Feature Splatting: Language-Driven Physics-Based Scene Synthesis and Editing

Ri-Zhao Qiu[1], Ge Yang[2,3], Weijia Zeng[1], and Xiaolong Wang[1]

[1] University of California San Diego
[2] Massachusetts Institute of Technology
[3] Institute for Artificial Intelligence and Fundamental Interactions
https://feature-splatting.github.io

Abstract. Scene representations using 3D Gaussian primitives have produced excellent results in modeling the appearance of static and dynamic 3D scenes. Many graphics applications, however, demand the ability to manipulate both the appearance and the physical properties of objects. We introduce *Feature Splatting*, an approach that unifies physics-based dynamic scene synthesis with rich semantics from vision language foundation models that are grounded by natural language. Our first contribution is a way to distill high-quality, object-centric vision-language features into 3D Gaussians, that enables semi-automatic scene decomposition using text queries. Our second contribution is a way to synthesize physics-based dynamics from an otherwise static scene using a particle-based simulator, in which material properties are assigned automatically via text queries. We ablate key techniques used in this pipeline, to illustrate the challenge and opportunities in using feature-carrying 3D Gaussians as a unified format for appearance, geometry, material properties and semantics grounded on natural language.

Keywords: Representation learning · Gaussian Splatting · Scene Editing · Physics Simulation

1 Introduction

What does a falling leaf know about autumn? A video of this moment, had someone captured it, may include a delicate dance with the autumn breeze. Although invisible to the camera, the shape of the wind is carved out by the leaf's swirling path, and thus, became fully visible to our eyes. This technique of using motion to make the invisible visible lies behind many artistic manipulations of images and movies. A leaf that falls straight down versus a leaf that floats across a busy street bouncing up and down tells very different stories. In the latter case we would intuitively reason about the presence of the wind despite not directly feeling its touch on our skins; and we would be drawn in — as if it were us who are being carried away.

In this work, we present Feature Splatting, a way to semi-automatically synthesize dynamic scenes from an otherwise static 3D capture, where we use open-text language queries to jointly manipulate the appearances and assign material

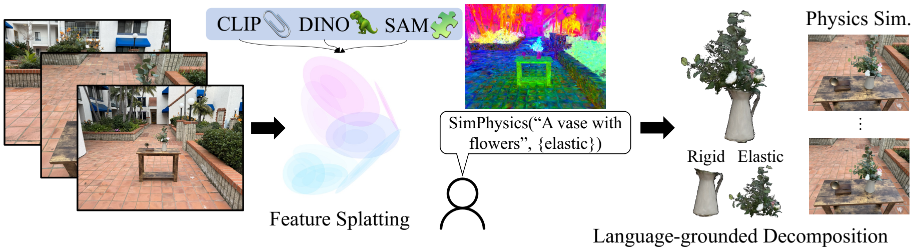

CLIP
DINO
SAM
Feature Splatting
SimPhysics("A vase with flowers", {elastic})
Rigid
Elastic
Language-grounded Decomposition
Physics Sim.

Fig. 1: Feature Splatting. An overview of the language-grounded scene physics editing pipeline. Given input images, feature splatting optimizes for a unified Gaussian representation that contains the geometry, texture, and semantics of the scene using features from large-scale 2D vision models [14,21,22]. With open-vocabulary scene decomposition, feature splatting segments an object and automatically determines the physical properties of components within the object. In this example, a user gives a query 'a vase with flowers'. Feature splatting extracts the vase with flowers in the scene, and further decomposes it into rigid and non-rigid parts, creating a dynamic scene of flowers swaying in the wind. (Best viewed in videos on project website).

properties governing the dynamic interactions. We do so by augmenting existing point-splatting methods that use 3D Gaussians as geometric primitives [12,31,33] with additional view-invariant features sourced from vision, and vision-language foundation models [14, 21, 22] within the same analysis-by-synthesis pipeline. We further extend to physics-based dynamic scene synthesis, by augmenting the static capture with particle-based interaction, where material categories and properties are assigned semi-automatically via natural language queries. The resulting format unifies photo-realism, rich semantics, and physics-based dynamic synthesis in a single format.

Both the extension to feature-carrying 3D Gaussians, and the physics-based dynamic synthesis involve unexpected technical challenges. To begin with, we found that Gaussian primitives from [12] effectively shares the same interpolation kernel for both the geometry and the radiance, whereas the 2D feature maps we source from reference camera views are both low-resolution and noisy. A naive distillation pipeline similar to NeRF-based methods [15] lead to poor results with a lot of high-frequency feature noise (see Fig. 3). We address this problem by introducing a novel method for extracting the feature maps, and a procedure for distilling them. On the dynamic synthesis side, we propose ways to in-fill existing static 3D Gaussian captures for volume-dependent physical effects, and ways to transform the Gaussian primitives under significant deformation that is distinct from, and perform better than prior approaches.

The scene modeling and synthesis pipeline, Feature Splatting, contains rich semantic priors that enables easy, language-driven edits involving decomposing a static 3D capture and associating each constituent with material properties for physics-based dynamic synthesis. Feature-carrying 3D Gaussians serve as a unified representation for appearance, semantics, geometry, and physics.

Our contribution is threefold:

1. **A method**, feature splatting, to augment static scenes with semantics and language-grounded physically realistic movements.
2. **Techniques to the algorithmic and systems challenges** towards a unified representation: **an MPM-based [10] physics engine** that is adapted to Gaussian-based representation; **a novel way to fuse features** from multiple foundation vision 2D models for accurate decomposition.
3. **A demonstration** that feature splatting is an excellent editing tool that enables *automatic* language-grounded scene editing.

2 Related Work

Novel View Synthesis. Recently, the computer graphics community has seen a growing interest in using differentiable rendering to optimize scene representations for novel view synthesis [1,7,12,18,20,28]. Recent techniques can be categorized into implicit methods [1,7,18,20,28] and explicit methods [12]. The representative work of implicit methods is Neural Radiance Fields (NeRFs), which was proposed by [18] and learns a neural network to predict the radiance of the scene. Later work in NeRFs advance the technique by accelerating the training process [20], alleviating the artifacts at the boundary of the scene [28], and addressing texture aliasing [1]. For explicit scene representation methods, Gaussian Splatting (GS) [12] is a recent method that achieves both fast training time and high rendering quality. GS represents the scene as a collection of Gaussians that can be explicitly manipulated. Our work is built on GS without compromising its novel view synthesis capability, where we leverage the explicit representations in GS to connect it with particle-based mechanics and feature distillation.

Scene Editing with Distilled Feature Fields. Various work has proposed ways to make edits with NeRFs. Existing works mainly focus on manipulating the appearance [8,11,15,16]. For instance, Distiled Feature Fields (DFF [15]) performs appearance editing via zero-shot open-text segmentation, in which it uses knowledge distillation to embed features from 2D foundation vision models. During rendering, DFF decomposes the scene by relating language query and distilled features to segment affected volumes. Appearance editing such as color change or removal can then be performed on segmented objects. NeRFShop [11] proposes an interactive pipeline that allows user input to make geometric modifications to NeRF. Instruct-NeRF2NeRF [8] takes a different approach to scene editing. Instead of injecting the rendering process, Haque et al. [8] proposes to use an off-the-shelf 2D editing method [3] to modify the images used to train NeRFs. Most closely related to our work, ClimateNeRF [16] proposes to inject the rendering process with physical simulation to simulate different weather effects. However, due to the intrinsic limitation of implicit scene representation, ClimateNeRF only supports modifying the ray marching progress in neural rendering, which limits it to simulating ray reflection, refraction, and diffraction for weather effects. In comparison, our method uses explicit representations to support object-centric physical simulation, which has much broader potential applications.

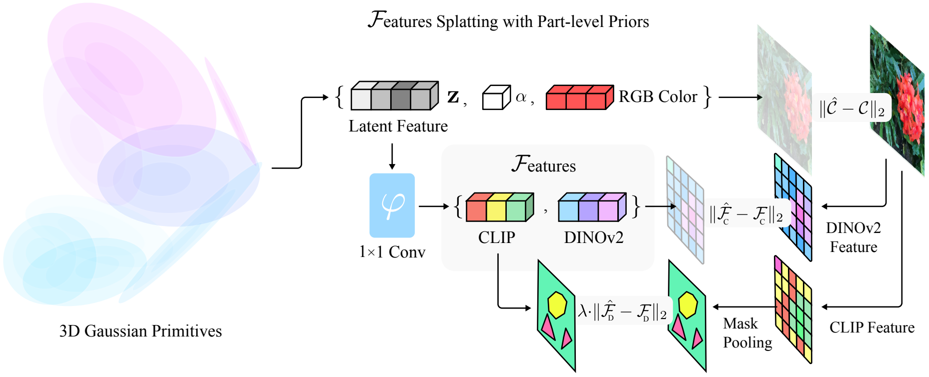

$\mathcal{F}$eatures Splatting with Part-level Priors
3D Gaussian Primitives
$\{$ $\mathbf{z}$, α, RGB Color $\}$
Latent Feature
φ
1×1 Conv
$\mathcal{F}$eatures
$\{$ CLIP , DINOv2 $\}$
$\|\hat{\mathcal{C}} - \mathcal{C}\|_2$
$\|\hat{\mathcal{F}}_{\text{C}} - \mathcal{F}_{\text{C}}\|_2$
DINOv2 Feature
$\lambda \cdot \|\hat{\mathcal{F}}_{\text{D}} - \mathcal{F}_{\text{D}}\|_2$
Mask Pooling
CLIP Feature

Fig. 2: Feature Splatting. Raw CLIP features are noisy and low-resolution. We improve the quality of the feature maps by pooling within part-level masks produced by the Segment Anything Model (SAM [14]). Jointly modeling features from DINOv2 [21] and CLIP is an optional regularization that offers minor additional improvements.

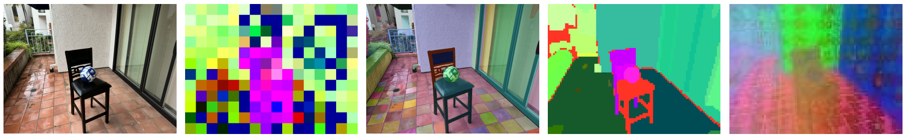
Concurrent Work. During the preparation of this manuscript, a few concurrent works appeared, that also studied ways to edit or simulate dynamic scenes via 3D Gaussians. The majority of these works focus solely on segmenting the scene via language-grounded semantics [14, 25, 30, 32]. Most closely related to our work, Xie *et al.* [29], propose a similar rendering-simulation pipeline that also uses material point methods (MPM). Semantic grounding in natural language is not part of their proposal, and they manually select and assign material properties to the Gaussians. The way these two works handle the rotation of Gaussians is also different. We include results that show rotation from deformation gradients, proposed in [29], which fails to maintain rendering quality when deformation is large. Along the line of works that perform segmentation, Feature3DGS [32] is most relevant to our work for its feature distillation designs to fuse 2D reference features using priors from multiple foundation models. We consider feature distillation one component of a larger simulation, rendering, and editing pipeline, with systems optimization techniques that speed up training by 30%.

3 Language-Driven Physics-Based Synthesis and Editing

We present three key components of our dynamic scene synthesis and editing pipeline: First, a way to distill rich semantic features from vision-language models into 3D Gaussians. Second, a way to decompose a scene into key constituents using open-text queries. Finally, a way to ground the material properties via language, as part of a physics-based dynamic scene synthesis procedure.

3.1 Differentiable Feature Splatting

Point and surface splatting methods [31, 33] represent a scene explicitly via a mixture of 2D or 3D Gaussian primitives. In the case of Gaussian Splatting [12],



(a) Input image (b) CLIP feat. (c) SAM masks (d) SAM-enhanced feat. (e) DINOv2 feat.

Fig. 3: Raw and Enhanced Feature Maps. CLIP features contain view-dependent noise that degrades the feature splats [22]. We mask-pool with masks produced by SAM [14], and regularization through joint modeling of DINOv2 features [21] to improve its quality. Color corresponds to top three PCA vectors.

the geometry is represented as a collection of 3D Gaussian, each being the tuple $\{\mathcal{X}, \mathbf{\Sigma}\}$ where $\mathcal{X} \in \mathbb{R}^3$ is the centroid of the Gaussian and $\varSigma$ is its covariance matrix in the world frame. This gives rise to the probability density function

$$G(\mathcal{X}, \varSigma) = \exp -\frac{1}{2}\mathcal{X}^\top \varSigma^{-1}\mathcal{X}\,. \tag{1}$$

To ensure $\varSigma$'s positive semi-definiteness during optimization, it is common practice to decompose it into a scaling matrix $\mathbf{S}$ and a rotation matrix $\mathbf{R}$ via $\varSigma = \mathbf{RSS}^\top\mathbf{R}^\top$. The color information in the texture is encoded with a spherical harmonics map $\mathbf{c}_i = \mathrm{SH}_\phi(\mathbf{d}_i)$, which is conditioned on the viewing direction ϕ.

Feature Splatting. Feature Splatting appends an additional vector $\mathbf{f}_i \in \mathbb{R}^d$ to each Gaussian, which is rendered in a view-independent manner because the semantics of an object shall remain the same regardless of view directions. The rasterization procedure starts with culling [12] the mixture by removing points that lay outside the camera frustum. The remaining Gaussians are projected to the image plane according to the projection matrix $\mathbf{W}$ of the camera. This projection also induces the following transformation on the covariance matrix $\varSigma$:

$$\varSigma^{'} = \mathbf{J}\,\mathbf{W}\,\varSigma\,\mathbf{W}^\top\mathbf{J}^\top\,, \tag{2}$$

where $\mathbf{J}$ is the Jacobian of the projection matrix $\mathbf{W}$. We can then render both the color and the visual features with the splatting algorithm:

$$\{\hat{\mathbf{F}}, \hat{\mathbf{C}}\} = \sum_{i \in N} \{\mathbf{f}_i, \mathbf{c}_i\} \cdot \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j)\,, \tag{3}$$

where α_i is the opacity of the Gaussian conditioned on $\varSigma^{'}$ and the indices $i \in N$ are in the ascending order determined by their distance to the camera origin.

Systems Considerations. Naïvely extending the color rasterizer in vanilla GS [12] to rasterize high-dimensional features causes expensive training time. We performed an in-depth analysis and found that the main bottleneck lies in the memory access pattern, which we addressed by designing custom CUDA kernels. The details are presented in the appendix.

Improving Reference Feature Quality Using Part-Priors. Our differentiable feature splatting is a generic method, where the resulting features depend on the reference features. Though CLIP [22] is a commonly used 2D vision models to obtain language-aligned features, naïvely splatting CLIP features results in low-quality 3D features given the coarseness of the CLIP features (see Fig. 3 and Fig. 7a). This issue is less pronounced in NeRF-based methods [13, 15] because the continuous representations of NeRF serves as an implicit regularization. On the other hand, explicit representations such as GS [12] have no such regularization and are prone to overfit to noises inherited from coarse reference feature maps.

We propose a way to improve the quality of the Gaussian features using object priors from DINOv2 [21] and the Segment Anything Model (SAM) [14]. Consider an input image. We first use SAM to generate a set of part-level masks $\{\mathbf{M}\}$ (see Fig. 3c). For a given binary mask $\mathbf{M}$ and the coarse CLIP feature map $\mathbf{F}_C$, we use Masked Average Pooling (MAP) to aggregate a single feature vector

$$w = \mathrm{MAP}(\mathbf{M}, \mathbf{F}_C) = \frac{\sum_{i \in \mathbf{F}_C} \mathbf{M}(i) \cdot \frac{\mathbf{F}_C(i)}{||\mathbf{F}_C(i)||}}{\sum_{i \in \mathbf{F}_C} \mathbf{M}(i)}\,, \tag{4}$$

where i is a pixel coordinate in the feature map. w is then assigned to all pixels that are within the part segmentation. If a pixel belongs to multiple parts, the pixel feature is obtained by averaging all relevant part features. This gives us a SAM-enhanced CLIP feature map (shown in Fig. 3d).

To further reduce the possibility of overfitting, we introduce a shallow MLP with two output branches that takes in the rendered features $\hat{\mathbf{F}}$ as intermediate features. The first branch renders the DINO [4] feature $\hat{\mathbf{F}}_D$ for its coherent part-level semantics (see Fig. 3e), and the second branch renders the CLIP [22] features $\hat{\mathbf{F}}_C$.

$$\hat{\mathbf{F}}_C, \hat{\mathbf{F}}_D = \mathrm{MLP}(\hat{\mathbf{F}})\,, \tag{5}$$

where $\hat{\mathbf{F}}_C$ is supervised using the SAM-enhanced CLIP feature map with cosine loss, and $\hat{\mathbf{F}}_D$ is supervised using the DINOv2 [21] feature map using cosine loss. We scale the CLIP term in the joint loss $\mathcal{L}_{\text{CLIP}} + \lambda \cdot \mathcal{L}_{\text{DINO}}$ with $\lambda = 0.1$, so the optimization focuses on language grounding and treat DINO features as a mild smoothing term. In sum, this would give us Gaussians with regularized CLIP features.

3.2 Language-guided Scene Decomposition

We first perform object- and part-level open-vocabulary scene decomposition, where we take language queries to coarsely select objects for editing (*e.g., a vase with flowers*) and feature splatting automatically decouples object components for simulation (*e.g., vase is rigid and stems of flowers are elastic*). We do so by identifying Gaussians whose CLIP features more closely align with positive queries over negative queries.

More specifically, given a positive vocabulary (*e.g.,* L^+ = *'bulldozer'*) and generic negative vocabularies $\{L_i^-\}_{i=0}^N$ (*i.e., 'objects' and 'things'*), we use frozen CLIP text encoder to obtain text embeddings for every vocabulary. Then we follow standard CLIP practice [22] and compute pair-wise cosine similarity between rasterized CLIP feature of every Gaussian and the text embeddings. A temperatured softmax is then applied to similarities to obtain probability distribution, where we select Gaussians whose similarity to L^+ passes a certain threshold $\tau = 0.6$ as the foreground object. We include segmentation results using negative text-queries in the appendix.

Basic Editing Primitives. Feature splatting supports various basic editing primitives, which can be easily composed to achieve more complex behavior. Let $\{\hat{\mathcal{X}}, \hat{\mathbf{\Sigma}}\} \subseteq \{\mathcal{X}, \mathbf{\Sigma}\}$ be the set of Gaussians selected to be edited. We briefly describe how basic editing primitives are implemented.

- **Object Removal.** Objects are removed by simply removing selected Gaussians $\{\mathcal{X}, \mathbf{\Sigma}\} \coloneqq \{\mathcal{X}, \mathbf{\Sigma}\} \setminus \{\hat{\mathcal{X}}, \hat{\mathbf{\Sigma}}\}$.
- **Translation.** Given a displacement vector $\mathbf{b}_1 \in \mathbb{R}^3$, objects can be easily displaced by shifting the Gaussian centroid $\hat{\mathcal{X}} \coloneqq \hat{\mathcal{X}} + \mathbf{b}_1$.
- **Rotation.** Given a rotation matrix $\mathbf{R}_1 \in \mathbf{SO}(3)$, we modify the covariance of selected Gaussians as $\hat{\Sigma} \coloneqq \mathbf{R}_1\hat{\mathbf{R}}\hat{\mathbf{S}}\hat{\mathbf{S}}^\top\hat{\mathbf{R}}^\top\mathbf{R}_1^\top$.
- **Scaling.** Given an axis-aligned scaling vector $\mathbf{s}_1 \in \mathbb{R}^3$, we can scale the size of objects via $\hat{\mathcal{X}} = \mathbf{s}_1\hat{\mathcal{X}}, \quad \hat{\Sigma} \coloneqq \hat{\mathbf{R}}(\mathbf{s}_1\hat{\mathbf{S}})(\mathbf{s}_1\hat{\mathbf{S}})^\top\hat{\mathbf{R}}^\top$.

3.3 Language-Driven Physics Synthesis.

Feature splatting can automatically choose physical properties for simulation, estimate collision surfaces, and predict gravity for simulation. Based on the explicit representation, we extend the material-point method (MPM) [26] using Taichi [10] to augment objects with various physical properties.

Decoupling Objects for Simulation. We construct a set of vocabularies for common rigid materials such as (*e.g., wood, ceramic, and steel*), which can be expanded with optional user inputs. Given a selected multi-part object (*e.g., a vase with flowers*), we perform another round of CLIP similarity comparison to select particles within the object that are aligned more closely to these materials. The selected particles are considered to be rigid during simulation.

Language-grounded Collision Surface Estimation. We feed a canonical set of queries including common planar objects (*e.g., floor and tabletop*) into the scene decomposition pipeline described above to obtain Gaussians of these objects. Then we apply RANSAC [6] to estimate these plane geometry, which are used as collision surfaces in physics simulation. The gravitational directional vector is estimated as the normal vector of plane geometry of "floor".

Taichi MPM for gaussians. Given an selected object with rigid parts, collision surfaces, and gravity, we may simulate physics with particle mechanics. While MPM methods [10, 26] can be directly applied to Gaussians by treating the Gaussian centroids $\mathcal{X}$ as point clouds, doing so results in unsatisfactory quality. Parallel to findings by Xie *et al.* [29], we find that simulating and displacing only the surface centroids leads to issues such as 1) object collapsing on contact with collision surfaces due to lack of internal support and 2) undesired artifacts when objects undergo deformation.

We build our gaussian-oriented material-point method (GS-Taichi-MPM) based on Taichi [10], which supports realistic physical simulation of various types of materials (*e.g., rigid, elastic, granular (sand), and liquid*). Our method goes beyond simple point-based physics simulation and makes use of gaussian-specific information, such as isotropic opacity and Gaussian covariances for volume preservation and covariance modification during deformation, to address the two aforementioned challenges.

- **Implicit Volume Preservation.** One of the open challenges in simulating physics from a few real-world images is volume preservation. For instance, without volume preservation, a volleyball simulated to hit the ground would collapse upon collision. Hence, we propose an implicit volume preservation technique using the opacity and covariances of gaussians. Specifically, we first densify surface points by sampling points on disks of surface gaussians using covariance and opacity information following [27]. With densified surface points, we then fill transparent supporting particles extending from the centroid of the object to the surface. The transparency of filled particles is intended to enforce identical rendering quality at $T = 0$ for a smooth and continuous transition from static scene to dynamics.
- **Estimating Rotation.** When the object undergoes deformation, artifacts may manifest without correcting the rotational component of the covariance matrix. Notably, Xie *et al.* [29] noted a similar issue and attempted to use the deformation gradient F of MPM to update Gaussian covariances. With slight abuse of the Σ notation, let $\mathbf{F} = U\Sigma V^\top$ be the per-particle defomration gradient and its SVD factorization, rotation from deformation can be estimated as $\mathbf{R}_1 = VU^\top$. However, deformation gradients capture mostly local deformation, and we empirically find that this approach fails when elastic objects undergo large deformation (see Fig. 7b).

  We propose to estimate the rotation matrix for *elastic* objects using normals. Since Gaussian splatting does not estimate normals due to challenges in sparse reconstruction [12], we use an alternative NN-based approach. In particular, for every Gaussian to be simulated, we find two of its nearest neighbors. The centroids of these three Gaussians form a plane, and we use the rotation of the plane normal throughout the object dynamics as a proxy. Compared to rotation from deformation, our approach yields fewer artifacts when objects undergo large deformation (see Fig. 7b).

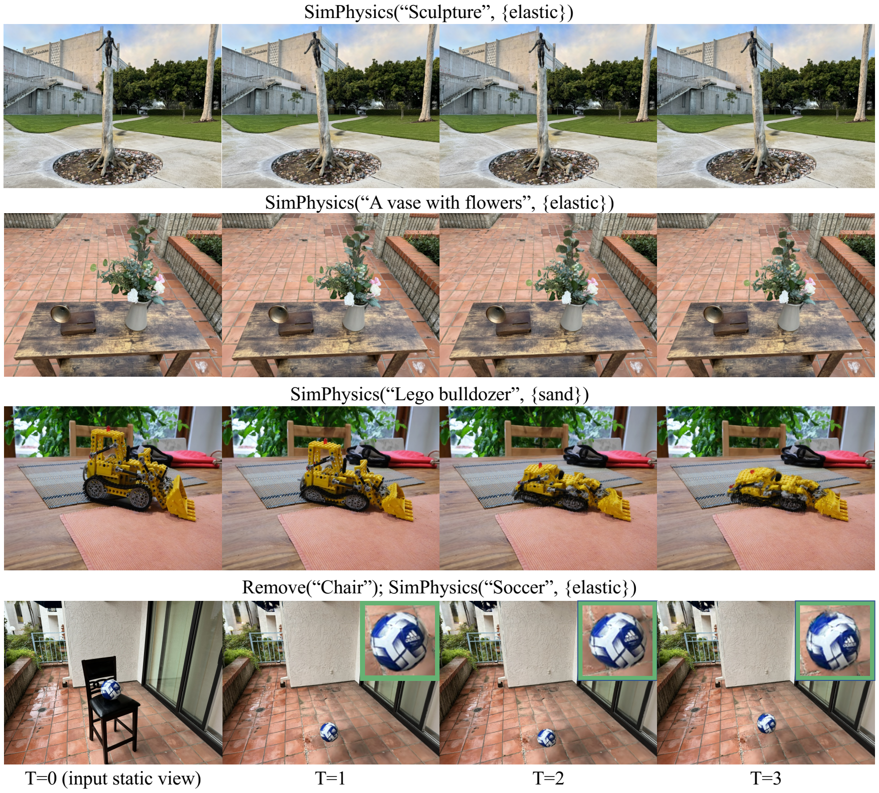

SimPhysics("Sculpture", {elastic})
SimPhysics("A vase with flowers", {elastic})
SimPhysics("Lego bulldozer", {sand})
Remove("Chair"); SimPhysics("Soccer", {elastic})
T=0 (input static view)
T=1
T=2
T=3

Fig. 4: Physics-based Dynamic Scene Synthesis. Rich semantic features in Feature Splatting enable semi-automatic assignment of material properties for synthesizing dynamic scenes from a single static 3D capture. We can use simple text queries to manipulate the physical property of specific objects and materials. From top to bottom: changing the elasticity, turning solid into granular material, modeling volume-dependent deformation in a falling volleyball. **For the best illustration with animations and moving cameras, please refer to videos on the project website.**

4 Experiments

We provide qualitative results on language-driven editing and dynamic synthesis, and quantitative comparisons against radiance-field based approaches when applicable.

Datasets. We use the deep blending dataset [9] and the Mip-NeRF360 [2] dataset, where we compute the camera intrinsics, extrinsic, and sparse point clouds using colmap [23, 24]. Following LERF [13], we evaluate the localization capability of feature splatting with 72 objects using language query in five localization scenes from LERF [13]. Finally, we collect a custom dataset to demonstrate the capability of our method to perform physical simulations. The custom data sequences are captured using the main camera of an iPhone 15 Pro with

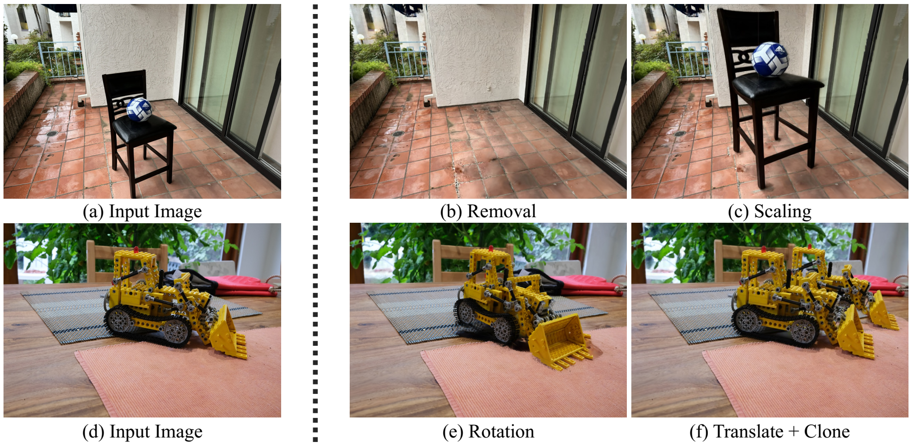


Fig. 5: Feature Splatting Editing Primitives.: We can remove, scale, rotate, translate, and clone objects in the scene using language.

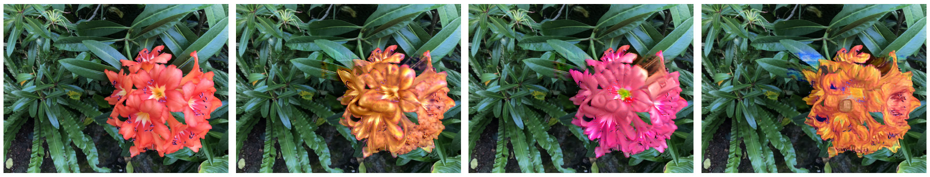
intrinsics, extrinsic, and initial sparse point cloud computed by colmap. We plan to release the custom dataset to aid future research.

Metrics. For physics editing, since there are no ground truth images or previous baselines to compare to, we focus on qualitative comparisons. To evaluate the localization accuracy, we follow the same evaluation protocol as LERF [13] and compute localization accuracy on testing views. We also evaluate how much different components in our systems contribute to training efficiency. Finally, we report PSNR and cosine feature rendering loss on training and validation views to validate necessity for fusing multiple models and show that feature splatting does not interfere with rendering quality.

4.1 Dynamic Scene Synthesis Results

We present synthesized dynamic scenes that involves elastic deformation, loose, granular materials, and volume-preserving deformations with a volleyball in Figure 4. These results are produced by selecting from a bank of preset material properties using sparse, text labels in natural language. Our physics-based synthesis pipeline creates realistic movements that reflects either the underlying material, or during editing, the intent of the user.

Realistic, Physic-Based Dynamics. Using features from large-scale 2D vision models [14, 21, 22], feature splatting is capable of not only segmenting objects for editing, but also ground physical properties of components within objects using language. In the second sequence, given the text prompt 'a vase with flowers', the flowers sway as if it were blown with winds, but the vase remains still as it is considered as rigid materials by feature splatting. The fourth sequence emphasizes volume preservation, where the elastic ball is internally filled and bounces off the ground.



(a) Original (b) "Golden Flower" (c) "pink Flower" (d) Van gogh Sun Flower

Fig. 6: Language guided appearance editing. we optimize the SH coefficients via cosine similarity between CLIP embedded language and image feature.

Spatial consistency. Conceptually, our method synthesizes views that are consistent across different camera viewports and timesteps of the physical simulations. For all sequences in Fig. 4, feature splatting is able to synthesize 3D-consistent views. When objects are removed or deformed from their original places, the regions that were originally occluded are nicely synthesized.

Temporal continuity. Our physical editing approach leverages differentiable physics simulation engine [10] and our volume preservation technique fills the interior of gaussians using transparent particles. Combined, our technique enforces a smooth and continuous transition from static scenes to object dynamics, where the rendered views are consistent with vanilla GS [12] at $T = 0$. Compared to some previous scene-editing methods that use black-box generation models [8], our synthesis process offers both temporal consistency and great explainability.

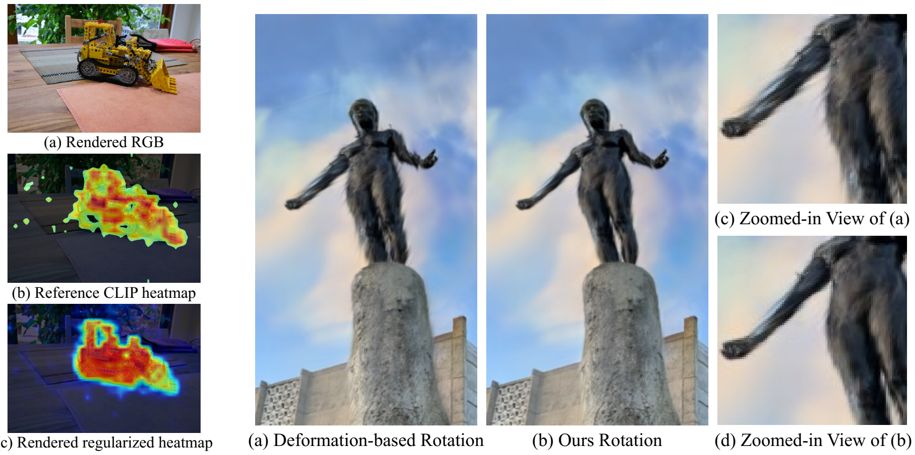
Real-time Efficiency. Feature splatting maintains the ability to perform real-time rasterization, similar to Gaussian splatting [12]. The bottleneck of our physics simulation pipeline is the Taichi physics simulation engine, which runs at an approximate average of *30 fps* on a desktop-grade GPU. Our feature splatting pipeline is optional during inference. Thus, with computed particle trajectory, images can be synthesized at approximately *100 fps*.

4.2 Editing Appearance and Geometry

With the unified representation of feature splatting that hosts geometry, texture, and semantics, we demonstrate how to interact with objects in the scene via basic editing primitives and language-guided appearance editing.

Geometric Editing. We showcase geometry editing using feature splatting, highlighting object removal, scaling, rotation, translation, and cloning, as illustrated in Fig. 5. With our open-vocabulary scene decomposition design that fuses features from multiple pre-trained models, these geometric operations are fully automatic and the editing results are artifact-free, which is of great potential for practical applications.

Appearance Editing. Thanks to the unified representations in feature splatting, editing the appearance of object is also straightforward. In Fig 6, we demonstrate the capability to edit the appearance of objects using CLIP



(a) Rendered RGB

(b) Reference CLIP heatmap

(c) Rendered regularized heatmap

(a) Deformation-based Rotation

(b) Ours Rotation

(c) Zoomed-in View of (a)

(d) Zoomed-in View of (b)

(a) Ablations of multi-model regularization. Without regularization, rendered features completely overfit to reference CLIP features with artifacts.

(b) Ablations of covariance rotation techniques. Though both methods perform similarly when the deformation is small, artifacts can emerge if we use rotations from deformation when the object undergoes large rotation and displacement.

Fig. 7: Ablations of the regularization effects of fusing features from multiple pretrained models and our rotation estimation technique.

guidance. We train a scene using feature splatting then update only the SHs of the selected Gaussian centroids using cosine loss between rendered image and CLIP with text prompt: "A photo of an *Adj.* flower". As demonstrated, feature splatting excels in both color and style editing while preserving the background in just 2,500 iterations.

4.3 Ablations

We provide quantitative results on the impact of the system improvements, and ablation studies on our proposed techniques on feature splatting and the physics-based dynamic synthesis.

System Optimization. Here we ablate the efficiency of our engineering contributions. For baseline, we trivially extend the vanilla GS [12] implementation to render N-dim features instead of 3-dim RGB values. The results are presented in Table. 1. Due to the prohibitively expensive memory requirement of baseline after densification, we compute the timing in the table by training only 1,000 iterations on provided sequences in the DB dataset [9].

Our optimized implementation has better timing than our baseline implementation and Feature-3DGS [32]. In comparison, feature splatting generally requires less than 1 hour on average for training, whereas Feature3DGS [32] empirically measured to require 6 hours. We believe that the efficiency of our feature training implementation can facilitate future research.

Table 1: Timing ablation of our optimization techniques. Feature splatting reduces the training time by over 60% compared to the baseline even when the feature dimension is the same. *Half2* stands for half2 compiler intrinsics, *GBuf* stands for shared gradient buffer, and *IMem* is interleaved memory access.

Feat. Dim.	Half2	GBuf	IMem	Time
768	-	-	-	3.21
	✓	-	-	1.96(-38.9%)
	✓	✓	-	1.54(-52.0%)
	✓	✓	✓	1.21(-62.3%)
256	Feature-3dgs [32]			0.61
	✓	✓	✓	0.41
32 (Ours)	✓	✓	✓	**0.06(-97.1%)**
0	Color-only GS [12]			0.05

Table 2: Comparison of localization accuracy between OWL-ViT [19], LERF [13], and feature splatting. Our method performs best on 2D localization and can directly localizing objects in 3D, which is not possible for LERF [13].

Loc. Space	Method	Accuracy
2D	OWL-ViT [19]	54.8%
	LERF [13]	80.3%
	Ours-CLIP-only	73.0%
	Ours-CLIP-DINO	71.4%
	Ours (full)	**81.7%**
3D	LERF [13]	-*
	Ours	50.7%

Fusing Features from Multiple Models. In Fig. 7a, we provide qualitative comparisons of heatmap generated by comparing text queries and rendered features to ablate the effectiveness of fusing multiple pre-trained vision models. We can see that the powerful ability of Gaussian splatting causes the guassian scene representation to *overfit* reference features, which results in the model fitting high-frequency details of reference features including imperfections in the coarse reference feature map. In comparison, with DINO regularization technique, the model learns a smooth feature representation that adheres to the boundary of objects, which is desirable. The overfitting without regularization is further quantitatively validated in Table. 3, where the unregularized model has better CLIP feature rendering on training views, but worse on validation views.

Rotating Gaussian Primitives. We compare against the scheme proposed by [29] in Fig. 7b without the additional regularization applied at modeling time. We observe that when the sculpture undergoes large deformation, the Gaussians rotated according to the deformation gradients technique from [29] exhibit obvious surface artifact. Our approach produce fewer artifacts without additional processing, nor modification to the modeling procedure.

Volume Preservation. We ablate the effect of our implicit volume preservation method in Fig. 8. For the bouncing ball scene, if the ball is simulated only using the centroids of Gaussians without densified surface and internal support particle, the ball collapses upon collision with the floor under the effect of gravity, as shown in the image with zoomed-in view in the red box. On the other hand, our infilling technique enables correct physics where the ball bounces off the ground. (Better shown in videos on the project website).

Localization. In Table. 2, we compare the 2D localization ability of feature splatting given text prompts. We use the localization dataset provided by

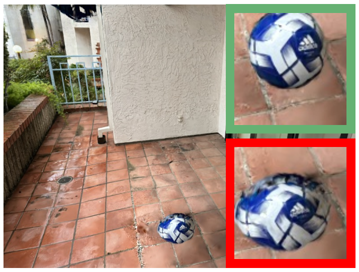


Fig. 8: Ablations of volume preservation. Without particles for internal support, elastic objects collapse and break upon collision (in red box); whereas correct simulated physics leads to a bouncing ball (in green box).

Table 3: Comparison of rendering quality and accuracy of rendered feature maps of feature splatting on collected datasets with 10% holdout views. Conceptually, feature splatting does not interfere with color rendering. $\mathcal{L}_{\text{CTRAIN}}$ indicates CLIP feature loss for training views, $\mathcal{L}_{\text{CVAL}}$ for validation views.

Method	PSNR↑	$\mathcal{L}_{\text{CTRAIN}}$	$\mathcal{L}_{\text{CVAL}}$ ↓	FPS
Vanilla GS [12]	26.45	N/A	N/A	129
Ours w/o DINO	26.48	**0.029**	0.048	102
Ours	26.47	0.032	**0.046**	102

LERF [13] and follow the evaluation protocol in LERF [13]. On the 2D setting, our full method achieves better accuracy than LERF, whereas removing DINOv2 [21] or SAM [14] reduces the accuracy, validating the need to fuse features. This is consistent with our findings in Table. 3. The performance drop when moving from 2D to 3D is expected because the scene includes many more candidates occluded from a single view. This introduces false positives not covered by the labels.

5 Conclusion

Limitation. One limitation of feature splatting is artifacts in the background that may arise after object removal or displacement. Though this is a trivial extension from existing works [5], feature splatting currently does not perform inpainting after object removal, which may result in artifacts under certain circumstances.

In conclusion, we proposed feature splatting that enables language-guided interaction for Gaussian splatting. We demonstrate the success and effectiveness of feature splatting for physic simulations, geometry, and appearance editing.

References

1. Barron, J.T., Mildenhall, B., Tancik, M., Hedman, P., Martin-Brualla, R., Srinivasan, P.P.: Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In: ICCV (2021)
2. Barron, J.T., Mildenhall, B., Verbin, D., Srinivasan, P.P., Hedman, P.: Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In: CVPR (2022)
3. Brooks, T., Holynski, A., Efros, A.A.: Instructpix2pix: Learning to follow image editing instructions. In: CVPR (2023)

4. Caron, M., Touvron, H., Misra, I., Jegou, H., Mairal, J., Bojanowski, P., Joulin, A.: Emerging properties in self-supervised vision transformers. In: ICCV (2021)
5. Chen, Y., Chen, Z., Zhang, C., Wang, F., Yang, X., Wang, Y., Cai, Z., Yang, L., Liu, H., Lin, G.: Gaussianeditor: Swift and controllable 3d editing with gaussian splatting. arXiv preprint arXiv:2311.14521 (2023)
6. Fischler, M.A., Bolles, R.C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM (1981)
7. Fridovich-Keil, S., Yu, A., Tancik, M., Chen, Q., Recht, B., Kanazawa, A.: Plenoxels: Radiance fields without neural networks. In: CVPR (2022)
8. Haque, A., Tancik, M., Efros, A.A., Holynski, A., Kanazawa, A.: Instruct-nerf2nerf: Editing 3d scenes with instructions. In: ICCV (2023)
9. Hedman, P., Philip, J., Price, T., Frahm, J.M., Drettakis, G., Brostow, G.: Deep blending for free-viewpoint image-based rendering. ACM Transactions on Graphics (ToG) (2018)
10. Hu, Y., Anderson, L., Li, T.M., Sun, Q., Carr, N., Ragan-Kelley, J., Durand, F.: Difftaichi: Differentiable programming for physical simulation. In: ICLR (2019)
11. Jambon, C., Kerbl, B., Kopanas, G., Diolatzis, S., Drettakis, G., Leimkühler, T.: Nerfshop: Interactive editing of neural radiance fields. Proceedings of the ACM on Computer Graphics and Interactive Techniques (2023)
12. Kerbl, B., Kopanas, G., Leimkühler, T., Drettakis, G.: 3d gaussian splatting for real-time radiance field rendering. ACM Transactions on Graphics (ToG) (2023)
13. Kerr, J., Kim, C.M., Goldberg, K., Kanazawa, A., Tancik, M.: Lerf: Language embedded radiance fields. In: ICCV (2023)
14. Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., Xiao, T., Whitehead, S., Berg, A.C., Lo, W.Y., et al.: Segment anything. arXiv preprint arXiv:2304.02643 (2023)
15. Kobayashi, S., Matsumoto, E., Sitzmann, V.: Decomposing nerf for editing via feature field distillation. NeurIPS (2022)
16. Li, Y., Lin, Z.H., Forsyth, D., Huang, J.B., Wang, S.: Climatenerf: Extreme weather synthesis in neural radiance field. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 3227–3238 (2023)
17. Luiten, J., Kopanas, G., Leibe, B., Ramanan, D.: Dynamic 3d gaussians: Tracking by persistent dynamic view synthesis. arXiv preprint arXiv:2308.09713 (2023)
18. Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., Ng, R.: Nerf: Representing scenes as neural radiance fields for view synthesis. In: ECCV (2020)
19. Minderer, M., Gritsenko, A., Stone, A., Neumann, M., Weissenborn, D., Dosovitskiy, A., Mahendran, A., Arnab, A., Dehghani, M., Shen, Z., et al.: Simple open-vocabulary object detection. In: ECCV (2022)
20. Müller, T., Evans, A., Schied, C., Keller, A.: Instant neural graphics primitives with a multiresolution hash encoding. ACM Transactions on Graphics (ToG) (2022)
21. Oquab, M., Darcet, T., Moutakanni, T., Vo, H., Szafraniec, M., Khalidov, V., Fernandez, P., Haziza, D., Massa, F., El-Nouby, A., et al.: Dinov2: Learning robust visual features without supervision. arXiv preprint arXiv:2304.07193 (2023)
22. Radford, A., Kim, J.W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al.: Learning transferable visual models from natural language supervision. In: ICML. PMLR (2021)
23. Schonberger, J.L., Frahm, J.M.: Structure-from-motion revisited. In: CVPR (2016)
24. Schönberger, J.L., Zheng, E., Pollefeys, M., Frahm, J.M.: Pixelwise view selection for unstructured multi-view stereo. In: ECCV (2016)

25. Shi, J.C., Wang, M., Duan, H.B., Guan, S.H.: Language embedded 3d gaussians for open-vocabulary scene understanding. arXiv preprint arXiv:2311.18482 (2023)
26. Sulsky, D., Chen, Z., Schreyer, H.L.: A particle method for history-dependent materials. Computer methods in applied mechanics and engineering (1994)
27. Tang, J., Ren, J., Zhou, H., Liu, Z., Zeng, G.: Dreamgaussian: Generative gaussian splatting for efficient 3d content creation. arXiv preprint arXiv:2309.16653 (2023)
28. Wang, P., Liu, Y., Chen, Z., Liu, L., Liu, Z., Komura, T., Theobalt, C., Wang, W.: F2-nerf: Fast neural radiance field training with free camera trajectories. In: CVPR (2023)
29. Xie, T., Zong, Z., Qiu, Y., Li, X., Feng, Y., Yang, Y., Jiang, C.: Physgaussian: Physics-integrated 3d gaussians for generative dynamics. arXiv preprint arXiv:2311.12198 (2023)
30. Ye, M., Danelljan, M., Yu, F., Ke, L.: Gaussian grouping: Segment and edit anything in 3d scenes. arXiv preprint arXiv:2312.00732 (2023)
31. Yifan, W., Serena, F., Wu, S., Öztireli, C., Sorkine-Hornung, O.: Differentiable surface splatting for point-based geometry processing. ACM Transactions on Graphics (TOG) (2019)
32. Zhou, S., Chang, H., Jiang, S., Fan, Z., Zhu, Z., Xu, D., Chari, P., You, S., Wang, Z., Kadambi, A.: Feature 3dgs: Supercharging 3d gaussian splatting to enable distilled feature fields. arXiv preprint arXiv:2312.03203 (2023)
33. Zwicker, M., Pfister, H., van Baar, J., Gross, M.: Surface splatting. In: Proceedings of the 28th annual conference on Computer graphics and interactive techniques. ACM, New York, NY, USA (Aug 2001)

A Language-Driven Appearance Editing via CLIP

We apply classifier guidance and directly optimize the appearance of select objects using gradients from the vision model in CLIP. We start with a batch of rendered 2D images. Then to localize parts of the scene that we would like to change, we build a mask m by computing pixel-wise scores. We compute the gradient for each pixel via the cosine similarity loss between the embedding of the current view I and the text query L:

$$\mathcal{L}_{\mathrm{CLIP}}(I, L) = 1 - \langle \mathrm{emb}_{\mathrm{img}}(I), \mathrm{emb}_{\mathrm{text}}(t) \rangle. \tag{6}$$

A notable benefit that Feature Splatting has over NeRF-based edits, is that local changes won't affect the entire scene. Distilled Feature Fields, for instance, require joint supervision with the original RGB images during editing, because modification of the neural network tends to affect the scene in a global manner. Therefore, conceptually, feature splatting converges faster for local edits since it updates only local parts of the scene.

B Implementation Details

B.1 Systems Considerations

Naive attempts at storing additional high-dimension semantic features in 3D Gaussians lead to a dramatic slowdown in the modeling stage [17]. Upon close inspection, we discovered that a sub-optimal memory access pattern is the bottleneck. The original Gaussian splatting code base writes the gradients directly to the global GPU DRAM during the backward pass. This parasitic effect is especially pronounced when the feature dimension is large because multiple gradient paths compete for limited access to the same DRAM slot.

Gradient Buffer in the L1 Cache. We resolve this problem by introducing a gradient buffer in the GPU L1 cache. We divide the images into 16 x 16 tiles. Each buffer has a size of 256, which means that each pixel can access this cache in an interleaved fashion. This significantly reduces the number of global memory access to the DRAM.

FP16 Tensors and Half2 Arithmetics. We replace the standard FP32 tensors with half-precision FP16 tensors, reducing the feature memory usage by half. We further improve by using the Half2 intrinsic functions. This CUDA compiler feature 'squeezes' two FP16 numbers into a single FP32 CUDA register, which further improves memory and arithmetic utilization.

We provide ablations on each of these techniques in the experiment. We plan to release our code as a reference implementation to facilitate future research that requires rasterizing any additional properties with Gaussian splatting.

B.2 Staged Feature Splatting Pipeline

The original Gaussian splatting trains for 30,000 iterations to capture all fine-grained texture details of the scene. However, for scene editing purposes, it is often redundant to optimize for such a long period for features. We empirically find that the semantics of large objects quickly emerge after a few hundred training iterations, and part-level semantics are well captured after a few thousand iterations. Therefore, to both improve the efficiency of feature splatting and to regularize features, we optimize features only for N_{feat} ($N_{feat} = 2,500$). For new Gaussians that are cloned/split from existing ones during adaptive densification, we copy the features of the source Gaussians to the new Gaussians.

B.3 Post-processing for Scene Decomposition

Thanks to the explicit representation, we present two optional post-processing steps to reduce false positives and false negatives for this selection process, which is comparable to the *closing* and *opening* morphological operations. To avoid false negatives on object boundary, we apply a KNN operation to select non-selected Gaussians whose neighbors are mostly foreground objects. To filter out false positive noises, which are common on the boundary of the scene (such as the sky) where Gaussians are not well-reconstructed, we apply a clustering step using DBSCAN to filter out scattered Gaussians.

In addition, optionally, the user can include additional object-specific negative vocabularies for better differentiation. If such vocabularies are provided, we simply carry out the open-vocabulary segmentation described in the main paper for the additional negative vocabularies, and perform a set subtract to remove Gaussians selected by the negative vocabularies from the selected foreground Gaussians.