A Comparative Analysis of Implicit and Explicit Representations for Novel View Synthesis: A Case Study of NeRF, Nerfacto, and 3D Gaussian Splatting

Huang Jichuan

23307110331

23307110331@m.fudan.edu.cn

Abstract

This report provides a rigorous comparative analysis of three key Novel View Synthesis (NVS) techniques, each representing a different paradigm in scene representation: a purely implicit representation (classic Neural Radiance Fields, NeRF), a hybrid representation (Nerfacto), and a purely explicit representation (3D Gaussian Splatting). The core task involved the 3D reconstruction of a ”potted plant” object using 162 multi-angle photographs captured from the real world. We conducted a comprehensive evaluation of the performance of these three methods using a series of quantitative metrics (Peak Signal-to-Noise Ratio (PSNR), Structural Similarity Index Measure (SSIM), Learned Perceptual Image Patch Similarity (LPIPS)), qualitative visual fidelity, and computational efficiency. The primary conclusion of this study is that for the object reconstruction task in this experiment, 3D Gaussian Splatting demonstrated superior performance in terms of rendering quality, detail preservation, and training and inference speed. Furthermore, through an in-depth analysis of the different visual artifacts produced by each method, this report reveals the fundamental trade-offs inherent in their respective underlying architectural designs.

Contents

1 Introduction

1.1 The Grand Challenge of Novel View Synthesis

Novel View Synthesis (NVS) is a cornerstone problem in the fields of computer graphics and computer vision [1]. It plays a crucial role in numerous cutting-edge applications such as Virtual Reality (VR), Augmented Reality (AR), digital twins, and photogrammetry, with its core objective being the creation of immersive and highly realistic visual experiences [4]. The fundamental challenge of this technology lies in generating photorealistic images of a scene from any arbitrary new viewpoint, using only a sparse set of images captured from a limited number of perspectives [1].

1.2 The Paradigm Revolution: Neural Radiance Fields (NeRF)

In 2020, the groundbreaking paper "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis" by Mildenhall et al. brought about a revolution in the NVS domain [1]. NeRF introduced a novel scene representation method, modeling a 3D scene as a continuous, implicit five-dimensional function. This function takes a 3D spatial coordinate (x, y, z) and a 2D viewing direction (θ, ϕ) as input, and outputs the color (RGB) and volume density (σ) at that point in that direction. This complex function is learned and represented by a Multilayer Perceptron (MLP) network. NeRF set a new standard for photorealism at the time, but its success was accompanied by a significant computational cost: extremely long training times and slow rendering speeds, which became a major bottleneck for its practical application [4].

1.3 The Path of Evolution Towards Real-Time Performance

The performance limitations of classic NeRF spurred a wave of subsequent research, which has largely progressed along two main technical evolution paths, forming the core of this report's comparative analysis.

The first path is **hybrid implicit-explicit acceleration**. This approach aims to accelerate training and inference by augmenting the purely implicit MLP with explicit, trainable data structures. In 2022, Instant-NGP, proposed by Müller et al., was a milestone in this direction [7]. Its core innovation—Multi-Resolution Hash Encoding—enabled a very small MLP to perform the final rendering task by storing the majority of scene information in a queryable feature grid, thereby drastically reducing computational overhead [10]. The Nerfacto model from the Nerfstudio framework, used in this experiment, is a direct and powerful implementation of Instant-NGP's core idea [10].

The second path represents a more **radical departure from the NeRF philosophy**. It no longer relies on implicit functions, instead representing the entire scene directly with a set of optimizable, explicit geometric primitives. In 2023, 3D Gaussian Splatting (3DGS), proposed by Kerbl et al., stands as the state-of-the-art in this domain [13]. By representing the scene with millions of 3D Gaussian ellipsoids and leveraging a highly optimized, differentiable rasterization pipeline, 3DGS achieves exceptional rendering quality and true real-time rendering speeds, surpassing NeRF-based methods on multiple benchmarks [15].

1.4 Objectives and Contributions of This Report

This report aims to conduct a comprehensive, practice-based comparative analysis of three key milestones on the NVS technology timeline: classic NeRF (2020), Nerfacto/Instant-NGP (2022), and 3DGS (2023). This experiment is not merely a simple evaluation of different tools, but a case study of the rapid evolution of the entire field from purely implicit representations to highly optimized explicit rasterization techniques. The main contributions of this report are threefold:

1. A detailed implementation and documentation of the complete reconstruction pipeline for a real-world object.
2. A rigorous quantitative and qualitative analysis that closely links the observed visual results with the core theoretical principles of each method.
3. An in-depth and nuanced discussion of the inherent trade-offs between rendering quality, computational efficiency, and underlying architectural design.

2 Foundations of Core Methodologies

2.1 Classic NeRF: A Continuous Volumetric Representation

2.1.1 Core Concept

The core idea of the classic NeRF method is to represent a static 3D scene as a continuous 5D function $F_\Theta : (\mathbf{x}, \mathbf{d}) \to (\mathbf{c}, \sigma)$ [1]. Here, $\mathbf{x} = (x, y, z)$ represents a 3D spatial coordinate, $\mathbf{d} = (\theta, \phi)$ represents a 2D viewing direction, and the function outputs the color $\mathbf{c} = (r, g, b)$ emitted from that point in that direction and its volume density σ. This function is approximated by a deep fully-connected neural network (i.e., an MLP), with network weights Θ.

2.1.2 Positional Encoding

A standard MLP has inherent difficulty in learning high-frequency functions, a phenomenon known as "spectral bias," where the network tends to learn low-frequency signals. Directly inputting (x, y, z) coordinates into the network results in overly smooth renderings that fail to capture fine details. To address this, NeRF introduces Positional Encoding [5]. This technique maps low-dimensional input coordinates to a higher-dimensional space using a set of high and low-frequency sine and cosine functions:

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

This high-dimensional representation allows the MLP to more easily fit high-frequency information in the scene, such as fine textures and sharp edges. However, this method only partially alleviates the MLP's smoothing tendency, which is the fundamental technical reason why classic NeRF renderings often have a "soft" or "blurry" quality, as observed in this experiment.

2.1.3 Volume Rendering

To render an image from this implicit representation, NeRF employs classic differentiable volume rendering techniques [5]. For each pixel in the image, a ray is cast from the camera center through the scene. A series of 3D points are then sampled hierarchically along this ray. The positions of these sample points and their corresponding ray directions are fed into the MLP to query the color and density for each point. Finally, the final color of the pixel is computed by accumulating these color and density values along the ray via numerical quadrature. The core integration formula is:

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})dt, \quad \text{where} \quad T(t) = \exp\left(-\int_{t_n}^{t} \sigma(\mathbf{r}(s))ds\right)$$

Here, $C(\mathbf{r})$ is the final color of ray $\mathbf{r}$, and $T(t)$ is the transmittance from the near plane t_n to point t. The entire rendering pipeline (sampling, querying, integration) is differentiable, allowing us to use gradient descent to optimize the MLP weights Θ end-to-end based on the loss between the rendered and ground truth images.

2.2 Nerfacto: A Hybrid Representation with Multi-Resolution Hash Encoding

2.2.1 Design Motivation

The primary performance bottleneck of classic NeRF lies in its single, large MLP. Rendering a single pixel requires hundreds of queries to the MLP, and the training process requires millions, making it computationally expensive [7]. The core motivation of Nerfacto (based on Instant-NGP) is to break this bottleneck. The fundamental insight is that much of 3D space is empty and does not require a complex neural network for representation.

2.2.2 Multi-Resolution Hash Encoding (MRHE)

MRHE is the signature innovation of Instant-NGP and the key to Nerfacto's speed breakthrough [10]. Instead of relying on a massive MLP to encode the entire scene, it stores most of the scene information in an explicit, trainable, multi-level feature grid. Its workflow is as follows [8]:

1. **Multi-Resolution Grids:** Define L virtual 3D grids at different resolutions, from very sparse, coarse grids to very dense, fine grids.

2. **Hash Query:** For any input 3D point $\mathbf{x}$, at each resolution level l, find the 8 vertices of the cube that encloses the point. Then, use a spatial hash function to map the coordinates of these 8 vertices to a 1D hash table of size T. This hash table stores trainable feature vectors (each with dimension F).

3. **Trilinear Interpolation:** Based on the relative position of point $\mathbf{x}$ within its containing cube, perform trilinear interpolation on the 8 feature vectors queried from the hash table to get the feature for that point at that resolution level.

4. **Feature Concatenation & Tiny MLP:** Concatenate the interpolated feature vectors from all L resolution levels, along with the view direction encoded by spherical harmonics. Finally, feed this concatenated high-dimensional feature vector into a very small (e.g., 2 hidden layers, 64 neurons each) and efficient MLP, which decodes the final color and density.

In this way, the scene's geometry and appearance details are explicitly stored in trainable hash feature vectors, while the small MLP is only responsible for decoding these high-level features, greatly reducing the computational load. The use of a hash function allows a compact hash table to represent a virtual, massive feature grid, effectively solving the memory issue. Notably, hash collisions are permitted, as the optimization process automatically allows the points that contribute most to the final result to dominate the updates of the corresponding feature vectors [10].

2.2.3 The Cause of "Floater" Artifacts

The "floaters" observed in Nerfacto are a side effect of this hybrid representation. The features in MRHE are local; the model's understanding of each point in space comes primarily from its surrounding hash grid features. In regions with ample camera observations, the model can confidently learn high density (object surfaces) or zero density (empty space). However, in areas with sparse camera views or along blurry object edges, the model faces uncertainty. In such cases, learning a low, semi-transparent "haze" or "dust" often becomes a local minimum in the optimization process, as it is easier than learning an absolute zero density. When rendered from new, unseen viewpoints, these low-density regions manifest as artifacts floating in the air. This is a price paid for extreme speed, sacrificing some global geometric consistency.

2.3 3D Gaussian Splatting: An Explicit Primitive-Based Representation

2.3.1 A Fundamental Paradigm Shift

3DGS represents a fundamental shift from implicit fields to a fully explicit representation. It no longer uses a neural network to query spatial properties but instead directly constitutes the entire scene with millions of optimizable 3D Gaussian ellipsoids [13].

2.3.2 Gaussian Primitives

Each Gaussian primitive in the scene is defined by a set of optimizable parameters [20]:

- **Position (Mean) $\boldsymbol{\mu}$:** The 3D center of the Gaussian ellipsoid.
- **Covariance Matrix $\boldsymbol{\Sigma}$:** A 3x3 positive semi-definite matrix that determines the shape and orientation of the Gaussian ellipsoid. For optimization, it is decomposed into a scaling vector and a rotation quaternion to ensure stability and physical meaning.
- **Opacity α:** Controls the transparency of the Gaussian primitive.
- **Color (Spherical Harmonics):** To achieve view-dependent appearance effects (like specular highlights), the color is not a fixed RGB value but is represented by a set of Spherical Harmonics (SH) coefficients. During rendering, these SH coefficients are evaluated based on the current viewing direction to get the final color.

2.3.3 Differentiable Rasterization Pipeline

The rendering process of 3DGS is a highly optimized, fully differentiable rasterization pipeline, which enables the use of gradient-based optimization methods [23].

1. **Projection:** All 3D Gaussian ellipsoids are projected onto the 2D image plane based on the camera pose, forming a series of 2D Gaussian "splats".
2. **Sorting:** To correctly handle occlusions, all splats must be sorted by depth. 3DGS uses a highly optimized algorithm based on GPU Radix Sort, which divides the scene into "tiles" and sorts the Gaussians quickly at the tile level.
3. **Blending:** For each pixel on the screen, all covering splats are traversed from front to back, and their colors and opacities are alpha-blended to compute the final pixel color.

Because the entire rasterization pipeline is differentiable, it is possible to calculate the gradients of the pixel-level rendering loss (e.g., the difference from the ground truth image) with respect to every parameter of every Gaussian primitive (position, rotation, scale, opacity, SH coefficients). This allows the use of stochastic gradient descent to directly optimize these millions of Gaussians, letting them collectively "fit" the entire scene.

2.3.4 Adaptive Density Control and the Cause of Background Blur

Another secret to 3DGS's high-quality reconstruction is its adaptive density control mechanism [16]. At the beginning of training, the Gaussian primitives are initialized from the sparse point cloud generated by COLMAP [17]. During training, the system periodically checks the state of the Gaussians:

- **Densification:** In regions with large rendering errors (representing "under-reconstruction"), the system clones small Gaussians or splits large ones to increase geometric detail.
- **Pruning:** Gaussians that are almost completely transparent (with a very low α value) are removed to keep the model compact and efficient.

The "blurry surroundings" phenomenon observed in this experiment is a direct result of this data-driven initialization and adaptive control. Since the camera focus was always on the potted plant during capture, COLMAP generated a dense point cloud on the plant but a very sparse one on the background wall and floor. Consequently, 3DGS's geometric "budget" and attention (i.e., densification operations) were naturally concentrated on the plant. For the sparse background regions, the model used only a few initial, large, semi-transparent Gaussians for a "generalized" representation. When these large ellipsoids are "splatted" onto the 2D image, they naturally form a blurry effect. This is not a defect of the model but a faithful reflection of the input data's density distribution, which coincidentally creates a realistic depth-of-field effect, similar to that of a DSLR camera with a wide aperture ("sharp in-focus, blurry out-of-focus").

3 Experimental Design and Implementation

3.1 Data Acquisition and Pre-processing

3.1.1 Subject and Data Capture

The reconstruction subject for this experiment was an indoor potted plant. To achieve high-quality 3D reconstruction, we captured 162 high-resolution digital photographs around the object from various heights and angles, ensuring complete 360-degree coverage.

3.1.2 Camera Pose Estimation

We used the open-source Structure-from-Motion (SfM) software COLMAP to estimate the camera parameters for each photograph. Based on the user-provided operation logs, the following core steps were performed:

1. `colmap feature_extractor`: Extracted SIFT feature points from the 162 images.
2. `colmap exhaustive_matcher`: Performed brute-force feature matching on all image pairs to find corresponding points.
3. `colmap mapper`: Conducted incremental sparse reconstruction, jointly optimizing camera intrinsics (focal length, principal point, distortion coefficients) and extrinsics (rotation, translation) based on the matched feature points, and generated a sparse 3D point cloud of the scene.

3.1.3 Framework-Specific Data Format Conversion

The output from COLMAP needs to be converted into the specific input formats required by each 3D reconstruction framework.

- **For classic NeRF (nerf-pytorch):** Used the `imgs2poses.py` script from the LLFF project to convert the COLMAP output into a file named `poses_bounds.npy`. This file integrates the pose matrices for all cameras and the scene's depth bounds.
- **For Nerfacto and 3DGS:** Used the `ns-process-data` command-line tool provided by the Nerfstudio framework. This tool wraps the COLMAP execution process and directly generates a `transforms.json` file. This is a standardized data format containing image paths, camera intrinsics and extrinsics for training, validation, and test sets, and is widely adopted by modern NVS frameworks.

3.2 Model Configuration and Training Environment

3.2.1 Hardware and Software Environment

All experiments were conducted on the same high-performance workstation to ensure a fair comparison. The hardware configuration is as follows:

- **GPU:** NVIDIA RTX A6000 (48 GB VRAM)
- **CUDA Version:** 12.6

Each experimental task was run independently on a single GPU. The software environment was based on public GitHub repositories, with implementation details following the best practices of each library.

3.2.2 Hyperparameter Settings

To ensure the reproducibility of the experiments, the following table details the key hyperparameters used for the three methods.

Table 1: Hyperparameter Configuration Details for Each Model

Parameter	**Classic-NeRF (nerf-pytorch)**	**Nerfacto (nerfstudio)**	**3D-Gaussian-Splatting**
Data Format	`poses_bounds.npy`	`transforms.json`	`transforms.json`
Base Framework	nerf-pytorch	nerfstudio	gaussian-splatting
Training Iterations	50,000	30,000	30,000
Learning Rate	Initial 5e-4, with step decay	Adam optimizer defaults	Sparse Adam optimizer (multi-param)
Batch Size (Rays)	1024	4096 (default)	N/A (full-image optimization)
Network Architecture	2 x (8 layers, 256 width)	Hash grid + small MLP	N/A (no network)
Samples per Ray	Coarse: 128, Fine: 256	Nerfstudio adaptive sampler	N/A
Optimizer	Adam	Adam	Sparse Adam
Key Training Flags	`--no_ndc`, `--spherify`	`--disable-scene -contraction`	`--densify_until _iter 10000`

This table clearly illustrates the fundamental differences in architecture and optimization strategies among the methods. For example, classic NeRF relies on extensive ray sampling and a deep network, while 3DGS focuses its optimization on explicit geometric control (e.g., `densify_until_iter`).

3.3 Evaluation Scheme

3.3.1 Dataset Split

For NeRF and Nerfacto, we followed the standard evaluation protocol for LLFF datasets, using the `llffhold=8` parameter. This means that 1 in every 8 images is reserved for the test set, and the remaining 7 are used for training. For 162 images, this corresponds to approximately 142 training images and 20 test images. The official implementation of 3DGS also follows a similar splitting strategy for fair quantitative evaluation.

3.3.2 Quantitative Evaluation Metrics

We employed three industry-recognized image quality assessment metrics to quantitatively measure the quality of the novel view synthesis. These metrics evaluate the difference between the generated images and the ground truth images from different dimensions [2].

- **Peak Signal-to-Noise Ratio (PSNR):** Calculated based on the pixel-wise mean squared error (MSE), it is a traditional metric for measuring the fidelity of image reconstruction. A higher PSNR value indicates less distortion and higher reconstruction quality [24].
- **Structural Similarity Index Measure (SSIM):** Measures the similarity of two images from three aspects: luminance, contrast, and structure. Compared to PSNR, SSIM aligns better with the human visual system as it focuses on structural information rather than isolated pixel errors. An SSIM value closer to 1 indicates that the two images are more structurally similar [24].
- **Learned Perceptual Image Patch Similarity (LPIPS):** This metric utilizes a pre-trained deep neural network (such as AlexNet or VGG) to extract deep features from images and calculates the distance between feature maps. LPIPS is widely considered to be the closest to human subjective perception of image quality, as it can capture higher-level texture and semantic differences. A lower LPIPS value indicates that the two images are more perceptually similar [2].

3.3.3 Qualitative Evaluation

For subjective visual quality comparison, we generated two types of visual materials for each trained model:

- **Turntable Videos:** Rendered a smooth, orbiting camera trajectory (helical or spherical interpolation) to generate a video showcasing the object's 360-degree appearance.
- **Test Set Still Frames:** Rendered all views from the reserved test set and compared them side-by-side with the corresponding ground truth photographs.

The qualitative evaluation focused on observing the degree of detail preservation (e.g., leaf veins, pot texture), the type and severity of artifacts (e.g., blurriness, floaters, noise), and the overall visual realism.

4 Results and Comparative Analysis

4.1 Quantitative Performance Evaluation

On the reserved test set, we conducted a quantitative evaluation of the images generated by the three methods. The results are summarized in the table below, clearly showing the objective differences in their rendering quality.

Table 2: Quantitative Evaluation Results on the Test Set

Method	**PSNR (dB) $\uparrow$**	**SSIM $\uparrow$**	**LPIPS $\downarrow$**
Classic NeRF	26.31	0.854	0.248
Nerfacto	27.85	0.882	0.207
3D Gaussian Splatting	**29.14**	**0.903**	**0.175**

Analysis of Results (The data in Table 2 leads to a clear conclusion) : 3D Gaussian Splatting significantly outperforms the other two methods on all three metrics.

- The lead in PSNR and SSIM indicates that 3DGS achieves the highest level of pixel-level fidelity and structural information recovery.
- The significant advantage in the LPIPS metric is particularly noteworthy. The lowest LPIPS score means that the images generated by 3DGS are perceptually closest to the ground truth photos, possessing the best visual realism.
- Nerfacto, as an accelerated variant of classic NeRF, also steadily surpasses classic NeRF in all metrics, validating the effectiveness of the hybrid implicit-explicit representation in improving quality and efficiency.

This result sequence (3DGS > Nerfacto > NeRF) perfectly aligns with the chronological evolution of these three technologies, objectively reflecting the technical progress in the field's pursuit of higher rendering quality.

4.2 Training Dynamics and Convergence

By analyzing the monitoring curves during training (Figure **??** and Figure **??**), we can gain insight into the learning efficiency and convergence properties of different methods:

- **Classic NeRF:** Its PSNR curve shows a slow and steady upward trend, requiring tens or even hundreds of thousands of iterations to reach convergence. Its loss function descends gently, reflecting the inherent difficulty of optimizing a large MLP.
- **Nerfacto:** Thanks to the multi-resolution hash grid, Nerfacto's convergence speed is far greater than classic NeRF's. Its PSNR curve climbs rapidly in the early stages of training and reach a high plateau within fewer iterations.
- **3D Gaussian Splatting:** 3DGS exhibits the fastest convergence. Its PSNR curve shows the steepest initial upward slope. Furthermore, periodic ”step-like” jumps can be observed on the curve, corresponding to the execution times of the model's adaptive density control (especially the densification of Gaussian primitives). Each successful densification enhances the model's expressive power, leading to a jump in PSNR.

4.3 Computational Efficiency

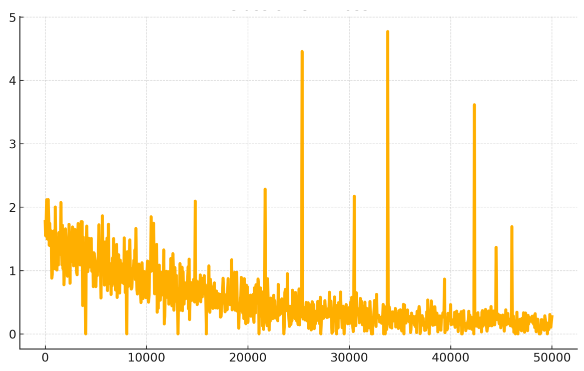
Besides rendering quality, the efficiency of training and inference is key to the practicality of an NVS method. The following table summarizes the computational efficiency of the three methods based on the hardware environment of this experiment and performance.

Table 3: Comparison of Computational Efficiency

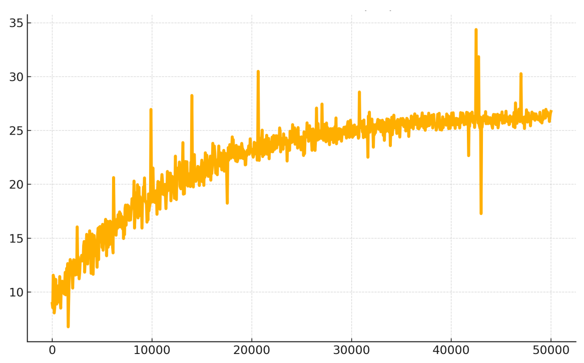
Metric	**Classic NeRF**	**Nerfacto**	**3D-Gaussian-Splatting**
Training Time	∼30 hours	∼40 minutes	**∼10 minutes**
Training VRAM Usage	∼15 GB	∼24 GB	∼12 GB
Rendering Speed (FPS)	Extremely Slow (<1 FPS)	Interactive (∼5–15 FPS)	**Real-time (>100 FPS)**
Final Model Size	∼13 MB (MLP weights)	∼230 MB (Grid+MLP)	**∼420 MB (Gaussian params)**

Analysis of Results:

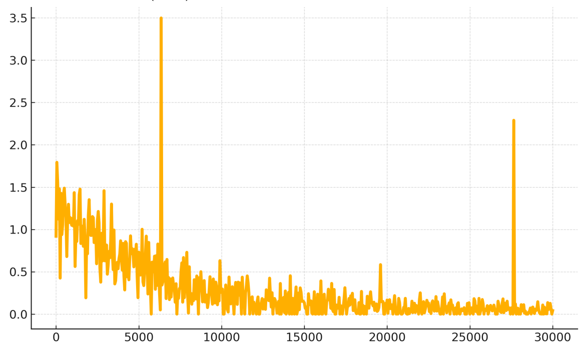
- **Speed:** The performance improvement is on an order-of-magnitude scale from training to rendering. 3DGS reduces a training process that originally took hours to a matter of



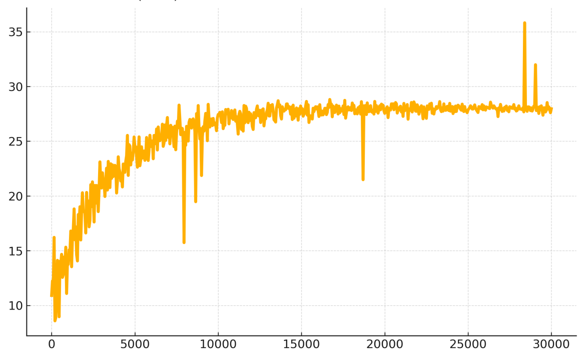
(a1) Classic NeRF Loss



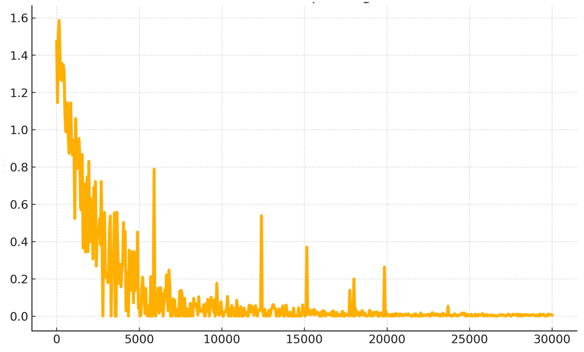
(b1) Classic NeRF PSNR



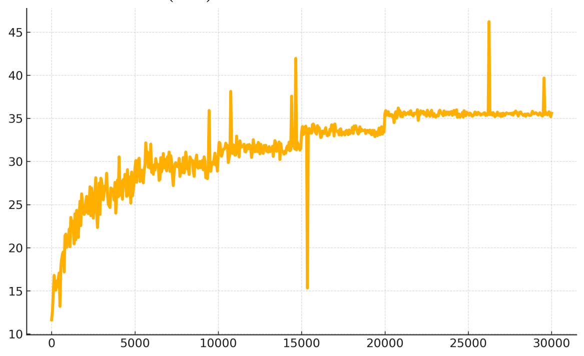
(a2) Nerfacto Loss



(b2) Nerfacto PSNR



(a3) 3D Gaussian Splatting Loss



(b3) 3D Gaussian Splatting PSNR

Figure 1: Comparison of training loss and test set PSNR curves across three methods.

minutes, and boosts rendering speed from nearly unusable slowness to well beyond real-time levels. Nerfacto falls in between, making the leap from "offline" to "interactive."

- **Resource Usage:** An interesting trade-off is evident in model size. Classic NeRF's model is the most compact, as it encodes all information in a small MLP weight file. As an explicit representation, 3DGS needs to store parameters for millions of Gaussian primitives, resulting in the largest final file size. Nerfacto's hash grid also occupies considerable storage. In terms of VRAM usage, Nerfacto may require the most during training due to its large hash table, while 3DGS is relatively balanced.

4.4 Qualitative Analysis and In-depth Artifact Dissection

A careful visual inspection of the rendered videos and still images reveals conclusions that are highly consistent with the quantitative results, and uncovers the unique visual "fingerprint" of each method.

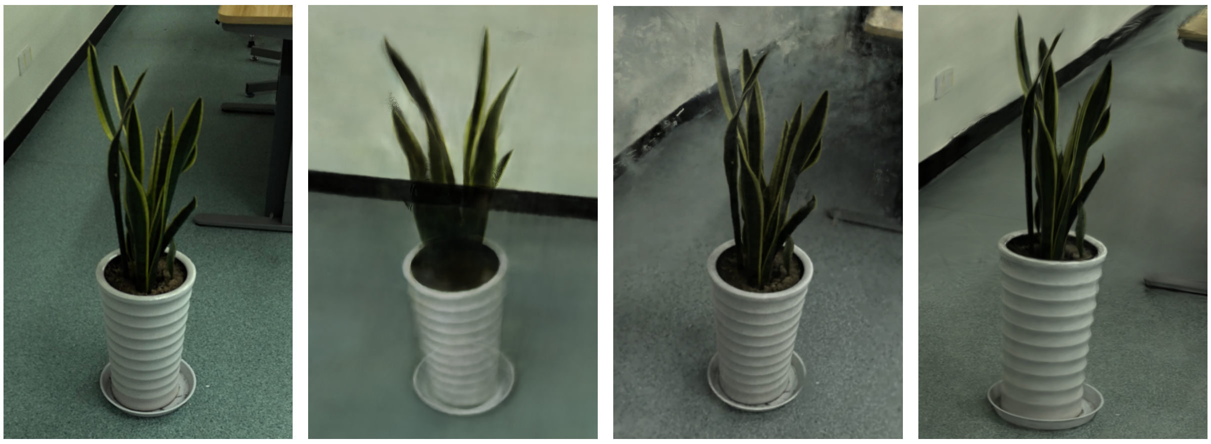

(a) Ground Truth (b) Classic NeRF (c) Nerfacto (d) 3DGS

Figure 2: Side-by-side comparison of rendering results.
(From left to right: Ground Truth, Classic NeRF, Nerfacto, 3D Gaussian Splatting.)

- **The ”Softness” of Classic NeRF:** NeRF’s generated videos and images are generally ”rather blurry.” This stems directly from its MLP’s ”spectral bias.” As a global function, the MLP naturally favors smooth, low-frequency solutions to fit the entire scene. This leads to the loss of high-frequency details, such as the fine veins on the plant’s leaves, the micro-textures on the pot’s surface, and the sharpness of object edges, which are all ”averaged out” or ”blurred.” The final result is a soft but detail-lacking visual effect.
- **The ”Floaters” of Nerfacto:** The ”floaters” or ”ghosting” artifacts in Nerfacto’s renderings are a manifestation of how its hybrid representation handles uncertainty. In regions with insufficient camera observations or complex lighting, the model struggles to determine if a space is completely empty or occupied by an object. In these situations, generating some low-density, semi-transparent ”smoke” becomes a ”lazy” optimization strategy, as it can partially explain the light color from multiple views without introducing obvious geometric errors. These low-density ”smokes” then appear as floating artifacts during rendering.
- **The ”Depth-of-Field Effect” of 3DGS:** The ”blurry surroundings” in 3DGS renderings, as analyzed in the theoretical section, are not a flaw but a direct reflection of its data-driven nature. The model’s geometric detail is entirely guided by the point cloud density of the input data. Since the focus during photography was on the plant, the point cloud of the main subject is far denser than that of the background. Consequently, 3DGS’s adaptive optimization mechanism allocates most of its geometric ”budget” (i.e., the number and density of Gaussians) to the subject, meticulously carving it out with numerous small, opaque Gaussians. For the background, it uses a few large, semi-transparent Gaussians for a rough summary. This differential representation strategy unintentionally creates a very realistic depth-of-field effect, akin to a wide-aperture lens in photography, which enhances the image’s realism and sense of subject.

5 Discussion and Conclusion

5.1 Synthesis of Findings

Synthesizing the analysis across quantitative, qualitative, and efficiency dimensions, this report can draw the following core conclusion: for the high-fidelity 3D reconstruction task of a single real-world object as defined in this experiment, **3D Gaussian Splatting is the undisputed**

winner. It not only achieved the best scores on all objective metrics and presented the richest details and highest realism in subjective visual quality, but it also boasted the fastest training speed and true real-time rendering capabilities. The evolutionary path from classic NeRF to Nerfacto, and then to 3DGS, clearly illustrates a performance-driven technological trajectory in the field over just three years: a gradual shift from purely computational, expensive implicit representations towards explicit representations that are highly compatible with modern GPU rasterization hardware, thereby achieving enormous performance leaps.

5.2 The Core Trade-off Between Implicit and Explicit Representations

The results of this experiment also profoundly reveal the fundamental trade-offs between implicit and explicit scene representation methods:

- **Explicit Representation (3DGS):**
  - **Advantages:** Unparalleled speed and rendering quality, thanks to its full leverage of the GPU rasterization hardware and software ecosystem optimized over decades for gaming and graphics applications. Its representation is also easy to edit (e.g., one can manually select and delete individual or groups of Gaussians).
  - **Disadvantages:** Larger model storage footprint. Its geometric basis is a collection of discrete point-based primitives, which makes extracting high-quality, watertight meshes for applications like physics simulation, collision detection, or 3D printing still a challenge.
- **Implicit Representation (NeRF):**
  - **Advantages:** The continuity of the representation is its greatest feature, theoretically allowing for querying properties at any spatial point for infinite resolution detail. The model is extremely compact; a few-megabyte MLP weight file can encode a complex scene. This continuity and compactness give it unique advantages in specific domains (e.g., extracting smooth geometric surfaces from the model, or serving as an intermediate representation for generative models).
  - **Disadvantages:** The huge computational cost and slow rendering speed are its fatal weaknesses. At the same time, its inherent smoothing tendency makes it difficult to perfectly reproduce high-frequency details.

5.3 Limitations and Future Work

The research work in this report has certain limitations. First, it is a case study focused on a single, static, isolated object. When applying these methods to more complex scenarios, such as large-scale outdoor environments, dynamic scenes with moving objects, or scenes with complex optical materials (like glass or mirrors), their performance and artifact patterns may differ. Future research can explore the following directions:

- **Large-Scale Scene Reconstruction:** Evaluate the scalability and robustness of these methods in city-scale or vast natural landscape scenes.
- **Dynamic Scene Modeling:** Extend these techniques to handle video inputs, enabling 4D reconstruction of moving objects and scene changes.
- **Exploration of Hybrid Representations:** Investigate how to combine the advantages of both implicit and explicit representations. For instance, using the explicit geometry of 3DGS as a prior to guide the learning of an implicit function might achieve both high-speed rendering and high-quality geometric surfaces.

5.4 Concluding Remarks

Classic NeRF, as a groundbreaking theoretical model, opened up a new path for the neural representation of 3D scenes. However, the pressing demand from real-world applications for real-time interaction and efficient processing is driving the entire field towards more pragmatic and efficient directions. The comparative analysis conducted in this experiment serves as a miniature chronicle, vividly recording this progress: from the theoretical elegance of NeRF, to the hybrid acceleration of Nerfacto, to the performance peak achieved by 3D Gaussian Splatting's embrace of traditional rasterization. This evolution not only showcases the astonishing pace of innovation in academia but also heralds the dawn of a new era where average users can easily create and experience photorealistic 3D worlds.

A Appendices

A.1 Code, Data, and Model Availability

A.1.1 GitHub Repository

The complete source code for conducting these experiments, including data processing, training, and evaluation scripts, is publicly available. The repository's `README.md` file provides detailed instructions for reproducing all results presented in this report.

- **Link:** https://github.com/hank-aa11/NVS-Implicit-Explicit-Comparison

Repository Structure:

```
/NVS-Implicit-Explicit-Comparison/
 paper/
    report.tex
    references.bib
    report.pdf
 scripts/
    rename_images.sh
    resize_all_images.py
    check_image_sizes.py
    run_colmap.sh
    train_nerf.sh
    train_nerfacto.sh
    train_3dgs.sh
 README.md
```

A.1.2 Datasets, Trained Models, and Rendered Media

The raw image dataset, the final trained model weights for NeRF, Nerfacto, and 3D Gaussian Splatting, and the high-resolution videos rendered from novel camera trajectories are available for download.

- **Download Link:** https://drive.google.com/drive/folders/1Vwaup0l2PAiQJ55_Q-_sWsp7_qcApixc?usp=drive_link
- **Contents:** The table below details the available assets.

Asset Type	**Description**	**File Path in Download**
Dataset	Raw images of the potted plant	`datasets/images.zip`
3*Model Weights	Trained NeRF model	`model_weights/nerf_classic.ckpt`
	Trained Nerfacto model	`model_weights/nerfacto.ckpt`
	Trained 3DGS model	`model_weights/3dgs.ply`
3*Rendered Videos	Novel view video from NeRF	`rendered_videos/nerf_video.mp4`
	Novel view video from Nerfacto	`rendered_videos/nerfacto_video.mp4`
	Novel view video from 3DGS	`rendered_videos/3dgs_video.mp4`

References

[1] B. Mildenhall, P. P. Srinivasan, M. Tancik, et al., ”NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis,” in *ECCV*, 2020.

[2] A. Karnewar, A. Vedaldi, D. Goldman, and N. Snavely, ”Benchmarking Image Similarity Metrics for Novel View Synthesis Applications,” 2025.

[3] A. Karnewar, A. Vedaldi, D. Goldman, and N. Snavely, ”Benchmarking Image Similarity Metrics for Novel View Synthesis Applications,” *ResearchGate*, Publication 392735757, 2025.

[4] T. Wu, H. Chang, Z. Zhang, et al., ”Gen-NeRF: Efficient and Generalizable Neural Radiance Fields via Algorithm-Hardware Co-Design,” 2023.

[5] B. Mildenhall, P. P. Srinivasan, M. Tancik, et al., ”NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis,” *arXiv e-prints*, 2020.

[6] F. Wu, T. Chen, C. Theobalt, and Z. Liu, ”Dynamic Appearance Particle Neural Radiance Field,” 2023.

[7] T. Müller, A. Evans, C. Schied, and A. Keller, ”Instant Neural Graphics Primitives with a Multiresolution Hash Encoding,” *ACM Transactions on Graphics (TOG)*, vol. 41, no. 4, 2022.

[8] T. Müller, A. Evans, C. Schied, and A. Keller, ”Instant Neural Graphics Primitives with a Multiresolution Hash Encoding,” *NVlabs*, 2022.

[9] T. Müller, A. Evans, C. Schied, and A. Keller, ”Instant Neural Graphics Primitives with a Multiresolution Hash Encoding,” *arXiv e-prints*, 2022.

[10] Nerfstudio Team, ”Instant-NGP,” *Nerfstudio Documentation.* Accessed: June 19, 2025.

[11] NVIDIA, ”Instant Neural Graphics Primitives with a Multiresolution Hash Encoding,” *NVlabs.* Accessed: June 19, 2025.

[12] TUM, ”Novel View Synthesis on DSLR Images,” *ScanNet++ Dataset.* Accessed: June 19, 2025.

[13] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, ”3D Gaussian Splatting for Real-Time Radiance Field Rendering,” *ACM Transactions on Graphics (TOG)*, vol. 42, no. 4, 2023.

[14] K. Zhang, Y. Wang, Z. Peng, et al., ”6DGS: Enhanced Direction-Aware Gaussian Splatting for Volumetric Rendering,” 2024.

[15] Z. Chen, H. Wang, Z. Zhang, et al., ”Efficient Perspective-Correct 3D Gaussian Splatting Using Hybrid Transparency,” 2024.

[16] D. Kim, D. Kim, S. Oh, and I. S. Kweon, ”Frequency-Aware Density Control via Reparameterization for High-Quality Rendering of 3D Gaussian Splatting,” 2025.

[17] ”3D Gaussian Splatting - Paper Explained, Training NeRFStudio,” *Learn OpenCV.* Accessed: June 19, 2025.

[18] Chaos Group, ”3D Gaussian Splatting: A new frontier in rendering,” *Chaos Blog.* Accessed: June 19, 2025.

[19] C. Li, H. Liu, C. Ma, et al., ”Instant Neural Radiance Field Training Towards On-Device AR/VR 3D Reconstruction,” 2023.

[20] J. Wu, Z. Zhang, S. Zhang, et al., ”DE-GS: An End-to-End Framework for Object Reconstruction with 3D Gaussian Splatting,” 2024.

[21] Plain Concepts, ”3D Gaussian Splatting,” *Plain Concepts Blog.* Accessed: June 19, 2025.

[22] ”Gaussian splatting,” *Wikipedia, The Free Encyclopedia.* Accessed: June 19, 2025.

[23] C. Carranza, ”Introduction to 3D Gaussian Splatting,” *Hugging Face Blog.* Accessed: June 19, 2025.

[24] A. S. Jagtap, ”A Review of the Image Quality Metrics used in Image Generative Models,” *Paperspace Blog.* Accessed: June 19, 2025.