Gaus Fusion

Tal Shwartz ID: 318795242
talshwartz1@mail.tau.ac.il
Yaniv Korobkin ID: 207329525
korobkin@mail.tau.ac.il

1 ABSTRACT

The project aims to enhance text-guided 3D shape generation by improving upon the Latent-NeRF framework. The current system utilizes the NeRF (Neural Radiance Fields) architecture with score distillation in latent to generate 3D shapes and textures. While effective, there is potential for improvement in generation quality, geometric fidelity, and convergence speed.

To address these challenges, our proposed method introduces two key modifications: replacing the NeRF architecture with Gaussian Splatting, a recent alternative that has shown promise in novel view synthesis, to leverage its advantages for potentially better shape generation outcomes; and implementing NSFD (Neural Score Field Distillation) instead of the vanilla SDS (Score Distillation Sampling) method, as NSFD has demonstrated better practical performance in similar applications.

Our experiments demonstrate significant improvements across multiple metrics. The Gaussian Splatting approach produced more detailed objects compared to the baseline Latent-NeRF, yielding higher PSNR and SSIM scores while maintaining comparable generation times. Meanwhile, the NSFD integration resulted in clearer, sharper scenes with greater multi-view consistency and reduced visual artifacts. Our combined approach generates more comprehensive viewing angles, resulting in more detailed 3D structures, though memory constraints currently limit the maximum number of gaussians that can be generated.

2 INTRODUCTION

The project focuses on improving text-guided 3D shape generation, specifically enhancing the Latent-NeRF framework. While recent advances in text-to-image generation have been remarkable, generating high-quality 3D content from text descriptions remains a significant challenge. The ability to create accurate 3D shapes and textures from text has vast applications in computer graphics, virtual reality, game development, and digital content creation. Automating 3D model generation through text inputs could drastically reduce the time and expertise required for manual 3D modeling, making content creation more accessible to artists, designers, and developers.

This project is crucial for several reasons. Current text-to-3D generation methods, while impressive, often struggle with geometric accuracy and generation speed. Enhancing 3D content generation could democratize 3D modeling, allowing non-experts to create complex 3D assets using natural language.

To address these challenges, our approach introduces two key innovations. First, we replace the traditional NeRF architecture with Gaussian Splatting, a recent technique that has shown promising results in novel view synthesis. Second, we implement Neural Score Field Distillation (NSFD) instead of the vanilla Score Distillation Sampling (SDS) method to achieve improved performance. These modifications aim to enhance generation quality, geometric fidelity, and convergence speed, ultimately advancing the state of text-to-3D synthesis.

Our experimental results demonstrate several improvements over the baseline Latent-NeRF approach. The Gaus Fusion implementation produced more detailed objects. Quantitatively, our method achieved higher PSNR and SSIM scores, indicating superior rendering quality. Additionally, our NSFD integration resulted in clearer and sharper scenes with better multi-view consistency and reduced artifacts. Notably, our approach generates a wider range of viewing angles, resulting in more comprehensive 3D structures.

3 RELATED WORK

Text-to-3D generation has seen remarkable progress in recent years, with several key approaches emerging. The foundational work on Neural Radiance Fields (NeRF) by Mildenhall et al. (2020) [1] revolutionized novel view synthesis by introducing implicit neural representations for 3D scenes. Based on this, Dream Fields (2021) [2] and DreamFusion (2022) [3] demonstrated the possibility of generating 3D content from text using score distillation, paving the way for more advanced text-to-3D synthesis techniques.

A significant advancement came with the Latent-NeRF framework, presented at CVPR 2023 [4], which moved the generation process into a compact latent space of a pretrained autoencoder. This approach addressed the computational inefficiencies of previous methods that operated directly in image space, making text-guided 3D generation more efficient. Additionally, Latent-NeRF introduced an innovative shape guidance technique called ”SketchShape,” allowing for more controlled and precise 3D shape generation.

Concurrent with these developments, Gaussian Splatting emerged in 2023 [5] as a promising alternative to NeRF for novel view synthesis. While both approaches aim to represent 3D scenes, Gaussian Splatting offers potential advantages in terms of rendering efficiency and geometric representation. Its ability to efficiently model complex scenes makes it a strong candidate for integration into text-to-3D frameworks.

Our approach differs from previous work in several key aspects. While Latent-NeRF relies on the traditional NeRF architecture, we propose integrating Gaussian Splatting into the latent framework to leverage its rendering and geometric benefits. Additionally, instead of using the standard Score Distillation Sampling (SDS) method, we adopt Noise-Free Score Distillation (NSFD) [6], building upon recent findings that suggest improved performance with this modified distillation approach. Finally, our work represents one of the first attempts to combine Gaussian Splatting with latent space optimization for text-guided 3D generation, further distinguishing it from prior methods.

The most closely related work to our approach is the original Latent-NeRF paper [4]; however, our method explores a fundamentally different architectural choice while maintaining the benefits of latent space optimization. Moreover, our use of NSFD [6] sets us apart from previous text-to-3D methods that relied on standard score distillation techniques, offering a novel direction for improving generation quality and efficiency in this field.

4 Data

To evaluate our approach, we used two datasets: MipNeRF 360 and Tanks and Temples.

MipNeRF 360 is a dataset featuring nine scenes (Bicycle, Garden, Stump, Room, Counter, Kitchen, Bonsai, Flowers, and Treehill) captured with handheld cameras, providing full 360° coverage. Each scene contains 60–150 high-resolution images showcasing complex lighting and reflective surfaces.

Tanks and Temples comprises 14 scenes, created using professional camera arrays and industrial scanning equipment to provide precise camera calibration and ground truth geometry. Each scene contains 150–300 images. Originally designed for evaluating multi-view stereo reconstruction methods, Tanks and Temples has since become a standard benchmark for novel view synthesis.

When processing our dataset for Gaussian Splatting, we ran COLMAP to extract camera parameters such as focal length, position, and orientation. Additionally, COLMAP undistorted the images to conform to ideal pinhole camera models.

5 Methods

First we want to explain briefly about NeRF.

5.1 NeRF

At its core, NeRF converts a collection of 2D images into a continuous neural representation of a 3D scene, enabling the generation of photorealistic novel viewpoints with remarkable fidelity.

The fundamental innovation of NeRF lies in its representation of a scene as a continuous 5D function implemented through a neural network. For any given 3D spatial location (x, y, z) and viewing direction (θ, ϕ), this network outputs both a volume density value and an RGB color. The volume density represents the presence of scene content at that location, while the RGB color captures what that point looks like when viewed from a particular direction. This dual output allows NeRF to model both the underlying geometry and the view-dependent appearance effects like specular reflections.

Rendering an image with NeRF involves a process based on volume rendering principles. For each pixel in the desired view, a ray is cast from the camera through that pixel into the scene. The algorithm samples points along this ray, feeds the coordinates into the neural network, and accumulates the resulting colors and densities to produce the final pixel color. This process is differentiable, enabling end-to-end training by comparing rendered views with ground truth images.

5.2 Latent NeRF

Our proposed method suggests to replace the NeRF part in the latent NeRF pipeline. Latent-NeRF is an innovative approach to text-guided 3D shape generation that operates in the latent space of a pre-trained Latent Diffusion Model (LDM) like Stable Diffusion.

Diffusion models operate on the principle of gradually adding noise to data and then learning to reverse this process. Mathematically, the forward diffusion process can be described as:

$$x_t = \sqrt{\alpha_t}x_0 + \sqrt{1-\alpha_t}\epsilon \tag{1}$$

where x_0 is the original data, x_t is the noisy version at timestep t, α_t is a noise schedule parameter, and $\epsilon \sim N(0, I)$ is random Gaussian noise.

Latent Diffusion Models (LDMs) like Stable Diffusion operate not directly on pixels but in a compressed latent space. First, an autoencoder is trained:

$$z = E(x),\ \widehat{x} = D(z) \tag{2}$$

Where E is the encoder, D is the decoder, and z is the latent representation. The diffusion process then happens in this latent space, which is more computationally efficient.

While traditional Neural Radiance Fields (NeRF) operate in RGB space, Latent-NeRF works directly in the compressed latent space of a pre-trained autoencoder. Instead of generating RGB colors at each 3D coordinate, it generates latent features that can be decoded into RGB later.

Like standard NeRF, Latent-NeRF represents a 3D scene using a neural network (MLP) that takes 3D coordinates (x, y, z) and viewing directions as input:

$$F_\theta\ : (x,\ y,\ z,\ d) \to (c_1,\ c_2,\ c_3, c_4, \sigma) \tag{3}$$

However, unlike standard NeRF which outputs RGB values, Latent-NeRF outputs four latent feature channels (c_1, c_2, c_3, c_4) corresponding to the latent space of Stable Diffusion, along with a volume density (σ). During training, the model renders 2D feature maps in the latent space by accumulating these latent features and densities along rays for each pixel:

$$C(r) = \sum_{i=1}^{N} T_i \alpha_i c_i \tag{4}$$

Where $T_i = \prod_{j=1}^{i-1}\ (1 - \alpha_j)$ and $\alpha_i = 1 - e^{-\sigma_i \delta_i}$.

The core innovation of Latent-NeRF lies in its adaptation of score distillation from latent diffusion models to guide 3D generation. During each training iteration, the system renders a latent feature map from the neural radiance field, then adds random noise to this latent image corresponding to a randomly chosen diffusion timestep:

$$z_t = \sqrt{\overline{\alpha_t}} + \sqrt{1 - \overline{\alpha_t}}\epsilon \tag{5}$$

This noised latent image is then fed into Stable Diffusion's denoiser network ϕ, which—conditioned on the target text prompt—attempts to predict the noise that was added:

$$\widehat{\epsilon} = \phi\left(z_t, t, T\right) \tag{6}$$

where T is the text prompt. The crucial insight comes in using the difference between the predicted noise and the actual added noise to derive gradient signals, which are backpropagated through the rendering process to update the Latent-NeRF's parameters. The score distillation loss gradient is:

$$\nabla_z L_{SDS} = w(t)\left(\widehat{\epsilon} -\ \epsilon\right) \tag{7}$$

These gradients effectively guide the 3D model toward generating latent features that, when denoised according to the text prompt, produce coherent and semantically aligned content, thus translating 2D diffusion model capabilities into consistent 3D generation without requiring any 3D training data. The process discussed above can be seen in figure 1.

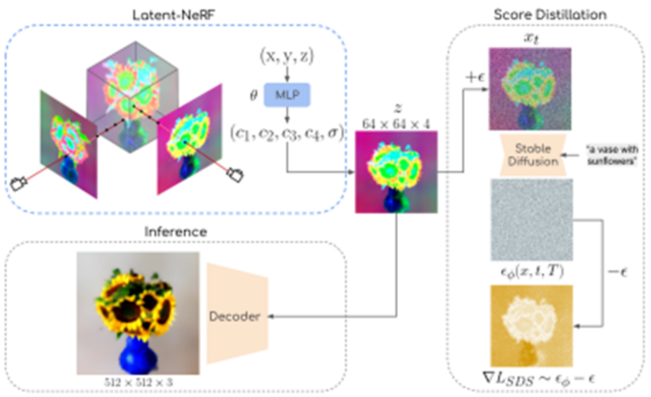

Fig. 1: At each training iteration we render the scene from a random view point to produce a feature map z. Then, z is noised with according to a random diffusion step t. The noised version of z, i.e., xt, is denoised using Stable Diffusion, with the input text prompt. Finally, the input noise is subtracted from the predicted noise by Stable Diffusion, to approximate per-pixel gradients that are back propagated to the NeRF representation.[4]

5.3 Gaussian Splatting

3D Gaussian Splatting begins with a collection of photos of a scene taken from different viewpoints. Using Structure-from-Motion (SfM), the system first calculates camera positions and generates a sparse point cloud of the scene. Instead of using these points merely as initialization for a neural network as in NeRF methods, Gaussian Splatting uses them as the foundation for explicit 3D primitives.

Each point is converted into a 3D Gaussian, which is mathematically described by:

$$G(x) = e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)} \tag{8}$$

where μ is the mean (position), and Σ is the covariance matrix that defines the shape and orientation of the Gaussian. These Gaussians are anisotropic, meaning they can stretch in different directions to efficiently represent surfaces.

The covariance matrix Σ is parameterized as:

$$\Sigma = RSS^T R^T \tag{9}$$

where R is a rotation matrix (represented by a quaternion during optimization) and S is a scaling matrix. This decomposition allows for more stable optimization.

The covariance matrix in camera coordinates is calculated as:

$$\Sigma = JW\Sigma W^T J^T \tag{10}$$

where W is the viewing transformation and J is the Jacobian of the affine approximation of the projective transformation.

During training, the system optimizes these Gaussians through a differentiable rendering process. For each training image, Gaussians are projected to 2D and ”splatted” onto the image plane. The term ”splatting” refers to how each 3D

Gaussian is converted to a 2D splat (an elliptical footprint) in screen space.

The rendering equation follows a volumetric rendering model similar to NeRF. The color C of a pixel is given by:

$$C = \sum_{i=1}^{N} T_i \alpha_i c_i \tag{11}$$

with transmittance $T_i = \prod_{j=1}^{i-1} (1-\alpha_j)$ and opacity $\alpha_i = \left(1-e^{-\sigma_i \delta_i}\right)$, where σ_i is the density and δ_i is the distance between samples.

The color of each Gaussian is represented using spherical harmonics to capture view-dependent effects:

$$c_i = \sum_{l=0}^{L} \sum_{m=-l}^{l} k_l^m Y_l^m(\theta, \phi) \tag{12}$$

where Y_l^m are spherical harmonic basis functions and k_l^m are the learnable coefficients.

During training, the system optimizes these Gaussians through a differentiable rendering process. The loss function combines L1 and a perceptual D-SSIM term:

$$L = (1-\lambda)L_1 + \lambda L_{D-SSIM} \tag{13}$$

What makes this approach particularly effective is its adaptive density control. As optimization progresses, the system strategically adds new Gaussians by either cloning them in under-reconstructed areas or splitting larger ones in areas needing finer detail. Simultaneously, it removes unnecessary Gaussians that have become transparent, maintaining efficiency.

The rendering pipeline is highly optimized for GPUs. It splits the screen into tiles, culls Gaussians against the view frustum, sorts them by depth, and then renders them with a specialized rasterizer. This design enables real-time performance, achieving hundreds of frames per second on modern GPUs—dramatically faster than NeRF approaches.

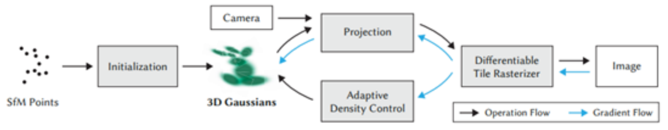

Fig. 2: Optimization starts with the sparse SfM point cloud and creates a set of 3D Gaussians. We then optimize and adaptively control the density of this set of Gaussians. During optimization we use our fast tile-based renderer. Once trained, the renderer allows real-time navigation for a wide variety of scenes.[5]

After optimization, the scene is represented by millions of optimized 3D Gaussians that can be rendered extremely efficiently, enabling real-time navigation through complex scenes with high visual fidelity. The process described above can be seen in figure 2.

In addition, there is also an adaptive density control, where we periodically add new gaussians to the 3D model in order to make the 3D model get more details and a more complex shape. In addition, gaussians that are essentially transparent are removed from the model for making the process more efficient.

The adaptive control of the Gaussians needs to populate empty areas. It focuses on regions with missing geometric features ("underreconstruction"), but also in regions where Gaussians cover large areas in the scene (which often correspond to "over-reconstruction").

For small Gaussians that are in under-reconstructed regions, we need to cover the new geometry that must be created. For this, it is preferable to clone the Gaussians, by simply creating a copy of the same size, and moving it in the direction of the positional gradient. On the other hand, large Gaussians in regions with high variance need to be split into smaller Gaussians. We replace such Gaussians by two new ones, and divide their scale by a factor determined experimentally. In the first case we detect and treat the need for increasing both the total volume of the system and the number of Gaussians, while in the second case we conserve total volume but increase the number of Gaussians.

Gaussians may shrink or grow and considerably overlap with others, but we periodically remove Gaussians that are very large in worldspace and those that have a big footprint in viewspace. This strategy results in overall good control over the total number of Gaussians.

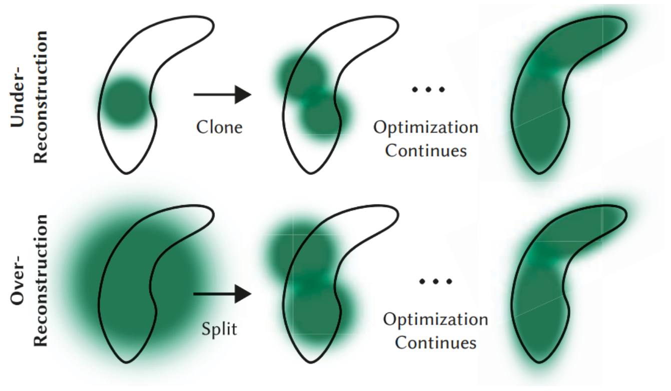

Fig. 3: Adaptive Gaussian densification scheme. Top row (underreconstruction): When small-scale geometry (black outline) is insufficiently covered, we clone the respective Gaussian. Bottom row (over-reconstruction): If small-scale geometry is represented by one large splat, we split it in two.

5.4 NFSD

Noise-Free Score Distillation (NFSD) enhances text-to-3D generation by isolating and removing unwanted noise components from the diffusion model guidance process. While the standard Score Distillation Sampling (SDS) method incorporates all components of the score function, including noise, NFSD focuses only on the useful signals.

The key insight is decomposing the score function into three components:

$$\epsilon_\phi^s(z_t; y, t) = \delta_D + \delta_N + s\delta_C \tag{14}$$

Where δ_D guides toward realistic images (domain correction), δ_C aligns with the text prompt (condition direction), and δ_N predicts noise. Traditional SDS uses all three in its gradient calculation:

$$\nabla_\theta L_{SDS} = w(t)\left(\delta_D + \delta_N + s\delta_C - \epsilon\right) \frac{\partial x}{\partial \theta} \tag{15}$$

This may lead to oversaturated results. NFSD eliminates the noise component by approximating δ_D based on the diffusion timestep:

$$\delta_D = \begin{cases} \epsilon_\phi\left(z_t; \varnothing, t\right), & if\ t < 200 \\ \epsilon_\phi\left(z_t; \varnothing, t\right) - \epsilon_\phi\left(z_t; y = p_{neg}, t\right), & otherwise \end{cases} \tag{16}$$

The resulting cleaner gradient focuses only on meaningful components:

$$\nabla_\theta L_{NFSD} = w(t)\left(\delta_D + s\delta_C\right) \frac{\partial x}{\partial \theta} \tag{17}$$

This approach allows NFSD to produce higher-quality results with more detailed features and natural colors.

5.5 Our Method

We propose replacing the neural radiance field with 3D Gaussian Splatting. Rather than using an MLP that outputs latent features, we would utilize explicit 3D Gaussians with feature attributes. Each Gaussian is characterized by its position, covariance matrix (scale), rotation, opacity, brightness, and color.

The rendering equation maintains a similar structure but leverages Gaussian splatting's more efficient rendering process. To enhance generation quality, we implement Noise-Free Score Distillation (NFSD) instead of traditional Score Distillation Sampling (SDS). While SDS employs gradients containing unwanted noise components, NFSD eliminates these noise terms by approximating only the domain correction and condition components.

We start by initializing the 3D points using Structure-from-Motion (SfM). During our initial attempts, the point cloud initialization resulted in a random distribution of points—each with arbitrary positions, brightness, and color. These attempts failed because the model struggled to recognize any coherent 3D shape, leading to poor outcomes. Through experimentation, we found that creating a dense point cloud at the origin helped the model recognize the presence of a 3D object. This initial step allowed the model to refine the cloud and gradually shape it to align with the description in the text prompt, resulting in more accurate and meaningful 3D structures.

Following initialization, we proceed with iterative rendering and optimization to evaluate how closely the generated image matches the text prompt. Due to the inherent ambiguities in the 3D-to-2D projection, it is inevitable that some geometry may be incorrectly placed. As a result, the optimization process must be capable of both generating new geometry and correcting or removing existing geometry that has been mispositioned. The quality of the covariance parameters of the 3D Gaussians plays a pivotal role in the compactness of the representation, as large homogeneous regions can be effectively captured using fewer, larger anisotropic Gaussians.

As previously described, the rendering process is integral to the model, with the rendered output being fed into the diffusion model (in our case, the Stochastic Diffusion Sampler or SDS, utilizing Stable Diffusion as the critic). The gradients, which are computed similarly to Latent-NeRF, are not passed to the parameters of the NeRF architecture. Instead, they are used to update the parameters of the individual Gaussians.

We've also incorporated the camera's viewing direction into the text prompt. This is achieved through a simple script that generates a prompt based on the camera's orientation. The available options for the viewing direction include: 'front', 'left side', 'back', 'right side', 'overhead', and 'bottom'. This directional information is then appended to the overall text prompt, making the resulting SDS (Scene Description) dependent on the camera's perspective.

Additionally, we incorporate adaptive density control during the optimization process. This mechanism periodically introduces new Gaussians into the 3D model, which allows for the gradual enhancement of the model's detail and the refinement of its shape. Concurrently, Gaussians that are effectively transparent or contribute little to the model's geometry are removed to improve computational efficiency.

The adaptive control of the Gaussians primarily focuses on filling in regions that are under-reconstructed or lacking sufficient geometric detail. This is accomplished by strategically adding Gaussians to these empty spaces. At the same time, the system addresses areas with excessive Gaussian coverage, often corresponding to over-reconstructed regions, where large, densely packed Gaussians have already captured sufficient detail.

For small Gaussians located in under-reconstructed areas, the approach is to duplicate them. We create a copy of each Gaussian of the same size and move it in the direction of the positional gradient, thereby expanding the coverage and filling the missing geometric details. Conversely, for large Gaussians in regions with high variance, we split them into smaller Gaussians. This division is performed by replacing the original large Gaussian with two new, smaller ones, and scaling down their size by a factor determined through experimental tuning. This approach ensures that we balance the total volume of the system while increasing the number of Gaussians, which improves the model's ability to capture finer details.

Throughout the training process, the Gaussians can expand, contract, and overlap with one another. To maintain efficient computation and model stability, we periodically remove Gaussians that are disproportionately large in world space or have a significant footprint in view space. This helps regulate the total number of Gaussians in the system, ensuring that the model maintains an optimal and manageable complexity. Overall, these strategies contribute to a more controlled, efficient, and accurate reconstruction of 3D geometry in alignment with the given text prompt.

By iterating through these steps, the system progressively refines the 3D model, optimizing both the number and configuration of Gaussians to ensure that the resulting 3D scene is as detailed, accurate, and visually aligned with the prompt as possible.

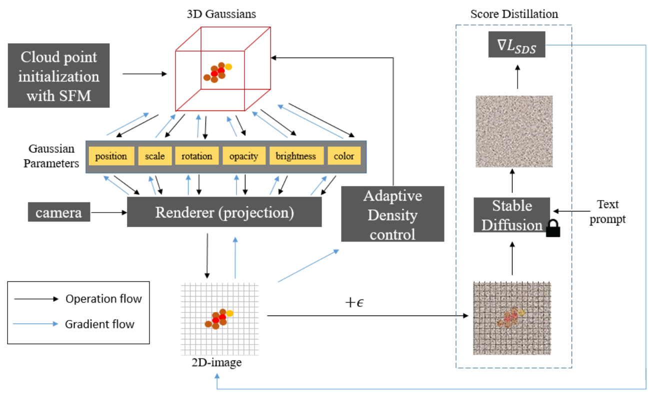

Fig. 4: Pipeline of our model. A text prompt is first encoded and processed through a latent diffusion model. The 3D scene is represented as a collection of 3D Gaussians defined by position, scale, rotation, and color parameters. The renderer projects these Gaussians to create multi-view images, which are compared against the latent diffusion model's expectations to generate optimization signals. This feedback loop continues until the Gaussian parameters converge to represent a 3D scene matching the text description

This approach combines Gaussian Splatting's efficiency and real-time performance with Latent-NeRF's text-guided generative capabilities, potentially enabling faster training, more detailed geometry, and real-time rendering of text-to-3D generated content. This process can be viewed in figure 3.

6 Experiments

We explored both the "Latent NeRF" model and the "Gaussian Splating" model in order to combine them and optimize the results. We tried different number of iterations on the basic "Latent NeRF" model and our "Gaus Fusion".

In different set of experiments we also replaced the SDS with the NFSD using only the basic "Latent NeRF" model and compared them. In this case they both ran for the same iterations and for the same hyperparameters.

6.1 Training

We trained a conditional diffusion model for 3D Gaussian Splatting using Stable Diffusion. Our implementation builds upon the CompVis/stable-diffusion-v1-4 checkpoint as the foundation. The optimization process runs for 5,000 iterations in the NFSD experiment and for a varying number of iterations in Gaus Fusion. We used the AdamW optimizer with a learning rate of 3e-4 and mixed precision training for computational efficiency.

To initiate the Gaussian model, we first applied Structure-from-Motion (SfM) to one of the nine scenes from the MipNeRF360 dataset. However, to improve flexibility and enable the generation of any 3D model, we then randomize the placement of these points, along with parameters such as opacity, color, brightness, and scaling. Afterward, we remove any Gaussians that are positioned too far from the scene's origin. This step is essential, as distant Gaussians introduce unwanted haze and blur, which negatively affect the training process. Through experimentation, we found that clustering many Gaussians near the scene's center helps the model gradually form a 3D shape during training. Over time, the diffusion model refines this shape, aligning it with the target description from the text prompt.

We observed that the initialization plays a crucial role in the flexibility of generating any desired 3D model, particularly from all viewing angles. To address this, we focused on identifying objects that are symmetrical and cylindrical, yet exhibit random surface variations. After extensive experimentation, the point cloud shown in figure 5 delivered the best results in terms of flexibility and adaptability.

Fig. 5: Flexible point cloud found for generating any 3D model

Additionally, we found that randomizing the camera's position and angle during each generation slowed down the training process and sometimes led to poor results. As a result, the most effective approach was to have the camera rotate 360 degrees around the object, providing more consistent and reliable outcomes.

The Gaussian Splatting representation uses spherical harmonics with degree 3 for appearance modeling. Densification occurs every 500 iterations beginning from iteration 1 until the end of training, with new points added based on a gradient threshold of 0.0002. To prevent premature convergence, opacity values are reset every 50 iterations, ensuring continued refinement of the 3D structure.

We implemented parameter-specific learning rates to optimize different aspects of the model. Position parameters start at 1.6e-4 and decay to 1.6e-6, while brightness, opacity, scaling, and rotation parameters maintain constant rates of 2.5e-3, 5e-3, 4e-3, and 5e-2 respectively. The learning rate of the color is equal to the learning rate of the brightness divided by 40. This tailored approach allows for precise control over the development of each attribute.

Additionally, through experimentation, we found that using multiple learning rates simultaneously created challenges during model training. To address this, we set most of the learning rates to zero, allowing the model to focus on only a few active ones. Every 100 iterations, we gradually activated other learning rates. In our approach, the learning rate for rotation remained nonzero throughout the entire training, while the learning rates for opacity, scale, brightness, color, and position were progressively set to nonzero values in that order, with all other parameters kept at zero.

In our NFSD implementation, we apply NFSD with a weight of 1.0 and a noise scale of 0.1, working in conjunction with the original SDS approach.

For text conditioning, the model accepts customizable prompts with optional directional conditioning and support for Textual Inversion concepts, offering flexibility in specifying the desired output. Comprehensive evaluation is performed by rendering multiple viewpoints, ensuring consistent 3D representation from all angles.

For initializing the 3D points we used Structure-from-Motion (SfM) applied to one of the nine scenes from the MipNeRF360 dataset. However, to enhance flexibility and enable the generation of any 3D model, we subsequently randomize the placement of these points, along with various parameters such as opacity, color, brightness, and scaling. Afterward, we remove the Gaussians that are situated far from the scene's origin. This step is crucial, as these distant Gaussians introduce undesirable haze and blur, which negatively impact the training process.

6.2 Metrics

To evaluate the quality of our generated 3D representations, we employ two widely used image quality metrics: Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM). PSNR measures the pixel-level fidelity between a generated image and its reference by calculating the ratio between the maximum possible power of the signal and the power of corrupting noise.

It is defined logarithmically (in decibels) as:

$$PSNR = 10 * \log_{10} \frac{MAX^2}{MSE} \tag{18}$$

Where MAX is the maximum possible pixel value and MSE is the Mean Squared Error between the generated and reference images.

While PSNR is computationally efficient and provides a straightforward assessment of reconstruction accuracy, it does not always correlate well with human perception of visual quality, especially when comparing images with different types of distortions.

To complement PSNR's limitations, we also utilize the Structural Similarity Index (SSIM), which better aligns with human visual perception by evaluating the structural information in images. SSIM considers image degradation as perceived changes in structural information while incorporating important perceptual phenomena such as luminance masking and contrast masking.

The SSIM index is calculated as:

$$SSIM(x, y) = l(x, y)^{\alpha} * c(x, y)^{\beta} * s(x, y)^{\gamma} \tag{19}$$

Where $l(x, y)$, $c(x, y)$, and $s(x, y)$ are the luminance, contrast, and structural comparison functions between images x and y, and α, β, and γ are positive weights. SSIM ranges from -1 to 1, with 1 indicating perfect structural similarity.

7 Results

7.1 Gaus Fusion

First, we replaced the NeRF component in the ”Latent NeRF” model with the Gaussian splatting model. Then, we ran our ”Gaus Fusion” for varying numbers of iterations, experimenting with two different initialization methods. The first method involved creating a very dense point cloud at the center, while the second method focused on finding a point cloud that is optimally flexible for any text prompt and from every direction (as shown in Fig. 5). The results for the first and second methods are presented in Figures 6 and 7, respectively. Figures 8 and 9 highlight the differences between the results of Latent NeRF and the first method. Notably, in Figure 9, we can observe how hyperparameters influence the image results, demonstrating the level of accuracy required to achieve good outcomes. It is important to mention that, due to GPU RAM limitations, we had to restrict the number of Gaussians used in the experiments.

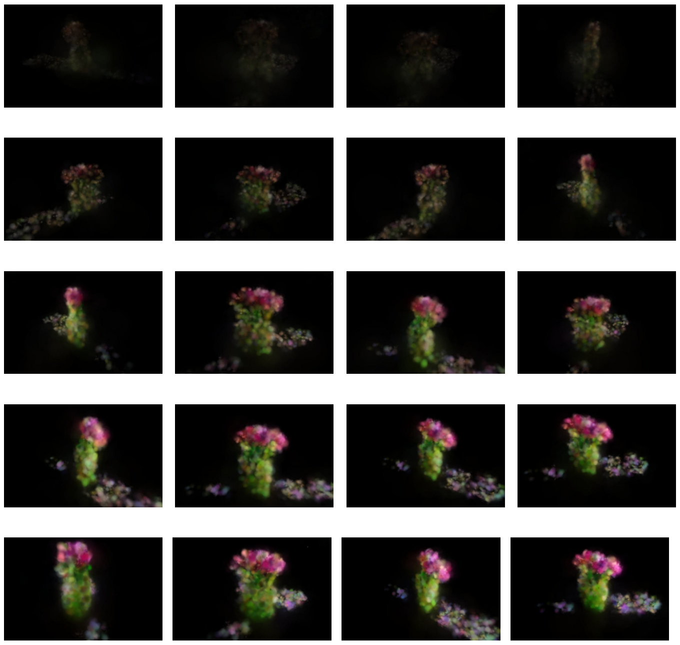

Fig. 6: Results for different numbers of iterations for ’vase with flowers’ 1st method: 1000,2000,3000,4000 and 5000

TABLE 1: Quantitative comparison of Latent NeRF vs. Gaus Fusion using figures 8 and 9

Scene	SSIM	PSNR (dB)
Bonsai	0.3131	12.78
Tractor	0.2021	7.44

We can see that as we increased the number of iterations, the results improved significantly: more objects appeared in the scene, and these objects displayed greater detail (evident in the leaves of both the stump and the flowers). We observe that as iterations increase, the number of gaussians also increases, allowing for better adjustment to the ground truth frame. We believe that without the limitation on the number of gaussians, we could achieve even more detailed and refined images.

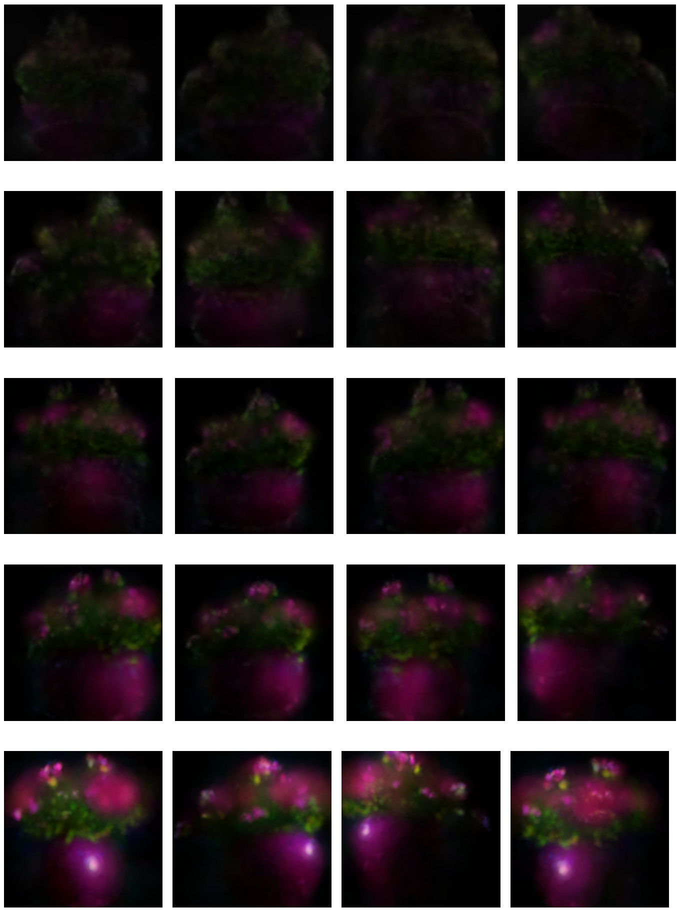

Fig. 7: Results for different numbers of iterations for ’vase with flowers’ 2nd method: 500,1000,1500,2000 and 3000

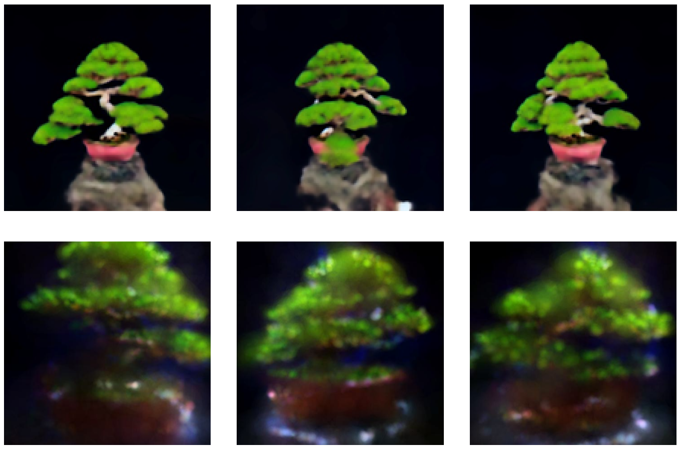

Fig. 8: Results for the basic ”Latent NeRF” (upper images) and our ”Gaus Fusion” 2nd method (bottom images) for an input of ’Bonsai’

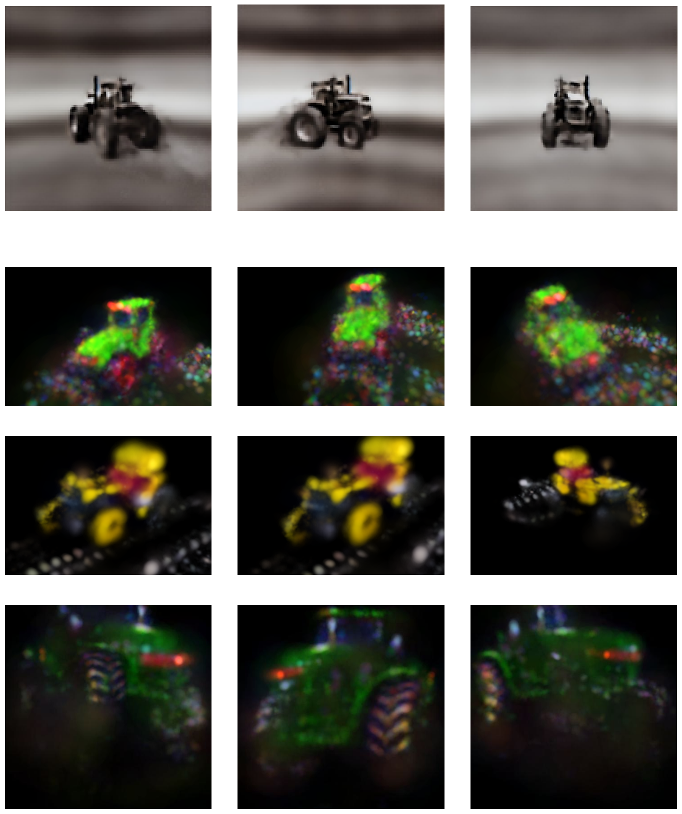

Fig. 9: Results for the basic ”Latent NeRF” (1st row) and our ”Latetnt Gaussian Splatting” (bottom images) for an input of ’Tractor’. The tractor in the the 2nd and 3rd rows were created with the 1st method but use different hyperparameters. This makes it evident that acquiring the correct parameters can make significant impact on the final model. The images at the bottom row were created by the 2nd method of Gaus-Fusion which yealded better results.

In comparison with the basic ”Latent NeRF” model, we can see that our model produces more detailed results (evident in the bonsai leaves and small objects in the tractor), as seen in Fig 6 and Fig 7. We should note that our model took 39 minutes to run, while the ”Latent NeRF” took only 26 minutes. However, the ”Latent NeRF” implementation uses optimized kernels leading to faster running time, which might explain this difference. Additionally, our approach achieved better results on both PSNR and SSIM metrics, as seen in Table 1.

In addition, we experimented with different image resolutions during rendering. We found that using smaller resolutions (e.g., 260x390 in Figure 6 or 230x230 in Figure 10) allowed us to achieve results in about 30 minutes, but with reduced accuracy. On the other hand, when working with higher resolutions (e.g., 460x460 in Figures 7 and 11), the results were significantly richer in quality and complexity, though the rendering time increased to around 1 hour. We believe there are two reasons for this. First, the version of Stable Diffusion we used was not the most recent, and newer versions could likely produce better results even at smaller resolutions. Second, we hypothesize that smaller images offer less space and impose more constraints on the SDS itself. As a result, working with larger images provides more freedom for the SDS to generate higher-quality results, albeit at the cost of longer runtime. Figures 10 and 11 produce different results depending on the size of the image.

Additionally, the 3D model of the tractor generated by the second method yielded decent results, but it took twice as long to produce compared to the bonsai model. We believe this discrepancy is due to the initial point cloud used in the second method, which was more spherical symmetric. This symmetry posed a greater challenge to the model, as it heavily depends on the viewing direction. Despite the longer processing time, the results were ultimately superior to those achieved by previous methods.

Overall, we observe that the Gaus Fusion is more effective at rendering detailed objects, while requiring approximately the same convergence time as the latent NeRF. It is also important to note that our model generates a wider range of viewing angles, resulting in a more comprehensive 3D structure.

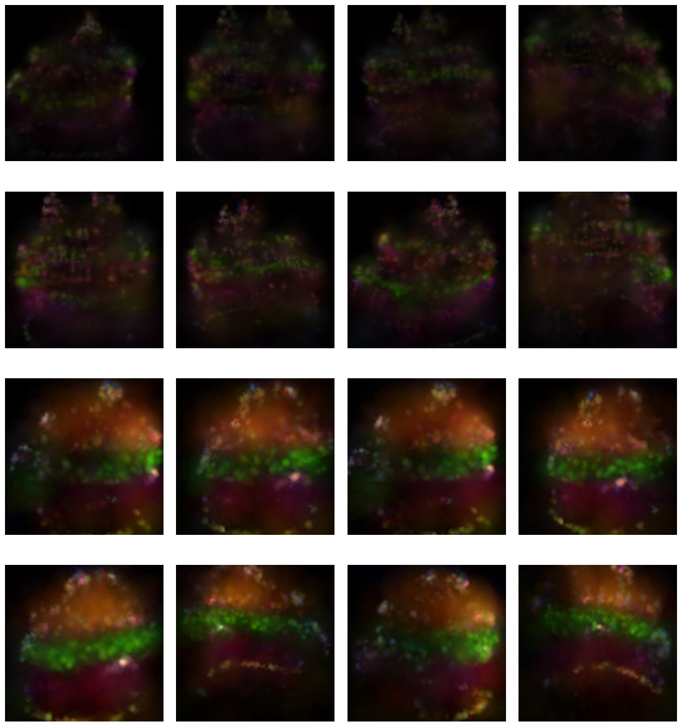

Fig. 10: Results for different numbers of iterations for ’burger’ with resolution of 230x230: 1000,2000,4000 and 6000

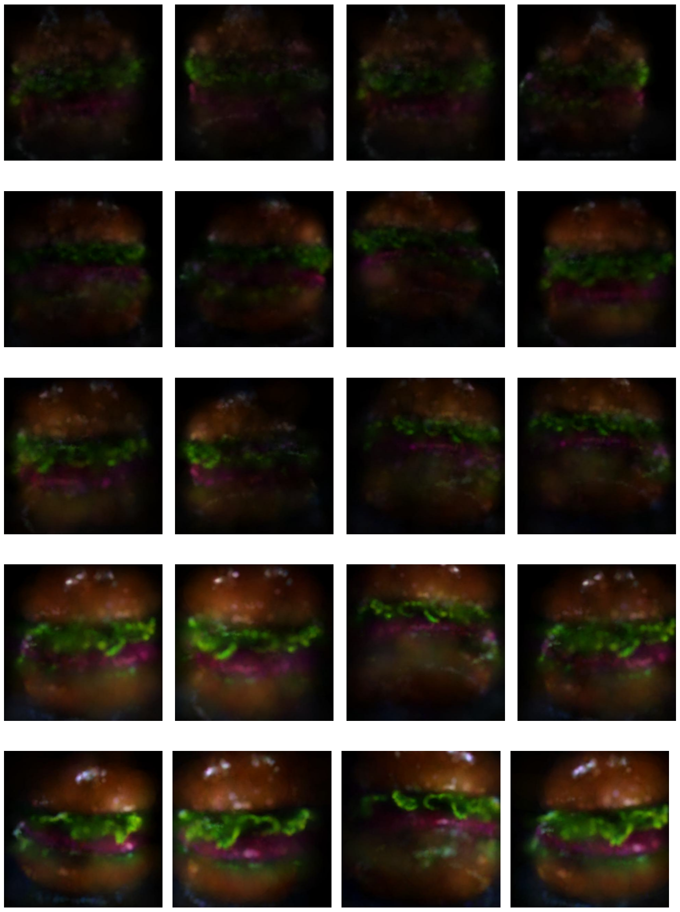
Fig. 11: Results for different numbers of iterations for 'burger' with resolution of 460x560: 1000,1500,2000, 3000 and 4500

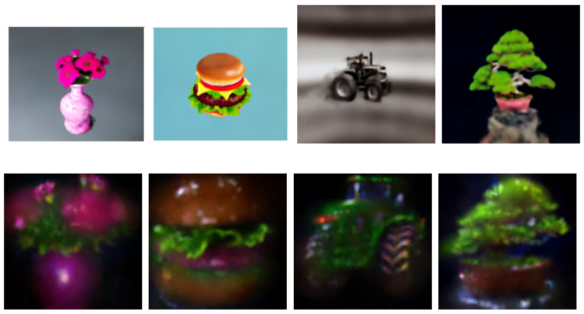
Fig. 12: Final comparison: Latent NeRF (first row) vs Gaus Fusion second method (second row)

7.2 NSFD Implementation

Second, we replaced the vanilla SDS with NSFD as discussed earlier. We conducted comparative experiments running Latent NeRF with vanilla SDS and Latent NeRF with NSFD. All models were trained with identical hyperparameters as outlined above, and for a consistent 5000 iterations. The results are presented in Fig 8,9 and 10.

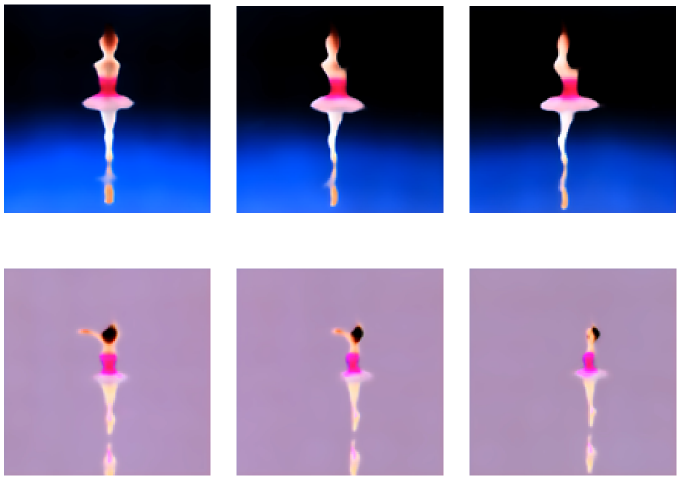
Fig. 13: Results for the SDS (upper images) and our NFSD (bottom images) for an input of 'Ballerina'

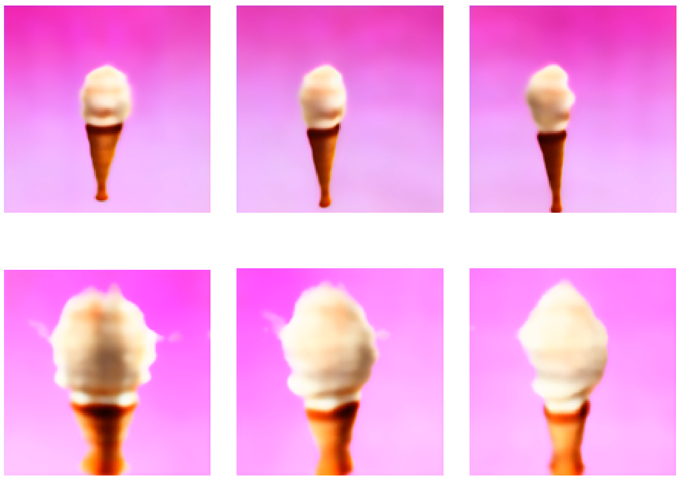
Fig. 14: Results for the SDS (upper images) and our NFSD (bottom images) for an input of 'Ice cream'

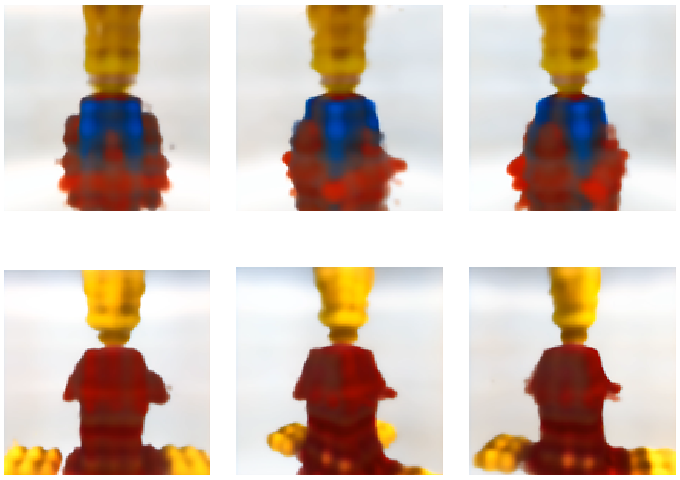

Fig. 15: Results for the SDS (upper images) and our NFSD (bottom images) for an input of 'Lego man'

TABLE 2: Quantitative comparison of SDS vs. NFSD

Scene	SSIM	PSNR (dB)
Ballerina	5.29	0.265
Ice cream	12.76	0.846
Lego man	12.89	0.7831

We can see that we have achieved better images, both in visual appearance and in the PSNR and SSIM values, as seen in Table 2. It is also important to note that the time required for each model to run was the same.

In the NSFD model, we observe that with the same convergence time, we obtained better results. The scene is clearer and sharper (as indicated by the PSNR and SSIM values), sometimes even displaying more details than the SDS.

8 Conclusion

Our investigation into the potential of replacing NeRF (Neural Radiance Fields) with Gaussian Splatting for text-to-3D generation has highlighted several significant advantages. Specifically, Gaus Fusion demonstrated substantial improvements in both the quality of generated 3D objects and the computational efficiency of the generation process. The first method of Gaus Fusion delivered decent results in 30 minutes, which is twice the time it took for Latent NeRF. However, this was not the full potential of Gaus Fusion in terms of quality, so it came with some trade-offs. The second method took about 1 hour to complete for most text prompts, which is roughly the same time as Latent NeRF, but the results were much more realistic and even surpassed those of Latent NeRF. This creates a trade-off between the two methods in terms of running time and output quality.

Moreover, the results obtained with Gaus-Fusion yielded higher Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM) scores, both of which are widely recognized metrics for assessing the quality of image reconstructions. These improved scores suggest that the generated 3D objects, when rendered from different viewpoints, exhibited superior rendering quality overall compared to the traditional NeRF approach. This indicates that Gaus Fusion has the potential to generate more detailed, accurate, and visually appealing 3D models while maintaining a reasonable generation time.

In addition to these performance improvements, the integration of the NSFD (Neural Surface Field Decomposition) technique with Latent NeRF proved to be particularly beneficial in enhancing training stability. This integration helped to generate more coherent 3D structures across different viewpoints, addressing one of the common issues encountered in previous text-to-3D generation methods: the inconsistency of geometry and texture when rendered from varying angles. Moreover, the adoption of NSFD significantly reduced the presence of visual artifacts, such as unnatural seams, ghosting, and texture distortions, which are frequently observed in the outputs of traditional 3D rendering pipelines. This improvement is crucial for achieving photorealistic 3D representations in applications ranging from virtual reality to digital twins.

Despite these considerable advancements, our work also identified several remaining limitations that need to be addressed for further optimization. One of the most prominent challenges lies in the generation of highly detailed surfaces and complex textures. While Gaus Fusion has demonstrated substantial success in rendering general 3D shapes, fine structure generation remains difficult, particularly for objects with very thin or intricate geometric details. This limitation is particularly apparent when attempting to generate objects with a high degree of texture complexity, where the model may struggle to capture the necessary fine details, leading to lower-quality reconstructions in those areas.

Another limitation concerns the substantial memory demands associated with Gaus-Fusion. The original Gaussian Splatting method, as outlined in the seminal article on this technique, requires a GPU with at least 24GB of memory to run effectively. At the time the article was published, such memory requirements were considered high, and not every researcher or developer had access to hardware with this level of memory capacity. This makes the method somewhat inaccessible for a large portion of the research community and practitioners who do not have access to high-end computational resources. Furthermore, during our experiments, we encountered the need to limit the number of generated Gaussians to avoid memory overflow, which in turn constrained the level of detail that could be captured in the 3D model.

To address these challenges, we recommend that future work focus on the development of hardware solutions with expanded GPU memory capacity. With the ability to handle a larger number of Gaussians, we would be able to generate more detailed 3D models with higher fidelity, improving the overall quality of the output. More importantly, reducing the memory footprint of Gaus-Fusion itself is critical. If Gaus-Fusion can be optimized to use less memory while still maintaining its ability to refine the shapes and increase the level of detail in the generated models, this could have a profound impact on the accessibility and scalability of the technique. A more memory-efficient implementation would lower the hardware requirements for running the model, making it more accessible to a wider range of users, from researchers working with limited computational resources to companies looking to deploy text-to-3D generation in production environments.

In addition to addressing memory efficiency, another area of improvement lies in the initialization process of the 3D points. The first method of Gaus-Fusion involved generating a large number of 3D Gaussians concentrated at the center of the scene. While this method is somewhat effective, we believe that exploring alternative initialization strategies and iteration methods could lead to further improvements in both the quality and flexibility of the generated models. We suggest that, for the first method, prior knowledge could be leveraged so that the parameters of neighboring Gaussians (after applying the gradient multiplied by the learning rate) are approximately equal. This might allow neighboring 3D Gaussians to adjust their placements coherently, forming a better shape rather than moving in random directions.

Although Gaus Fusion produced impressive results in the second method, where we utilized an optimized point cloud initialization, it still exhibits noticeable blurring along the edges. We believe this issue arises from the inherent properties of gaussians. To address this, we recommend exploring alternative point cloud representations beyond 3D gaussians or incorporating a refinement mechanism to enhance detail and sharpen the 3D model.

Another promising avenue for improvement is the implementation of specialized computational kernels designed to optimize the rendering process. While the 30-minute generation time in the first method represents a notable improvement compared to previous approaches, it still entails a significant time investment, particularly for projects requiring rapid iteration or large-scale generation of multiple 3D models. By developing computational kernels tailored specifically to the needs of Gaussian Splatting, we could potentially reduce the rendering time for each object, making the overall generation process more efficient. Such optimizations could decrease the time required to produce complex 3D models, benefiting a wide range of applications—from real-time rendering in virtual environments to the automated creation of 3D assets for gaming and simulation. This is especially relevant for the second method, where the generation time was twice as long as in the first method.

Furthermore, while the second method yielded decent results for objects like tractors, which are highly dependent on viewing direction, it took twice as long to generate compared to objects with more cylindrical or spherical symmetry. Therefore, we suggest exploring alternative point cloud initialization methods that could improve results for objects with different shapes, such as box-like or other non-symmetric forms. One approach could involve creating a 'point cloud bank' and developing an algorithm that selects the most appropriate point cloud from the bank based on the text prompt, optimizing the efficiency of 3D object generation.

In addition, we have not taken depth information into account and have focused solely on the visual results of the RGB-rendered images. For instance, there may be 3D points rendered in black that are not visible against a black background. Incorporating depth into the model could potentially enhance the overall results.

Furthermore, the development of more efficient algorithms for Gaussian placement and distribution could help improve both the quality of the generated models and the speed of the training process. Current methods rely heavily on iterative adjustments to the Gaussians based on positional gradients, which can be computationally expensive. By exploring alternative approaches to Gaussian placement, such as adaptive sampling techniques or machine learning-based strategies, it may be possible to accelerate the generation of high-quality 3D models while reducing the computational overhead.

In conclusion, while Gaus Fusion has demonstrated significant advantages in terms of performance and quality for text-to-3D generation, there remain several challenges that need to be addressed. Key areas for improvement include memory efficiency, fine structure generation, and the optimization of initialization and iteration strategies. By exploring novel hardware solutions, refining memory management, and implementing more efficient algorithms, it will be possible to further enhance the capabilities of Gaus-Fusion and make it a more accessible and scalable solution for the generation of high-quality 3D models from text descriptions.

Appendix

Gaus Fusion GitHub: https://github.com/TalShwartz1/Latent_Gausian_Splatting/tree/main
NFSD GitHub: https://github.com/TalShwartz1/Latent-NFSD/tree/main

References

[1] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, “NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis,” *Communications of the ACM*, vol. 65, no. 1, pp. 99-106, Jan. 2022.

[2] A. Jain, B. Mildenhall, J. T. Barron, P. Abbeel, and B. Poole, “Zero-Shot Text-Guided Object Generation with Dream Fields,” *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[3] B. Poole, A. Jain, J. T. Barron, and B. Mildenhall, “DreamFusion: Text-to-3D using 2D Diffusion,” *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.

[4] G. Metzer, E. Richardson, O. Patashnik, R. Giryes, and D. Cohen-Or, “Latent-NeRF for Shape-Guided Generation of 3D Shapes and Textures,” *arXiv preprint arXiv:2211.07600*, 2022.

[5] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, “3D Gaussian Splatting for Real-Time Radiance Field Rendering,” *ACM Transactions on Graphics*, vol. 42, no. 4, Article 0, Aug. 2023.

[6] O. Katzir, O. Patashnik, D. Cohen-Or, and D. Lischinski, “Noise-Free Score Distillation,” *arXiv preprint arXiv:2310.17590*, 2023.

[7] C. Saharia, W. Chan, S. Saxena, L. Li, J. Whang, E. Denton, S. K. S. Ghasemipour, B. K. Ayan, S. S. Mahdavi, R. G. Lopes, T. Salimans, J. Ho, D. J. Fleet, and M. Norouzi, “Photorealistic text-to-image diffusion models with deep language understanding,” *arXiv preprint*, 2022.

[8] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, “High-resolution image synthesis with latent diffusion models,” *arXiv preprint*, 2021.