Keep your ARMS and LEGS: Assuaging VRAM and Training Speed constraints in Language Embedded 3D Gaussian Splats

Benjamin Xia Stone Yang Jiayin Meng Inan Xu

UC San Diego

Abstract

3D Gaussian Splatting (3DGS) provides real-time photorealistic representations of captured scenes, widely reappropriated as the backbone representation for geometric and semantic processing. Of the many methods out there, Foundational Model Embedded Gaussian Splatting (FMGS) incorporates 3DGS with vision-language embeddings from CLIP and DINO models for open vocabulary scene understanding. We propose a modified architecture for FMGS where the CLIP and DINO models take the rendered scene as input, rather than the multi-resolution hash encoding. This reduces the memory footprint and training time on a variety of scenes while maintaining accuracy and performance. Our modifications are publicly available as a fork of FMGS: `github.com/stonecodecs/foundation-model-embedded-3dgs`.

1. Introduction

Recent advancements in foundation models has motivated better 3D scene understanding. Traditional 3D scene understanding was constrained to closed vocabulary sets [3] or focused solely on either segmentation or object detection. These constraints are challenges for integration with real-world environments. Foundation models are trained on large datasets with impressive capability in few-shot learning, enabling better generalization. Flamingo, proposed by Alayrac et al. [1], demonstrates impressive performance on open-vocabulary understanding for images. Inspired by this, Zuo et al. [10] introduced Foundation Model Embedded Gaussian Splatting (FMGS), achieving strong performance on open-vocabulary object detection for 3D scenes.

However, one limitation of these foundation models in 3D is their enormous memory footprint. For instance, FMGS requires up to 24GB of GPU VRAM for training which exceeds the maximum capability of most GPUs. This is due to the need to compute and store high-dimensional feature predictions from CLIP and DINO for each Gaussian during training from the multi-resolution hash encoding (MHE) representations, which results in a large memory footprint. To make these foundation models more accessible and easier to train, we aim to reduce memory cost of such models during training while preserving performance.

Our key contributions can be summarized as follows:

- We modified the original FMGS architecture to instead directly render the multi-resolution encodings (MHE) rather than pass the MHE as inputs to CLIP and DINO.
- We utilized parameter-efficient CNNs to produce the feature maps and compare them against CLIP and DINO during training.

2. Related Work

We briefly review the components involved in FMGS: 3D scene representation, scene understanding, and vision-language models.

2.1. 3D Scene Representation

There are several popular methods for representing 3D scenes. One method is to use neural radiance field representations (NeRFs) [6], which leverage neural networks to map each point's spatial location and viewing direction to a volume density and radiance value. However, one limitations of NeRFs is their slow training and inference times due to the high number of samples per ray during raymarching. Instant-NGP proposed a multi-resolution hash encoding (MHE) structure to accelerate training of NeRFs. This MHE simultaneously treats coarse regions of the image as a 1:1 mapping of values and dense regions as a hash table for constant lookup time, done in such a way that is easily parallelizable on GPUs.

3D Gaussian Splatting (3DGS) proposed representing points in the scene as gaussians, then optimizing each gaussian's parameters using differentiable rasterization. They achieved significantly faster rendering time compared to NeRFs while maintaining quality. However, downstream tasks utilizing 3DGS may encounter memory limitations

given that feature vectors are typically built per point in the scene, and the set of points grows large in size for complex scenes.

2.2. 3D Scene Understanding

Scene understanding aims to extend image or 3D representations with contextual information. One area of scene understanding is object detection, often either described as closed or open vocabulary object detection. Traditional methods for closed vocabulary object detection operate on 2D images. R-CNN [4] trains a deep convolutional neural network to extract features and classify extracted regions in the image. More recent methods have leverage transformers. Minderer et al. [7] proposed OWL-ViT, a simple open vocabulary object detection model using vision transformers. This significantly improved adaptability to real-world environments.

Open vocabulary scene understanding is strongly desired for its better performance in practical applications. Caron et al. [2] proposed a self-supervised distillation method using transformers without labels, showcasing that vision transformers can understand scenes. Transformers are better suited for open vocabulary tasks than deep convolutional networks because they generalize from large datasets while also performing well on finetuned target datasets. In object detection, Contrastive Language-Image Pretraining (CLIP) [8] simultaneously trains an image encoder and text encoder and predicts correct pairings for effective open vocabulary object detection in 2D images. Building on this, PointCLIP [9] generalized to 3D scenes by aligning CLIP encodings in a 3D point cloud with 3D category texts.

2.3. Foundation Models for Scene Understanding

Incorporating language into 3D scenes is strongly desired for various downstream tasks. Combining contributions from NeRF and CLIP, Language Embedded Radiance Fields (LERF) [5] renders CLIP encodings alongside training rays in NeRFs to produce dense relevancy maps on text queries. This is particularly useful for scene understanding as it produces a relevancy map for each point in the 3D scene.

Similar to LERF, Zuo et al. [10] introduced Foundation Model Embedded Gaussian Splatting (FMGS), which achieves strong performance in open-vocabulary object detection by applying the contributions from CLIP and DINO into 3D reconstructions via gaussian splatting. FMGS leverages CLIP to capture global, multi-scale image semantics through self-supervised distillation, and employs DINO to enforce pixel-level alignment of CLIP features. Furthermore, instead of attaching feature vectors to every Gaussian, which is both computationally and memory intensive, FMGS adopts multi-resolution hash encoding (MHE) to represent the scene's language content, providing a more lightweight and efficient alternative.

3. Method

Our model is based on the existing FMGS work (Zuo et al.[10]) referenced in Section 2.3. As such, we borrow the same general pipeline. First, the model takes in a set of input RGB images $\mathbf{I} \in \mathbb{R}^{3\times H\times W}$. Then, it estimates the image poses $\pi_i, \forall i \in \mathbf{I}$ as well as the sparse pointcloud $\mathbf{P}$ of the 3D scene using COLMAP's Structure from Motion algorithm. This pointcloud is then used as an initialization for 3DGS.

After 3DGS training is complete, the model trains the multi-resolution hash encodings (MHEs) by grounding 2D CLIP embeddings into our 3D scene representation. As noted in the original FMGS paper, CLIP embeddings are global and are unsuited for pixel-aligned feature extraction; therefore, DINO feature maps, having high quality segmentation results, are used as regularizers to improve semantic locality.

Our key contribution is a modification to the original FMGS architecture. Instead of processing the multi-resolution encodings (MHE) as inputs to DINO and CLIP MLPs to produce our feature map renderings, we instead directly render the MHE to produce feature maps sent into parameter-efficient CNNs to output the final feature maps in which we compare against CLIP and DINO feature maps for supervision. Our adjusted architecture can be seen in Figure 1 (using the original FMGS architecture diagram from [10]).

3.1. MHE Feature Field

Here, we briefly go into specifics for the MHE, similar to the original FMGS paper. For every Gaussian mean (center) $\mathbf{x}$ of our 3DGS, we query the MHE grid to get a feature vector $\mathbf{q} = \text{MHE}(\mathbf{x}) \in \mathbb{R}^{192}$. We then query all spatial dimensions in the given viewpoint to render a MHE feature field $\mathbf{Q} \in \mathbb{R}^{192\times H\times W}$, where H is the height, and W is the width. We then feed this feature field $\mathbf{Q}$ into our CLIP and DINO CNNs to produce our final 512-dim CLIP feature map $\hat{\mathbf{F}}_{CLIP} = \text{CNN}_{CLIP}(\mathbf{Q})$ and 384-dim DINO feature map $\hat{\mathbf{F}}_{DINO} = \text{CNN}_{DINO}(\mathbf{Q})$. These predicted 2D feature maps are analogous to the rasterized MLP outputs in the original FMGS paper.

3.2. CLIP & DINO embeddings

From our feature field, we now have $\hat{\mathbf{F}}_{CLIP}$ and $\hat{\mathbf{F}}_{DINO}$. To supervise these predicted feature maps, we generate ground truth feature maps $\mathbf{F}_{CLIP}$ and $\mathbf{F}_{DINO}$ from our two foundational models. We also ensure that the predicted feature map is spatially consistent with the target pixels and maintains feature-level similarity along object boundaries.

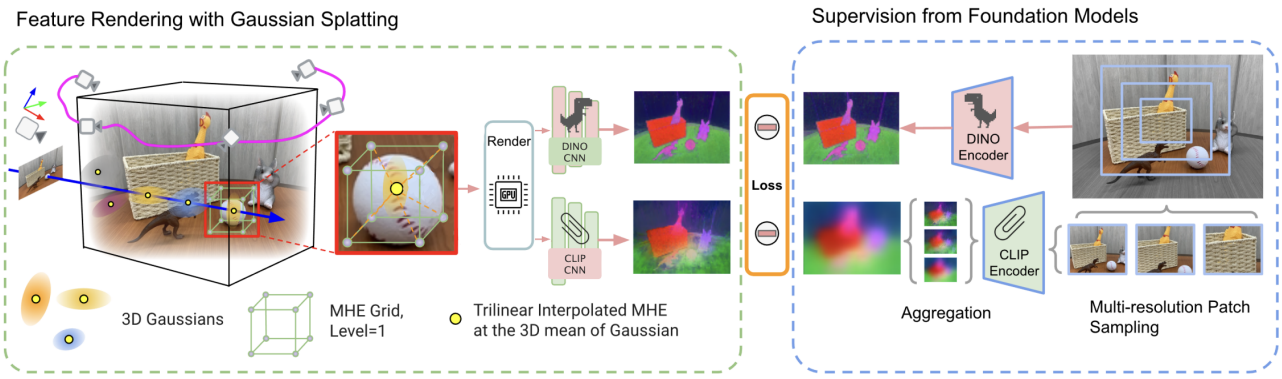

Figure 1. Our method's adjusted FMGS architecture.

Hybrid CLIP Feature Map

To effectively capture language features at multiple scales within the same image, we use the pre-trained CLIP model to construct a multi-scale pyramid of CLIP feature maps. We then upsample all lower-resolution feature maps to match the resolution of the largest one, and compute their average to generate the hybrid feature map $\mathbf{F}_{CLIP}$. We define the CLIP loss in the same way as FMGS, using the Huber loss to measure the difference between the predicted feature map and the target CLIP feature map:

$$\mathcal{L}_{CLIP} = \begin{cases} 0.5 \cdot |\hat{\mathbf{F}}_{CLIP} - \mathbf{F}|^2, & \text{if } |\hat{\mathbf{F}} - \mathbf{F}| < \delta \\ \delta \cdot \left(|\hat{\mathbf{F}} - \mathbf{F}| - 0.5 \cdot \delta \right), & \text{otherwise} \end{cases} \tag{1}$$

where δ is a hyperparameter.

DINO Feature Map

We feed the original input image into the pre-trained DINO model to generate the target DINO feature map $\mathbf{F}_{DINO}$ for regularization to improve the semantic locality. The DINO feature map loss is defined as follows:

$$\mathcal{L}_{DINO} = |\hat{\mathbf{F}}_{DINO} - \mathbf{F}_{DINO}|^2 \tag{2}$$

This regularization encourages the predicted features to be more fine-grained to capture semantically meaningful boundaries.

Pixel-alignment between Predicted Feature Fields

Like the original FMGS paper, to enforce local consistency between the predicted CLIP and DINO feature maps, we define the pixel-alignment loss using dot product similarity. Specifically, for each pixel, we compute the difference in dot products between the center and its surrounding neighbors (within a $\mathbf{K} \times \mathbf{K}$ window) across normalized CLIP and DINO embeddings. The loss is defined as:

$$\mathcal{L}_{\text{pixel}} = \frac{1}{K^2 - 1} \sum_{i \in \mathcal{P}} \sum_{\substack{j \in \mathcal{N}(i) \\ j \neq i}} \left| \hat{\mathbf{F}}_{D,i}^\top \hat{\mathbf{F}}_{D,j} - \hat{\mathbf{F}}_{C,i}^\top \hat{\mathbf{F}}_{C,j} \right|$$

We use gradient stopping on the predicted DINO features, ensuring the DINO CNN remains unaffected by this loss. This loss encourages the CLIP feature map to follow the local similarity structure of the DINO features.

Training Loss

Our final loss function is:

$$\mathcal{L}_{\text{total}} = \lambda \mathcal{L}_{CLIP} + (1 - \lambda) \mathcal{L}_{DINO} + \gamma \mathcal{L}_{\text{pixel}} \tag{3}$$

which follows the same formulation as proposed in FMGS.

3.3. Query Pipeline

Given a query view and an open-vocabulary query, our goal is to localize the query within the view. To achieve this, we generate a relevancy map that highlights the semantically relevant regions. Specifically, we first render the MHE features and pass these features through our trained CLIP CNN to obtain the predicted CLIP feature map. We also compute the CLIP embedding for the query prompt. The relevancy map is then computed by comparing the predicted CLIP features with the query embedding. We follow the same softmax based scoring strategy as FMGS, which uses canonical phrase embeddings for normalization.

The canonical phrase prompts, such as "object", "stuff", "things", and "texture", provide a semantic reference frame that allows the model to judge how uniquely a pixel is aligned with the query prompt and give higher relevancy score when the predicted feature is closer to the query embedding than the canonical phrase embeddings.

4. Experiments

We evaluated our model on open-vocabulary object detection using scenes from the LERF dataset for direct compar-

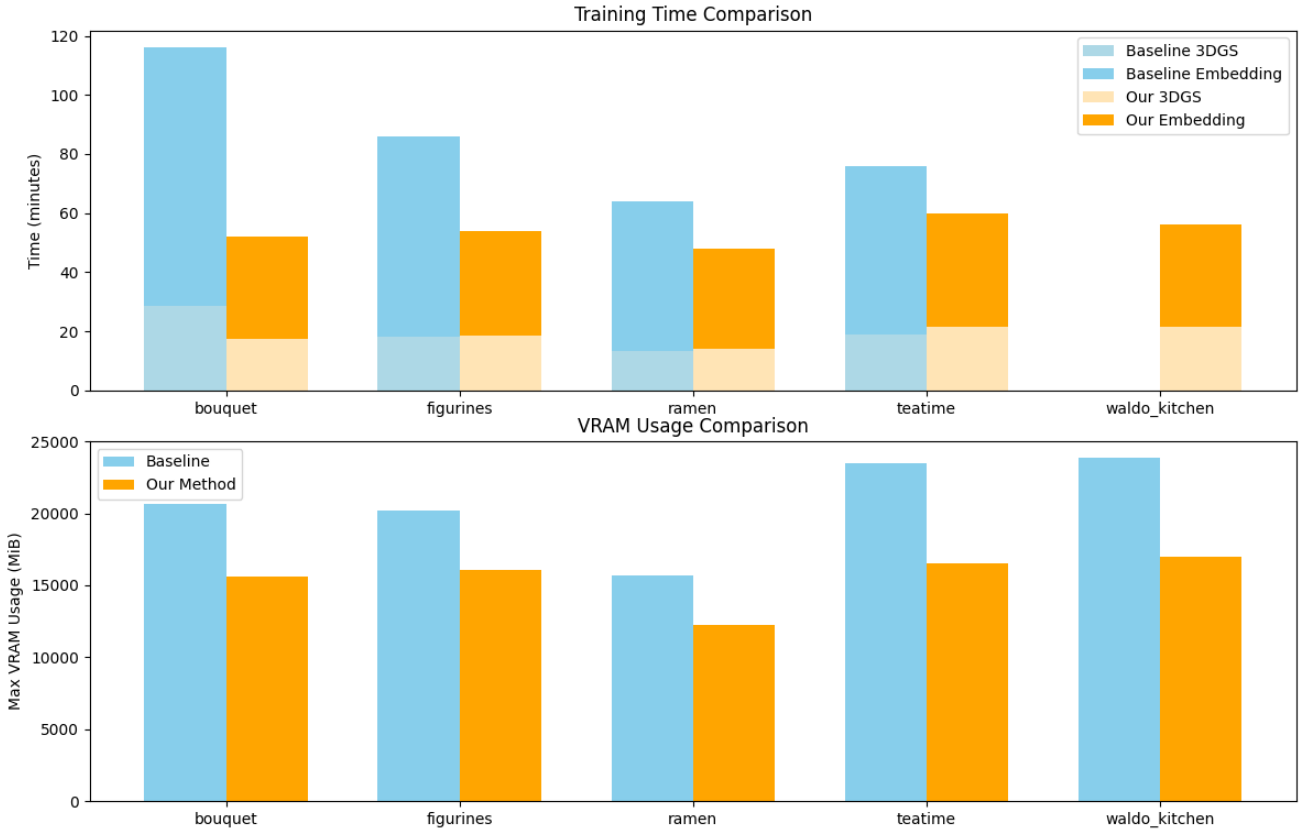

Figure 2. Runtime and VRAM usage comparison

ison with the results from FMGS. These five scenes are labeled 'bouquet', 'figurines', 'ramen', 'teatime', and 'waldo kitchen'. We show our model is competitive against FMGS, while training 37.45% faster and reducing VRAM use by 24.08%, on average.

4.1. Open-Vocabulary Object Detection

We evaluate our model's performance on object understanding using the same five scenes of the LERF dataset defined previously. Each scene has multiple (bounding box, label) ground truth pairs for each object within them, and multiple test views where the model is queried to "predict" the most relevant pixels corresponding to the queried text prompt. More precisely, we judge based on an 'accuracy' metric, where accuracy is defined as the sum of correct queries over the amount of total queries, analogous to an exam where every question is worth one point. A query is "correct" if the maximally valued pixel in the generated relevancy map is inside the GT bounding box, and "incorrect" when outside of it. The results are shown in Table 1. (best and second-best; DNF is for Did Not Finish, due to GPU out-of-memory errors.)

Note: We could not replicate the numbers they demonstrated on their paper with their demo. For this, we compare with the empirical results we perceived from the model.

Scene	LERF [5]	FMGS	Ours
bouquet	83.3%	91.7%	100.0%
figurines	87.2%	79.5 %	79.5%
ramen	62.5%	80.0 %	82.5%
teatime	96.9%	87.5 %	90.6%
kitchen	85.2%	DNF	72.2%
Average Acc.	**83.0%**	**84.68%**	**84.96%**

Table 1. Comparison of accuracy (%).

For more qualitative comparisons between FMGS and our method, Figure 4 provides insights on the discrepancy between relevancy maps generated by the two models. Red regions correspond to higher relevancy, while those closer to blue are lower. In these non-cherry-picked, randomly sampled views & queries of our four successfully rendered scenes, we notice subtle differences. One pattern is that our method usually results in slightly lower energy in high-

Figure 3. FMGS (left) and our method (right) for querying 'rubics cube' in the 'figurines' scene. Some of the background is erroneously considered relevant by FMGS, but is correctly omitted by our method. Note that the total relevancy energy of our method is lower (not as red).

relevancy regions, but is still easily distinguished over low relevancy regions. This can be beneficial in some cases, as it can lower false positive relevancies, as shown in an example from Figure 3. Other discrepancies that we perceived between the models were inconsistent variations of "smoothness" of high to low transitioning relevancy regions (demonstrated in ramen scene's 'green onion', but oppositely so in figurines' 'chairs'), and within the negative regions where one model may predict lower relevancy than the other ('opposites' can be found in figurines' 'waldo' and both teatime queries).

4.2. Computational Results

Figure 2 shows our method's computational improvements over traditional FMGS on both training runtime and VRAM usage. Table 2 and Table 3 below display these results numerically. All results were performed on a single RTX 3090 GPU with 24GB VRAM.

NOTE: 'waldo kitchen' scene crashed due to an out-of-memory error; therefore, incomplete results are indicated with a Did Not Finish (DNF) indicator.

	bouquet	fig	ramen	tea	waldo
FMGS	116	86.1	64	76	DNF
Ours	**52**	**54**	**48**	**60**	**56**

Table 2. Comparison of **training time** (min). Lower is better.

	bouquet	fig	ramen	tea	waldo
FMGS	20.685	20.165	15.687	23.473	>24
Ours	**15.583**	**16.079**	**12.275**	**16.519**	**16.993**

Table 3. Comparison of **VRAM** usage (GB). Lower is better.

Something to note is that the 'ramen' scene inherently takes less VRAM and training time than the other scenes, having the lowest absolute difference of memory savings. This may indicate that our method scales better for larger-scale scenes with many Gaussians, which is a successful result that digs into the key motivations for this report.

Downsampling Features

More experiments (omitted here) were performed regarding downsampling the feature dimensions to further reduce VRAM requirements. We modified the kernels of the DINO and CLIP CNNs with additional layers for downsampling the feature dimensions, and upsampling back for supervision. Specifically, in the DINO and CLIP CNNs, two convolutional layers with kernel size $256 \times 256 \times 3$ were added for downsampling and two ConvTranspose2D layers of the same size were added for upsampling. However, the results degraded the model's performance so significantly ($< 50\%$ accuracy) that the computation-performance trade-off would not be worthwhile.

5. Conclusion

We presented a modified architecture of Zuo et al.'s [10] Foundation Model Embedded Gaussian Splatting (FMGS). We replaced the original FMGS architecture that uses CLIP and DINO MLPs to decode the MHE into feature maps to instead directly render the MHE features into a feature field and apply CNNs to transform into CLIP and DINO feature maps for supervision. Our approach significantly reduces both training time and VRAM usage while maintaining competitive or improved accuracy compared to the original FMGS architecture. We evaluated our changes quantitatively and qualitatively on several scenes from the LERF dataset, experimenting with other memory optimizations as well. Our work provides a practical way toward scaling foundation model supervision to larger scenes and more resource-constrained environments.

References

[1] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katie Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob Menick, Sebastian Borgeaud, Andrew Brock, Aida Nematzadeh, Sahand Sharifzadeh, Mikolaj Binkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karen Simonyan. Flamingo: a visual language model for few-shot learning, 2022.

[2] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers, 2021.

[3] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes, 2017.

[4] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2014.

[5] Justin* Kerr, Chung Min* Kim, Ken Goldberg, Angjoo Kanazawa, and Matthew Tancik. Lerf: Language embedded

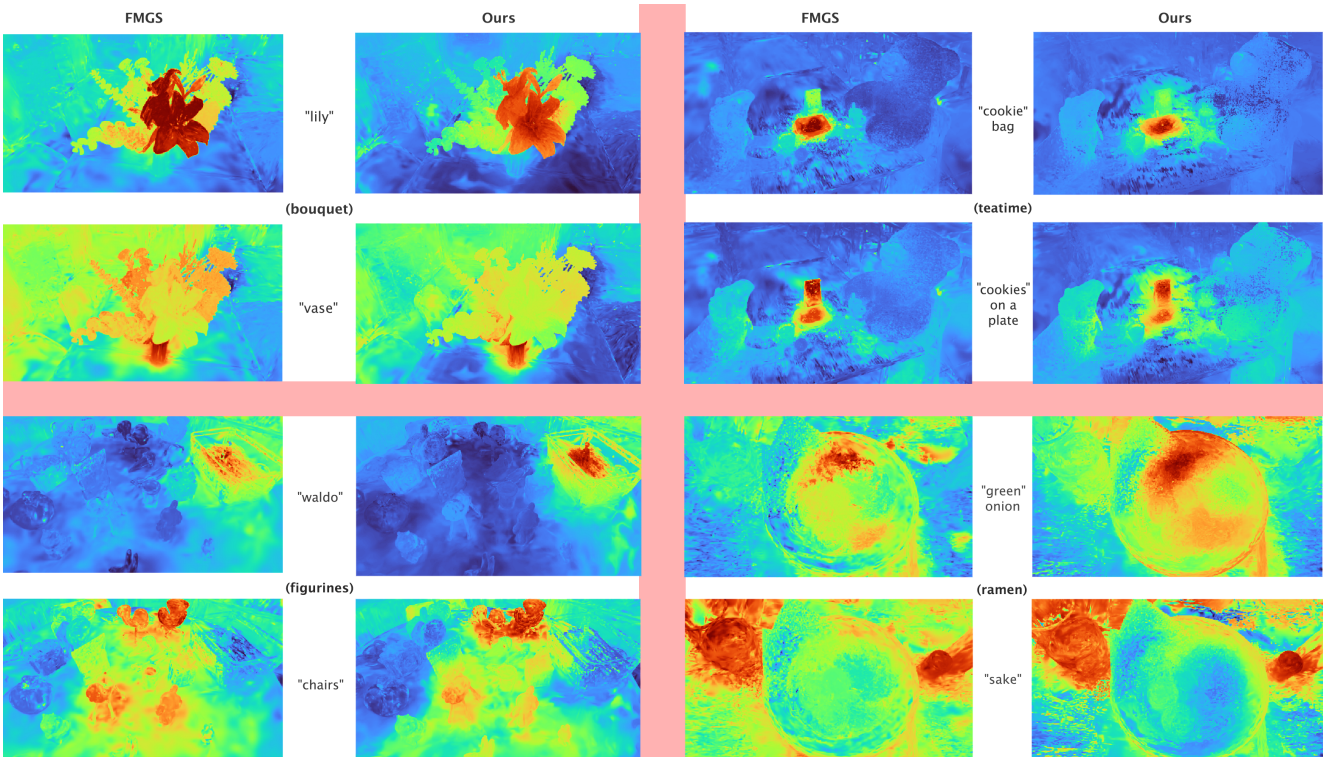

Figure 4. Relevancy maps generated by the original FMGS (left) and our approach (right).

radiance fields. In *International Conference on Computer Vision (ICCV)*, 2023.

[6] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis, 2020.

[7] Matthias Minderer, Alexey Gritsenko, Austin Stone, Maxim Neumann, Dirk Weissenborn, Alexey Dosovitskiy, Aravindh Mahendran, Anurag Arnab, Mostafa Dehghani, Zhuoran Shen, Xiao Wang, Xiaohua Zhai, Thomas Kipf, and Neil Houlsby. Simple open-vocabulary object detection with vision transformers, 2022.

[8] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021.

[9] Renrui Zhang, Ziyu Guo, Wei Zhang, Kunchang Li, Xupeng Miao, Bin Cui, Yu Qiao, Peng Gao, and Hongsheng Li. Pointclip: Point cloud understanding by clip, 2021.

[10] Xingxing Zuo, Pouya Samangouei, Yunwen Zhou, Yan Di, and Mingyang Li. Fmgs: Foundation model embedded 3d gaussian splatting for holistic 3d scene understanding. *arXiv preprint arXiv:2401.01970*, 2024.