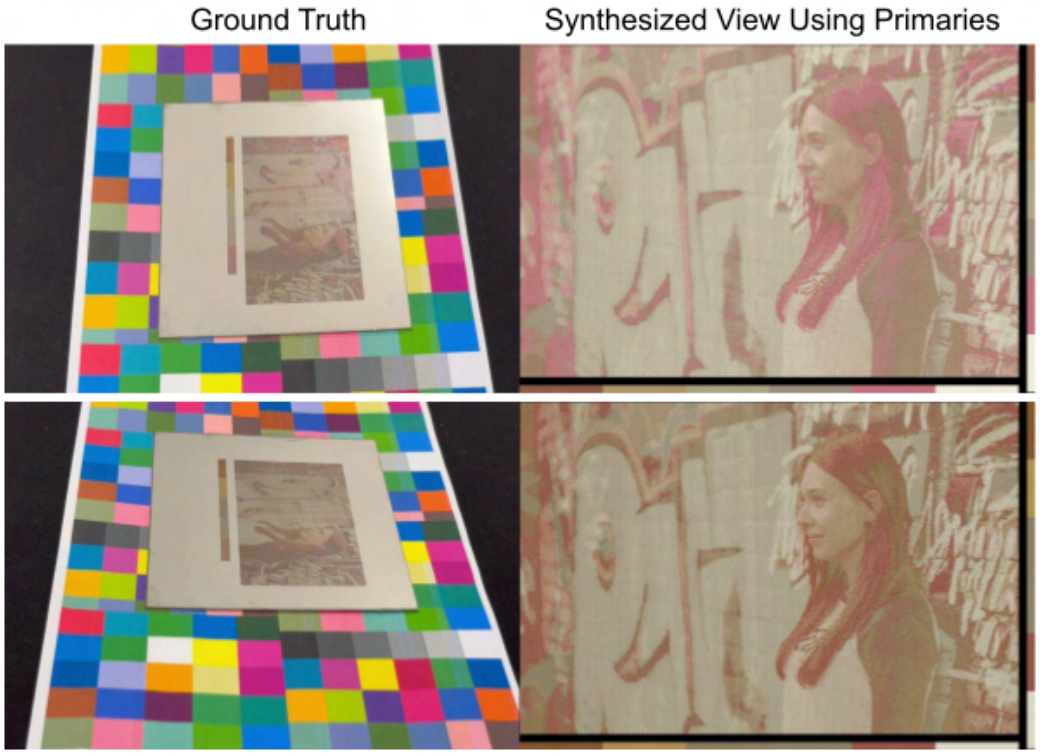


Figure 8-6: Comparison between real structural color object and synthesized views for respective view direction - Visualization shows our method able to achieve the decent renderings similar to ground truths.

Figure 8-7: Visualization of few perspective transformation of original synthesized views with color checkerboard in background.

Figure 8-8: Visualization of StructColorPrimaries scene optimized with 3D-GS in SIBR viewer while navigating scene.

8.3 Simulation Using gsplat

As shown in Figure 8-6, the aforementioned method uses a single Gaussian to represent the radiance field of the primary patch. However, in practice, any pixel in the primary patch that represents the color of the primary is actually the result of α-blending of multiple Gaussians, each contributing to that pixel. Consequently, the resulting color of the pixel does not exactly match the primary for a given camera pose, and thus the synthesized views do not perfectly align with the ground truths.

To address this, we utilize the depth maps generated by gsplat to estimate the optical flow of pixels representing the colors of the primaries, as shown in Figure 8-9. We then use the tracked positions of these pixels, determined by the estimated optical flow, to obtain the colors of the primaries from different viewing directions, as depicted in pipeline in Figure 8-10.

Figure 8-12 clearly demonstrate that the ground truth and synthesized views using gsplat with just primaries are remarkably similar.

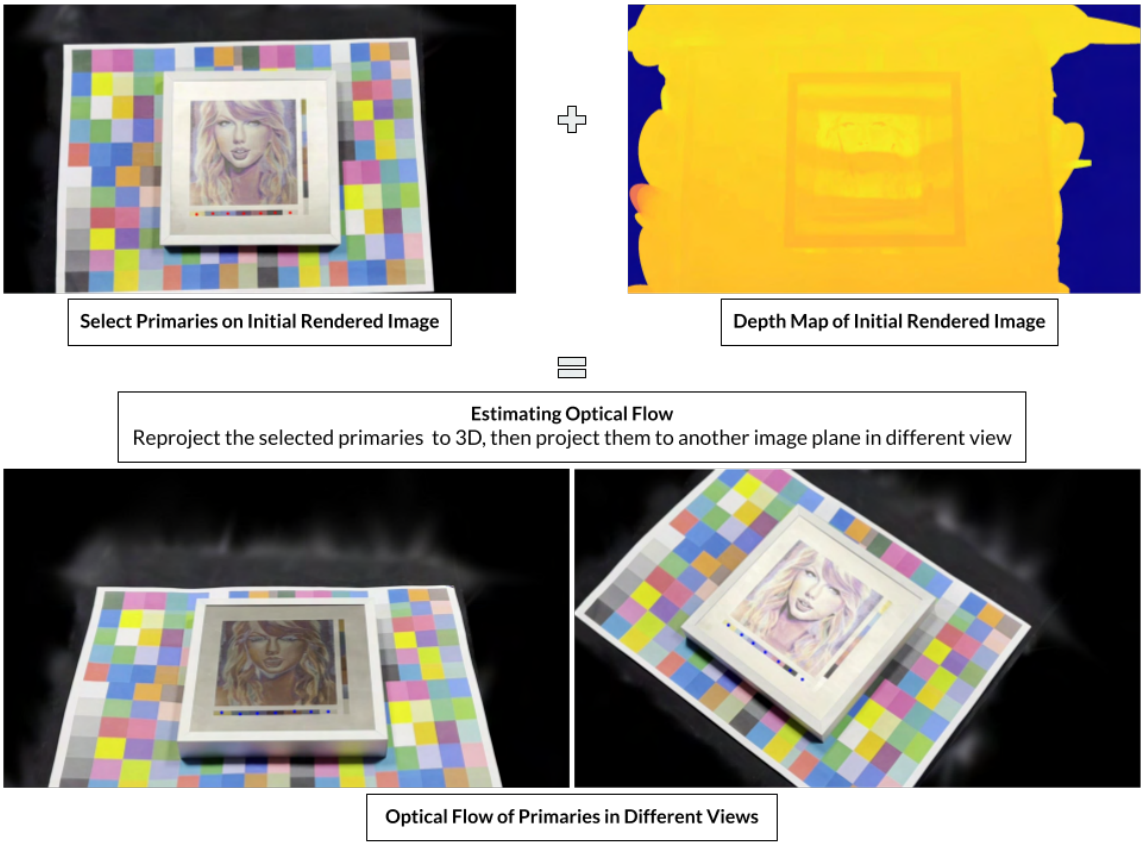


Figure 8-9: Estimating Optical Flow of Primaries in Different Views.

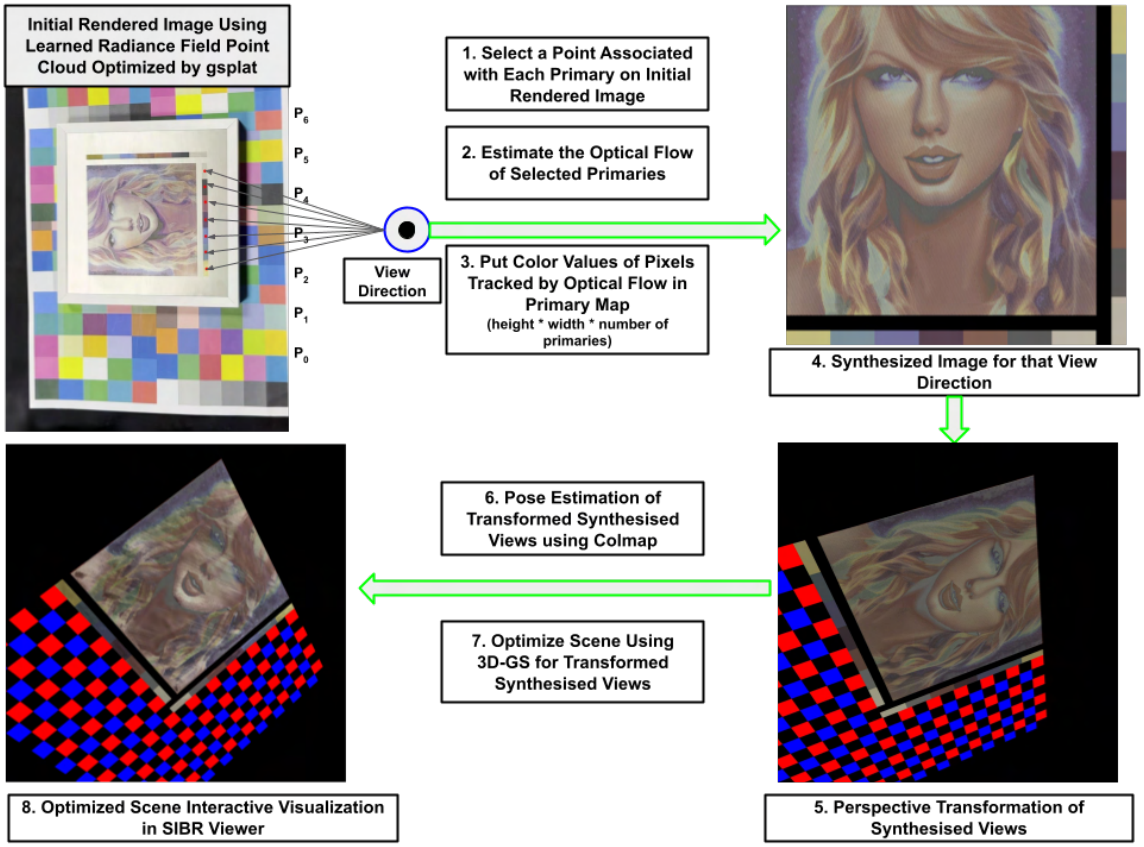


Figure 8-10: Pipeline of Synthesizing Arbitrary Images of Structural Color Object for Arbitrary Viewing Directions Using gsplat: We begin by using gsplat to learn the radiance field of the primaries. Next, we estimate the optical flow and track the primaries, using the depth map of the view obtained from optimized scene for each viewing direction. We then follow the same steps as in the previous pipeline.

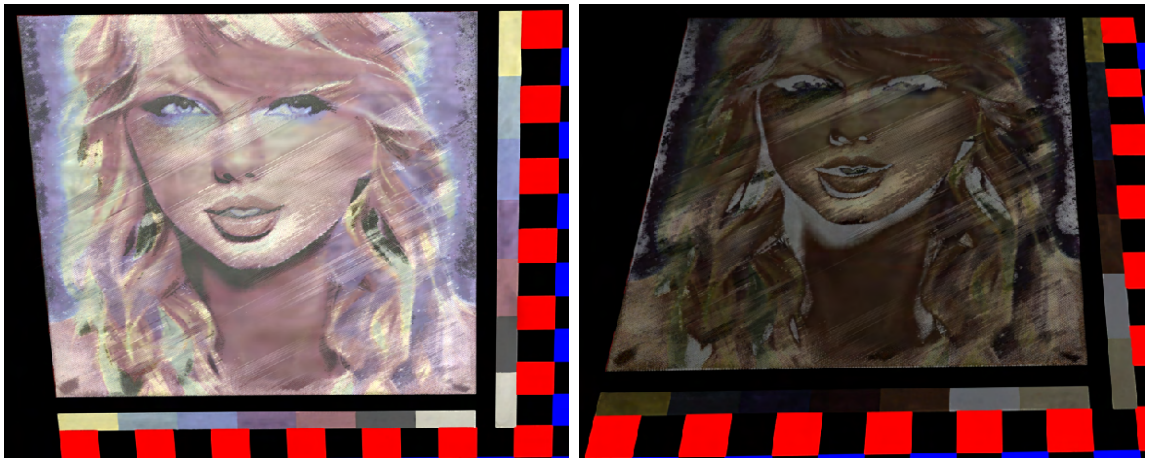


Figure 8-11: Interactive visualization in SIBR viewer of a scene optimized with views synthesized from just primaries.

Figure 8-12: Comparison between real structural color object and synthesized views using gsplat for respective view direction.

8.4 Simulation Using CoTracker

CoTracker [31] is a transformer-based *Optical Flow* model [6] that tracks dense points in a frame jointly across a video sequence. As shown in the Figure 8-13, CoTracker differs from most existing state-of-the-art approaches such as RAFT [72] that track points independently, ignoring their correlation. CoTracker compute convolutional features $\phi(I_t)$ for every frame and process them with sliding windows. In order to initialize track features Q, CoTracker bilinearly sample from $\phi(I_t)$ with starting point locations P. Locations P also serve to initialize estimated tracks $\hat{P}$. As CoTracker tracks more points jointly which significantly improves tracking quality.

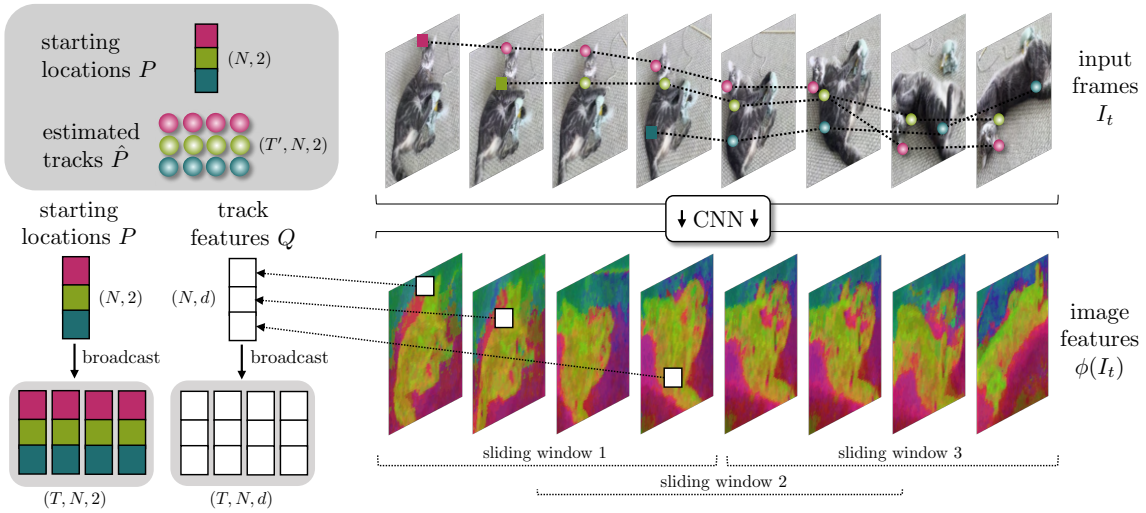


Figure 8-13: Visualization of CoTracker architecture which tracks dense points in a frame jointly across a video sequence. Image from [31]

We begin by manually selecting six points on six primary patches, each representing one of the six primaries, on the first image from the StructColorPrimaries scene. We then track these points on subsequent images from the StructColorPrimaries scene using CoTracker [31], as shown in Figure 8-14. The output is tracked color values of primaries across the image sequence for respective view directions. We then place these colors, associated with primaries, at locations in the image grid given by our primary map, similar to what we do in simulation using 3D-GS. Figure 8-15 shows some synthesized images of structural color painting using tracked colors of primaries for respective viewing directions using CoTracker. Figure 8-16 shows colors associated with each Primary for a given viewing direction.

We can visually analyze from Figures 8-15 (StructColorPrimaries Scene), 8-16 (StructColorPrimaries Scene), and 8-17 (StructColorTaylorSwift) that ground truth and synthesized views are approximately similar.

Figure 8-14: Visualization of tracked selected primary points in a frame jointly across an image sequence.

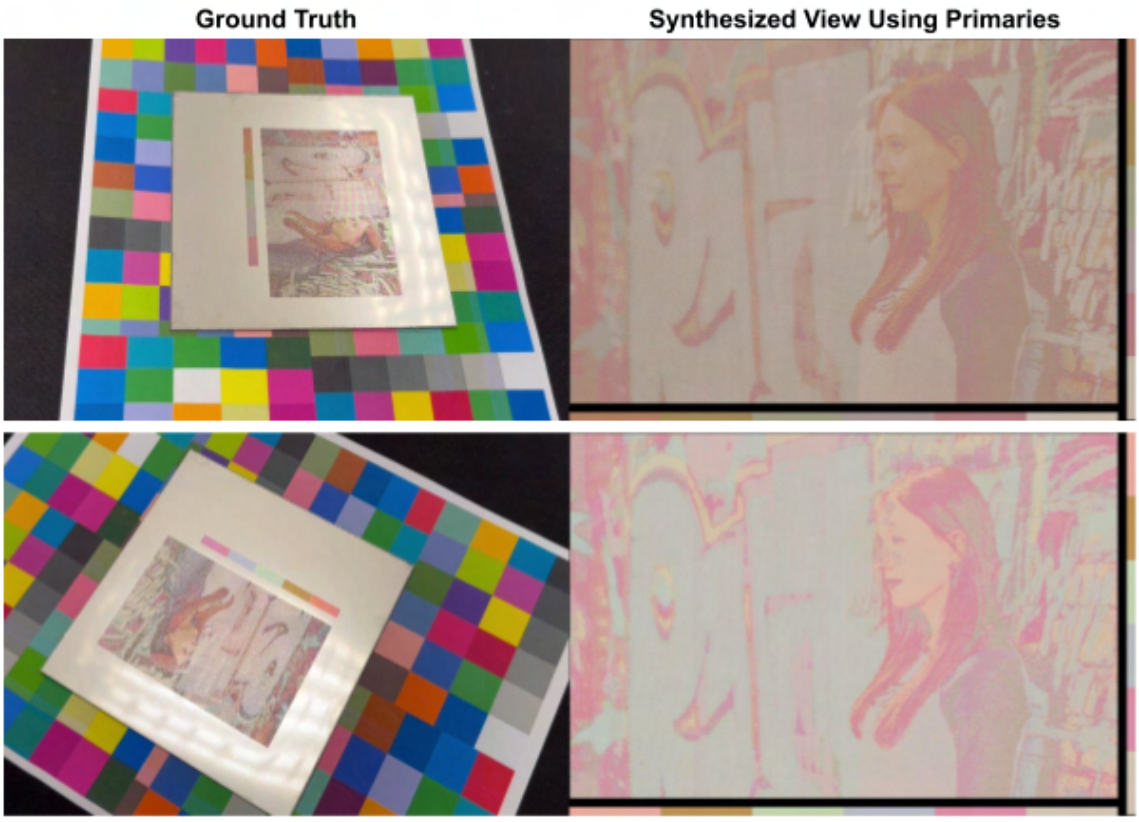


Figure 8-15: Comparison between real structural color object and synthesized views for respective view direction - Visualization shows CoTracker method able to achieve the decent renderings similar to ground truths.

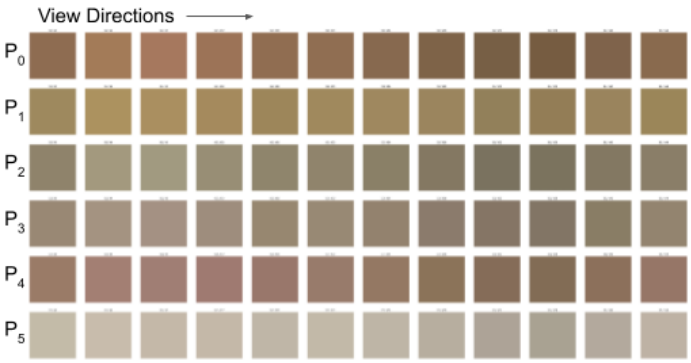


Figure 8-16: Visualization of tracked colors of different primaries with respect to given viewing directions using CoTracker [31].

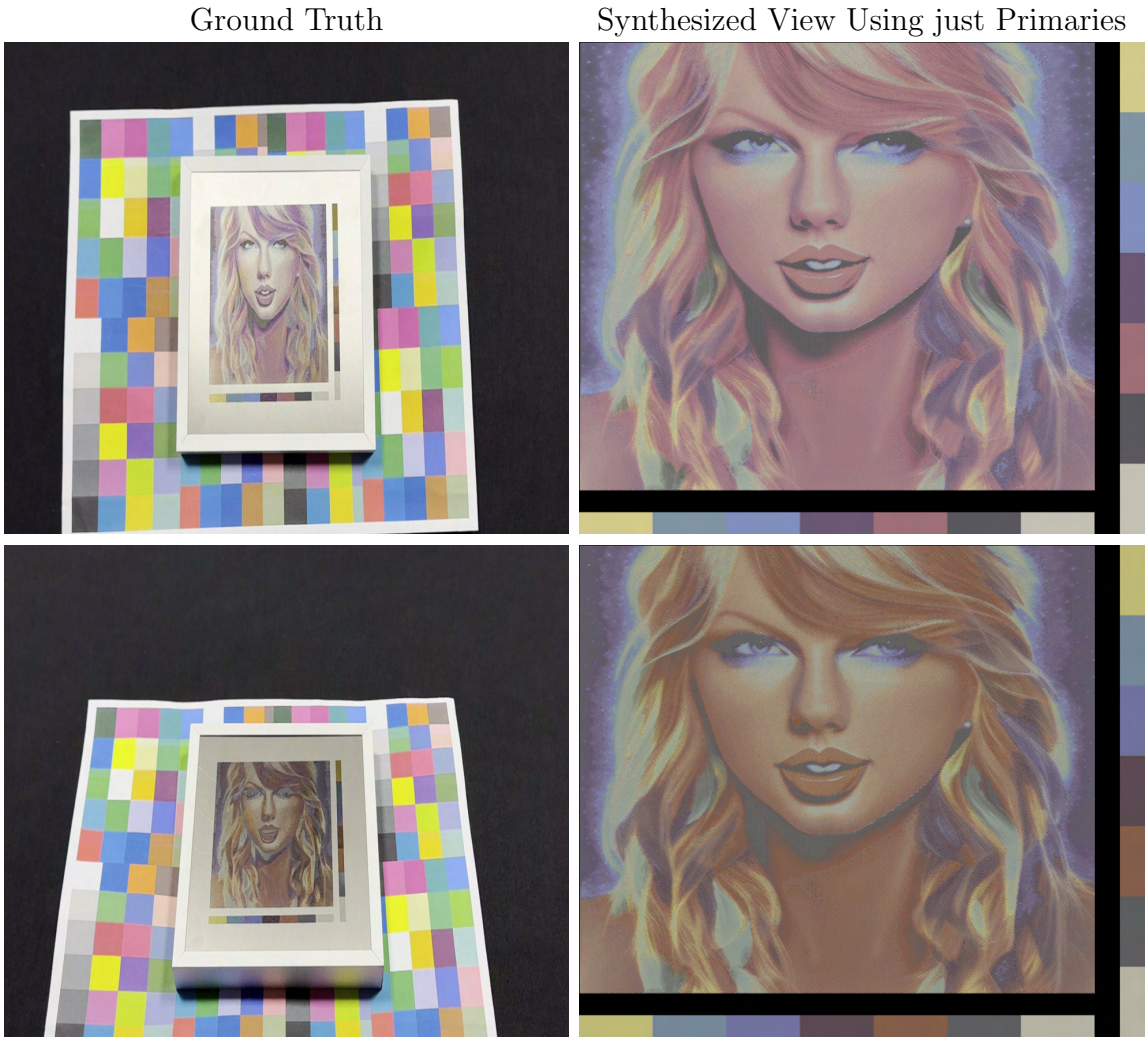


Figure 8-17: Comparison between real structural color object and synthesized views using CoTracker for respective view direction.

8.5 Findings

We found that synthesized views using gsplat-based optical flow method [87] and the CoTracker method [31] are visually more similar to the ground truth than the views synthesized by 3D-GS [32], which queries the color of only a single Gaussian at the front representing each primary. However, CoTracker occasionally fails to track the selected primary points through a sequence of images. Therefore, we utilize the gsplat-based optical flow method for the subsequent steps in the pipeline.

There are still some key points to consider with this pipeline:

- We assume the radiance field or color over the primary patch image remains the same.
- The viewing direction from the camera center to the tracked positions of the respective primaries in the rendered views differs from their actual locations in the original image or primary map, leading to approximations in the rendered colors at those positions in the primary map. Consequently, we refer to these simulated structural color objects as pseudo-structural-color objects.
- For some viewing angles, the primaries are not always exposed to bright light, while only a part of the structural color painting is exposed to the bright light, resulting in different radiance at the primary patch compared to the part of the painting that is less exposed, as shown in Figure 8-18. For these views, it is not reasonable to compare synthesized views to the ground truths.
- We use a halftone map as our primary map. The halftone image acts as a map of locations that indicate the coordinates on which primaries will be deposited via laser printing. We found that the halftone map fails to mimic the rendering like continuous tone images. Additionally, there are no poses for halftone map available.
- Perspective transformation of synthesized views and again estimating the poses of transformed views introduce another level of pseudoness in the simulation.
- We cannot really compare the synthesized and ground truth images using similarity metrics due to presence of background in the original images of StructColorPrimaries and StructColorTaylorSwift datasets.
- As discussed in Section 8.2 and 8.4, we also believe that laser-marked primaries on the metal plate, as per given primary map, interact with each other due to the shiny and highly reflective nature of metal plates and create a perspective illusion for viewers caused by the thin-film interference effect and enhances the visual experience of observers. While digital simulations can approximate this effect by placing the colors of the primaries tracked from different viewing directions according to the primary map, but it is still challenging to fully replicate the dynamic and vivid appearance seen on the actual metal plate.

Nevertheless, as shown in the Figures 8-12 and 8-11, we successfully attain satisfactory results for the task of synthesizing views of structural color objects using just primary patches and interactively visualizing the object in the SIBR viewer.

Furthermore, we can synthesize any structural color painting for different viewing directions without the need to laser print it on metal substrates by using the already tracked primary colors with the gsplat-based optical flow method and primary map, as illustrated in Figure 8-19.

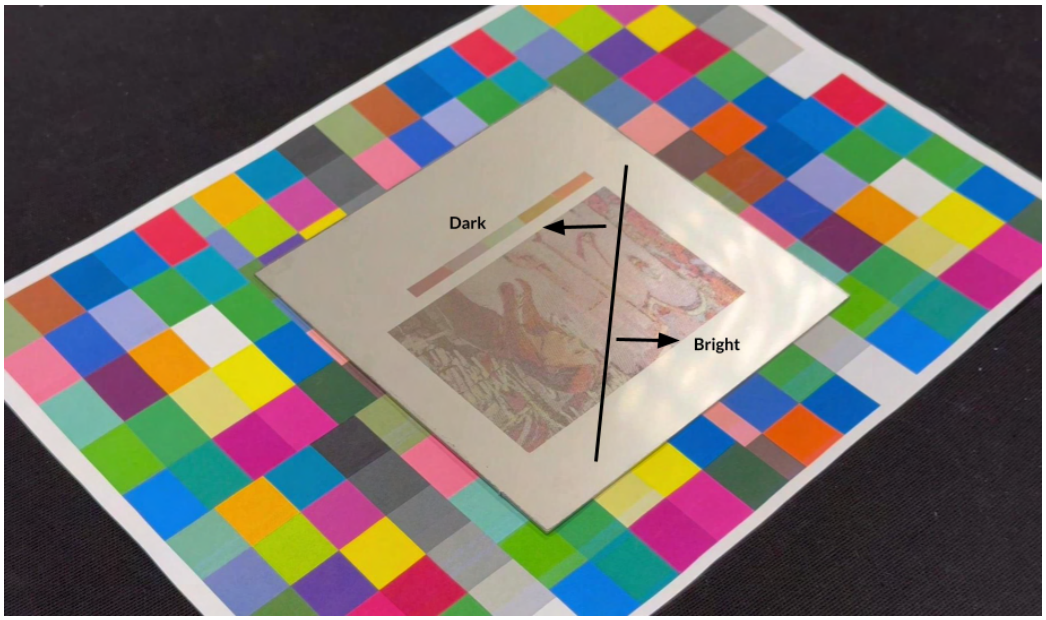


Figure 8-18: Varied radiance when only some part of the structural color object exposed to the bright light for some viewing angles.

Figure 8-19: Synthesizing *Cat* views using the already tracked primary colors with the gsplat-based optical flow method and primary map for different viewing directions.

Chapter 9

Conclusion and Future Work

We introduce novel view synthesis of structural color objects using radiance field methods, and transform these class of objects into their digital twins, enabling immersive, realistic, and interactive visualization for researchers and users.

9.1 Conclusion

In this work, we explore two different but related research directions: novel view synthesis of structural color objects and simulation of structural color objects with just primaries.

We first introduce the challenges that arise when we perform novel view synthesis of scenes containing structural color objects captured with video. The aim of the the task is to perform photo-realistic multiview reconstruction of structural color objects. Current Radiance Field methods have recently revolutionized novel view synthesis of scenes captured with multiple photos or videos, mostly as a result of advances in machine learning. These radiance methods work well for scenes with diffuse surfaces, but they struggle to represent the scene with shiny objects with reflective surfaces. Multiview reconstruction of reflective objects with structural colors is extremely challenging because specular reflections are view-dependent and thus violate the multiview consistency, which is the cornerstone for most multiview reconstruction methods. However, previous works do not address materials exhibiting structural color properties. Thus, there is lack of benchmarking of state-of-the-art (SOTA) Radiance Field methods for scene involving objects with structural colors.

Therefore, we introduce synthetic dataset called StructColorToaster scene, which consist of toaster object that exhibit complicated geometry and realistic non-Lambertian materials, specifically material mimicking structural colors or related phenomenon like Pearlescence and Iridescence effects. Then, we present a qualitative and quantitative evaluation of a SOTA novel view synthesis methods such as NeRF [48], Instant-NGP [51], Mip-NeRF [3], NeRFacto [70], Ref-NeRF [76], and 3D-GS [32] on this StructCol-

orToaster scene. The reasons for choosing these SOTA radiance field methods include available related existing work to solve the problem at hand, what they can do, and our need to grasp how NeRF methods work overall.

We show that among all the radiance field methods selected for experiments on StructColorToaster scene, 3D-GS outweighs all other methods considered in our experiments, both qualitatively and quantitatively. Therefore, we opt for 3D-GS for our subsequent research using our real-world scene dataset of laser printed structural color objects, because it offers the advantages of both implicit and explicit representation-based methods.

We then introduce a capture setup to generate the real-world scene dataset containing StructColorPainting and StructColorTaylorSwift scenes using laser printed structural color objects. We perform experiments by modifying various settings of 3D-GS, such as initialization with SfM point cloud, hyperparameters such as scene background color (black, white, and random), spherical harmonics degree, number of training iterations, iteration where densification stops, how frequently to reset opacity, scaling learning rate, and regularizing the loss function for a better distribution of points over a surface. We show that our techniques of initialization with Cleaned-SfM points and Anisotropy-Regularizer outperforms the vanilla 3D-GS.

In other line of work, we delve into a relatively unexplored research direction while trying to answer the question: *Is it possible to simulate the laser printed structural color objects solely using the patches of primaries laser printed on the metal substrates?* These primaries are employed in laser printing of these structural color objects. We demonstrate that our methods can efficiently and strongly simulate structural color objects for arbitrary camera poses using the Optical Flow methods based on 3D-GS (gsplat) and CoTracker.

9.2 Future Work

Our work marks the beginning of novel view synthesis for structural color objects and opens up potential avenues for various future endeavors. The following are some possible future research directions:

- **Capture Setup and Lighting**. Our capture setup is very simple and minimalistic. We use a handheld smartphone camera along with ceiling-mounted lights directed towards the ground for illumination. However, for improved image quality of structural color objects, which are highly reflective and glossy, a more advanced camera rig and lighting setup are necessary. This would mitigate noise and shadows, resulting in superior image quality and consequently, improved reconstruction quality.

- **Popping and Blending Artifacts**. Even with Anisotropy-Regularizer, 3D-GS introduces popping and blending artifacts in regions with high view dependent appearance. Possible reasons for these artifacts and aliasing effects are the trivial rejection of Gaussians via a guard band in the rasterizer, which can cause optimization to produce large Gaussians and 3D-GS's simple visibility algorithm, which can cause Gaussians to suddenly switch depth or blending order. Possible solutions to reduce these artifacts are: employing a more structured culling strategy for accurate per-pixel depth computation, implementing antialiasing techniques, and more sophisticated regularization for a homogeneous distribution of Gaussians over the object's surface.

- **Estimation of the Material Properties, Illumination, and Geometry**. To enhance the neural rendering in scenes with reflective surfaces featuring structural colors, we can apply shading function on 3D Gaussians to estimate additional shading attributes such as Diffuse, Tint, Roughness Normal, Residual, and Differentiable Environment Lighting Map. Additionally, we can also extend 3D-GS by employing learning based Inverse Rendering to estimate the material properties, illumination, and geometry of the structural color object from multi-view images to relight the scene and observe how the structural color object behaves under different lighting conditions.

- **Web Viewer**. Currently, there are many web viewers available such as gsplat.js and PlayCanvas Viewer, but the majority of them lack support for Spherical Harmonics (SHs) coefficients beyond degree 0 and also suffer from slower rendering speeds. There is possibility of developing an in-house web viewer that offers enhanced features in this regard. There is also room for improving SIBR viewer to address the problem of aliasing and holes during interactive visualization of structural color objects.

Bibliography

[1] Michal Aibin. Focal length and intrinsic camera parameters. https://www.baeldung.com/cs/focal-length-intrinsic-camera-parameters, March 2024. (cited on page 32, 33)

[2] Kara-Ali Aliev, Artem Sevastopolsky, Maria Kolos, Dmitry Ulyanov, and Victor Lempitsky. Neural point-based graphics, 2020. (cited on page 56)

[3] Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. *ICCV*, 2021. (cited on page 19, 43, 49, 53, 55, 78, 95, 137)

[4] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. *CVPR*, 2022. (cited on page 19, 49, 50, 53, 55)

[5] Carolina Bento. Multilayer perceptron explained with a real-life example and python code: Sentiment analysis, 2021. (cited on page 23)

[6] M.J. Black and P. Anandan. A framework for the robust estimation of optical flow. In *1993 (4th) International Conference on Computer Vision*, pages 231–236, 1993. (cited on page 131)

[7] Jin-Xiang Chai, Xin Tong, Shing-Chow Chan, and Heung-Yeung Shum. Plenoptic sampling. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, page 307–318, USA, 2000. ACM Press/Addison-Wesley Publishing Co. (cited on page 20)

[8] Guikun Chen and Wenguan Wang. A survey on 3d gaussian splatting, 2024. (cited on page 58)

[9] Christopher B. Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction, 2016. (cited on page 25)

[10] George Degen. Bio-inspired materials: Structural color in nature. https://www.sbnature.org/publications/blog/2/posts/66/bio-inspired-materials-structural-color-in-nature, Oct 2020. (cited on page 73)

[11] Robert A. Drebin, Loren Carpenter, and Pat Hanrahan. Volume rendering. In *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '88, page 65–74, New York, NY, USA, 1988. Association for Computing Machinery. (cited on page 25)

[12] Emilien Dupont, Miguel Angel Bautista, Alex Colburn, Aditya Sankar, Carlos Guestrin, Josh Susskind, and Qi Shan. Equivariant neural rendering, 2020. (cited on page 19)

[13] Fridovich-Keil and Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *CVPR*, 2022. (cited on page 19)

[14] Yasutaka Furukawa and Carlos Hernández. Multi-view stereo: A tutorial, 2015. (cited on page 18, 20)

[15] Yasutaka Furukawa and Jean Ponce. Accurate, dense, and robust multiview stereopsis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(8):1362–1376, 2010. (cited on page 35)

[16] Jiao Geng, Liye Xu, Wei Yan, Liping Shi, and Min Qiu. High-speed laser writing of structural colors for full-color inkless printing. *Nature Communications*, 14, 02 2023. (cited on page 97, 98)

[17] Riccardo Gherardi, Michela Farenzena, and Andrea Fusiello. Improving the efficiency of hierarchical structure-and-motion. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1594–1600, 2010. (cited on page 36)

[18] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative arersarial networks, 2014. (cited on page 19)

[19] Robert Green. Spherical harmonic lighting: The gritty details, 2003. (cited on page 36, 38)

[20] Najlaa Hamza and Ghassan Majeed. A new approach for 3d face modeling using multi-view-stereo and icp algorithms, 06 2020. (cited on page 20)

[21] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994. (cited on page 22)

[22] Hecht-Nielsen. Theory of the backpropagation neural network. In *International 1989 Joint Conference on Neural Networks*, pages 593–605 vol.1, 1989. (cited on page 22)

[23] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. *ACM Trans. Graph.*, 37(6), dec 2018. (cited on page 23)

[24] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks, 2018. (cited on page 23)

[25] Yasushi Ito and Björn Engquist. *Delaunay Triangulation*, pages 332–334. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015. (cited on page 20)

[26] Idonotexist Iwillnotexist. Perspective transform. `https://stackoverflow.com/a/20089412/23239726`, Nov 2020. (cited on page 39)

[27] Farid Javadnejad. Small unmanned aircraft systems (uas) for engineering inspections and geospatial mapping, 01 2018. (cited on page 35)

[28] Arnulf Jentzen, Benno Kuckuck, and Philippe von Wurstemberger. Mathematical introduction to deep learning: Methods, implementations, and theory, 2023. (cited on page 21, 22)

[29] Yan Joe. Gaussian splatting algorithm details. `https://github.com/joeyan/gaussian_splatting/blob/main/MATH.md`, April 2024. (cited on page 61)

[30] Sing Bing Kang, Yin Li, Xin Tong, and Heung-Yeung Shum. Image-based rendering. *Foundations and Trends® in Computer Graphics and Vision*, 2(3):173–258, 2007. (cited on page 18)

[31] Nikita Karaev, Ignacio Rocco, Benjamin Graham, Natalia Neverova, Andrea Vedaldi, and Christian Rupprecht. Cotracker: It is better to track together. *arXiv:2307.07635*, 2023. (cited on page 14, 15, 124, 131, 133, 134)

[32] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), July 2023. (cited on page 14, 15, 16, 19, 26, 43, 55, 56, 60, 70, 71, 78, 93, 95, 105, 124, 134, 137)

[33] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. (cited on page 22, 60)

[34] Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019. (cited on page 19)

[35] Georgios Kopanas, Thomas Leimkühler, Gilles Rainer, Clément Jambon, and George Drettakis. Neural point catacaustics for novel-view synthesis of reflections. *ACM Transactions on Graphics*, 41(6):1–15, November 2022. (cited on page 58)

[36] Samuli Laine and Tero Karras. High-performance software rasterization on gpus. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, HPG '11, page 79–88, New York, NY, USA, 2011. Association for Computing Machinery. (cited on page 56)

[37] Yann LeCun, Y. Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–44, 05 2015. (cited on page 21)

[38] Peng Li, Mao Wang, Jinyu Fu, and Yankun Wang. Plane detection based on an improved ransac algorithm. In *2023 IEEE 3rd International Conference on Computer Communication and Artificial Intelligence (CCAI)*, pages 211–215, 2023. (cited on page 106)

[39] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural volumes: learning dynamic renderable volumes from images. *ACM Transactions on Graphics*, 38(4):1–14, July 2019. (cited on page 19)

[40] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural volumes: Learning dynamic renderable volumes from images. *ACM Trans. Graph.*, 38(4):65:1–65:14, July 2019. (cited on page 25)

[41] David G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ICCV '99, page 1150, USA, 1999. IEEE Computer Society. (cited on page 35)

[42] G. Markus and P. Hanspeter. Point-based graphics. *Point-Based Graphics*, 01 2007. (cited on page 26, 56)

[43] Ricardo Martin-Brualla, Noha Radwan, Mehdi S. M. Sajjadi, Jonathan T. Barron, Alexey Dosovitskiy, and Daniel Duckworth. Nerf in the wild: Neural radiance fields for unconstrained photo collections, 2021. (cited on page 53)

[44] Leonard McMillan and Gary Bishop. Plenoptic modeling: an image-based rendering system. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, page 39–46, New York, NY, USA, 1995. Association for Computing Machinery. (cited on page 18, 20)

[45] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space, 2019. (cited on page 25, 26)

[46] Moustafa Meshry, Dan B Goldman, Sameh Khamis, Hugues Hoppe, Rohit Pandey, Noah Snavely, and Ricardo Martin-Brualla. Neural rerendering in the wild, 2019. (cited on page 23)

[47] Ben Mildenhall, Pratul P. Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: practical view synthesis with prescriptive sampling guidelines. *ACM Trans. Graph.*, 38(4), jul 2019. (cited on page 19)

[48] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020. (cited on page 19, 22, 26, 28, 31, 43, 44, 45, 46, 49, 53, 55, 74, 78, 79, 95, 137)

[49] Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, Peter Hedman, Ricardo Martin-Brualla, and Jonathan T. Barron. MultiNeRF: A Code Release for Mip-NeRF 360, Ref-NeRF, and RawNeRF, 2022. (cited on page 78)

[50] Thomas Müller. tiny-cuda-nn. https://github.com/NVlabs/tiny-cuda-nn, April 2021. (cited on page 47, 52, 53)

[51] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, July 2022. (cited on page 19, 26, 43, 46, 48, 53, 55, 78, 82, 95, 137)

[52] Simon Niklaus, Long Mai, Jimei Yang, and Feng Liu. 3d ken burns effect from a single image. *ACM Trans. Graph.*, 38(6), nov 2019. (cited on page 19)

[53] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation, 2019. (cited on page 25, 26)

[54] Soumik Rakshit. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. https://wandb.ai/geekyrakshit/mip-nerf-360/reports/Mip-NeRF-360-Unbounded-Anti-Aliased-Neural-Radiance-Fields–VmlldzoxOTc4Mjk4, May 2022. (cited on page 51)

[55] Tianhe Ren, Shilong Liu, Ailing Zeng, Jing Lin, Kunchang Li, He Cao, Jiayu Chen, Xinyu Huang, Yukang Chen, Feng Yan, Zhaoyang Zeng, Hao Zhang, Feng Li, Jie Yang, Hongyang Li, Qing Jiang, and Lei Zhang. Grounded sam: Assembling open-world models for diverse visual tasks, 2024. (cited on page 108, 109)

[56] Gernot Riegler and Vladlen Koltun. Free view synthesis, 2020. (cited on page 23)

[57] Annabel Romero. Structural color: When light transforms into art. `https://opuntiavisual.org/opuntiavisual/2016/9/4/structural-color-when-light-transforms-into-art`, Sept 2016. (cited on page 74)

[58] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015. (cited on page 23)

[59] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. (cited on page 35, 36, 97, 99, 102, 105)

[60] Johannes L. Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4104–4113, 2016. (cited on page 20)

[61] Meng-Li Shih, Shih-Yang Su, Johannes Kopf, and Jia-Bin Huang. 3d photography using context-aware layered depth inpainting. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. (cited on page 19)

[62] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhöfer. Deepvoxels: Learning persistent 3d feature embeddings, 2019. (cited on page 25)

[63] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations, 2020. (cited on page 19, 25)

[64] Noah Snavely, Steven M. Seitz, and Richard Szeliski. Modeling the world from internet photo collections. *Int. J. Comput. Vis.*, 80(2):189–210, 2008. (cited on page 36)

[65] Shao-Hua Sun, Minyoung Huh, Yuan-Hong Liao, Ning Zhang, and Joseph J Lim. Multi-view to novel view: Synthesizing novel views with self-learned confidence. In *European Conference on Computer Vision*, 2018. (cited on page 18)

[66] Chris Sweeney, Torsten Sattler, Tobias Höllerer, Matthew Turk, and Marc Pollefeys. Optimizing the viewing graph for structure-from-motion. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 801–809, 2015. (cited on page 36)

[67] Andrea Tagliasacchi and Ben Mildenhall. Volume rendering digest (for nerf), 2022. (cited on page 28)

[68] Matt Tancik, Ben Mildenhall, Pratul Srinivasan, Jon Barron, and Angjoo Kanazawa. Neural volumetric rendering for computer vision. `https://sites.google.com/berkeley.edu/nerf-tutorial/home`, 2022. (cited on page 28, 30, 31, 34, 44)

[69] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains, 2020. (cited on page 26, 45)

[70] Matthew Tancik, Ethan Weber, Evonne Ng, Ruilong Li, Brent Yi, Justin Kerr, Terrance Wang, Alexander Kristoffersen, Jake Austin, Kamyar Salahi, Abhik Ahuja, David McAllister, and Angjoo Kanazawa. Nerfstudio: A modular framework for neural radiance field development. In *ACM SIGGRAPH 2023 Conference Proceedings*, SIGGRAPH '23, 2023. (cited on page 43, 51, 52, 53, 55, 60, 78, 88, 95, 137)

[71] Mohammad Mustafa Taye. Theoretical understanding of convolutional neural network: Concepts, architectures, applications, future directions. *Computation*, 11(3), 2023. (cited on page 19)

[72] Zachary Teed and Jia Deng. Raft: Recurrent all-pairs field transforms for optical flow, 2020. (cited on page 131)

[73] Justus Thies, Michael Zollhöfer, Christian Theobalt, Marc Stamminger, and Matthias Nießner. Ignor: Image-guided neural object rendering, 2020. (cited on page 23)

[74] Bill Triggs, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. Bundle adjustment — a modern synthesis. In Bill Triggs, Andrew Zisserman, and Richard Szeliski, editors, *Vision Algorithms: Theory and Practice*, pages 298–372, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. (cited on page 35)

[75] Richard Tucker and Noah Snavely. Single-view view synthesis with multiplane images. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. (cited on page 19)

[76] Dor Verbin, Peter Hedman, Ben Mildenhall, Todd Zickler, Jonathan T. Barron, and Pratul P. Srinivasan. Ref-NeRF: Structured view-dependent appearance for neural radiance fields. *CVPR*, 2022. (cited on page 43, 53, 54, 55, 74, 76, 78, 91, 95, 137)

[77] Qianqian Wang, Zhicheng Wang, Kyle Genova, Pratul Srinivasan, Howard Zhou, Jonathan T. Barron, Ricardo Martin-Brualla, Noah Snavely, and Thomas Funkhouser. Ibrnet: Learning multi-view image-based rendering, 2021. (cited on page 23)

[78] Wikipedia. Spherical coordinate system — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Spherical%20coordinate%20system&oldid=1209923161`, 2024. [Online; accessed 18-March-2024]. (cited on page 37)

[79] Wikipedia. Structural coloration — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Structural%20coloration&oldid=1208206459`, 2024. [Online; accessed 06-April-2024]. (cited on page 73)

[80] Olivia Wiles, Georgia Gkioxari, Richard Szeliski, and Justin Johnson. Synsin: End-to-end view synthesis from a single image, 2020. (cited on page 19)

[81] Daniel E. Worrall, Stephan J. Garbin, Daniyar Turmukhambetov, and Gabriel J. Brostow. Interpretable transformations with encoder-decoder networks, 2017. (cited on page 19)

[82] Weihao Xia and Jing-Hao Xue. A survey on deep generative 3d-aware image synthesis. In *ACM Computing Surveys (CSUR)*, 2023. (cited on page 18)

[83] Tianyi Xie, Zeshun Zong, Yuxing Qiu, Xuan Li, Yutao Feng, Yin Yang, and Chenfanfu Jiang. Physgaussian: Physics-integrated 3d gaussians for generative dynamics, 2024. (cited on page 112)

[84] Xoft. Spherical harmonics. `https://xoft.tistory.com/50`, Oct 2023. (cited on page 38)

[85] Qiangeng Xu, Zexiang Xu, Julien Philip, Sai Bi, Zhixin Shu, Kalyan Sunkavalli, and Ulrich Neumann. Point-nerf: Point-based neural radiance fields, 2023. (cited on page 26, 56)

[86] Zexiang Xu, Sai Bi, Kalyan Sunkavalli, Sunil Hadap, Hao Su, and Ravi Ramamoorthi. Deep view synthesis from sparse photometric images. *ACM Trans. Graph.*, 38(4), jul 2019. (cited on page 24)

[87] Vickie Ye and Angjoo Kanazawa. Mathematical supplement for the `gsplat` library, 2023. (cited on page 70, 124, 134)

[88] Zongxin Ye, Wenyu Li, Sidun Liu, Peng Qiao, and Yong Dou. Absgs: Recovering fine details for 3d gaussian splatting, 2024. (cited on page 70)

[89] Wang Yifan, Felice Serena, Shihao Wu, Cengiz Öztireli, and Olga Sorkine-Hornung. Differentiable surface splatting for point-based geometry processing. *ACM Transactions on Graphics*, 38(6):1–14, November 2019. (cited on page 26, 56)

[90] Alex Yu, Sara Fridovich-Keil, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks, 2021. (cited on page 26)

[91] Zehao Yu, Anpei Chen, Binbin Huang, Torsten Sattler, and Andreas Geiger. Mip-splatting: Alias-free 3d gaussian splatting. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024. (cited on page 70, 71)

[92] Kate Yurkova. A comprehensive overview of gaussian splatting. `https://towardsdatascience.com/a-comprehensive-overview-of-gaussian-splatting-e7d570081362#6c97`, Dec 2023. (cited on page 56, 57, 61)

[93] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric, 2018. (cited on page 27)

[94] Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. Stereo magnification: Learning view synthesis using multiplane images. In *SIGGRAPH*, 2018. (cited on page 24)

[95] S. Zinger, L. Do, and P. H. N. de With. Free-viewpoint depth image based rendering. *J. Vis. Comun. Image Represent.*, 21(5–6):533–541, jul 2010. (cited on page 19)

[96] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. Ewa volume splatting. In *Proceedings Visualization, 2001. VIS '01.*, pages 29–538, 2001. (cited on page 58, 59, 64)