PHANTOMSPLAT: STEALTHY POST-RECONSTRUCTION ATTACKS ON 3D GAUSSIAN SPLATTING

Anonymous authors
Paper under double-blind review

ABSTRACT

3D Gaussian Splatting (3DGS) has become a widely adopted representation for 3D scene reconstruction, real-time rendering, and downstream perception. As optimized 3DGS representations are increasingly distributed and reused as scene assets, their integrity after reconstruction becomes an important security concern: a compromised asset may be directly consumed by downstream systems even when the original data and reconstruction process are trusted. Existing 3DGS attacks primarily target the construction stage through poisoned training views or perturbed inputs, leaving this post-reconstruction failure mode largely unexplored. We study whether an already optimized 3DGS asset can be transformed into a self-contained, view-dependent malicious representation while remaining visually plausible under benign inspection. We propose **PhantomSplat**, which injects a small set of low-opacity adversarial Gaussians while keeping the original representation fixed. The design targets three requirements for asset-level security evaluation: localized adversarial behavior, benign-view consistency, and structural inconspicuousness. Surface-constrained placement reduces detectable geometric artifacts, while bounded opacity optimization limits degradation under normal views. A first-order rendering surrogate further makes this constrained optimization practical without repeated full-scene rendering. Across diverse scenes and downstream perception models, PhantomSplat induces strong target-view effects while largely preserving the original asset under benign views. On Mip-NeRF360, PhantomSplat achieves 33.96 dB attack-view PSNR while retaining 29.62 dB benign-view PSNR, compared with 30.36 dB for the clean asset, with an average optimization time of 4.25 minutes. These findings reveal a gap in current 3DGS security assumptions and motivate post-reconstruction integrity verification before scene assets are deployed in downstream systems.

1 INTRODUCTION

3D Gaussian Splatting (3DGS) represents 3D scenes using anisotropic Gaussian primitives, enabling high-fidelity real-time rendering and fully differentiable optimization (Kerbl et al., 2023). Since its introduction, 3DGS has been increasingly adopted across digital asset creation (Chen et al., 2024), entertainment (Luiten et al., 2024), autonomous driving (Deng et al., 2025), and robotics (Keetha et al., 2024). Beyond serving as the output of a scene-reconstruction process, optimized 3DGS representations are increasingly reused as deployable scene assets in downstream rendering, perception, and simulation pipelines. As a result, the integrity of the released 3DGS asset itself becomes part of the security boundary: even when the original data and reconstruction process are trusted, downstream systems ultimately rely on the integrity of the asset they receive.

Recent work has exposed several vulnerabilities in 3DGS, including view-dependent deception (Ke et al., 2025), object poisoning (Hull et al., 2025), and generation corruption (Qiao et al., 2026). However, these attacks primarily intervene before or during representation construction, through poisoned training views, adversarial inputs, or compromised generation processes. This focus leaves a distinct gap after a trusted reconstruction process and before downstream deployment. In such settings, downstream users may directly consume a distributed 3DGS representation without rerunning or verifying its original reconstruction process. Consequently, even when the source data and reconstruction pipeline are trusted, a subsequently modified asset can still introduce malicious behavior into downstream rendering or perception systems. Security mechanisms centered on data

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

sanitization or reconstruction-time robustness therefore do not fully address the integrity of deployed 3DGS assets.

This motivates our central question: Can an already optimized 3DGS asset be effectively modified after reconstruction to exhibit localized, view-dependent behavior while remaining visually and structurally inconspicuous under benign inspection? Addressing this question is non-trivial because a meaningful post-reconstruction manipulation must satisfy three competing requirements. First, it must induce a targeted effect from selected viewpoints without globally altering the scene. Second, it should preserve the appearance of the original asset under benign views so that the manipulation is not readily exposed through ordinary rendering inspection. Third, the modification itself should avoid introducing conspicuous geometric artifacts, such as isolated or abnormally clustered Gaussians, that could reveal the tampering at the representation level. These requirements are inherently in tension in 3DGS. For example, increasing the influence of injected primitives to strengthen the target-view effect may also alter their appearance under benign views or require more conspicuous modifications to the underlying Gaussian structure.

To meet competing requirements simultaneously, we propose **PhantomSplat**, a post-reconstruction attack built on a simple idea: rather than modifying an already optimized 3DGS asset globally, we leave the original representation fixed and introduce only a small set of carefully placed Gaussians whose effects are concentrated around selected viewpoints. This localized modification allows the attack to produce a strong target-view effect while limiting changes elsewhere in the scene. We realize this idea using low-opacity Gaussians placed near the original scene surfaces, which helps preserve benign-view appearance and reduces conspicuous structural artifacts. The low-opacity design also enables a first-order approximation that decouples the injected primitives from the much larger frozen scene, allowing the original scene contribution to be cached for efficient optimization, while cached first-order gradients are used to constrain benign-view degradation. Together, these designs enable localized adversarial behavior while preserving benign-view fidelity and structural inconspicuousness at practical optimization cost. We evaluate PhantomSplat across diverse scenes, viewpoints, and downstream perception models, and find that the manipulated assets can induce strong target-view effects while remaining visually close to the original assets under benign views, with the adversarial behavior localized around the selected viewpoint rather than globally altering the scene. Surface-constrained injection further reduces simple structural outlier signals, while the resulting renderings remain capable of affecting downstream perception models. Our results demonstrate that successful reconstruction and benign visual inspection alone are insufficient to guarantee the integrity of a subsequently distributed 3DGS asset, motivating asset-level verification before deployment.

In summary, our contributions are:

1. We identify and formalize a **stealthy post-reconstruction threat model for 3DGS assets**, in which an already optimized scene representation can be manipulated after a trusted reconstruction process but before downstream deployment, while remaining visually and structurally inconspicuous under benign inspection. This exposes an underexplored attack surface in the 3DGS asset lifecycle.
2. We propose **PhantomSplat**, a constrained post-reconstruction attack that injects low-opacity adversarial Gaussians while keeping the original representation fixed. By exploiting the low-opacity regime, PhantomSplat approximately decouples the injected primitives from the frozen scene for efficient optimization, while benign-view constraints and surface-constrained placement preserve visual consistency and reduce structural artifact.
3. We provide a **systematic security characterization** across diverse scenes and downstream perception models, evaluating attack effectiveness, benign- and novel-view consistency, viewpoint locality, structural detectability, and computational cost. Our results motivate further investigation of asset-level integrity verification for deployed 3DGS representations.

2 Related Work

2.1 Adversarial Vulnerabilities of 3D Scene Representations

Adversarial vulnerabilities in 3D scene representations are closely tied with the underlying representation. For explicit representations such as point clouds, attackers can directly perturb, add, or

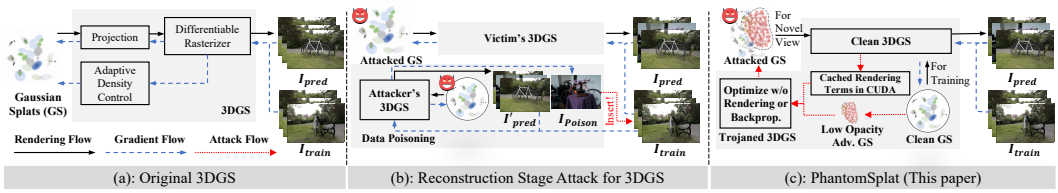

Figure 1: Comparison of 3DGS pipelines under different adversarial threat models. (a) Original 3DGS: A clean Gaussian representation is constructed through standard rendering-based optimization. (b) Reconstruction stage attack: The attacker injects poisoned views into the training data, requiring the victim to retrain the full 3DGS representation. (c) PhantomSplat: Our attack directly manipulates an already optimized clean 3DGS representation to produce a trojaned 3DGS representation, avoiding training data, full retraining, and iterative rendering/backpropagation overhead.

Table 1: Comparison of representative 3DGS attack methods. For retraining scope, **Full** denotes optimization-phase training from scratch, **Inject** denotes optimizing injected Gaussians while keeping the original 3DGS representation fixed, and ✗ denotes no retraining of the 3DGS representation. View-specific target image denotes user-specified content.

Method	Threat Model		Attack Capabilities		
	Attack Stage	Retraining Scope	View-Specific Target Image	Innocent-View Stealth	Structural Anomaly Evasion
Poison-splat Lu et al. (2025)	Reconstruction	Full	✗	✗	✗
StealthAttack Ke et al. (2025)	Reconstruction	Full	✓	✓	✗
Gausstrap Hong et al. (2025)	Reconstruction	Full	✓	✓	✗
COMPLICITSPLAT Hull et al. (2025)	Reconstruction	Full	✓	✓	✗
AdvSplat Qiao et al. (2026)	Input/Generation	✗	✗	✗	✗
PhantomSplat (ours)	Post-reconstruction	Inject	✓	✓	✓

remove geometric primitives to mislead downstream models (Xiang et al., 2019; Zhang et al., 2023; Naderi & Bajić, 2023). In contrast, Neural Radiance Fields (NeRFs) (Mildenhall et al., 2020; Tancik et al., 2022) encode scenes implicitly in neural network parameters, and existing adversarial attacks therefore primarily intervene through training data or input views (Horváth & Józsa, 2023; Jiang et al., 2024a; Fu et al., 2023; Jiang et al., 2024b).

3DGS (Kerbl et al., 2023) combines an explicit set of editable Gaussian primitives with differentiable, view-dependent rendering, enabling real-time rendering for simulation (Zheng et al., 2024; Keetha et al., 2024; Han et al., 2025) and large-scale environments (Wu et al., 2024; Liu et al., 2024). This representation enables direct access to scene parameters while retaining the ability to produce viewpoint-dependent appearance, creating a distinct security concern once optimized 3DGS representations are distributed as reusable assets.

2.2 Attacks on 3D Gaussian Splatting

Existing attacks on 3DGS primarily intervene before or during scene reconstruction. At the reconstruction stage, methods such as StealthAttack (Ke et al., 2025), Gausstrap (Hong et al., 2025), and COMPLICITSPLAT (Hull et al., 2025) manipulate training views to induce localized or view-dependent adversarial behavior in the reconstructed scene. In contrast, Poison-splat (Lu et al., 2025) poisons the reconstruction process to trigger excessive Gaussian densification and impose substantial computational overhead. A separate line of work targets downstream perception models through adversarial 3DGS-based representations or rendered views (Meng et al., 2026; Zhang et al., 2025; Zeybey et al., 2024; Lou et al., 2025). AdvSplat (Qiao et al., 2026), meanwhile, attacks feed-forward 3DGS generators by perturbing their input images rather than modifying an optimized scene representation.

As summarized in Table 1, existing methods differ in their attack assumptions and capabilities, yet very few works directly examines the integrity of an already optimized 3DGS asset. In contrast, we keep the original reconstruction fixed and study post-reconstruction manipulation of the resulting representation, exposing an asset-level security gap that persists even when the training data and reconstruction process are trusted.

3 PRELIMINARIES

3.1 3D GAUSSIAN SPLATTING

A detailed introduction to 3D Gaussian Splatting (3DGS) is provided in the Appendix A.1.

3.2 POST-RECONSTRUCTION ADVERSARIAL GAUSSIAN INJECTION

Threat model. We consider a post-reconstruction asset-manipulation setting in which an attacker obtains an already optimized 3DGS representation $\mathcal{G} = \{G_i\}_{i=1}^{n}$ and constructs a manipulated asset

$$\widetilde{\mathcal{G}} = \mathcal{G} \cup \Phi,$$

where $\Phi = \{\phi_i\}_{i=1}^{m}$ is a small set of injected adversarial Gaussians with $m \ll n$. The original Gaussian set $\mathcal{G}$ remains unchanged. The resulting asset can then be distributed and directly consumed by downstream rendering, simulation, or perception systems.

The attacker aims to induce a target visual effect from a selected victim viewpoint P_{vic}, while preserving consistency with the original asset under benign inspection views. We assume that the attacker has access to the optimized 3DGS asset, a target region on the asset, and either a specific victim viewpoint or a predefined domain over which the victim viewpoint can be selected. To enforce benign-view consistency, we consider a set of available or attacker-specified inspection poses $\mathcal{P}_{\mathrm{ben}}$. For each $P \in \mathcal{P}_{\mathrm{ben}}$, the clean asset itself provides the reference image $\mathcal{R}(\mathcal{G}, P)$. Thus, the attack does not require access to the original images used for 3DGS reconstruction.

Problem formulation. Given a victim viewpoint P_{vic} and target image I_{trg}, the attacker optimizes only the injected primitives Φ:

$$\begin{aligned} \Phi^* = \arg\min_{\Phi \in \mathcal{C}} \quad & \mathcal{L}_{\mathrm{adv}}\left(\mathcal{R}(\mathcal{G} \cup \Phi, P_{\mathrm{vic}}), I_{\mathrm{trg}}\right) \\ \text{s.t.} \quad & \mathcal{L}_{\mathrm{sth}}(\mathcal{G}, \Phi) \le \epsilon_{\mathrm{sth}}, \end{aligned} \tag{1}$$

where $\mathcal{R}(\cdot, P)$ denotes the 3DGS renderer and $\mathcal{L}_{\mathrm{adv}}$ measures the discrepancy between the attacked rendering and the target image. The stealth loss $\mathcal{L}_{\mathrm{sth}}$ measures the deviation of the manipulated asset from the original asset over benign inspection views. The feasible set $\mathcal{C}$ constrains the injected Gaussians to satisfy the prescribed opacity, injection-budget, and surface-placement restrictions.

4 PHANTOMSPLAT: STEALTHY POST-RECONSTRUCTION 3DGS MANIPULATION

4.1 OVERVIEW

PhantomSplat injects a small set of adversarial Gaussians Φ into an optimized 3DGS asset $\mathcal{G}$ while keeping the original Gaussians fixed (Figure 2). The framework has two main components: First, low-opacity Gaussians are initialized near the target surface, with geometry fixed and only opacity and appearance optimized to induce the desired target-view effect. Second, benign-view consistency is enforced through a first-order constraint with gradients precomputed from clean reference renderings. To optimize efficiently under these constraints, we develop a first-order approximation of alpha compositing that decouples the injected Gaussians from the frozen scene, enabling cached scene contributions and avoiding repeated rendering and backpropagation through the full asset. Pseudocode for the complete pipeline is provided in Appendix B.1.

4.2 SURFACE-CONSTRAINED LOW-OPACITY INJECTION

Given a target region on the clean 3DGS asset, we identify the corresponding object-associated Gaussians and construct a closed surface hull from their centers. The centers of the adversarial Gaussians are sampled on this hull using area-weighted barycentric sampling and offset along the surface normals to form multiple layers. This initialization keeps the injected Gaussians close to the existing scene geometry.

Let $\Phi = \{\phi_i\}_{i=1}^{m}$ denote the injected Gaussians, with $m \ll |\mathcal{G}|$. Each Gaussian is initialized with small scale, low opacity $o_i^0 < 0.01$, and zero appearance features. During optimization, we impose

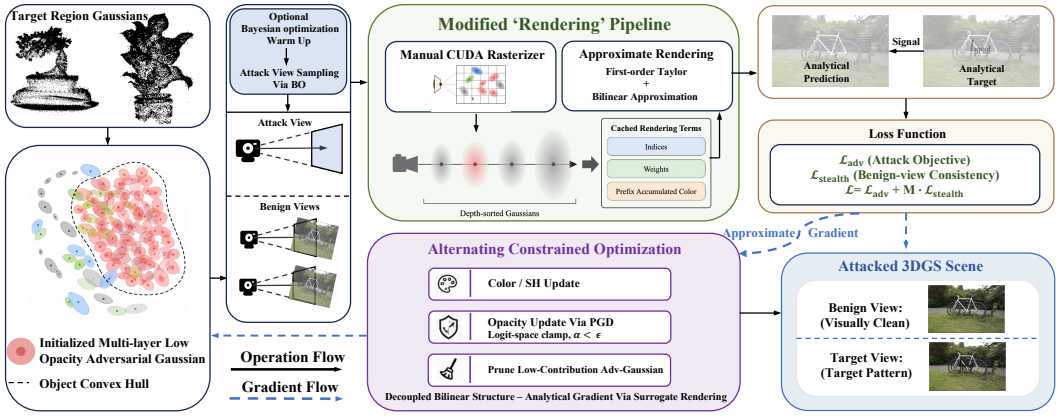

Figure 2: Overview of PhantomSplat. Given a target region and victim viewpoint, PhantomSplat initializes a small set of surface-constrained low-opacity adversarial Gaussians and optimizes their appearance and opacity using gradients derived from a first-order approximation of alpha compositing. When the victim viewpoint is not specified, an optional viewpoint-search procedure can be applied.

the low-opacity constraint $0 \le o_i \le o_{\max}$. After initialization, position, scale, and rotation are fixed, while only opacity and appearance are optimized. This surface-constrained, low-opacity design reduces conspicuous artifacts and enables the first-order rendering approximation introduced next.

4.3 Decoupled Rendering and Benign-View Consistency

For a fixed camera pose and pixel p, let $\mathcal{I}_{\text{inj}}$ and $\mathcal{I}_{\text{base}}$ denote the depth-ordered injected and original Gaussians contributing to the pixel. Because the original Gaussians remain fixed throughout optimization, their rendering contributions can be precomputed. Moreover, since each injected Gaussian satisfies $o_i \le o_{\max}$, its effective pixel opacity remains small. We exploit these properties to derive a first-order rendering surrogate that separates the dynamic contribution of the injected Gaussians from the frozen clean scene. For readability, we suppress the dependence on p and the camera pose below.

First-order decoupled rendering. Standard alpha compositing uses the accumulated transmittance

$$T_i = \prod_{j<i} (1 - \alpha_j). \tag{2}$$

where $j < i$ indicates that Gaussian j lies in front of Gaussian i. For the low-opacity injected Gaussians, we approximate their contribution to the transmittance to first order:

$$\prod_{\substack{j \in \mathcal{I}_{\text{inj}} \\ j<i}} (1 - \alpha_j) \approx 1 - \sum_{\substack{j \in \mathcal{I}_{\text{inj}} \\ j<i}} \alpha_j, \tag{3}$$

This approximation neglects products of two or more injected opacities within the transmittance factor. The transmittance due to the original Gaussians is retained exactly:

$$T_i^{\text{base}} = \prod_{\substack{j \in \mathcal{I}_{\text{base}} \\ j<i}} (1 - \alpha_j). \tag{4}$$

Because the original Gaussians and the geometry of the injected Gaussians remain fixed, T_i^{base} is constant throughout optimization.

For each Gaussian $k \in \mathcal{I}_{base}$, define its clean-scene color contribution as

$$q_k^{\text{base}} = T_k^{\text{base}} \alpha_k c_k. \tag{5}$$

The clean rendered color and the original-scene contribution behind injected Gaussian i are then

$$C_{\text{base}} = \sum_{k \in \mathcal{I}_{\text{base}}} q_k^{\text{base}}, \qquad \Gamma_i = \sum_{\substack{k \in \mathcal{I}_{\text{base}} \\ k>i}} q_k^{\text{base}}. \tag{6}$$

Using these quantities, we approximate the synthesized pixel color as

$$C_p \approx C_{\text{base}} + \sum_{i \in \mathcal{I}_{\text{inj}}} \left[T_i^{\text{base}} \left(1 - \sum_{\substack{j \in \mathcal{I}_{\text{inj}} \\ j<i}} \alpha_j \right) \alpha_i c_i - \alpha_i \Gamma_i \right]. \tag{7}$$

The first term inside the brackets represents the injected Gaussian's color contribution, while the second accounts for its attenuation of the original scene behind it.

For a fixed camera pose, the depth ordering and all original-scene quantities are computed once and cached. A running prefix sum over the injected Gaussians then allows each affected pixel to be evaluated in $O(|\mathcal{I}_{\text{inj}}|)$ time without repeatedly traversing the full scene. The full derivation is provided in Appendix B.4.

Benign-view consistency and constrained optimization. We optimize the injected Gaussians to match the target image from the victim viewpoint using

$$\mathcal{L}_{\text{adv}} = (1-\lambda)\mathcal{L}_1(I_{\text{vic}}, I_{\text{trg}}) + \lambda \mathcal{L}_{\text{D-SSIM}}(I_{\text{vic}}, I_{\text{trg}}), \tag{8}$$

where I_{vic} is computed using the rendering surrogate in Eq. 7.

To preserve the clean appearance under benign views without repeatedly rendering them, we approximate changes in the benign-view consistency loss using a first-order Taylor expansion around the initialization Φ^0. For each benign camera pose $P \in \mathcal{P}_{\text{ben}}$, the estimated loss change is

$$\Delta\widetilde{\mathcal{L}}_{\text{sth}}^P(\Phi) = \sum_{i=1}^{m} \left[g_{o,i}^P (o_i - o_i^0) + (g_{c,i}^P)^\top (c_i - c_i^0) \right], \tag{9}$$

where $g_{o,i}^P$ and $g_{c,i}^P$ are the gradients of the benign-view consistency loss with respect to opacity and appearance, respectively. These gradients are precomputed at Φ^0, with the clean rendering $\mathcal{R}(\mathcal{G}, P)$ as the reference image.

We penalize estimated loss increases that exceed the prescribed stealth tolerance ϵ_{sth}:

$$\mathcal{R}_{\text{sth}}(\Phi) = M \sum_{P \in \mathcal{P}_{\text{ben}}} \max\left(0,\, \Delta\widetilde{\mathcal{L}}_{\text{sth}}^P(\Phi) - \epsilon_{\text{sth}}\right), \tag{10}$$

where $M > 0$ controls the penalty strength.

The resulting inner-loop problem is

$$\begin{aligned} \min_{\Phi} \quad & \mathcal{J}_{\text{inner}}(\Phi) = \mathcal{L}_{\text{adv}}(\Phi) + \mathcal{R}_{\text{sth}}(\Phi) \\ \text{s.t.} \quad & 0 \le o_i \le o_{\max}, \qquad i = 1, \dots, m. \end{aligned} \tag{11}$$

We alternate between appearance and opacity updates, projecting each opacity onto $[0, o_{\max}]$ after every opacity update. Together, the rendering surrogate and cached benign-view gradients allow PhantomSplat to optimize the injected Gaussians without repeated rendering and backpropagation through the full scene during the inner loop.

Optional victim-viewpoint search. Our primary formulation assumes a specified victim viewpoint P_{vic}. When this viewpoint is not specified, PhantomSplat treats P_{vic} as an outer-loop variable and performs Bayesian optimization over a predefined camera domain. Each candidate pose is evaluated by running the inner optimization described above. Further details are provided in Appendix B.2.

5 Experiments

5.1 Experimental Setup

Dataset. To evaluate the performance of our method, we conduct experiments on three datasets: Mip-NeRF360 (Barron et al., 2021), Tanks & Temples (Knapitsch et al., 2017), and Free (Wang et al., 2023). Together, these datasets cover complex 360° scenes, real-world indoor and outdoor environments, and unbounded settings with freely moving cameras.

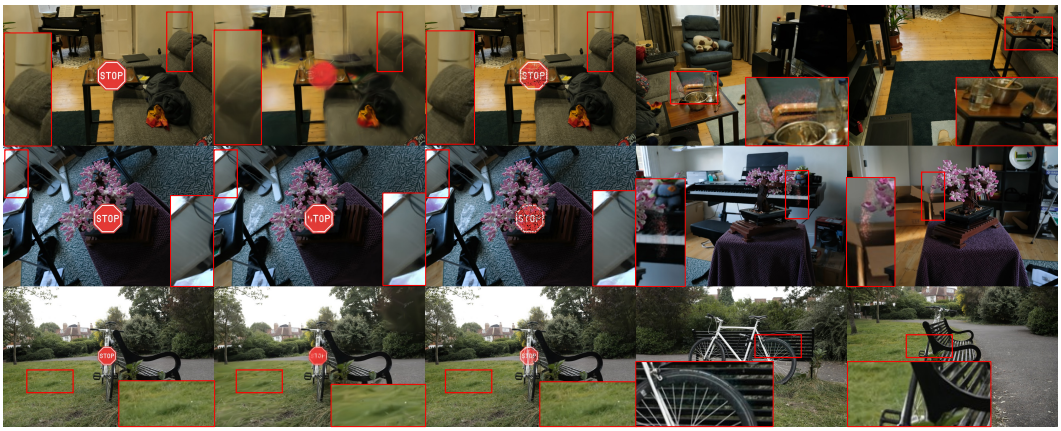

Figure 3: Qualitative comparisons on three scenes (Bike, Bonsai, and Room). The target image is constructed by overlaying the target object onto the clean rendering at the attack view. Our method generates clear target-view results while preserving clean-view fidelity, whereas StealthAttack introduces severe scene degradation. We also evaluate other signs and different attack targets, with details provided in the Appendix D.5.

Evaluation Metrics. Following StealthAttack (Ke et al., 2025), we evaluate image-space quality using PSNR, SSIM (Wang et al., 2004), and LPIPS (Zhang et al., 2018). We report these metrics on three view sets: (1) **Attack view**: the selected victim viewpoint, where the metrics measure reconstruction fidelity between the attacked rendering and the target image. (2) **Attacked training views**: the manipulated 3DGS rendered at the original training camera poses. In our evaluation, these poses instantiate the benign-view set P_{ben} and are used to measure benign-view fidelity. We regard the attack as preserving benign-view fidelity when the PSNR remains above 25 dB and drops by no more than 3 dB relative to the clean asset. (3) **Clean training-view references**: the clean 3DGS rendered at the same training camera poses, used as the reference for measuring degradation after the attack.

Since pixel-level metrics may not fully reflect semantic attack effectiveness, we further adopt two downstream semantic metrics: normalized CLIP (Radford et al., 2021) similarity and YOLO (Redmon et al., 2016) detection confidence:

$$R_{\text{CLIP}} = \frac{\text{CLIP}(I_{\text{render}}, T)}{\text{CLIP}(I_{\text{target}}, T)}, \qquad C_{\text{YOLO}} = \max_{b \in \mathcal{B}_c} p_{\text{YOLO}}(b, c), \tag{12}$$

where I_{render} is the rendered attack view, I_{target} is the target attack view, T is the text prompt, and C_{YOLO} denotes the maximum YOLO confidence for the target class c. All measurements are conducted using a standard stop sign with a standard prompt, while in the case studies we also use other symbols to evaluate the capability of our model. Higher CLIP ratios and YOLO scores indicate better preservation of the target semantics and therefore provide complementary measures of attack effectiveness.

Baseline. To our knowledge, no prior 3DGS attack directly targets an already optimized representation in the post-reconstruction setting. We therefore consider two complementary baselines: StealthAttack (Ke et al., 2025), a representative reconstruction-stage poisoning attack used as a cross-setting reference, and a direct post-reconstruction optimization baseline that optimizes the same injected Gaussians with the standard 3DGS renderer and optimizer. The latter provides a controlled comparison for evaluating the benefit of our rendering surrogate and Taylor-based stealth constraint. Implementation details are provided in Appendix C.1.

5.2 Attack Effectiveness

As shown in Table 2, our attack achieves strong performance across diverse scenes, with complete results reported in Appendix D. Compared with StealthAttack, our method reduces the average pipeline/optimization time from 135.35 minutes to 4.25 minutes, while improving the average attack-view PSNR from 27.37 to 33.96. Meanwhile, the training-view PSNR only slightly decreases from 30.36 to 29.62, indicating that our attack preserves the original scene quality. This efficiency gain

Table 2: Comparison of attack effectiveness, stealthiness, and average optimization time on Mip-NeRF360 scenes. "Attack View" and "Training View" report PSNR; higher PSNR indicates better reconstruction quality, while lower time is better. "w/o Decoupling" denotes directly optimizing the injected Gaussians through the standard 3DGS renderer without our decoupled rendering surrogate.

Method	Type	Avg. Time↓	BI.	FL.	GA.	ST.	TR.	RO.	CO.	KI.	BO.	Avg
3DGS	Training View PSNR ↑	17.04 min	26.59	24.31	30.29	31.33	24.82	35.73	31.39	34.45	34.37	30.36
StealthAttack	Attack View PSNR ↑	135.35 min	26.14	22.23	26.78	27.18	23.60	22.52	28.86	27.66	**32.25**	27.37
	Training View PSNR ↑		25.15	17.77	6.89	20.35	19.32	25.21	26.70	24.72	31.41	23.35
PhantomSplat w/o	Attack View PSNR ↑	83.64 min	29.63	33.05	32.11	31.89	28.90	27.77	27.50	27.81	28.02	29.68
	Training View PSNR ↑		24.73	22.58	28.41	28.96	23.37	30.81	29.24	30.55	32.36	27.89
PhantomSplat	Attack View PSNR ↑	**4.25 min**	**34.09**	**38.62**	**37.06**	**36.85**	**33.65**	**31.69**	**31.40**	**31.62**	30.70	**33.96**
	Training View PSNR ↑		**26.53**	**24.12**	**30.17**	**30.79**	**24.65**	**32.84**	**30.90**	**32.60**	**33.97**	**29.62**

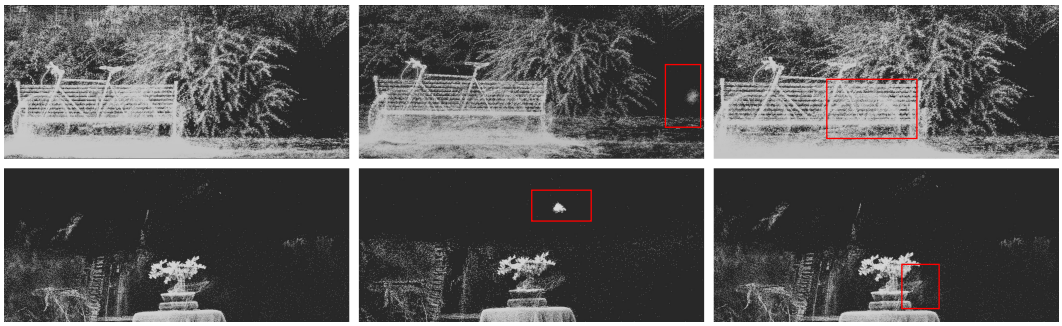

Figure 4: Point-cloud visualizations of the original 3DGS, StealthAttack, and our method. Our adversarial Gaussians are distributed around the object and remain difficult to detect, whereas StealthAttack produces isolated point clusters that are more easily identified.

comes from avoiding StealthAttack's expensive voxel-grid construction, which projects each SfM point onto all cameras to identify density regions. Instead, our method updates adversarial Gaussians using approximate gradients without repeated full-scene rendering during inner-loop optimization.

5.3 Attack Stealthiness

We visualize representative benign views in Fig. 3, where our low-opacity adversarial Gaussians remain barely perceptible even from nearby viewpoints. Point-cloud visualizations in Fig. 4 further assess stealthiness in 3D space. We also apply a DBSCAN (Ester et al., 1996)-based defense to remove small disconnected Gaussian clusters; Fig. 5 shows the removed adversarial Gaussians and the resulting visual artifacts after filtering. Overall, our attack causes less degradation to the original reconstruction and leaves weaker cues for detection. Appendix D.7 further shows that the adversarial effect is localized around the optimized viewpoint and diminishes as the camera moves away.

5.4 Impact on Downstream Models

We consider two representative downstream victim models: CLIP (Radford et al., 2021) and YOLO (Redmon et al., 2016). For these two models, we summarize the quantitative results in Table 3. We observe that even when the inserted target image does not perfectly reproduce the

DBSCAN Filtering Results

Method	Before	After	Δ	Adv. Rem.
StealthAttack	6328	329	5999	5.2%
PhantomSplat	3989	3853	136	96.6%

Before/After denote the number of adversarial Gaussians before and after DBSCAN filtering. Δ denotes the decrease in the number after DBSCAN filtering.

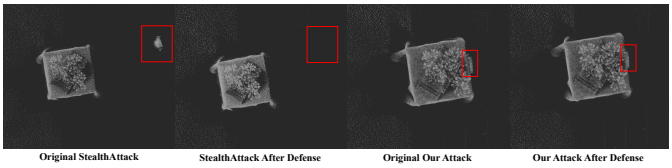

Figure 5: Robustness against DBSCAN-based cluster filtering on the Mip-NeRF360 dataset. The table reports the number of adversarial Gaussians before and after filtering, while the right visualization shows the filtered point-cloud comparison on the Bonsai scene.

Table 3: Comparison of semantic attack effectiveness on Mip-NeRF360 scenes. Higher values indicate stronger preservation of the target semantics. N.D. denotes no detection of the target class.

Method	Type	BI.	FL.	GA.	ST.	TR.	RO.	CO.	KI.	BO.	Avg
Target Image	CLIP	0.265	0.286	0.307	0.291	0.296	0.230	0.238	0.217	0.291	0.269
	YOLO	0.924	0.927	0.936	0.918	0.937	0.963	0.977	0.966	0.961	0.946
StealthAttack Ke et al. (2025)	CLIP Ratio ↑	0.844	1.007	1.081	**0.969**	**1.021**	0.748	1.034	**0.986**	**0.935**	0.950
	YOLO ↑	0.879	0.913	**0.932**	0.888	0.918	*N.D.*	0.971	**0.937**	**0.937**	0.922
PhantomSplat	CLIP Ratio ↑	**0.990**	**1.020**	**1.090**	0.960	1.001	**1.011**	**1.052**	0.913	0.761	**0.982**
	YOLO ↑	**0.933**	**0.920**	0.823	**0.937**	**0.950**	**0.964**	**0.975**	0.928	0.892	**0.925**

original image, it can still induce strong responses in downstream perception models. The complete downstream-task results are provided in the Appendix D.3.

5.5 Ablation and Sensitivity Analysis

We use the Mip-NeRF360 and conduct ablation studies to evaluate the main design choices of PhantomSplat. For optimization, *Direct Opt* directly updates the inserted Gaussians through the standard 3DGS renderer with Adam (Kingma & Ba, 2017). *Decoupled Opt* uses our cached first-order approximation to avoid repeated full-scene rendering, and *Decoupled + Taylor* further incorporates the Taylor-based stealth constraint to limit clean-view degradation. We also ablate the opacity bound ϵ and the number of adversarial Gaussian layers, both of which balance attack strength against clean-view stealthiness. As shown in Table 4, our default setting achieves the best overall trade-off among attack effectiveness, clean-view fidelity, and efficiency.

Although *Direct Opt* uses the exact 3DGS renderer, it is slow and can be difficult to optimize due to the highly coupled alpha-compositing process of low-opacity Gaussians. In contrast, our approximate renderer is used as a render-free optimization surrogate: the full renderer is not invoked during inner-loop training, and the approximate objective provides a smoother target signal for updating the injected primitives. We provide an approximation error analysis in Appendix D.6, showing that this surrogate maintains a controlled discrepancy from the true rendering pipeline during optimization.

Table 4: Ablation studies on key components and hyperparameters of PhantomSplat. A.PSNR and T.PSNR denote attack-view and training-view PSNR, respectively. Bold entries indicate the default configuration used in our main experiments.

(a) Optimization strategy.

Method	A.PSNR↑	T.PSNR↑	Time↓
Direct Opt	29.68	27.89	83.64
Decoupled Opt	34.42	28.47	24.28
Decoupled + Taylor	33.96	29.62	4.25

(b) Maximum opacity ϵ.

ϵ	A.PSNR↑	T.PSNR↑	T.SSIM↑	T.LPIPS↓
0.01	28.73	30.12	0.9041	0.0724
0.05	33.96	29.62	0.9041	0.0725
0.10	35.18	28.13	0.8876	0.0832

(c) Number of layers.

Layers	A.PSNR↑	T.PSNR↑	T.SSIM↑	T.LPIPS↓
4	30.02	29.68	0.9040	0.0832
8	33.96	29.62	0.9041	0.0725
12	34.42	27.48	0.8729	0.0932

6 Conclusion

We propose PhantomSplat, a post-reconstruction parameter-space attack on 3D Gaussian Splatting representations. Unlike reconstruction-time data-poisoning attacks, PhantomSplat directly manipulates an already optimized 3DGS asset by injecting a small set of low-opacity adversarial Gaussians while keeping the original scene representation fixed. By combining surface-constrained initialization with render-free inner-loop optimization based on cached first-order approximations and Taylor-based clean-view constraints, PhantomSplat induces target-view adversarial effects while preserving clean-view fidelity. Experiments across multiple scenes and downstream perception models show that PhantomSplat achieves strong attack-view reconstruction and semantic attack effects with substantially lower optimization time and fewer isolated geometric artifacts than the strong data-poisoning baseline. These results highlight released 3DGS assets as a practical post-reconstruction attack surface beyond training-data poisoning and reconstruction-time threats. While our evaluation focuses on selected victim viewpoints and standard outlier-based defenses, extending the analysis to broader camera trajectories and stronger asset-level detectors remains an important direction for future work. We hope this work motivates stronger verification and detection methods for trojaned 3DGS representations.

AI USE STATEMENT

In this work, we used generative AI tools to assist with debugging and editing research code, interpreting and presenting experimental results, and editing the manuscript, including formatting and figure preparation. All AI-assisted suggestions, derivations, code changes, and written content were reviewed and verified by the authors. The authors take full responsibility for the final content of this work, including all technical claims, experimental results, and artifacts produced with the aid of generative AI.

REFERENCES

Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields, 2021. URL https://arxiv.org/abs/2103.13415.

Yiwen Chen, Zilong Chen, Chi Zhang, Feng Wang, Xiaofeng Yang, Yikai Wang, Zhongang Cai, Lei Yang, Huaping Liu, and Guosheng Lin. Gaussianeditor: Swift and controllable 3d editing with gaussian splatting. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 21476–21485, 2024.

Tianchen Deng, Xuefeng Chen, Yi Chen, Qu Chen, Yuyao Xu, Lijin Yang, Le Xu, Yu Zhang, Bo Zhang, Wuxiong Huang, et al. Gaussiandwm: 3d gaussian driving world model for unified scene understanding and multi-modal generation. *arXiv preprint arXiv:2512.23180*, 2025.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pp. 226–231. AAAI Press, 1996.

Yonggan Fu, Ye Yuan, Souvik Kundu, Shang Wu, Shunyao Zhang, and Yingyan Celine Lin. Nerfool: Uncovering the vulnerability of generalizable neural radiance fields against adversarial perturbations. *arXiv preprint arXiv:2306.06359*, 2023.

Xiaoshen Han, Minghuan Liu, Yilun Chen, Junqiu Yu, Xiaoyang Lyu, Yang Tian, Bolun Wang, Weinan Zhang, and Jiangmiao Pang. Re^{3}sim: Generating high-fidelity simulation data via 3d-photorealistic real-to-sim for robotic manipulation, 2025. URL https://arxiv.org/abs/2502.08645.

Jiaxin Hong, Sixu Chen, Shuoyang Sun, Hongyao Yu, Hao Fang, Yuqi Tan, Bin Chen, Shuhan Qi, and Jiawei Li. Gausstrap: Stealthy poisoning attacks on 3d gaussian splatting for targeted scene confusion, 2025. URL https://arxiv.org/abs/2504.20829.

András Horváth and Csaba M Józsa. Targeted adversarial attacks on generalizable neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3718–3727, 2023.

Matthew Hull, Haoyang Yang, Pratham Mehta, Mansi Phute, Aeree Cho, Haorang Wang, Matthew Lau, Wenke Lee, Wilian Lunardi, Martin Andreoni, and Duen Horng Chau. Complicitsplat: Downstream models are vulnerable to blackbox attacks by 3d gaussian splat camouflages, 2025. URL https://arxiv.org/abs/2508.11854.

Wenxiang Jiang, Hanwei Zhang, Xi Wang, Zhongwen Guo, and Hao Wang. Nerfail: Neural radiance fields-based multiview adversarial attack. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 21197–21205, 2024a.

Wenxiang Jiang, Hanwei Zhang, Shuo Zhao, Zhongwen Guo, and Hao Wang. Ipa-nerf: Illusory poisoning attack against neural radiance fields. *arXiv preprint arXiv:2407.11921*, 2024b.

Bo-Hsu Ke, You-Zhe Xie, Yu-Lun Liu, and Wei-Chen Chiu. Stealthattack: Robust 3d gaussian splatting poisoning via density-guided illusions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 27400–27411, 2025.

Nikhil Keetha, Jay Karhade, Krishna Murthy Jatavallabhula, Gengshan Yang, Sebastian Scherer, Deva Ramanan, and Jonathon Luiten. Splatam: Splat, track & map 3d gaussians for dense rgb-d slam. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), July 2023. URL https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.

Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics*, 36(4), 2017.

Yang Liu, He Guan, Chuanchen Luo, Lue Fan, Naiyan Wang, Junran Peng, and Zhaoxiang Zhang. Citygaussian: Real-time high-quality large-scale scene rendering with gaussians, 2024. URL https://arxiv.org/abs/2404.01133.

Tianrui Lou, Xiaojun Jia, Siyuan Liang, Jiawei Liang, Ming Zhang, Yanjun Xiao, and Xiaochun Cao. 3d gaussian splatting driven multi-view robust physical adversarial camouflage generation, 2025. URL https://arxiv.org/abs/2507.01367.

Jiahao Lu, Yifan Zhang, Qiuhong Shen, Xinchao Wang, and Shuicheng Yan. Poison-splat: Computation cost attack on 3d gaussian splatting, 2025. URL https://arxiv.org/abs/2410.08190.

Jonathon Luiten, Georgios Kopanas, Bastian Leibe, and Deva Ramanan. Dynamic 3d gaussians: Tracking by persistent dynamic view synthesis. In *2024 International Conference on 3D Vision (3DV)*, pp. 800–809. IEEE, 2024.

Lingzhuang Meng, Mingwen Shao, Yuanjian Qiao, Wenjie Liu, and Xiang Lv. Gaussian splitting attack: Gaussian splatting-based multi-view 3d adversarial attack. *Pattern Recognition*, 172: 112466, 2026. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2025.112466. URL https://www.sciencedirect.com/science/article/pii/S003132032501129X.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis, 2020. URL https://arxiv.org/abs/2003.08934.

Hanieh Naderi and Ivan V Bajić. Adversarial attacks and defenses on 3d point cloud classification: A survey. *IEEE Access*, 11:144274–144295, 2023.

Yiran Qiao, Yiren Lu, Yunlai Zhou, Rui Yang, Linlin Hou, Yu Yin, and Jing Ma. Advsplat: Adversarial attacks on feed-forward gaussian splatting models. *arXiv preprint arXiv:2603.23686*, 2026.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021. URL https://arxiv.org/abs/2103.00020.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016. URL https://arxiv.org/abs/1506.02640.

Matthew Tancik, Vincent Casser, Xinchen Yan, Sabeek Pradhan, Ben Mildenhall, Pratul P. Srinivasan, Jonathan T. Barron, and Henrik Kretzschmar. Block-nerf: Scalable large scene neural view synthesis, 2022. URL https://arxiv.org/abs/2202.05263.

Peng Wang, Yuan Liu, Zhaoxi Chen, Lingjie Liu, Ziwei Liu, Taku Komura, Christian Theobalt, and Wenping Wang. F^2-nerf: Fast neural radiance field training with free camera trajectories, 2023. URL https://arxiv.org/abs/2303.15951.

Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004. doi: 10.1109/TIP.2003.819861.

Guanjun Wu, Taoran Yi, Jiemin Fang, Lingxi Xie, Xiaopeng Zhang, Wei Wei, Wenyu Liu, Qi Tian, and Xinggang Wang. 4d gaussian splatting for real-time dynamic scene rendering, 2024. URL https://arxiv.org/abs/2310.08528.

Chong Xiang, Charles R Qi, and Bo Li. Generating 3d adversarial point clouds. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 9136–9144, 2019.

Abdurrahman Zeybey, Mehmet Ergezer, and Tommy Nguyen. Gaussian splatting under attack: Investigating adversarial noise in 3d objects, 2024. URL https://arxiv.org/abs/2412.02803.

Jinlai Zhang, Lyujie Chen, Binbin Liu, Bo Ouyang, Qizhi Xie, Jihong Zhu, Weiming Li, and Yanmei Meng. 3d adversarial attacks beyond point cloud. *Information Sciences*, 633:491–503, 2023.

Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric, 2018. URL https://arxiv.org/abs/1801.03924.

Yixun Zhang, Lizhi Wang, Junjun Zhao, Wending Zhao, Feng Zhou, Yonghao Dang, and Jianqin Yin. 3dgaa: Realistic and robust 3d gaussian-based adversarial attack for autonomous driving, 2025. URL https://arxiv.org/abs/2507.09993.

Wenzhao Zheng, Junjie Wu, Yao Zheng, Sicheng Zuo, Zixun Xie, Longchao Yang, Yong Pan, Zhihui Hao, Peng Jia, Xianpeng Lang, and Shanghang Zhang. Gaussianad: Gaussian-centric end-to-end autonomous driving, 2024. URL https://arxiv.org/abs/2412.10371.

A More Background Information

A.1 3D Gaussian Splatting

3D Gaussian Splatting (3DGS) (Kerbl et al., 2023) represents a scene using a set of explicit Gaussian primitives $\mathcal{G} = \{G_i\}_{i=1}^{n}$. Each Gaussian is parameterized by its center μ_i, covariance Σ_i, opacity o_i, and appearance features c_i. Given a camera pose P, the Gaussians are projected onto the image plane, ordered by depth, and alpha-composited to render each pixel. Specifically, the color at pixel p is

$$C(p) = \sum_{i \in \mathcal{N}(p)} T_i(p)\, \alpha_i(p)\, c_i(P), \qquad T_i(p) = \prod_{j<i} \left(1 - \alpha_j(p)\right), \tag{13}$$

where $\mathcal{N}(p)$ denotes the depth-ordered set of Gaussians contributing to pixel p, $\alpha_i(p)$ denotes the effective opacity of Gaussian i at pixel p, determined by its learned opacity o_i and projected Gaussian footprint, and $c_i(P)$ denotes its view-dependent appearance under camera pose P. The transmittance $T_i(p)$ accounts for the accumulated attenuation caused by Gaussians preceding i along the viewing direction.

The Gaussian parameters are optimized from multi-view observations by minimizing reconstruction error between rendered and observed images, typically using a combination of $\mathcal{L}_1$ and D-SSIM losses Wang et al. (2004). Because multiple depth-ordered Gaussians jointly determine each pixel and their contributions vary with viewpoint, modifications to a small subset of primitives can induce view-dependent changes in the rendered appearance.

To fully articulate the structural consistency enforced during optimization, it is necessary to explicitly decompose the D-SSIM formulation. For two local image windows x and y (extracted from the rendered image V_k' and ground truth V_k, respectively), the base SSIM index is explicitly composed of three comparative metrics: luminance $l(x, y)$, contrast $c(x, y)$, and structure $s(x, y)$. The generalized formula is:

$$\text{SSIM}(x, y) = [l(x, y)]^{\alpha} \cdot [c(x, y)]^{\beta} \cdot [s(x, y)]^{\gamma} \tag{14}$$

Under the standard configuration of 3DGS, the weights are set to $\alpha = \beta = \gamma = 1$. This simplifies the combination into the following explicit differentiable equation:

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{15}$$

where the terms are defined as follows:

- **Luminance:** μ_x and μ_y denote the local pixel means of the respective windows.
- **Contrast:** σ_x and σ_y denote the local pixel standard deviations.
- **Structure:** σ_{xy} represents the covariance between x and y.
- **Stability Constants:** C_1 and C_2 are small constants included to prevent numerical instability when the denominator approaches zero.

The structural loss term $\mathcal{L}_{\text{D-SSIM}}$ leverages this formulation (typically evaluated as $1 - \text{SSIM}(x, y)$ or $(1 - \text{SSIM}(x, y))/2$) to preserve high-frequency structural details that might otherwise be smoothed over by the $\mathcal{L}_1$ loss alone.

The reconstruction of a 3D scene is formulated as an optimization problem, where the parameters of the Gaussian set $\mathcal{G}$ are refined by minimizing the discrepancy between the rendered image V'_k and the ground truth V_k across N training views. The objective function is defined as a weighted combination of $\mathcal{L}_1$ loss and structural similarity (SSIM) loss:

$$\min_{\mathcal{G}} \mathcal{L} = \sum_{k=1}^{N} \left[(1 - \lambda)\mathcal{L}_1(V'_k, V_k) + \lambda \mathcal{L}_{\text{D-SSIM}}(V'_k, V_k) \right], \tag{16}$$

where λ is a hyperparameter balancing per-pixel accuracy and structural consistency.

Furthermore, 3DGS periodically (e.g., 100 iterations) incorporates an adaptive density control strategy to dynamically adjust the Gaussian population $\mathcal{G}$. When the view-space positional gradient $\nabla_{\boldsymbol{\mu}}$ exceeds a predefined threshold τ_g, the representation is refined through densification (splitting or cloning) to capture missing details. Conversely, Gaussians with negligible opacity ($\alpha_i < \tau_\alpha$) are pruned to eliminate redundancy. This adaptive mechanism significantly enhances reconstruction quality while minimizing computational overhead, thereby facilitating high-fidelity real-time rendering.

B Supplementary Details for the Methodology

B.1 Pseudocode

The detailed pseudocode of this method is provided in Algorithm 1.

B.2 Full Algorithm

In this section, we mainly describe the derivation and justification of the efficient inner-loop computation, as well as the output view sampling strategy.

B.2.1 Efficient Inner-Loop Optimization

Decoupled Analytical Approximation for Alpha Blending. Repeatedly rendering the frozen scene is a major computational bottleneck during optimization. For a fixed camera pose and pixel p, let $\mathcal{I}_{\text{inj}}$ and $\mathcal{I}_{\text{base}}$ denote the injected and original Gaussians contributing to the pixel, respectively. All indices follow a shared front-to-back depth ordering. We suppress the dependence on the pixel and camera pose below.

Standard alpha compositing uses the accumulated transmittance

$$T_i = \prod_{j<i} (1 - \alpha_j). \tag{17}$$

For each injected Gaussian, the effective pixel opacity is $\alpha_i = o_i w_i$, where $w_i \in [0, 1]$ is its fixed projected Gaussian weight. The constraint $0 \le o_i \le o_{\max}$ therefore implies $0 \le \alpha_i \le o_{\max}$. We exploit these small injected opacities to approximate their transmittance contribution to first order, while retaining the original-scene transmittance exactly:

$$T_i \approx T_i^{\text{base}} \left(1 - \sum_{\substack{j \in \mathcal{I}_{\text{inj}} \\ j<i}} \alpha_j \right), \qquad T_i^{\text{base}} = \prod_{\substack{j \in \mathcal{I}_{\text{base}} \\ j<i}} (1 - \alpha_j). \tag{18}$$

Algorithm 1 The PhantomSplat Attack Framework

Input: Pre-trained 3DGS $\mathcal{I}_{large}$, target convex hull $\mathcal{V}_{hull}$, benign views Θ_{ben}, target artifact I_{trg}, iteration limits K_{outer}, T_{inner}, stealth tolerance ξ, penalty M, bound ϵ.
Output: Optimized adversarial parameters $\Phi^* = \{(\alpha_i, c_i)\}$, Optimal victim angle θ^*_{vic}.

// Phase 1: Initialization & Global Precomputation
Initialize adversarial Gaussians $\mathcal{I}_{small}$ within $\mathcal{V}_{hull}$
Set initial parameters $\Phi^{(0)}$: $\alpha_i < 0.01$, $c_i \leftarrow 0$, positions fixed
Execute full forward/backward pass on Θ_{ben} using original model
Precompute and cache initial gradients: $g_{\alpha,i} \leftarrow \nabla_{\alpha_i}\mathcal{L}|_{\Phi^{(0)}}$, $g_{c,i} \leftarrow \nabla_{c_i}\mathcal{L}|_{\Phi^{(0)}}$
Initialize Gaussian Process (GP) surrogate model for Bayesian Optimization
// Phase 2: Outer-Loop (Victim Angle Search)
for $k = 1, 2, \ldots, K_{outer}$ **do**
  Sample candidate angle $\theta_{vic} = \arg\min_\theta$ Acquisition(θ) on GP
  // Phase 3: View-Specific Precomputation
  Render $\mathcal{I}_{large}$ at θ_{vic} to precompute constants: $C_{base}, \Gamma_i, Const1_i$
  // Phase 4: Inner-Loop (Alternating Projected Gradient Descent)
  for $t = 1, 2, \ldots, T_{inner}$ **do**
    Stealth Check: $\Delta\tilde{\mathcal{L}}(\Phi) \approx \sum_i \left[g_{\alpha,i}(\alpha_i - \alpha_i^{(0)}) + g_{c,i}(c_i - c_i^{(0)}) \right]$
    Compute penalty indicator: $\mathbb{I}_{sth} \leftarrow \mathbb{I}(\Delta\tilde{\mathcal{L}}(\Phi) > \xi)$
    Fast Rendering: Compute pixel colors C_p at θ_{vic} in $O(|\mathcal{I}_{small}|)$
    Update Color: $c_i^{(t+1)} \leftarrow c_i^{(t)} - \eta_c \nabla_{c_i}(\mathcal{L}_{adv} + M \cdot \mathbb{I}_{sth} \cdot \Delta\tilde{\mathcal{L}})$
    Update Opacity: $\alpha_i^{(t+1)} \leftarrow \alpha_i^{(t)} - \eta_\alpha \nabla_{\alpha_i}(\mathcal{L}_{adv} + M \cdot \mathbb{I}_{sth} \cdot \Delta\tilde{\mathcal{L}})$ using $c^{(t+1)}$
    Projection: $\alpha_i^{(t+1)} \leftarrow$ clamp$(\alpha_i^{(t+1)}, 0, \epsilon)$ ▷ Enforce opacity bound
  end for
  Evaluate true $\mathcal{L}_{adv}$ at θ_{vic} and update GP surrogate model
end for
return Φ^*, θ^*_{vic}

Because the original Gaussians and the geometry of the injected Gaussians remain fixed, T_i^{base} can be precomputed.

For each original Gaussian k, define its clean-scene color contribution as

$$q_k^{\text{base}} = T_k^{\text{base}} \alpha_k c_k. \tag{19}$$

Substituting the transmittance approximation into the rendering equation gives

$$\begin{aligned} C_p \approx & \sum_{i \in \mathcal{I}_{\text{inj}}} T_i^{\text{base}} \left(1 - \sum_{\substack{j \in \mathcal{I}_{\text{inj}} \\ j<i}} \alpha_j \right) \alpha_i c_i \\ & + \sum_{k \in \mathcal{I}_{\text{base}}} \left(1 - \sum_{\substack{j \in \mathcal{I}_{\text{inj}} \\ j<k}} \alpha_j \right) q_k^{\text{base}}. \end{aligned} \tag{20}$$

Although all original-scene quantities are fixed, evaluating the second sum directly would still require traversing the original Gaussians at every update. To avoid this cost, we precompute

$$C_{\text{base}} = \sum_{k \in \mathcal{I}_{\text{base}}} q_k^{\text{base}}, \qquad \Gamma_i = \sum_{\substack{k \in \mathcal{I}_{\text{base}} \\ k>i}} q_k^{\text{base}}, \tag{21}$$

where C_{base} is the clean rendered color and Γ_i collects the original-scene contribution behind injected Gaussian i.

Reordering the sums over the original and injected Gaussians yields the cached rendering surrogate

$$C_p \approx C_{\text{base}} + \sum_{i \in \mathcal{I}_{\text{inj}}} \left[T_i^{\text{base}} \left(1 - \sum_{\substack{j \in \mathcal{I}_{\text{inj}} \\ j<i}} \alpha_j \right) \alpha_i c_i - \alpha_i \Gamma_i \right]. \tag{22}$$

For a fixed camera pose, the depth ordering and all original-scene quantities are computed once and cached. A running prefix sum over the injected Gaussians then allows each affected pixel to be evaluated in $O(|\mathcal{I}_{\text{inj}}|)$ time without repeatedly traversing the original scene. This complexity applies to subsequent surrogate evaluations, rather than the initial precomputation.

Render-Free Stealth via Taylor Linearization: A second computational bottleneck is evaluating benign-view consistency throughout optimization. Direct evaluation would require repeatedly rendering the benign camera poses in $\mathcal{P}_{\text{ben}}$ and backpropagating their consistency losses. We avoid these repeated evaluations by caching a first-order approximation at the initialization Φ^0.

For each benign pose $P \in \mathcal{P}_{\text{ben}}$, let $\mathcal{L}^P_{\text{sth}}(\Phi)$ denote the benign-view consistency loss, using the clean rendering $\mathcal{R}(\mathcal{G}, P)$ as the reference image. Before the inner optimization begins, we compute

$$g^P_{o,i} = \nabla_{o_i} \mathcal{L}^P_{\text{sth}}(\Phi)\big|_{\Phi=\Phi^0}, \qquad g^P_{c,i} = \nabla_{c_i} \mathcal{L}^P_{\text{sth}}(\Phi)\big|_{\Phi=\Phi^0}, \tag{23}$$

where the gradients are taken with respect to the injected opacity and appearance parameters, respectively.

The change in benign-view consistency loss is approximated by

$$\mathcal{L}^P_{\text{sth}}(\Phi) - \mathcal{L}^P_{\text{sth}}(\Phi^0) \approx \Delta\widetilde{\mathcal{L}}^P_{\text{sth}}(\Phi), \tag{24}$$

with

$$\Delta\widetilde{\mathcal{L}}^P_{\text{sth}}(\Phi) = \sum_{i=1}^{m} \left[g^P_{o,i}(o_i - o_i^0) + (g^P_{c,i})^\top (c_i - c_i^0) \right]. \tag{25}$$

These cached gradients allow us to estimate benign-view loss changes without rendering or backpropagating through the benign views again during the inner loop.

The adversarial objective encourages the rendered image at the victim viewpoint to match the target image:

$$\mathcal{L}_{\text{adv}}(\Phi) = (1-\lambda)\mathcal{L}_1(I_{\text{vic}}, I_{\text{trg}}) + \lambda\mathcal{L}_{\text{D-SSIM}}(I_{\text{vic}}, I_{\text{trg}}), \tag{26}$$

where I_{vic} is computed using the rendering surrogate in Eq. (22.)

Rather than requiring exact reconstruction of every benign view, we allow an estimated loss increase of up to ϵ_{sth} relative to initialization. We penalize increases beyond this tolerance using

$$\mathcal{R}_{\text{sth}}(\Phi) = M \sum_{P \in \mathcal{P}_{\text{ben}}} \max\left(0,\, \Delta\widetilde{\mathcal{L}}^P_{\text{sth}}(\Phi) - \epsilon_{\text{sth}}\right), \tag{27}$$

where $M > 0$ controls the penalty strength. We use $M = 10^5$ in our experiments.

The penalty is zero when all estimated benign-view loss increases remain within the tolerance. When a view exceeds the tolerance, its active hinge term contributes a gradient scaled by M, discouraging further increases in the estimated loss. This is a soft penalty on a local approximation: it does not guarantee satisfaction of the tolerance for either the surrogate or the true benign-view loss.

The resulting inner-loop problem is

$$\begin{aligned} \min_{\Phi} \quad & \mathcal{J}_{\text{inner}}(\Phi) = \mathcal{L}_{\text{adv}}(\Phi) + \mathcal{R}_{\text{sth}}(\Phi) \\ \text{s.t.} \quad & 0 \le o_i \le o_{\max}, \qquad i = 1, \dots, m. \end{aligned} \tag{28}$$

Together, the cached rendering surrogate and benign-view gradients eliminate repeated full-scene rendering and backpropagation during the inner-loop updates.

Alternating Optimization via Projected Gradient Descent Opacity and appearance are coupled through the blending expression in Eq. 22, even though geometry remains fixed. We handle this coupling through alternating updates and explicitly enforce the opacity bounds through projection.

Let $\mathbf{o}$ and $\mathbf{c}$ collect the injected opacity and appearance parameters, respectively. At inner-loop iteration t, we first hold opacity fixed and update appearance:

$$\mathbf{c}^{(t+1)} = \mathbf{c}^{(t)} - \eta_c \nabla_{\mathbf{c}} \mathcal{J}_{\text{inner}}\left(\mathbf{o}^{(t)}, \mathbf{c}^{(t)}\right), \tag{29}$$

where $\eta_c > 0$ is the appearance learning rate.

We then hold the updated appearance fixed and take a projected opacity step:

$$\mathbf{o}^{(t+1)} = \Pi_{[0,o_{\max}]^m}\left[\mathbf{o}^{(t)} - \eta_o \nabla_{\mathbf{o}} \mathcal{J}_{\text{inner}}\left(\mathbf{o}^{(t)}, \mathbf{c}^{(t+1)}\right)\right], \tag{30}$$

where $\eta_o > 0$ is the opacity learning rate and $\Pi_{[0,o_{\max}]^m}$ clips each opacity to $[0, o_{\max}]$.

These alternating updates optimize the two parameter blocks sequentially while maintaining the prescribed opacity bounds. The projection guarantees the per-Gaussian opacity bound throughout optimization. The accuracy of the rendering surrogate additionally depends on the cumulative injected opacity along each ray remaining sufficiently small. Details of the gradient computation are provided in Appendix B.5.

B.3 Outer-Loop: Victim Angle Search

The pose-dependent attack objective is highly non-convex with respect to the camera pose P_{vic}, making local gradient-based viewpoint search sensitive to initialization and potentially leading to suboptimal viewpoints. We therefore use Bayesian optimization to search the low-dimensional camera domain without requiring gradients of the pose-dependent outer objective:

- **Surrogate Model:** We model the objective $f(P_{\text{vic}}) = \mathcal{L}_{adv}(\Phi, P_{\text{vic}})$ using a Gaussian Process (GP).
- **Acquisition Function:** We use an acquisition function, such as Expected Improvement (EI) or Upper Confidence Bound (UCB), to balance exploring new regions of the camera trajectory and exploiting angles known to yield high pixel-wise differences.
- **Iteration:** At each BO step, we sample $P_t = \arg\max_\theta \text{Acq}(P_{\text{vic}})$, evaluate the true $\mathcal{L}_{adv}$ by rendering the view, and update the GP until we identify the most vulnerable victim angle P^*_{vic}.

B.4 Details for Decoupled Analytical Approximation

For a fixed camera pose and pixel p, let $\mathcal{I}_{\text{inj}}$ and $\mathcal{I}_{\text{base}}$ denote the sets of injected and original Gaussians contributing to the pixel, respectively. All indices follow a shared front-to-back depth ordering, so $j < i$ indicates that Gaussian j lies in front of Gaussian i. We suppress the dependence on the pixel and camera pose where convenient.

Standard alpha compositing gives the pixel color

$$C_p = \sum_{i \in \mathcal{I}_{\text{inj}} \cup \mathcal{I}_{\text{base}}} T_i \alpha_i c_i, \qquad T_i = \prod_{j<i}(1-\alpha_j), \tag{31}$$

where α_i is the effective pixel opacity, c_i is the color for the fixed viewing direction, and T_i is the accumulated transmittance before Gaussian i. During the inner optimization, only the opacity and appearance of the injected Gaussians are updated. Their geometry and all parameters of the original Gaussians remain fixed.

Separating fixed and variable transmittance. We factor the accumulated transmittance into contributions from the original and injected Gaussians:

$$T_i = T_i^{\text{base}} U_i, \qquad T_i^{\text{base}} = \prod_{\substack{j \in \mathcal{I}_{\text{base}} \\ j<i}} (1-\alpha_j), \qquad U_i = \prod_{\substack{j \in \mathcal{I}_{\text{inj}} \\ j<i}} (1-\alpha_j). \tag{32}$$

The original-scene transmittance T_i^{base} is fixed and retained exactly. Only the injected transmittance U_i is approximated.

For an injected Gaussian, the effective pixel opacity is $\alpha_i = o_i w_i$, where $w_i \in [0,1]$ is its fixed projected Gaussian weight at the pixel. Thus, the opacity constraint $0 \le o_i \le o_{\max}$ implies $0 \le \alpha_i \le o_{\max}$. Define the cumulative injected opacity before Gaussian i as

$$S_i = \sum_{\substack{j \in \mathcal{I}_{\text{inj}} \\ j<i}} \alpha_j. \tag{33}$$

Expanding the product defining U_i gives

$$U_i = 1 - S_i + \mathcal{O}(S_i^2) \approx 1 - S_i. \tag{34}$$

This approximation discards products of two or more injected opacities within the transmittance factor, yielding

$$T_i \approx T_i^{\text{base}}(1 - S_i). \tag{35}$$

Its accuracy depends on the cumulative injected opacity along the ray and therefore on both the individual opacities and the number of overlapping injected Gaussians.

Collecting frozen-scene contributions. For each original Gaussian $k \in \mathcal{I}_{\text{base}}$, define its clean-scene color contribution as

$$q_k^{\text{base}} = T_k^{\text{base}} \alpha_k c_k. \tag{36}$$

The clean rendered color and the clean-scene contribution behind injected Gaussian i are

$$C_{\text{base}} = \sum_{k \in \mathcal{I}_{\text{base}}} q_k^{\text{base}}, \qquad \Gamma_i = \sum_{\substack{k \in \mathcal{I}_{\text{base}} \\ k > i}} q_k^{\text{base}}, \tag{37}$$

respectively. These quantities remain fixed throughout the inner optimization.

Substituting the transmittance approximation into the compositing equation gives $C_p \approx \widetilde{C}_p$, where

$$\widetilde{C}_p = \sum_{i \in \mathcal{I}_{\text{inj}}} T_i^{\text{base}}(1 - S_i)\alpha_i c_i + \sum_{k \in \mathcal{I}_{\text{base}}} (1 - S_k) q_k^{\text{base}}. \tag{38}$$

The first sum contains the injected Gaussians' color contributions. The second contains the original-scene contributions attenuated by the injected Gaussians in front of them. Reordering the finite sums in the second term yields

$$\begin{aligned} \sum_{k \in \mathcal{I}_{\text{base}}} (1 - S_k) q_k^{\text{base}} &= C_{\text{base}} - \sum_{k \in \mathcal{I}_{\text{base}}} \sum_{\substack{i \in \mathcal{I}_{\text{inj}} \\ i < k}} \alpha_i q_k^{\text{base}} \\ &= C_{\text{base}} - \sum_{i \in \mathcal{I}_{\text{inj}}} \alpha_i \sum_{\substack{k \in \mathcal{I}_{\text{base}} \\ k > i}} q_k^{\text{base}} \\ &= C_{\text{base}} - \sum_{i \in \mathcal{I}_{\text{inj}}} \alpha_i \Gamma_i. \end{aligned} \tag{39}$$

Consequently, the rendering surrogate becomes

$$\widetilde{C}_p = C_{\text{base}} + \sum_{i \in \mathcal{I}_{\text{inj}}} \left[T_i^{\text{base}} \left(1 - \sum_{\substack{j \in \mathcal{I}_{\text{inj}} \\ j < i}} \alpha_j \right) \alpha_i c_i - \alpha_i \Gamma_i \right], \tag{40}$$

which recovers Eq. (22) in Section 4. The first term inside the brackets represents the injected Gaussian's color contribution after attenuation by the original and preceding injected Gaussians. The second accounts for its attenuation of the original-scene contribution behind it.

It is worth noting that "first-order" here refers specifically to the approximation of the transmittance factors associated with the injected Gaussians. When these approximated transmittance terms are multiplied by an injected Gaussian's own opacity, selected second-order terms are retained in the resulting color expression. Therefore, Eq. (40) should not be interpreted as a strict first-order Taylor expansion of the full pixel color.

Cached evaluation. For a fixed camera pose, the depth ordering and the quantities T_i^{base}, C_{base}, and Γ_i are computed once and cached. During each subsequent optimization step, the prefix sums S_i are updated in a single front-to-back pass over the injected Gaussians. Each affected pixel can therefore be evaluated in $O(|\mathcal{I}_{\text{inj}}|)$ time without traversing the original Gaussians again.

B.5 Details for Gradient Computation

For a fixed victim viewpoint P_{vic}, we optimize the injected opacity and appearance parameters by minimizing

$$\mathcal{J}_{\mathrm{inner}}(\Phi) = \mathcal{L}_{\mathrm{adv}}(\Phi) + M \sum_{P \in \mathcal{P}_{\mathrm{ben}}} \max\left(0,\, \Delta\widetilde{\mathcal{L}}^{P}_{\mathrm{sth}}(\Phi) - \epsilon_{\mathrm{sth}}\right). \tag{41}$$

The adversarial term is differentiated through the cached rendering surrogate, whereas the benign-view penalty uses the parameter gradients precomputed at initialization. No additional benign-view rendering is required during these updates.

Pixel opacities and appearance parameters. Let Ω_{vic} denote the victim image's pixel set. For each pixel p, we write the effective opacity of injected Gaussian i as

$$\alpha_{i,p} = o_i w_{i,p}, \tag{42}$$

where o_i is the optimized opacity and $w_{i,p} \in [0,1]$ is the fixed projected Gaussian weight. Thus, o_i is shared across pixels, while $\alpha_{i,p}$ depends on the pixel.

To distinguish the optimized appearance parameters from their evaluated RGB colors, let $c_i^{\mathrm{vic}} \in \mathbb{R}^3$ denote the color obtained from c_i at P_{vic}, and define its Jacobian as

$$B_i^{\mathrm{vic}} = \frac{\partial c_i^{\mathrm{vic}}}{\partial c_i}. \tag{43}$$

The rendering expressions in the main text and Appendix B.4 denote this evaluated RGB color simply by c_i because the camera pose is fixed. For directly optimized RGB colors, $c_i^{\mathrm{vic}} = c_i$ and $B_i^{\mathrm{vic}} = I_3$, where $I_3 \in \mathbb{R}^{3\times 3}$ is the identity matrix. When appearance is represented by coefficients, B_i^{vic} accounts for the corresponding color-evaluation map.

Victim-view image gradients. The targeted adversarial loss is

$$\mathcal{L}_{\mathrm{adv}} = (1-\lambda)\mathcal{L}_1(I_{\mathrm{vic}}, I_{\mathrm{trg}}) + \lambda\mathcal{L}_{\text{D-SSIM}}(I_{\mathrm{vic}}, I_{\mathrm{trg}}), \tag{44}$$

where I_{vic} is assembled from the surrogate pixel colors $\widetilde{C}_p$ in Eq. 40. Define the gradient with respect to each RGB pixel as

$$\delta_p = \nabla_{\widetilde{C}_p} \mathcal{L}_{\mathrm{adv}} \in \mathbb{R}^3. \tag{45}$$

For a summed $\mathcal{L}_1$ loss and $\mathcal{L}_{\text{D-SSIM}} = (1 - \mathrm{SSIM})/2$, this becomes

$$\delta_p = (1-\lambda)\,\mathrm{sign}\left(\widetilde{C}_p - I_{\mathrm{trg},p}\right) - \frac{\lambda}{2}\nabla_{\widetilde{C}_p}\mathrm{SSIM}(I_{\mathrm{vic}}, I_{\mathrm{trg}}), \tag{46}$$

where the sign function acts componentwise. If $\mathcal{L}_1$ is averaged over pixel channels, its gradient includes the corresponding normalization factor. At zero residual, we use the subgradient $\mathrm{sign}(0) = 0$.

Derivatives of the rendering surrogate. We make the pixel dependence explicit and define the injected-opacity prefix sum

$$S_{i,p} = \sum_{\substack{j \in \mathcal{I}_{\mathrm{inj}}(p) \\ j<i}} \alpha_{j,p}. \tag{47}$$

Similarly, let the quantities $T_{i,p}^{\mathrm{base}}$ and $\Gamma_{i,p}$ be the fixed original-scene transmittance and suffix color contribution shown in Appendix B.4. For each injected Gaussian contributing to pixel p, differentiating the rendering surrogate with respect to its appearance parameters gives

$$\frac{\partial \widetilde{C}_p}{\partial c_i} = T_{i,p}^{\mathrm{base}}(1 - S_{i,p})\alpha_{i,p}B_i^{\mathrm{vic}}. \tag{48}$$

For directly optimized RGB colors, this Jacobian reduces to $T_{i,p}^{\mathrm{base}}(1 - S_{i,p})\alpha_{i,p}I_3$.

Differentiating with respect to the effective pixel opacity gives

$$\frac{\partial \widetilde{C}_p}{\partial \alpha_{i,p}} = T_{i,p}^{\mathrm{base}}(1 - S_{i,p})c_i^{\mathrm{vic}} - \sum_{\substack{k \in \mathcal{I}_{\mathrm{inj}}(p) \\ k>i}} T_{k,p}^{\mathrm{base}}\alpha_{k,p}c_k^{\mathrm{vic}} - \Gamma_{i,p}. \tag{49}$$

The three terms account for Gaussian i's own color contribution, its attenuation of subsequent injected Gaussians, and its attenuation of the original scene behind it, respectively.

Because the optimized variable is o_i, rather than the pixel-dependent $\alpha_{i,p}$, the chain rule yields

$$\frac{\partial \widetilde{C}_p}{\partial o_i} = w_{i,p} \frac{\partial \widetilde{C}_p}{\partial \alpha_{i,p}}. \tag{50}$$

The rendering derivatives are zero for pixels to which Gaussian i does not contribute. The suffix sum in the opacity derivative can be computed in a single back-to-front pass over the injected Gaussians, without traversing the original scene.

Cached benign-view gradients. For each benign pose $P \in \mathcal{P}_{\text{ben}}$, define the hinge-activation indicator

$$\mathbb{I}^P_{\text{sth}}(\Phi) = \begin{cases} 1, & \Delta\widetilde{\mathcal{L}}^P_{\text{sth}}(\Phi) > \epsilon_{\text{sth}}, \\ 0, & \Delta\widetilde{\mathcal{L}}^P_{\text{sth}}(\Phi) \le \epsilon_{\text{sth}}. \end{cases} \tag{51}$$

At the hinge boundary, this definition selects the zero subgradient.

Since the estimated benign-view loss change is linear in the parameter deviations,

$$\Delta\widetilde{\mathcal{L}}^P_{\text{sth}}(\Phi) = \sum_{i=1}^{m} \left[g^P_{o,i}(o_i - o_i^0) + (g^P_{c,i})^\top (c_i - c_i^0) \right], \tag{52}$$

its parameter gradients are the cached quantities $g^P_{o,i}$ and $g^P_{c,i}$. Consequently,

$$\nabla_{o_i} \mathcal{R}_{\text{sth}}(\Phi) = M \sum_{P \in \mathcal{P}_{\text{ben}}} \mathbb{I}^P_{\text{sth}}(\Phi) g^P_{o,i}, \tag{53}$$

and

$$\nabla_{c_i} \mathcal{R}_{\text{sth}}(\Phi) = M \sum_{P \in \mathcal{P}_{\text{ben}}} \mathbb{I}^P_{\text{sth}}(\Phi) g^P_{c,i}. \tag{54}$$

The activation indicators are recomputed from the current parameter deviations, but the cached gradients remain fixed. In particular, $g^P_{c,i}$ is computed with respect to the same appearance parameterization used by the optimizer.

Combined parameter gradients and projected updates. Combining the victim-view and benign-view terms gives

$$\begin{aligned} \nabla_{o_i} \mathcal{J}_{\text{inner}}(\Phi) = & \sum_{p \in \Omega_{\text{vic}}} w_{i,p} \left(\frac{\partial \widetilde{C}_p}{\partial \alpha_{i,p}} \right)^\top \delta_p \\ & + M \sum_{P \in \mathcal{P}_{\text{ben}}} \mathbb{I}^P_{\text{sth}}(\Phi) g^P_{o,i}, \end{aligned} \tag{55}$$

and

$$\begin{aligned} \nabla_{c_i} \mathcal{J}_{\text{inner}}(\Phi) = & \sum_{p \in \Omega_{\text{vic}}} T^{\text{base}}_{i,p} (1 - S_{i,p}) \alpha_{i,p} (B^{\text{vic}}_i)^\top \delta_p \\ & + M \sum_{P \in \mathcal{P}_{\text{ben}}} \mathbb{I}^P_{\text{sth}}(\Phi) g^P_{c,i}. \end{aligned} \tag{56}$$

Both contributions enter with positive signs because the inner objective minimizes the targeted image-matching loss plus the benign-view penalty.

These gradients are used in the alternating updates described in the main text. After updating appearance, the opacity step is

$$o_i^{(t+1)} = \Pi_{[0, o_{\max}]} \left[o_i^{(t)} - \eta_o \nabla_{o_i} \mathcal{J}_{\text{inner}} \left(\mathbf{o}^{(t)}, \mathbf{c}^{(t+1)} \right) \right], \tag{57}$$

where $\mathbf{o}$ and $\mathbf{c}$ collect the injected opacity and appearance parameters, respectively. The projection clips the optimized opacity o_i, rather than independently clipping its pixel-dependent values $\alpha_{i,p}$.

If the implementation instead optimizes logits z_i with $o_i = \sigma(z_i)$, the additional chain rule is

$$\nabla_{z_i} \mathcal{J}_{\text{inner}} = o_i (1 - o_i) \nabla_{o_i} \mathcal{J}_{\text{inner}}. \tag{58}$$

For $0 < o_{\max} < 1$, the upper-opacity bound is then enforced by clamping $z_i \le \log(o_{\max}/(1 - o_{\max}))$ after each logit update. The gradient and update must be evaluated in the same parameterization.



Target object. We use a standard octagonal stop sign with the CLIP prompt "an octagonal stop sign". The target size is fixed to 125×125 pixels.

Table 5: Target object and key hyperparameter settings used in PhantomSplat.

Hyperparameter	**Value**
Initialization mode	Convex hull
Surface samples	90,000
Number of layers	8
Initial opacity	0.004
Maximum opacity ϵ	0.05
Initial scale	0.003
Inflation offset	0.005
XYZ jitter	0.0001
Tangential jitter	0.002
Target size	125×125 px
λ_{adv}	1.0
Big-M	10^5
Stealth tolerance ξ	10^{-7}
LR color / opacity	0.01 / 0.005
Stealth loss L1 / SSIM	0.8 / 0.2
BO steps	25
Inner-loop steps per BO step	1400
BO azimuth range	$[-180°, 180°]$
BO elevation range	$[-15°, 90°]$
BO radius range	$[-0.6, 0.6]$

C Experiment Details

Hyperparameter Settings Unless otherwise specified, we initialize adversarial Gaussians using convex-hull surface sampling with 90,000 surface samples and 8 inflation layers. The initial opacity is set to 0.004, and the maximum opacity is constrained by $\epsilon = 0.05$ during optimization. The initial Gaussian scale is 0.003, with an inflation offset of 0.005 between layers, isotropic position jitter of 0.0001, and tangential jitter of 0.002. The injected target is rendered with a fixed image-space size of 125×125 pixels.

For optimization, we use $\lambda_{\text{adv}} = 1.0$, Big-$M = 10^5$, and stealth tolerance $\xi = 10^{-7}$. The learning rates for color and opacity are 0.01 and 0.005. For the stealth loss, we use an L1/SSIM weighting of $0.8/0.2$. For Bayesian victim-view search, we use 25 outer-loop steps and 1400 inner-loop updates per candidate view, with azimuth, elevation, and radius ranges of $[-180°, 180°]$, $[-15°, 90°]$, and $[-0.6, 0.6]$, respectively. The detailed hyperparameter settings are summarized in Table 5. All pre-trained 3DGS and baseline models are trained for 30,000 iterations. Experiments are conducted on an NVIDIA A100-SXM4-40GB GPU for 3DGS training and an AMD EPYC 7413 24-core CPU for voxel-grid preprocessing in StealthAttack.

For Mip-NeRF360, we use the $1/4$-resolution image set to train the 3DGS models. For Tanks & Temples, we use the Intermediate subset for evaluation. For the Free dataset, we reconstruct scenes using images at full resolution. All datasets use the standard train/test split.

We use a standard octagonal stop sign as the target object in all experiments. The CLIP prompt is set to "an octagonal stop sign". For consistency across scenes, the injected target is rendered with a fixed image-space size of 125×125 pixels.

C.1 Baseline Details

We use the same target images and follow the reported StealthAttack hyperparameters to provide a controlled cross-setting comparison. However, we note that StealthAttack operates under a reconstruction-time poisoning threat model, whereas PhantomSplat targets post-training asset manipulation. To maximize the attack effectiveness of the target image, we adopt a linear noise scheduler with a starting sigma of 100. All models are trained for 30,000 iterations.

Regarding training time, we find that the preprocessing time for voxel-grid construction is much longer than the actual training time. Therefore, we consider it an important factor and include it in the total pipeline training time. The qualitative comparison of baseline settings is shown in Figure 6, and the complete results are provided in the experiments below.



Figure 6: Baseline settings on the Bike scene. The leftmost result shows the pure low-density attack without noise, the middle result shows the noise-based attack with $\sigma = 30$, and the rightmost result shows the attack with $\sigma = 100$. Since $\sigma = 100$ achieves the best attack performance, we adopt $\sigma = 100$ for all scenes. However, it also degrades scene fidelity and other evaluation metrics, including training-view PSNR, SSIM, and LPIPS.

Table 6: Training time comparison (minutes) on Mip-NeRF360 using NVIDIA A100-SXM4-40GB. For StealthAttack, we report the total time including voxel-grid construction and training.

3D Scene	Ours Optimization	3DGS	StealthAttack (Voxel Grid + Training)
Bicycle	8.98	28.05	38.81 + 51.30
Bonsai	3.17	12.03	210.45 + 19.61
Counter	4.20	10.27	134.17 + 19.04
Flowers	3.56	19.23	20.08 + 44.02
Garden	3.25	24.53	93.58 + 35.32
Kitchen	4.58	11.87	259.49 + 19.13
Room	3.45	9.87	127.06 + 18.42
Stump	3.70	21.48	15.24 + 42.18
Treehill	3.36	21.05	26.23 + 44.05
Average	**4.25**	**17.04**	**135.35**

D Extended Experiments

D.1 Comprehensive Comparison of Training Time

Table 6 compares the training time of PhantomSplat, StealthAttack, and the original 3DGS on the Mip-NeRF360 dataset. Table 7 further reports the results of PhantomSplat and the original 3DGS on the Tanks & Temples and Free datasets. These complete results show that our method achieves consistent time efficiency across different datasets.

D.2 Comprehensive Results of Our Method on Three Datasets

As shown in Table 8, PhantomSplat achieves stronger attack effectiveness and stealthiness than StealthAttack on Mip-NeRF360. Specifically, our method obtains higher attack-view PSNR in most scenes and consistently preserves better training-view fidelity across all scenes. Together with the complete results on the other two datasets in Table 9, these evaluations show that PhantomSplat remains effective while preserving training-view fidelity across diverse 3D scenes.

Table 7: Training time comparison of our method on Tanks & Temples and Free datasets using an NVIDIA A100-SXM4-40GB GPU.

3D Scene	Ours Attack Training (min)	Original 3DGS (min)
Tanks & Temples		
Family	3.99	23.25
Francis	5.35	25.66
Horse	3.26	23.75
Lighthouse	4.31	21.46
M60	9.03	22.90
Panther	15.03	28.88
Playground	3.41	29.53
Train	6.55	29.60
Average	**6.37**	**25.63**
Free		
grass	3.07	36.70
hydrant	4.41	39.55
lab	5.48	21.38
pillar	5.72	22.65
road	3.68	25.07
sky	8.18	14.73
stair	3.63	18.67
Average	**4.88**	**25.54**

Table 8: Complete image-space results on Mip-NeRF360 scenes. We report PSNR, SSIM, and LPIPS for both the attack view and training views.

Method	View / Metric	Avg. Time	BI.	FL.	GA.	ST.	TR.	RO.	CO.	KI.	BO.	Avg
3DGS	Training View PSNR	17.04	26.59	24.31	30.29	31.33	24.82	35.73	31.39	34.45	34.37	30.36
	Training View SSIM		0.862	0.777	0.919	0.911	0.810	0.968	0.949	0.970	0.969	0.904
	Training View LPIPS		0.109	0.191	0.045	0.077	0.170	0.029	0.049	0.023	0.026	0.080
StealthAttack Ke et al. (2025)	Attack View PSNR	135.35	26.14	22.23	26.78	27.18	23.6	22.52	28.86	27.66	32.25	27.37
	Attack View SSIM		0.796	0.676	0.805	0.74	0.701	0.394	0.927	0.907	0.969	0.800
	Attack View LPIPS		0.268	0.356	0.233	0.38	0.429	0.766	0.087	0.132	0.073	0.260
	Training View PSNR		25.15	17.77	6.89	20.35	19.319	25.21	26.70	24.72	31.41	23.35
	Training View SSIM		0.792	0.368	0.007	0.425	0.563	0.785	0.799	0.740	0.931	0.676
	Training View LPIPS		0.237	0.595	0.809	0.62	0.468	0.369	0.283	0.310	0.168	0.363
PhantomSplat	Attack View PSNR	4.25	34.09	38.62	37.06	36.85	33.65	31.69	31.40	31.62	30.70	33.96
	Attack View SSIM		0.991	0.996	0.996	0.996	0.994	0.987	0.966	0.987	0.969	0.987
	Attack View LPIPS		0.007	0.003	0.004	0.005	0.009	0.014	0.015	0.013	0.034	0.012
	Training View PSNR		26.53	24.12	30.17	30.79	24.65	32.84	30.90	32.60	33.97	29.62
	Training View SSIM		0.862	0.778	0.919	0.909	0.809	0.962	0.947	0.985	0.966	0.904
	Training View LPIPS		0.113	0.182	0.041	0.016	0.171	0.034	0.045	0.023	0.029	0.073

Table 9: Image-space reconstruction quality across three datasets. We report PSNR, LPIPS, and SSIM for the attack view, training views after attack, and original model training views.

3D Scene	Attack View			Training View			Original Model Training View		
	PSNR ↑	LPIPS ↓	SSIM ↑	PSNR ↑	LPIPS ↓	SSIM ↑	PSNR ↑	LPIPS ↓	SSIM ↑
Mip-NeRF360									
Bicycle	34.09	0.0069	0.9906	26.53	0.1130	0.8616	26.59	0.1094	0.8624
Bonsai	30.70	0.0342	0.9694	33.97	0.0287	0.9661	34.37	0.0258	0.9694
Counter	31.40	0.0150	0.9660	30.90	0.0446	0.9470	31.39	0.0490	0.9490
Garden	37.06	0.0040	0.9960	30.17	0.0405	0.9190	30.29	0.0450	0.9190
Kitchen	31.62	0.0130	0.9870	32.60	0.0230	0.9850	34.45	0.0230	0.9700
Room	31.69	0.0140	0.9873	32.84	0.0337	0.9620	35.73	0.0290	0.9680
Stump	36.85	0.0050	0.9960	30.79	0.0160	0.9090	31.33	0.0770	0.9110
Flowers	38.62	0.0030	0.9960	24.12	0.1817	0.7780	24.31	0.1910	0.7770
Treehill	33.65	0.0090	0.9940	24.65	0.1710	0.8090	24.82	0.1700	0.8100
Average	**33.96**	**0.0116**	**0.9869**	**29.62**	**0.0725**	**0.9041**	**30.36**	**0.0799**	**0.9040**
Tanks & Temples									
Family	26.29	0.0192	0.9822	27.60	0.0834	0.8985	27.62	0.0810	0.8990
Francis	30.12	0.0110	0.9810	30.22	0.1010	0.8810	30.45	0.1025	0.9094
Horse	28.99	0.0130	0.9890	27.35	0.0850	0.9100	27.65	0.0820	0.9120
Lighthouse	31.78	0.0120	0.9910	26.42	0.0982	0.9042	27.33	0.0947	0.9035
M60	28.12	0.0210	0.9870	30.12	0.0808	0.9016	30.43	0.0796	0.9304
Panther	34.56	0.0150	0.9940	24.15	0.2291	0.8354	24.42	0.2277	0.8387
Playground	27.12	0.0391	0.9640	28.54	0.1432	0.8638	28.66	0.1407	0.8747
Train	35.13	0.0101	0.9690	25.07	0.1489	0.8154	25.19	0.1474	0.8382
Average	**30.26**	**0.0176**	**0.9822**	**27.43**	**0.1212**	**0.8762**	**27.72**	**0.1195**	**0.8882**
Free									
Grass	35.22	0.0320	0.9910	33.29	0.0239	0.9620	33.89	0.0202	0.9640
Hydrant	30.09	0.0410	0.9730	32.44	0.0318	0.9580	33.07	0.0238	0.9611
Lab	35.32	0.0180	0.9930	31.70	0.0268	0.9620	32.20	0.0232	0.9631
Pillar	30.53	0.0350	0.9920	35.36	0.0314	0.9490	35.42	0.0267	0.9531
Road	28.43	0.0510	0.9850	33.62	0.0254	0.9540	33.73	0.0241	0.9580
Sky	26.42	0.0360	0.9890	32.44	0.0369	0.9890	33.22	0.0289	0.9624
Stair	31.54	0.0290	0.9790	32.23	0.0295	0.9500	34.51	0.0278	0.9534
Average	**31.08**	**0.0346**	**0.9860**	**33.01**	**0.0294**	**0.9606**	**33.72**	**0.0250**	**0.9593**

D.3 Comprehensive Downstream Model Results of Our Method on Three Datasets

Table 10 reports the complete downstream-model evaluation across all three datasets. The results show that the injected target is consistently recognized by downstream models across evaluated scenes, demonstrating the semantic effectiveness of the attack.

Table 10: Attack-view semantic effectiveness across three datasets. We report semantic similarity using CLIP and object-level detection confidence using YOLO.

3D Scene	Attack View (Semantic)				
	$\text{CLIP}_r \uparrow$	CLIP_t	Ratio ↑	$\text{YOLO}_r \uparrow$	YOLO_t
Mip-NeRF360					
Bicycle	0.263	0.265	0.99	0.933	0.924
Bonsai	0.223	0.291	0.76	0.892	0.961
Counter	0.250	0.238	1.05	0.975	0.977
Garden	0.336	0.307	1.09	0.823	0.936
Kitchen	0.197	0.217	0.91	0.928	0.966
Room	0.232	0.230	1.01	0.964	0.963
Stump	0.280	0.291	0.96	0.937	0.918
Flowers	0.292	0.286	1.02	0.92	0.927
Treehill	0.297	0.296	1.00	0.95	0.937
Average	**0.263**	**0.269**	**0.98**	**0.925**	**0.946**
Tanks & Temples					
Family	0.273	0.290	0.94	0.932	0.954
Francis	0.290	0.310	0.94	0.968	0.947
Horse	0.250	0.257	0.97	0.949	0.959
Lighthouse	0.265	0.281	0.94	0.943	0.968
M60	0.252	0.254	0.99	0.954	0.972
Panther	0.279	0.284	0.98	0.912	0.929
Playground	0.299	0.332	0.90	0.882	0.943
Train	0.297	0.294	1.01	0.901	0.931
Average	**0.276**	**0.288**	**0.96**	**0.930**	**0.950**
Free					
Grass	0.229	0.242	0.95	0.911	0.942
Hydrant	0.269	0.282	0.95	0.932	0.947
Lab	0.271	0.274	0.99	0.943	0.959
Pillar	0.268	0.266	1.01	0.917	0.932
Road	0.244	0.259	0.94	0.932	0.935
Sky	0.288	0.291	0.99	0.922	0.915
Stair	0.239	0.280	0.85	0.938	0.936
Average	**0.258**	**0.271**	**0.95**	**0.928**	**0.938**

D.4 Comprehensive Visualization Results of Our Method on Mip-NeRF360

This Figure 7 presents the complete qualitative results on all Mip-NeRF360 scenes. All visualizations follow the same layout: from left to right, the columns show the target image, the StealthAttack result, our attack-view result, and two stealth-view results of our model.

D.5 Generalization to Different Sign Types

In addition to the standard stop-sign target, we evaluate two additional settings, as shown in Figure 8. The warning-sign attack is used to evaluate the generality of PhantomSplat across different semantic targets, while the transparent-bicycle attack represents a more challenging objective that goes beyond inserting a visible target pattern.

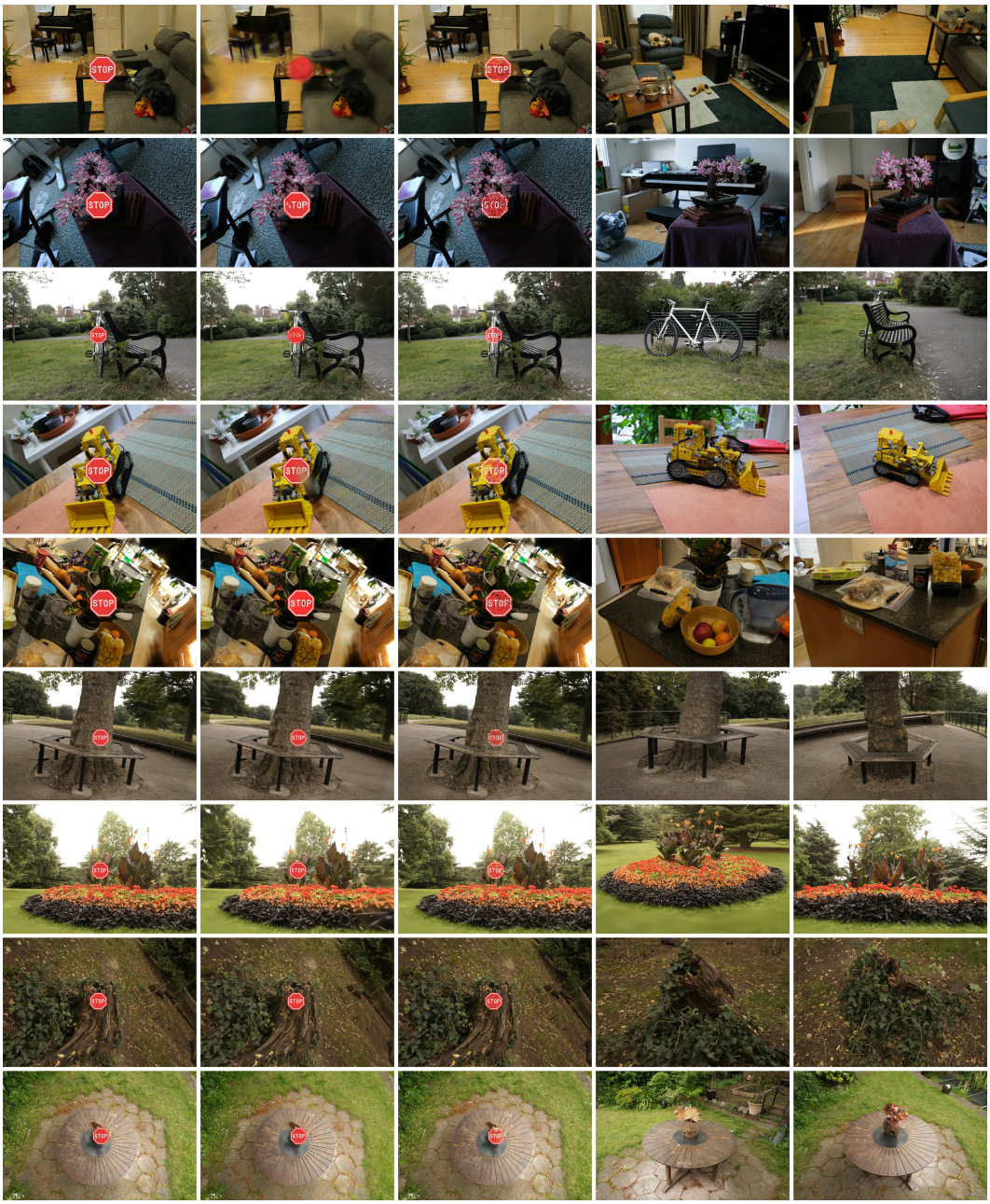

Figure 7: Complete qualitative comparison on all Mip-NeRF360 scenes. For each scene, we use a consistent layout from left to right: target image, StealthAttack result, our attack-view result, and two stealth-view renderings of our model. The results show that PhantomSplat reconstructs the target pattern in the attack view while preserving visual consistency from stealth views.

For the warning-sign setting, we keep the same optimization configuration and replace only the target sign pattern. This shows that PhantomSplat can generalize to different target semantics without modifying the original 3DGS reconstruction. This flexibility is difficult for reconstruction-time poisoning attacks, which typically require the malicious objective to be specified before training and embedded into the poisoned data.

For the transparent-bicycle setting, we set the target image to the clean background image without the bicycle, encouraging the adversarial Gaussians to make the bicycle surface visually consistent with the surrounding background. Unlike sign insertion, this objective requires suppressing or camouflaging existing object appearance rather than adding a salient target pattern, making it a more challenging case. The result demonstrates that PhantomSplat can support both generalized semantic target insertion and more difficult camouflage-style manipulation within the same post-training optimization framework in a very short time.



Transparent bicycle setting **Stealth View 1** **Stealth View 2**



Bicycle with 'warning' sign **Stealth View 1** **Stealth View 2**

Figure 8: Qualitative results for additional attack settings. The warning-sign attack demonstrates that PhantomSplat generalizes to different sign types, while the transparent-bicycle attack shows that adversarial Gaussians can make the bicycle surface blend into the background by using the background image as the target.

D.6 Approximation error analysis.

PhantomSplat performs render-free adversarial primitive optimization: during the inner-loop updates, it does not repeatedly invoke the full 3DGS renderer or backpropagate through the complete Gaussian scene. Instead, the approximate renderer serves as an optimization surrogate that provides efficient guidance for updating the injected low-opacity Gaussians. Although this surrogate is not intended to exactly reproduce the full renderer at every step, its discrepancy should remain controlled so that the optimized primitives transfer to the true rendering pipeline.

To verify this, we compare the images produced by the approximate renderer and the full 3DGS renderer using the same adversarial Gaussians at different optimization iterations. As shown in Table 11 and Figure 9, the approximation error remains within a reasonable range before final convergence, with low L1/MSE and consistently high PSNR. This indicates that the approximate renderer provides a stable guidance signal while making the optimization substantially easier than direct full-render optimization.

Table 12: Viewpoint-locality evaluation. GT PSNR measures the similarity between the attacked rendering and the ground-truth appearance at different camera offsets. Higher values indicate recovery toward the original scene appearance.

View Offset	GT PSNR ↑
-20°	26.41
-15°	24.88
-10°	23.59
-5°	18.34
-2°	14.82
$\mathbf{0}^\circ$	**13.28**
$+2^\circ$	15.06
$+5^\circ$	19.39
$+10^\circ$	23.77
$+15^\circ$	25.12
$+20^\circ$	24.73

Table 11: Approximation error analysis across optimization iterations. Using the same adversarial Gaussians, we compare our approximate renderer with the full 3DGS renderer.

Iteration	L1 ↓	MSE ↓	PSNR ↑
200	3.46e-6	3.56e-8	74.47
400	1.50e-4	1.74e-5	47.57
800	2.59e-3	3.13e-4	35.05
1400	1.30e-3	6.55e-4	31.84

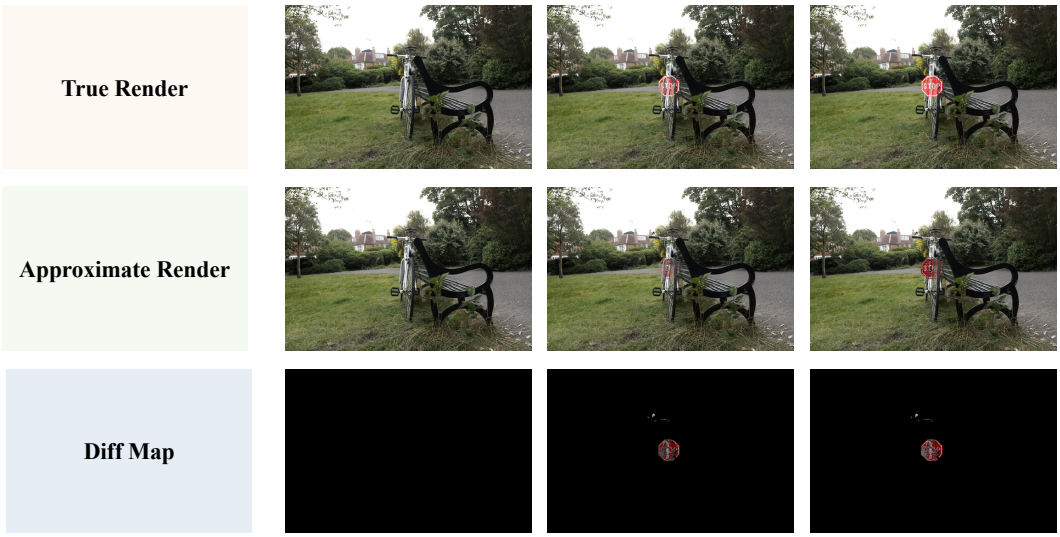

Figure 9: Visual error map between the approximate renderer and the full 3DGS renderer. The small residual differences indicate that the render-free approximation remains close enough to the true rendering pipeline to guide adversarial primitive optimization.

D.7 Viewpoint Locality

We further evaluate the viewpoint locality of PhantomSplat by gradually shifting the camera away from the optimized attack viewpoint while keeping the remaining camera configuration fixed. As shown in Fig. 10, the attack effect is strongly localized around the optimized viewpoint. At the attack view (0°), the rendering substantially deviates from the ground-truth appearance, resulting in a GT PSNR of 13.28 dB. However, the rendering rapidly recovers as the viewpoint moves away: the GT PSNR increases to approximately 24 dB at $\pm 10^\circ$ and reaches about 25–26 dB at larger offsets. This trend indicates that the adversarial effect is confined to a relatively small neighborhood around the optimized viewpoint, rather than globally altering the scene appearance.



Figure 10: Viewpoint-locality analysis of PhantomSplat. We gradually shift the camera away from the optimized attack viewpoint while keeping the remaining camera configuration fixed. The adversarial effect is concentrated near the optimized viewpoint and progressively disappears as the viewpoint moves away.