Efficient Mesh Reconstruction based on 3D Gaussian Splatting with L1 Regularization and Pruning

Abstract

We propose a method to accelerate the extraction of meshes from 3D Gaussian Splatting. After Neural Radiance Fields demonstrated amazing results, 3D Gaussian Splatting has achieved significantly faster speeds and high-quality rendering of novel views. However, extracting a high-quality mesh from millions of 3D Gaussians remains challenging. Surface-Aligned Gaussian Splatting (SuGaR) introduces regularization to distribute Gaussians on object surfaces and extract surface meshes, but it faces issues of slow extraction and messy result. Our contribution includes introducing an L1 regularization term and a pruning strategy for opacity during 3D Gaussian Splatting training, reducing unnecessary Gaussians and obtaining a higher quality reconstruction initialization. We also introduce L1 regularization in SuGaR to encourage Gaussians to adhere closely to surfaces while retaining detail, and apply Poisson Reconstruction for mesh extraction. Additionally, we provide an optional refinement strategy that binds Gaussians to the surface mesh, further optimizing model details. Our method retrieves high-quality, editable rendered meshes within minutes, surpassing the original in quality while reducing training time. Our code can be accessed through https://github.com/Boreas-OuO/SuGaRL1.

Keywords: 3D Gaussian Splatting (3DGS), L1 Regularization, Poisson Reconstruction, Alpha Compositing

基于 L1 正则化和剪枝的高效 3D 高斯网格重建与渲染

摘要

我们提出了一种加速从 3D 高斯喷溅中提取网格的方法。建立在神经辐射场的进展基础上，3D 高斯喷溅实现了显着更快的速度和高质量的新视图渲染。然而，从数百万个微小的 3D 高斯中提取高质量的网格仍然具有挑战性。表面对齐的高斯喷溅（SuGaR）引入了正则化来将高斯分布在对象表面上并提取表面网格，但它面临着提取速度慢和网格杂乱的问题。我们的贡献包括在 3D 高斯喷溅训练中引入 L1 正则化项和一个透明度剪枝策略，减少不必要的高斯并获得更高质量的重建初始化。我们还在 SuGaR 中引入 L1 正则化，鼓励高斯密切粘附在表面上同时保留细节，并应用泊松重建进行网格提取。此外，我们提供了一个可选的细化策略，将高斯绑定到表面网格，进一步优化模型细节。我们的方法在数十分钟内提取高质量、可编辑的渲染网格，超越原始质量同时减少训练时间。本文所涉及的代码可以通过以下链接获取 https://github.com/Boreas-OuO/SuGaRL1。

关键词： 3D 高斯喷溅 (3DGS), L1 正则化, 泊松重建, Alpha 混合

Contents

1 Introduction

Meshes and points are widely used to represent 3D scenes due to their explicit nature and suitability for rapid GPU/CUDA-based rasterization [16]. Recent advancements in Neural Radiance Field (NeRF) [1] methods diverge from traditional approaches by employing continuous scene representations. These methods typically optimize a Multi-Layer Perceptron (MLP) through volumetric ray-marching to synthesize novel views of captured scenes. Following the emergence of NeRF [1], 3D Gaussian Splatting (3DGS) [16] has gained popularity for its ability to capture and render 3D scenes from novel viewpoints. This technique optimizes the positions, orientations, appearances (represented as spherical harmonics), and alpha blending of numerous small 3D Gaussians based on a training set of images. This process allows for the capture of both the geometry and appearance of the scene [16]. Due to the efficiency of rendering Gaussians compared to rendering a neural field, 3D Gaussian Splatting offers a significant speed advantage over NeRF. This efficiency allows 3D Gaussian Splatting to capture a scene in just a few minutes on a single GPU.

While 3D Gaussian Splatting enables highly realistic renderings of scenes, extracting the surface of the scene from the resulting Gaussians remains a significant challenge [2]. After optimization by 3DGS, the Gaussians do not form an ordered structure and often do not accurately correspond to the actual surface of the scene [2]. In many pipelines, it is desirable to represent the scene as a mesh for its benefits in editing, sculpting, animating, and relighting. However, due to the unstructured nature of the Gaussians Splatting, extracting a mesh from them is extremely challenging.

Another issue with 3DGS is its densification process, which aims to capture scene details with high fidelity [2]. This process results in a drastic increase in the number of Gaussians used to represent the scene, often reaching millions with varying scales and rotations. The majority of these Gaussians are extremely small to reproduce fine texture and details in the scene. This leads to a density function that is close to zero almost everywhere making it difficult for algorithms like Marching Cubes [3] to extract proper level sets from such a sparse density function, as depicted in Figure 1.

While extracting the surface remains a significant challenge, recent work [2] has shed light on this issue by proposing a method to extract Gaussians that closely adhere to the surface of objects. This work introduces regularization terms to penalize the deviation

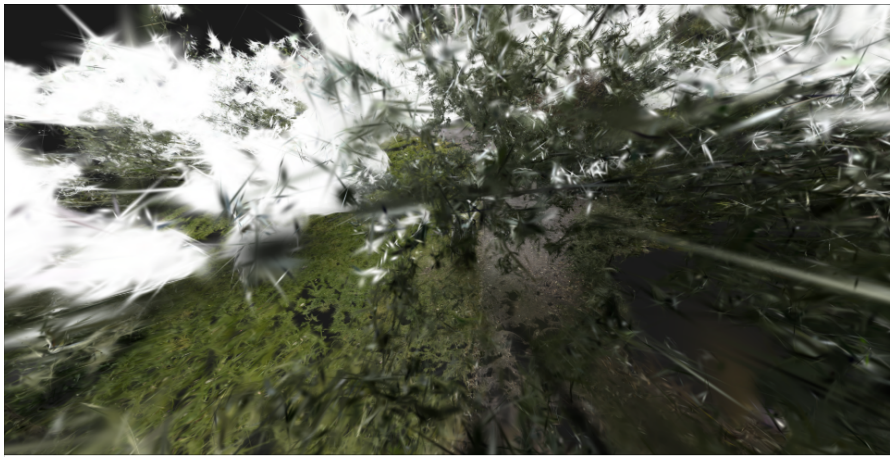

Figure 1: 3DGS optimizes a large number of low-opacity Gaussians inside objects to represent surface details, which makes it difficult to effectively reconstruct grids from these messy Gaussians.

between the Gaussians and the surface of the object, aiming to make them as close as possible. However, despite efforts to penalize the Signed Distance Function (SDF) with a regularization term to ensure the Gaussians adhere closely to the surface of objects, it has been observed that this approach does not guarantee the sparsity of the Gaussians. Even after training for 15,000 epochs, there are still residual Gaussians with low opacity inside objects (Figure 1). These residual Gaussians not only fail to contribute to the rendering results but also adversely affect the rendering and reconstruction outcomes.

In this paper, we propose a method to improve the efficiency and effectiveness of 3D Gaussian Splatting by introducing $\mathcal{L}_1$ regularization terms and entropy-loss on opacity of Gaussians during training to promote sparsity, a stage we refer to as ”Pruning”. Additionally, we incorporate $\mathcal{L}_1$ regularization on opacity during the Aligned Gaussian process to encourage Gaussians to be evenly distributed over the scene surface, thereby capturing scene geometry more accurately while reducing the number of Gaussians that are not aligned with the surface. Our approach involves deriving a volume density from the Gaussians, assuming they are flat and well distributed over the scene surface while maintaining sparsity. By minimizing the difference between this derived density and the actual density computed from the Gaussians during optimization, we encourage the 3D Gaussians to better represent the surface geometry.

Introducing a regularization term into the optimization of a Gaussian Splatting model has proven beneficial, making it easier to sample from the surface Gaussians. Similar

Figure 2: The model we trained can be directly opened in mesh editing software like Blender for further creative work.

to SuGaR [2], the regularization term can be evaluated using a density function, and extracting level sets of this function provides a natural approach for sampling. However, the optimization process of the 3D Gaussian Space involves densification to better represent high-frequency surface details, resulting in numerous low-opacity Gaussians, as shown in Figure 1. To address this, we have adopted a method inspired by SuGaR, efficiently sampling points on the visible part of a level set of the density function. This approach allows us to apply the Poisson reconstruction algorithm to these points, obtaining a triangle mesh representing the surface. Notably, this method is scalable and significantly faster than other approaches, such as the Marching Cubes algorithm, which can take at least 24 hours on a single GPU for methods relying on Neural Signed Distance Functions (SDFs) for mesh extraction from radiance fields.

As shown in the Figure Figure 2 and 8, our method is capable of reconstructing higher-quality meshes, thereby reducing the workload for post-mesh editing. Similar to SuGaR, extracting the mesh also requires determining the positions of points on the object's surface, known as the **Level Set**. To obtain these points, we rely on the depth maps of the Gaussians seen from the training viewpoints, which can be acquired by extending the Gaussian Splatting rasterizer. This approach enables us to efficiently sample points on the visible part of the level set of the density function, ensuring that our reconstructed mesh aligns closely with the original surface geometry.

Finally, following SuGaR's approach, we have developed a refinement strategy that

jointly optimizes the mesh extraction process. This method further utilizes Gaussian splatting optimization on the extracted scene surface mesh, enhancing the final render quality. By refining the surface mesh using Gaussian splatting, we can improve the alignment of the mesh with the underlying geometry, resulting in a more accurate representation of the scene. This refinement strategy allows us to achieve higher render quality, making our method suitable for applications where visual fidelity is crucial, such as virtual reality simulations, computer-aided design, and scientific visualization. Overall, this refinement strategy enhances the capabilities of our approach, providing users with a powerful tool for generating high-quality meshes from 3D Gaussian Spaces. In summary, our contributions include:

- Reduces the time consumption of the Gaussian pruning process by about 18% by introducing the L1 regularization term.
- Introducing L1 regularization in the alignment Gaussian training process, thus balancing the smoothness and complexity of the extracted model.

2 Related Works

We will begin with a brief overview of traditional rendering and reconstruction techniques, followed by an exploration of Neural Radiance Fields (NeRF) research. NeRF shares similar ideas with 3D Gaussian (3DGS) Splatting, particularly in the context of rendering; radiance fields are a vast area, so we only mention directly related work. Subsequently, we will introduce 3D Gaussian Splatting and reconstruction methods based on 3DGS.

2.1 Traditional Scene Reconstruction and Rendering

The evolution of novel-view synthesis predates the advent of NeRF, with initial efforts centered around light fields and fundamental scene reconstruction methods [4][5]. However, these early techniques were constrained by their dependence on dense sampling and structured capture, posing significant difficulties in dealing with intricate scenes and lighting conditions. The advent of Structure-from-Motion (SfM) [6] enabled an entire new domain where a collection of photos could be used to synthesize novel views

and scene reconstruction. SfM is used to estimate a sparse point cloud during camera calibration, initially employed for basic 3D space visualization. The subsequent development of multi-view stereo (MVS) [6] algorithms has enabled impressive full 3D reconstruction. These approaches re-project and blend input images into the novel view camera, using geometry to inform this re-projection. While these methods have achieved impressive results in numerous cases, they often struggle to fully recover from unreconstructed regions or ”over-reconstruction,” which occurs when MVS generates non-existent geometry [16].

2.2 Neural Radiance Fields

Early approaches to novel-view synthesis incorporated deep learning techniques [7], where Convolutional Neural Networks (CNNs) were utilized to estimate blending weights [8]. However, a key limitation of many of these methods is their reliance on Multi-View Stereo (MVS)-based geometry [16], which can lead to issues such as temporal flickering when CNNs are employed for final rendering. The concept of volumetric representations for novel-view synthesis was pioneered by Soft3D [9]. However, rendering with volumetric ray-marching is computationally expensive, primarily due to the substantial number of samples needed to query the volume[16]. Neural Radiance Fields (NeRF) [1] introduced techniques such as importance sampling and positional encoding to enhance quality, but their use of a large Multi-Layer Perceptron (MLP) had a negative impact on speed. The success of NeRF has led to an explosion of subsequent methods aimed at improving both quality and speed, often through the introduction of regularization strategies. The current state-of-the-art in image quality for novel-view synthesis is Mip-NeRF360 [10]. While the rendering quality is exceptional, training and rendering times remain extremely high.

Recent methods in neural radiance fields have primarily concentrated on achieving faster training and rendering speeds by leveraging three design choices: the utilization of spatial data structures to store (neural) features for subsequent interpolation during volumetric ray-marching, different encoding strategies, and adjusting the capacity of Multi-Layer Perceptions (MLPs) [16]. Among these methods, InstantNGP [11] is noteworthy for its use of a hash tables and an occupancy grid to accelerate computation, in addition to employing a smaller MLPs for modeling density and appearance. While Instant

NGP shows exceptional results, these methods can encounter difficulties in effectively representing empty space, particularly depending on the scene or capture type. Furthermore, the choice of structured grids used for acceleration often limits image quality, and rendering speed is hampered by the necessity to query numerous samples for each ray-marching step.

2.3 3D Gaussian Splatting for Rendering and Reconstruction

Generally the NeRF-liked methods are considered as implicit radiance fields as they represent the light distribution in a scene without explicitly defining the geometry of the scene, while the explicit radiance field directly represents the distribution of light in a discrete spatial structure, such as a voxel grid [12] or a set of point [13]. 3D Gaussian Splatting (3DGS) represents a shift from implicit to explicit radiance fields. This strategy harnesses the respective strengths of both methodologies, employing 3D Gaussians as a malleable and effective mode of representation. These Gaussian models are optimized to accurately represent the scene, merging the advantages of neural network-driven optimization and precise, organized data storage [14]. This combined method targets superior rendering quality, facilitating quicker training and real-time performance, especially beneficial for intricate scenes and high-resolution outcomes.

While 3D Gaussian Splatting enable highly realistic renderings of scenes, extracting the scene's surface from them remains a challenging task, as depicted in the figure. 3D Gaussian Splatting represents the scene as a large set of Gaussians, which can lead to a significant number of Gaussians with low opacity inside the scene (e.g., within the interior of a 3D object). This makes it difficult to compute the boundary positions of objects from the scene's density, thus hindering mesh extraction. The Surface-Aligned Gaussian Splatting (SuGaR) [2] method provides a creative solution by introducing an entropy loss to encourage the opacity of Gaussians to approach binary values. It then introduces a series of Signed Distance Functions (SDFs) combined with density to align the Gaussians with the surface of objects. Subsequently, the method samples points on these Gaussians to obtain level sets, which are then used with Poisson reconstruction [15] to extract the mesh.

Despite providing an efficient approach, making mesh extraction feasible, this method still requires a lengthy sparsification process (approximately 2,000 iterations), which

we refer to as **Pruning**, where unnecessary or low-opacity Gaussians are pruned. Additionally, this method suffers from slow optimization speeds. One reason for this is the need for a large number of Gaussians to represent the scene, which consumes a significant amount of memory. By introducing our regularization term, we can further prune unnecessary Gaussians, optimizing the required memory and thus improving the optimization speed.

3 Research Problem

To better demonstrate our work, we will first provide definitions for the concepts commonly used in our work in this section, and then describe the problem our paper aims to address.

3.1 Basic Definition

Scene Representation involves representing a 3D model of a scene from a collection of images or other data. It usually can be described as a function $F_\Theta : (\mathbf{x}, \mathbf{d}) \rightarrow (\mathbf{c}, \sigma)$, where the input of the function is a 3D location $\mathbf{x} = (x, y, z)$, and 2D viewing $\mathbf{d} = (\theta, \phi)$ and whose output is an emitted colour $\mathbf{c} = (r, g, b)$ and volume density σ [1].
Generally, we consider NeRF-like methods as Implicit Radiance Fields, where the radiance at any point is not stored explicitly but is computed on-the-fly by querying the neural network. Hence, the function can be written as

$$F_{\text{implicit}}(\mathbf{x}, \mathbf{d}) = \text{NeuralNetwork}(\mathbf{x}, \mathbf{d}), \tag{1}$$

while the 3D Gaussian representation is formulated as:

$$F_{\text{3DGS}}(\mathbf{x}, \mathbf{d}) = \sum_i G(\mathbf{x}, \mu_i, \Sigma_i) \cdot c_i(\mathbf{d}), \tag{2}$$

where c represents the view-dependent color, and G corresponds to the Gaussians defined by a full 3D covariance matrix $\Sigma_i \in R^{3\times3}$ and mean $\mu_i \in R^3$:

$$G(\mathbf{x}, \mu_i, \Sigma_i) = \exp(-\frac{1}{2}(\mathbf{x} - \mu_i)^T \Sigma_i^{-1}(\mathbf{x} - \mu_i)). \tag{3}$$

For more information on explicit and implicit radiance fields please refer to the survey [14]. Since the 3D Gaussian representation can be viewed as a linear combination of Radial Basis Functions (RBFs), it inspires us to consider optimizing this method through linear algebra.

Rendering is a term that specifically refers to the process of converting computer-readable information, such as the scene representation, into pixel-based images. In the context of our work, rendering involves taking the scene representation, which includes information about the scene's geometry, lighting, and materials, and using this information to generate images that simulate how the scene would look from a specific viewpoint.

Signed Distance Function (SDF) is a mathematical function that defines the distance from any point in space to the nearest surface of a geometric object, such as a shape or a surface. If Ω is a subset of a metric space, for example, $\mathbb{R}^3$ in our paper, then the signed distance function F is defined by

$$F(\mathbf{x}) = \begin{cases} d(\mathbf{x}, \partial\Omega) & \text{if } \mathbf{x} \in \Omega, \\ -d(\mathbf{x}, \partial\Omega) & \text{if } \mathbf{x} \in \Omega,^c \end{cases} \tag{4}$$

where $\partial\Omega$ denotes the boundary of Ω and for any $\mathbf{x} \in \mathbb{R}^3$, and

$$d(\mathbf{x}, \partial\Omega) := \inf_{\mathbf{y} \in \partial\Omega} d(\mathbf{x}, \mathbf{y}). \tag{5}$$

Level Set refers to a collection of points in a measure space corresponding to a specific value that represents the surface $\partial\Omega$ where the Signed Distance Function (SDF) equals that value. Formally, for a given threshold τ and SDF $F(\mathbf{x})$, the Level Set $\mathcal{P}$ is defined as:

$$\mathcal{P} := \{\mathbf{x} \in \mathbb{R}^3 : F(\mathbf{x}) = \tau\}, \tag{6}$$

3.2 Problem Definition

Problem 1: View Synthesis. *Given a set of input images $\{\mathcal{I}_k\}$ of a scene, represent the scene using 5D vector-valued volumetric scene function $F_\Theta : (\mathbf{x}, \mathbf{d}) \rightarrow (\mathbf{c}, \sigma)$.*

Problem 2: Reconstruction. *Find a level set of n distinct points $\mathcal{P} = \{(x_i, y_i, z_i)_{i=1}^n\}$ on a surface M in $\mathbb{R}^3$, and then find a surface M' that reasonably approximates M.*

Our approach involves representing a scene using 3D Gaussians: we combine tens of thousands of Gaussians in world space and then render images $\{\mathcal{I}'_k\}$ from training viewpoints by splatting. We optimize the coefficients and color parameters of the Gaussians by minimizing the error between the rendered images $\{\mathcal{I}_k\}$ and the ground truth. We then sample a level set $\mathcal{P}$ with a specified level parameter μ. Finally, we extract the scene's surface mesh by running Poisson reconstruction on the level set $\mathcal{P}$ [15].
We focus on efficiently extracting these level sets, with the goal of using them for future reconstruction work. For now, we are using Poisson Reconstruction [15]. Additionally, similar to SuGaR, we provide an optional refinement strategy that binds Gaussians to the surface of the mesh. This strategy jointly optimizes these Gaussians and the mesh through Gaussian splatting rendering.

4 Method

4.1 Initialization

3DGS takes a set of images $\{\mathcal{I}_k\}$ of a static scene as inputs and utilizes Structure-from-Motion (SfM) [6] to calibrate cameras and generate a sparse point cloud. From these points, 3DGS initializes a set of 3D Gaussians G_i with a position μ, covariance matrix Σ, opacity α, and spherical harmonics (SH), enabling a highly flexible optimization field [16], as shown in Figure 3.(a). While random initialization can be used as an alternative to obtaining initialization from SfM, we did not adopt this method because it cannot guarantee the accuracy of rendering and reconstruction [16].

4.2 Rendering Procedure

To better describe the concept and process of volume rendering, we start with introducing the scene representation and point-based volume rendering in NeRF, and then discuss the volume rendering process in 3DGS, as 3DGS modifies the point-based approach.

Point-Based Volume Rendering Light reaching our eyes (or camera) as a result of light being reflected by an object or being emitted by a light source is likely to be absorbed as it travels through a volume of space filled up with some particles. We describe

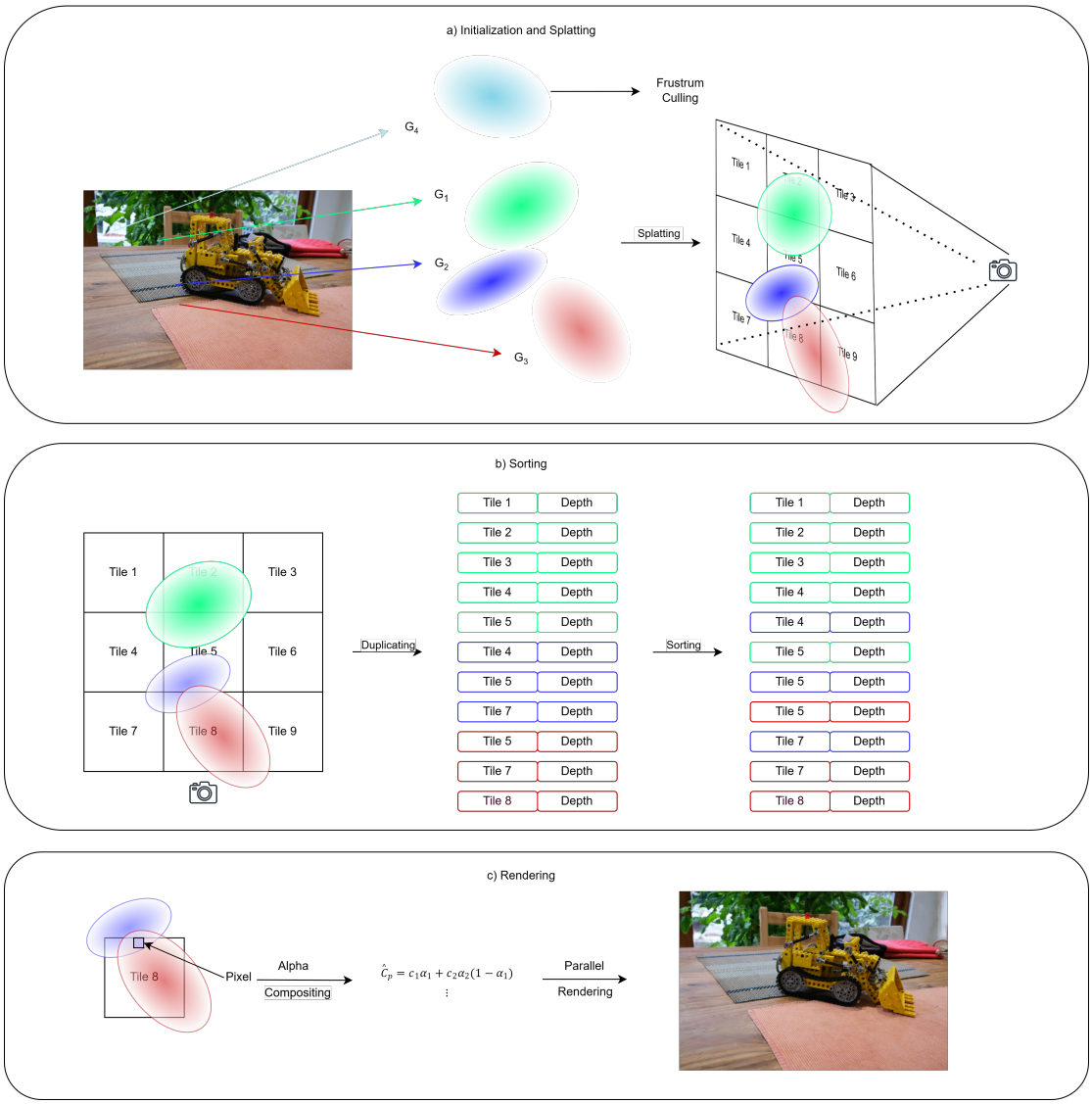

Figure 3: (a) Through the initialization, Frustum Culling, and Splatting process, Gaussians in the actual space are projected onto the camera plane. (b) 3D Gaussian Splatting divides the image to be rendered into non-overlapping patches (referred to as tiles in the original paper), then replicates the Gaussians that cover several tiles, reorders them according to their tile and depth, and assigns a unique id to each Gaussian. (c) Calculates the color of each pixel using alpha-compositing. To accelerate this process, 3D Gaussian Splatting employs techniques such as parallel computation.

this phenomenon using principles from classical volume rendering [17] . The volume density function can be interpreted as the differential probability of ray ending at a tiny particle located at **x**. The anticipated color $C(\mathbf{r})$ of a camara ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ between the near bounds t_n and far bounds t_f is given by:

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})dt, \tag{7}$$

where $\sigma(\mathbf{r}(t))$ is the density function with location input **x** and $\mathbf{c}(\mathbf{r}(t), \mathbf{d})$ is the color when viewing a particle at location **x** from direction **d**. Therefore, NeRF can render the color of different materials under different angles and lighting conditions, such as glass. The function $T(t)$ represents the accumulated transmittance along the ray from t_n to t_f which can be calculated as:

$$T(t) = \exp(-\int_{t_n}^{t_f} \sigma(\mathbf{r}(s))ds). \tag{8}$$

The proof of the above formula is also shown in Appendix A.1. However, to implement the formula in a program, we must first express it in a form suitable for Riemann sum approximation. In simple NeRF pipeline, the stratified sampling approach is applied as the MLP would only be queried at a fixed discrete set of locations [1]:

$$t_i \sim \mathcal{U}\left[t_n + \frac{i-1}{N}\left(t_f - t_n\right), t_n + \frac{i}{N}\left(t_f - t_n\right)\right]. \tag{9}$$

Then We utilize these samples to approximate $C(\mathbf{r})$ using the quadrature rule outlined in Max's review on volume rendering [18]:

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i \left(1 - \exp\left(-\sigma_i \delta_i\right)\right) \mathbf{c}_i, \text{ where } T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right), \tag{10}$$

the $\delta_i = t_{i+1} - t_i$ is the distance of two adjacent sample points. This form of function $\hat{C}(\mathbf{r})$ is trivially differentiable and can be reduced into tradintional alpha compositing [19] with alpha values $\alpha_i = 1 - \exp(\sigma_i \delta_i)$ [1], a typical neural point-based method [20] calculates the color $\hat{C}$ of a pixel by blending N ordered points that overlap the pixel:

$$\hat{C} = \sum_{i \in \mathcal{N}} c_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j). \tag{11}$$

Research and experiments shown that deep neural networks are biased towards learning lower frequency functions [1] [21], to encourage the neural network learn the higher

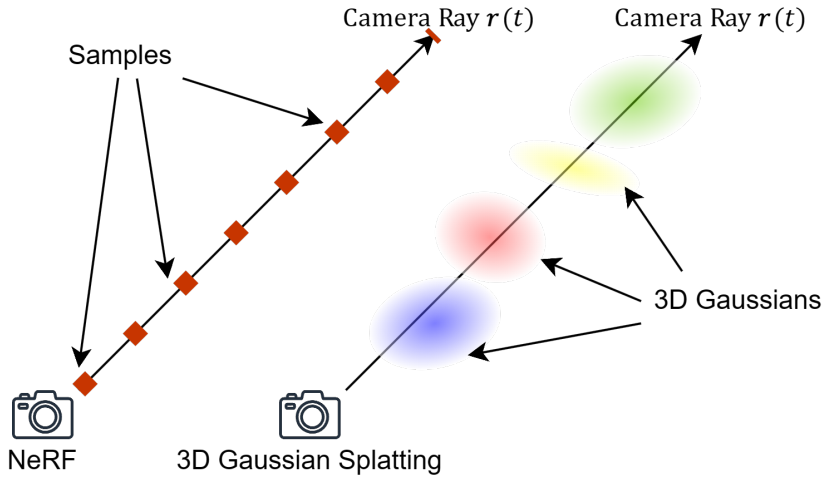

Figure 4: The distinction in the mapping process between NeRF and 3D Gaussian Splatting.

frequency features, numerous endeavors have been undertaken: encoding the position input $xyz\theta\phi$ into a higher dimensional space using fourier position encoding [1]; using multi-resolution hash encoding to significantly reduce the input dimension of the model, speeding up the training process [11]; employing local radial basis functions and multi-frequency sinusoid functions to interpolate features at continuous query points [14].

3D Gaussian Splatting In the process of rendering new views in NeRF, we cast a ray for each pixel of the new view and then sample along this ray to query the properties of the sample points. This process can be viewed as *backward mapping*. It requires continuously querying the most detailed points along the ray and then performing interpolation calculations, which consumes a large amount of computational resources and leads to slow rendering and training speeds. Therefore, a natural approach is *forward mapping*, which involves directly projecting the attributes of ”volume data” in camera space onto the image plane corresponding to the pose of the camera. Then, these projected data are sorted according to their distances from the plane and alpha-composited. There are many forms of ”volume data”, such as triangle primitive elements, which is one of our future research directions. Currently, we continue the idea of 3D Gaussian Splatting, using Gaussian functions to represent the volume data in camera space.
Due to the faster computation of projection compared to querying, the time complexity is lower, and 3D Gaussian Splatting can train a scene on a single GPU in a few minutes. To render a picture more efficiently, the following techniques have been applied:

Frustum Culling. As shown in Figure 3.(a), 3DGS splits the screen into ”tiles”, then culls 3D Gaussians against the view frustum and each tile. Gaussians at extreme positions (i.e., Gaussians with mean too close to the near plane and far outside the view frustum) are ignored by a ”guard band” [14].

Gaussian Splatting. Previous work [22] demonstrate how to project the 3D Gaussians to 2D for rendering:

$$\Sigma' = JW\Sigma W^T J^T, \tag{12}$$

where W is a viewing transformation depends on the camera, and J is the Jacobian of affine approximation of projective transformation, the details proof can be viewed in Appendix A.2. Σ' is a 3×3 variance matrix in the camera space, if we skip the third row and column we can obtain a 2×2 matrix with the same structure and properties as if we would start from planar points with normals [16], as in previous work [20]. From the fisrt two rows and columns of Σ' we can compute the standard derivation σ_1 and σ_2 of the 2D Gaussian slices in the camera space using the eigenvalues. To save computational resources, we only consider an ellipse with a semi-major axis of $a = 3\max\{\sigma_1, \sigma_2\}$ and a semi-minor axis of $b = 3\min\{\sigma_1, \sigma_2\}$ as the effective projection of the 3D Gaussian on the camera plane. The covariance matrix Σ have physical meaning only when they are semi-definite, as this properties may be destroyed during optimization, 3DGS represents the covariance matrix Σ with a scaling matrix S and rotation matrix R:

$$\Sigma = RSS^T R^T, \tag{13}$$

3DGS restore both factors R and S separately to allow optimization independently: a 3D scaling vector s and a quaternion q to represent rotation.

Rendering by Pixels & Tiles. Before delving into the final version of 3D GS which utilizes several techniques to boost parallel computation, we first elaborate its simpler form to offer insights into its working mechanism. Given the position of a pixel p, its distance to all overlapping Gaussians, i.e., the depths of these Gaussians, can be computed through the viewing transformation W , forming a sorted list of Gaussians $\mathcal{N}$, As shown in Figure 3.(b). Then, alpha compositing is adopted to compute the final color of this pixel, as shown in Figure 3.(c) [14]:

$$\hat{C}_p = \sum_{i \in \mathcal{N}} c_i \alpha'_i \prod_{j=1}^{i-1} (1 - \alpha'_j), \tag{14}$$

$$\text{where } \alpha'_i = \alpha_i G(\mathbf{x}, \mu_i, \Sigma_i) = \alpha_i \exp(-\frac{1}{2}(x - \mu_i)^T \Sigma_i^{-1}(x - \mu_i)), \tag{15}$$

c_i represents the learned color of Gaussian G_i, while the final opacity α'_i is the product of the learned opacity α_i and the Gaussian. Here, x and $\mu_\mathbf{i}$ are the coordinates in camera space.

To avoid the potentially slower rendering speeds compared to NeRF when processing the above procedure pixel by pixel, 3D Gaussian Splatting divides the image into multiple non-overlapping patches, referred to as "tiles" in the original text. Each tile consists of **16** $\times$ **16** pixels, and rendering is performed separately for each tile. The specific method involves using a sorting technique similar to radix sort to sort the Gaussians based on their index in the tiles and their distance. If a Gaussian covers multiple tiles, it is duplicated for each tile it covers during rendering, as illustrated in the figure. Parallel computation and other methods are then applied to accelerate rendering for each pixel and tile.

4.3 Densification

In our work, because this part of the pipeline generates many dense Gaussians inside the 3D objects, we refer to the optimization of this aspect of 3D Gaussian Splatting as **Densification**. The optimization process involves successive iterations of rendering and computing the loss between resulting image and the training views in the captured dataset. However, due to the ambiguities of 3D to 2D projection, geometry may be incorrectly placed. To address this issue, 3D Gaussian Splatting uses a sigmoid activation function for α to constrain it to the $[0, 1)$ range, as well as an exponential activation function for the scale of the covariance for similar reasons. The loss function combines $\mathcal{L}_1$ with a D-SSIM term:

$$\mathcal{L}_D = (1 - \lambda)\mathcal{L}_1 + \lambda \mathcal{L}_{D-SSIM}, \tag{16}$$

For a given target image $\mathcal{I}_s$ and the synthesized image $\mathcal{I}'_s$, the loss functions $\mathcal{L}_1$ and $\mathcal{L}_{D-SSIM}$ are:

$$\mathcal{L}_1(\mathcal{I}_s, \mathcal{I}'_s) = \frac{1}{H \times W} \sum_{i=1}^{W} \sum_{j=1}^{H} |\mathcal{I}_s(i,j) - \mathcal{I}'_s(i,j)|, \tag{17}$$

$$\mathcal{L}_{D-SSIM}(\mathcal{I}_s, \mathcal{I}'_s) = 1 - \text{D-SSIM}(\mathcal{I}_s, \mathcal{I}'_s), \tag{18}$$

H and W denote the height and width of the images respectively and (i, j) indicates a single pixel of an image. D-SSIM represents the structural similarity index measure in the difference domain which is defined as:

$$\text{D-SSIM}(\mathcal{I}_s, \mathcal{I}'_s) = \frac{2\mu_{\mathcal{I}_s}\mu_{\mathcal{I}'_s} + c_1}{\mu_{\mathcal{I}_s}^2 + \mu_{\mathcal{I}'_s}^2 + c_1} \cdot \frac{2\sigma_{\mathcal{I}_s\mathcal{I}'_s} + c_2}{\sigma_{\mathcal{I}_s}^2 + \sigma_{\mathcal{I}'_s}^2 + c_2}, \tag{19}$$

where $\mu_{\mathcal{I}_s}$ and $\mu_{\mathcal{I}'_s}$ are the means of the target image $\mathcal{I}_s$ and the synthesized image $\mathcal{I}'_s$, respectively; $\sigma_{\mathcal{I}_s}^2$ and $\sigma_{\mathcal{I}'_s}^2$ are the variances of $\mathcal{I}_s$ and $\mathcal{I}'_s$, respectively; $\sigma_{\mathcal{I}_s\mathcal{I}'_s}$ is the covariance between $\mathcal{I}_s$ and $\mathcal{I}'_s$; c_1 and c_2 are constants added for numerical stability.
In this part, we use the same $\lambda = 0.2$ as in 3D Gaussian Splatting.

4.4 Pruning

In densification, to represent the texture, material, and other details of the scene, these Gaussians often have low opacity and are repeated. Although this part of the work can effectively represent the scene, it is difficult to extract the surface of objects from these Gaussians. Therefore, when 3D Gaussian Splatting can effectively represent the scene (for example, after iterating more than 7000 times), we prune the unnecessary Gaussians. In practice, we introduce an $\mathcal{L}_1$ regularization term for opacity and an entropy loss term into the loss function. The updated objective function is then:

$$\mathcal{L}_P = \mathcal{L}_D + \mathcal{E}(\alpha) + \mathcal{L}_1(\alpha), \tag{20}$$

where $\mathcal{L}_D$ is the loss function in Densification, and $\mathcal{E}(\alpha)$ and $\mathcal{L}_1(\alpha)$ is the entropy loss and L1 loss on opacity:

$$\mathcal{E}(\alpha) = -\sum_i \left(\alpha_i \log \alpha_i + (1-\alpha_i)\log(1-\alpha_i)\right), \mathcal{L}_1(\alpha) = \sum_i |\alpha_i| \tag{21}$$

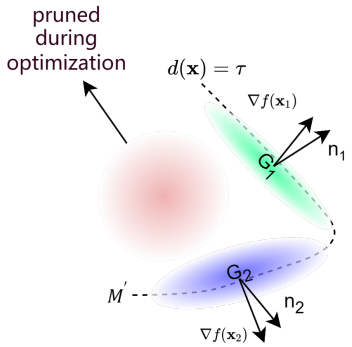

Figure 5: During the process of aligning Gaussians to the surface, low-opacity Gaussians inside the object are either optimized away or gradually brought closer to the surface. This is achieved by computing $d(\mathbf{x}) = \tau$ to obtain a series of level sets, such as $\mathbf{x_1}$, $\mathbf{x_2}$, and then calculating the normals of the points belonging to the level set, $\nabla f(\mathbf{x_1})$, $\nabla f(\mathbf{x_2})$. Subsequently, the error between the normals n_1, n_2 of the Gaussians and the direction of the Gaussians is optimized to refine the normals of the Gaussians.

4.5 Reconstruction

Despite reducing the unnecessary number of Gaussians through pruning, extracting a mesh from a 3D Gaussian Space (3DGS) remains a challenging task. To address this, we aim to reduce the Gaussians within 3D objects, aligning them with the surface, a process referred to as Gaussian Aligning with the Surface (SuGaR) [2], which essentially means spatial Gaussian sparsification. For this purpose, we consider the density function $d : R^3 \rightarrow R_+$ in 3DGS space, computed as the sum of Gaussian values weighted by their alpha coefficients at any space location $\mathbf{x}$:

$$d(\mathbf{x}) = \sum_i \alpha_i G(\mathbf{x}, \mu_i, \Sigma_i). \tag{22}$$

We want the Gaussians to be sparse within the 3D object and have higher opacities on the surface. Thus, when computing the Gaussian value at $\mathbf{x}$, we can approximate that only the Gaussian closest to $\mathbf{x}$ contributes significantly, i.e.:

$$d(\mathbf{x}) \approx \alpha_k G(\mathbf{x}, \mu_k, \Sigma_k), \tag{23}$$

SuGaR defines this ”closest Gaussian” as g^*, which is the Gaussian with the largest contribution at point $\mathbf{x}$:

$$g^* = \arg\min_g \{(\mathbf{x} - \mu_i)^T \Sigma_i^{-1} (\mathbf{x} - \mu_i)\}. \tag{24}$$

Additionally, to make the Gaussian "close" to the surface of the 3D object, SuGaR ensures that the Gaussian is as "flat" as possible, which means one component of the scaling vector s should be close to 0. This allows further approximation of the Gaussian value:

$$(\mathbf{x} - \mu_i)^T \Sigma_i^{-1} (\mathbf{x} - \mu_i) \approx \frac{1}{s_i^2} \langle \mathbf{x} - \mu_i, n_i \rangle^2, \tag{25}$$

s_i is the smallest scaling factor of the Gaussian, and n_i is the direction of the corresponding axis. Therefore, the density can be further approximated as:

$$\bar{d}(\mathbf{x}) = \exp(-\frac{1}{2s_i^2} \langle \mathbf{x} - \mu_i, n_i \rangle^2), \tag{26}$$

introducing an SDF combined with density ensures Gaussian Aligning with the Surface:

$$\bar{f}(x) = \pm s_g * \sqrt{-2 \log(\bar{d}(\mathbf{x}))}, \tag{27}$$

the regularization term $\mathcal{R}$ at this point is:

$$\mathcal{R} = \frac{1}{|\mathcal{X}|} \sum_{\mathbf{x} \in \mathcal{X}} |\hat{f}(\mathbf{x}) - \bar{f}(\mathbf{x})|, \tag{28}$$

where $\bar{f}(\mathbf{x})$ is the ideal SDF, $\hat{f}(\mathbf{x})$ is the SDF on the surface created by the current Gaussians, and $\mathcal{X}$ is the set of sample points:

$$\mathcal{X} = \{\mathbf{x} \in R^3 | \mathbf{x} \sim \prod_i G(\cdot; \mu_i, \Sigma_i)\}, \tag{29}$$

in addition to aligning the Gaussians with the surface of the object, we also need to ensure that these Gaussians have normals consistent with the surface's normals. This means the normal of the SDF f, $\frac{\nabla f(\mathbf{x})}{\|\nabla f(\mathbf{x})\|_2}$, should align with the normal n_i of the SDF $\bar{f}$:

$$\mathcal{R}_{\text{Norm}} = \frac{1}{|\mathcal{X}|} \sum_{\mathbf{x} \in \mathcal{X}} \left\| \frac{\nabla \bar{f}(\mathbf{x})}{\|\nabla \bar{f}(\mathbf{x})\|_2} - n_{g^*} \right\|_2^2. \tag{30}$$

Furthermore, we introduce an $\mathcal{L}_1$ penalty term for opacity, so the loss function at this stage becomes:

$$\mathcal{L}_A = \mathcal{L}_D + \mathcal{R} + \mathcal{R}_{\text{Norm}} + \mathcal{L}_1(\alpha). \tag{31}$$

4.6 Level Set and Mesh Extraction

To effectively extract the surface of a 3D object, we need to obtain a series of level sets on the object's surface. Fortunately, when we optimize the Gaussians on the surface of the 3D object, we can conveniently calculate these level sets. Specifically, we set a level parameter μ. When the Gaussian value $d(\mathbf{x})$ at a point $\mathbf{x}$ equals μ, we consider this point to belong to the level set: $\mathbf{x} \in \mathcal{P}$, As shown by the dashed line in the Figure 5. Formally, we sample random pixel points on the image to obtain the depth m and the direction ray v of the camera corresponding to these pixels. Then, we sample n points on the ray $m + t_j v$, where $t_j \in [-3\sigma_i(v), 3\sigma_i(v)]$. Here, $\sigma_i(v)$ is the standard deviation of the 3D Gaussian g_i in the direction of the camera. The interval $[-3\sigma_i(v), 3\sigma_i(v)]$ is the confidence interval for the 99.7 confidence level of the 1D Gaussian function of t along the ray, where $t_j = -3\sigma_i(v) + \frac{6j\sigma_i(v)}{n}$ [14].

Next, we calculate the density value $d_j = d(m + t_j v)$ for each sampled point on the ray until $d_j < \mu < d_{j+1}$. Then, we linearly interpolate to approach the points $\mathbf{x}$ that actually belong to the level set:

$$t^* = t_j + \frac{\mu - d_j}{d_{j+1} - d_j} \cdot \frac{6\sigma_i(v)}{n}, \tag{32}$$

after obtaining these level set points, we apply Poisson reconstruction [15] to reconstruct the mesh of the 3D object.

5 Experiments

5.1 Implementation details

All our models are optimized on a single Nvidia A10 Tensor Core GPU with 24GB of memory. Additionally, the CPU is an Intel(R) Xeon(R) Platinum 8369B CPU @ 2.90GHz, with 30GB of memory, and the operating system is Ubuntu 22.04.

Regularization. Initially, we optimize a Gaussian Splatting model without regularization over 7,000 iterations. This allows the 3D Gaussians to position themselves freely

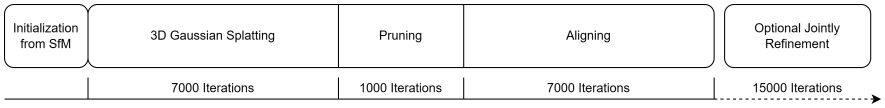

Figure 6: The training pipeline of our method. It is noteworthy that in SuGaR, pruning requires 2000 iterations, but due to the introduction of $\mathcal{L}_1$ regularization in our approach, we do not need as many iterations to achieve a very good result.

without additional constraints. we perform only 1,000 iterations, incorporating an extra entropy loss term and applying $\mathcal{L}_1$ normalization to the opacities α_i of all Gaussians. This is in contrast to SuGaR, which uses 2000 iterations. The loss function at this stage is denoted as $\mathcal{L}_P$. This regularization method encourages the opacities to become binary, promoting sparsity in the spatial representation. We refer to these 1,000 iterations as **pruning**, as we remove Gaussians with opacity values $\alpha_i < 0.5$ every 500 iterations.

Alignment. Next, we enter the Reconstruction phase, where we perform 6,000 iterations with the regularization term. In this phase, we optimize the 3D Gaussian Space by introducing Signed Distance Fields (SDFs) to align it with the surface of the scene. The loss function at this stage is denoted as $\mathcal{L}_R$. Additionally, unlike SuGaR, we use the K-Nearest Neighbors (KNN) algorithm to query the 32 nearest Gaussians to a spatial point $\mathbf{x}$ when computing the Gaussian values at $\mathbf{x}$, which achieves higher accuracy. We then recompute the neighbors every 500 iterations.

Joint Refinement. We follow the refinement strategy in SuGaR [2], through optional jointly refine the mesh and the bound 3D Gaussians for 15,000 iterations. Since this process takes 1-2 hours on our equipment, we only performed this process on the Mip-NeRF360 [10] dataset.

5.2 Real-Time Rendering of Real Scenes

To evaluate our model, we adopt the methodology outlined in the original 3D Gaussian Splatting paper [16] and assess various iterations of our approach following refinement on real 3D scenes from the Mip-NeRF360 [10] dataset. However, due to licensing constraints and the unavailability of the Flowers and Treehill scenes, we evaluate all methods using only 7 scenes from the Mip-NeRF360 dataset, rather than the complete set of 9 scenes. Additionally, due to limitations in GPU memory size, we resize the input images of outdoor scenes to 1/4 of their original size to reduce memory consumption.

Table 1: In the evaluation of rendering quality on the Mip-NeRF360 dataset [10], our method in the reconstruction with mesh achieved a level similar to SuGaR, and even surpassed our prototype SuGaR in outdoor scenes, the ”R-SuGaR-15k” refers to the model obtained after training the joint refinement for 15,000 epochs. Furthermore, our method performs well compared to NeRF and 3D Gaussian Splatting methods.

Scene	Indoor scenes			Outdoor scenes			Average on all scenes		
Metric	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
No mesh (except Ours)									
Plenoxels [23]	24.83	0.766	0.426	22.02	0.542	0.465	23.62	0.670	0.443
INGP-Base [11]	28.65	0.840	0.281	23.47	0.571	0.416	26.43	0.725	0.339
INGP-Big [11]	29.14	0.863	0.242	23.57	0.602	0.375	26.75	0.751	0.299
Mip-NeRF360 [10]	31.58	0.914	0.182	25.79	0.746	0.247	29.09	0.842	0.210
3DGS [15]	30.41	0.920	0.189	26.40	0.805	0.173	28.69	0.870	0.182
Ours	29.07	0.901	0.218	24.48	0.704	0.297	27.32	0.821	0.252
With mesh									
Mobile-NeRF [24]	-	-	-	21.95	0.470	0.470	-	-	-
NeRFMeshing [25]	23.83	-	-	22.23	-	-	23.15	-	-
BakedSDF [26]	27.06	0.836	0.258	-	-	-	-	-	-
R-SuGaR-15K[2]	29.43	0.910	0.216	24.40	0.699	0.301	27.27	0.820	0.253
Ours	29.07	0.901	0.218	24.48	0.704	0.297	27.32	0.821	0.252

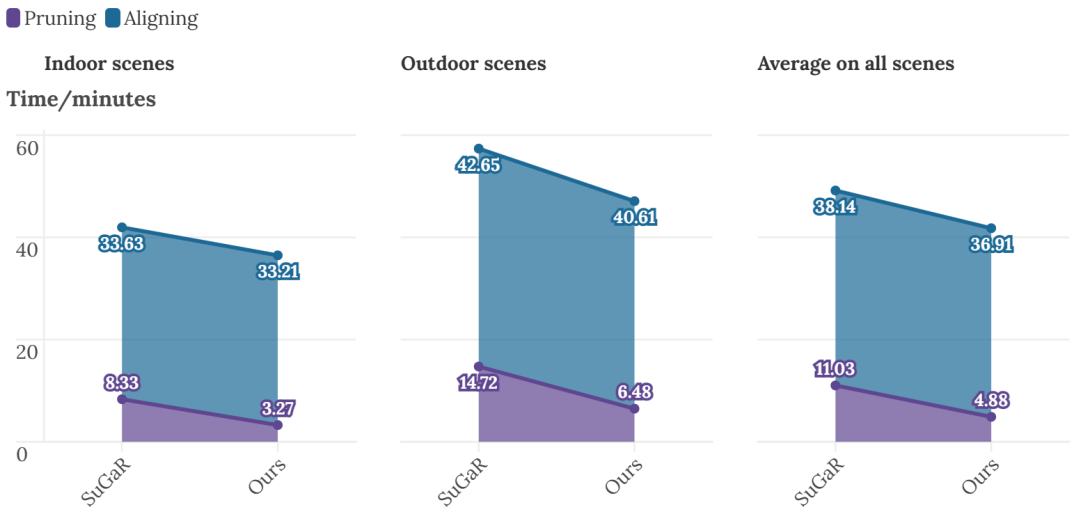

Figure 7: Comparison of the training time of SuGaR with our method On our test platform. Please note that the numbers in the bar chart represent the time taken for each training stage (Pruning and Aligning the Gaussians with the surface). For rigor, we conducted an experiment on three identical devices, and the data in the chart is the average of three experiments on Mip-NeRF360 dataset. The reconstruction time and variance for each scene can be seen in Table 2.

In our experimental results, we did not observe a significant impact on the rendering quality caused by this scaling operation.

We calculate the standard metrics PSNR, SSIM, and LPIPS [27] to assess the rendering quality of our method using the extracted meshes and their corresponding bound surface Gaussians. It is worth noting that Mobile-NeRF [14], NeRFMeshing [25], and SuGaR [2] similarly do not utilize plain textured mesh rendering. We compare our approach to several baselines, among which Plenoxels [13], INGP [11], Mip-NeRF360 [10], and 3DGS [16] focus solely on novel view synthesis tasks, while Mobile-NeRF [14], NeRFMeshing [25], BakedSDF [26], and SuGaR [2] rely on reconstructed meshes for rendering. Our experimental results on the MiP-NeRF360 dataset are presented in the Table 1.

Furthermore, compared to SuGaR, we have accelerated the training speed by approximately **15%** for indoor scenes and **22%** for outdoor scenes in the Mip-NeRF360 dataset by introducing L1 regularization. On average, the acceleration is about **18%**. The specific time consumption is shown in the Figure 7. It can be seen that in outdoor scenes,

Figure 8: The results of our algorithm for reconstructing and extracting grids from images. The term **"Texture"** refers to the material of the grid surface, **"Mesh"** is the colored representation obtained by coloring the Texture on the mesh, and **"Hybrid"** denotes the high-quality rendering of the scene with the Gaussian Splatting rasterizer. The results of the Garden and Kitchen scenes are from the Mip-NeRF360 dataset [10], while the results of Elaina was from our own dataset.

Table 2: Detailed timings (in **minutes**) and variances. Since different test platforms may lead to different results, we present here the timing test results for various scenes in Mip-NeRF360 and our own Elainas dataset.

Scene	Pruning (SuGaR)	Aligning (SuGaR)	Pruning (Ours)	Aligning (Ours)
Kitchen	8.35 (0.25)	33.61 (0.53)	3.19 (0.18)	33.11 (0.42)
Room	8.27 (0.29)	33.52 (0.71)	2.11 (0.37)	32.98 (0.61)
Bonsai	8.37 (0.17)	33.76 (0.52)	4.51 (0.15)	33.55 (0.49)
Stump	13.85 (0.42)	40.89 (0.58)	6.78 (0.43)	35.21 (0.65)
Garden	15.43 (0.48)	44.71 (0.61)	7.91 (0.52)	39.05 (0.73)
Bicycle	14.87 (0.31)	42.37 (0.49)	4.74 (0.29)	36.46 (0.57)
Elaina	11.73 (0.39)	38.89 (0.54)	5.78 (0.34)	34.12 (0.62)

due to the need for more Gaussians to represent the scene, our L1 regularization can more effectively reduce unnecessary Gaussians during the pruning process, thus significantly reducing the required training time compared to indoor scenes. In Figure 7, we can observe that our method significantly improves speed in Pruning training, while the acceleration effect is not significant in Aligning training. This is because in the pruning training process, both our method and SuGaR use the loss function $\mathcal{L}_A$, which explicitly calculates and propagates gradients using the 3DGS CUDA code, so reducing the number of Gaussians directly reduces the training time. However, in Aligning training, we directly use SuGaR's PyTorch code, without explicitly calculating and propagating gradients, which means that reducing the number of Gaussians does not significantly speed up the training process. Therefore, one of our future directions is to explicitly derive the gradients for this part and implement them using CUDA.

In addition, considering the number of Gaussians after pruning, our method can reduce the number of Gaussians needed for reconstruction by about **10%** to **30%** and it can vary across scenes. Since too few Gaussians cannot effectively represent the scene, after multiple experiments, we set the coefficient of the L1 regularization term in the aligning training to 10^{-7} and the coefficient of the L1 regularization term in the pruning process to 10^{-5}. This setting ensures that the opacity α_g of unnecessary Gaussians decreases more quickly to below the threshold, without making the number of Gaussians in the

scene too small to represent the scene effectively.

5.3 Hybrid representation

We also propose a hybrid representation based on SuGaR, where 3D Gaussians aligned with the surface are used to color the mesh surface. This approach builds on differentiable rendering to combine the advantages of mesh-based and volumetric methods, enabling surface reconstruction and improved editability. Specifically, we utilize a hybrid volume-surface representation, which produces high-quality meshes suitable for downstream graphics applications while efficiently capturing view-dependent appearance. Similar approaches optimize neural signed distance functions (SDF) through training neural radiance fields, where density is a differentiable transformation of the SDF [28] [29]. Due to space constraints, we do not elaborate on these works here. The final triangle mesh can be reconstructed from the SDF using the Marching Cubes algorithm [3]. Our experimental results, as shown in Figure 8, demonstrate that our method effectively extracts the mesh of the scene while ensuring high-quality rendering results.

6 Conclusion

While 3DGS can efficiently synthesize and render new viewpoints, this method cannot extract mesh models from the scene. SuGaR innovatively introduced regularization terms for distance and direction, and trained Gaussians to fit the surface of objects through sampling on Gaussians. However, it still faces issues such as slow training, excessive memory consumption by Gaussians, and excessively messy extracted meshes. We propose an improved pruning method by introducing an opacity L1 regularization term during the training process of aligning Gaussians to surfaces. Our method significantly reduces the memory and training time required by SuGaR while maintaining high-quality rendering and mesh quality. Additionally, we can balance the complexity of the extracted mesh between fast and relatively smooth by adjusting the parameters of the regularization term, avoiding overly messy extracted meshes. With this work, we advance the mesh extraction of 3D Gaussian Splatting, providing more possibilities for realistic and virtual interactions.

7 Limitation

Although we were able to reduce the number of unnecessary Gaussians by introducing the L1 regularization term, thus speeding up the training, we did not find a good way to directly measure the importance of these Gaussians, so we resorted to comparing them in terms of results, i.e., comparing the time required during training and the number of Gaussians under the same rendering metrics, such as PSNR as well as SSIM, and pretty much the same intuitive observation of the extracted mesh quality, comparing the time required during training and the number of Gaussians. However, since we will perform Prune and Densify operations on the Gaussian during the training process, which leads to a large variation in the number of Gaussians, in our experiments, the number of Gaussians in the outdoor scene fluctuates in the range of 10kt̃00K during the training process. Combining the above reasons, in this work we only compare the time needed for training. We plan to introduce in the future a metric that can measure the completeness of these Gaussians, which intuitively is a measure of the ability of Gaussain to represent the space. Since in our pipeline, the density of points in space can be viewed as a linear combination of nearby Gaussians Basis, and the high frequency details in the scene can be viewed as points in space with high gradients, if we can correctly fit the gradients of linear combinations of Gaussians to the high frequency points in the scene, then we can significantly improve the quality of the image, which is reflected in the PSNR and so on. This is reflected in metrics such as PSNR.

In addition, due to performance constraints, we only tested on the Mip-NeRF360 dataset as well as our own constructed dataset. However, the Mip-NeRF360 dataset is not divided into training and testing sets, and does not provide the ground truth of the depth and normal maps, so we cannot compare more metrics. If we were able to compare the depth information, we would be able to compare the quality of our extracted meshes more intuitively. In future work, we consider doing tests on more datasets so that we can focus on comparing the quality of the grids and find more targets that can be optimized.

8 Acknowledgements

I am sincerely grateful to my institution for providing me with a valuable learning and communication platform. I also extend my appreciation to every instructor who guided me through my undergraduate subjects, laying a strong foundation for my further education.

I would like to express my gratitude to my supervisor, Dr. Yizun Lin, for his patient guidance and invaluable support throughout the writing of my undergraduate thesis. Despite our inability to meet in person at times, he made the effort to organize discussions to review my research progress and experimental results, providing valuable feedback on areas for improvement. His academic insights and encouragement have played a crucial role in shaping my research and academic development.

Additionally, I am profoundly grateful to my family and friends for their constant encouragement and support throughout my life's journey, as their unwavering belief in me has been my source of strength, helping me overcome challenges. I am especially thankful to my four college roommates for their support and guidance in both life and studies. I also appreciate my classmates for their enriching discussions and suggestions, which have enhanced my research. The unwavering support and understanding from my family have served as a constant wellspring of strength and motivation.

Finally, I want to thank myself for my tenacious work. There is still much to be done, and I wish myself luck in maintaining my drive and putting in the effort to move closer to a promising future.

A Appendix

A.1 Derivation of Volume Rendering

We only prove the parts we use here, for a more detailed proof refer to this articles [18]. We consider the light absorption model: assuming uniformly distributed particles with density ρ, considering a ray in space with starting point $\mathbf{o}$, direction $\mathbf{d}$ of the form $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$, then the change in light intensity after passing through a cross-section of area A in space is:

$$\frac{dI}{ds} = -\rho(s)AI(s) = -\sigma(s)I(s), \tag{33}$$

where σ is called the extinction coefficient, also known as opacity, $I(s)$ is the light intensity at position s on the ray. This differential equation describes the process of light being absorbed by particles during propagation. After integrating it simultaneously, we get:

$$\int_0^s \frac{dI}{I(s)} = \int_0^t -\sigma(s)ds, \tag{34}$$

simplifying gives the solution to the differential equation:

$$I(t) = I_0 \exp(-\int_0^t \sigma(s)ds), \tag{35}$$

where I_0 is the initial light intensity. However, when light passes through particles in real world, it will also mix with the light emitted by the particles, so in reality, there will be an additional source term $g(s)$ at position s. The model at this time is:

$$\frac{dI}{ds} = g(s) - \sigma(s)I(s), \tag{36}$$

rearranging and multiplying by $\exp(\int_0^s \sigma(t)dt)$, we get:

$$\frac{d}{ds}\left(I(s)\exp\int_0^s \sigma(t)dt\right) + I(s)\sigma(s)\left(\exp\int_0^s \sigma(t)dt)\right) = g(s)\exp(\int_0^s \sigma(t)dt), \tag{37}$$

solving using the product rule gives:

$$I(t) = I_0 \exp(-\int_0^t \sigma(s)ds) + \int_0^t (g(s)\exp(-\int_s^t \sigma(\xi)d\xi))ds, \tag{38}$$

the first term on the right side $I_0 \exp(-\int_0^t \sigma(s)ds)$ is the background light, which is set to 0 in NeRF, the light intensity emitted at position s is $g(s) = \sigma(s)I(s)$, then we have:

$$I(t) = \int_0^t (\sigma(s)I(s)T(s)ds, \text{where } T(s) = \exp(-\int_0^t \sigma(\xi)d\xi), \tag{39}$$

we then replace the single-channel light intensity with a 3-channel color C to obtain the formula for Point-Based Volume rendering (7).

A.2 Derivation of Projective Transformation

This transformation is mainly used to transform Gaussians in camera space to ray space, as shown in Figure 9. Camera space is defined such that the origin of the camera coordinate system is at the center of projection, and the projection plane is defined as $t_2 = 1$. Ray space is defined as a space where a point is represented by a column vector of three coordinates $\mathbf{u} = (u_0, u_1, u_2)^T$. Given a center of projection and a projection plane, these three coordinates are interpreted geometrically as follows: The coordinates u_0 and u_1 specify a point on the projection plane. The ray intersecting the center of projection and the point (u_0, u_1) on the projection plane is called a viewing ray. Using the abbreviation $\hat{\mathbf{u}} = (u_0, u_1)^T$, we refer to the viewing ray passing through (u_0, u_1) as $\hat{u}$. The third coordinate u_2 specifies the Euclidean distance from the center of projection to a point on the viewing ray. To simplify the notation, we will use any of the synonyms $\mathbf{u}$, $(\hat{u}, u_2)^T$, or $(u_0, u_1, u_2)^T$ to denote a point in ray space [22]. Considering a point $\mathbf{t} = (t_0, t_1, t_2)$ in camera space, the projective transformation maps the point $\mathbf{t}$ in camera space to a point $\mathbf{u} = \mathbf{m}(\mathbf{t})$ in ray space:

$$\begin{pmatrix} u_0 \\ u_1 \\ u_2 \end{pmatrix} = \mathbf{m}(\mathbf{t}) = \begin{pmatrix} t_0/t_2 \\ t_1/t_2 \\ \|\mathbf{t}\| \end{pmatrix}, \tag{40}$$

however, this does not satisfy the definition of an affine transformation, so a local affine approximation $\mathbf{m}_{\mathbf{t}_k}$ is introduced:

$$\mathbf{m}_{\mathbf{t_k}}(\mathbf{t}) = \mathbf{u}_k + \mathbf{J}_{\mathbf{t}_k}(\mathbf{t} - \mathbf{t}_k). \tag{41}$$

This is the second-order Taylor expansion of $\mathbf{m}$ at $\mathbf{t}_k$, where $\mathbf{u}_k = \mathbf{m}(\mathbf{t}_k)$ is the center of Gaussians in ray space, and $\mathbf{J}_{\mathbf{t}_k}$ is a Jacobian matrix defined as the partial derivative of

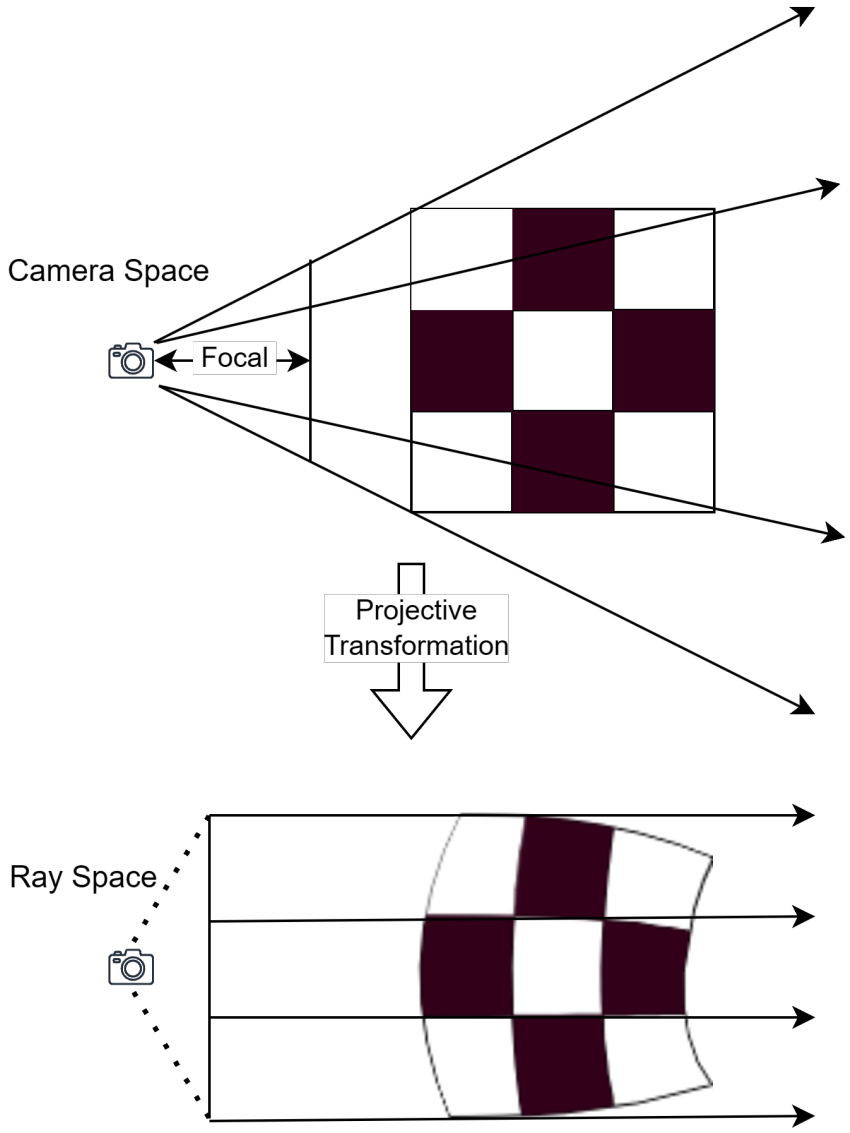

Figure 9: The upper part of the figure represents camera space, while the lower part represents ray space. We need to perform projective mapping based on the distance between the image in the upper part and the camera, as well as the focal length.

m at $\mathbf{t}_k$:

$$\mathbf{J}_{\mathbf{t}_k} = \frac{\partial \mathbf{m}}{\partial \mathbf{t}}(\mathbf{t}_k) = \begin{pmatrix} \frac{\partial u_0}{\partial t_0} & \frac{\partial u_0}{\partial t_1} & \frac{\partial u_0}{\partial t_2} \\ \frac{\partial u_1}{\partial t_0} & \frac{\partial u_1}{\partial t_1} & \frac{\partial u_1}{\partial t_2} \\ \frac{\partial u_2}{\partial t_0} & \frac{\partial u_2}{\partial t_1} & \frac{\partial u_2}{\partial t_2} \end{pmatrix} = \begin{pmatrix} 1/t_{k,2} & 0 & -t_{k,0}/t_{k,2}^2 \\ 0 & 1/t_{k,2} & -t_{k,1}/t_{k,2}^2 \\ \frac{t_{k,0}}{\|\mathbf{t}_k\|} & \frac{t_{k,1}}{\|\mathbf{t}_k\|} & \frac{t_{k,2}}{\|\mathbf{t}_k\|} \end{pmatrix} \tag{42}$$

Consider a point **x** in the world space. Applying an affine transformation to **x**, we can change its coordinate to the camera space: $\mathbf{t} = \mathbf{W}\mathbf{x} + \mathbf{b}$, where **W** and **b** are coefficients determined by the camera. Then, applying the projective mapping, we have $\mathbf{u} = \mathbf{m}(\mathbf{t}_k) \approx \mathbf{u}_k + \mathbf{J}_k(\mathbf{t} - \mathbf{t}_k)$. Combining these two transformations, we obtain:

$$\mathbf{u} = \mathbf{u}_k + \mathbf{J}_{\mathbf{t}_k}(W\mathbf{x} + b - \mathbf{t}_k). \tag{43}$$

We denote this equation (43) as $\mathbf{u} = \phi(\mathbf{x})$. The Gaussian function will be transformed into:

$$G(\mathbf{u}, \mu, \Sigma) = |\mathbf{J}_{t_k} W| G(\phi(\mathbf{x}), \phi(\mu), \Sigma''), \tag{44}$$

for a Gaussian in world space with covariance matrix Σ_k, its covariance matrix Σ_k'' in ray space is given by:

$$\Sigma_k'' = \mathbf{J}_{\mathbf{t}_k} \Sigma_k' \mathbf{J}_{\mathbf{t}_k}^T = \mathbf{J}_{\mathbf{t}_k} W \Sigma_k W^T \mathbf{J}_{\mathbf{t}_k}^T, \tag{45}$$

where Σ' is the covariance in camera space, and W is the matrix of parameters for the camera's viewpoint. Ignoring the subscript $\mathbf{t}_k$ gives the equation of (12).

References

[1] 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*, page 405 – 421. Jan 2020. doi: 10.1007/978-3-030-58452-8_24. URL http://dx.doi.org/10.1007/978-3-030-58452-8_24.

[2] Antoine Gu'edon and Vincent Lepetit. Sugar: Surface-aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh rendering. Nov 2023.

[3] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques - SIGGRAPH '87*, Jan 1987. doi: 10.1145/37401.37422. URL http://dx.doi.org/10.1145/37401.37422.

[4] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques - SIGGRAPH '96*, Jan 1996. doi: 10.1145/237170.237200. URL http://dx.doi.org/10.1145/237170.237200.

[5] Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, and Michael Cohen. Unstructured lumigraph rendering. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, Aug 2001. doi: 10.1145/383259.383309. URL http://dx.doi.org/10.1145/383259.383309.

[6] Noah Snavely, Steven M. Seitz, and Richard Szeliski. Photo tourism. *ACM Transactions on Graphics*, page 835–846, Jul 2006. doi: 10.1145/1141911.1141964. URL http://dx.doi.org/10.1145/1141911.1141964.

[7] John Flynn, Ivan Neulander, James Philbin, and Noah Snavely. Deepstereo: Learning to predict new views from the world's imagery. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2016. doi: 10.1109/cvpr.2016.595. URL http://dx.doi.org/10.1109/cvpr.2016.595.

[8] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering.

ACM Transactions on Graphics, page 1 – 15, Dec 2018. doi: 10.1145/3272127.3275084. URL http://dx.doi.org/10.1145/3272127.3275084.

[9] Eric Penner and Li Zhang. Soft 3d reconstruction for view synthesis. *ACM Transactions on Graphics*, page 1–11, Dec 2017. doi: 10.1145/3130800.3130855. URL http://dx.doi.org/10.1145/3130800.3130855.

[10] 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. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct 2021. doi: 10.1109/iccv48922.2021.00580. URL http://dx.doi.org/10.1109/iccv48922.2021.00580.

[11] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics*, page 1 – 15, Jul 2022. doi: 10.1145/3528223.3530127. URL http://dx.doi.org/10.1145/3528223.3530127.

[12] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2022. doi: 10.1109/cvpr52688.2022.00538. URL http://dx.doi.org/10.1109/cvpr52688.2022.00538.

[13] Sara Fridovich-Keil, Giacomo Meanti, Frederik Warburg, Benjamin Recht, and Angjoo Kanazawa. K-planes: Explicit radiance fields in space, time, and appearance. Jan 2023.

[14] Guikun Chen and Wenguan Wang. A survey on 3d gaussian splatting.

[15] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. *Symposium on Geometry Processing,Symposium on Geometry Processing*, Jun 2006.

[16] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. Aug 2023.

[17] James T. Kajiya and Brian P. Von Herzen. Ray tracing volume densities. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques - SIGGRAPH ' 84*, Jan 1984. doi: 10.1145/800031.808594. URL http://dx.doi.org/10.1145/800031.808594.

[18] N. Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, page 99–108, Jun 1995. doi: 10.1109/2945.468400. URL http://dx.doi.org/10.1109/2945.468400.

[19] Thomas Porter and Tom Duff. Compositing digital images. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques - SIGGRAPH ' 84*, Jan 1984. doi: 10.1145/800031.808606. URL http://dx.doi.org/10.1145/800031.808606.

[20] Georgios Kopanas, Thomas Leimkühler, Gilles Rainer, Clément Jambon, and George Drettakis. Neural point catacaustics for novel-view synthesis of reflections. *ACM Transactions on Graphics*, page 1 – 15, Dec 2022. doi: 10.1145/3550454.3555497. URL http://dx.doi.org/10.1145/3550454.3555497.

[21] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, FredA. Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. *arXiv: Machine Learning,arXiv: Machine Learning*, Jun 2018.

[22] Matthias Zwicker, Hanspeter Pfister, Jeroenvan Baar, and Markus Gross. Ewa volume splatting. *IEEE Visualization,IEEE Visualization*, Oct 2001.

[23] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2022. doi: 10.1109/cvpr52688.2022.00542. URL http://dx.doi.org/10.1109/cvpr52688.2022.00542.

[24] Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures.

[25] Marie-Julie Rakotosaona, Fabian Manhardt, DiegoMartin Arroyo, Michael Niemeyer, Abhijit Kundu, and Federico Tombari. Nerfmeshing: Distilling neural radiance fields into geometrically-accurate 3d meshes. Mar 2023.

[26] Lior Yariv, Peter Hedman, Christian Reiser, Dor Verbin, PratulP. Srinivasan, Richard Szeliski, JonathanT. Barron, and Ben Mildenhall. Bakedsdf: Meshing neural sdfs for real-time view synthesis. Feb 2023.

[27] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun 2018. doi: 10.1109/cvpr.2018.00068. URL http://dx.doi.org/10.1109/cvpr.2018.00068.

[28] Zhaoshuo Li, Thomas Müller, Alex Evans, RussellH Taylor, Mathias Unberath, Ming-Yu Liu, and Chen-Hsuan Lin. Neuralangelo: High-fidelity neural surface reconstruction.

[29] Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. Volume rendering of neural implicit surfaces. *Neural Information Processing Systems,Neural Information Processing Systems*, Dec 2021.