# DeSplat: Decomposed Gaussian Splatting for Distractor-Free Rendering

## Supplementary Material

## A. Method Details

Here, we provide additional details about DeSplat. In general, DeSplat follows the same settings as Splatfacto from Nerfstudio [34], utilizing a warm-up phase of 500 steps and image downscaling factor of two at the beginning of training. We present modifications that improve the separation of static elements and distractors next by disabling opacity reset and modifying the colour representation. Additionally, we describe how we combine DeSplat with the appearance modelling in [14] and the background model in [40] which we use in the Photo Tourism experiments.

**Disabling Opacity Reset for Distractor Gaussians** Kerbl et al. [9] introduced an opacity reset mechanism in 3DGS that removes inactive Gaussian points, *e.g.* ones located close to camera views. However, in DeSplat we only apply opacity reset on static Gaussians and disable it for the distractor Gaussians, as we experimentally found that opacity reset can cause confusion between occluders and static objects.

**RGB Colour Representation for Distractor Gaussians** We modify the colour representation for the distractor Gaussians to model the colour in RGB space and use clamping to set its range instead of using a sigmoid function. Modelling the colours as RGB vectors instead of using SH coefficients alleviates the need to compensate for view-dependent effects using SH as the distractor Gaussians are per view. Additionally, applying clamping into the range $[0, 1]$ of the RGB colours instead of using a sigmoid function enabled better learning of distractors with colours at the extremes white (1) and black (0).

**Appearance Embeddings** We show that DeSplat can be combined with MLPs for modelling appearance variations with per-view image embeddings. This is commonly used for NeRF [18] and 3DGS [14, 30, 40, 41] in the Photo Tourism data set which consists of web images with varying weather conditions and lighting scenarios. We follow the appearance modelling from Kulhanek et al. [14], which uses per-image embeddings $\{\mathbf{e}_j\}_{j=1}^{N_{\text{train}}}$ with $N_{\text{train}}$ as the number of training images to handle varying appearances and illuminations, and per-Gaussian embeddings $\{\mathbf{z}_i\}_{i=1}^{N}$ to handle varying colours for each Gaussian under different appearances. The per-image embeddings $\mathbf{e}_j \in \mathbb{R}^{d_\mathbf{e}}$, per-Gaussian embeddings $\mathbf{z}_i \in \mathbb{R}^{d_\mathbf{z}}$, and the 0-th order SH coefficient $\bar{\mathbf{c}}_i$ are input to an MLP $f_{\boldsymbol{\phi}}$ as

$$(\boldsymbol{\beta}, \boldsymbol{\gamma}) = f_{\boldsymbol{\phi}}(\mathbf{e}_j, \mathbf{z}_i, \bar{\mathbf{c}}_i), \tag{6}$$

where $\boldsymbol{\beta}, \boldsymbol{\gamma} \in \mathbb{R}^3$ are the shift and scale of an affine transformation respectively, and $\boldsymbol{\phi}$ are the parameters of the MLP. The view-dependent colour of the $i$-th Gaussian $\mathbf{c}_i$ is then modulated by

$$\tilde{\mathbf{c}}_i = \boldsymbol{\gamma} \odot \mathbf{c}_i + \boldsymbol{\beta}, \tag{7}$$

where $\odot$ is an element-wise multiplication. The toned colour $\tilde{\mathbf{c}}_i$ is then passed to the rasterization. In App. C, we show the results for DeSplat on Photo Tourism where we apply the appearance modelling on the static Gaussians.

**Background Model** For background modelling, we follow the approach introduced in Splatfacto-W [40]. Instead of previous methods that use a unified colour for the background, we leverage the same per-image embeddings used for appearance modelling to predict the Spherical Harmonics (SH) coefficients $b$ for the background using an MLP. Prior to applying alpha blending, we render the background RGB colour $\mathbf{c}_{\text{BG}}$ from the predicted SH coefficients:

$$b = f_{\text{BG}}(\mathbf{e}_j). \tag{8}$$

To encourage the opacity of both distractor and static Gaussians corresponding to these areas to be low, we also apply the opacity regularization from Splatfacto-W [40] for both distractor Gaussians and static Gaussians:

$$\mathcal{L}_{\text{bg}} = \sum_{\mathbf{r} \in \mathcal{P}} |\alpha_d(\mathbf{r}) + (1 - \alpha_d(\mathbf{r})) \cdot \alpha_s(\mathbf{r}))|, \tag{9}$$

where $\alpha(\mathbf{r})$ is the per-pixel accumulation of the Gaussians at pixel $\mathbf{r}$. The set $\mathcal{P}$ is defined as $\mathcal{P} = \{(\mathbf{r}) \mid M(\mathbf{r}) > 0.6\}$.

Similar to Splatfacto-W [40], we also use a per-pixel mask $M$ to filter the area where the error of predicted background colour and ground truth $\epsilon(\mathbf{r})$ is smaller than a threshold $\mathcal{T}_\epsilon$, with a box filter $\mathcal{B}_{3\times3}$ to force the smoothness on the background

$$\tilde{e}(\mathbf{r}) = \epsilon(\mathbf{r}) \leq \mathcal{T}_\epsilon, \tag{10}$$

$$M(\mathbf{r}) = (\tilde{e}(\mathbf{r}) * \mathcal{B}_{3\times3}). \tag{11}$$

The final loss function of our method on Photo Tourism data set is:

$$\mathcal{L} = \mathcal{L}_{\text{GS}} + \lambda_d \alpha_d + \lambda_{\text{bg}} \mathcal{L}_{\text{bg}}. \tag{12}$$

## B. Additional Experimental Settings

### B.1. Data Sets

**RobustNeRF [29]** We run experiments on all five scenes *Statue*, *Android*, *Crab (1)*, *Crab (2)*, and *Yoda*. All images are downscaled $8\times$ as instructed in [29]. The scenes are indoors and are captured under different occluder settings.

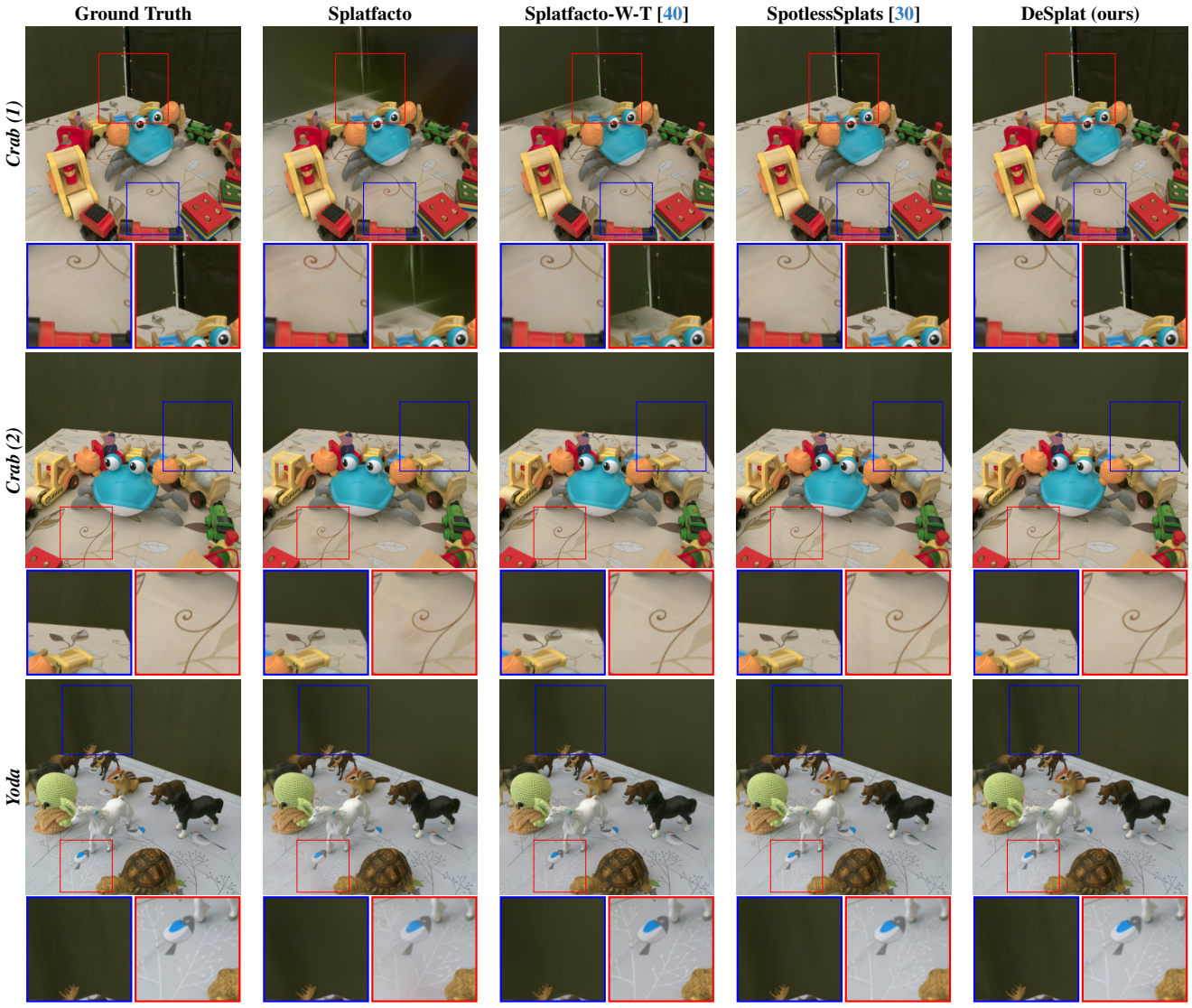


Figure A10. **Additional qualitative results on the RobustNeRF data set [29].** In the *Crab (1)*, *Crab (2)* and *Statue* scenes, DeSplat reconstructs static objects and backgrounds accurately.

The *Crab(2)* and *Yoda* scenes include both clean and cluttered image variants from the same viewpoint, enabling us to perform ablation studies by varying the ratio of clean to cluttered training images (see Fig. 9(left)).

**On-the-go [26]** We run experiments on the scenes *Mountain*, *Fountain*, *Corner*, *Patio*, *Spot*, and *Patio-High* that are commonly used for reporting quantitative performance metrics [14, 26, 30]. These scenes are further categorized according to three different occlusion rates: low (*Mountain*, *Fountain*), medium (*Corner*, *Patio*), and high (*Spot*, *Patio-High*) occlusion. All images are downscaled $8\times$, except the *Patio* scene which is downscaled $4\times$. Furthermore, we use *Arc de Triomphe* for visualization purposes (see Fig. 4).

**Photo Tourism [32]** We run experiments on the scenes *Brandenburg Gate*, *Sacre Coeur*, and *Trevi Fountain* that are commonly used for reporting quantitative performance metrics [7, 14, 40, 41, 47]. The images for these scenes are complex, exhibiting different resolutions and varying illumination and weather effects, which necessitates view-dependent appearance modelling [18]. We follow the evaluation protocol from Martin-Brualla et al. [18] where a per-image embedding is optimized on the left half of the evaluation image and then evaluated on its right half. Furthermore, we follow the test-time optimization used by Kulhanek et al. [14] to optimize the per-image embeddings for the evaluation images. Table A5 shows the reported metrics for the three scenes.

**Mip-NeRF 360 [1]** We run experiments on seven scenes

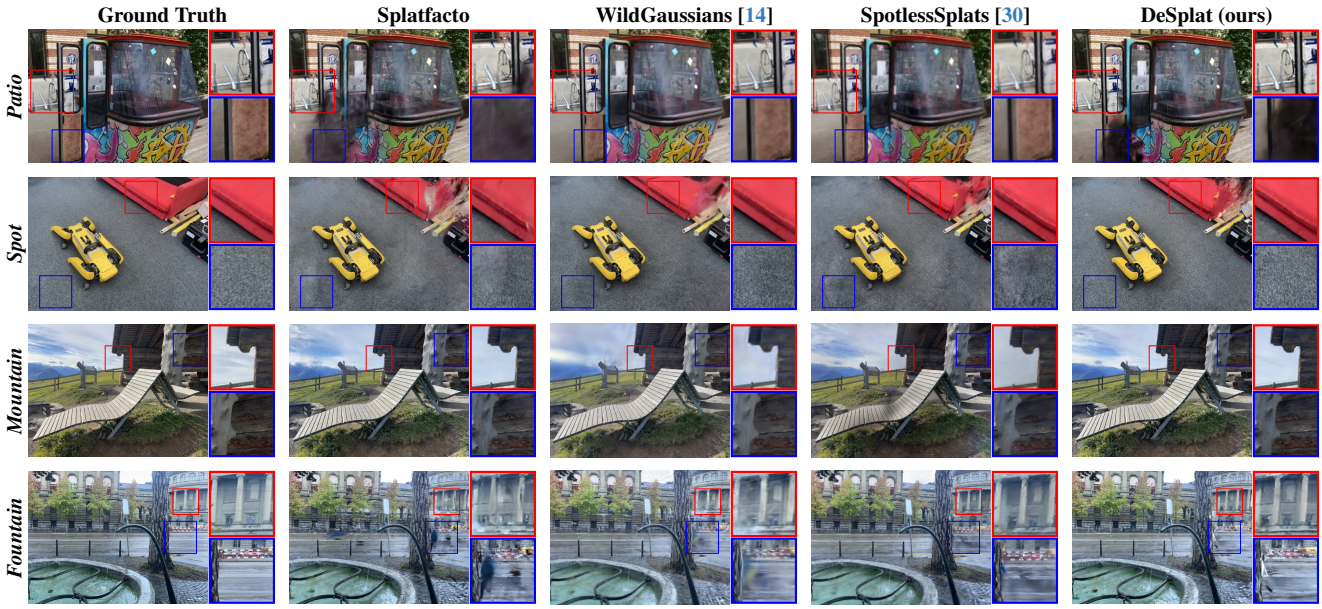


Figure A11. **Additional qualitative results on On-the-go data set [26].**

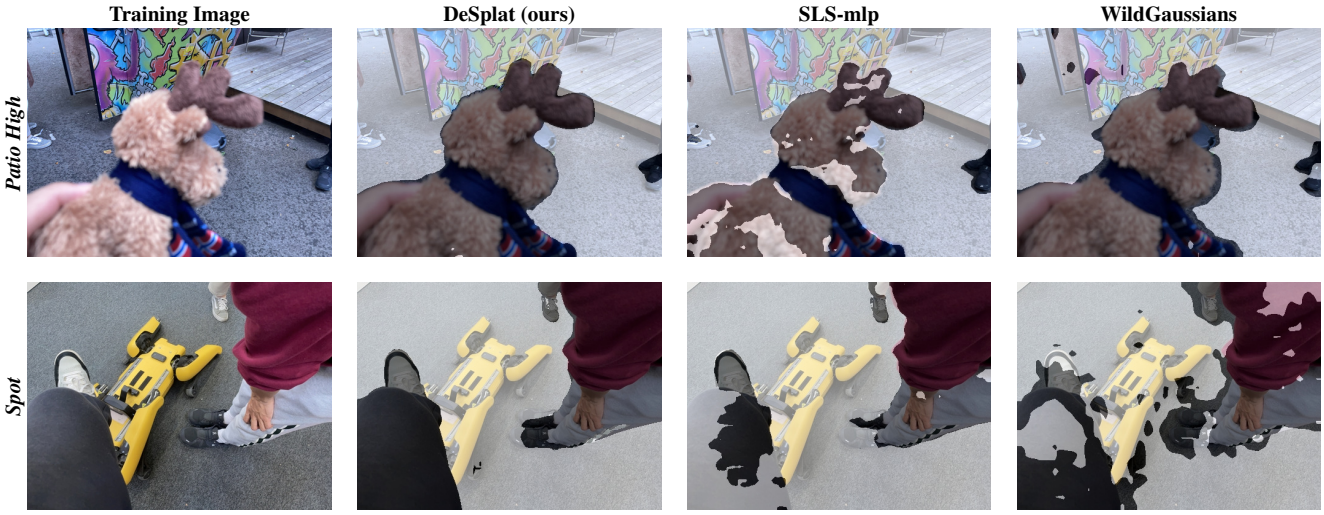


Figure A12. **Example distractor masks for different methods on On-the-go data set [26].**

*bicycle*, *bonsai*, *counter*, *garden*, *kitchen*, *room* and *stump* on Mip-NeRF 360 data set, as a comparison for performance on data sets that do not contain occluders. All indoor images are downscaled 2×, and outdoor images are downscaled 4×.

## B.2. Baselines

For all baselines, except Splatfacto and Splatfacto-W, we compare the performance of DeSplat against the PSNR, SSIM, and LPIPS metrics that were reported in their corresponding works to ensure consistency. For Splatfacto and Splatfacto-W, we run experiments for these using `gsplat` [45] version 1.0.0 and `nerfstudio` [34] version 1.1.4.

For the qualitative results, we run experiments with SpotLessSplats [30] and WildGaussians [14] to obtain visualizations (*e.g.*, see Fig. 6). We run SpotLessSplats using the reimplementation [1] in the `gsplat` library (version 1.1.1). For WildGaussians, we use the official codebase and trained model checkpoint files for evaluation and qualitative images.

## B.3. Implementation Details

We base our implementation of DeSplat on Nerfstudio codebase [34, 45] (`nerfstudio` version 1.1.4 and `gsplat` version 1.0.0). For a fair comparison of computational efficiency, all ablation experiments are conducted on a single

[1] https://github.com/lilygoli/SpotLessSplats/tree/main

Table A5. **Performance comparison between our method and the baselines on the Photo Tourism data set** [32]. The first, second, and third values are highlighted. DeSplat perform significantly better than Splatfacto, indicating that the explicit scene separation is useful for these scenes.

| Method | *Brandenburg Gate* PSNR↑ | SSIM↑ | LPIPS↓ | *Sacre Coeur* PSNR↑ | SSIM↑ | LPIPS↓ | *Trevi Fountain* PSNR↑ | SSIM↑ | LPIPS↓ |
|---|---|---|---|---|---|---|---|---|---|
| Splatfacto [34] | 19.50 | 0.885 | 0.183 | 17.15 | 0.831 | 0.210 | 17.63 | 0.696 | 0.334 |
| Splatfacto-W [40] | 26.87 | 0.932 | 0.124 | 22.53 | 0.876 | 0.158 | 22.66 | 0.769 | 0.224 |
| Splatfacto-W-A [40] | 27.50 | 0.930 | 0.130 | 22.62 | 0.876 | 0.156 | 22.81 | 0.770 | 0.228 |
| Splatfacto-W-T [40] | 26.16 | 0.925 | 0.131 | 22.78 | 0.878 | 0.155 | 22.88 | 0.772 | 0.228 |
| SWAG [7] | 26.33 | 0.929 | 0.139 | 21.16 | 0.860 | 0.185 | 23.10 | 0.815 | 0.208 |
| GS-W [47] | 27.96 | 0.932 | 0.086 | 23.24 | 0.863 | 0.130 | 22.91 | 0.801 | 0.156 |
| Wild-GS [41] | 29.65 | 0.933 | 0.095 | 24.99 | 0.878 | 0.127 | 24.45 | 0.808 | 0.162 |
| WildGaussians [14] | 27.77 | 0.927 | 0.133 | 22.56 | 0.859 | 0.177 | 23.63 | 0.766 | 0.228 |
| **DeSplat (ours) - A** | 26.72 | 0.918 | 0.132 | 22.28 | 0.876 | 0.159 | 23.06 | 0.774 | 0.229 |
| **DeSplat (ours)** | 25.04 | 0.920 | 0.142 | 20.14 | 0.868 | 0.178 | 23.31 | 0.775 | 0.226 |

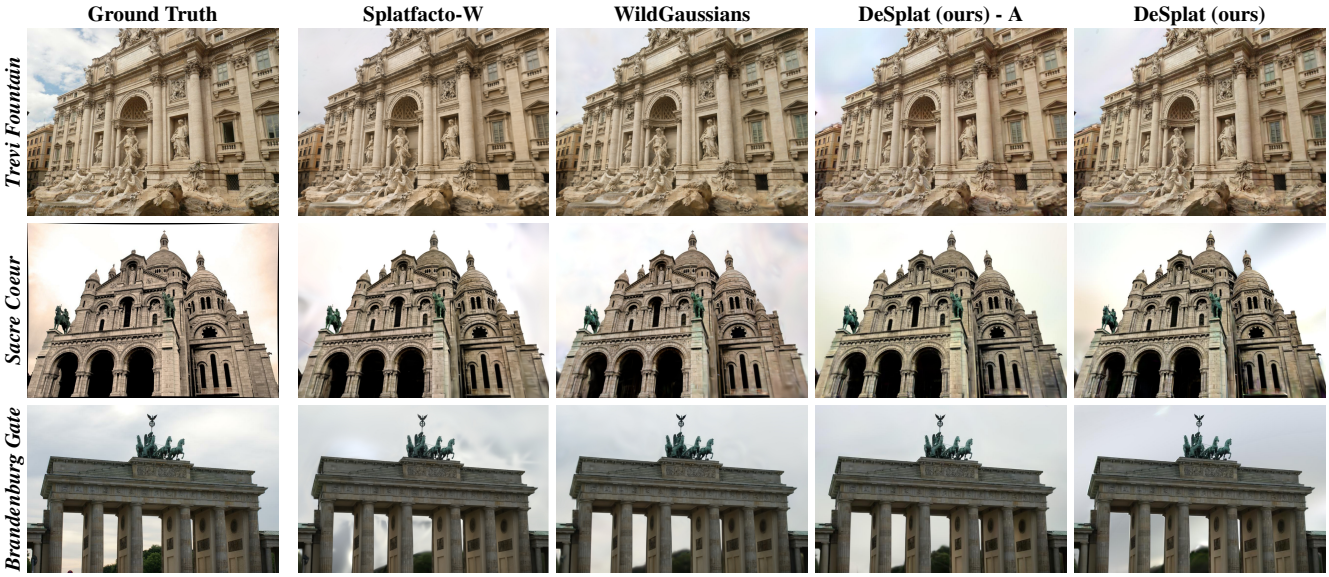


Figure A13. **Qualitative results on the Photo Tourism data set** [32]. including Our method demonstrates high rendering quality in the *Trevi Fountain*, *Brandenburg Gate*, and *Sacre Coeur* scenes.

Tesla V100 GPU. Next, we present the hyperparameters used for the data sets.

**RobustNeRF, On-the-go and Mip-NeRF 360** We use the same hyperparameter settings for DeSplat on the scenes from RobustNeRF [29], On-the-go [26] and Mip-NeRF 360 [1] data sets. More specifically, we follow the default hyperparameter setting for Splatfacto [34] and train for 30k iterations. Densification of distractor Gaussians stops after 15k iteration. We initialize 1000 distractor points in every training image (see Sec. 3.1 for details) and initialize the static points using the standard method based on the COLMAP point cloud, as employed in Splatfacto. All parameters of static and distractor points are optimized using Adam optimizer [11]. For the distractor points, we set the learning rates for the quaternions, scales and RGB colours to 0.01, 0.05 and 0.025 respectively, while the learning rates are kept as the Splatfacto default for the means and opacities. The regularization on the alpha-blending weights are set to $\lambda_d = 0.01$ and $\lambda_s = 0.01$ in the loss in Eq. (5).

**Photo Tourism** We train each scene for 200k iterations. We employ the appearance modelling from WildGaussians [14] which uses an MLP with 2 hidden layers of size 128. The per-image embedding $\mathbf{e}_j \in \mathbb{R}^{d_\mathbf{e}}$ has dimension of $d_\mathbf{e} = 32$, while the per-Gaussian embedding $\mathbf{z}_i \in \mathbb{R}^{d_\mathbf{z}}$ has dimension $d_\mathbf{z} = 24$ and is initialized using Fourier frequencies with 4 components. All parameters above are optimized using Adam optimizer [11] with learning rate 0.0005 for the MLP, 0.001 for the per-image embeddings $\mathbf{e}_j$ and 0.005 for the per-Gaussian embeddings $\mathbf{z}_i$.

For background modelling, we use the background MLP from [40] consisting of 3 hidden layers with 128 hidden units each. The Spherical Harmonics (SH) degree of the background model is set to 4. The learning rates for the background model encoder and the 0th-order SH prediction

Table A6. **Performance comparison on the Mip-NeRF 360 data set [1].**

| | outdoor | | | | | | | | | indoor | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | *bicycle* | | | *garden* | | | *stump* | | | *room* | | | *counter* | | | *kitchen* | | | *bonsai* | |
| **Method** | **PSNR↑** | **SSIM↑** | **LPIPS↓** | **PSNR↑** | **SSIM↑** | **LPIPS↓** | **PSNR↑** | **SSIM↑** | **LPIPS↓** | **PSNR↑** | **SSIM↑** | **LPIPS↓** | **PSNR↑** | **SSIM↑** | **LPIPS↓** | **PSNR↑** | **SSIM↑** | **LPIPS↓** | **PSNR↑** | **SSIM↑** | **LPIPS↓** |
| Splatfacto | 24.99 | 0.761 | 0.171 | 27.24 | 0.853 | 0.096 | 26.30 | 0.771 | 0.164 | 31.87 | 0.921 | 0.164 | 28.69 | 0.899 | 0.166 | 31.30 | 0.920 | 0.107 | 31.72 | 0.938 | 0.137 |
| **DeSplat** | 24.80 | 0.756 | 0.171 | 27.03 | 0.852 | 0.096 | 25.54 | 0.751 | 0.176 | 31.51 | 0.920 | 0.164 | 28.29 | 0.897 | 0.166 | 30.93 | 0.919 | 0.105 | 31.42 | 0.936 | 0.138 |

head are 0.002, while the learning rate for the remaining SH orders is set to 0.0001. The Adam optimizer is used, along with an Exponential decay scheduler with decay rates of 0.0001, 0.0002, and 0.00001 separately. The regularization weights are configured as follows: $\lambda_s = 0$, $\lambda_d = 0.01$, and $\lambda_{\text{bg}} = 0.15$. The threshold $\mathcal{T}_\epsilon$ for regularization is set to 0.003. All other settings are kept consistent with the parameters used for the On-the-go and RobustNerf data sets.

We also perform an ablation study using only the appearance embedding model while disabling the background model, which is denoted as DeSplat-A. In this ablation, the learning rates and MLP structure for the appearance embedding model are kept the same as described above, while the regularization weights are set as follows: $\lambda_s = 0.01$, $\lambda_d = 0.01$, and $\lambda_{\text{bg}} = 0$.

We follow the evaluation protocol [14, 18] and learn per-image appearance embeddings for test images. During evaluation, we train per-image appearance embedding for 128 iterations using the Adam optimizer with a learning rate of 0.01. All other parameters remain frozen during this period. We learn the embedding on the left half of the image, and then evaluate metrics on the right half.

# C. Additional Experimental Results

## C.1. More Qualitative Results

**Comparison on RobustNeRF data set** We present qualitative results for the *Crab (1)*, *Crab (2)*, and *Yoda* scenes from the RobustNeRF data set in Fig. A10. Our method achieves state-of-the-art results across all three scenes. However, we observe that our occluder removal does not always outperform all baselines. In some cases, where other baselines successfully remove occluders completely, our method occasionally leaves small artefacts. Despite this, our approach excels in reconstructing the background with exceptional clarity. For more details see Fig. A10.

**Comparison on On-the-go data set** We also compare our method with other baselines on the *Patio*, *Spot*, *Mountain*, and *Fountain* scenes from the On-the-go data set. In the *Patio* scene, our DeSplat does not perform well, as it fails to remove all occluders effectively. This is due to a person wearing a black jacket standing in one place for a long period, resulting in black artefacts. For detailed analysis, see the failure analysis section App. C.3. In the *Spot* scene, DeSplat produces fewer artefacts and reconstructs more detailed background information. As shown in Fig. A11, our method achieves finer reconstruction on the floor, and the wrinkles on the sofa closely resemble the ground truth. In the *Mountain* scene, our method better reconstructs the global colour and texture, especially for the wooden hut. Outside of the zoomed-in areas, white artefacts in the bottom-right corner and black artefacts on the chair, which are visible in SpotlessSplats, are not present in our results. However, our method has difficulty reconstructing the sky, as moving clouds are sometimes incorrectly identified as dynamic elements. More details can be found in C.3. In the *Fountain* scene, while none of the baselines can fully remove artefacts near the trunk, our method successfully reconstructs the column of the background architecture.

We also compare the ability of DeSplat, SpotLessSplats (using Stable Diffusion features) and WildGaussians (using DINOv2 features) to detect distractors on the On-the-go data set, as shown in Fig. A12. Since we do not use masks for training, we follow the same criteria as SLS for rendering the mask, selecting areas where the distractor's opacity is greater than 0.5 as the mask region. Despite the lack of semantic supervision, our method demonstrates a higher capability in rendering accurate masks in certain scenes compared to our baselines.

**Comparison on Photo Tourism data set** We evaluate our method on the *Brandenburg Gate*, *Sacre Coeur*, and *Trevi Fountain* scenes from the Photo Tourism data set, comparing it with the baselines. As shown in Table A5, after incorporating appearance embeddings, our DeSplat outperforms some baselines and achieves results close to the state-of-the-art. However, the metrics slightly degrade when incorporating background modelling, which aligns with the results of Splatfacto-W. Removing background Gaussians results in the loss of high-frequency details in the background, which reduces image quality. Moreover, it remains challenging to fully separate the background sky from the foreground, adding complexity to scene reconstruction. The gap between our method and the baselines is smallest for the *Trevi Fountain* scene, as in this scene, the distribution of distractors in the training images is relatively concentrated, making it easier for our method to learn dynamic elements effectively. From Fig. A13, our method produces visually appealing results with appearance embeddings, reconstructing fine details and achieving a smoother background compared to Splatfacto-W and WildGaussians.

**Comparison on Mip-NeRF 360 data set** The results on the Mip-NeRF 360 dataset demonstrate that DeSplat can be applied to scenes without distractors due to its adaptive density control of distractor points. As shown in Table A6,

our method performs comparably to Splatfacto across all scenes in the Mip-NeRF 360 dataset, with only a slight performance drop. This decline is primarily caused by the complexity of certain textures, such as sharp light variations on grass, reflections on the metal bowl in the *counter* scene, and fine details like flower petals in the *bonsai* scene. These factors make it more challenging to distinguish distractors from static objects.

### C.2. Computational Efficiency

In Table A7 and Table A8, we compare the memory usage (MB) and training time of DeSplat with Splatfacto [34], Splatfacto-W [40], SLS-mlp, SLS-mlp with utilization-based pruning (UBP) [30], and WildGaussians [14]. We also report the memory usage required to store the distractor Gaussians for DeSplat. Our rendering speed (FPS) is compared with that of Splatfacto, as reported in Table A9 using `ns-eval` in Nerfstudio [34], which calculates FPS based on both rendering time and evaluation time (including PSNR, SSIM, and LPIPS). We observe that DeSplat achieves a rendering speed comparable to Splatfacto, as DeSplat remains a purely splatting-based method. Since other baselines, such as SLS and WildGaussians, use different frameworks or rasterization libraries, it is difficult to make direct comparisons in terms of rendering speed. While DeSplat introduces a slight memory overhead compared to Splatfacto, only the static Gaussians need to be retained for novel view synthesis. Training time increases by a few minutes due to the additional memory usage. Finally, we observe that DeSplat achieves a rendering speed similar to Splatfacto, is more memory-efficient, and requires less training time than other distractor-free methods.

### C.3. Failure Cases

Our DeSplat performs well, particularly in scenarios with minimal changes in lighting and weather, and where occluders vary across camera views, such as in the *Yoda* and *Crab* scenes. However, its performance decreases in outdoor data sets with complex lighting and weather variations, because without the help of appearance embedding, the occluders may also explain part of the background. For example, in *Mountain* scene, the cloud in sky is not identical in every frame, leading some of the distractor Gaussians to explain the cloud and sky. Another limitation arises when occluders persist across many frames. Since our method separates distractors based on photometric inconsistencies between views, occluders that remain in the same or similar positions across multiple frames are misclassified as static objects. A notable example is the *Patio* scene. Consequently, as illustrated in Fig. A14 for reference.

## D. Future Directions

Investigating how to combine appearance modelling with the decomposed Gaussians for separation and better controllability is a key next step to improve DeSplat's applicability for large-scale 3D reconstruction tasks [32]. Since DeSplat is a pure splatting method, an interesting future direction is to incorporate semantic features from foundation models for improving the separation between static objects and distractors, which has shown to effectively remove distractors in 3DGS [14, 30]. Additionally, informing the initialization step of the distractor Gaussians with plausible spatial locations and shapes of the occluders, and assigning Gaussian sets per occluder could potentially yield a more fine-grained scene decomposition. Finally, extending DeSplat to handle dynamics in videos is an interesting future direction, since DeSplat may confuse distractors as being static if the distractor do not move significantly across the views which is typical for videos. We hope that this work can spur more research for explicit 3D scene decomposition based on 3DGS.

Table A7. **Comparison of memory size.** Comparison of memory size (MB) for DeSplat, Splatfacto, Splatfacto-W, SLS-mlp, and WildGaussians on the RobustNeRF [29] and On-the-go [26] data sets. To ensure consistency, we use Splatfacto-W-light with all functions enabled.

| Method | RobustNeRF | | | | | On-the-go | | | | | | Average |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Statue | Android | Crab(1) | Crab(2) | Baby Yoda | Mountain | Fountain | Corner | Patio | Spot | Patio-High | |
| Splatfacto | 38.19 | 51.95 | 25.38 | 26.86 | 24.19 | 123.63 | 145.82 | 67.45 | 73.81 | 70.76 | 106.83 | 68.62 |
| Splatfacto-W* | 42.86 | 53.49 | 40.13 | 37.57 | 23.95 | 139.90 | 165.23 | 64.01 | 74.81 | 85.04 | 114.11 | 76.46 |
| SLS-mlp | 110.14 | 129.38 | 34.41 | 47.40 | 54.50 | 108.85 | 264.34 | 142.25 | 128.09 | 107.20 | 192.79 | 119.94 |
| SLS-mlp (+UBP) | 42.24 | 56.79 | 15.11 | 18.90 | 23.59 | 53.48 | 109.61 | 45.70 | 58.06 | 39.56 | 60.45 | 47.59 |
| WildGaussians | - | - | - | - | - | 215.64 | 480.52 | 232.98 | 140.14 | 124.37 | 230.31 | 237.33 |
| **DeSplat (static)** | 46.99 | 67.80 | 35.89 | 39.40 | 38.75 | 118.27 | 163.73 | 67.03 | 55.69 | 61.28 | 102.52 | 72.49 |
| **DeSplat (distractors)** | 16.82 | 18.15 | 6.35 | 7.23 | 11.24 | 23.26 | 12.51 | 21.03 | 38.51 | 40.33 | 35.65 | 21.01 |

Table A8. **Comparison of training time.** Comparison of training time for DeSplat, Splatfacto, Splatfacto-W, SLS-mlp, and WildGaussians on the RobustNeRF [29] and On-the-go [26] data sets. To ensure consistency, we use Splatfacto-W-light with all functions enabled.

| Method | RobustNeRF | | | | | On-the-go | | | | | | Average |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Statue | Android | Crab(1) | Crab(2) | Baby Yoda | Mountain | Fountain | Corner | Patio | Spot | Patio-High | |
| Splatfacto | 9:32 | 11:42 | 9:02 | 10:43 | 10:17 | 9:46 | 10:34 | 8:56 | 8:55 | 9:02 | 11:11 | 9:58 |
| Splatfacto-W* | 24:25 | 24:08 | 23:44 | 25:26 | 25:24 | 24:13 | 24:53 | 15:59 | 17:38 | 19:05 | 20:44 | 22:19 |
| SLS-mlp | 25:21 | 24:52 | 21:57 | 25:33 | 25:02 | 27:39 | 29:06 | 22:47 | 19:31 | 22:16 | 21:41 | 24:09 |
| SLS-mlp (+UBP) | 24:09 | 23:13 | 21:59 | 25:16 | 25:55 | 26:22 | 27:14 | 19:54 | 18:16 | 19:56 | 21:08 | 23:02 |
| WildGaussians | - | - | - | - | - | 58:06 | 1:20:30 | 58:41 | 52:10 | 1:00:46 | 58:49 | 1:01:30 |
| **DeSplat (ours)** | 13:40 | 13:33 | 13:03 | 14:01 | 13:26 | 11:36 | 14:20 | 10:39 | 12:11 | 12:43 | 13:10 | 12:56 |

Table A9. **Rendering Speed.** Comparison of rendering speed (FPS) between DeSplat and Splatfacto.

| Method | RobustNeRF | | | | | On-the-go | | | | | | Average |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Splatfacto | 91.04 | 72.98 | 102.85 | 103.03 | 99.83 | 102.87 | 90.32 | 101.58 | 101.60 | 89.85 | 79.13 | 94.01 |
| **DeSplat** | 104.98 | 101.86 | 109.16 | 112.02 | 114.11 | 110.16 | 89.12 | 136.13 | 108.24 | 96.73 | 101.16 | 107.61 |

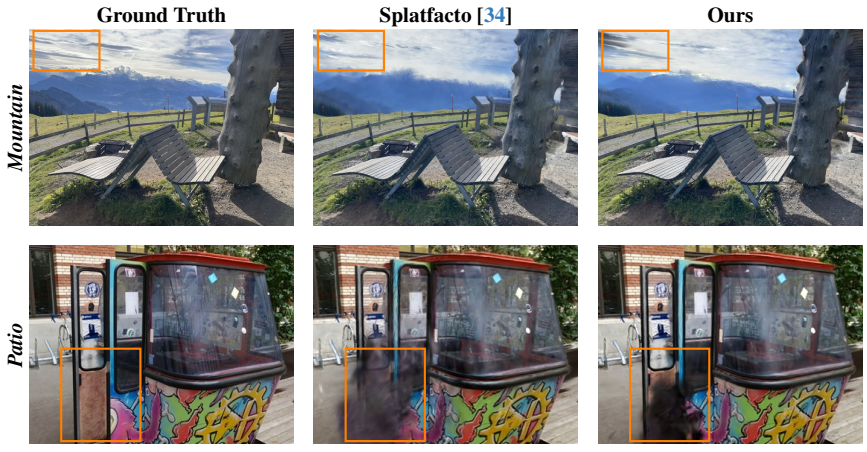


Figure A14. **Failed Cases in the *Mountain* and *Patio* Scenes.** The upper-left corner of the *Mountain* scene reveals a visible hole, while the black artifacts in the *Patio* scene appear denser compared with Splatfacto.