DivAS: Interactive 3D Segmentation by Depth-Weighted Voxel Aggregation

Anonymous authors
Paper under double-blind review

Abstract

Interactive 3D segmentation of a reconstructed scene should not require a representation-specific optimization loop. We observe that the recipe for lifting 2D foundation-model masks into 3D, namely prompting a few views, refining the resulting masks with rendered depth, and fusing the multi-view evidence into a voxel grid, is shared across scene representations. What remains representation-specific is only the depth signal returned by the renderer and the occupancy prior that gates fusion. We present **DivAS** (Depth-interactive Voxel Aggregation Segmentation), an optimization-free, training-free framework that realizes this recipe as a single interaction-and-fusion skeleton with lightweight, representation-specific adapters, instantiated on both Gaussian Splatting (GS) and NeRF backbones. On standard forward-facing and unbounded benchmarks, the GS instantiation attains segmentation quality competitive with state-of-the-art optimization-based methods, and the best on LLFF, while being the only one to reach this quality within the consumer-hardware memory envelope at standard resolution. Both instantiations run end-to-end around 2× faster than feature-field baselines, with a per-update fusion-kernel cost below 70 ms. Because segmentation evidence is gathered from a small, bounded set of anchor views, user effort and computation remain independent of the training-set size. The same skeleton applied to a NeRF backbone matches or exceeds the performance of optimization-based NeRF baselines, confirming that the recipe transfers across fundamentally different 3D representations.

1 Introduction

The ability to segment and annotate 3D scenes efficiently underpins a growing list of computer-vision and graphics workflows: content creation, asset re-use, scene editing, robotics, and autonomous perception. Recent years have witnessed remarkable progress in two parallel directions: *neural implicit and explicit representations* for 3D scene modeling and *foundation models* for general-purpose image understanding. Yet the practitioner who wishes to combine the two is forced to choose between several representation-specific pipelines, each with its own training loop, hyperparameter set, and code base.

Neural Radiance Fields (NeRF) (Mildenhall et al., 2020) represent a 3D scene as a continuous volumetric function optimized via differentiable volume rendering. Variants such as Instant-NGP (Müller et al., 2022) reduce training to minutes, and Mip NeRF 360° (Barron et al., 2022) extends the formulation to unbounded outdoor environments. NeRF density fields encode rich geometric cues: volumetric density, integrated depth, and view-dependent appearance that make them attractive substrates for scene understanding. More recently, 2D Gaussian Splatting (2DGS) (Huang et al., 2024) and 3D Gaussian Splatting (3DGS) (Kerbl et al., 2023) have shifted the field towards an explicit, real-time scene primitive: a discrete set of oriented surfels carrying opacity and view-dependent color, which can be splatted into the image plane at hundreds of frames per second. The two representations differ fundamentally in geometry: NeRF is volumetric and continuous, GS is sparse and explicit; NeRF returns depth integrated along a ray, GS returns per-pixel depth statistics (median, expected) over a discrete sample list.

In parallel, the Segment Anything Model (SAM) (Kirillov et al., 2023), trained on billions of masks, has established a foundation for zero-shot, prompt-based 2D segmentation. SAM's success raises a practical

question: *Can we achieve comparable zero-shot segmentation performance in 3D scenes, regardless of whether the scene is stored as a NeRF or as a 2DGS surfel set?*

The bifurcated state of 3D segmentation. The community's response to this question has been representation-specific. NeRF segmentation methods take diverse routes: SA3D (Cen et al., 2023) runs an iterative inverse-rendering loop with self-prompted view propagation, SANeRF-HQ (Liu et al., 2024) propagates a multi-view user-prompts across the training views via a learned refinement network, and ISRF (Goel et al., 2023) distills DINO (Caron et al., 2021) features into a feature field for query-based selection. GS segmentation methods are equally varied: SAGA (Cen et al., 2025) distills SAM features per Gaussian and clusters similar primitives at query time, Gaussian Grouping (Ye et al., 2024) associates a learnable identity feature with each primitive trained against video object segmentation pseudo-labels, and SA3D-GS Cen et al. (2024) replays the SA3D inverse-rendering loop on top of Gaussian primitives. Despite this surface diversity, every one of these methods shares a backbone commitment and an end goal: to turn 2D user prompts into per-primitive (or per-sample) 3D evidence in a single chosen representation. Two further costs follow from this duplication: (i) every method commits to a single backbone, so a practitioner who trains a 2DGS scene cannot reuse the segmentation tooling developed for NeRF, and vice versa; and (ii) almost all of these methods are *optimization-based*, running a per-scene inverse-rendering loop whose cost scales with the training-set size N_{train} and routinely takes tens of minutes to an hour per segmentation session, depending on the scene complexity.

Insight: one interaction-and-fusion recipe, two representation-specific adapters. We argue that a single recipe, depth-weighted voxel voting, can replace all of the backbone-specific machinery above. The recipe factors into five stages: (1) sample a set of anchor views over the camera distribution, (2) prompt SAM on a centroid view of the target, (3) refine the resulting 2D mask using a depth-derived weighting, (4) aggregate the multi-view evidence into a pre-computed voxel-occupancy grid, and (5) display the running 3D mask back to the user for the next click. These five stages, namely the anchor-view sampling strategy, centroid-view exploration, SAM prompting workflow, depth-guided mask refinement, voxel-space evidence aggregation, and interactive fusion loop, are shared by the NeRF and 2DGS instantiations. We therefore describe DivAS as a *shared interaction-and-fusion framework with representation-specific adapters* rather than as a fully representation-independent method. The framework is common at the interaction-and-fusion level, while only two adapter layers are specialized per representation: (i) the extraction of depth and occupancy information from the underlying renderer, and (ii) the projection of the final voxel segmentation back to the native scene representation. In NeRF these adapters operate on volumetric densities, whereas in 2DGS they operate on anisotropic surfels. The shared skeleton stays unchanged even though the underlying rendering primitives differ substantially, which is what lets the same interaction pipeline transfer across fundamentally different 3D scene representations.

The cost story: per-view latency vs. per-session optimization. The efficiency advantage of optimization-free segmentation is not fewer user actions, since a depth-weighted voxel-vote method still solicits as many anchor and centroid prompts as the user finds useful, but rather *when* the cost is paid and how interactive the loop feels. Optimization-based baselines such as SAGA and SA3D amortize their cost into a single, long, monolithic per-scene optimization pass that runs after the user has finished annotating and can take tens of minutes to over an hour, depending on scene complexity. DivAS instead pays its cost *incrementally, per annotated view*: SAM inference on the centroid view for 2DGS and rendering time in NeRF plus a single CUDA voxel-fusion kernel call (below 70 ms across all benchmark scenes) that re-fuses the entire view buffer accumulated so far. Once a view's annotations are committed, the updated 3D mask is immediately back-projected onto the next anchor view as visual feedback, so the user observes the segmentation evolve *across successive view annotations* rather than *after* a long batch optimization.

Bounded anchor view set. A second, structural advantage of optimization-free voxel voting is that the set of anchor views from which segmentation evidence is collected is decoupled from the training-set size. SAGA and SA3D operate by inverse rendering across all training cameras; their cost is therefore $O(N_{\text{train}})$ and scales with denser captures. DivAS instead works from a fixed-size set of anchor views, obtained for unbounded 360° scenes by Fibonacci-sphere distribution and then ranked by geometry (González, 2010),

and for forward-facing LLFF by manual selection. The anchor count is therefore a *bounded view-processing budget* that does not grow with N_{train}. The bounded anchor budget suffices for a 60-image LLFF capture and a 300-image Mip-NeRF 360° capture, and the per-anchor centroid views that the user generates by clicking into each anchor are likewise bounded regardless of how the scene was captured.

Scope and applicability. DivAS targets static scenes that have already been reconstructed as a NeRF or 2DGS representation. The contribution is an interactive segmentation framework that operates *on top of* the reconstructed scene rather than a new capture-time or reconstruction-time method. Following the standard evaluation protocols used by prior interactive segmentation work, we report results on the forward-facing LLFF (Mildenhall et al., 2019) benchmark and the inward-facing Mip-NeRF 360° (Barron et al., 2022) benchmark. The segmentation skeleton itself, depth-weighted SAM refinement, voxel-occupancy fusion, and back-projected GUI feedback, makes no assumption beyond a renderer that exposes a depth statistic and an opacity volume, so it is in principle applicable to any reconstructed scene from which those two signals can be read. The Fibonacci-sphere anchor-view scheduler, in contrast, is designed for inward-facing captures where the scene can be reasonably approximated by a viewing sphere, and is the only component of the pipeline that carries a layout assumption. For more general camera layouts (outdoor street captures, drive-throughs, irregular trajectories), the Fibonacci scheduler may not propose the most informative anchors. In those settings the GUI exposes manual anchor selection, which trades a small amount of additional user interaction for unrestricted camera placement. This is a tradeoff between automation and user effort rather than a hard limitation of the segmentation framework.

Consumer-hardware accessibility. Peak VRAM, and its dependence on input resolution, is an often-overlooked dimension of segmentation cost. Optimization-based GS methods such as SAGA and Gaussian Grouping keep the trained primitives, gradient buffers, and a per-Gaussian learned feature resident throughout their per-scene optimization, so at the standard $\times 4$ down-sampling their peak VRAM exceeds the consumer-GPU envelope and effectively restricts them to data-center hardware. SAGA fits consumer GPUs only by dropping to the more aggressive $\times 8$ down-sampling, which halves linear resolution and degrades fine geometry. DivAS instead runs at the standard $\times 4$ within the consumer envelope on both NeRF and 2DGS backbones, because it keeps no learnable parameters or feature volumes co-resident with the scene at fusion time, so peak VRAM scales with image resolution and active-voxel count rather than with the trained model.

DivAS: a unified, optimization-free framework. Guided by these observations, we present **DivAS** - a unified framework for interactive 3D segmentation that introduces no additional training or optimization beyond the underlying scene representation. Given a pre-trained NeRF or 2DGS scene, DivAS performs segmentation through direct geometric reasoning and multi-view voxel voting rather than a per-scene inverse-rendering optimization loop. The framework exposes a common interaction-and-fusion skeleton on top of two backbone adapters:

- The **NeRF adapter** populates a voxel-occupancy grid by ray-marching the trained density field, producing a grid analogous to the density grid that NeRF itself maintains for empty-space skipping and uses the alpha-composited expected depth as the per-pixel geometric prior for multi-view voxel voting.
- The **2DGS adapter** populates the same voxel-occupancy grid by scattering trained surfel opacities directly into voxels. 2DGS has no continuous density field to ray-march and uses the rasterized median depth as the geometric prior, together with lightweight safeguards for the anisotropic floaters and transmissive materials introduced by 2DGS-specific geometry.

The shared skeleton Fibonacci-sphere anchor selection, geometric view ranking, centroid-zoom view generation, depth-weighted SAM mask refinement, and a single CUDA voxel-fusion kernel with thick-structure and thin-structure paths are identical between the two adapters. The user's interactive experience (click, watch the 3D mask update, refine) is also identical: a single GUI drives both backbones.

Contributions. The contributions of this paper are:

- **A shared interaction-and-fusion framework with representation-specific adapters.** We isolate the lifting recipe that NeRF and Gaussian-Splatting segmentation methods share into a single, training-free interactive pipeline, where only two lightweight adapters are specialized per backbone, one for renderer depth and occupancy extraction and one for back-projection to the native representation, preserving SAM's zero-shot generalization without per-scene optimization.
- **Depth-guided voxel fusion.** Unlike feature-driven affinity or optimization-based inverse rendering, DivAS reweights every SAM mask by the rendered scene depth and aggregates the multi-view evidence into a probabilistic voxel-occupancy grid through a single CUDA kernel that enforces depth, density, and spatial consistency per voxel, lifting masks only where they are geometrically coherent and at a per-call latency below 70 ms across all benchmark scenes.
- **Thin-structure recovery.** A footprint-based thin-structure path complements the center-ray thick path, aggregating mask and depth evidence over each voxel's full projected extent to recover fine structures such as Trex ribs, fern fronds, and wires that single-pixel center-ray voting drops, which we find is especially critical for the surface-only 2DGS representation.
- **Efficient 2DGS voxel-to-surfel mapping.** For the explicit 2DGS backbone, we adapt the tile-sorted rasterization machinery of 2DGS (Huang et al., 2024) into a coverage-based gather that back-projects the fused voxel grid to per-surfel labels. Beyond the borrowed tile scaffold, the mapping contributes a per-voxel view-coverage gate, a depth-free tile sort, a tight one-sigma coverage test, and an anisotropy (needle) filter. Because each surfel is scored by the foreground coverage over its own projected footprint, the mapping thresholds every surfel on its individual area rather than imposing the single strict global threshold used by the baselines, producing sharp, topologically accurate surfel masks at interactive rates without per-primitive feature training.

Together, these choices keep the per-session compute bounded to a small set of anchor and centroid views that scales with object complexity rather than with the training-set size N_{train}, and hold peak VRAM under 15 GB on both backbones, within the consumer-hardware envelope that the data-center footprints of SAGA and Gaussian Grouping exceed (Section 4.2).

The remainder of the paper is structured as follows. Section 2 reviews prior work in NeRF segmentation, GS segmentation, and 2D foundation-model lifting. Section 3 presents the shared skeleton and its 2DGS instantiation in detail, with the NeRF instantiation provided in the appendix as evidence that the recipe transfers. Section 3.5.2 details the shared CUDA voxel-fusion kernel and its thin-structure path. Section 4 reports quantitative and qualitative results on both benchmarks, comparisons against SA3D, SA3D-GS, SAGA, and ablation studies.

2 Related Work

2.1 3D Scene Representations

Neural Radiance Fields (NeRF) (Mildenhall et al., 2020) represent scenes implicitly and synthesize novel views through differentiable volume rendering, with later variants improving efficiency, training speed, and unbounded-scene support (Müller et al., 2022; Chen et al., 2022; Garbin et al., 2021; Fridovich-Keil et al., 2022; Barron et al., 2021). We build on Instant-NGP (Müller et al., 2022) for its real-time rendering, which keeps interactive segmentation responsive. 3D Gaussian Splatting (3DGS) (Kerbl et al., 2023) instead represents the scene as an explicit set of anisotropic primitives rasterized through a tile-based differentiable pipeline, trading NeRF's continuous density field for a sparse, surface-attached primitive cloud with real-time high-resolution rendering. 2D Gaussian Splatting (2DGS) (Huang et al., 2024) specializes these primitives to flat surfels aligned with the surface, improving multi-view depth consistency. Because 2DGS surfels live only on the visible surface, the rasterizer returns a single planar median depth per pixel rather than a volume-rendered integral, and the per-pixel splat count n_{samp} is a natural proxy for local depth uncertainty. These two signals, planar depth and splat count, are the contact points exploited by the 2DGS instantiation of our framework.

2.2 2D Segmentation Foundation Models

2D segmentation has progressed from CNN-based models (Badrinarayanan et al., 2017; Chen et al., 2017) to transformer-based architectures that leverage global context (Dosovitskiy, 2020; Xie et al., 2021; Liu et al., 2021; Strudel et al., 2021). The Segment Anything Model (SAM) (Kirillov et al., 2023) established a foundation for zero-shot, prompt-based segmentation, with subsequent efficiency-oriented refinements (Zhang et al., 2023; Xiong et al., 2024). Transferring SAM's 2D capabilities into 3D remains an open question, and our work bridges this gap by integrating SAM's zero-shot priors with a representation-agnostic framework that operates over either a NeRF density field or 2DGS surfels.

2.3 3D Segmentation in NeRFs

Early works to segment NeRFs, such as Semantic-NeRF (Zhi et al., 2021), trained a separate semantic field alongside density and color, and NeSF (Vora et al., 2021) extends this idea to a generalizable 3D semantic segmentation network learned over the density field from only 2D semantic supervision. Both target fixed semantic categories under supervision rather than interactive, zero-shot selection of arbitrary objects. Other methods like NVOS (Ren et al., 2022), SA3D (Cen et al., 2023), and ISRF (Goel et al., 2023) proposed interactive segmentation but typically require per-scene fine-tuning or optimization. A parallel line of work focuses on feature alignment N3F (Tschernezki et al., 2022), DFF (Kobayashi et al., 2022), and on projecting 2D semantic features into the NeRF space, which can struggle with feature ambiguity and require retraining. LERF (Kerr et al., 2023) fuses CLIP (Radford et al., 2021) features with the 3D scene to enable language-driven queries, but targets coarse, language-based concepts rather than high-precision interactive segmentation of arbitrary objects.

The most closely related NeRF-side baselines to DivAS are SA3D (Cen et al., 2023) and SANeRF-HQ (Liu et al., 2024), which also combine SAM with 3D representations. SA3D lifts 2D SAM masks to NeRF and employs a self-prompting mechanism to propagate segmentation across views, but errors in the initial mask or occluded regions can easily propagate. SANeRF-HQ (Liu et al., 2024) formulates the problem as a per-scene optimization, training a refinement network over SAM masks. This sacrifices SAM's zero-shot generalization and incurs significant computation time. In contrast, DivAS is fully optimization-free, preserves SAM's zero-shot capability through a depth-weighted refinement step guided by NeRF geometry, and replaces slow optimization with a real-time CUDA-accelerated voxel-fusion kernel that aggregates geometrically consistent masks.

2.4 3D Segmentation in Gaussian Splatting

Per-primitive learned features. Gaussian Grouping (Ye et al., 2024) attaches a learnable identity feature to every Gaussian primitive and trains it against video-object-segmentation pseudo-labels obtained from a 2D tracker. At query time, primitives are grouped by feature similarity. SAGA (Cen et al., 2025) pushes this idea further, distilling SAM features into a per-Gaussian semantic vector and clustering similar primitives by feature affinity at query time. Both methods commit to a per-primitive learned feature volume that must remain resident with the Gaussian set during optimization and inference, and both run a per-scene optimization pass whose cost scales with the number of Gaussians and the number of training cameras. LangSplat (Qin et al., 2024) attaches CLIP (Radford et al., 2021) language features to 3D Gaussians for language-driven queries, sharing the same per-primitive feature-volume cost.

Mask-score optimization. SA3D-GS (Cen et al., 2024) replays the SA3D inverse-rendering loop on top of Gaussian primitives by optimizing an unbounded per-Gaussian mask score against rendered SAM masks. Because the mask score is unconstrained, peripheral high-transparency Gaussians acquire inflated scores and, after thresholding, materialize as floater artifacts in the final 3D mask, degrading IoU even when the optimization itself converges.

Common limitations. Every method above is optimization-based, commits to the 3DGS backbone, and offers no direct migration path to NeRF except SA3D. These methods must also keep learnable per-primitive features co-resident with the primitive cloud during the optimization pass. This coexistence constraint pushes the VRAM peak well beyond the envelope of consumer GPUs. SAGA explicitly works around this by

operating at $\times 8$ image downsampling on consumer hardware, sacrificing linear resolution and thin-structure fidelity. DivAS sidesteps all three limitations on the GS representation. No per-primitive features are trained, no mask score is optimized, and it provides a representation-agnostic solution for both NeRF and GS.

2.5 Cross-Representation Interactive 3D Segmentation

Two lines of prior work span multiple 3D representations. SA3D (Cen et al., 2023) has been ported from NeRF to 3DGS as SA3D GS (Cen et al., 2024), but the port reuses the same unbounded mask-score optimization and yields visibly poorer segmentation quality on both benchmarks (Section 4.2). Closer in spirit is SAMa (Fischer et al., 2026), which targets material-aware selection rather than object segmentation. SAMa lifts the 2D predictions of a SAM-like model into a depth-projected 3D point cloud that acts as a representation-agnostic bridge, so a selection transfers to a NeRF, a 3DGS scene, or a mesh through nearest-neighbor lookups, optimization-free and at interactive speed. SAMa shows that a single depth-based selection pipeline can span multiple representations without per-asset optimization, and we regard it as the closest existing point of comparison to our representation-agnostic goal.

DivAS shares this optimization-free, depth-based, cross-representation philosophy, but differs in the tasks it addresses and the geometric machinery it uses. First, DivAS performs object-level interactive segmentation rather than material-part decomposition, so the selection target and the evaluation protocol differ. Second, where SAMa uses depth only to build a point-cloud bridge and resolves selections by nearest-neighbor transfer, DivAS uses the rendered depth of the trained scene as a per-mask weighting and as a geometric-consistency gate during multi-view voxel voting, aggregating evidence into a probabilistic occupancy grid that is mapped back to NeRF density or to 2DGS surfels through representation-specific adapters. Third, DivAS bounds the user-effort budget to a fixed number of anchor and centroid views that is independent of the training-set size N_{train}, and it operates at the standard $\times 4$ downsampled image resolution within a 7–15 GB VRAM footprint, whereas the optimization-based GS baselines SAGA and Gaussian Grouping must keep per-Gaussian features resident and drop to $\times 8$ downsampling on consumer hardware, sacrificing thin-structure fidelity. We therefore position DivAS as an object-level interactive segmentation framework that combines depth-weighted voxel aggregation with a shared cross-representation interaction pipeline, enabling optimization-free segmentation on both NeRF and 2DGS scene representations.

3 DivAS

3.1 Overview

We present **DivAS**, a training-free, human-in-the-loop framework for interactive 3D segmentation built as a *shared interaction-and-fusion framework with representation-specific adapters*. A single algorithmic skeleton operates over both volumetric NeRF using InstaNGP and explicit 2D Gaussian-Splatting (2DGS) representations as backbones, with only the rendering and occupancy adapters specialized per representation. DivAS couples the open-world generalization of a 2D foundation model (SAM (Kirillov et al., 2023)) with the geometric consistency of a trained 3D scene, and requires no per-scene optimization, no learned per-primitive mask features, and no 2D mask tracker. As illustrated for the 2DGS instantiation in Figure 1, with the corresponding NeRF instantiation in Appendix A.1 (Figure 8), a DivAS session proceeds through stages that are conceptually identical across representations. The user marks a few non-overlapping object regions with point prompts on global *anchor views* (Section 3.2). Each prompt spawns a zoomed-in novel *centroid view* centered on the selected point using the novel view rendering property of these 3D representations, which increases the object's apparent scale and is refined to be free of occluders (Section 3.3). On each centroid view, SAM masks are reweighted by the rendered depth so that pixels inconsistent with the prompt geometry are suppressed (Section 3.4). The refined masks are aggregated by a single CUDA kernel that votes per voxel under geometric consistency checks, producing a probabilistic *occupancy grid* (Section 3.5). For the explicit 2DGS backbone, this grid is then mapped back to per-surfel labels (Section 3.6). Finally, the fused 3D mask is projected onto the next view as feedback, and the loop repeats until coverage is complete (Section 3.7).

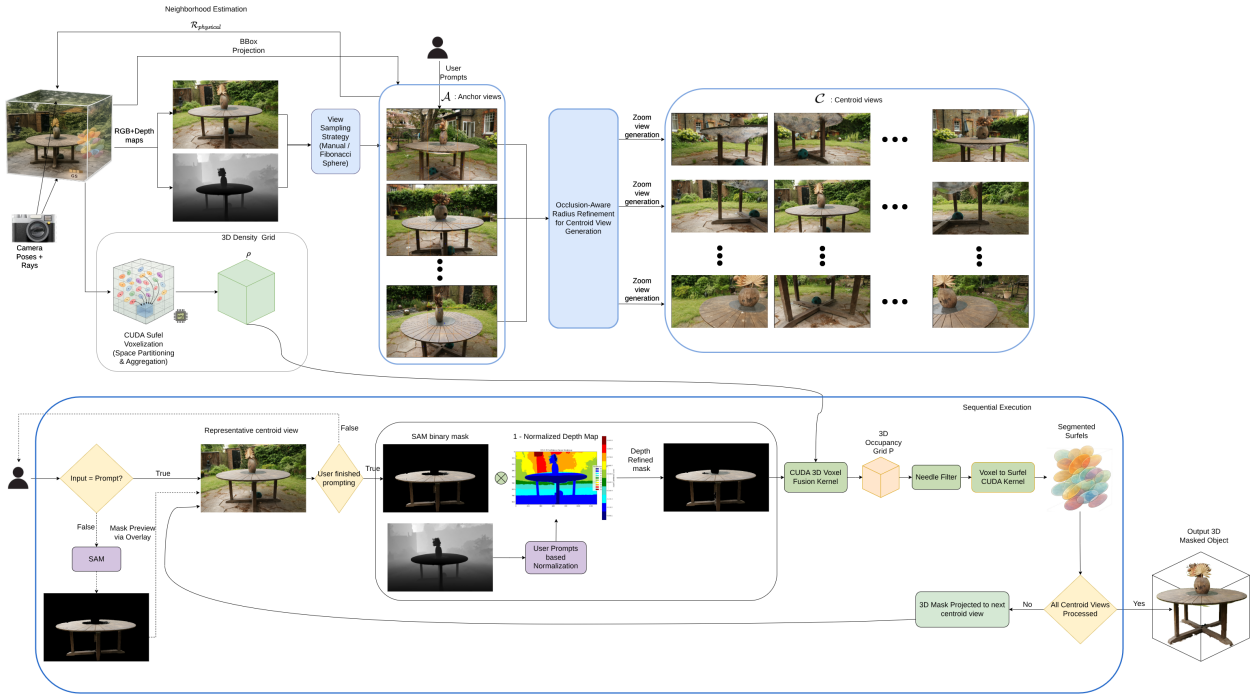


Figure 1: **2DGS instantiation of DivAS.** An occupancy grid is first *scattered* from anisotropic Gaussian surfels and rendered through surfel rasterization to obtain RGB and depth buffers. Unlike NeRFs, explicit 2DGS geometry is highly sensitive to sparse directional coverage and view-dependent occlusions; the Fibonacci anchor-view scheduler therefore adapts each sampling radius to remain within the valid training-camera shell and avoid unsupported viewpoints. An occlusion-aware centroid-view refinement stage further generates zoomed centroid views that suppress floaters and incomplete geometry. During interaction, SAM provides real-time mask previews as the user refines prompts in each centroid view, after which the resulting masks are fused synchronously using a CUDA-based voxel aggregation kernel. The fused occupancy grid is then projected back onto the Gaussian representation using a needle filter, which suppresses anisotropic sliver artifacts and preserves geometric continuity across surfels, followed by a voxel-to-surfel mapping kernel. The corresponding NeRF instantiation and its pipeline details are provided in Appendix A.1 (Figure 8).

Preliminaries. A NeRF scene representation models the scene as a continuous volumetric radiance field parameterized by a multi-resolution hash-grid encoding (Müller et al., 2022). Given a 3D position $\boldsymbol{x} \in \mathbb{R}^3$ and viewing direction $\boldsymbol{d}$, a lightweight MLP predicts a volume density $\sigma(\boldsymbol{x})$ and view-dependent color $c(\boldsymbol{x}, \boldsymbol{d})$. Rendering proceeds through differentiable volume rendering, where densities and colors sampled along each camera ray are composited using alpha accumulation to produce the final RGB image and depth map.

A 2D Gaussian Splatting model (Huang et al., 2024) represents the scene as a set $\mathbb{G} = \{g_i\}_{i=1}^{m}$ of oriented 2D surfels. Each surfel carries a center $\boldsymbol{\mu}_i \in \mathbb{R}^3$, a unit quaternion q_i encoding the surfel orientation, two tangent scales $(s_{i,1}, s_{i,2})$, an opacity α_i, and view-dependent color coefficients. Rendering proceeds by splatting each surfel onto the image plane and alpha-blending in front-to-back order along the ray.

Throughout, we denote the occupancy grid by a density field ρ on a Morton-indexed voxel grid whose edge length we write Δ_x for the NeRF multi-resolution grid (cascade-dependent, $\Delta_x = 2B_\ell/G$ at level ℓ) and h_v for the single-cascade 2DGS grid (a constant adaptive value per scene), the per-voxel segmentation probability by $P(v) \in [0, 1]$, the raw and depth-refined SAM masks for view s by $\boldsymbol{M}_s$ and $\widetilde{\boldsymbol{M}}_s$, and the rendered per-pixel depth buffer by the tuple $(\boldsymbol{D}_{\min}, \boldsymbol{D}_{\text{ref}}, \boldsymbol{D}_{\max}, \boldsymbol{D}_{\text{mw}}, \boldsymbol{N}_{\text{samp}})$, where $\boldsymbol{D}_{\min}$ is the depth of the first significant weighted sample, $\boldsymbol{D}_{\max}$ is the depth of the sample when cumulative density reaches τ_{cw} beyond which background primitves could be sampled, $\boldsymbol{D}_{\text{mw}}$ is the max-weighted sample depth, $\boldsymbol{D}_{\text{ref}}$ is the backbone's primary surface estimate ($\boldsymbol{D}_{\text{exp}}$, the expected depth, for NeRF; $\boldsymbol{D}_{\text{med}}$, the median depth, for 2DGS) and $\boldsymbol{N}_{\text{samp}}$ stores the number of primitives traversed up to the surface. For set of views the

depth tuple is represented by $(\{\boldsymbol{D}_{\min}^{(s)}\}_{s=1}^{S}, \{\boldsymbol{D}_{\text{ref}}^{(s)}\}_{s=1}^{S}, \{\boldsymbol{D}_{\max}^{(s)}\}_{s=1}^{S}, \{\boldsymbol{D}_{\text{mw}}^{(s)}\}_{s=1}^{S}, \{\boldsymbol{N}_{\text{samp}}^{(s)}\}_{s=1}^{S})$ where symbols have usual meanings.

The proposed framework maintains a unified interaction, propagation, and geometric-consensus formulation across both NeRF and Gaussian-Splatting scene representations. While the high-level pipeline remains shared, several core components, including rendering, occupancy construction, and multi-view geometric fusion, require representation-specific realizations due to the fundamentally different geometric primitives underlying volumetric radiance fields and anisotropic Gaussian splats. Accordingly, the remainder of this section is organized component-wise: each subsection first introduces the shared formulation and then expands the representation-specific implementations where the underlying computational or geometric behavior differs substantially.

3.2 Anchor View Selection

Shared sampling and ranking. DivAS supports both bounded forward-facing scenes and unbounded 360° scenes. For forward-facing captures, a small set of manually selected anchor views covers the object with minimal camera motion, since most of the geometry is visible from a narrow cone of viewpoints. For unbounded scenes, uniform angular coverage is required, and we generate a compact, low-discrepancy scene-specific candidate set with *Fibonacci-sphere* sampling (González, 2010). This yields uniform spherical coverage in $O(N)$, even for small N (we use N=12).

Candidate views are oriented toward the scene center and then ranked by a geometric *informativeness* score balancing three factors: (i) *diversity* D_i, the mean angular separation to the other views. (ii) *cardinal coverage* C_i, proximity to the canonical axes $\{\pm\mathbf{x}, \pm\mathbf{y}, \pm\mathbf{z}\}$ and (iii) *pitch extremity* which favors informative top-down and bottom-up perspectives that improve 3D understanding. The score for a view is a weighted sum: $I(i) = 0.4D_i + 0.3C_i + 0.3P_i$ The top-ranked views are retained as anchor viewpoints for subsequent interaction. Since the ranking is purely geometric and independent of object semantics, the GUI provides a toggle to draw anchors from the *full* Fibonacci set, allowing the user to prioritize views that better expose an occluded or off-center target in multi-object or object-centric scenes.

Representation-aware anchor placement. Both backbones place the K retained anchors on a sphere centered at $\mathbf{c}_{\text{scene}}$ along the Fibonacci directions but differ in the *radius* at which each anchor is placed, and this difference is dictated by how each representation behaves away from the training cameras. For NeRF, a single global radius equal to the median training-camera distance suffices in every direction. The property of NeRF being bounded, normalized in a cube, its volumetric field renders a coherent image from any point on that shell and degrades gracefully even where training views are sparse, with no hard occlusion penalty. The explicit 2DGS representation is substantially more sensitive to occlusions and incomplete directional coverage than volumetric NeRFs. Because anisotropic surfels are only reliably reconstructed within the shell spanned by the training cameras, using a fixed global sampling radius may place anchor views in unsupported regions, leading to floaters, holes, and severe view-dependent artifacts. We therefore make the anchor radius adaptive for each Fibonacci direction,

$$r_i = \min\big(r_i^{\text{knn}}, r_i^{\text{cap}}\big), \tag{1}$$

where r_i^{knn} estimates a locally representative radius from nearby training-camera density, and r_i^{cap} constrains the anchor within the valid directional camera shell. This adaptive formulation keeps anchor views both geometrically representative and within reliably reconstructed regions, substantially improving robustness under sparse coverage and view-dependent occlusions. Since 2DGS camera poses are not canonically aligned during preprocessing, we additionally estimate a global scene up-direction before generating the Fibonacci sampling sphere. Further implementation details are deferred to the supplementary material.

Representation-specific rendering. The segmentation framework is agnostic to how a view is rendered. Only the rendering adapter differs. For **NeRF**, RGB and depth are produced by volumetric ray-marching of the trained density field, with the depth buffer read directly from the ray-integration loop. In NeRF, the expected depth D_{exp} represents where the surface lies in the density cloud marched by the ray. For **2DGS**, the depth buffer is produced by surfel rasterization. In 2DGS, median depth D_{med} is the primary surface proxy because it is robust to floaters near the camera.

3.3 Centroid-View Expansion and Zoom Refinement

Why centroid views are necessary. SAM's mask decoder produces a low-resolution (256×256) logit map, which is upsampled to the input resolution. When the target occupies a small fraction of a global anchor view or an image in the dataset, a common case for thin or distant structures, this coarse decoder produces ragged boundaries and systematic false negatives in fine regions. To counter this, every point prompt in anchor view spawns a *centroid view*. A zoomed-in novel render in which the back-projected 3D click point is set as the camera's look-at target and the camera is moved toward it, enlarging the object's apparent scale. Magnifying the object in the image plane directly mitigates the decoder's resolution bottleneck and improves recall on thin structures. The full centroid set across A anchors and K prompts per anchor is:

$$\mathcal{C} = \bigcup_{i=1}^{A} \bigcup_{j=1}^{K} \mathcal{A}_{ij}. \tag{2}$$

2DGS instantiation: occlusion-aware centroid refinement. The fixed-fraction zoom does not transfer to 2DGS, where scenes are unbounded and a constant push can drive the camera through the target or into foreground floaters. We instead size each centroid view physically from the local surfel neighborhood around the click, then refine the orbital radius with an occlusion-aware probe that pulls the camera in until the target is no longer dominated by occluders. The neighborhood radius estimate, the lateral-spread sizing, the dynamic occlusion test, and the per-click algorithm are detailed in Appendix C.3.

Iterative centroid processing. Centroid views are processed one at a time, and after each fusion, the running 3D mask is projected back as an overlay (Section 3.7), so the user re-prompts only regions not yet covered. This converts the set $\mathcal{C}$ of Equation 2 into a sequence in which each new view both adds a fresh viewpoint and corrects residual gaps from prior views, the mechanism by which sparse user input expands to complete object coverage.

3.4 Depth-Guided Interactive Segmentation

Shared interaction loop. Centroid views improve SAM's precision by enlarging object scale, yet SAM often produces false positives and boundary artifacts at this scale. Similar observations were made in SANeRF-HQ (Liu et al., 2024), which reported that segmentation errors in both 2D and 3D are more likely to occur near object boundaries. Motivated by this, we refine SAM masks using the rendered depth map from 3D representations as geometric priors. For each centroid, the user supplies positive (and optionally negative) point prompts. SAM produces a candidate mask $\boldsymbol{M}_s$, mapped to a $[0, 1]$ confidence by the sigmoid. We then reweight $\boldsymbol{M}_s$ by the rendered depth, so that pixels whose depth is inconsistent with the prompted surface are suppressed, producing the depth-refined mask $\widetilde{\boldsymbol{M}}_s$ used for fusion. The running fused 3D mask is overlaid on the current view so the user re-prompts only uncaptured regions in an interactive correction loop rather than a one-shot prediction. Two facets of this loop differ by backbone, and we present them as a single systems-design trade-off driven by rendering speed and scene boundedness, not as two separate pipelines.

Execution scheduling: asynchronous (NeRF) vs. synchronous (2DGS). The dominant interactive cost varies across backbones, so the optimal SAM scheduling differs accordingly. **NeRF** rendering is slow, so SAM inference is cheap by comparison and easy to hide: DivAS-NeRF runs SAM *asynchronously* in a background worker, processing the preceding centroid views $\{c_{i-1}, c_{i-2}\}$ while the user annotates c_i, and transitions from asynchronous to sequential execution only when (i) c_i is the last view of its anchor group, or (ii) the cache holds at least three pending masks. At that point, pending inferences are synchronized and fused together as shown in Figure 8. **2DGS** rendering is fast enough that each centroid view is generated in milliseconds compared to 1-2s taken by NeRF, so a *synchronous* call enables immediately incorporating the resulting mask into the interaction loop.

Mask-channel selection and view acceptance. SAM emits three masks per view, ordered by ambiguity. DivAS-GS always retains the single mid-granularity mask at channel index 1, which was the most consistent

across centroid views because each centroid view focuses on a part of an object and inherently gets masked by the mid-granularity mask of SAM. A view is *rejected* (its mask discarded) if it carries no positive prompt, or if SAM's channel-1 score is below 0.5, indicating SAM is not committed to a salient object and any forced mask would be noise. In this case, this centroid view is not considered in the fusion step.

Depth-weighting: bounded normalization (NeRF) vs. band-pass (2DGS). DivAS-NeRF weights the SAM mask by the inverse of the min-max-normalized expected depth,

$$\widetilde{\boldsymbol{M}}_s = \boldsymbol{M}_s \odot \big(1 - \hat{\boldsymbol{Z}}_s\big), \qquad \hat{\boldsymbol{Z}}_s = \frac{\boldsymbol{D}_{\text{exp}} - \min(\boldsymbol{D}_{\text{exp}})}{\max(\boldsymbol{D}_{\text{exp}}) - \min(\boldsymbol{D}_{\text{exp}})}, \tag{3}$$

which provides well-defined foreground-background separation precisely because the trained AABB clips each ray to a bounded, artifact-free depth range, suppressing distant pixels and avoiding large depth values, thereby preventing squashing of the normalized range. This rule *cannot* transfer to 2DGS where surfels at sky distance or on far facades leave the median-depth raster uncapped, so a global min-max normalization (or any single-anchor depth falloff) collapses foreground and near-background to nearly identical weights and the depth separation vanishes (Figure 2a). Because the object's depth extent is not known a priori, there is no AABB to bracket where the object lives. DivAS-GS turns the user's own prompts into a depth *band*. Reading the median depth at every positive prompt and at the view center (a near-free anchor, since the centroid view looks directly at the target along the orbital axis), we form

$$\mathbb{D}_s = \big\{\boldsymbol{D}^{(s)}_{\text{med}}(p_{x,k})\big\}_{k=1}^{\mathbb{P}_s} \cup \big\{\boldsymbol{D}^{(s)}_{\text{med}}(\tfrac{w}{2}, \tfrac{h}{2})\big\}, \quad d^{(s)}_{\min} = \min \mathbb{D}_s,\ d^{(s)}_{\max} = \max \mathbb{D}_s. \tag{4}$$

A robust far estimate $d_{\text{far}} = \mathcal{P}_{98}(\boldsymbol{D}^{(s)}_{\text{med}})$ discards the top 2% tail (sky, far facades, reconstruction holes) so the falloff slope $1/(d_{\text{far}} - d^{(s)}_{\max})$ stays bounded and dimensionally consistent across scenes of very different scale. Pixels inside the band keep full weight. And, pixels outside fall off linearly toward d_{far}:

$$\begin{aligned} \boldsymbol{\Delta}_- &= \max(d^{(s)}_{\min} - \boldsymbol{D}_{\text{med}}, 0), \quad \boldsymbol{\Delta}_+ = \max(\boldsymbol{D}_{\text{med}} - d^{(s)}_{\max}, 0), \\ \boldsymbol{D}_{\text{band}} &= \text{clip}\Big(\tfrac{\boldsymbol{\Delta}_- + \boldsymbol{\Delta}_+}{\max(d_{\text{far}} - d^{(s)}_{\max},\, 10^{-6})},\, 0,\, 1\Big), \qquad \widetilde{\boldsymbol{M}}_s = \boldsymbol{M}_s\,(1 - \boldsymbol{D}_{\text{band}}). \end{aligned} \tag{5}$$

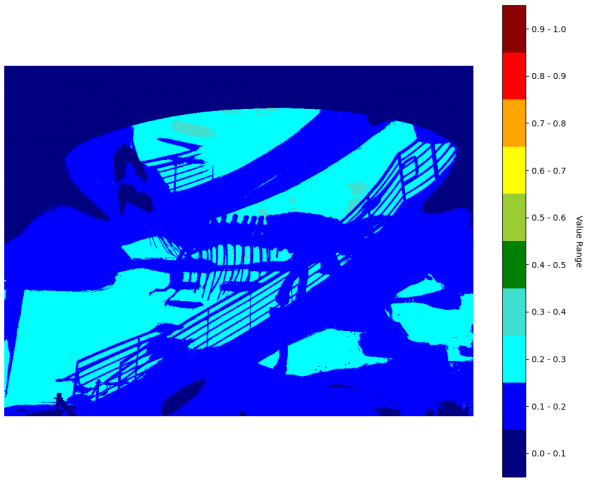
At most one of $\boldsymbol{\Delta}_-, \boldsymbol{\Delta}_+$ is nonzero per pixel. A pixel at d_{far} receives weight 0 and one at the band edge weight 1, and the floored denominator prevents division blow-up when the object spans to the far edge (reducing the weight to a pure inside/outside indicator). The band keeps a spatially extended object, e.g., a Trex skeleton spanning meters between a rib click and a skull click at unit weight, while still rejecting background whose median depth lies outside the prompted band (Figure 2b). The band-pass thus recovers normalized depth for the unbounded 2DGS case, the role that bounded normalization plays for NeRF.

3.5 Representation-Specific 3D Fusion

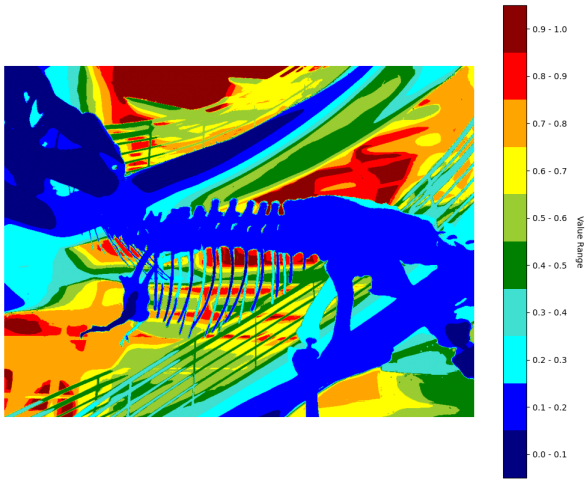
Shared voxel-voting concept. The fusion stage aggregates the depth-refined masks of all retained views into the occupancy grid. In both backbones, a single CUDA kernel assigns *one thread per voxel*. For active voxel v (world center $\boldsymbol{x}_v$, edge h_v) and view s, it projects $\boldsymbol{x}_v$ into view s, reads the mask confidence and depth channels at the projected pixel, and admits a vote only if the voxel is both *spatially* and *depth* consistent with the surface the renderer reports there. Accepted votes are accumulated with a boundary-attenuating depth weight that is maximal when the voxel projects to the center of the surface segment and decays toward its edges. Voxels missed by this center-aligned check are re-examined by a *thin-structure* path that aggregates evidence over the voxel's full 2D footprint. After all S views, the per-voxel probability is the depth-weighted, thresholded consensus over a thick-path view set $\mathbb{C}_v$ and a thin-path set $\mathbb{T}_v$:

$$P(v) = \begin{cases} \dfrac{\displaystyle\sum_{c\in\mathbb{C}_v} m^{(c)}_v\, w^{(c)}_v \;+\; \sum_{t\in\mathbb{T}_v} \mathbb{1}\big[t^{(t)}_v \geq \tau_{\text{thin}}\big]\, t^{(t)}_v}{\displaystyle\sum_{c\in\mathbb{C}_v} w^{(c)}_v \;+\; \sum_{t\in\mathbb{T}_v} \mathbb{1}\big[t^{(t)}_v \geq \tau_{\text{thin}}\big]}, & \text{denom} > \varepsilon, \\ 0, & \text{otherwise}, \end{cases} \tag{6}$$

$$\mathbb{V}_{\text{seg}} = \big\{\, v : P(v) \geq \tau_{\text{cons}} \,\big\},$$



(a) NeRF-style min-max normalization on a 2DGS depth raster, far background compresses the range and the cue collapses.



(b) Click-defined band $[d^{(s)}_{\min}, d^{(s)}_{\max}]$ with $\mathcal{P}_{98}$ far estimate, the full object stays at unit weight.

Figure 2: Depth-weighting for 2DGS (Eq. 5). The prompt-defined band replaces NeRF's min-max normalization and is robust to the unbounded depth range of 2DGS scenes.

where $m_v^{(c)}$ is the refined SAM confidence at the projected pixel, $w_v^{(c)}$ the depth weight, $t_v^{(t)}$ the thin-structure score, and τ_{thin} is the thin-structure acceptance threshold, and $\mathbb{V}_{\text{seg}}$ the set of voxels reported as segmented at the consensus threshold τ_{cons}. Equation 6 the *compact fusion* rule is shared verbatim. What differs is how the occupancy grid ρ is constructed and how the depth-consistency block is expressed in each backbone's native depth semantics. We detail the two realizations below.

3.5.1 NeRF backbone 3D Fusion

For the NeRF backbone, occupancy comes from the ray-marched density field, and the depth-consistency block of Equation 6 is expressed in the renderer's per-pixel depth tuple $(\boldsymbol{D}_{\min}, \boldsymbol{D}_{\exp}, \boldsymbol{D}_{\max}, \boldsymbol{D}_{\text{mw}}, \boldsymbol{N}_{\text{samp}})$. A voxel votes through the thick path when its center projects within an adaptive spatial and depth tolerance of the rendered surface, weighted by a Gaussian falloff from the surface midpoint. Voxels whose center falls between the sparse mask-active pixels of a thin structure are recovered by a footprint-based thin path that aggregates mask and depth evidence over the voxel's full projected bounding box. The full derivation, the spatial and depth tolerances, the depth-weighted vote, the thin-structure coverage test, and the two kernel algorithms appear in Appendix A.2.

3.5.2 Gaussian-Splatting Surfel Voxel Fusion

Unlike the volumetric NeRF backbone, the 2DGS backbone operates on an explicit, anisotropic, and unbounded surfel representation, so the NeRF fusion kernel cannot be applied directly. The GS instantiation introduces representation-specific adaptations that address surfel anisotropy, thin-structure handling, and surfel-to-voxel propagation. These adaptations constitute the primary technical contribution of the 2DGS extension.

Occupancy by Morton-scattered opacity. The 2DGS occupancy grid is a scene-adaptive compression layer that turns millions of explicit surfels into a compact voxel field on which a single interactive segmentation kernel is feasible. Its design goal is to derive every quantity, the center, the extent, the voxel edge, and the occupancy threshold, from per-scene statistics so that it generalizes across scene scales without manual tuning.

(i) Robust scene normalization. We first estimate a stable working frame directly from the surfel cloud using percentile-based estimators. The scene center $\boldsymbol{c}_{\text{scene}}$ is the *coordinate-wise median* of the Gaussian centers $\{\boldsymbol{\mu}_i\}_{i=1}^m$, and the scene extent is the 99*th percentile* of the radial distance to that center to avoid floaters,

$$\boldsymbol{c}_{\text{scene}} = \text{median}_i(\boldsymbol{\mu}_i), \qquad r_{\text{fg}} = \mathcal{P}_{99}\big(\{\|\boldsymbol{\mu}_i - \boldsymbol{c}_{\text{scene}}\|\}_{i=1}^m\big), \qquad b = \lceil 1.05\, r_{\text{fg}} \rceil, \tag{7}$$

The coordinate-wise median and the 99th-percentile radius are insensitive to this outlier tail, so a few floaters shift neither the working origin $\boldsymbol{c}_{\text{scene}}$ nor the symmetric cube half-extent b, which carries a five-percent margin to absorb borderline surfels. Gaussians outside the cube are removed by an AABB filter, and all downstream work proceeds in the re-centered frame.

The voxel edge is then set automatically from per-scene Gaussian statistics, combining the surfel-local resolution, the median Gaussian scale $\bar{s}$, with a scene-global lower bound:

$$h_v = \max\big(\alpha_v\, \bar{s},\ \beta_v\, r_{\text{scene}}/g_0\big), \qquad (\alpha_v,\, \beta_v,\, g_0) = (3.0,\, 1.0,\, 256), \tag{8}$$

The first term $\alpha_v \bar{s}$ follows the Gaussian 3σ rule so that a typical surfel lands inside one voxel rather than fragmenting across neighbors, while the second term $\beta_v r_{\text{scene}}/g_0$ caps grid growth on large scenes to keep GPU memory bounded. Their maximum balances geometric fidelity against memory with no scene-specific tuning, and the resolution g is the smallest power of two with $g \geq 2b/h_v$, capped at $g_{\max}$.

Surviving surfels are aggregated into the density grid $\boldsymbol{\rho}$ by a Morton-indexed opacity scatter-add. A voxel acts as an aggregator that stores only the summed opacity of its surfels, not their identities, and per-surfel labels are recovered later by re-projecting the segmented grid (Section 3.6). Voxels above a transmittance floor τ_{TF} form a compact active-voxel list, in practice well under one percent of the dense g^3 grid, and every downstream kernel runs only on this list. This aggregation is what makes a single CUDA fusion kernel interactive. The working state collapses from millions of primitives to a sparse scalar field. This launch cost scales with foreground complexity rather than grid volume. Segmentation is decoupled from costly per-surfel visibility reasoning, and the construction transfers across scene scales because all of its quantities are per-scene statistics.

Planar-to-ray depth correction. Both renderers report depth as camera-Z (planar), whereas thick-path visibility is evaluated along the viewing ray. For an off-axis pixel the two differ by the obliquity factor $\cos\theta_i$, the cosine of the angle between the camera-forward axis $\hat{\boldsymbol{e}}^i_{\text{fwd}}$ of view i and the unit ray direction $\hat{\boldsymbol{d}}_i$ through the projected pixel. We recover the ray-space depths by dividing each planar depth by this factor,

$$\cos\theta_i = \langle \hat{\boldsymbol{e}}^i_{\text{fwd}},\, \hat{\boldsymbol{d}}_i \rangle, \qquad d^{\text{ray}}_{\bullet} = \frac{d_{\bullet}}{\cos\theta_i}, \qquad \bullet \in \{\text{min, med, max}\}. \tag{9}$$

For NeRF this mismatch is largely absorbed by the thin-structure fallback, so its segmentation stays stable. For 2DGS, routing visible voxels through the thin path is undesirable, because it inflates fusion cost and weakens the discriminative power of the thick-path checks on anisotropic-surfel depths. We therefore apply Equation 9 before the visibility test, so that the thick path correctly handles visible surfel-supported geometry, while the thin path is reserved for genuinely sub-voxel structures such as ribs and fern fronds. The resulting ray-space depths $d^{\text{ray}}_{\min}, d^{\text{ray}}_{\text{med}}, d^{\text{ray}}_{\max}$ feed the spatial gate of Equation 13.

Footprint statistics via integral images. The per-pixel splat-traversal count n_{samp} from the 2DGS rasterizer is the only scene-geometry signal the kernel sees about how the visible surface is built at a pixel, and two GS-specific effects make it informative only when summarized over the voxel's projected footprint rather than read at the center pixel. First, depth localization, NeRF fills object interiors, so its sample count grows with depth thickness and warrants *relaxing* the depth tolerance, whereas 2DGS fits surfels only on the visible surface, so a high count signals stacked surfels and *less* certain depth, which warrants the opposite reaction. Second, layered pixels: at narrow inter-part gaps (fern fronds, Trex, ribs, bonsai flowers) a ray crosses both the foreground splats bordering the gap and the background behind, giving an anomalously large n_{samp} that flags a false-positive foreground pixel whose depth reference must switch from d_{med} to the front-hit $d_{\min}$. Both effects need the local mean $\bar{n}_{\text{samp}}$ and standard deviation σ_n over each voxel's footprint, which we obtain in $O(1)$ per voxel from per-view integral images of n_{samp} and n^2_{samp}.

Layered-pixel and see-through detection. Two pixel-level conditions signal that the median depth d_{med} is *not* the visible surface and must be replaced by the front-hit $d_{\min}$:

$$\text{layered} \equiv n_{\text{samp}} > \bar{n}_{\text{samp}} + \sigma_n, \qquad \text{seethrough} \equiv d_{\text{med}} > \gamma_{\text{ST}}\, d_{\text{cam}}, \tag{10}$$

The layered predicate flags a splat-count anomaly relative to the local footprint, while the see-through predicate is a transmittance-gap test anchored on the orbital camera-target distance d_{cam}, which is scene-scale invariant and behaves consistently from LLFF to Mip-NeRF 360° with a single user-tunable γ_{ST}. Both arise from surfel anisotropy under oblique views, a ray can enter a slanted surfel near its front while most of its opacity lies deeper, so $d_{\min} > d_{\text{med}}$, and rather than trust an ambiguous central vote, we defer such voxels to the behind-surface gate and the thin path.

Asymmetric depth-reference and depth gate. When the pixel is either layered or see-through (and $d_{\min}$ is trusted), the reference depth switches from d_{med} to the front surface $d_{\min}$. The depth gate becomes one-sided:

$$\text{valid_depth} = \begin{cases} (x_d - d_{\min}) \le t_{\text{depth}}, & \text{layered or see-through,} \\ \|x_d - d_{\text{med}}\| \le t_{\text{depth}}, & \text{otherwise.} \end{cases} \tag{11}$$

The one-sided form $(x_d - d_{\min})$, rather than $\|x_d - d_{\min}\|$ is deliberate. Behind a thin transmissive front, a genuine object voxel still has $x_d > d_{\min}$, and an absolute value would silently reject it. Empirically, this single change removes the distant false positives seen on the flower and leaf structures of *Bonsai* and *Garden*.

Per-sample depth-tolerance tightening. In 2DGS, a high per-pixel splat count signals stacked surfels and therefore *less* certain depth, so the tolerance must *tighten* where the count is anomalously high. We define the tolerance from the local *excess* $E = \max(n_{\text{samp}} - \bar{n}_{\text{samp}}, 0)$:

$$\eta = \frac{1}{1 + \beta E}, \qquad t_{\text{depth}} = \gamma\, h_v\, \eta. \tag{12}$$

where β, γ are fixed across datasets and h_v is the voxel size. Tightening on the excess rather than the raw n_{samp} prevents uniformly thick objects, where every footprint pixel has a similar count, from being penalized across the board, while still precisely shrinking the band where stacked surfels make the depth ambiguous.

Spatial gate. In the spatial check, the voxel center is projected onto the camera ray, $t^\star = \langle \boldsymbol{x}_c - \boldsymbol{o}_i, \hat{\boldsymbol{d}}_i \rangle$, and clamped to the valid surface segment, $\hat{t} = \text{clip}(t^\star, d^{\text{ray}}_{\min}, d^{\text{ray}}_{\text{ref}})$, with $d^{\text{ray}}_{\text{ref}} = d^{\text{ray}}_{\min}$ when the pixel is layered or see-through and $d^{\text{ray}}_{\text{med}}$ otherwise. Here, $d^{\text{ray}}_{\min}, d^{\text{ray}}_{\text{med}}$ are obtained from the planar depths $d_{\min}, d_{\text{med}}$ through Equation 9. The clamped surface point $\boldsymbol{p}^\star = \boldsymbol{o}_i + \hat{t}\hat{\boldsymbol{d}}_i$ must lie within the depth-gradient-modulated tolerance,

$$\|\boldsymbol{x}_c - \boldsymbol{p}^\star\| \le h_v\, \pi_{\text{dg}}(u^i_x, v^i_x), \tag{13}$$

with the depth-gradient factor π_{dg} that relaxes the tolerance on flat surfaces and tightens it at depth discontinuities.

Depth-weighted vote and view-coverage bitmask. A voxel that clears both gates votes with a boundary-attenuating Gaussian weight that is maximal at the surface-segment center and decays toward its edges,

$$w^{(i)}_{\text{depth}} = \exp(-\alpha_1\, \hat{r}^2), \qquad \hat{r} = \frac{\|\hat{t} - d^{\text{ray}}_{\text{ctr}}\|}{d^{\text{ray}}_{\text{hw}}}, \tag{14}$$

with segment center $d^{\text{ray}}_{\text{ctr}} = \frac{1}{2}(d^{\text{ray}}_{\min} + d^{\text{ray}}_{\max})$ and half-width $d^{\text{ray}}_{\text{hw}} = \frac{1}{2}(d^{\text{ray}}_{\max} - d^{\text{ray}}_{\min})$ where $d^{\text{ray}}_{\max}$ is ray-space equivalent of planar depth $d_{\max}$. On every accepted thick vote, the kernel updates the running accumulators *and* sets bit i of the per-voxel view-coverage bitmask $\boldsymbol{v}_x$ for the voxel x,

$$W_x \mathrel{+}= m^i_x\, w^{(i)}_{\text{depth}}, \qquad Z_x \mathrel{+}= w^{(i)}_{\text{depth}}, \qquad \boldsymbol{v}_x \mathrel{|}= (1 \ll i), \tag{15}$$

recording which views actually voted for x. This bitmask has no NeRF analogue and is the bridge from voxel space to surfel space (Section 3.6).

Behind-surface gate before the thin path. Before the thin-path fallback, an asymmetric far-gate removes voxels lying strictly behind the visible surface:

$$d_{\text{surf}} = \begin{cases} d_{\text{min}}, & d_{\text{min}} \text{ trusted and see-through,} \\ d_{\text{med}}, & \text{otherwise,} \end{cases} \qquad \text{behind-fail} \equiv x_d > d_{\text{surf}} + 2\,h_v. \tag{16}$$

Rejecting only voxels more than two voxel edges behind the surface, while leaving voxels in front of a far background pixel eligible, discards the bulk of the behind-surface false positives that the central pixel could not see, yet keeps genuine thin foreground that the central pixel happened to miss reachable by the thin path.

Thin-structure path. The thick path tests consistency at the voxel center only, so slender high-density structures (Trex ribs, bonsai flowers) project to too few center-aligned pixels and are silently rejected. The thin path instead aggregates depth- and mask-consistent evidence over the voxel's full projected footprint, gated by eligibility on the behind-surface test and the thin-structure density floor $\rho_{\text{thin_thresh}}$. Over the footprint, the kernel keeps two counts, the depth-consistent pixels n_{depth} and the SAM-and-depth-consistent pixels $n_{m\wedge d}$, and, to guard against 2DGS depths drifting along oblique surfaces, it unprojects each mask-positive pixel and tracks its squared distance to the voxel center together with the peak SAM confidence,

$$d^2_{\text{min-to-surf}} \leftarrow \min\!\big(d^2_{\text{min-to-surf}},\, \|\boldsymbol{p}_w^{(x_p,y_p)} - \boldsymbol{x}_c\|^2\big), \qquad m_{\max} \leftarrow \max(m_{\max}, \widetilde{\boldsymbol{M}}_i(x_p, y_p)) \tag{17}$$

The unprojection has no NeRF analogue and is essential, since even depth-consistent pixels can unproject far from the voxel center on a tilted surface, and the distance term prevents certifying voxels whose footprint coverage is real but geometrically detached.

Degeneracy floor, distance gate, final score. Three conditions close the thin path. First, the depth-consistent pixels must contain sufficient support,

$$n_{\text{depth}} \geq n_{\min}, \qquad n_{\min} = \max\!\big(n_{\text{abs}},\, \lceil \rho_{\text{occ}}\, A_{\mathbb{B}} \rceil\big), \tag{18}$$

where $A_{\mathbb{B}} = |\mathbb{B}^i_x|$ is the pixel area of the projected footprint $\mathbb{B}^i_x$. The absolute floor n_{abs}=25 suppresses sparsely sampled footprints, and the occupancy fraction ρ_{occ}=0.1 requires larger footprints to contribute proportionally more evidence, together rejecting isolated pixels and rasterization artifacts. Second, the coverage ratio is taken over the depth-consistent support,

$$p_{\text{covered}} = \frac{n_{m\wedge d}}{n_{\text{depth}}}. \tag{19}$$

Dividing by n_{depth} rather than by the footprint area $A_{\mathbb{B}}$ prevents large geometrically irrelevant regions (background, occluders) from deflating the ratio, so it answers "among the footprint parts that could plausibly belong to this voxel, how many does SAM call foreground?". Third, the closest mask-valid surface intersection must be geometrically near the voxel center. When all three hold, the thin score is the peak footprint confidence, otherwise zero:

$$t^i_x = \begin{cases} m_{\max}, & n_{\text{depth}} \geq n_{\min} \;\wedge\; p_{\text{covered}} \geq \rho_{\text{cover}} \;\wedge\; d_{\text{min-to-surf}} \leq (1.5\,h_v)^2, \\ 0, & \text{otherwise.} \end{cases} \tag{20}$$

The thin path enters the compact-fusion accumulator with the value t^i_x. The complete implementation, the preprocessing pipeline, and the CUDA execution scheme are provided in Appendix B, and algorithms in Algorithm 3, Algorithm 4, and Appendix B.5.

3.6 Voxel-to-Surfel Mapping and Aggregation

Why per-surfel labels are needed. For the volumetric NeRF backbone the segmented voxel grid *is* the output. For the explicit 2DGS backbone, however, the deliverable is a per-Gaussian foreground indicator $\boldsymbol{f} \in \{0,1\}^M$, used both to recolor surfels for the live overlay and to produce the final masked render,

and a naive "label every Gaussian that intersects a segmented voxel" is geometrically inadequate because voxelization quantizes each anisotropic surfel to a single home voxel, so a disc that legitimately overlaps a segmented voxel may have its center in an adjacent background voxel, and elongated surfels fitted inside one voxel extend far beyond it and project as visible spikes when their home voxel is background. We therefore build $\boldsymbol{f}$ as a monotone chain of three filter passes, $\boldsymbol{f}^{(1)} \succeq \boldsymbol{f}^{(2)} \succeq \boldsymbol{f}$, each of which only turns surfels off: Pass 1 admits a permissive starting set via a Morton-indexed bitfield lookup, Pass 2 culls per-scene needle floaters, and Pass 3 culls residual boundary bleeders by footprint-coverage analysis, yielding the final indicator $\boldsymbol{f} \equiv \boldsymbol{f}^{(3)}$(we use $\boldsymbol{f}$ and $\boldsymbol{f}^{(3)}$ interchangeably in what follows).

Pass 1: permissive Morton-indexed admission. To guarantee the mapping is geometrically identical to grid construction, so that a Gaussian is never assigned to a voxel it was not voxelized into, we re-execute the same NaN/AABB filter and Morton encoding on the live Gaussian set and read the segmentation bitfield $\mathbf{B}^{\text{seg}}$ packed from $\mathbf{P}$:

$$\boldsymbol{\mu}_i^{\text{shft}} = \boldsymbol{\mu}_i - \boldsymbol{c}_{\text{scene}}, \quad m_i = \texttt{morton3D}\big(\lfloor(\boldsymbol{\mu}_i^{\text{shft}} + b)/h_v\rfloor\big), \quad \boldsymbol{f}_i^{(1)} = \mathbf{B}^{\text{seg}}_{m_i}. \tag{21}$$

The pass is deliberately permissive: it admits every Gaussian whose center falls in a segmented voxel, including boundary surfels whose center sits just inside a segmented voxel while much of their splatted footprint projects to the background. Leaving these in would bleed into the masked render, so their removal is deferred to later passes. Pass 1 mainly shrinks the candidate set to surfels on or near the object's surface.

Pass 2: scene-adaptive needle-floater culling. 2DGS training minimizes photometric loss without object awareness, so it leaves needle-like surfels whose aspect ratio

$$a_i = \max(s_{i,1}, s_{i,2})/(\min(s_{i,1}, s_{i,2}) + \epsilon) \tag{22}$$

exceeds roughly 1000. These violate the one-surfel per-voxel assumption used to size h_v, spike density in unrelated voxels through their long axis, and project as visible spikes even when their opacity contribution to the segmented voxel is negligibl e. A fixed cull threshold either over-prunes or under-prunes across scenes, so we use a self-calibrating 99th-percentile threshold

$$\tau_a = \mathcal{P}_{99}(\{a_i\}_{i=1}^{M}), \tag{23}$$

and apply it as

$$\mathbb{F}_i^{(2)} = \mathbb{F}_i^{(1)} \wedge \big(a_i \le \tau_a\big). \tag{24}$$

Anchoring on the per-scene aspect distribution always discards exactly its top 1% tail, and the cull is applied only to the segmentation-time surfel pool, not the photorealistic render. We place it at the mapping stage rather than at grid construction because a needle's harm to segmentation quality appears only when the masked viewpoint is rendered, and removing needles first also lowers the dominant Pass 3 gather cost.

Pass 3: footprint-coverage culling of boundary bleeders. The remaining failure mode is boundary bleeders and surfels surviving Pass 2, whose splatted disc mostly covers pixels that SAM never labeled as foreground. Pass 3 measures this coverage directly with a tile-sorted gather that reuses the 2DGS rasterizer's tile decomposition but is semantically inverted: it counts coverage rather than blending color, so it carries no depth ordering, and its tile-sort keys hold only the tile id,

$$\text{key}_{(t,i)} = \text{tile_id}(t) \ll 32, \tag{25}$$

Two design choices make the coverage geometrically faithful and thin-feature preserving. First, each candidate surfel is gated by the per-voxel view-coverage bitmask $\boldsymbol{v}$ (Equation 15), so a surfel is scored in a view only when its mapped voxel was actually voted positive there, restricting evidence to genuinely contributing views. Second, the gather enumerates pixels within a tight 1σ projected box rather than the rasterizer's 3σ box: thin and distal structures (chair legs, fern fronds, Trex ribs) have a 1σ core that lands cleanly on the SAM mask but a $1\sigma \to 3\sigma$ tail that bleeds into background, so restricting the count to the core both preserves these features and decisively culls near-boundary background surfels whose core lies off the mask,

while the per-pixel acceptance inside the box stays the rasterizer's exact 3σ Cramér ellipse test. On each accepted pixel, the kernel accumulates two per-surfel counts, the visited footprint pixels and those SAM labels foreground,

$$\text{total_fp}_i \mathrel{+}= 1, \qquad \text{if } \widetilde{\boldsymbol{M}}_s(x_p, y_p) \geq \tau_M \text{then mask_fp}_i \mathrel{+}= 1 \tag{26}$$

with no opacity, transmittance, or alpha-blending. The per-surfel coverage is the foreground fraction over visited pixels, folded across views by a running maximum,

$$\kappa_s^{(i)} = \frac{\text{mask_fp}_i^{(s)}}{\max(\text{total_fp}_i^{(s)}, 1)}, \qquad \kappa_i \leftarrow \max\big(\kappa_i,\, \kappa_s^{(i)}\big). \tag{27}$$

where the maximum, rather than the mean, is taken because a single clean view is sufficient evidence of membership and averaging would penalize surfels occluded in other views. The final indicator then rejects bleeders below a coverage threshold τ_κ,

$$\mathbb{F}_i = \mathbb{F}_i^{(2)} \wedge \big(\kappa_i \geq \tau_\kappa\big) \tag{28}$$

Because κ_i measures each surfel's own foreground footprint coverage, τ_κ thresholds every surfel on its individual area rather than applying a single strict global score threshold across all primitives, as the optimization-based GS baselines do. Foreground membership is instead settled upstream by the multi-view voxel-occupancy consensus, so τ_κ acts purely as a per-surfel coverage cull and not as the background-rejection knob, a decoupling we analyze under render-resolution change in Appendix E. After this, the masked render recolors the segmented Gaussians and composites the thresholded overlay on the RGB shown to the user after each centroid view. The complete implementation, the preprocessing, the tile-sorted gather pipeline, and the CUDA execution are provided in Appendix B.4, with the full pseudocode in Algorithm 5 (Appendix B.5). The chain preserves geometric continuity and topology while enforcing surfel-level mask consistency.

3.7 Iterative Mask Propagation

Shared propagation. After fusing the current view, DivAS projects the fused 3D occupancy back onto the next centroid view and overlays it, so the user immediately sees what has been captured and re-prompts only the missing regions. Views are processed sequentially, each fusion both incorporates a new viewpoint and corrects residual gaps from earlier views, so sparse user input progressively expands to complete object coverage. The session terminates once all centroid views are processed, and the segmented voxels (NeRF) or surfels (2DGS) are returned as the final 3D mask.

Representation-specific execution. The backbones differ only in how the overlay is produced and timed, mirroring the SAM-scheduling trade-off of Section 3.4. DivAS-NeRF relies on the *asynchronous* SAM cache. Masks accumulate in the background and are fused in batches, so propagation feedback occurs only when a group synchronizes. DivAS-GS propagates in *real time*, so after every centroid view, it runs the fusion kernel and the three-pass voxel-to-surfel mapping, recoloring segmented Gaussians for an immediate live overlay. In both cases, the per-update segmentation kernel stays below 70 ms across all benchmark scenes (Table 21); this is the only per-update cost, and it grows with the number of committed centroid views rather than with N_{train}.

In summary, DivAS is a single interactive-segmentation framework whose view scheduler, SAM interface, geometry-aware voxel fusion, and propagation loop are shared, and whose only representation-specific components are the rendering and occupancy adapters. The NeRF and 2DGS instantiations differ in concrete, well-motivated ways fixed-fraction vs. occlusion-aware centroid views, bounded normalization vs. band-pass depth weighting, asynchronous vs. synchronous SAM, ray-marched density vs. scatter-and-map occupancy, and a 2DGS-only voxel-to-surfel stage. But these are realizations of one formulation, not two pipelines.

4 Experiments

4.1 Experimental Setup

Datasets. We evaluate our method on two types of scenes: bounded and unbounded. For bounded scenes, we use the LLFF (Mildenhall et al., 2019) dataset, which contains multiple forward-facing real-world scenes. For unbounded scenes, we use the Mip-NeRF 360° (Barron et al., 2022) dataset, which includes complex, large-scale environments. To enable quantitative evaluation, we use the NVOS (Ren et al., 2022) dataset. NVOS provides a hand-curated ground-truth mask for a single test view per scene in LLFF. For Mip-NeRF 360°, no true mask exists, so following SANeRF-HQ Liu et al. (2024), we evaluate on the publicly released Mip-NeRF 360 benchmark masks distributed by the authors. These masks are part of the benchmark package and were generated using the annotation pipeline described in SANeRF-HQ rather than being the inference outputs of SANeRF-HQ itself. All methods are evaluated against the same released masks. We considered 6 representative scenes from each dataset.

Baselines. We instantiate the DivAS framework on two scene representations and refer to the two instantiations as **DivAS** (InstaNGP (Müller et al., 2022) backbone) and **DivAS-GS** (2DGS (Huang et al., 2024) backbone). We compare them against five representative optimization-based 3D-segmentation baselines that span both representations. On the **NeRF** side: (a) **SA3D** (Cen et al., 2023), which lifts SAM masks into a TensoRF (Chen et al., 2022) representation through per-scene mask-inverse-rendering optimization, (b) **SANeRF-HQ** (Liu et al., 2024), which aggregates SAM masks in a high-quality InstaNGP (Müller et al., 2022) feature field. On the **Gaussian-Splatting** side: (c) **Gaussian Grouping** (Ye et al., 2024), which trains a per-Gaussian identity embedding supervised by a 2D mask tracker (DEVA), (d) **SAGA** (Cen et al., 2025), which learns a contrastive affinity feature field over the Gaussian scene and (e) **SA3D-GS** (Cen et al., 2024), the 3DGS (Kerbl et al., 2023) backbone variant of SA3D, which optimizes a per-Gaussian mask score by inverse rendering. We evaluate the official `sa3d-gs` implementation released by the original SA3D authors through the public SegmentAnythingin3D repository. Since no separate peer-reviewed publication exists for this extension, we follow the released implementation and default settings provided by the authors. DivAS and DivAS-GS are both training-free instantiations of the single framework we introduce in this paper, and we report them together to demonstrate that the same pipeline transfers across scene representations rather than to serve as external points of comparison. All five baselines are run with their official open-source implementations and default hyperparameters. Gaussian Grouping, in particular, incurs a workflow cost that the IoU numbers alone do not capture. After the per-scene identity field is trained, the user has no GUI access to select a desired 3D object and must perform a linear scan of the rendered identity map across the scene's dataset images to identify which identity IDs correspond to the target. A single object is frequently split across multiple IDs (in the *Garden* scene, we had to collect over a dozen disjoint IDs). These identity IDs are an artifact of Gaussian Grouping's own training and are specific to each trained model. Our reported Gaussian Grouping numbers apply a "Union Merging" step at evaluation time, merging all these foreground IDs into the final mask so that the results represent the *best-case* performance of the method. A naive user might miss some identities, leading to holes in the final 3D mask. A timing asymmetry exists for methods that do not provide an integrated interactive interface. In SA3D-GS, the user supplies the initial object pixel before propagation begins, and the reported runtime therefore excludes the time required to identify the target location in the rendered scene. Similarly, Gaussian Grouping assumes that foreground Gaussian identities have already been identified from the rendered identity maps prior to execution, and this manual selection step is not included in the reported runtime. By contrast, SAGA and DivAS integrate target selection into the interactive workflow, so user interaction is naturally included in the measured end-to-end time. We follow the original evaluation protocols of all methods when reporting runtimes.

Evaluation protocol. We evaluate each method at its recommended operating point and explicitly report the corresponding configuration. For methods whose performance depends on render resolution or threshold selection, we additionally provide a sensitivity analysis. For DivAS, SA3D, SANeRF-HQ, DivAS-GS, SA3D-GS, and Gaussian Grouping, the predicted mask is rendered *directly at the native ground-truth (full) resolution* from the scene representation, against the NVOS ground-truth masks, no post-hoc upsampling of the predicted mask is performed at any stage. SAGA, by contrast, is strongest at its training resolution

rather than at full resolution. We therefore report SAGA at its train-resolution operating point, and to keep the comparison free of information-free interpolation, we downsample the ground-truth mask to the render resolution rather than upsampling the prediction. We thus report a sensitivity analysis (Section E) that sweeps over resolution and threshold for all three density-thresholding methods. For DivAS-GS, the voxel-occupancy grid construction (~ 500 ms) is launched asynchronously and overlaps with the SAM model and UI loading at session start. It therefore introduces no additional time for preprocessing and is not counted separately in any reported timing.

Metrics. For segmentation quality, we report test-view **IoU** and pixel **accuracy** against the ground-truth mask in the LLFF dataset. We report mIoU (mean IoU) and mAcc (mean Accuracy) averaged across all evaluation views per scene in the Mip-360° dataset. For efficiency, we report two metrics that indicate the method's memory and time cost throughout its pipeline. First, the End-to-End Time from scene construction to 3D mask generation. This is because methods like Gaussian Grouping learn identity encodings per-Gaussian during scene reconstruction, whereas SA3D-GS alters the densification rule defined by the vanilla 3DGS method for segmentation. Second, the **peak VRAM** usage. For every method, we report **peak VRAM** as the maximum GPU memory observed while running it on a given scene (for the staged baselines SAGA, SA3D-GS, and Gaussian Grouping, this is the maximum taken across all stages of their pipeline), so that hardware-feasibility claims reflect the true worst-case footprint.

Bounded view processing Unlike propagation-based methods that traverse all training views, DivAS processes only a small set of anchor and centroid views selected during interaction across its both instantiations. Table 1 reports the range of training images available per dataset, together with the number of centroid views processed by DivAS during a segmentation session. Although Mip-NeRF 360° scenes contain up to 301 training images, DivAS processes at most 20 centroid views during segmentation. Similarly, LLFF scenes require only 5-10 processed centroid views despite containing up to 61 training images. This bounded view-processing budget is reflected in the runtime results reported in Section H.

Table 1: Training images available versus centroid views actually processed by DivAS during a segmentation session. Despite a large variation in capture size, the processed-view budget stays small and bounded, independent of the training-set size.

Dataset	Training Views	Processed Centroid Views
LLFF	19-61	5-10
Mip-NeRF 360°	175-301	8-20

4.2 Quantitative Results

Table 2 gives the overall summary of the results. It reports each method at its best stable operating point, showing both benchmarks, peak memory, and end-to-end runtime in a single view. The per-scene breakdowns that follow support every entry. DivAS-GS and Gaussian Grouping at full (native) resolution, SAGA at its stronger train-resolution setting, and SA3D-GS at full resolution. Gaussian Grouping has the worst performance across the three axes, i.e., IoU/mIoU scores, Peak VRAM, and End-to-End Time. SAGA dominates the mIoU score in the Mip-NeRF 360° dataset, second best in the LLFF dataset, but has the maximum Peak VRAM and End-to-End Time close to the Gaussian Grouping time. SA3D-GS performs worst in segmentation quality, with the lowest IoU/mIoU scores, but remains efficient, ranking second-best in Peak VRAM and least in End-to-End Time. Our proposed method, DivAS-GS, achieves optimal performance across the three axes: the best IoU on the LLFF dataset, the second-best mIoU, the least VRAM, and the second-best End-to-End Time, with segmentation quality close to SOTA on Mip-NeRF 360° and better on LLFF.

We compare DivAS (NeRF backbone) and DivAS-GS (2DGS backbone) against the baselines on both benchmarks. Tables 3 and 4 report segmentation quality (IoU and accuracy) per scene for all seven methods, grouped by representation backbone. Tables 5 and 6 then report, *per scene*, the efficiency by end-to-end time

Table 2: Overall benchmark summary. Each method is reported at the operating point that gives its best *stable* quantitative performance (Section 4.1). Eval. Res. is Evaluation Resolution. End-to-End Time is in minutes:seconds (mm:ss). Second best per column is underline, Best per column is **bold**.

Method	Eval. Res.	Mip-NeRF 360° mIoU ↑	LLFF IoU ↑	Peak VRAM (GB) ↓	End-to-End Time (mm:ss, 360° / LLFF) ↓
Gaussian Grouping (Ye et al., 2024)	full	0.8764	0.8534	60.95	53:20 / 34:15
SAGA (Cen et al., 2025)	train	**0.9594**	0.9151	65.52	52:42 / 33:28
SA3D-GS (Cen et al., 2024)	full	0.6512	0.7502	16.96	**22:41 / 17:08**
DivAS-GS (ours)	full	0.9476	**0.9230**	**14.40**	23:55 / 20:03

Table 3: Quantitative comparison on the Mip-NeRF 360° dataset. We report mIoU and mAcc scores for different methods. Best (**bold**) and second-best (underlined) are marked *within each representation block*, not across blocks.

	NeRF backbone						**Gaussian-Splatting backbone**							
	SA3D		SANeRF-HQ		DivAS (ours)		Gaussian Grouping		SAGA		SA3D-GS		DivAS-GS (ours)	
Scene	mIoU ↑	mAcc ↑	mIoU ↑	mAcc ↑	mIoU ↑	mAcc ↑	mIoU ↑	mAcc ↑	mIoU ↑	mAcc ↑	mIoU ↑	mAcc ↑	mIoU ↑	mAcc ↑
Bonsai	0.9120	**0.9910**	0.9060	0.9890	**0.9200**	**0.9910**	0.6012	0.9555	0.9351	0.9929	0.5608	0.9492	**0.9396**	**0.9932**
Garden (no vase)	0.9030	0.9820	**0.9520**	**0.9910**	0.9430	0.9900	**0.9745**	**0.9956**	0.9694	0.9948	0.7769	0.9542	0.9620	0.9933
Garden (with vase)	0.7580	0.9540	0.9280	**0.9880**	**0.9360**	**0.9880**	0.9567	0.9925	**0.9726**	**0.9952**	0.6672	0.9229	0.9584	0.9926
Kitchen (lego)	**0.9700**	**0.9960**	0.8360	0.9780	0.9170	0.9880	0.9570	0.9942	**0.9703**	**0.9959**	0.8306	0.9719	0.9569	0.9940
Counter (flowerpot)	0.3640	0.9520	**0.9020**	**0.9940**	0.8900	0.9930	0.9438	0.9958	**0.9515**	**0.9968**	0.4589	0.9522	0.9423	0.9966
Room (chair)	0.8310	0.9770	**0.9520**	**0.9990**	0.8700	0.9960	0.8254	0.9943	**0.9576**	**0.9987**	0.6126	0.9761	0.9262	0.9976
Mean	0.7900	0.9750	**0.9130**	0.9900	**0.9130**	**0.9910**	0.8764	0.9880	**0.9594**	**0.9957**	0.6512	0.9544	0.9476	0.9946

and peak VRAM for the Gaussian-Splatting methods. We report this per-scene to show the efficiency gain from DivAS-GS. Peak VRAM, in particular, shows that a scene that demands 60 GB dictates the required GPU class, despite the low mean.

Backbone of the Gaussian-Splatting comparison. Throughout the Gaussian-Splatting comparison, DivAS-GS is evaluated on a 2DGS (Huang et al., 2024) reconstruction, while SAGA, Gaussian Grouping, and SA3D-GS are evaluated using their official 3DGS (Kerbl et al., 2023) implementations. We therefore interpret the reported results as properties of complete segmentation systems rather than isolated segmentation modules. Importantly, the same interaction-and-fusion skeleton also achieves competitive performance against SA3D and SANeRF-HQ in the NeRF setting, suggesting that the observed gains are not solely attributable to the 2DGS backbone.

Table 3 represents the segmentation quality on the Mip-NeRF 360° dataset, across both scene representations. All methods are evaluated against the same SANeRF-HQ pseudo-ground-truth masks. The Gaussian-Splatting methods follow the best-stable protocol of Section 4.1, where SAGA is evaluated at training resolution. Under the NeRF backbone, SA3D performs well in simple object-centric settings (Kitchen), but struggles in complex, multi-material scenes (Garden with a vase) or in occluded scenes (Counter with a flowerpot), where single-view prompts fail to capture the target region across multiple viewpoints. Our approach maintains geometric consistency under such multi-material, occluded scenes by fusing multi-view cues via a 3D voxel kernel, resulting in smoother, more coherent 3D segmentation. Because the Mip-NeRF 360° references are SANeRF-HQ pseudo-labels rather than manually annotated ground truth, they occasionally penalize our IoU in scenes like Counter. The qualitative comparison (see Figure 15) shows that these discrepancies tend to coincide with imperfect pseudo-labels around thin or partially occluded structures, where our predictions remain geometrically consistent with the visible object boundaries. Overall, our method matches the quantitative accuracy of SANeRF-HQ (Liu et al., 2024) while being multi-view consistent. This demonstrates the effectiveness of amortized (i.e., optimization-free) 3D fusion without per-scene optimization.

As shown in the GS backbone, SA3D-GS performs significantly worse than its NeRF counterpart, suggesting that the unbounded mask-score objective is more sensitive to floater artifacts and boundary leakage. Gaussian Grouping leads to inconsistent results. In the Garden scene, it achieves the maximum mIoU of 0.975, whereas in the Bonsai scene it drops to 0.60. This is because of the cumulative effect of SAM false negatives

Table 4: Quantitative comparison on the NVOS dataset. We report IoU and mAcc scores for different methods. Best(**bold**) and second-best (underlined) are marked within each representation block.

	NeRF backbone						Gaussian-Splatting backbone							
	SA3D		SANeRF-HQ		DivAS (ours)		Gaussian Grouping		SAGA		SA3D-GS		DivAS-GS (ours)	
Scene	IoU ↑	Acc ↑	IoU ↑	Acc ↑	IoU ↑	Acc ↑	IoU ↑	Acc ↑	IoU ↑	Acc ↑	IoU ↑	Acc ↑	IoU ↑	Acc ↑
Fern	0.8220	0.9430	**0.8250**	**0.9440**	0.8030	0.9330	0.8230	0.9420	**0.8403**	**0.9479**	0.7121	0.8842	0.8115	0.9374
Fortress	0.9690	0.9940	0.9720	**0.9950**	**0.9750**	**0.9950**	0.9797	0.9962	0.9808	0.9964	0.9442	0.9891	**0.9883**	**0.9978**
Horns (center)	0.9730	0.9950	0.9690	0.9950	**0.9780**	**0.9960**	0.9769	0.9960	**0.9805**	**0.9966**	0.8710	0.9747	0.9729	0.9953
Horns (left)	0.9360	**0.9960**	**0.9390**	**0.9960**	0.9350	**0.9960**	0.9262	0.9954	0.9285	0.9955	0.6428	0.9670	**0.9351**	**0.9960**
Orchids	0.8800	0.9790	0.7780	0.9560	**0.8950**	**0.9820**	0.8578	0.9749	0.9365	0.9894	0.6458	0.9388	**0.9408**	**0.9901**
Trex	0.8310	0.9770	0.8410	0.9790	**0.8690**	**0.9820**	0.5569	0.9430	0.8241	0.9755	0.6851	0.9478	**0.8891**	**0.9853**
Mean	0.9020	**0.9810**	0.8870	0.9780	**0.9090**	**0.9810**	0.8534	0.9746	0.9151	**0.9836**	0.7502	0.9503	**0.9230**	**0.9836**

and the inability of 2D tracker (DEVA) to propagate masks across the views for thin or repeated structures like bonsai petals. On scenes where the target is a single connected blob of solid material (*Garden-no-vase*, *Kitchen-lego*), the tracker is reliable, and Gaussian Grouping is competitive, occasionally beating DivAS-GS on raw mIoU. SAGA is marginally ahead of DivAS-GS, winning four of six scenes with a mean mIoU 0.01 higher. The largest gap is 0.03 in the Room (chair) scene, where the floor and chair legs are not well-defined due to poorly learned geometry and reconstruction, leading to ambiguity about whether the query voxel belongs to the chair leg or the floor.

Table 4 evaluates segmentation quality on the NVOS benchmark across both NeRF and Gaussian-Splatting-based scene representations. Following the official training protocol of each method, we use the original inference pipelines rather than the user interaction strokes provided by NVOS. Under the NeRF backbone block, SA3D achieves competitive quantitative performance, indicating that optimization over volumetric voxel geometry remains considerably more stable than its Gaussian-Splatting counterpart. SANeRF-HQ (Liu et al., 2024) shows a sharp performance drop in the Orchids scene. It incorrectly segments unprompted flower buds that visually resemble the user-selected flowers. This over-segmentation arises from its global optimization process, which lacks fine spatial prompt control. In contrast, our centroid-view design restricts segmentation to the prompted regions, enabling localized refinement. As a result, our method avoids false activations on nearby, visually similar structures. Our method, DivAS, achieves the highest mean IoU of 0.909 and comparable pixel accuracy of 0.981 among all baselines. Our method dominates in four out of six scenes, highlighting the advantage of interactive centroid-view refinement in reducing false negatives on fine object details.

Within the Gaussian-Splatting backbone, the performance gap between geometry-aware and feature-driven approaches becomes more pronounced on scenes with complex depth interactions. Similar to the observations on Mip-NeRF 360°, SA3D-GS performs substantially worse than its NeRF counterpart, suggesting that optimization over anisotropic Gaussian primitives is more sensitive to floater artifacts and boundary leakage. Gaussian Grouping shows degraded performance in the Trex scene. It is unable to segment Trex ribs and other parts. It confuses them with the background because of its proximity to the object parts in the image space. SAGA wins in two out of six scenes where there is a 0.02-point IoU score difference in the fern scene with DivAS-GS at 0.811 because of poor geometric separation between the background and the semi-transparent fern leaves, leading to background bleeding. SAGA optimizes this in image space at the cost of culling object parts, as shown in Figure 3. Our approach maintains geometric consistency and an induced depth prior in the SAM mask, achieving a winning score in 4 of 6 scenes, with a mean IoU that is 0.007 higher than the second-best SAGA method. The 0.065 large margin in the Trex scene of SAGA at 0.8254 is because depth-gated voxel aggregation of DivAS-GS separates the interleaved rib bones and background layers that SAGA's appearance-only affinity features merge. These results confirm that depth-guided SAM masks and voxel-level fusion yield stable and accurate segmentation across forward-facing scenes.

Tables 5 and 6 compare the computational efficiency of Gaussian-Splatting-based segmentation methods in terms of end-to-end time and peak GPU VRAM usage. In contrast to optimization-heavy feature-field approaches whose computational cost scales with prolonged feature optimization and scene-level processing, DivAS-GS maintains an amortized training-free pipeline with substantially lower end-to-end time overhead.

Table 5: Per-scene efficiency comparison on the Mip-NeRF 360° dataset. For each method, we report *E2E Time* (end-to-end wall-clock time) and *Peak VRAM*. Best is **bold**, second-best underlined.

	Gaussian Grouping		SAGA		SA3D-GS		DivAS-GS (ours)	
Scene	E2E Time (mm:ss)↓	Peak VRAM (GB)↓	E2E Time (mm:ss)↓	Peak VRAM (GB)↓	E2E Time (mm:ss)↓	Peak VRAM (GB)↓	E2E Time (mm:ss)↓	Peak VRAM (GB)↓
Bonsai	38:00	31.58	35:40	21.76	**15:07**	8.06	17:35	**7.98**
Garden (no vase)	80:00	60.95	81:27	65.52	**34:40**	16.96	36:55	**10.62**
Garden (with vase)	80:00	60.93	81:30	65.52	**34:41**	16.74	35:02	**14.40**
Kitchen (lego)	42:00	29.94	43:34	23.10	**19:01**	**9.19**	19:35	13.93
Counter (flowerpot)	35:00	22.39	36:47	24.17	**16:22**	**7.83**	17:25	7.93
Room (chair)	45:00	31.48	37:15	24.32	**16:14**	8.50	16:58	**7.00**
Mean / max-VRAM	53:20	60.95	52:42	65.52	**22:41**	16.96	23:55	**14.40**

Table 6: Per-scene efficiency comparison on the LLFF dataset. For each method, we report *E2E Time* (end-to-end wall-clock time) and *Peak VRAM*. Best is **bold**, second-best underlined.

	Gaussian Grouping		SAGA		SA3D-GS		DivAS-GS (ours)	
Scene	E2E Time (mm:ss)↓	Peak VRAM (GB)↓	E2E Time (mm:ss)↓	Peak VRAM (GB)↓	E2E Time (mm:ss)↓	Peak VRAM (GB)↓	E2E Time (mm:ss)↓	Peak VRAM (GB)↓
Fern	33:30	16.48	32:24	14.30	**17:50**	**6.64**	23:19	8.52
Fortress	33:30	14.57	28:34	11.52	**15:26**	**6.64**	17:06	7.93
Horns (center)	34:03	16.78	34:05	17.81	**18:09**	**6.64**	19:46	7.95
Horns (left)	34:03	16.78	33:34	17.81	18:05	**6.64**	**17:25**	6.94
Orchids	39:41	22.23	40:01	21.96	**18:11**	**6.69**	23:05	8.11
Trex	30:45	13.76	32:07	15.17	**15:07**	**6.64**	19:40	7.89
Mean / max-VRAM	34:15	22.23	33:28	21.96	**17:08**	**6.69**	20:03	8.52

On the Mip-NeRF 360° benchmark, DivAS-GS achieves segmentation quality comparable to the strongest-performing methods in Table 3 while requiring less than 25% of the peak VRAM consumed by SAGA and reducing end-to-end runtime by approximately 2×-2.2×. Compared to Gaussian Grouping, DivAS-GS provides competitive segmentation quality at a lower overall end-to-end cost. Although SA3D-GS achieves the lowest runtime and among the smallest memory footprints, its segmentation quality remains substantially lower than both SAGA and DivAS-GS.

A similar trend is observed on the LLFF dataset, where DivAS-GS maintains the strongest balance between segmentation quality and computational efficiency. Relative to SAGA and Gaussian Grouping, DivAS-GS reduces peak VRAM consumption to below 40% while achieving approximately 1.5×-1.7× faster end-to-end execution. While SA3D-GS remains computationally lightweight, its segmentation accuracy exhibits a substantial gap compared to the results reported in Table 4. Overall, these results demonstrate that DivAS-GS achieves competitive segmentation quality while remaining significantly more practical for deployment on consumer-grade GPUs than optimization-intensive feature-field baselines.

NeRF instantiation efficiency. We evaluate the end-to-end runtime of DivAS (NeRF instantiation) against the optimization-based baseline SANeRF-HQ (Liu et al., 2024) on a single NVIDIA A100 GPU. As shown in Table 7, DivAS achieves a consistent 1.7-2.1× speedup across diverse scenes. The gain stems from the optimization-free design. While SANeRF-HQ requires substantial computation to train object fields (about 3 minutes of non-interactive wait time), the DivAS CUDA fusion kernel incurs negligible latency, below 65 ms per call across all evaluated scenes (finer-grained kernel timings in Appendix H). The DivAS runtime is therefore dominated almost entirely by user interaction, enabling a fluid, real-time workflow that scales independently of the dataset size.

Render-resolution and threshold robustness. DivAS-GS is resolution-stable at its single canonical threshold (τ=0.1), with mean IoU changing by at most 0.30 points between full and training resolution, because it decides foreground membership upstream by multi-view voxel-occupancy consensus and uses τ only as a binarizer, whereas threshold-based baselines such as SAGA must balance background rejection

Table 7: **Total runtime comparison** (NeRF instantiation). Time in seconds. DivAS consistently outperforms the optimization-based baseline by eliminating the training bottleneck.

Scene	SANeRF-HQ	DivAS	Speedup
Fern (Bounded)	231.0	**120.0**	1.92×
Trex (Bounded)	327.5	**190.0**	1.72×
Bonsai (360°)	514.0	**258.0**	1.99×
Kitchen (360°)	516.0	**250.0**	2.06×

against thin-structure retention through a single τ. The full sensitivity table and analysis are provided in Appendix E.

4.3 Qualitative Analysis

For the NeRF instantiation, DivAS preserves both global structure and thin parts such as Trex ribs and hands while avoiding the over-segmentation of SANeRF-HQ and the single-view misses of SA3D, on both LLFF and Mip-NeRF 360°. The full qualitative comparisons (Figure 14 and Figure 15) are provided in Appendix F, consistent with our use of the NeRF backbone as evidence that the framework transfers across representations.

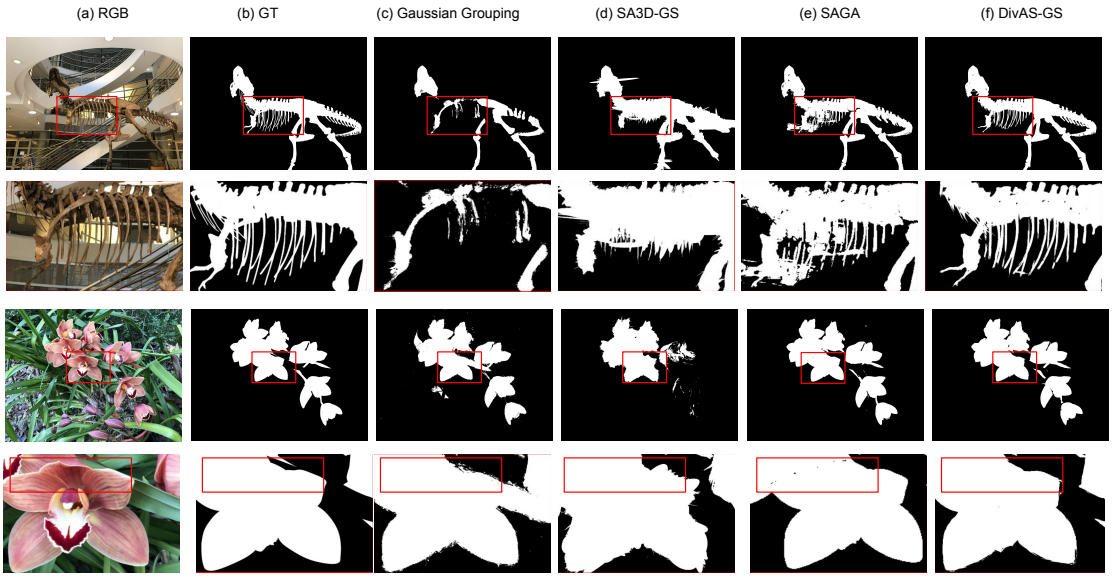


Figure 3: Results of different methods for GS backbone on LLFF.

Figure 3 visualizes segmentation results on the LLFF dataset for the Gaussian-Splatting backbone. In rows 1 and 2 on the *Trex* scene, Gaussian Grouping confuses the rib structures with the background, producing large false negatives in the mask. SA3D-GS misses global parts of the Trex such as the tail and simultaneously exhibits background bleeding into the rib region. We attribute this dual failure to the unbounded mask-score objective, which does not suppress the persistent floaters in its 3DGS (Kerbl et al., 2023) representation. SAGA preserves the global structure (tail, ribs) but cannot separate the background from the ribs in the absence of geometric consistency, producing over-segmentation around ribs and hands. Driven by the depth-weighted mask refinement and the centroid-view strategy, DivAS-GS stays closest to the ground-truth mask, free of the artifacts visible in the other three methods. Row 3 shows the *Orchids* scene in which Gaussian Grouping again suffers from background bleeding, while SA3D-GS fails to reconstruct all the orchids in the scene. In row 4, the red box highlights random holes introduced by SAGA within the solid body

of the object, the resolution-threshold pathology described in paragraph E, whereas DivAS-GS remains geometrically consistent, with no background bleeding or holes in the solid body.

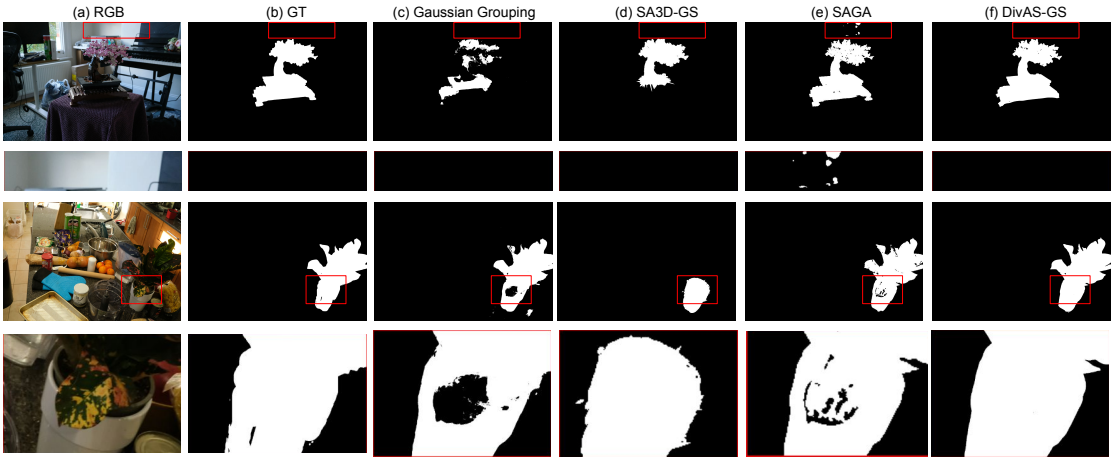


Figure 4: Results of different methods for GS backbone on Mip-NeRF 360°.

Figure 4 visualizes segmentation results on the Mip-NeRF 360° dataset for the Gaussian-Splatting backbone. In row 1, Gaussian Grouping misses the stem of the plant, confusing it with the background, while SA3D-GS fails to propagate its single-view prompt across viewpoints, producing under-segmentation that drops parts of the object. Row 2 illustrates the geometric inconsistency of SAGA on the *Bonsai* plant, where background bleeders penetrate the foreground. DivAS-GS remains geometrically consistent with no background bleeders and produces a binary mask close to the ground truth. On the *Counter (flowerpot)* scene in row 3, SA3D-GS misses the plant leaves due to the occlusion limitations of single-view prompting. In row 4, Gaussian Grouping treats a yellow leaf as background, producing a hole in the mask, while SAGA again exhibits random holes inside the solid body, degrading the resulting binary mask. DivAS-GS produces a geometrically consistent binary mask close to the references, which on this benchmark are SANeRF-HQ pseudo-labels rather than manual ground truth.

4.4 Ablation Studies

We present ablations on two design decisions shared across the NeRF and GS instantiations. Where the two instantiations produce qualitatively different behavior, we report and explain them separately.

Cumulative-weight threshold $\tau_{\mathbf{cw}}$. The cumulative-weight cutoff τ_{cw} controls the point along each ray at which the renderer declares "sufficient foreground density has been observed", determining the upper bound of the depth range $\boldsymbol{D}_{\max}$ fed to the fusion kernel. We sweep τ_{cw} over $\{0.15 - 0.9\}$ on four representative scenes *Trex* and *Fern* from LLFF, and *Bonsai* and *Garden (without vase)* from Mip-NeRF 360°.

NeRF instantiation The NeRF backbone behaves differently, its IoU stays stable for $\tau_{\text{cw}} \in [0.3, 0.75]$ and degrades at higher cutoffs as the widening valid depth range admits background voxels. We report the full NeRF sweep, figure, and analysis in Appendix G.1.

2DGS instantiation As shown in Figure 5, the 2DGS instantiation behaves *differently* from NeRF at high τ_{cw}: its IoU is essentially flat across the entire sweep, including the $\tau_{\text{cw}} \geq 0.9$ regime where the NeRF IoU drops sharply. In the 2DGS rasterizer, τ_{cw} controls how many of the surfel layers stacked along the ray (the cumulative alpha accumulation) are folded into $\boldsymbol{D}_{\max}$. Because 2DGS surfels are packed tightly onto the visible surface with very little depth spread, raising τ_{cw} admits almost no additional depth, so $\boldsymbol{D}_{\max}$ and hence the segmentation remain nearly unchanged. This is the opposite of the NeRF behavior, where a high τ_{cw} widens the valid depth range enough to admit background voxels and degrade precision: the volumetric density field carries appreciable mass between the surface and the background, whereas the surfel field does

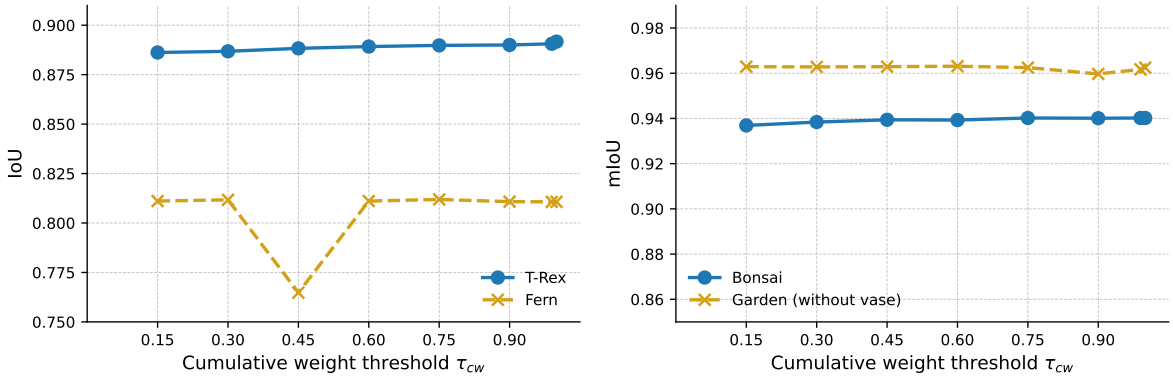


Figure 5: IoU vs. cumulative-weight target τ_{cw} for the 2DGS instantiation on representative LLFF and Mip-NeRF 360° scenes.

not. Since the 2DGS score is essentially constant over the whole range, we adopt τ_{cw}=0.75 to match the NeRF default for consistency across instantiations.

	Thin-structure		**Depth-weight**	
Scene	**w (IoU)**	**w/o (IoU)**	**w (IoU)**	**w/o (IoU)**
	NeRF instantiation			
LLFF Dataset				
Trex	**0.869**	0.673	**0.869**	0.867
Fern	0.803	**0.814**	**0.803**	0.801
Horns_left	**0.935**	0.883	**0.935**	0.929
Mip-NeRF 360° Dataset				
Bonsai	**0.920**	0.915	**0.920**	0.918
Garden (without vase)	**0.943**	0.899	0.943	**0.946**
	2DGS instantiation			
LLFF Dataset				
Trex	**0.889**	0.597	**0.889**	0.885
Fern	**0.812**	0.544	**0.812**	0.809
Mip-NeRF 360° Dataset				
Bonsai	**0.940**	0.822	**0.940**	0.938
Garden (without vase)	**0.962**	0.763	**0.962**	0.962

Table 8: Ablations on the thin-structure handling path and depth-based SAM mask weighting, reported side-by-side. Bold marks the higher IoU within each ablation pair independently. Removing the thin-structure module reduces IoU, especially on scenes with fine details (e.g., *Trex*, *Garden*), and the drop is markedly larger for the 2DGS instantiation. Removing the depth-weight results in small but consistent IoU drops, with stronger qualitative effects (faint boundaries or background bleeding).

Thin-structure path and depth-weighted mask refinement. The left half of Table 8 shows the impact of the thin-structure footprint path for both instantiations. For the NeRF instantiation, removing the module causes a catastrophic drop on *Trex* (0.869 → 0.673) and a substantial drop on *Horns_left* and *Garden*. The *Fern* scene shows a slight IoU degradation with the module (0.803 vs. 0.814), which we attribute to high depth uncertainty in the pretrained NeRF for this scene: the relaxed 2D check operating on this ambiguous geometric prior can include a few background voxels. This minor trade-off is outweighed by the recall gains on complex thin geometries.

For the 2DGS instantiation, the impact is considerably larger, IoU falls from 0.889 to 0.597 on *Trex* and from 0.812 to 0.544 on *Fern*, with substantial losses on *Bonsai* (−0.117) and *Garden* (−0.199). We attribute this difference to a structural property of the two representations. In NeRF, the volumetric density field populates voxels both at the visible surface and inside the object volume, so the thick-structure path can recover a voxel that lies just behind the center-aligned ray by finding a neighboring density-filled voxel, and the accumulated opacity along any ray is spread across many adjacent samples, so a voxel missed by the

strict center check is unlikely to be structurally isolated. In the 2DGS representation, surfels are anisotropic, randomly oriented, and located *only on the visible surface*, the object interior is hollow. A voxel grid cell that the thick-structure path fails to vote for, which maps to a cluster of surface surfels with no surrounding filled neighbors to provide a fallback, and missing that voxel means losing the entire cluster of Gaussian primitives it controls. The thin-structure footprint path, which aggregates evidence over the voxel footprint rather than just the voxel center, provides the only geometric path to recovering such isolated clusters, making it substantially more critical for the 2DGS instantiation.

Depth-weighted mask refinement. The right half of Table 8 shows that removing the depth-based mask weighting causes a negligible quantitative change but a clear qualitative degradation in both instantiations. In the NeRF instantiation, zoomed regions exhibit faint ghost artifacts and background voxels are falsely activated as foreground, because denser sampling in zoomed views increases false SAM responses on nearby background pixels. In the 2DGS instantiation, removing the band-pass depth weighting (Equation 5) causes consistent IoU drops of at most 0.004 on all four scenes. The click-depth band already restricts most background activations through the behind-surface gate in the fusion kernel, and the SAM false positives are marked when the background-projection sits between two thin foreground structures, so depth weighting acts as a complementary soft suppression that cleans up these ambiguities. Its primary observable effect is qualitative, without it, pixels at depths just outside the band that SAM ambiguously labels foreground survive into the fusion step. Depth weighting is therefore a cheap but qualitatively important refinement in both instantiations, and we keep it enabled in all experiments.

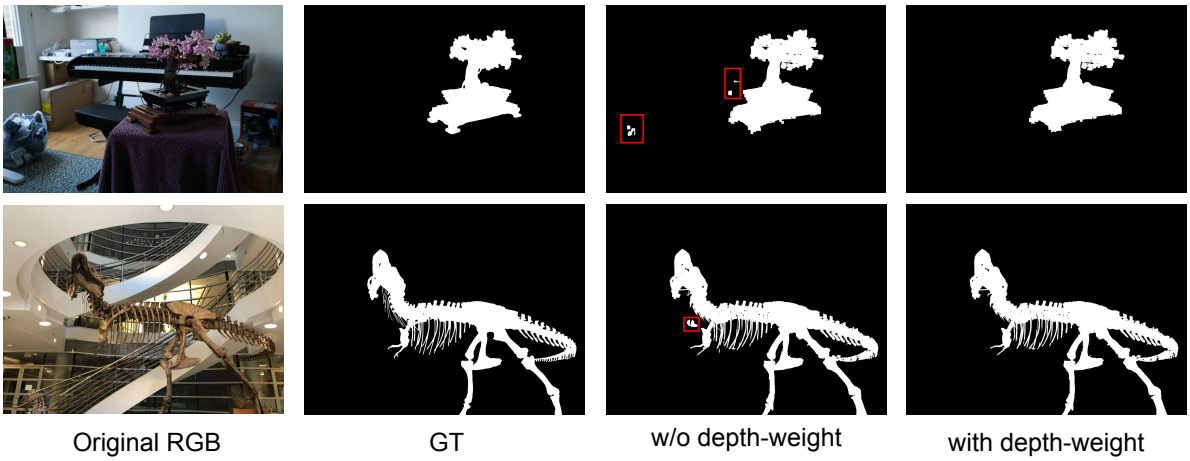


Figure 6: Qualitative ablation of depth weighting in the refined SAM mask. **Top row:** NeRF instantiation on the *bonsai* scene from Mip-NeRF 360°. **Bottom row:** 2DGS instantiation on the *Trex* scene from LLFF. Red boxes highlight isolated false positives that the depth weight removes.

Qualitative effect of depth weighting. NeRF instantiation (top row). As shown in Figure 6, without depth weighting, SAM responses on the *bonsai* scene of Mip-NeRF 360° activate background voxels that do not belong to the object. The red boxes highlight isolated false positives. These errors arise because zoomed centroid views sample dense rays near the background, and thus background pixels are projected between the thin foreground structures, such as bonsai flower petals, etc., increasing the spurious high confidence of SAM on these pixels. The NeRF depth map suppresses these activations by penalizing pixels far away in the mask. As a result, the depth-weighted mask remains clean and boundary-aligned, while the non-weighted mask exhibits ghost artifacts at the leaf-background interface.

2DGS instantiation (bottom row). The same failure mode appears on *Trex* from LLFF with the 2DGS instantiation, but the underlying mechanism differs slightly. Without the band-pass depth weight of Equation 5, SAM ambiguities at the narrow gaps between adjacent rib bones leak onto the staircase that sits directly behind the skeleton. The zoomed centroid frames place the staircase only marginally further back than the bones along the camera ray, and SAM cannot disambiguate them from texture alone. The red

boxes highlight these inter-rib bleeds in the non-weighted column. Once the user prompts are translated into a depth band $[d^{(s)}_{\min}, d^{(s)}_{\max}]$ around the visible skeleton, the band-pass weight attenuates every pixel whose median depth falls outside the band, eliminating the inter-rib bleeds while leaving the on-bone activations untouched. The qualitative trend exactly mirrors the NeRF behavior in the top row with clean boundaries and no ghost artifacts, despite the underlying weighting being mechanism-specific (AABB normalization depth-weight factor vs. band-pass weight). This consistency across instantiations supports our claim that depth weighting is a representation-agnostic refinement.

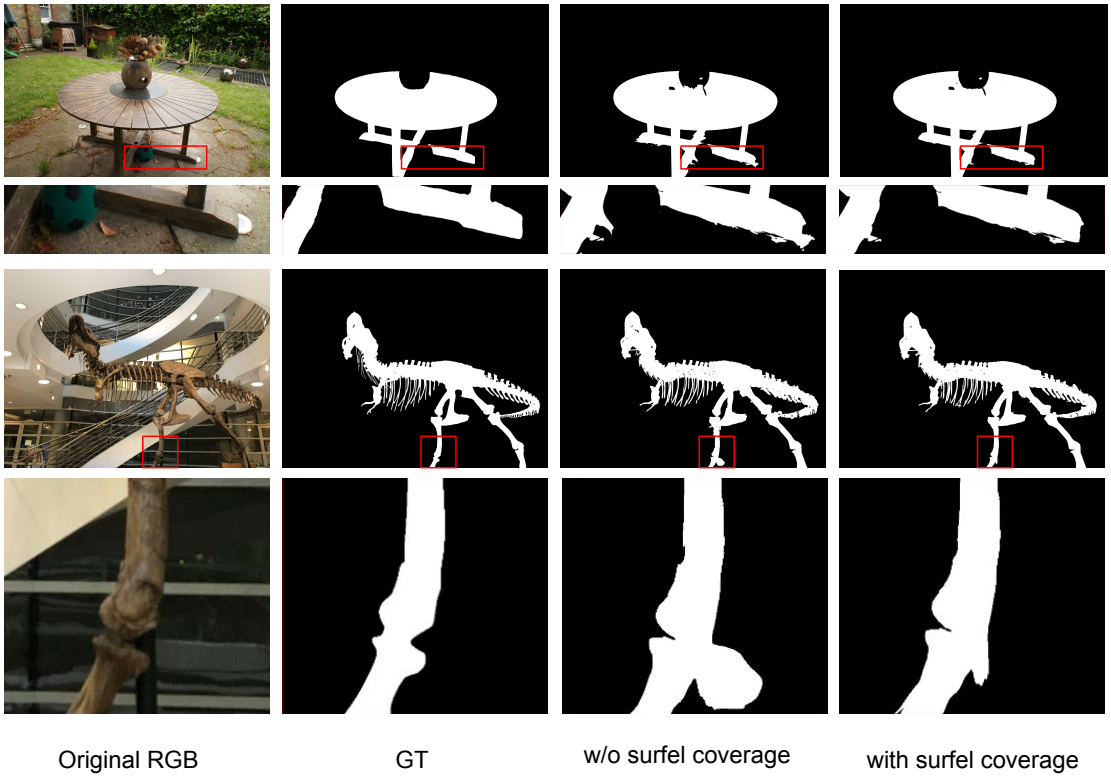


Figure 7: Qualitative effect of the surfel-footprint coverage filter (Pass 3) in 2DGS instantiation. **Rows 1-2:** *Garden* (without vase) from Mip-NeRF 360°, with row 2 showing zoomed-in crops of the highlighted regions. **Rows 3-4 :** *Trex* from LLFF, with row 4 showing the corresponding zoom-ins.

Variant	**Needle Filter**	**Coverage Mapping**	**IoU (*Trex*)**	**mIoU (*Garden*)**
Full (Pass 1+2+3)	✓	✓	**0.8891**	**0.9620**
Pass 1+2 only	✓	✗	0.8748	0.9482
Pass 1 only	✗	✗	0.8748	0.9482

Table 9: Ablation of the needle filter (Pass 2) and the surfel-footprint coverage mapping (Pass 3) of the 2DGS voxel-to-surfel mapping. Bold marks the best score per scene.

Needle filter and surfel-footprint coverage (2DGS only). Needle filter (Pass 2). Needle filtering serves as a lightweight geometric prior that removes highly anisotropic surfels prior to coverage analysis. Since the subsequent coverage stage already rejects many boundary artifacts, the quantitative gains are modest, but the filter reduces the number of candidate surfels processed during refinement. Concretely, needle-shaped surfels with high aspect ratio defined in the Equation 23 typically have low individual opacity and are visually occluded by the denser, well-rounded surfels around them, so their removal does not change

the rendered mask. They are still expensive to process in Pass 3, because their elongated footprint produces a large screen-space bounding box. Pruning them up-front is therefore a compute optimization.

Surfel-footprint coverage (Pass 3). In both scenes shown in Figure 7, zoom-ins in rows 2 and 4, the without-coverage columns show characteristic background bleeding at object boundaries, disc surfels whose centers legitimately sit inside a segmented voxel but whose 2D footprint mostly covers non-foreground pixels (table leg in *Garden*, bones joint in leg in *Trex*). Pass 3 measures, per surfel, the fraction of its in-ellipse pixels that the band-pass SAM mask labels foreground (Equation 27) and culls surfels whose best-view coverage falls below τ_κ. The result is a sharper, more topologically accurate boundary on the photo-realistic render. The quantitative gains in Table 9 (+0.014 IoU on *Trex* and +0.014 mIoU on *Garden*) align with the qualitative sharpening.

5 Conclusion

We presented DivAS, a training-free, zero-shot framework for interactive 3D segmentation that operates over both pre-trained NeRF and 2DGS reconstructions through a single shared algorithmic skeleton with representation-specific adapters. By combining geometric priors from 3D representations with zero-shot 2D SAM masks inside a shared depth-weighted CUDA voxel-fusion kernel, DivAS achieves accurate 3D object extraction without per-scene optimization or per-primitive feature training on either representation. We discuss limitations and future work in Appendix I.

A central practical consequence of the optimization-free design is that, among the object-level GS-segmentation methods we evaluate, DivAS-GS is the only one that stays competitive with state-of-the-art segmentation quality while keeping its peak VRAM within the consumer-hardware envelope at the standard $\times 4$ downsampling, where SAGA and Gaussian Grouping do not. SA3D-GS is the only baseline that maintains a comparable footprint but at a substantially lower segmentation quality. DivAS-GS attains the best LLFF IoU and the second-best Mip-NeRF 360° mIoU among the Gaussian splatting baselines, while simultaneously holding the lowest peak VRAM and a near-best end-to-end runtime, a combination that no prior GS-segmentation method achieves under the same resource constraint. The same skeleton applied to NeRF matches or exceeds the optimization-based NeRF baselines at roughly $2\times$ lower end-to-end cost. Taken together, these results suggest that a single optimization-free lifting recipe, shared across backbones through representation-specific adapters, can deliver segmentation quality competitive with state-of-the-art optimization-based methods while operating well within a consumer-hardware memory budget across both NeRF and GS backbones, and we expect this direction to support future tools for interactive, optimization-free 3D editing and scene understanding on the consumer hardware that its intended audience actually owns.

References

Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39 (12):2481–2495, 2017.

Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 5855–5864, 2021.

Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 5470–5479, 2022.

Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 9650–9660, 2021.

Jiazhong Cen, Zanwei Zhou, Jiemin Fang, Wei Shen, Lingxi Xie, Dongsheng Jiang, Xiaopeng Zhang, Qi Tian, et al. Segment anything in 3d with nerfs. *Advances in Neural Information Processing Systems*, 36:25971–25990, 2023.

Jiazhong Cen, Jiemin Fang, Zanwei Zhou, Chen Yang, Lingxi Xie, Xiaopeng Zhang, Wei Shen, and Qi Tian. Segment anything in 3d with radiance fields, 2024. URL `https://arxiv.org/abs/2304.12308`.

Jiazhong Cen, Jiemin Fang, Chen Yang, Lingxi Xie, Xiaopeng Zhang, Wei Shen, and Qi Tian. Segment any 3d gaussians. In *Proceedings of the AAAI conference on artificial intelligence*, pp. 1971–1979, 2025.

Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *European conference on computer vision*, pp. 333–350. Springer, 2022.

Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

Alexey Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Michael Fischer, Iliyan Georgiev, Thibault Groueix, Vladimir G Kim, Tobias Ritschel, and Valentin Deschaintre. Sama: Material-aware 3d selection and segmentation. In *2026 International Conference on 3D Vision (3DV)*, pp. 1812–1822. IEEE, 2026.

Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 5501–5510, 2022.

Stephan J Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. Fastnerf: High-fidelity neural rendering at 200fps. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 14346–14355, 2021.

Rahul Goel, Dhawal Sirikonda, Saurabh Saini, and PJ Narayanan. Interactive segmentation of radiance fields. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4201–4211, 2023.

Álvaro González. Measurement of areas on a sphere using fibonacci and latitude–longitude lattices. *Mathematical geosciences*, 42(1):49–64, 2010.

Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2d gaussian splatting for geometrically accurate radiance fields. In *ACM SIGGRAPH 2024 conference papers*, pp. 1–11, 2024.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, George Drettakis, et al. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4):139–1, 2023.

Justin Kerr, Chung Min Kim, Ken Goldberg, Angjoo Kanazawa, and Matthew Tancik. Lerf: Language embedded radiance fields. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 19729–19739, 2023.

Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 4015–4026, 2023.

Sosuke Kobayashi, Eiichi Matsumoto, and Vincent Sitzmann. Decomposing nerf for editing via feature field distillation. *Advances in neural information processing systems*, 35:23311–23330, 2022.

Yichen Liu, Benran Hu, Chi-Keung Tang, and Yu-Wing Tai. Sanerf-hq: Segment anything for nerf in high quality. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3216–3226, 2024.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 10012–10022, 2021.

Ben Mildenhall, Pratul P Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Transactions on Graphics (ToG)*, 38(4):1–14, 2019.

Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision*, pp. 405–421. Springer, 2020.

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. In *ACM SIGGRAPH*, 2022.

Minghan Qin, Wanhua Li, Jiawei Zhou, Haoqian Wang, and Hanspeter Pfister. Langsplat: 3d language gaussian splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 20051–20060, 2024.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pp. 8748–8763. PmLR, 2021.

Zhongzheng Ren, Aseem Agarwala, Bryan Russell, Alexander G Schwing, and Oliver Wang. Neural volumetric object selection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 6133–6142, 2022.

Robin Strudel, Ricardo Garcia, Ivan Laptev, and Cordelia Schmid. Segmenter: Transformer for semantic segmentation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 7262–7272, 2021.

Vadim Tschernezki, Iro Laina, Diane Larlus, and Andrea Vedaldi. Neural feature fusion fields: 3d distillation of self-supervised 2d image representations. In *2022 International Conference on 3D Vision (3DV)*, pp. 443–453. IEEE, 2022.

Suhani Vora, Noha Radwan, Klaus Greff, Henning Meyer, Kyle Genova, Mehdi SM Sajjadi, Etienne Pot, Andrea Tagliasacchi, and Daniel Duckworth. Nesf: Neural semantic fields for generalizable semantic segmentation of 3d scenes. *arXiv preprint arXiv:2111.13260*, 2021.

Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. *Advances in neural information processing systems*, 34:12077–12090, 2021.

Yunyang Xiong, Bala Varadarajan, Lemeng Wu, Xiaoyu Xiang, Fanyi Xiao, Chenchen Zhu, Xiaoliang Dai, Dilin Wang, Fei Sun, Forrest Iandola, et al. Efficientsam: Leveraged masked image pretraining for efficient segment anything. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16111–16121, 2024.

Mingqiao Ye, Martin Danelljan, Fisher Yu, and Lei Ke. Gaussian grouping: Segment and edit anything in 3d scenes. In *European conference on computer vision*, pp. 162–179. Springer, 2024.

Chaoning Zhang, Dongshen Han, Yu Qiao, Jung Uk Kim, Sung-Ho Bae, Seungkyu Lee, and Choong Seon Hong. Faster segment anything: Towards lightweight sam for mobile applications. *arXiv preprint arXiv:2306.14289*, 2023.

Shuaifeng Zhi, Tristan Laidlow, Stefan Leutenegger, and Andrew J Davison. In-place scene labelling and understanding with implicit scene representation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 15838–15847, 2021.

Supplementary Material for: DivAS: Interactive 3D Segmentation by Depth-Weighted Voxel Aggregation

A Implementation Details

A.1 NeRF instantiation

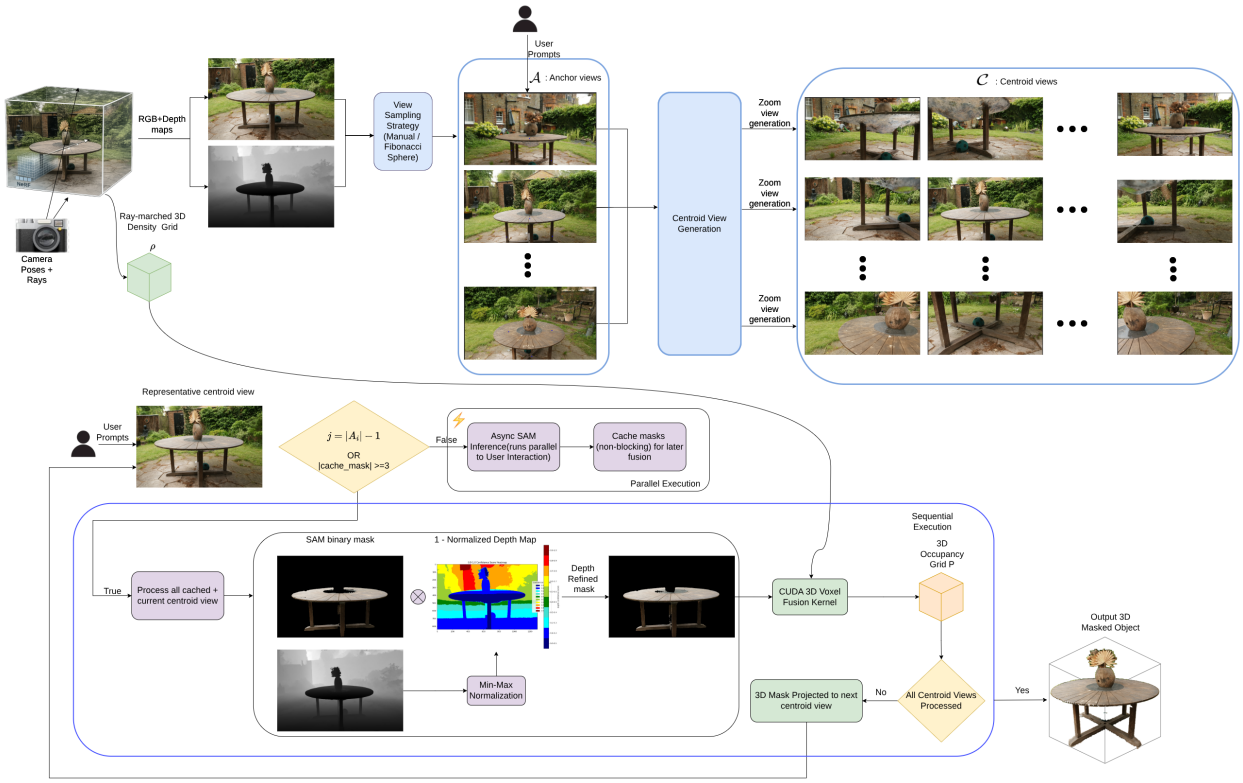


Figure 8: **NeRF instantiation of DivAS.** RGB and depth are rendered through volumetric ray marching over the hash-grid NeRF representation. The occupancy grid is obtained by ray-marching the trained density field. For each centroid view c_j in an anchor group $\mathcal{A}_i$, SAM inference runs asynchronously while the user continues interacting with subsequent views. When the cache holds at least three processed masks or the group ends, all cached and current masks are synchronized, refined using inverse-normalized NeRF depth maps, and fused by a CUDA-based 3D voxel kernel into a probabilistic occupancy grid $\mathcal{P}$. The fused 3D mask is iteratively projected back onto subsequent centroid views to guide progressive scene-wide refinement.

NeRF training setup. For LLFF, we use the original `torch-ngp` implementation with uniform sampling. For Mip-NeRF 360°, we use the hierarchical proposal sampling from SANeRF-HQ (Liu et al., 2024). Following Instant-NGP (Müller et al., 2022), the scene-bound parameter is selected per scene as the smallest value that still covers the full scene while maintaining clear depth separation between foreground and background.

SANeRF-HQ (Liu et al., 2024) uses a 16-level hash grid with 2-dimensional features for its NeRF field, and an additional 16-level grid with feature dimension 8 for its SAM feature field. Our LLFF configuration follows the Instant-NGP default, a faster 8-level hash grid with 4-dimensional features, which is well-suited to the forward-facing capture geometry of LLFF. For Mip-NeRF 360°, we match SANeRF-HQ's 16-level hash grid and 2-dimensional feature configuration to handle the wider depth range of the unbounded scenes.

These differences affect only the underlying NeRF reconstruction and do not change the DivAS segmentation pipeline.

The per-scene bound values used in our NeRF reconstructions are summarized in Table 10.

Scene	Bound (`aabb_scale`)
Fern	2
Fortress	4
Orchids	4
Horns	8
Trex	32

Table 10: Bound parameters used for NeRF training. These depend only on the underlying NeRF reconstruction and are not part of the DivAS method itself.

Voxel grids. We use a 128^3 occupancy grid for LLFF and a 256^3 grid for Mip-NeRF 360°, balancing ray-marching speed with the resolution needed to avoid boundary leakage. Two 3D grids are maintained for the NeRF instantiation, one for the NeRF density field used by the geometric consistency checks, and one for the voxel-wise segmentation probability produced by the fusion kernel. These grid resolutions were kept fixed across all NeRF experiments.

Training schedule. The underlying NeRF is trained for 25K iterations on LLFF and 20K iterations on Mip-NeRF 360°, using images downsampled by $\times 4$ throughout. The same schedule is used consistently across all reported NeRF experiments.

CUDA fusion-kernel hyperparameters. The fusion kernel uses dataset-specific values for the tolerance and density hyperparameters in the upper block of Table 11, while the remaining entries are held fixed across both benchmarks. The tuned hyperparameters are specified per reconstruction pipeline rather than per scene. As mentioned at A.1, LLFF uses uniform ray sampling, whereas Mip-NeRF 360 employs the hierarchical proposal-sampling strategy. These reconstruction pipelines produce substantially different depth distributions, density accumulation profiles, and ray-sample statistics. Consequently, parameters such as γ and λ_{range} primarily serve as normalization factors that adapt the fusion kernel to the statistical scale of the underlying reconstruction pipeline, rather than altering the segmentation logic itself. These values were selected through small sweeps reported in Section G. Once selected, a single parameter set is used for all scenes within a benchmark without further per-scene or per-object tuning. Importantly, these parameters are fixed before interactive segmentation begins and are not adjusted during user interaction. The user is therefore exposed to a single operating configuration for a given reconstruction pipeline.

A.2 NeRF-Backbone 3D Fusion Details

Occupancy from ray-marched density. For the NeRF backbone, the trained scene is represented as a continuous volumetric density field. During rendering, each pixel is obtained by ray marching through the scene and accumulating density contributions along the camera ray. Besides the rendered RGB image, the renderer provides a depth tuple $(\boldsymbol{D}_{\min}, \boldsymbol{D}_{\text{ref}}, \boldsymbol{D}_{\max}, \boldsymbol{D}_{\text{mw}}, \boldsymbol{N}_{\text{samp}})$ mentioned at Section. 3.1 for every pixel. These quantities capture the geometric structure observed along the ray and serve as the primary inputs to the 3D fusion kernel.

Voxel-to-view projection and geometric checks. Each voxel center $\boldsymbol{x}_c$ projects to normalized image coordinates $(u_x^i, v_x^i) = \pi_i(\boldsymbol{x}_c)$, out-of-frame voxels skip view i, otherwise the SAM probability m_x^i is sampled at the pixel. A voxel passing the mask threshold ($m_x^i \geq 0.5$) and the density floor ($\boldsymbol{\rho}_x \geq \rho_{\text{thresh}}$) then undergoes two geometric checks. Let $d_{\min}, d_{\text{exp}}, d_{\max}, d_{\text{mw}}, n_{\text{samp}}$ be the values from the depth buffer tuple for the pixel x_p, y_p. **Ray-voxel alignment** We enforce geometric consistency by measuring the distance between the voxel center and the image ray corresponding to the pixel (u_x^i, v_x^i). The ray origin and direction $(\boldsymbol{o}_i, \hat{\boldsymbol{d}}_i)$ are derived through $\boldsymbol{K}_i, \boldsymbol{T}_i$, which are the intrinsic and extrinsic parameters of the camera related

Parameter	LLFF	Mip-NeRF 360°	Purpose
`base_tolerance_multiplier` γ	4.0	0.5	Base scale for the depth tolerance used in geometric checks.
`per_sample_bonus` β	0.2	0.2	Incremental relaxation added per supporting depth sample.
`max_bonus` $b_{\max}$	10	1	Upper bound on the accumulated tolerance bonus.
`depth_range_factor` λ_{range}	0.05	5.0	Used to control the distance tolerance check.
`density_thresh` ρ_{thresh}	10	5	Density required for a voxel to be considered valid.
`thin_density_thresh` $\rho_{\text{thin_thresh}}$	50	30	Higher density threshold used to detect thin structures.
`thin_percent_cover` $\rho_{\text{percent_cover_thresh}}$	0.20	0.30	Fractional coverage of voxel by the masked pixels.
`grid_resolution`	128^3	256^3	Occupancy and segmentation voxel-grid resolution.
`cumulative_weight` τ_{cw}	0.75	0.75	Cumulative-weight cutoff fixing the upper depth bound $\boldsymbol{D}_{\max}$.
`thin_accept` τ_{thin}	0.6	0.6	Thin-structure acceptance threshold in compact fusion.
`consensus_threshold` τ_{cons}	0.5	0.5	Multi-view consensus threshold marking a voxel segmented.

Table 11: NeRF instantiation hyperparameters and the values used in all reported runs. The upper block holds the tolerance and density parameters that are tuned per reconstruction pipeline (LLFF versus Mip-NeRF 360°), and the lower block holds the structural and threshold constants, which are held fixed across both benchmarks apart from the voxel-grid resolution. Sensitivity scans for the tunable parameters are provided in Section G.

to the pixel. The voxel to ray depth is computed as the projection $t_{\text{proj}} = \langle \boldsymbol{x}_c - \boldsymbol{o}_i, \hat{\boldsymbol{d}}_i \rangle$, clamps it to the NeRF surface band $t_c = \text{clamp}(t_{\text{proj}}, d_{\min}, d_{\max})$ which ensures that the voxel lies within the ray segment corresponding to the rendered surface. The closest point on the segment of the ray is then $\boldsymbol{p}_c = \boldsymbol{o}_i + \hat{\boldsymbol{d}}_i t_c$ mapped through scene contraction for unbounded scenes. The geometric deviation used for consistency testing is:

$$\delta_x^i = \|\mathbf{x}_c - \mathbf{p}_c\|. \tag{29}$$

This is the value on which the distance tolerance criterion is applied for geometric consistency.

Depth-gradient factor The spatial tolerance is modulated using a depth-gradient term $g_p^i = \pi_{\text{depth_gradient}}(x_p, y_p, \boldsymbol{D}_{\min}, \boldsymbol{D}_{\max})$, relaxing it on flat surfaces and tightening it at depth discontinuities.

Distance tolerance: We define a spatial proximity threshold between a voxel and its corresponding ray as:

$$\tau_{\text{spatial}} = \Delta_x \, g_p^i + \lambda_{\text{range}}(d_{\max} - d_{\min}). \tag{30}$$

where Δ_x is the cascade-level voxel edge defined earlier and the multi-resolution grid follows (Müller et al., 2022), and λ_{range} is a scene-dependent hyperparameter. This adaptively increases the tolerance in smooth regions and decreases it at depth edges. The second term, $\lambda_{\text{range}}(d_{\max} - d_{\min})$, adds slack in proportion to the width of the rendered depth band $[d_{\min}, d_{\max}]$, which is wide for NeRF because the volumetric density fills the solid interior of the object. A voxel lying anywhere between the first surface hit and the back of the object is therefore legitimately consistent with the rendered surface, and this term admits such interior voxels. The 2DGS instantiation omits it, since its surfels lie only on the visible surface and report a precise single-surface depth with no interior band to absorb.

Depth tolerance To prevent background leakage, we impose a depth consistency check on each voxel. Let x_d denote the voxel depth in this current camera view. The depth tolerance adapts to the NeRF sampling density n_{samp}.

$$\tau_{\text{depth}} = (\gamma + b)\Delta_x, \text{ where } b = \min(\beta \cdot n_{\text{samp}}, b_{\max}) \tag{31}$$

with $\beta, b_{\max}, \gamma$ are hyperparameters. A voxel x is considered geometrically consistent if

$$\delta_x^i \leq \tau_{\text{spatial}} \;\wedge\; \|x_d - d_{\text{exp}}\| \leq \tau_{\text{depth}} \tag{32}$$

Voxels satisfying both conditions follow the *thick-structure* path for depth-weighted score aggregation, while others are handled by the thin-structure refinement step.

Depth-weighted vote aggregation. Following the volumetric-rendering property of NeRF, along each camera ray, most of the density concentrates near the surface intersection, while contributions from distant samples fade. Hence, voxels projected to the expected surface center should have a stronger influence than those at the depth boundaries. The accepted votes are weighted by a Gaussian falloff from the segment midpoint $\mu_d = \frac{1}{2}(d_{\min} + d_{\max})$ with half-width $h_d = \max(\frac{1}{2}(d_{\max} - d_{\min}), \epsilon)$ and normalized offset $r = |t_c - \mu_d|/h_d$:

$$w_{\text{depth}} = \exp(-\alpha_1 r^2), \qquad p_x = \frac{\sum_i m_x^i \, w_{\text{depth}}}{\sum_i w_{\text{depth}}}. \tag{33}$$

where α_1 is a fixed falloff constant controlling how quickly confidence decays with depth distance. Voxels near the range midpoint ($r \approx 0$) thus receive higher weights, while those farther away are smoothly down-weighted, approximating a Gaussian attenuation around the true surface. Because the weights are normalized, α_1 affects only the relative sharpness and not the scale of fusion.

Thin-structure path: motivation. The thick path of Equation 32 tests consistency at a single projected pixel at the voxel center. This center-ray assumption breaks down for thin, high-density structures (Trex ribs, wires, etc.) whose entire 3D extent projects to a handful of image pixels. In these cases, the voxel center frequently lands *between* the few mask-active pixels or at the background, so the SAM probability sampled at the center is below 0.5, and the thick gate silently rejects an otherwise valid voxel. Recovering these structures requires *footprint-based* reasoning, in which evidence is aggregated over the voxel's full 2D projected extent rather than at a single pixel, and thus we need a thin-structure path.

Three-criterion eligibility. To avoid spuriously activating background voxels, the thin path is run only when all three conditions hold for the voxel x: (i) the view is sufficiently *zoomed in* so the projected footprint is sufficiently large to be locally probed, already satisfied by the centroid view (ii) the SAM mask at the projected pixel is non-trivial, $m_x^i > 0.1$, ruling out a clear background and (iii) the NeRF-predicted density at the voxel exceeds the higher thin-structure threshold, $\rho_x \geq \rho_{\text{thin_thresh}}$, restricting evidence to geometrically plausible fine structures.

Voxel footprint construction. Given voxel center $\mathbf{x}_c$ and edge Δ_x, we form the eight corners $\mathbf{x}_j^i = \mathbf{x}_c + \frac{\Delta_x}{2}\boldsymbol{\epsilon}_j, \quad \boldsymbol{\epsilon}_j \in \{-1, +1\}^3, \quad j = 1, \ldots, 8$, Project each corner of the voxel into view i using its camera parameters and take the axis-aligned bounding box normalized coordinates converted to discrete pixel indices using the spatial resolution of the view given as $\mathbb{B}_x^i = \{(x_p, y_p) | x_{\text{start}} \leq x_p \leq x_{\text{end}}, y_{\text{start}} \leq y_p \leq y_{\text{end}}\}$. This defines the footprint of total pixel count $N_{\text{pixels}} = (x_{\text{end}} - x_{\text{start}} + 1) \times (y_{\text{end}} - y_{\text{start}} + 1)$. This is the discrete 2D region over which mask and depth evidence is aggregated for voxel x in view i.

Per-pixel depth gate and supportive-pixel counting. Let x_d be the voxel depth along the camera-forward axis in view i. For each footprint pixel $(x_p, y_p) \in \mathbb{B}_x^i$, the kernel reads the mask $m = \boldsymbol{M}_i(x_p, y_p)$. Let $d_{\min}, d_{\text{exp}}, d_{\max}, d_{\text{mw}}, n_{\text{samp}}$ be the values from the depth buffer tuple for the pixel (x_p, y_p). It then evaluates a footprint-local depth tolerance that reuses the per-sample bonus of Equation 31 but with a *doubled* base multiplier to relax the depth tolerance because of the wide depth band learned by the NeRF for the thin structures, reflecting the larger effective scale of the bounding-box test compared with the single-pixel thick gate:

$$\tau_{\text{depth}}^{\text{thin}}(x_p, y_p) = (2\gamma + b)\,\Delta_x \qquad \text{where, } b = \min\big(\beta\, n_{\text{samp}},\, b_{\max}\big) \tag{34}$$

A pixel is declared *supportive* when it is simultaneously claimed by SAM and depth-consistent with the voxel:

$$\text{support}(x_p, y_p) \;=\; \mathbb{1}\big[\, m > 0.5 \;\wedge\; \|\, x_d - d_{\text{exp}} \,\| \leq \tau_{\text{depth}}^{\text{thin}}(x_p, y_p)\big]. \tag{35}$$

Two scalars are accumulated as the kernel sweeps the bounding box. The support count and the maximum mask value seen anywhere in the footprint,

$$n_{\text{valid}} \;=\; \sum_{(x_p, y_p) \in \mathbb{B}_x^i} \text{support}(x_p, y_p), \qquad m_{\max} \;=\; \max_{(x_p, y_p) \in \mathbb{B}_x^i} \boldsymbol{M}_i(x_p, y_p). \tag{36}$$

Coverage ratio and partial-credit confidence. The coverage ratio is the supportive fraction of the footprint,

$$p_{\text{covered}} \;=\; n_{\text{valid}} \,/\, N_{\text{pixels}} \tag{37}$$

and the thin-structure confidence for voxel x in view i is assigned by a hard coverage gate that returns the peak mask activation in the footprint only when the supportive fraction meets the threshold:

$$t_x^i \;=\; \begin{cases} m_{\max}, & p_{\text{covered}} \geq \rho_{\text{cover}}, \\ 0, & \text{otherwise.} \end{cases} \tag{38}$$

Returning $m_{\max}$ rather than p_{covered} when the gate fires gives the voxel partial credit at the strongest evidence the footprint contains, while a sub-threshold footprint contributes nothing, leaving the denominator of Equation 6 unchanged.

Integration with compact fusion. The thin path enters the compact fusion of Equation 6 through the thin-view set $\mathbb{T}_v$. A view i joins $\mathbb{T}_v$ when $t_x^i \geq \tau_{\text{thin}}$, contributing t_x^i to the numerator and 1 to the denominator. Because the thick path is exclusive, the same (x, i) pair does not contribute through both paths. Thin evidence *augments* rather than dilutes the thick aggregation, recovering thin structures that the center-ray test would otherwise drop. The complete pseudocode of the 3D voxel fusion kernel is defined in Algorithm 1, which carries out the thick-structure segmentation via center-ray depth-weighted aggregation, and Algorithm 2 carries out the thin-structure recovery via footprint-based coverage analysis. Their outputs are combined per voxel by the compact-fusion rule of Equation 6.

A.3 2DGS instantiation

All timings and memory footprints are measured on a single NVIDIA A100 card. 2DGS models are trained at default hyperparameters (Huang et al., 2024). DivAS-GS adaptively determines the occupancy grid structure from the underlying Gaussian statistics of the reconstructed scene, avoiding manual voxel-resolution tuning across datasets. We therefore do not hand-select a per-dataset grid size. SAM-ViT-H (Kirillov et al., 2023) is used throughout. All scenes are trained by downsampling the images by 4. As the Gaussian Grouping authors noted, the SAM mask quality affects 3D segmentation results. We generate the SAM masks on images downsampled by 2 to meet the memory-footprint constraint, then downsample to match the training image resolution. For consistency across methods and to account for VRAM limitations, we train on $\times 4$ downsampled images of the scene. For other methods, the SAM mask is generated at the same resolution as the training images.

The DivAS-GS pipeline exposes a single set of hyperparameters that is shared across LLFF and Mip-NeRF 360° (unlike the NeRF instantiation, the 2DGS kernels are not retuned per dataset). Table 12 groups them by pipeline stage. Voxel-grid construction is fully data-driven from the per-scene Gaussian statistics and exposes only the four constants of the adaptive rule, the segmentation kernels expose the kernel-side constants used by the thick path, the thin-structure path, and the voxel-to-surfel mapping. All values listed are the defaults used to produce the numbers in Section 4.2.

B 2DGS Fusion: Implementation Details

These notes describe the execution mechanics that the main text (Section 3.5.2) summarizes, and they accompany the 2DGS pseudocode in Algorithm 3 and Algorithm 4. The symbols follow the definitions in Section 3.5.2.

B.1 Occupancy grid construction

The occupancy grid is built in four passes over the trained surfels. Surfels are first filtered against the symmetric cube by an axis-aligned bounding-box test in the re-centered frame. Each survivor is mapped to an integer voxel coordinate and to its one-dimensional Morton (Z-order) index, the standard bit-interleaving

Algorithm 1 Thick-Structure Segmentation via Depth-Weighted Voxel Aggregation

Require: voxel grid $\mathbb{S}$, refined masks $\{\widetilde{\boldsymbol{M}}_i\}_{i=1}^{S}$ with associated depth maps $(\{\boldsymbol{D}_{\min}^{(s)}\}_{s=1}^{S}, \{\boldsymbol{D}_{\exp}^{(s)}\}_{s=1}^{S}, \{\boldsymbol{D}_{\max}^{(s)}\}_{s=1}^{S}, \{\boldsymbol{N}_{\text{samp}}^{(s)}\}_{s=1}^{S})$, density grid $\boldsymbol{\rho}$, 3D occupancy grid $\mathbf{P}$, cameras $\{\boldsymbol{K}_i, \boldsymbol{T}_i\}$, hyperparameters $(\gamma, \beta, b_{\max}, \lambda_{\text{range}}, \rho_{\text{thresh}}, \rho_{\text{thin thresh}}, \alpha_1, \tau_{\text{thin}})$

for each voxel $x \in \mathbb{S}$ **in parallel do**
  retrieve voxel center $\boldsymbol{x}_c$ and edge Δ_x
  if $\boldsymbol{\rho}_x < \rho_{\text{thresh}}$ **then**
    $P^{\text{thick}}(x) \leftarrow 0$; **continue** ▷ density-based empty-voxel skip
  end if
  $W_x \leftarrow 0,\ Z_x \leftarrow 0$
  for each view $i = 1, \ldots, S$ **do**
    project voxel center: $(u_x^i, v_x^i) \leftarrow \pi_i(\boldsymbol{x}_c)$; **if** out-of-frame **continue**
    sample $m_x^i \leftarrow \widetilde{\boldsymbol{M}}_i(u_x^i, v_x^i)$ and obtain voxel depth x_d in view i
    check_thin ← **True** ▷ flag for thin-path fallback eligibility
    if $m_x^i \geq 0.5$ **then** ▷ thick path entry: center-pixel mask gate
      $d_{\min}, d_{\exp}, d_{\max}, n_{\text{samp}} \leftarrow (\boldsymbol{D}_{\min}^{(i)}, \boldsymbol{D}_{\exp}^{(i)}, \boldsymbol{D}_{\max}^{(i)}, \boldsymbol{N}_{\text{samp}}^{(i)})_{u_x^i, v_x^i}$
      compute closest point on the camera ray and the ray-voxel distance δ_x^i ▷ Equation 29
      $g_p^i \leftarrow \pi_{\text{depth_gradient}}(u_x^i, v_x^i, d_{\min}, d_{\max})$
      $\tau_{\text{spatial}} \leftarrow \Delta_x\, g_p^i + \lambda_{\text{range}}(d_{\max} - d_{\min})$ ▷ Equation 30
      $b \leftarrow \min(\beta\, n_{\text{samp}},\ b_{\max}), \quad \tau_{\text{depth}} \leftarrow (\gamma + b)\, \Delta_x$ ▷ Equation 31
      if $\delta_x^i \leq \tau_{\text{spatial}}$ **and** $\|x_d - d_{\exp}\| \leq \tau_{\text{depth}}$ **then** ▷ Equation 32
        $\mu_d \leftarrow \frac{1}{2}(d_{\min} + d_{\max}),\ h_d \leftarrow \max(\frac{1}{2}(d_{\max} - d_{\min}), \epsilon)$
        $r \leftarrow |t_c - \mu_d| / h_d$
        $w_{\text{depth}} = \exp(-\alpha_1 r^2)$ ▷ Equation 33
        $W_x \mathrel{+}= m_x^i\, w_{\text{depth}}$; $Z_x \mathrel{+}= w_{\text{depth}}$ ▷ depth-weighted center-ray vote
        check_thin ← **False** ▷ thick vote accepted, skip thin fallback
      end if
    end if
    // Thin-structure fallback: invoke Algorithm 2 when the thick path is skipped or rejected
    if check_thin **and** $m_x^i > 0.1$ **and** $\boldsymbol{\rho}_x \geq \rho_{\text{thin thresh}}$ **then**
      $t_x^i \leftarrow \textsc{ThinStructure}(x, i)$ ▷ call Algorithm 2
      if $t_x^i \geq \tau_{\text{thin}}$ **then**
        $W_x \mathrel{+}= t_x^i$; $Z_x \mathrel{+}= 1$ ▷ thin-pathway contribution
      end if
    end if
  end for
  // Multi-view consensus
  if $Z_x > \varepsilon$ **then**
    $\mathbf{P}_x \leftarrow W_x / Z_x$
  else
    $\mathbf{P}_x \leftarrow 0$
  end if
end for
return $\mathbf{P}$

Table 12: DivAS-GS hyperparameters used in all reported runs. The constants below are dataset-independent unless noted.

Stage / Symbol	Value	Role
Voxel-grid construction (Section 3.5.2)		
α_v	3.0	Voxel-size scale on the median Gaussian footprint (3σ rule)
β_v	1.0	Voxel-size scale on the scene radius/reference resolution
g_0	256	Reference grid resolution in the scene-scale term
$g_{\max}$	512 / 1024	Grid-resolution cap, LLFF / Mip-NeRF 360°
τ_{TF}	0.5	Transmittance floor defining the active-voxel list
AABB safety pad	1.05	$b = \lceil 1.05\, R_{\mathrm{fg}} \rceil$
Fusion kernel thick path (Algorithm 3)		
γ	1.0	Base depth-tolerance multiplier (in units of h_v)
β	0.05	Inverted per-sample tightening strength
$b_{\max}$	0.5	Cap on the legacy bonus term
γ_{ST}	1.5	See-through gap as a multiple of d_{cam}
τ_{thin}	0.6	Thin-structure acceptance threshold
τ_{cons}	0.5	Multi-view consensus threshold
Behind-margin	$2\, h_v$	Asymmetric far-gate offset on the surface depth
Fusion kernel thin path (Algorithm 4)		
ρ_{cover}	0.5	Coverage ratio threshold on $n_{m \wedge d}/n_{\mathrm{depth}}$
ρ_{occ}	0.1	Fractional pixel floor of voxel footprint for the same pool
n_{abs}	25	Absolute pixel-count floor on the projected footprint
3D distance gate	$1.5\, h_v$	Max unprojected-pixel distance to the voxel center
Voxel-to-surfel mapping (Algorithm 5)		
τ_a	$\mathcal{P}_{99}(\{a_i\}_{i=1}^{M})$	Per-scene needle aspect threshold
Tangent containment	1σ	Pass 3 disc-overlap criterion
τ_M	0.5	Pixel-shader mask threshold in Pass 3
τ_κ	0.5	Final boundary-bleeder coverage threshold

Algorithm 2 Thin-Structure Segmentation via Footprint-Based Coverage Analysis

Require: voxel center $\boldsymbol{x}_c$, edge Δ_x, voxel depth x_d, depth maps $(\{\boldsymbol{D}^{(s)}_{\min}\}_{s=1}^S, \{\boldsymbol{D}^{(s)}_{\exp}\}_{s=1}^S, \{\boldsymbol{D}^{(s)}_{\max}\}_{s=1}^S, \{\boldsymbol{N}^{(s)}_{\text{samp}}\}_{s=1}^S)$, refined mask $\widetilde{\boldsymbol{M}}_i$ of view i, camera $(\boldsymbol{K}_i, \boldsymbol{T}_i)$; hyperparameters $(\gamma, \beta, b_{\max}, \rho_{\text{cover}})$

Form the 8 voxel corners $\{\boldsymbol{x}^i_j = \boldsymbol{x}_c + \frac{\Delta_x}{2}\boldsymbol{\epsilon}_j\}_{j=1}^8$, $\boldsymbol{\epsilon}_j \in \{-1, +1\}^3$

$(u^x_j, v^x_j) \leftarrow \pi_i(\boldsymbol{x}^i_j)$ Project corners.

Construct the axis-aligned bounding box $\mathbb{B}^i_x = [u_{\min}, u_{\max}] \times [v_{\min}, v_{\max}]$ in pixel coordinates, clipped to the image

$N_{\text{pixels}} \leftarrow \|\mathbb{B}^i_x\|$; $\quad n_{\text{valid}} \leftarrow 0$; $\quad m_{\max} \leftarrow 0$

for each pixel $(x_p, y_p) \in \mathbb{B}^i_x$ **do**

  $m \leftarrow \widetilde{\boldsymbol{M}}_i(x_p, y_p)$

  $d_{\min}, d_{\exp}, d_{\max}, n_{\text{samp}} \leftarrow (\boldsymbol{D}^{(i)}_{\min}, \boldsymbol{D}^{(i)}_{\exp}, \boldsymbol{D}^{(i)}_{\max}, \boldsymbol{N}^{(i)}_{\text{samp}})_{x_p, y_p}$

  $b \leftarrow \min(\beta\, n_{\text{samp}},\, b_{\max})$, $\quad \tau^{\text{thin}}_{\text{depth}} \leftarrow (2\gamma + b)\,\Delta_x$ ▷ Equation 34

  if $m > 0.5$ **and** $\|x_d - d_{\exp}(x_p, y_p)\| \le \tau^{\text{thin}}_{\text{depth}}$ **then** ▷ Equation 35

    $n_{\text{valid}} \mathrel{+}= 1$

  end if

  $m_{\max} \leftarrow \max(m_{\max},\, m)$ ▷ Equation 36

end for

$p_{\text{covered}} \leftarrow n_{\text{valid}} \,/\, N_{\text{pixels}}$ ▷ Equation 37

if $p_{\text{covered}} \ge \rho_{\text{cover}}$ **then**

  return $m_{\max}$ ▷ Equation 38

else

  return 0

end if

that places spatially adjacent voxels at adjacent linear addresses so that scatter and lookup remain cache-coherent. Because many surfels typically fall in a single voxel, the density grid is then populated by a Morton-indexed scatter-add of surfel opacities, with each voxel retaining only the aggregate, not the contributing surfel identities. Finally, voxels below the transmittance floor τ_{TF} are dropped, the thresholded grid is materialized in packed form for fast occupancy lookup, and a compact active-voxel list $\{m : \boldsymbol{\rho}[m] \ge \tau_{\text{TF}}\}$ of surviving indices is assembled alongside it. The active fraction is small in practice. On *Fern* at g=512, for example, only about 0.07% of voxels survive, with roughly nine surfels sitting on each active voxel on average, so all subsequent kernels iterate this list rather than the dense grid of g^3 entries.

Computational advantages. The scene-adaptive compression of the explicit Gaussian scene, tailored for interactive 3D segmentation. Aggregating many surfels into one voxel is beneficial on several fronts that compound: it (a) reduces the working state from millions of primitives to a g^3 scalar field plus a sparse active list, cutting both kernel memory and warp-divergence costs, (b) lets the segmentation kernel iterate only over active voxels, so its launch cost scales with the foreground complexity of the scene rather than with the dense grid volume, (c) decouples downstream segmentation from per-surfel propagation, which would otherwise require expensive visibility and consensus reasoning on every primitive in every view, and (d) generalizes across scene scales without manual retuning, because every quantity used to build the grid center, extent, voxel edge, and occupancy threshold are derived from per-scene statistics. The resulting representation makes a single CUDA fusion kernel feasible at interactive rates for scenes containing millions of surfels.

Sparse active-voxel iteration. The fusion kernel assigns one CUDA thread to one entry of the active-voxel list. The thread decodes the Morton index m back to integer grid coordinates, reconstructs the world-space voxel center $\boldsymbol{x}_c$ in the scene-centered frame at edge length h_v, and skips the dense empty-voxel density test $\boldsymbol{\rho}_x < \rho_{\text{thresh}}$ because membership in the active list already implies occupancy. For each foreground view, the thread projects the center to normalized image coordinates $(u^i_x, v^i_x) = \pi_i(\boldsymbol{x}_c)$, skips out-of-frame projections, samples the refined SAM confidence $m^i_x = \widetilde{\boldsymbol{M}}_i(u^i_x, v^i_x)$ at the projected pixel, reads the per-pixel

depth tuple $(\boldsymbol{D}_{\min}, \boldsymbol{D}_{\text{med}}, \boldsymbol{D}_{\max}, \boldsymbol{D}_{\exp}, \boldsymbol{N}_{\text{samp}})_i$ at the same location, and computes the voxel planar depth x_d in that view.

B.2 Footprint statistics via integral images

Recovering the local mean and standard deviation of the splat-traversal count over a voxel's projected footprint by direct per-voxel summation would be prohibitive, since the grid can approach a billion voxels and a single footprint can span hundreds of pixels. Instead, once per view the kernel precomputes summed-area tables of the per-pixel splat count and of its square. The footprint mean and variance then follow from a four-corner inclusion-exclusion query in constant time per voxel. The same two tables serve both the thick-path tolerance tightening and the thin-path coverage gate, so the per-view preprocessing cost is amortized across all voxels that project into the view.

B.3 Thin-structure path execution

The thin path is entered only when the thick path did not vote for the current view and the voxel passes both the behind-surface gate and the thin-structure density floor. The kernel forms the eight voxel corners $\{\boldsymbol{x}_c \pm \frac{h_v}{2}\boldsymbol{\epsilon}_j\}_{j=1}^{8}$, $\boldsymbol{\epsilon}_j \in \{-1, +1\}^3$, projects them into the view, and takes their axis-aligned pixel bounding box as the footprint, reusing the same integral-image statistics $(\bar{n}_{\text{samp}}, \sigma_n)$ as the thick path. It then sweeps the footprint pixels, applying the per-pixel tightened depth gate and the layered or see-through asymmetric test, and increments the depth-consistent and the mask-and-depth-consistent counters accordingly. For each mask-positive pixel, the kernel unprojects it with the layered-aware depth, using the front-hit depth when the pixel is layered and the median depth otherwise, and updates the running minimum squared distance to the voxel center and the running maximum SAM confidence. The accumulated counters and distances are what feed the three closing conditions stated in the main text.

B.4 Voxel-to-Surfel Mapping: Implementation Details

These notes describe the execution of the three-pass voxel-to-surfel mapping summarized in Section 3.6, and they accompany the pseudocode in Algorithm 5. No new equations are introduced here; the symbols follow the definitions in Section 3.6.

Tile-sorted gather pipeline Pass 3 reuses the structure of the 2DGS rasterizer of (Huang et al., 2024), namely, preprocess, prefix sum, tile-key duplication, radix sort, tile-range identification, and a tile-parallel pixel kernel, because that tile decomposition is what makes a per-surfel gather tractable across millions of surfels. Only the differences from the rasterizer are described below, whereas the shared scaffolding is borrowed without modification.

Preprocess and view gating In the preprocess stage, each surfel surviving the needle cull is first gated against the per-voxel view-coverage bitmask $\boldsymbol{v}$ produced by the fusion kernel (Equation 15). Surfel i is admitted into a view s only when the bit for its mapped voxel x is set, which has no rasterization analogue and bounds the effective per-view surfel count to those that genuinely contributed segmentation evidence. The remaining preprocess work is identical to the rasterizer's, recovering the world-space tangent semi-axes $(\boldsymbol{t}_i^{(0)}, \boldsymbol{t}_i^{(1)})$ from $(\boldsymbol{q}, \boldsymbol{s})$ from the surfel rotation and scales, projecting the center and the four tangent tips $\boldsymbol{\mu}_i \pm \boldsymbol{t}_i^{(0,1)}$ to pixel coordinates, computing the projected-ellipse inverse determinant for the in-pixel test, and counting the tiles each surfel touches.

Depth-free tile sort Because coverage counting has no front-to-back ordering dependency, the duplicated tile keys carry only the tile id and leave the lower 32 bits zero, so the CUB radix sort runs on the tile-id bits $(32 + \lceil \log_2 n_{\text{tiles}} \rceil)$ alone, saving sort work and memory bandwidth relative to the rasterizer, which packs per-tile depth into the low bits for blending.

Coverage gather kernel The tile-parallel kernel loads each tile's surfels into shared memory in the same cooperative batches as the 2DGS (Huang et al., 2024) rasterizer and visits the tile's pixels. The only

divergence is the bounding box, which determines which pixels are visited. DivAS-GS emits a tight one-sigma box whose half-extents are the projected lengths of the longer tangent semi-axis in each image axis, with no 3× scaling, so the one-sigma to three-sigma annulus is never enumerated. On every visited pixel, the kernel applies the rasterizer's exact three-sigma ellipse cutoff in tangent-plane coordinates and, on a hit, issues two atomic increments to the per-surfel visited and mask-positive counters. After a view completes, a per-surfel reduction forms the view coverage and folds it into the running cross-view maximum, which is used by the final threshold τ_κ.

B.5 2DGS Kernel Algorithms

This subsection lists the three CUDA kernels of the 2DGS instantiation referenced in the main text: the thick-path depth-weighted fusion kernel (Algorithm 3), the thin-structure footprint-coverage fallback (Algorithm 4), and the voxel-to-surfel mapping (Algorithm 5).

C View Selection and View Generation

This section provides a detailed description of the view-generation strategies used in our framework. For LLFF scenes, which are forward-facing, a small set of anchor views is selected manually because most of the object geometry is visible from a narrow cone of viewpoints, so a few clicks suffice. For Mip-NeRF 360° scenes, rely exclusively on automatically generated *Fibonacci views*. These are obtained by sampling N uniformly distributed directions on a sphere using Fibonacci sampling and converting each direction into a camera-to-world (c2w) matrix. The synthesized views are then ranked using our geometric view-scoring strategy and the top-K are surfaced as anchors. The following subsections describe the Fibonacci sampling procedure, the geometric ranking algorithm, and the centroid-aligned refinement views used during progressive segmentation. The same LLFF-manual, Mip-NeRF-Fibonacci division is used by both the NeRF and the 2DGS instantiations.

C.1 Fibonacci Sphere Sampling

To generate uniformly distributed viewpoints for unbounded Mip-NeRF 360° scenes, we adopt the Fibonacci lattice sampling method (González, 2010). This approach distributes N points on a unit sphere with minimal angular clustering and excellent low-discrepancy properties.

Let the golden ratio be $\phi = \frac{1+\sqrt{5}}{2}$, $\phi^{-1} \approx 0.618$. The azimuth of the i-th sample is chosen using the golden-angle increment: $\theta_i = 2\pi i \phi^{-1}$, which produces an angular separation of approximately 137.5° between consecutive points. The corresponding elevation is computed with an equal-area parameterization: $\phi_i = \arccos\big(1 - \frac{2i}{N}\big)$.

Each (θ_i, ϕ_i) pair defines a direction vector on the sphere, which we convert into a camera-to-world (c2w) matrix to synthesize a novel view of the scene. All posed training images are processed once to generate these N *Fibonacci views*, which form the candidate pool for geometric ranking. In our experiments, N=12 provides sufficient angular diversity for Mip-NeRF 360°, from which we retain the top-K (K=5) ranked views for user annotation as shown in Figure 9.

2DGS instantiation. The Fibonacci procedure is representation-agnostic, it only requires camera-to-world matrices and a renderer to synthesize the candidate images, and it is used for all Mip-NeRF 360° scenes in both instantiations. For the 2DGS instantiation we reuse the same azimuth and elevation construction (Equation C.1), then render each candidate through the 2DGS rasterizer (Section 3.5.2). The 2DGS case differs in the *radius* at which each anchor is placed. Because surfels are reliably reconstructed only within the shell spanned by the training cameras, a fixed global radius can place anchors in unsupported regions and introduce floaters and holes. We therefore make the radius adaptive per Fibonacci direction (Section 3.2), keeping the camera within the valid directional shell. The pool size N=12 and the retained top-K=5 are unchanged from the NeRF case, so the bound on per-scene user effort holds across both backbones.

Algorithm 3 Thick-Structure Segmentation for 2DGS via Depth-Weighted Voxel Aggregation

Require: active-voxel list $\{m_a\}_{a=1}^{n_{\text{active}}}$; refined masks $\{\widetilde{\boldsymbol{M}}_i\}_{i=1}^{S}$ with depth maps $(\{\boldsymbol{D}_{\text{min}}^{(s)}\}_{s=1}^{S}, \{\boldsymbol{D}_{\text{med}}^{(s)}\}_{s=1}^{S}, \{\boldsymbol{D}_{\text{max}}^{(s)}\}_{s=1}^{S}, \{\boldsymbol{N}_{\text{samp}}^{(s)}\}_{s=1}^{S})$, per-view camera-target distances $\boldsymbol{d}_{\text{cam}}$, integral images $\{\mathbf{I}_n, \mathbf{I}_{n^2}\}$, density grid $\boldsymbol{\rho}$, 3D occupancy grid $\mathbf{P}$,, view camera matrices $\{\boldsymbol{C}_i\}_{i=1}^{S}$ (each $\boldsymbol{C}_i$ represent the c2w matrix), voxel view mask $\boldsymbol{v}$, hyperparameters $(\gamma, \beta, \gamma_{\text{ST}}, \rho_{\text{thin_thresh}}, \tau_{\text{thin}}, \alpha_1, \tau_{\text{cons}}, \varepsilon)$

for each $a = 1, \ldots, n_{\text{active}}$ **in parallel do**
  decode Morton index m_a to grid coordinates and reconstruct world center $\boldsymbol{x}_c$, edge h_v, density $\boldsymbol{\rho}_x$
  $W_x \leftarrow 0,\ Z_x \leftarrow 0,\ \text{vmask} \leftarrow 0$
  for each view $i = 1, \ldots, S$ **do**
    extract camera-forward axis $\hat{\boldsymbol{e}}_{\text{fwd}}^i \leftarrow \boldsymbol{C}_i[2]$ and camera origin $\boldsymbol{o}_i \leftarrow \boldsymbol{C}_i[3]$ ▷ rows of c2w matrix
    project: $(u_x^i, v_x^i) \leftarrow \pi(\boldsymbol{x}_c, \boldsymbol{C}_i)$; **if** out-of-frame **continue**
    sample $m_x^i \leftarrow \widetilde{\boldsymbol{M}}_i(u_x^i, v_x^i)$ and obtain voxel depth x_d in view i
    $d_{\text{min}}, d_{\text{med}}, d_{\text{max}}, n_{\text{samp}} \leftarrow (\boldsymbol{D}_{\text{min}}^{(i)}, \boldsymbol{D}_{\text{med}}^{(i)}, \boldsymbol{D}_{\text{max}}^{(i)}, \boldsymbol{N}_{\text{samp}}^{(i)})_{u_x^i, v_x^i}$
    Derive pixel ray direction $\hat{\boldsymbol{d}}_i$ through (u_x^i, v_x^i) from $\boldsymbol{C}_i$
    $\cos\theta \leftarrow \langle \hat{\boldsymbol{e}}_{\text{fwd}}^i, \hat{\boldsymbol{d}}_i \rangle$, $d_\bullet^{\text{ray}} \leftarrow d_\bullet / \cos\theta$ ▷ ray-space conversion, Equation 9; $\bullet \in \{\text{min}, \text{med}, \text{max}\}$
    check_thin ← **True**
    if $m_x^i \geq 0.5$ **then** ▷ thick path entry
      $(\bar{n}_{\text{samp}}, \sigma_n) \leftarrow$ four-corner query of $(\mathbf{I}_n, \mathbf{I}_{n^2})_i$ over the projected footprint
      layered $\leftarrow (n_{\text{samp}} > \bar{n}_{\text{samp}} + \sigma_n)$, seethru $\leftarrow (d_{\text{med}} > \gamma_{\text{ST}}\, d_{\text{cam}}^{(i)})$ ▷ Equation 10
      trusted $\leftarrow (d_{\text{min}} \leq d_{\text{med}})$, front $\leftarrow$ (layered $\vee$ seethru) $\wedge$ trusted
      if not trusted **then**
        skip thick path vote for this view ▷ tilted-pixel anomaly; defer to thin path
      else
        $d_{\text{ref}} \leftarrow d_{\text{min}}$ if front else d_{med}, $\hat{t} \leftarrow \text{clamp}\big(\langle \boldsymbol{x}_c - \boldsymbol{o}_i, \hat{\boldsymbol{d}}_i \rangle,\ d_{\text{min}},\ d_{\text{ref}}\big)$
        $\delta_{\text{dist}} \leftarrow \|\boldsymbol{x}_c - (\boldsymbol{o}_i + \hat{t}\,\hat{\boldsymbol{d}}_i)\|$, $\tau_{\text{spatial}} \leftarrow h_v\, \pi_{\text{dg}}(u_x^i, v_x^i)$ ▷ Equation 13
        $E \leftarrow \max(n_{\text{samp}} - \bar{n}_{\text{samp}}, 0)$, $t_{\text{depth}} \leftarrow \gamma h_v / (1 + \beta E)$ ▷ Equation 12
        valid_d ← asymmetric gate using x_d, d_{min}, d_{med}, front ▷ Equation 11
        if $\delta_{\text{dist}} \leq \tau_{\text{spatial}}$ **and** valid_d **then**
          $d_{\text{ctr}} \leftarrow \frac{1}{2}(d_{\text{min}}^{\text{ray}} + d_{\text{max}}^{\text{ray}})$, $d_{\text{hw}} \leftarrow \max\big(\frac{1}{2}(d_{\text{max}}^{\text{ray}} - d_{\text{min}}^{\text{ray}}), \varepsilon\big)$
          $w_{\text{depth}} \leftarrow \exp\big(-\alpha_1\, ((\hat{t} - d_{\text{ctr}})/d_{\text{hw}})^2\big)$ ▷ Equation 14
          $W_x \mathrel{+}= m_x^i\, w_{\text{depth}}$, $Z_x \mathrel{+}= w_{\text{depth}}$, vmask $\mathrel{|}= (1 \ll i)$, check_thin ← **False**
        end if
      end if
    end if
  // *Behind-surface gate before thin fallback*
    $d_{\text{surf}} \leftarrow d_{\text{min}}$ if (trusted $\wedge$ seethru) else d_{med}, behind_fail $\leftarrow (x_d > d_{\text{surf}} + 2h_v)$ ▷ Equation 16
    if check_thin **and** not behind_fail **and** $\boldsymbol{\rho}_x \geq \rho_{\text{thin thresh}}$ **then**
      $t_x^i \leftarrow$ ThinStructureGS(x, i) ▷ call Algorithm 4
      if $t_x^i \geq \tau_{\text{thin}}$ **then**
        $W_x \mathrel{+}= t_x^i$, $Z_x \mathrel{+}= 1$, vmask $\mathrel{|}= (1 \ll i)$
      end if
    end if
  end for
  $\mathbf{P}_x \leftarrow W_x / Z_x$ if $Z_x > \varepsilon$ else 0 ▷ Equation 6
  $\boldsymbol{v}_x \leftarrow$ vmask if $\mathbf{P}_x \geq \tau_{\text{cons}}$ else 0 ▷ Equation 15
end for
return $(\mathbf{P}, \boldsymbol{v})$

Algorithm 4 Thin-Structure Segmentation for 2DGS via Two-Counter Footprint Coverage

Require: voxel x with center $\boldsymbol{x}_c$, edge h_v, voxel depth x_d, view i with refined mask $\widetilde{\boldsymbol{M}}_i$, depth maps $(\{\boldsymbol{D}^{(s)}_{\min}\}_{s=1}^S, \{\boldsymbol{D}^{(s)}_{\text{med}}\}_{s=1}^S, \{\boldsymbol{D}^{(s)}_{\max}\}_{s=1}^S, \{\boldsymbol{N}^{(s)}_{\text{samp}}\}_{s=1}^S)$, camera $(\boldsymbol{K}_i, \boldsymbol{T}_i)$, integral images $\mathbf{I}_n, \mathbf{I}_{n^2}$, hyperparameters $(\gamma, \beta, \rho_{\text{cover}}, n_{\text{abs}}, \rho_{\text{occ}}, \varepsilon)$

Form the 8 voxel corners $\{\boldsymbol{x}^i_j = \boldsymbol{x}_c + \frac{h_v}{2}\boldsymbol{\epsilon}_j\}_{j=1}^8$, $\boldsymbol{\epsilon}_j \in \{-1, +1\}^3$

$(u^x_j, v^x_j) \leftarrow \pi_i(\boldsymbol{x}^i_j)$ Project corners.

Construct the axis-aligned bounding box $\mathbb{B}^i_x = [u_{\min}, u_{\max}] \times [v_{\min}, v_{\max}]$ in pixel coordinates, clipped to the image

area $A_{\mathbb{B}} = (u_{\max} - u_{\min}) \times (v_{\max} - v_{\min})$

$n_{\min} \leftarrow \max(n_{\text{abs}}, \lceil \rho_{\text{occ}} A_{\mathbb{B}} \rceil)$ ▷ Equation 18

$(\bar{n}_{\text{samp}}, \sigma_n) \leftarrow$ four-corner query of $(\mathbf{I}_n, \mathbf{I}_{n^2})_i$ over $\mathbb{B}^i_x$

$n_{\text{depth}} \leftarrow 0, \quad n_{m\wedge d} \leftarrow 0, \quad m_{\max} \leftarrow 0, \quad d^2_{\text{min_to_surf}} \leftarrow +\infty$

for each pixel $(x_p, y_p) \in \mathbb{B}^i_x$ **do**

  read $\widetilde{\boldsymbol{M}}_i(x_p, y_p)$

  $d_{\min}, d_{\text{med}}, d_{\max}, n_{\text{samp}} \leftarrow (\boldsymbol{D}^{(i)}_{\min}, \boldsymbol{D}^{(i)}_{\text{med}}, \boldsymbol{D}^{(i)}_{\max}, \boldsymbol{N}^{(i)}_{\text{samp}})_{x_p, y_p}$

  if $d_{\text{med}} < \varepsilon$ **then continue**

  end if

  $E \leftarrow \max(n_{\text{samp}} - \bar{n}_{\text{samp}}, 0), \quad \eta \leftarrow 1/(1 + \beta E), \quad \tau^{\text{thin}}_{\text{depth}} \leftarrow \gamma h_v \eta$

  layered $\leftarrow (n_{\text{samp}} > \bar{n}_{\text{samp}} + \sigma_n)$

  valid_d $\leftarrow$ asymmetric per-pixel gate ▷ Equation 11

  if valid_d **then**

    $n_{\text{depth}} \mathrel{+}= 1$

    if $\widetilde{\boldsymbol{M}}_i(x_p, y_p) > 0.5$ **then**

      $n_{m\wedge d} \mathrel{+}= 1, \quad m_{\max} \leftarrow \max(m_{\max}, \widetilde{\boldsymbol{M}}_i(x_p, y_p))$

      unproject (x_p, y_p) to $\boldsymbol{p}_w^{(x_p, y_p)}$ using $d_{\min}$ if layered else d_{med}

      $d^2_{\text{min_to_surf}} \leftarrow \min(d^2_{\text{min_to_surf}}, \|\boldsymbol{p}_w^{(x_p, y_p)} - \boldsymbol{x}_c\|^2)$ ▷ Equation 17

    end if

  end if

end for

if $n_{\text{depth}} < n_{\min}$ **then return** 0

end if ▷ degeneracy floor

$p_{\text{covered}} \leftarrow n_{m\wedge d}/n_{\text{depth}}$ ▷ Equation 19

if $p_{\text{covered}} \geq \rho_{\text{cover}}$ **and** $d^2_{\text{min-to-surf}} \leq (1.5\, h_v)^2$ **then**

  return $m_{\max}$ ▷ Equation 20

else

  return 0

end if

Algorithm 5 Voxel-to-Surfel Mapping for 2DGS

Require: trained surfel set $\mathbb{G} = \{g_i = (\boldsymbol{\mu}_i, (s_{i,1}, s_{i,2}), \boldsymbol{q}_i, \alpha_i)\}_{i=1}^{M}$; scene center $\boldsymbol{c}_{\text{scene}}$, AABB half-extent b, voxel edge h_v, segmentation bitfield $\mathbf{B}^{\text{seg}}$(packed from $\mathbf{P}$), per-voxel view-coverage bitmask $\boldsymbol{v}$, refined SAM masks $\{\widetilde{\boldsymbol{M}}_s\}_{s=1}^{S}$ over the S retained views, thresholds (τ_a, τ_M, τ_κ)

// Pass 1: Bitfield lookup (permissive starting set)

for $i \in \{i : \neg\text{NaN}(\boldsymbol{\mu}_i) \wedge \|\boldsymbol{\mu}_i - \boldsymbol{c}_{\text{scene}}\|_\infty < b\}$ **in parallel do**

  $m_i \leftarrow \texttt{morton3D}\big(\lfloor(\boldsymbol{\mu}_i - \boldsymbol{c}_{\text{scene}} + b)/h_v\rfloor\big)$ ▷ Voxel index of g_i

  $\boldsymbol{f}_i^{(1)} \leftarrow \mathbf{B}^{\text{seg}}_{m_i}$ ▷ Equation 21

end for

// Pass 2: Needle-floater culling

for each i **in parallel do**

  $a_i \leftarrow \max(s_{i,1}, s_{i,2}) \,/\, (\min(s_{i,1}, s_{i,2}) + \epsilon)$ ▷ aspect ratio

  $\boldsymbol{f}_i^{(2)} \leftarrow \boldsymbol{f}_i^{(1)} \wedge (a_i \le \tau_a)$ ▷ Equation 24

end for

// Pass 3 Footprint coverage analysis (no rendering, no alpha, no depth ordering, no color)

initialize $\kappa_i \leftarrow 0$ for all i

for each view $s = 1, \ldots, S$ **do**

(i) Voxel view mask pruning. *Process surfel g_i in view s only when its associated voxel was marked for that view.*

  $\boldsymbol{a}_s \leftarrow \{\, i \,:\, \boldsymbol{f}_i^{(2)}{=}1 \;\wedge\; s \in \boldsymbol{v}_{m_i} \,\}$ ▷ semantic view restriction

(ii) Surfel projection and 1σ bounding-box construction.

  for each $i \in \boldsymbol{a}_s$ **in parallel do**

    Recover the two world-space tangent semi-axes $(\boldsymbol{t}_i^{(0)}, \boldsymbol{t}_i^{(1)})$ from $(\boldsymbol{q}_i, s_{i,1}, s_{i,2})$

    Project the center $\boldsymbol{\mu}_i$ and the four tangent tips $\boldsymbol{\mu}_i \pm \boldsymbol{t}_i^{(0,1)}$ into view s to obtain the screen-space surfel centre $(c_{i,x}, c_{i,y})$ and the projected tangent vectors $(\boldsymbol{\alpha}_i^{(0)}, \boldsymbol{\alpha}_i^{(1)})$

    1-σ bbox $\mathbb{B}_{i,s}^{1\sigma} \leftarrow [c_{i,x} \pm e_x] \times [c_{i,y} \pm e_y]$, $e_x = \max(|\alpha_{i,x}^{(0)}|, |\alpha_{i,x}^{(1)}|)$, e_y analogous ▷ tighter than the rasterizer's 3σ bbox, excludes the $1\sigma \to 3\sigma$

  end for

(iii)Tile assignment and tile-key sort.

  Enumerate, for $i \in \boldsymbol{a}_s$, the tiles overlapped by $\mathbb{B}_{i,s}^{1\sigma}$, emitting the set of surfel-tile pairs $\{(t, i)\}$

  Sort the pairs by tile id ▷ Equation 25

  Identify, for each tile t, the contiguous range of surfels assigned to it

(iv) Tile-parallel footprint gather, 1σ bbox traversal + 3σ Cramer acceptance.

  Initialize per-surfel counters $\text{total_fp}_i^{(s)} \leftarrow 0$, $\text{mask_fp}_i^{(s)} \leftarrow 0$ for $i \in \boldsymbol{a}_s$

  for each tile t **in parallel do**

    for each pixel (x_p, y_p) inside tile t **do**

      for each surfel i assigned to tile t with $(x_p, y_p) \in \mathbb{B}_{i,s}^{1\sigma}$ **do** ▷ Step A: 1σ bbox traversal

        Evaluate the standard 3σ ellipse test, if cleared $\text{total_fp}_i^{(s)}$ += 1,

        if $\widetilde{\boldsymbol{M}}_s(x_p, y_p) \ge \tau_M$ **then** $\text{mask_fp}_i^{(s)}$ += 1 ▷ If clears the ellipse test ▷ Equation 26

        end if

      end for

    end for

  end for

(v) Per-view coverage and max-over-views aggregation.

  for each $i \in \boldsymbol{a}_s$ **in parallel do**

    $\kappa_i^{(s)} \leftarrow \text{mask_fp}_i^{(s)} \,/\, \max(\text{total_fp}_i^{(s)}, 1)$, $\kappa_i \leftarrow \max\big(\kappa_i, \kappa_i^{(s)}\big)$ ▷ Equation 27, one clean view suffices averaging would dilute partial occlusions

  end for

end for

// Final indicator: threshold on max-view coverage

for each i **in parallel do**

  $\boldsymbol{f}_i \leftarrow \boldsymbol{f}_i^{(2)} \wedge (\kappa_i \ge \tau_\kappa)$ ▷ Equation 28 $\boldsymbol{f}_i \equiv \boldsymbol{f}_i^{(3)}$

end for

return $\boldsymbol{f}$

Figure 9: Fibonacci-based sampled views.

Figure 10: Top-5 views via geometric view ranking.

C.2 Geometric View Ranking Algorithm

To identify the most informative viewpoints, we rank the Fibonacci-sampled views using a composite geometric score. This scoring mechanism prioritizes views that maximize spatial diversity and align with canonical scene axes. We compute three components for each candidate view v_i to determine its final rank.

Diversity Score (D_i). We calculate the mean pairwise cosine distance between the viewing direction of view v_i and all other candidates. This penalizes clustering, ensuring the selected views are distributed uniformly around the scene.

Cardinal Coverage (C_i). We favor views that align with standard canonical perspectives (front, back, left, right, top, bottom). We compute the inverse distance between the view vector and the nearest global coordinate axis.

Pitch Extrema (P_i). We normalize the absolute pitch angle of the camera to reward high-latitude viewpoints, helping reveal geometry often occluded in purely horizontal trajectories (e.g., object tops).

The final score for view i is a weighted sum:

$$I(i) = 0.4\, D_i + 0.3\, C_i + 0.3\, P_i, \tag{39}$$

, where D_i, C_i, and P_i denote normalized diversity, coverage, and pitch-extremity scores, respectively.

The ranking procedure is summarized in Algorithm 6, and an example result is shown in Figure 10.

Algorithm 6 Geometric View Ranking

Require: Candidate trajectory $\mathcal{T} = \{v_i\}_{i=1}^{N}$, weights w_d, w_c, w_p
  Define canonical axes $A = \{\pm\mathbf{x}, \pm\mathbf{y}, \pm\mathbf{z}\}$
  for each view $v_i \in \mathcal{T}$ **do**
    Diversity: Mean cosine distance to other views
    $D_i = \frac{1}{N-1}\sum_{j\neq i}(1 - \mathbf{d}_i \cdot \mathbf{d}_j)$
    Cardinality: Inverse distance to nearest axis
    $C_i = \max_{a\in A}(1/(\|\mathbf{d}_i - \mathbf{a}\| + \epsilon))$
    Pitch: Normalized absolute pitch
    $P_i = |\theta_i|/90°$
  end for
  Normalize scores D, C, P to $[0, 1]$
  Compute final score: $I_i = w_d D_i + w_c C_i + w_p P_i$
  return Top-K views with highest I_i

2DGS instantiation. The ranking score depends only on the viewing direction $\mathbf{d}_i$ and the elevation angle, both of which are read directly from the c2w matrix of each candidate camera. The score is therefore independent of the underlying representation. We apply Algorithm 6 unchanged to the 2DGS Fibonacci pool and surface the same top-K=5 as anchors.

C.3 Centroid View Generation

For each selected anchor view shown in Figure 11, the user provides a point prompt on the object. We generate a localized *centroid view* by back-projecting this 2D point into 3D space and advancing the camera along the view vector. This operation effectively zooms in on the target region, reducing background clutter and increasing the sampling resolution for the subsequent segmentation step as shown in Algorithm 7.

NeRF instantiation The 3D look-at point is obtained by back-projecting the 2D prompt using the $\boldsymbol{D}_{\text{mw}}$ depth along the ray, rather than the expected-depth from volume rendering integral of Liu et al. (2024). This yields a sharper estimate of the true surface crossing and is computed in $O(1)$ inside the existing ray-marching loop. The camera is then advanced toward the look-at point by a fixed zoom fraction (typically 0.67 for LLFF and 0.47 for Mip-NeRF 360°) though they are interactively adjustable. Because the trained NeRF is bounded by its AABB, every ray is automatically clipped to a finite, artifact-free depth range, so a fixed push never drives the camera into an undefined region.

Algorithm 7 Centroid View Generation (NeRF instantiation)

Require: Anchor view camera C_{anchor}, 2D point prompt $u \in \mathbb{R}^2$, zoom fraction $\lambda \in (0, 1)$
1: **Unprojection:**
    Retrieve depth $d = \text{DepthMap}(u)$
    Compute 3D point $x_{\text{world}} = \text{BackProject}(u, d, C_{\text{anchor}})$
2: **Camera Update:**
    Get current position $p = \text{Pos}(C_{\text{anchor}})$
    New position $p' = p + \lambda(x_{\text{world}} - p)$
    New look-at direction $f' = \text{normalize}(x_{\text{world}} - p')$
3: Form new view matrix C_{new} looking at x_{world} from p'
4: **return** C_{new}

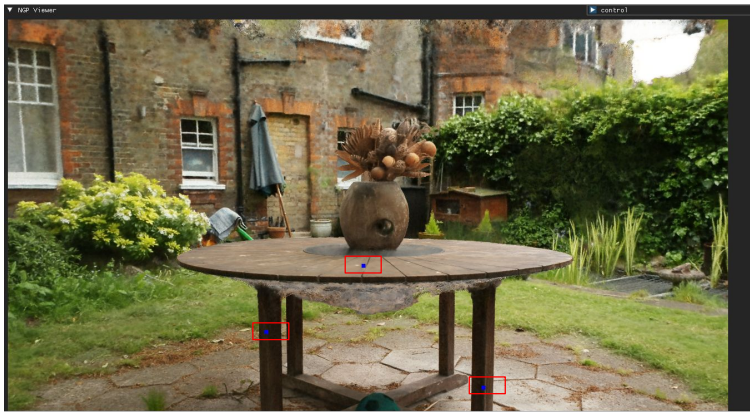


Figure 11: Anchor view with user prompts.

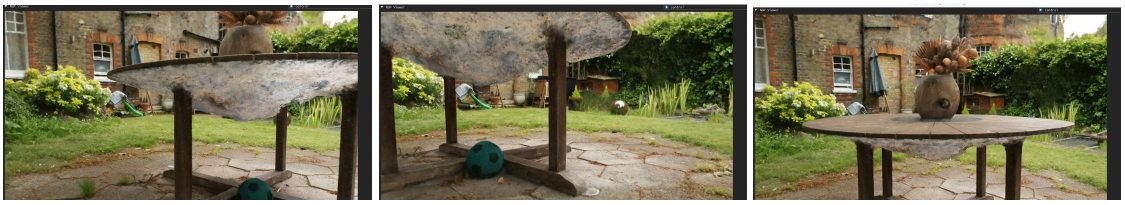


Figure 12: Centroid zoom views based on user prompts on the anchor view in Figure 11.

These centroid views, shown in Figure 12, serve as the input to SAM. The resulting high-resolution 2D masks are then fused via the voxel-aggregation kernel described in Section A.2.

2DGS instantiation: physical-radius neighborhood and occlusion-aware refinement. The centroid procedure is unchanged in spirit; the representation-specific element is the unprojection depth d at the prompt pixel u, read from the 2DGS rasterizer's median-depth output ($\boldsymbol{D}_{\text{med}}$) rather than the NeRF expected-depth channel. Because 2DGS scenes are unbounded, the fixed-fraction push of Algorithm 7 does not transfer; each 2DGS centroid view is instead generated in two strategies (Algorithm 8): a neighborhood-driven radius estimate, followed by an occlusion-aware radius refinement.

Strategy 1: Local Gaussian neighborhood and initial radius. Let $z = \boldsymbol{D}_{\text{med}}(\boldsymbol{u})$ be the median depth at the clicked pixel $\boldsymbol{u}$ and $vx_w = \texttt{BackProject}(\boldsymbol{u}, z)$ the unprojected 3D point. We first define a *physical search radius* as the world-space extent that projects to a fixed patch fraction p_f of the image width,

$$R_{\text{phys}} = p_f\, z\, \tan(\text{FOV}_x/2), \text{where}, \quad p_f = 0.25, \tag{40}$$

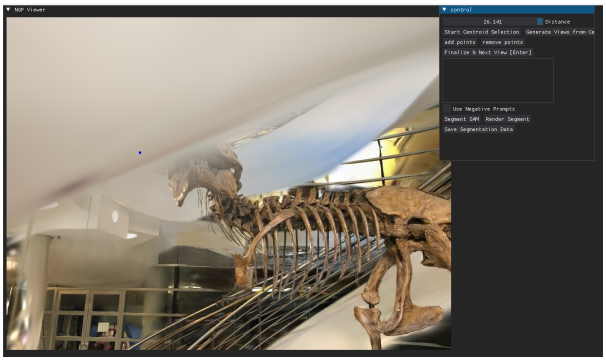
and gather the local neighborhood of opaque surfels inside this sphere whose opacity is represented by α,

$$\mathcal{N} = \big\{ i : \|\mu_i - \boldsymbol{x}_w\| < R_{\text{phys}} \ \wedge \ \alpha_i > \alpha_{\min} \big\}, \text{where, } \mu_i \text{ is surfel center} \in \mathcal{G},\ \alpha_{\min} = 0.1. \tag{41}$$

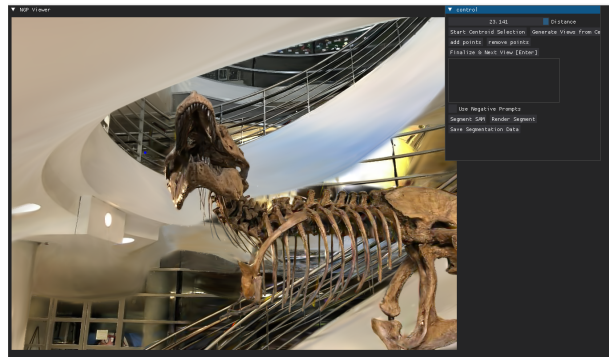
If $|\mathcal{N}| < 5$ the prompt landed in a sparsely populated region (thin structure, far background, or partial occlusion). We set the centroid radius $r_{\text{init}} = z\, p_f$ and let Strategy 2 refine it. For dense neighborhoods, we estimate the lateral spread *perpendicular to the view direction* so the radius is invariant to depth-aligned elongation. With unit view direction $\hat{\mathbf{v}}$ and $\mathbf{r}_i = \mu_i - \mathbf{x}_w$, the in-plane offset and its robust extent are

$$\mathbf{r}_i^{\perp} = \mathbf{r}_i - (\mathbf{r}_i \cdot \hat{\mathbf{v}})\hat{\mathbf{v}}, \qquad R_{\text{lat}} = \mathcal{P}_{90}\big(\{\|\mathbf{r}_i^{\perp}\|\}_{i \in \mathcal{N}}\big), \tag{42}$$

where the 90th percentile rejects stray boundary surfels. We then place the orbital camera so that the neighborhood of lateral half-width R_{lat} subtends a target fraction f_{fill} of the frame. We set the centroid



(a) Raw depth-driven radius (r_{init}=26.14): occluders intrude on the line of sight.



(b) Occlusion-aware radius (r_k=23.14): unobstructed view of the target.

Figure 13: Effect of the centroid-view refinement (Algorithm 8). The dynamic occlusion test pulls the orbital radius in until the target is no longer dominated by foreground occluders.

radius:

$$r_{\text{init}} = \frac{R_{\text{lat}}}{f_{\text{fill}}\,\tan(\text{FOV}_x/2)}, \qquad f_{\text{fill}} = 0.5, \tag{43}$$

so that larger clusters push the camera back and tighter clusters draw it in.

Strategy 1 also returns a 2D visualization box on the *anchor* view, so the user can immediately verify that the click captured the intended object extent without waiting for the centroid render.

Strategy 2: Occlusion-aware radius refinement. A persistent failure mode is that r_{init} can place the centroid camera *inside* a foreground floater, so the rendered view is dominated by occluders rather than the target as shown in Figure 13. We probe a sequence of K_r decreasing radii $\{r_k\}$ stepping linearly from r_{init} down to $0.3\,r_{\text{init}}$, rasterize each candidate at low resolution so the render costs $\sim$ 10 ms, and accept the *largest* radius whose central depth patch is not dominated by occluders. The key design choice is a *dynamic* occlusion threshold tied to the actual target geometry rather than to the camera radius: projecting the cached neighborhood $\{\mu_i\}_{i\in\mathcal{N}}$ into the probed camera and taking the nearest positive camera-Z value z_{near},

$$\tau_{\text{occ}}(r_k) = 0.85\,z_{\text{near}}, \tag{44}$$

(falling back to $0.7\,r_k$ only when the projection is empty), the 0.85 factor leaves a 15% safety band so grazing target surfels are not misread as occluders. Over a central patch $\mathcal{P}$ of fractional size f_{fill}, the occluded fraction is

$$\mathcal{O}(r_k) = \frac{|\{(x,y)\in\mathcal{P} : D_{\text{med}}(x,y) < \tau_{\text{occ}}(r_k)\}|}{|\mathcal{P}|}, \tag{45}$$

and we accept the first r_k (in decreasing order) with $\mathcal{O}(r_k) < \tau_{\mathcal{O}} = 0.15$, falling back to the most zoomed-in radius if none qualifies. Scanning from large to small deliberately yields the most *generous* unobstructed framing, preserving the largest plausible context around the click. Figure 13b shows the effect of the refined radius, which pulls the camera to a vantage that looks past the occluders onto the target.

D Additional Mathematical Formulation

This section provides the analytical form of the depth-gradient factor π_{depth}, the only component of the fusion kernel that was deferred from the main paper.

D.1 Depth gradient factor π_{depth}

The depth-gradient factor adapts the distance tolerance at object boundaries. It relaxes the tolerance in flat areas and tightens the tolerance at boundary regions where there is a sharp depth discontinuity.

Algorithm 8 2DGS Centroid-View Generation (per click)

Require: anchor camera $\boldsymbol{C}_{\text{anchor}}$, user prompt $\boldsymbol{u}$, Gaussians $\mathcal{G}$, $\texttt{render}(w2c, w', h')$, p_f=0.25, f_{fill}=0.5, $\alpha_{\min}$=0.1, probes K_r=5, τ_O=0.15

$z \leftarrow \boldsymbol{D}_{\text{med}}(\boldsymbol{u})$; $\quad \boldsymbol{x}_w \leftarrow \texttt{BackProject}(\boldsymbol{u}, z, \boldsymbol{C}_{\text{anchor}})$
// Strategy 1: physical neighborhood
$R_{\text{phys}} \leftarrow p_f\, z\, \tan(\text{FOV}_x/2)$
$\mathcal{N} \leftarrow \{i : \|\mu_i - \boldsymbol{x}_w\| < R_{\text{phys}} \wedge \alpha_i > \alpha_{\min}\}$ ▷ Equation 41
if $|\mathcal{N}| < 5$ **then** ▷ sparse fallback
  $r_{\text{init}} \leftarrow z\, p_f$; $\quad B_{\text{anchor}} \leftarrow \varnothing$
else
  $\mathbf{r}_i^{\perp} \leftarrow \mathbf{r}_i - (\mathbf{r}_i \cdot \hat{\mathbf{v}})\hat{\mathbf{v}}$; $\quad R_{\text{lat}} \leftarrow \mathcal{P}_{90}(\{\|\mathbf{r}_i^{\perp}\|\})$
  $r_{\text{init}} \leftarrow R_{\text{lat}}/(f_{\text{fill}} \tan(\text{FOV}_x/2))$ ▷ Equation 43
  $\mathbb{B}_{\text{anchor}} \leftarrow$ projected bounding box of $\{\mu_i\}_{i \in \mathcal{N}}$ on anchor view
end if
// Strategy 2: occlusion-aware refinement
$(w', h') \leftarrow (\max(\sigma w, 64), \max(\sigma h, 64))$ ▷ Low resolution image rendering
$\boldsymbol{P} \leftarrow$ central patch of size f_{fill}
$\{r_k\} \leftarrow \text{linspace}(r_{\text{init}}, 0.3\, r_{\text{init}}, K_r)$; $\quad r_\star \leftarrow r_{K_r - 1}$
for $k = 0, \ldots, K_r - 1$ **do** ▷ largest→smallest
  $w2c_k \leftarrow \texttt{BuildPose}(\boldsymbol{C}_{\text{anchor}}, \boldsymbol{x}_w, r_k)$
  $\boldsymbol{D} \leftarrow \texttt{render}(w2c_k, \sigma w, \sigma h)[\,\cdot, \cdot, \boldsymbol{D}_{\text{med}}]$
  $\mathbb{D}_{\boldsymbol{P}} \leftarrow \{\boldsymbol{D}_{\text{med}}(x, y) \in \boldsymbol{P} \mid \boldsymbol{D}_{\text{med}}(x, y) > 0.01\}$
  if $|\mathbb{D}_{\boldsymbol{P}}| = 0$ **then**
    continue
  end if
  $\mathbf{p}_i^{cam} \leftarrow \boldsymbol{R}_{w2c}\mu_i + \boldsymbol{t}_{w2c}, \quad \forall i \in \mathcal{N}$
  $\mathbb{Z} \leftarrow \{(\boldsymbol{p}_i^{cam})_z \mid (\boldsymbol{p}_i^{cam})_z > 0\}$
  if $|\mathbb{Z}| > 0$ **then**
    $z_{\text{near}} \leftarrow \min(\mathbb{Z})$
    $\tau_{\text{occ}} \leftarrow 0.85\, z_{\text{near}}$ ▷ Equation. 44
  else
    $\tau_{\text{occ}} \leftarrow 0.7\, r_k$
  end if
  $\mathbb{O}(r_k) \leftarrow$ occluded fraction over central patch
  $\mathbb{O}(r_k) \leftarrow \dfrac{|\{d \in \mathbb{D}_{\boldsymbol{P}} : d < \tau_{\text{occ}}\}|}{|\mathbb{D}_{\boldsymbol{P}}|}$ ▷ Equation. 45
  if $\mathbb{O}(r_k) < \tau_O$ **then**
    $r_\star \leftarrow r_k$; **break** ▷ largest unoccluded radius
  end if
end for
return centroid camera $C_\star = \texttt{BuildPose}(\boldsymbol{C}_{\text{anchor}}, \boldsymbol{x}_w, r_\star)$, $\mathbb{B}_{\text{anchor}}$

For each pixel (x_p, y_p), we estimate a local depth gradient using central differences on the refined depth range $(\boldsymbol{D}_{\min}(x_p, y_p), \boldsymbol{D}_{\max}(x_p, y_p))$. Let, $d(x, y) = \frac{\boldsymbol{D}_{\min}(x_p, y_p) + \boldsymbol{D}_{\max}(x_p, y_p)}{2}$ be the center depth. The horizontal and vertical finite differences are:

$$g_x = d(x_p + 1, y_p) - d(x_p - 1, y_p),$$

$$g_y = d(x_p, y_p + 1) - d(x_p, y_p - 1).$$

We compute the squared gradient magnitude:

$$G = g_x^2 + g_y^2,$$

Table 13: Resolution and threshold sensitivity across methods. We report *per-dataset mean* mIoU/IoU for each method. "Full Resolution" renders the mask *directly* at native GT resolution from the scene representation. Per dataset column, the best value is **bold** and the second-best is underlined.

Method	Configuration	Mip-NeRF 360°	LLFF
DivAS-GS (ours)	Full Resolution, τ=0.1 *(primary)*	0.9476	0.9230
DivAS-GS (ours)	Training Resolution, τ=0.1	0.9448	**0.9238**
SAGA	Full Resolution, τ=0.5 *(primary)*	0.9533	0.8739
SAGA	Training Resolution, τ=0.5	**0.9594**	0.9151
SAGA	Full Resolution, τ=0.1	0.9377	0.8925
SA3D-GS	Full Resolution, τ=0.1 *(primary)*	0.6512	0.7502
SA3D-GS	Training Resolution, τ=0.1	0.6489	0.7481

and define the depth-gradient factor as an exponential decay: $g_{\text{depth}}(x, y) = \exp\big(-k\, G\big), \qquad k > 0$. Flat regions yield $G \approx 0$ and $g_{\text{depth}} \approx 1$, while sharp discontinuities produce a large G and a small decay factor.

2DGS instantiation. The 2DGS fusion kernel reuses the same pixel-wise depth-gradient factor, π_{depth}, as the NeRF kernel. The depth source changes, central differences are taken on the $(\boldsymbol{D}_{\min}, \boldsymbol{D}_{\max})$ buffers produced by the 2DGS rasterizer rather than by NeRF ray marching.

E Additional GS Experiments

We provide the render-resolution and threshold-sensitivity analysis for the Gaussian-Splatting methods, which the main paper (Section 4.2) summarizes.

Background. "Resolution" here refers to the *image render resolution* at which the 2D mask is rasterized. DivAS-GS derives its occupancy structure adaptively from the scene's Gaussian statistics mentioned at 3.5.2 and exposes no manually tuned voxel resolution. SAGA and SA3D-GS form a mask by assigning each Gaussian a scalar affinity to the query feature (SAGA) or an optimized mask score (SA3D-GS), respectively, and keeping the primitives whose values exceed a threshold τ. The foreground is therefore a hard, *per-primitive* decision in which a single τ must both reject background primitives and retain thin foreground primitives. Argmax-based methods (Gaussian Grouping, SANeRF-HQ) instead assign a label to every primitive or pixel and are not subject to this trade-off. We sweep render resolution and τ to characterize the threshold-based methods. DivAS-GS, which never thresholds a per-primitive scalar, is reported at its single canonical operating point.

DivAS-GS is resolution-stable at a single threshold. Table 13 shows that DivAS-GS IoU changes by at most 0.30 points between full and training resolution, with the sign inconsistent across benchmarks (full resolution higher on Mip-NeRF 360° by 0.30 pt while training resolution higher on LLFF by 0.08 pt). This near-invariance holds under a fixed τ=0.1 without any resolution-specific tuning, a property we attribute to the architectural decoupling described below. For SAGA, τ must serve two competing roles: (i) rejecting background and (ii) retaining thin foreground, whose covering primitives carry affinities that fall between the two clusters, i.e., foreground/background. On LLFF at τ=0.5, the training-resolution score is high (0.9151), but the full-resolution score is 0.8739; lowering τ to 0.1 improves full resolution only partially (0.8925) by re-admitting culled boundary gaussians, so no single τ is at once complete and clean. The resulting gaps(holes) in thin geometry are not a 2D sampling effect. They are visible at both full and training resolution (Section 4.3), and reducing τ recovers them only partially while degrading other regions. Consequently, the configuration that maximizes SAGA's IoU (training resolution, τ=0.5) does not correspond to a structurally clean mask. The score and its mask topology disagree.

DivAS-GS decouples geometric separation from binarization. DivAS-GS does not threshold a per-primitive scalar. Foreground membership is decided *upstream* by multi-view voxel-occupancy consensus.

While mapping back the voxel to surfel, each surfel multi-view occupancy consensus that falls below the threshold is removed from the foreground set before any mask threshold is applied. The mask threshold τ=0.1 then operates exclusively as a binarizer on an already-clean surfel set, free to remain low without re-admitting background. Because membership rests on geometric and cross-view agreement rather than on a single primitive's appearance, thin structures that are occupied and consistently observed are retained at both resolutions. This is why DivAS-GS achieves 0.9230 LLFF IoU at full resolution, surpassing SAGA's best result across all resolution threshold settings (0.9151). Raising τ from 0.1 to 0.5 on DivAS-GS only over-culls thin-structure surfels (reproducing holes), confirming that 0.1 is the natural operating point for binarization and that background rejection has already been handled. Unlike SAGA, SA3D-GS shows essentially no resolution sensitivity (Δ<0.3 pt across all twelve scenes), confirming that the floater artifacts and missing object parts induced by the unbounded mask-score objective are equally visible at both resolutions, leaving a consistent quality floor of 0.65 on Mip-NeRF 360° and 0.75 on LLFF regardless of how the mask is measured.

F NeRF Instantiation: Qualitative Results

We compare the DivAS (NeRF) instantiation against SA3D and SANeRF-HQ on representative LLFF and Mip-NeRF 360° scenes.

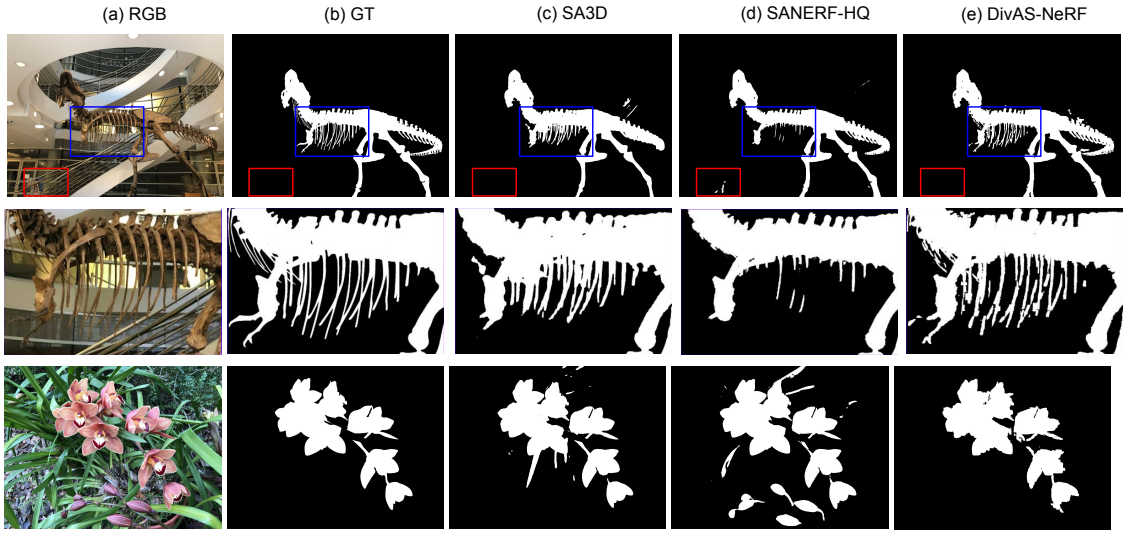


Figure 14: Results of different methods for the NeRF instantiation on LLFF.

As shown in Figure 14, we visualize segmentation results for the *Trex* object across different methods in the NeRF backbone. SA3D captures fine structures like ribs but misses global parts such as the tail, and slightly leaks into the background (row 2). SANeRF-HQ recovers the complete shape but fails on thin features like the ribs and hands. It also exhibits over-segmentation by incorrectly marking background regions with similar appearance, highlighted in the red box (row 1). This behavior aligns with its tendency toward false positives observed in the *Orchids* scene. Our method preserves both global structure and delicate parts, such as ribs and hands, while avoiding false activations and over-segmentation.

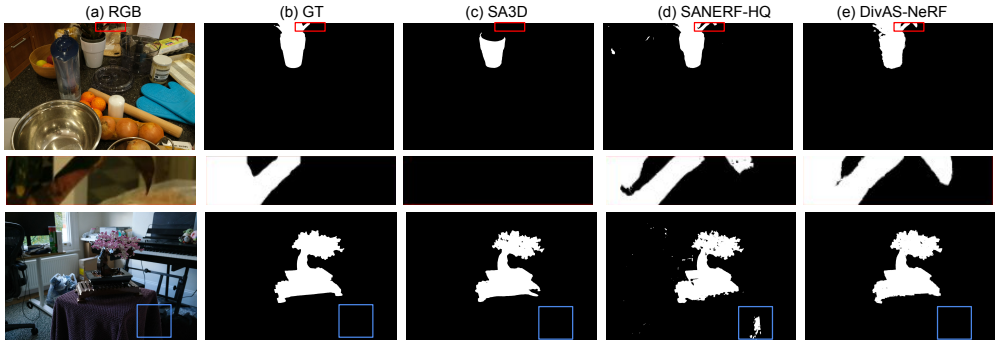


Figure 15: Results of different methods for the NeRF instantiation on Mip-NeRF 360°.

In Figure 15, we compare methods on the *Counter (flowerpot)* and *Bonsai* scenes from Mip-NeRF 360° in the NeRF backbone. In the first row, the red box marks a region where the pseudo ground truth misses part of the flowerpot's leaf. The zoomed-in crops (row 2) show that our method better preserves object boundaries and maintains multi-view geometric consistency. SA3D misses occluded plant regions due to its single-view prompting strategy. SANeRF-HQ does not match the complete leaf structure shown in row 2 compared to our method. In the third row *Bonsai*, SANeRF-HQ again shows over-segmentation, assigning distant background areas as part of the target object (blue box). Our method produces sharper, more localized masks that are consistent with the true object geometry.

G Additional Ablations

The ablations in Section 4.4 cover the sensitivity analysis of the hyperparameters used in NeRF and the corresponding counterpart in 2DGS instantiation. In contrast to the NeRF case, most of the 2DGS defaults are scene-specific. The reason is structural: the voxel size is adaptive per scene, and the surfel primitives are anisotropic and randomly oriented, so a single global value cannot be generalized across scenes. We expose these as user-tunable sliders. The defaults shipped work well across the scenes; the user can expect approximately ±1-1.5 percentage points of IoU around the reported results in Section 4.2 without tuning.

G.1 Cumulative-weight threshold τ_{cw} (NeRF)

The cumulative-weight cutoff τ_{cw} sets the upper bound of the depth range $\boldsymbol{D}_{\max}$ that is fed to the fusion kernel. We sweep it over {0.15-0.9} on the four representative scenes of Section 4.4, namely *Trex* and *Fern* from LLFF and *Bonsai* and *Garden (without vase)* from Mip-NeRF 360°. As shown in Figure 16, the IoU remains stable for $\tau_{\text{cw}} \in [0.3, 0.75]$ and drops sharply at higher values (≥ 0.9). At such thresholds, the valid depth range widens until background voxels become activated, reducing segmentation precision. The IoU of Mip-NeRF 360° scenes (*Bonsai*, *Garden*) is nearly insensitive to τ_{cw} because hierarchical sampling places the bulk of density samples near the surface, so the cumulative weight converges quickly regardless of the cutoff. LLFF scenes show more variation because uniform sampling along the ray requires a higher τ_{cw} to exhaust the foreground density and can include background at higher values. Although τ_{cw}=0.6 gives a marginally higher IoU on *Trex*, qualitative evaluation on *Fern* favors 0.75 for smoother and more complete masks. We fix τ_{cw}=0.75 as the NeRF default.

G.2 Base tolerance multiplier γ

Figure 17 reports the effect of the base tolerance multiplier γ on IoU for LLFF and mIoU for Mip-NeRF 360° scenes. Across LLFF (Trex and Fern), IoU remains stable for $\gamma \in [0.1, 2.0]$ and shows a mild drop at larger values. However, the qualitative results in Figure 18 reveal an important behavior that IoU alone does not capture.

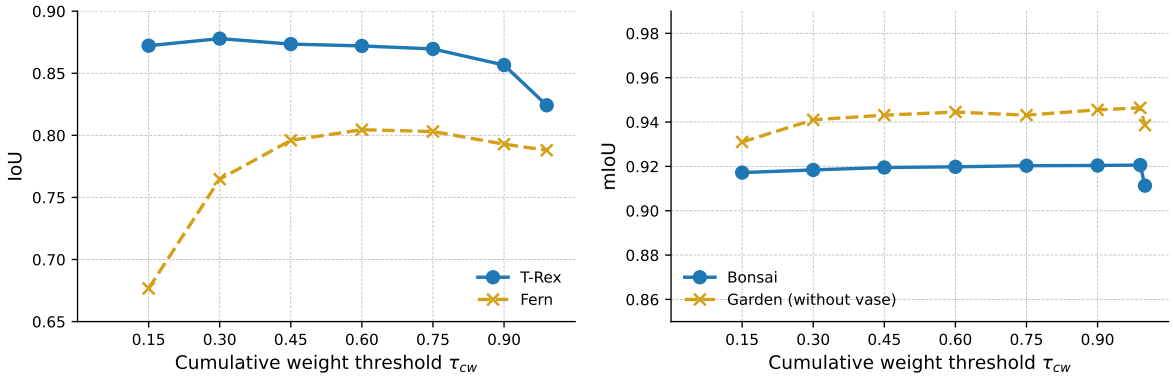


Figure 16: IoU vs. cumulative-weight target τ_{cw} for the NeRF instantiation on representative LLFF and Mip-NeRF 360° scenes.

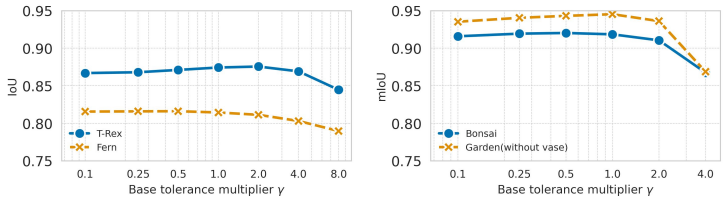


Figure 17: IoU/mIoU vs. base tolerance multiplier γ.

For the Trex scene, γ=2.0 produces a slightly higher IoU than γ=4.0, but it systematically fails to recover thin structures around the jaw, ribs, spine, and near the head boundary of the object. The zoomed-in crops (row 2) highlight how γ=4.0 restores the head skeleton while still suppressing background noise. Because thin-structure completeness is essential for our interactive segmentation setting, we adopt γ=4.0 for LLFF scenes.

For Mip-NeRF 360° (Bonsai and Garden), the trend differs. mIoU is nearly flat in the interval $\gamma \in [0.1, 2.0]$, with a slight peak at γ=0.5, and then sharply degrades at γ=4.0 (Figure 17). The qualitative comparisons in Figure 19 support this, γ=0.1 under-segments thin boundaries, whereas γ=4.0 introduces strong background leakage. γ=0.5 provides the best trade-off, preserving edge detail without introducing artifacts.

Overall, while IoU curves are relatively flat across a wide range of γ, the qualitative sensitivity to thin structures and background leakage motivates our scene-dependent defaults (γ=4.0 for LLFF, γ=0.5 for Mip-NeRF 360°).

2DGS instantiation. The 2DGS analog is scene-specific. This is because of the reason stated in G. We therefore expose this parameter as a user-tunable slider with a per-scene default.

G.3 Low-sensitivity kernel-parameter sweeps

The remaining NeRF fusion-kernel parameters change the score only marginally across their full sweeps. To keep the presentation compact, we stack the LLFF and Mip-NeRF 360° measurements side by side and group the four parameters into two paragraphs rather than separate subsections.

Per-sample bonus β and maximum tolerance bonus $b_{\max}$. Table 14 reports the effect of the per-sample bonus β, which controls the depth tolerance based on the contribution of the number of samples involved in ray marching for a pixel. Across all scenes, the IoU variation with respect to β is small. Extremely low values, such as β=0.01, yield slightly higher IoU in Trex and Fern, but the gains are numerically marginal

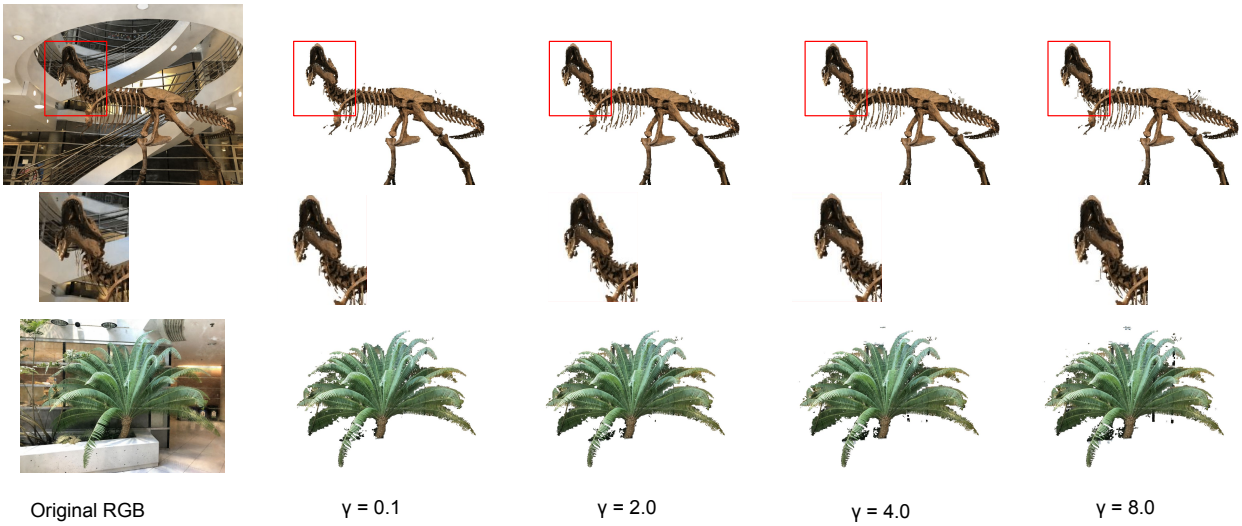


Figure 18: Qualitative ablations for LLFF scenes on the base tolerance multiplier γ.

Figure 19: Qualitative ablations for Mip-NeRF 360° scenes on the base tolerance multiplier γ.

Table 14: Effect of the per-sample bonus β and the maximum tolerance bonus $b_{\max}$ on IoU (LLFF: Trex, Fern) and mIoU (Mip-NeRF 360°: Bonsai, Garden), NeRF instantiation.

β						$b_{\max}$					
	LLFF			Mip-NeRF 360°			LLFF			Mip-NeRF 360°	
β	Trex	Fern	β	Bonsai	Garden	$b_{\max}$	Trex	Fern	$b_{\max}$	Bonsai	Garden
0.01	0.877	0.807	0.01	0.919	0.935	1	0.876	0.808	0.10	0.918	0.933
0.05	0.876	0.812	0.02	0.919	0.938	5	0.870	0.807	0.25	0.919	0.937
0.10	0.875	0.810	0.05	0.920	0.943	10	0.869	0.803	0.50	0.920	0.939
0.20	0.869	0.803	0.10	0.920	0.943	20	0.869	0.802	1	0.920	0.943
0.30	0.865	0.800	0.20	0.919	0.943	40	0.869	0.802	2	0.920	0.943
0.40	0.860	0.799	0.40	0.919	0.944				4	0.920	0.943

Table 15: Effect of the depth range factor λ_{range} and the density threshold ρ_{thresh} on IoU (LLFF: Trex, Fern) and mIoU (Mip-NeRF 360°: Bonsai, Garden), NeRF instantiation.

λ_{range}						ρ_{thresh}				
	LLFF			Mip-NeRF 360°			LLFF		Mip-NeRF 360°	
λ_{range}	Trex	Fern	λ_{range}	Bonsai	Garden	ρ_{thresh}	Trex	Fern	Bonsai	Garden
0.01	0.869	0.803	0.1	0.921	0.942	1	0.852	0.767	0.920	0.946
0.02	0.869	0.804	0.5	0.920	0.943	5	0.866	0.797	0.920	0.943
0.05	0.869	0.803	1	0.920	0.943	10	0.869	0.803	0.920	0.939
0.10	0.866	0.800	2	0.920	0.943	20	0.871	0.787	0.920	0.933
0.20	0.863	0.797	5	0.920	0.943	50	0.871	0.637	0.921	0.925
0.40	0.862	0.797	10	0.920	0.943					
0.80	0.862	0.797								

and do not accurately reflect the actual segmentation quality. Qualitatively, very small β behaves similarly to a low γ value (Section G.2), leading to underestimation of thin structures and missing fine skeletal parts even when IoU appears high. Larger values ($\beta \geq 0.2$) instead provide a more stable tolerance growth and consistently recover fine structures without introducing background leakage, so we adopt β=0.2 as a balanced default for the NeRF instantiation. The same table reports the maximum accumulated tolerance bonus $b_{\max}$, which caps how much β can relax the geometric-consistency threshold. The IoU again changes only marginally as $b_{\max}$ varies over a wide range. Extremely small values (e.g., $b_{\max}$=1) slightly increase IoU on LLFF for the same thin-structure reason discussed for β, while larger values saturate quickly and behave indistinguishably beyond $b_{\max}\geq$10. We therefore adopt $b_{\max}$=10 for LLFF and $b_{\max}$=1 for Mip-NeRF 360°.

2DGS instantiation. The role of the per-sample bonus is inverted in the 2DGS kernel and is therefore not directly comparable to the NeRF parameter swept in Table 14. In NeRF, a high n_{samp} on a pixel indicates dense ray sampling and the bonus *relaxes* the tolerance to admit the additional valid samples. In 2DGS, a high n_{samp} within the footprint indicates many overlapping surfels along the ray, which signals that the rasterizer-reported depth is noisy, so the kernel reuses the same per-sample statistic to *tighten* rather than relax the tolerance.

Depth range factor λ_{range} and density threshold ρ_{thresh}. λ_{range} regulates the depth tolerance used in the geometric-consistency check. Its effect is most visible near foreground-background boundaries, where depth discontinuities may occur. In LLFF, uniform ray sampling produces larger depth gaps, requiring a strict tolerance (λ_{range}=0.01-0.05). In contrast, Mip-NeRF 360° uses hierarchical sampling, concentrating samples near surfaces and permitting a more relaxed setting (λ_{range}=5.0). As Table 15 shows, IoU varies only marginally with λ_{range}, indicating that the parameter plays a stabilizing but low-impact role. The density

Figure 20: Qualitative ablation for Fern on the density threshold ρ_{thresh} (NeRF instantiation).

threshold ρ_{thresh} controls which voxels are considered valid by pruning low-density background regions. Since NeRF scenes typically occupy only 1-3% of the volume, a balanced threshold is essential. Low values (ρ=1) admit background leakage, while high values (e.g., ρ=50) begin to suppress thin or semi-transparent object parts, as the Fern drop to 0.637 at ρ=50 in Table 15 and the qualitative comparison in Figure 20 make clear. LLFF scenes therefore benefit from a moderately strict ρ_{thresh}=10. Mip-NeRF 360° employs a higher-resolution voxel grid (256^3) together with hierarchical sampling, producing a sharper, more localized density field, so the effect of ρ_{thresh} is visually subtle across a wide range of values. This aligns with common NeRF practice, for example, Instant-NGP (Müller et al., 2022) uses ρ=10 for a coarser 128^3 grid, where higher grid resolutions naturally require lower pruning thresholds. We adopt ρ_{thresh}=5 as the default for Mip-NeRF 360°, which best preserves bonsai and garden structures while suppressing background artifacts.

2DGS instantiation. The 2DGS kernel disables λ_{range}. The reason is that the 2DGS rasterizer reports a single planar median-depth value d_{med} around which all surfels tightly pack on the surface of the hollow object, whereas NeRF learns a wide band $[d_{\min}, d_{\max}]$ that includes the object's internal body. The spatial tolerance is therefore driven entirely by the voxel size h_v multiplied by γ and the inverted bonus, and no range-derived slack is needed. The interpretation of voxel occupancy also differs substantially from the NeRF case. Because the 2DGS occupancy grid is constructed by scattering surfel opacity into Morton-indexed voxels, each occupied voxel often corresponds directly to a small set of surface surfels rather than to a volumetric density. We accumulate the opacity contributions of all surfels mapped to a voxel and observe that the median non-empty voxel occupancy is approximately 1.0 across the evaluated scenes, so we adopt a permissive density floor of ρ_{thresh}=0.5, corresponding to half of the typical occupied-voxel support. Unlike the NeRF case, where density thresholding primarily removes volumetric background regions, overly aggressive pruning in 2DGS can permanently eliminate surface surfels because no neighboring occupied voxels exist behind the visible shell to recover the geometry. For this reason, both the thick and thin paths share the same density floor, which is fixed across all scenes and datasets.

G.4 Thin percent cover threshold $\rho_{\text{percent cover thresh}}$

ρ_{cover}	Trex	Fern
0.05	0.862	0.792
0.10	0.868	0.797
0.20	0.869	0.803
0.40	0.858	0.811
0.50	0.852	0.813

Table 16: Ablation on the percentage cover threshold $\rho_{\text{percent cover thresh}}$ (shorthand ρ_{cover}) on IoU for LLFF scenes (NeRF instantiation).

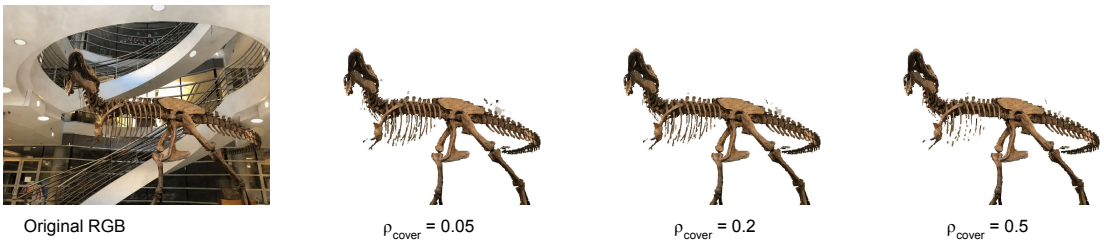


Figure 21: Qualitative ablation for Trex on the thin percent cover threshold (shorthand ρ_{cover}).

ρ_{cover}	Bonsai	Garden (without vase)
0.1	0.920	0.944
0.2	0.921	0.944
0.3	0.920	0.943
0.4	0.920	0.943
0.5	0.918	0.944

Table 17: Ablation on the percentage cover threshold $\rho_{\text{percent cover thresh}}$ (shorthand ρ_{cover}) on mIoU for Mip-NeRF 360° scenes (NeRF instantiation).

Tables 16 and 17 report the effect of the thin percent-cover threshold $\rho_{\text{percent cover thresh}}$ (denoted ρ_{cover}). This parameter determines how many masked pixels must project into a voxel before it is considered a valid thin-structure candidate.

In LLFF scenes, each voxel aggregates contributions from multiple refined SAM mask pixels. While the thick-structure path only checks alignment at the voxel center, many boundary voxels receive partial coverage due to imperfect pixel-voxel alignment or thin geometric structures. The thin-structure path uses ρ_{cover} to detect such cases. Higher thresholds favor conservative labeling and prevent leakage, whereas lower thresholds may trigger over-segmentation.

As seen in Table 16, LLFF is moderately sensitive to this parameter. Qualitative examples in Figure 21 illustrate the trade-off, small values (e.g., ρ_{cover}=0.05) introduce noticeable background leakage, while large values (e.g., ρ_{cover}=0.50) miss thin structures. We therefore adopt ρ_{cover}=0.20 for LLFF.

In contrast, Mip-NeRF 360° is substantially more robust (Table 17). The 256^3 voxel grid provides twice the resolution of LLFF, reducing ambiguity in partial coverage. A slightly larger threshold (ρ_{cover}=0.30) works well in practice and avoids unnecessary voxel marking while keeping computation minimal.

2DGS instantiation. The 2DGS analog is scene-specific. This is because of the reason stated in G. It plays the same role as accepting a thin-structure voxel only when a sufficient fraction of its projected footprint is consistent with the foreground mask and depth.

G.5 Thin density threshold $\rho_{\text{thin_thresh}}$

The thin-structure density threshold $\rho_{\text{thin_thresh}}$ decides when the algorithm activates the thin-structure path. This path uses voxel-footprint projection and recovers details that the thick path often misses. Unlike the standard density threshold, which mainly removes background voxels, $\rho_{\text{thin_thresh}}$ must be higher. It should fire only when NeRF predicts strong density, which signals likely object occupancy. Thin regions, such as the Trex rib bones, show this clearly. They have high NeRF density but very small spatial support. The thick path ignores them, but the thin path can recover them. As Figure 22 shows, low values (e.g., ρ_{thin}=5) cause background leakage, while very large values (e.g., 100) start removing fine structures. The IoU results in Tables 18 and 19 show the same trend. For LLFF, we set $\rho_{\text{thin thresh}}$=50. This value preserves

ρ_{thin}	Trex	Fern
5	0.840	0.779
10	0.842	0.783
30	0.860	0.799
50	0.869	0.803
100	0.868	0.808

Table 18: Ablation on the thin-structure density threshold $\rho_{\text{thin_thresh}}$ (shorthand ρ_{thin}) on IoU for LLFF scenes (NeRF instantiation).

ρ_{thin}	Bonsai	Garden (without vase)
10	0.920	0.945
30	0.920	0.943
50	0.921	0.943
100	0.921	0.943

Table 19: Ablation on the thin-structure density threshold $\rho_{\text{thin thresh}}$ (shorthand ρ_{thin}) on mIoU for Mip-NeRF 360° scenes (NeRF instantiation).

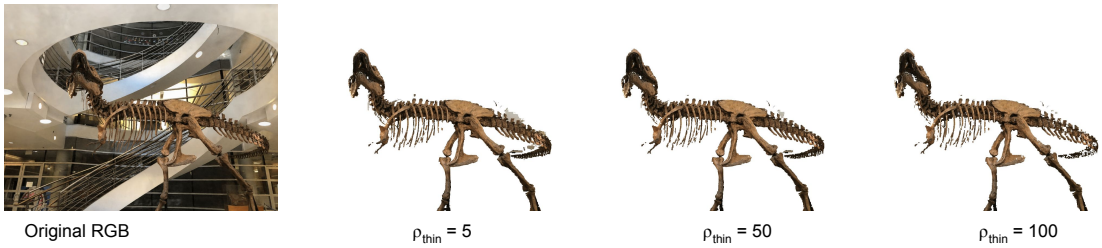


Figure 22: Qualitative ablation for Trex on the thin density threshold $\rho_{\text{thin thresh}}$ (NeRF instantiation).

thin structures while avoiding extra projection work on background voxels. For Mip-NeRF 360°, the finer 256^3 grid gives sharper densities, so a slightly lower value works well there, and we use $\rho_{\text{thin_thresh}}$=30.

2DGS instantiation. As argued in the 2DGS paragraph of Section G.3, the thick and thin paths of the 2DGS kernel share the same density floor, ρ_{thresh}=0.5. We therefore do not maintain a separate $\rho_{\text{thin_thresh}}$ parameter for the 2DGS instantiation. The eligibility check for the thin path inside Algorithm 4 uses the same floor as the thick path.

G.6 2DGS-specific kernel hyperparameters

Two 2DGS hyperparameters that have no NeRF counterparts and are scene-specific are exposed as sliders in the UI.

See-through gap $\gamma_{\mathbf{ST}}$ (slider `see_through_gap`). γ_{ST} controls when the thick path is allowed to enter the thin fallback for a voxel whose center-aligned ray hits empty space between the foreground and the next opaque surface. The acceptance condition is given by Equation 10, anchored to the camera-to-target distance rather than to h_v so that the gate stays scale-invariant across the per-scene adaptive voxel size. Large values open the gate aggressively and admit far-side leakage, while very small values close the gate even on legitimate thin features at oblique angles.

Surfel coverage threshold (slider `surfel_coverage`). The final mask threshold in Pass 3 of the voxel-to-surfel mapping (Algorithm 5, Section 3.6) decides which surfels are kept after the tile-sorted footprint coverage accumulation. Default 0.5. Smaller values yield slightly broader 3D masks, at the expense of boundary surfels with low coverage, leading to over-segmentation. Larger values produce a sharper boundary at the cost of dropping low-coverage thin surfels, leading to holes in the mask.

H Complexity and Runtime Analysis

The end-to-end NeRF-instantiation runtime against SANeRF-HQ (a consistent 1.7-2.1× speedup) is reported in the main paper, Section 4.2 (Table 7). This section provides the finer-grained propagation-time analysis behind that result.

Propagation-time comparison with SA3D. We compare DivAS (NeRF instantiation) against the optimization-based baseline SA3D (Cen et al., 2023). The total propagation cost is reported in Table 20. The SA3D column is the cumulative cost of the mask inverse-rendering pass summed over all training iterations of the SA3D optimization, whereas the DivAS column $T^{\text{tot}}_{\text{pipe}}$ is the cumulative cost of forward render, depth-weighted SAM mask refinement, and CUDA fusion-kernel summed over all centroid views. The speedup column is the ratio of the cumulative cost of SA3D vs DivAS. We additionally report two finer-grained DivAS kernel timings, the cumulative fusion-kernel time $T^{\text{tot}}_{\text{kern}}$ and the per-call mean $\bar{T}_{\text{kern}}$, since these dominate the cost of any single user-driven update. The peak GPU memory consumption is reported only for DivAS. The DivAS peak VRAM is included only to characterize the absolute footprint of our method, not as a competitive claim against SA3D in NeRF instantiation.

As shown in Table 20, DivAS is 3.60× to 33.10× faster than SA3D at the per-method propagation level, with a mean speedup of 18.07×. The gap widens substantially with dataset size, where the speedup is $\sim$ 3.6-8.3× on the bounded LLFF scenes but jumps to $\sim$ 27.3-33.1× on the unbounded Mip-NeRF 360° scenes, which have many more training cameras and a denser scene representation. The reason is structural, SA3D performs a global mask inverse-rendering optimization whose per-iteration cost scales with the number of training cameras and with the trained NeRF field, so its propagation time grows roughly linearly with the training-set size N_{train}. DivAS instead processes centroid views which are dependent on the complexity of the object instead of the training set size. The DivAS propagation budget is therefore largely independent of N_{train}, so larger or denser captures, the regime that inflates SA3D's optimization cost the most, do not slow DivAS down by the same factor. On the absolute kernel cost, $\bar{T}_{\text{kern}}$ stays below 65 ms per call across all four scenes, and the cumulative $T^{\text{tot}}_{\text{kern}}$ stays below 240 ms even on Garden, the bulk of the DivAS propagation

Table 20: Propagation-time comparison between SA3D and DivAS (NeRF instantiation), Peak VRAM consumption by DivAS on four representative scenes. SA3D and $T_{\text{pipe}}^{\text{tot}}$ are in seconds, kernel timings are in milliseconds, and lower is better for the time columns. The Mean row contains arithmetic means across the four scenes for all columns except Peak VRAM, which reports the maximum per-scene peak.

Scene	SA3D (s) ↓	$T_{\text{pipe}}^{\text{tot}}$ (s) ↓	$T_{\text{kern}}^{\text{tot}}$ (ms) ↓	$\bar{T}_{\text{kern}}$ (ms) ↓	Peak VRAM (GB)	Speedup ↑
LLFF Dataset						
Fern	19.934	**5.535**	79	26.0	7.81	3.60×
Trex	45.065	**5.457**	185	62.0	6.86	8.26×
Mip-NeRF 360° Dataset						
Bonsai	172.314	**5.207**	142	28.3	7.94	33.10×
Garden (no vase)	282.419	**10.349**	237	47.4	7.95	27.30×
Mean	129.933	**6.637**	**160.8**	40.9	7.95	**18.07**×

wall-clock is therefore centroid-view rendering and SAM mask refinement, not the fusion kernel itself. Peak GPU memory is bounded by ~ 7.95 GB on the four scenes, well within the limits of consumer hardware.

Complexity Analysis (NeRF instantiation). The fusion kernel scales linearly with the number of active voxels. Each voxel performs a fixed number of projections and depth checks across $|\mathbf{M}|$ depth-refined SAM masks, giving $\mathrm{O}(|\mathbf{M}|)$ cost for the thick-structure path. For thin-structure voxels (approximately 40% of active voxels), we additionally scan a small 2D footprint of $\mathrm{O}(F)$ pixels ($F \approx$ 200-300). Thus, the per-voxel cost becomes $\mathrm{O}(|\mathbf{M}|)$ or $\mathrm{O}(|\mathbf{M}| \times F)$ depending on the local structure type. Since voxel updates are fully parallelized on the GPU and most of the 3D volume is removed by NeRF density pruning, the effective runtime scales as $\mathcal{O}(V_{\text{eff}})$, where V_{eff} denotes density-valid voxels.

2DGS instantiation. The 2DGS fusion kernel preserves the linear-in-active-voxels scaling but operates on a Morton-scattered single-cascade grid constructed once from the trained surfel opacities (Section 3.5.2). The thick path is $\mathrm{O}(|\mathbf{M}|)$ per voxel as in the NeRF case. The voxel footprint in thin path is determined entirely by the projected voxel extent. Footprint statistics over $\boldsymbol{N}_{\text{samp}}$ are still amortized through integral images so that the local mean and variance of $\boldsymbol{N}_{\text{samp}}$ inside this bounding box are read in $\mathrm{O}(1)$, but the per-pixel inner loop itself walks every pixel in F. The per-voxel thin-path cost is therefore $\mathrm{O}(|\mathbf{M}| \times F)$ with the same shape as the NeRF analysis above, only the F regime differs. Empirically (per-call logs on *Garden*, the largest Mip-NeRF 360° scene we evaluate), F ranges from ~ 1.3k to ~ 6.7k pixels with a mean of ~ 2.8k, well above the $F \approx$ 200-300 regime of the NeRF kernel because the 2DGS voxel size h_v scales with the median surfel footprint rather than with a fixed grid resolution and the voxel grid is single cascade compared to multi-cascade in NeRF. The voxel-to-surfel mapping in Algorithm 5 adds three monotone passes whose dominant cost is the tile-sorted preprocess→sort→gather pipeline of Pass3, again linear in the total number of surfels touched.

Metrics. We report five efficiency quantities in Table 21. All wall-clock columns labelled "Train" and "Interact." are reported in **mm:ss** (minutes:seconds), and all GPU-time columns ($T_{\text{pipe}}^{\text{tot}}$, $\bar{T}_{\text{pipe}}$, $\bar{T}_{\text{kern}}$) are reported in **ms** (milliseconds). The pipeline and kernel timings are GPU-side, the time the GPU actually spends inside the rendering, depth-weighting, fusion-kernel, and voxel-to-surfel stages, and exclude user-driven idle periods. (i) The **scene-construction time** "Train" is the one-shot 2DGS (Huang et al., 2024) scene training time. (ii) The **interactive wall time** "Interact." is the wall-clock time of the DivAS-GS method. (iii) The cumulative **pipeline time** $T_{\text{pipe}}^{\text{tot}}$ is the sum of render, depth-weighting, fusion-kernel, and voxel-to-surfel mapping GPU costs over all *centroid* views. This time is from the first user click to the last accepted view, and timing starts when the user requests centroid view generation, at which point all anchor views have already been processed. (iv) The per-call **kernel time** $\bar{T}_{\text{kern}} = T_{\text{kern}}^{\text{tot}}/K_{\text{seg}}$ is the per-call mean of fusion kernel plus voxel-to-surfel mapping, with K_{seg} is count of those centroid views for which a SAM mask was actually fused (views the user accepted without adding a prompt are excluded from this average since the kernel does not run on them). (v) The **per-view pipeline mean** $\bar{T}_{\text{pipe}}$ uses K_{total}, the total

Table 21: DivAS-GS GPU timing analysis of different components on a representative subset of four scenes (two LLFF, two Mip-NeRF 360°).

Scene	Train (mm:ss)	Interact. (mm:ss)	$T_{\text{pipe}}^{\text{tot}}$ (ms)	$\bar{T}_{\text{pipe}}$ (ms)	$\bar{T}_{\text{kern}}$ (ms)
LLFF Dataset					
Fern	18:52	04:27	367	41	21
Trex	13:15	06:26	304	38	27
Mip-NeRF 360° *Dataset*					
Bonsai	12:33	05:02	528	31	28
Garden (no vase)	27:31	09:24	1381	73	67

number of centroid views shown, in the denominator, so the number is directly comparable to the per-view mask-inverse-rendering cost of SA3D-style baselines.

Compute-time observation. Across the four representative scenes in Table 21, the cumulative GPU pipeline cost $T_{\text{pipe}}^{\text{tot}}$ did not cross 1.5 s for an entire interactive segmentation run, and the per-call kernel time $\bar{T}_{\text{kern}}$ stays under 70 ms on every scene. The bulk of the "Interact." wall-clock budget is therefore not GPU compute but user interaction time, the user clicking on object regions, inspecting the back-projected 3D mask, and accepting or refining the next centroid view. This is in sharp contrast to optimization-based baselines, whose total runtime is dominated by an upfront, non-interactive offline optimization pass that the user does not have control. By collapsing the per-update cost to a sub-second pipeline call, DivAS-GS keeps the user in the loop and provides immediate, fine-grained feedback at every view rather than after a multi-minute compute stage.

Propagation-time comparison with SA3D-GS. We compare DivAS-GS against the optimization-based baseline SA3D-GS (Cen et al., 2023). The total propagation cost is reported in Table 22. The SA3D-GS column is the cumulative cost of forward render, loss evaluation through the gaussian primitives, and backward summed over all training iterations of the SA3D-GS optimization, whereas the DivAS-GS column is the cumulative cost of forward render, depth-weight mask refinement, fusion kernel, and voxel-to-surfel mapping summed over all centroid views, and the speedup column is the ratio SA3D-GS / DivAS-GS. As shown in Table 22, DivAS-GS is 2.62× to 13.66× faster than SA3D-GS at the per-method propagation level, with a mean speedup of 7.49×. The gap widens dramatically with scene dataset size, the speedup is $\sim$ 2.6-4.8× on the bounded LLFF scenes but jumps to $\sim$ 8.9-13.7× on the unbounded Mip-NeRF 360° scenes. The reason is structural, SA3D-GS performs a global optimization whose per-iteration cost scales with the number of training cameras and the number of trainable Gaussians, so the propagation time grows linearly with the dataset size N_{train}. DivAS-GS instead processes $3-4$ centroid views per anchor, and the per-call cost is dominated by the active-voxel count, which depends only on the foreground complexity of the scene. This makes the DivAS-GS propagation budget independent of N_{train}, so larger or denser captures do not slow it down compared to the SA3D-GS optimization-based method.

I Limitations and Future Work

DivAS operates on top of an already-reconstructed NeRF or 2DGS scene and therefore inherits the geometric envelope of that reconstruction. Where the underlying backbone yields sparse, noisy, or otherwise low-confidence geometry, for example, under wide-baseline captures or strongly under-sampled regions, the depth-weighted voting can mis-aggregate evidence, and the reconstruction error propagates into the segmentation mask. A second, related case is foreground-background depth ambiguity, when the target object lies very close to a textured background surface in the centroid view, the depth-derived weighting becomes less discriminative, which could lead to background bleeding. The automatic Fibonacci anchor scheduler is geometry-aware but not illumination-aware, so if a proposed anchor observes the object under poor lighting, hard shadows, or weak texture contrast, SAM may return false negatives that propagate into the fused

Table 22: Total propagation time (ms) summed over the full interactive run on the four representative scenes. Lower is better.

Scene	SA3D-GS (ms) ↓	DivAS-GS (ms) ↓	Speedup
LLFF Dataset			
Fern	959.989	**366.515**	2.62×
Trex	1450.179	**303.772**	4.77×
Mip-NeRF 360° *Dataset*			
Bonsai	7213.945	**528.080**	13.66×
Garden (no vase)	12307.233	**1381.418**	8.91×
Mean	5482.836	**644.946**	**7.49**×

3D mask, in our experience this is usually mitigated by picking an alternative anchor through the set of fibonacci views or nudging the centroid-view camera, though it remains an automation gap rather than a hard failure. The centroid zoomed views reduce false negatives and mitigate false positives of the SAM model, but not in all cases, so overall, the segmentation quality of the model is upper-bounded by the SAM model. Transparent and highly reflective materials i.e., non-Lambertian surfaces remain challenging in this setting which is limitation of both NeRF and 2DGS representations which often reconstruct such regions with ambiguous geometry and not able to reconstruct well (e.g., glass in the middle of the table in the Garden scene), and DivAS may then segment only the parts of the surface for which the underlying geometry is locally well-defined, even when the corresponding 2D SAM masks look reasonable. Non-inward-facing trajectories were not evaluated in this work and remain open.

Several directions are open for future work. Extending the automatic view scheduler beyond the inward-facing viewing-sphere assumption would broaden the automated regime to irregular capture trajectories. Making the anchor scheduler illumination-aware so that the GUI proposes lit, well-textured views by default would reduce the residual SAM false-negative rate without additional user effort. Finally, extending the framework to dynamic scenes and more unconstrained capture protocols is a natural next step given that the shared interaction-and-fusion recipe already transfers across representations through representation-specific adapters.