Journal of Machine Learning Research 26 (2025) 1-17 Submitted 9/24; Revised 1/25; Published 2/25

gsplat: An Open-Source Library for Gaussian Splatting

Vickie Ye[1,†] VYE@BERKELEY.EDU
Ruilong Li[1,†] RUILONGLI@BERKELEY.EDU
Justin Kerr[1,*] JUSTIN_KERR@BERKELEY.EDU
Matias Turkulainen[2,*] MATIAS.TURKULAINEN@AALTO.FI
Brent Yi[1,*] BRENTYI@BERKELEY.EDU
Zhuoyang Pan[3,*] PANZHY@SHANGHAITECH.EDU.CN
Otto Seiskari[4,*] OTTO.SEISKARI@SPECTACULARAI.COM
Jianbo Ye[5,*] JIANBOYE.AI@GMAIL.COM
Jeffrey Hu[*] HUJH14@GMAIL.COM
Matthew Tancik[6,††] MATT@LUMALABS.AI
Angjoo Kanazawa[1,††] KANAZAWA@EECS.BERKELEY.EDU
[1] *UC Berkeley* [2] *Aalto University* [3] *ShanghaiTech University* [4] *SpectacularAI* [5] *Amazon* [6] *Luma AI*
[†] *Project Lead,* [*] *Core Developer,* [††] *Project Mentor*

Editor: Zeyi Wen

Abstract

gsplat is an open-source library designed for training and developing Gaussian Splatting methods. It features a front-end with Python bindings compatible with the PyTorch library and a back-end with highly optimized CUDA kernels. gsplat offers numerous features that enhance the optimization of Gaussian Splatting models, which include optimization improvements for speed, memory, and convergence times. Experimental results demonstrate that gsplat achieves up to 10% less training time and 4× less memory than the original Kerbl et al. (2023) implementation. Utilized in several research projects, gsplat is actively maintained on GitHub. Source code is available at https://github.com/nerfstudio-project/gsplat under Apache License 2.0. We welcome contributions from the open-source community.

Keywords: Gaussian Splatting, 3D reconstruction, novel view synthesis, PyTorch, CUDA

1. Introduction

Gaussian Splatting, a seminal work proposed by Kerbl et al. (2023) is a rapidly developing area of research for high fidelity 3D scene reconstruction and novel view synthesis with wide interest in both academia and industry (Fei et al., 2024; Chen and Wang, 2024; Bao et al., 2024; Wu et al., 2024). It outperforms many of the previous NeRF-based (Mildenhall et al., 2020) methods in several important areas, including i) computational efficiency for training and rendering, ii) ease of editing and post-processing, and iii) deployability on hardware-constrained devices and web-based technologies. In this paper, we introduce gsplat, an open-source project built around Gaussian Splatting that aims to be an efficient and user-friendly library. The underlying concept is to enable a simple and easily modifiable API for PyTorch-based projects developing Gaussian Splatting models. gsplat supports the latest

research features and is developed with modern software engineering practices in mind. Since its initial release in October 2023, `gsplat` has garnered 67 contributors and over 2.5k stars on GitHub. `gsplat` is released under the Apache License 2.0. Documentation and further information are available on the website at:

`http://docs.gsplat.studio/`

The closest prior work implementing open-source Gaussian Splatting methods include GauStudio (Ye et al., 2024a) which consolidates various research papers into a single code repository, several PyTorch-based reproductions (Patas, 2023; Huang, 2023), and the more recent *Slang.D* (Kopanas, 2024) and *Brush* (Brussee, 2024) reimplementations using the Slang.D and Rust programming languages, respectively. Unlike previous work, `gsplat` not only seeks to implement the original 3DGS work with performance improvements, but aims to provide an easy-to-use and modular API interface allowing for external extensions and modifications, promoting further research in Gaussian Splatting. We welcome contributions from students, researchers, and the open-source community.

2. Design

`gsplat` is a standalone library developed with efficiency and modularity in mind. It is installed from PyPI on both Windows and Linux platforms, and provides a PyTorch interface. For speed considerations, many operations are programmed into optimized CUDA kernels and exposed to the developer via Python bindings. In addition, a native PyTorch implementation is also carried in `gsplat` to support iteration on new research ideas. `gsplat` is designed to provide a simple interface that can be imported from external projects, allowing easy integration of the main Gaussian Splatting functionality as well as algorithmic customization based on the latest research. With well-documented examples, test cases verifying the correctness of CUDA operations, and further documentation hosted online, `gsplat` can also serve as an education resource for new researchers entering the field.

```python
import torch
from gsplat import rasterization
# Initialize a 3D Gaussian:
mean = torch.tensor([[0.,0.,0.01]], device="cuda")
quat = torch.tensor([[1.,0.,0.,0.]], device="cuda")
color = torch.rand((1, 3), device="cuda")
opac = torch.ones((1,), device="cuda")
scale = torch.rand((1, 3), device="cuda")
view = torch.eye(4, device="cuda")[None]
K = torch.tensor([[[1., 0., 120.], [0., 1., 120.],
    [0., 0., 1.]]], device="cuda") # camera intrinsics
# Render an image using gsplat:
rgb_image, alpha, metadata = rasterization(
    mean, quat, scale, opac, color, view, K, 240,
    240)
```

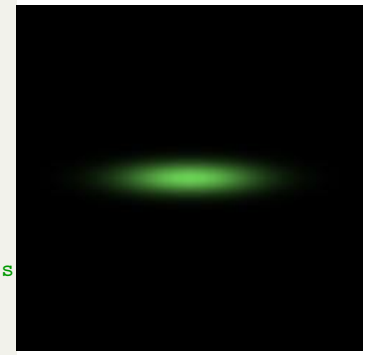

Figure 1: Implementation of the main 3D Gaussian rendering process using the `gsplat` (v1.3.0) library with only 13 lines of code. A single Gaussian is initialized (left codeblock) and rendered as an RGB image (right).

3. Features

The gsplat librarconsists of features and algorithmic implementations relating to Gaussian Splatting. With a modular interface, users can choose to enable features with simple API calls. Here, wy briefly describe some of the algorithmic enhancements provided by gsplat which are not present in the original 3DGS implementation by Kerbl et al. (2023).

Densification strategies. A key component of the Gaussian Splatting optimization procedure consists of densification and pruning of Gaussians in under- and over-reconstructed regions of the scene respectively. This has been an active area of research, and the gsplat library supports some of the latest densification strategies. These include the Adaptive Density Control (ADC) proposed by Kerbl et al. (2023), the Absgrad method proposed in Ye et al. (2024b), and the Markov Chain Monte Carlo (MCMC) method proposed in Kheradmand et al. (2024). gsplat's modular API allows users to easily change between strategies. For further details regarding densification strategies, we refer to A.1.

```
from gsplat import MCMCStrategy, rasterization
strategy = MCMCStrategy() #Initialize the strategy
strategy_state = strategy.initialize_state()
for step in range(1000): # Training loop
    render_image, render_alpha, info = rasterization(...)
    strategy.step_pre_backward(...)# Pre-backward step
    loss = ... # Compute the loss
    loss.backward() # Backward pass
    strategy.step_post_backward(...) # Post-backward step

```

Figure 2: Code-block for training a Gaussian model with a chosen densification strategy.

Pose optimization. The Gaussian rendering process (seen in Figure 1) in gsplat is fully differentiable, enabling gradient flow to Gaussian parameters $\mathcal{G}(c, \Sigma, \mu, o)$ and to other parameters such as the camera view matrices $\mathcal{P} = [\boldsymbol{R} \mid \boldsymbol{t}]$, which were not considered in the original work. This is crucial for mitigating pose uncertainty in datasets. Specifically, gradients of the reconstruction loss are computed with respect to the rotation and translation components of the camera view matrix, allowing for optimization of initial camera poses via gradient descent. More details are in A.2.

Depth rendering. Rendering depth maps from a Gaussian scene is important for applications such as regularization and meshing. gsplat supports rendering depth maps using an optimized RGB+Depth rasterizer that is also fully differentiable. gsplat supports rendering depth maps using the accumulated z-depth for each pixel and the alpha normalized expected depth. Definitions are found in A.3.

N-Dimensional rasterization. In addition to rendering three-channel RGB images, gsplat also supports rendering higher-dimensional feature vectors. This is motivated by algorithms that combine learned feature maps with differentiable volume rendering (Kobayashi et al., 2022; Kerr et al., 2023). To accommodate the storage needs of these features, the gsplat backend allows for adjustments to parameters affecting memory allocation during training, such as kernel block sizes.

Anti-aliasing. Viewing a 3D scene represented by Gaussians at varying resolutions can cause aliasing effects, as seen in prior 3D representations (Barron et al., 2021, 2022). When

the resolution decreases or the scene is viewed from afar, individual Gaussians smaller than a pixel in size produce aliasing artifacts due to sampling below the Nyquist rate. Mip-Splatting (Yu et al., 2024) proposes a low pass filter on projected 2D Gaussian covariances, ensuring a Gaussian's extent always spans a pixel. `gsplat` supports rendering with the 2D anti-aliasing mode introduced in Yu et al.. Definitions are found in A.4

4. Evaluation

Overall comparison. We compare the training performance and efficiency of `gsplat` training against the original implementation by Kerbl et al. on the MipNeRF360 dataset (Barron et al., 2022). We use the standard ADC densification strategy and equivalent configuration settings for both. We report average results on novel-view synthesis, memory usage, and training time using an A100 GPU (`PyTorch` v2.1.2 and `cudatoolkit` v11.8) at 7k and 30k training iterations in Table 1.

Table 1: Comparison of `gsplat` training performance with the original 3DGS (Kerbl et al.) implementation on the MipNeRF360 dataset. Results are averaged over 7 scenes.

	PSNR ↑	SSIM ↑	LPIPS ↓	Memory (GB) ↓	Time (min) ↓
3DGS -7K	27.23	.8290	.2041	7.7	4.64
GSPLAT -7K	27.23	.8311	.2027	**4.3**	**3.36**
3DGS -30K	28.95	.8702	.1381	9.0	26.19
GSPLAT -30K	**29.00**	**.8715**	**.1357**	5.6	19.39

We achieve the same rendering performance as the original implementation whilst using less memory and significantly reducing training time.

Feature comparison. Furthermore, we analyze the impact of features provided in the `gsplat` library in Table 2. Additional quantitative evaluations can be found in Appendix B.

Table 2: `gsplat` feature comparison on the MipNeRF360 dataset averaged over 7 scenes.

	PSNR↑	SSIM↑	LPIPS↓	Num GS	Mem (GB) ↓	Time (min) ↓
GSPLAT	29.00	.8715	.1357	3.24 M	5.62	19.39
w/ ABSGRAD	29.11	**.8778**	**.1241**	2.47 M	4.40	18.10
w/ MCMC	**29.18**	.8745	.1446	**1.00 M**	**1.98**	**15.42**
w/ ANTIALIASED	29.03	.8711	.1389	3.38 M	5.87	19.52

Acknowledgments

We thank the many open-source users for their valuable contributions to `gsplat`: fwilliams (Francis Williams), niujinshuchong (Zehao Yu), and FantasticOven2 (Weijia Zeng). This project was funded in part by NSF:CNS-2235013 and IARPA DOI/IBC No. 140D0423C0035; MT was funded by the Finnish Center for Artificial Intelligence (FCAI); JK and BY are supported by the NSF Research Fellowship Program, Grant DGE 2146752.

References

Yanqi Bao, Tianyu Ding, Jing Huo, Yaoli Liu, Yuxin Li, Wenbin Li, Yang Gao, and Jiebo Luo. 3d gaussian splatting: Survey, technologies, challenges, and opportunities. *arXiv preprint arXiv:2407.17418*, abs/2407.17418, 2024. URL https://api.semanticscholar.org/CorpusID:271404782.

Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. *International Conference on Computer Vision (ICCV)*, 2021.

Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Arthur Brussee. Brush - universal splats. https://github.com/ArthurBrussee/brush, 2024.

Guikun Chen and Wenguan Wang. A survey on 3d gaussian splatting. *arXiv preprint arXiv:2401.03890*, 2024.

Paul S. Dwyer and M. S. Macphail. Symbolic Matrix Derivatives. *The Annals of Mathematical Statistics*, 19(4):517 – 534, 1948. doi: 10.1214/aoms/1177730148. URL https://doi.org/10.1214/aoms/1177730148.

Ben Fei, Jingyi Xu, Rui Zhang, Qingyuan Zhou, Weidong Yang, and Ying He. 3d gaussian splatting as new era: A survey. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–20, 2024. doi: 10.1109/TVCG.2024.3397828.

Michael B. Giles. An extended collection of matrix derivative results for forward and reverse mode algorithmic dieren tiation. 2008. URL https://api.semanticscholar.org/CorpusID:17431500.

Binbin Huang. torch-splatting. https://github.com/hbb1/torch-splatting, 2023.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), July 2023.

Justin Kerr, Chung Min Kim, Ken Goldberg, Angjoo Kanazawa, and Matthew Tancik. Lerf: Language embedded radiance fields. In *International Conference on Computer Vision (ICCV)*, 2023.

Shakiba Kheradmand, Daniel Rebain, Gopal Sharma, Weiwei Sun, Yang-Che Tseng, Hossam Isack, Abhishek Kar, Andrea Tagliasacchi, and Kwang Moo Yi. 3d gaussian splatting as markov chain monte carlo. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Sosuke Kobayashi, Eiichi Matsumoto, and Vincent Sitzmann. Decomposing nerf for editing via feature field distillation. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL https://arxiv.org/pdf/2205.15585.pdf.

George Kopanas. Slang.d gaussian splatting rasterizer. https://github.com/google/slang-gaussian-rasterization, 2024.

Wenkai Liu, Tao Guan, Bin Zhu, Lili Ju, Zikai Song, Dan Li, Yuesong Wang, and Wei Yang. Efficientgs: Streamlining gaussian splatting for large-scale high-resolution scene representation. *arXiv preprint arXiv:2404.12777*, 2024.

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

Janusch Patas. gaussian splatting cuda. https://github.com/MrNeRF/gaussian-splatting-cuda, 2023.

K. B. Petersen and M. S. Pedersen. The matrix cookbook, nov 2012. URL http://www2.compute.dtu.dk/pubdb/pubs/3274-full.html. Version 20121115.

Tong Wu, Yu-Jie Yuan, Ling-Xiao Zhang, Jie Yang, Yan-Pei Cao, Ling-Qi Yan, and Lin Gao. Recent advances in 3d gaussian splatting. *Computational Visual Media*, 10(4): 613–642, 2024.

Chongjie Ye, Yinyu Nie, Jiahao Chang, Yuantao Chen, Yihao Zhi, and Xiaoguang Han. Gaustudio: A modular framework for 3d gaussian splatting and beyond. *arXiv preprint arXiv:2403.19632*, 2024a.

Zongxin Ye, Wenyu Li, Sidun Liu, Peng Qiao, and Yong Dou. Absgs: Recovering fine details for 3d gaussian splatting. *arXiv preprint arXiv:2404.10484*, 2024b.

Zehao Yu, Anpei Chen, Binbin Huang, Torsten Sattler, and Andreas Geiger. Mip-splatting: Alias-free 3d gaussian splatting. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

Supplementary Material

In this supplementary material we provide further details regarding the features present in the `gsplat` library in Appendix A. We give additional quantitative comparisons in Appendix B. Furthermore, we present additional details regarding the mathematical implementation of the forward pass in Appendix C and backward pass in Appendix D, which are at the core of the `gsplat` library. Lastly, we explain conventions used in the `gsplat` library in Appendix E.
`gsplat` is constantly being updated and improved. For example, recent enhancements have enabled multi-GPU training support for large-scale scene reconstruction. For most recent updates, check the commit history at `https://github.com/nerfstudio-project/gsplat`.

Mathematical Symbols and Abbreviations

We denote the camera matrix with $\mathcal{P} \in \mathrm{SE}(3)$, consisting of a rotation $\mathbf{R} \in \mathbb{R}^{\mathbf{3\times 3}}$ and translation $\mathbf{t} \in \mathbb{R}^{\mathbf{3\times 1}}$. The intrinsic matrix of the camera $\mathcal{K} \in \mathbb{R}^{3\times 3}$ includes the focal lengths (f_x, f_y) and the principal points (c_x, c_y). A Gaussian primitive in three dimensions is denoted with $\mathcal{G}$ and we refer to $\boldsymbol{\mu}, \boldsymbol{\Sigma}, o, \boldsymbol{c}$ as its mean, covariance, opacity, and color, respectively. The Gaussian center is denoted as $\boldsymbol{\mu}$ in world coordinates, $\tilde{\boldsymbol{\mu}}$ in camera coordinates, and $\boldsymbol{\mu'}$ in image space.

Appendix A. Further Details for `gsplat` Features

A.1 Densification Strategies

As of July 2024, `gsplat` supports the following densification strategies.

A.1.1 ADC

The Adaptive Density Control (ADC) method was originally proposed by Kerbl et al. (2023). During training, the positional gradients $\nabla_{\tilde{\mu}_n}\mathcal{L} = \|\frac{\partial \mathcal{L}}{\partial \tilde{\boldsymbol{\mu}}_{\boldsymbol{n}}}\|$ are tracked for a single Gaussian primitive $\mathcal{G}_n(\boldsymbol{\mu}_n, \boldsymbol{\Sigma}_n, c_n, o_n)$ and average over multiple renderings with camera views $\{\mathcal{P}\}_{k=1}^{M}$. If the accumulated positional gradients for a primitive exceed a user set threshold $\mathcal{T}$ (default is 0.0002), a Gaussian is either split or cloned. Gaussians are split if the extent of the primitive, measured by the size of the largest scale of a Gaussian, is beyond another threshold (set to 0.01); otherwise, the Gaussian is simply cloned.
The ADC system periodically culls Gaussian primitives based on their opacity values, o_n. Gaussians with opacity values below a threshold (set at 0.005) are removed at fixed intervals during training. Additionally, the ADC system periodically resets all Gaussian opacities to a small value to further encourage the culling of more Gaussians during training.

A.1.2 Absgrad

In the ADC densification strategy, the view space positional gradients for a Gaussian $\nabla_{\tilde{\mu}_n}\mathcal{L} = \sum_{k=1}^{M}(\frac{\delta \mathcal{L}_x}{\delta \tilde{\mu}_x}, \frac{\delta \mathcal{L}_y}{\delta \tilde{\mu}_y})$ are tracked across M camera views during training and a criterion for splitting and duplicating is set by a threshold. Ye et al. and Liu et al. discovered that this formulation of positional gradient accumulation can result in gradient collisions, where negative and positive view-space gradients cancel each other out, result-

ing in a poor densification heuristic. They propose to accumulate gradients using absolute sums $\nabla_{\tilde{\mu}_n}\mathcal{L} = \sum_{k=1}^{M}(|\frac{\delta\mathcal{L}_x}{\delta\tilde{\mu}_x}|, |\frac{\delta\mathcal{L}_y}{\delta\tilde{\mu}_y}|)$ instead. `gsplat` supports training with both versions of view-space accumulated gradients. The Absgrad feature is enabled with a simple API call:

```
for step in range(1000): # Training loop
    rgb_image, alpha, meta_data = rasterization(
        ...,
        absgrad = True) # Absgrad feature is enabled
    loss = ...
    loss.backward()

```

Figure 3: Training with the Absgrad view space gradients enabled.

A.1.3 MCMC

The authors in Kheradmand et al. (2024) adopt an alternative Bayesian perspective to the densification strategy in Gaussian Splatting. The authors reformulate Gaussian Splatting densification as a Stochastic Gradient Langevin Dynamic (SGLD) update rule and rewrite stochastic gradient descent updates, expressed as with $\mathcal{G} \leftarrow \mathcal{G} - \lambda_{\text{lr}} \cdot \nabla_{\mathcal{G}} \mathbb{E}_{\mathbf{I}\sim\mathcal{I}}\left[\mathcal{L}(\mathcal{G};\mathbf{I})\right]$ as SGLD updates

$$\mathcal{G} \leftarrow \mathcal{G} - \lambda_{\text{lr}} \cdot \nabla_{\mathcal{G}} \mathbb{E}_{\mathbf{I}\sim\mathcal{I}}\left[\mathcal{L}_{\text{total}}\left(\mathcal{G};\mathbf{I}\right)\right] + \lambda_{\text{noise}} \cdot \boldsymbol{\epsilon} \tag{1}$$

controlled by hyperparameters λ_{noise} and λ_{lr} and a noise term $\boldsymbol{\epsilon}$ applied to the center locations μ of Gaussians.

A.2 Pose optimization

Gradients of the reconstruction loss are computed to the rotation and translation components of a given camera view matrix using:

$$\frac{\delta\mathcal{L}}{\delta \boldsymbol{t}} = -\sum_n \frac{\delta\mathcal{L}}{\delta\tilde{\boldsymbol{\mu}}_n}, \quad \frac{\delta\mathcal{L}}{\delta \boldsymbol{R}} = -\left[\sum_n \frac{\delta\mathcal{L}}{\delta\tilde{\boldsymbol{\mu}}_n}(\boldsymbol{\mu}_n - \boldsymbol{t})^\intercal\right]\boldsymbol{R} \tag{2}$$

A.3 Depth rendering

The definitions for accumulated depth and expected depth at a pixel (x, y) are

`Accumulated depth` $$d^{acc}_{x,y} = \sum_{n=1}^{N} z_n \cdot \alpha_n \cdot T_n \tag{3}$$

`Expected depth` $$d^{exp}_{x,y} = \frac{\sum_{n=1}^{N} z_n \cdot \alpha_n \cdot T_n}{\sum_{n=1}^{N} \alpha_n \cdot T_n} \tag{4}$$

where $T_n = \prod_{j=1}^{n-1}(1-\alpha_j)$ is the accumulated transparency of depth-sorted Gaussians at pixel (x, y).

A.4 Anti-aliasing

`gsplat` supports rendering under the classic and anti-alias modes which modify the screen-space 2D gaussian sizes $\mathcal{G}^{2D}$ as follows:

`Classic mode`
$$\mathcal{G}^{2D} = o_n \cdot \exp\left(-\frac{1}{2}(\boldsymbol{p}-\boldsymbol{\mu}_n)^\intercal(\boldsymbol{\Sigma}_n^{2D}+s\cdot\mathbf{I})^{-1}(\boldsymbol{p}-\boldsymbol{\mu}_n)\right) \tag{5}$$

`Anti-alias mode`
$$\mathcal{G}^{2D} = \sqrt{\frac{|\boldsymbol{\Sigma}_n^{2D}|}{|\boldsymbol{\Sigma}_n^{2D}+s\cdot\mathbf{I}|}}\cdot o_n \cdot \exp\left(-\frac{1}{2}(\boldsymbol{p}-\boldsymbol{\mu}_n)^\intercal(\boldsymbol{\Sigma}_n^{2D}+s\cdot\mathbf{I})^{-1}(\boldsymbol{p}-\boldsymbol{\mu}_n)\right) \tag{6}$$

where s is set as a hyper-parameter during training, default is 0.3, to ensure that a 2D Gaussian's size spans the width of a single pixel.

Appendix B. Additional Evaluations

We provide additional quantitative evaluation for the various features provided in the `gsplat` library. We ablate performance using default settings, with Absgrad and MCMC densification strategies, as well as using antialiased rendering. We report per scene novel-view synthesis metrics on the MipNeRF360 dataset in Table 3, Table 4, and Table 5 as well as memory usage in Table 6.

Table 3: Per scene PSNR metrics on the MipNeRF360 dataset.

	Bicycle	**Bonsai**	**Counter**	**Garden**	**Kitchen**	**Room**	**Stump**
GSPLAT	25.29	32.21	29.01	27.39	31.37	31.23	26.51
ABSGRAD	25.44	31.98	29.07	27.47	31.65	31.43	26.71
MCMC 1 MILL	25.27	32.54	29.40	27.03	31.39	32.01	26.66
MCMC 2 MILL	25.52	32.99	29.56	27.40	31.99	32.34	26.90
MCMC 3 MILL	**25.58**	**33.13**	**29.65**	**27.65**	**32.21**	**32.40**	**26.93**
ANTIALIASED	25.31	32.27	29.01	27.33	31.34	31.53	26.44

Table 4: Per scene SSIM metrics on the MiPNeRF360 dataset.

	Bicycle	**Bonsai**	**Counter**	**Garden**	**Kitchen**	**Room**	**Stump**
GSPLAT	.7665	.9423	.9089	.8678	.9278	.9191	.7682
ABSGRAD	.7837	.9437	.9099	**.8713**	.9295	.9236	.7825
MCMC 1 MILL	.7665	.9482	.9172	.8510	.9305	.9295	.7782
MCMC 2 MILL	.7832	.9515	.9208	.8652	.9345	.9325	.7904
MCMC 3 MILL	**.7889**	**.9528**	**.9220**	.8709	**.9358**	**.9333**	**.7935**
ANTIALIASED	.7654	.9428	.9084	.8657	.9277	.9194	.7684

Appendix C. Forward Pass

A 3D Gaussian is parametrized by its mean $\boldsymbol{\mu} \in \mathbb{R}^3$, covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{3\times3}$ decomposed into a scaling vector $\boldsymbol{s} \in \mathbb{R}^3$ and a rotation quaternion $\boldsymbol{q} \in \mathbb{R}^4$, opacity $o \in \mathbb{R}$, and a feature vector $\boldsymbol{c} \in \mathbb{R}^N$. For the remainder of the derivations, we denote $\boldsymbol{c} \in \mathbb{R}^3$ as color

Table 5: Per scene LPIPS metrics on the MipNeRF360 dataset. LPIPS is computed using AlexNet features.

	Bicycle	**Bonsai**	**Counter**	**Garden**	**Kitchen**	**Room**	**Stump**
GSPLAT	.1703	.1329	.1546	.0754	.0955	.1651	.1558
ABSGRAD	**.1391**	.1267	.1499	**.0724**	.0924	.1523	.1361
MCMC 1 MILL	.1978	.1237	.1425	.1107	.0974	.1469	.1724
MCMC 2 MILL	.1662	.1166	.1335	.0915	.0905	**.1403**	.1466
MCMC 3 MILL	.1521	**.1140**	**.1292**	.0833	**.0884**	.1383	**.1369**
ANTIALIASED	.1811	.1329	.1565	.0822	.0957	.1657	.1585

Table 6: Per scene memory consumption (in GB) metrics on the MipNeRF360 dataset.

	Bicycle	**Bonsai**	**Counter**	**Garden**	**Kitchen**	**Room**	**Stump**
GSPLAT	10.47	2.41	2.36	9.89	3.16	2.84	8.20
ABSGRAD	8.75	**1.91**	2.02	6.36	2.84	2.75	6.15
MCMC 1 MILL	**1.84**	2.06	**2.02**	**1.81**	**2.05**	**2.14**	**1.82**
MCMC 2 MILL	3.21	3.51	3.57	3.18	3.51	3.84	3.17
MCMC 3 MILL	4.75	5.11	5.59	4.54	4.97	5.38	4.59
ANTIALIASED	11.30	2.41	2.34	10.10	3.17	2.81	8.97

encoded via spherical harmonics similar to the original work by Kerbl et al. (2023); however, the derivations also apply to other N-dimensional vectors. To render a view from the Gaussian scene, we compute their projected 2D means and extents in the camera plane. Visible 2D Gaussians are then sorted by depth and composited from front to back using the discrete rendering equation to construct the output image.

C.1 Projection of Gaussians

The render camera is described by its extrinsics $\mathcal{P}$, which transforms points from the world coordinate space to the camera coordinate space, and its intrinsics $\mathcal{K}$ which projects Gaussians from camera coordinates to image coordinates:

$$\mathcal{P} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}, \quad \mathcal{K} = \begin{bmatrix} f_x & 0 & cx \\ 0 & f_y & cy \\ 0 & 0 & 1 \end{bmatrix} \tag{7}$$

A visible 3D Gaussians $\mathcal{G}_n(\boldsymbol{\mu}, \boldsymbol{\Sigma}, o, \boldsymbol{c})$ in world space is mapped into camera space using:

$$\hat{\boldsymbol{\mu}}_{\boldsymbol{n}} = R^\top(\mu_n - p), \quad \hat{\boldsymbol{\Sigma}}_{\boldsymbol{n}} = R^\top \Sigma R, \quad \hat{\boldsymbol{c}}_{\boldsymbol{n}} = \texttt{SH}(\frac{\mu_n - t}{\|\mu_n - t\|}) \tag{8}$$

Furthermore, the camera coordinate Gaussian $\hat{\mathcal{G}}_n(\hat{\boldsymbol{\mu}}_{\boldsymbol{n}}, \hat{\boldsymbol{\Sigma}}_{\boldsymbol{n}}, o_n, \hat{\boldsymbol{c}}_{\boldsymbol{n}})$ projects to a image space 2D Gaussian $\hat{\mathcal{G}}^{2D}_n(\boldsymbol{\mu'}, d, \boldsymbol{\Sigma'})$ with z-depth d via:

$$d = \tilde{\mu}_z, \quad \mu' = (\tilde{\mu}_x/d, \tilde{\mu}_y/d), \quad \Sigma' = J^\top \hat{\Sigma} J \tag{9}$$

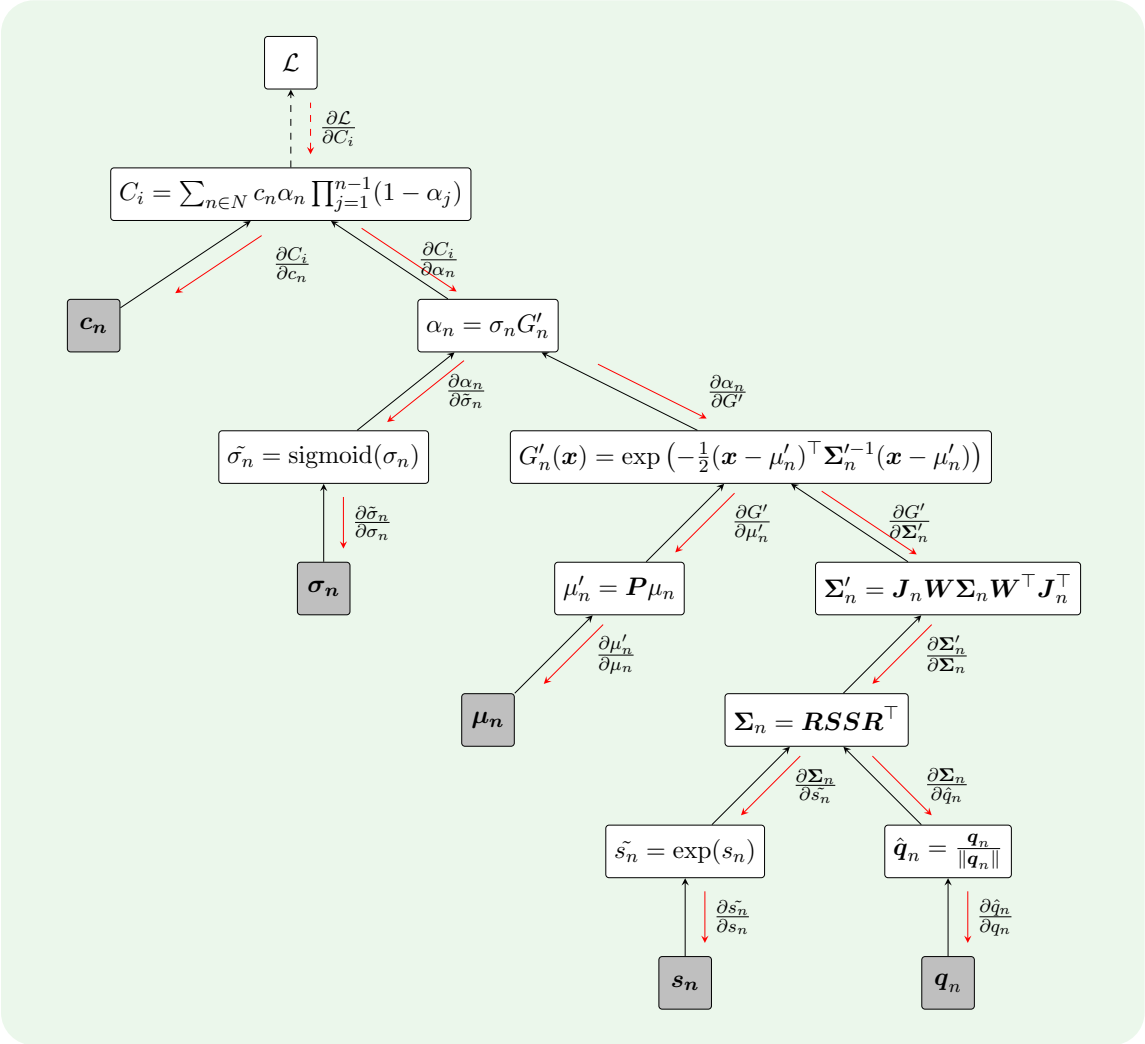

Figure 4: An illustration of the forward (Appendix C) and backward (Appendix D) computation graphs of the main `gsplat` Gaussian Splatting rendering function for Gaussian parameters $c, \sigma, \mu, s,$ and q.

We approximate the projection of camera space $\hat{\boldsymbol{\Sigma}}_{\boldsymbol{n}}$ to image space with a first-order Taylor expansion located at the pose $\mathcal{P}$. Specifically, we compute the affine transform $J \in \mathbb{R}^{2\times 3}$ as:

$$J = \frac{1}{d}\begin{bmatrix} 1 & 0 & -\tilde{\mu}_x/d \\ 0 & 1 & -\tilde{\mu}_y/d \end{bmatrix} \tag{10}$$

Note, unlike the original implementation by Kerbl et al. (2023), we do not use the OpenGL NDC coordinate system as an intermediate representation. Thus, a 2D Gaussian $\mathcal{G}_n^{2D}(\boldsymbol{\mu'}, \boldsymbol{\Sigma'}, o, \boldsymbol{c})$ is defined in image coordinates with the covariance matrix $\boldsymbol{\Sigma}' \in \mathbb{R}^{2\times 2}$:

$$\boldsymbol{\Sigma}_n^{2D} = J^\top R^\top \boldsymbol{\Sigma} R J. \tag{11}$$

We further map from image to pixel coordinates for rasterization. See Appendix E for more details.

C.2 Rasterization of Gaussians

We directly follow the tile sorting method introduced by Kerbl et al., which bins the 2D Gaussians into 16×16 tiles and sorts them per tile by depth. For each Gaussian, we compute the axis-aligned bounding box around the 99th percentile ellipse of each 2D projected covariance (3 standard deviations), and include it in a tile bin if its bounding box intersects with the tile. We then apply the tile sorting algorithm as presented in Appendix C of Kerbl et al. (2023) to get a list of Gaussians sorted by depth for each tile. We then rasterize the sorted Gaussians within each tile. For a color at a pixel $\boldsymbol{p}_{(x,y)}$, let i index the N Gaussians involved in that pixel.

$$\hat{\mathbf{C}}_{\mathbf{x},\mathbf{y}} = \sum_{n \in N} \mathbf{c}_n \alpha_n T_i, \quad \text{where } T_i = \prod_{j=1}^{i-1} (1 - \alpha_j) \tag{12}$$

We compute α_n with the 2D covariance $\mathbf{\Sigma}_n^{2D} \in \mathbb{R}^{2\times 2}$ and opacity parameters:

$$\alpha_n = o_n \cdot \exp\left(-\frac{1}{2} (\boldsymbol{p}_{(x,y)} - \boldsymbol{\mu}_n)^\intercal (\mathbf{\Sigma}_n^{2D})^{-1} (\boldsymbol{p}_{(x,y)} - \boldsymbol{\mu}_n) \right) \tag{13}$$

We compute T_i online as we iterate through the Gaussians front to back.

Appendix D. Backward Pass

D.1 Computing Gradients of Gaussians

We now compute the gradients of a scalar loss with respect to the input Gaussian parameters. That is, given the gradient of a scalar loss $\mathcal{L}$ with respect to each pixel of the output image, we propagate the gradients backward toward the original input parameters with standard chain rule mechanics. In the following we will use the Frobenius inner product in deriving the matrix derivatives Giles (2008):

$$\langle X, Y \rangle = \mathrm{Tr}(X^\top Y) = \mathrm{vec}(X)^\top \mathrm{vec}(Y) \in \mathbb{R} \tag{14}$$

and can be thought of as a matrix dot product. The Frobenius inner product has the following properties:

$$\langle X, Y \rangle = \langle Y, X \rangle \tag{15}$$

$$\langle X, Y \rangle = \langle X^\top, Y^\top \rangle, \tag{16}$$

$$\langle X, YZ \rangle = \langle Y^\top X, Z \rangle = \langle XZ^\top, Y \rangle, \tag{17}$$

$$\langle X, Y + Z \rangle = \langle X, Y \rangle + \langle X, Z \rangle \tag{18}$$

Suppose we have a scalar function f of $X \in \mathbb{R}^{m\times n}$, and that $X = AY$, with $A \in \mathbb{R}^{m\times p}$ and $Y \in \mathbb{R}^{p\times n}$. We can write the gradient of f with respect to an arbitrary scalar $\theta \in \mathbb{R}$ as

$$\frac{\partial f}{\partial \theta} = \langle \frac{\partial f}{\partial X}, \frac{\partial X}{\partial \theta} \rangle, \tag{19}$$

for which we use the shorthand

$$\partial f = \langle \frac{\partial f}{\partial X}, \partial X \rangle. \tag{20}$$

Here $\frac{\partial f}{\partial \theta} \in \mathbb{R}$, $\frac{\partial f}{\partial X} \in \mathbb{R}^{m \times n}$, and $\frac{\partial X}{\partial \theta} \in \mathbb{R}^{m \times n}$.
In this case, it is simple to continue the chain rule. Letting $G = \frac{\partial f}{\partial X}$, we have

$$\begin{aligned}
\frac{\partial f}{\partial \theta} &= \left\langle G, \frac{\partial (AY)}{\partial \theta} \right\rangle \\
&= \left\langle G, \frac{\partial A}{\partial \theta} Y \right\rangle + \left\langle G, A \frac{\partial Y}{\partial \theta} \right\rangle \\
&= \left\langle G Y^\top, \frac{\partial A}{\partial \theta} \right\rangle + \left\langle A^\top G, \frac{\partial Y}{\partial \theta} \right\rangle.
\end{aligned}$$

From here, we read out the elements of the gradients of f with respect to A and Y by letting $\theta = A_{ij}$ and $\theta = Y_{ij}$ respectively, and find that

$$\frac{\partial f}{\partial A} = G Y^\top \in \mathbb{R}^{m \times p}, \quad \frac{\partial f}{\partial Y} = A^\top G \in \mathbb{R}^{p \times n} \tag{21}$$

D.2 Depth Compositing Gradients

We start with propagating the loss gradients of a pixel i back to the Gaussians that contributed to the pixel. Specifically, for a Gaussian n that contributes to the pixel i, we compute the gradients with respect to color $\frac{\partial \mathcal{L}}{\partial c_n} \in \mathbb{R}^3$, opacity $\frac{\partial \mathcal{L}}{\partial o_n} \in \mathbb{R}$, the 2D means $\frac{\partial \mathcal{L}}{\partial \mu'_n} \in \mathbb{R}^2$, and 2D covariances $\frac{\partial \mathcal{L}}{\partial \mathbf{\Sigma}'_n} \in \mathbb{R}^{2 \times 2}$, given the $\frac{\partial \mathcal{L}}{\partial C_i} \in \mathbb{R}^3$. In the forward pass, we compute the contribution of each Gaussian to the pixel color from front to back, i.e. Gaussians in the back are downstream of those in the front. As such, in the backward pass, we compute the gradients of the Gaussians from back to front. For the color, we have

$$\frac{\partial C_i(k)}{\partial c_n(k)} = \alpha_n \cdot T_n \tag{22}$$

for each channel k. We save the final T_N value from the forward pass and compute next T_{n-1} values as we iterate backward:

$$T_{n-1} = \frac{T_n}{1 - \alpha_{n-1}} \tag{23}$$

For the α gradient, for each channel k we have the scalar gradients

$$\frac{\partial C_i(k)}{\partial \alpha_n} = c_n(k) \cdot T_n - \frac{S_n(k)}{1 - \alpha_n} \text{ where } S_n = \sum_{m > n} c_m \alpha_m T_m. \tag{24}$$

We can also compute S_{n-1} as we iterate backward over Gaussians:

$$\begin{aligned}
S_N(k) &= 0 \\
S_{n-1}(k) &= c_n(k) \alpha_n T_n + S_n(k).
\end{aligned} \tag{25}$$

For the opacity and sigma, we have scalar gradients

$$\frac{\partial \alpha_n}{\partial o_n} = \exp\left(-\sigma_n\right), \quad \frac{\partial \alpha_n}{\partial \sigma_n} = -o_n \exp\left(-\sigma_n\right) \tag{26}$$

For the 2D mean, we have the Jacobian

$$\frac{\partial \sigma_n}{\partial \mu'_n} = \frac{\partial \sigma_n}{\partial \Delta_n} = \Sigma'^{-1}_n \Delta_n \in \mathbb{R}^2 \tag{27}$$

For the 2D covariance, we let $Y = \Sigma'^{-1}_n$, which has a straightforward Jacobian from σ_n :

$$\frac{\partial \sigma_n}{\partial Y} = \frac{1}{2} \Delta_n \Delta_n^\top \in \mathbb{R}^{2\times 2}. \tag{28}$$

To continue back-propagating through $Y \in \mathbb{R}^{2\times 2}$, we let $G = \frac{\partial \sigma_n}{\partial Y}$ and write the gradients with respect to a scalar variable x as

$$\frac{\partial \sigma_n}{\partial x} = \left\langle G, \frac{\partial Y}{\partial x} \right\rangle. \tag{29}$$

We use the identity [Petersen and Pedersen (2012), Dwyer and Macphail (1948)] that $\frac{\partial Y}{\partial x} = -Y \frac{\partial \Sigma'_n}{\partial x} Y$, and have

$$\begin{aligned} \frac{\partial \sigma_n}{\partial x} &= \left\langle G, -Y \frac{\partial \Sigma'_n}{\partial x} Y \right\rangle \\ &= \left\langle -Y^\top G Y^\top, \frac{\partial \Sigma'_n}{\partial x} \right\rangle \end{aligned} \tag{30}$$

The gradient of σ_n with respect to Σ'_n is then

$$\frac{\partial \sigma_n}{\partial \Sigma'_n} = -\frac{1}{2} \Sigma'^{-1}_n \Delta_n \Delta_n^\top \Sigma'^{-1}_n \tag{31}$$

D.3 Projection Gradients

Given the gradients of $\mathcal{L}$ with respect the projected 2D mean μ' and covariance Σ' of a Gaussian, we can continue to backpropagate the gradients of its 3D means μ and covariances Σ. Here we deal only with a single Gaussian at a time, so we drop the subscript n and compute the gradients $\frac{\partial \mathcal{L}}{\partial \mu} \in \mathbb{R}^3$ and $\frac{\partial \mathcal{L}}{\partial \Sigma} \in \mathbb{R}^{3\times 3}$, given the gradients $\frac{\partial \mathcal{L}}{\partial \mu'} \in \mathbb{R}^2$ and $\frac{\partial \mathcal{L}}{\partial \Sigma'} \in \mathbb{R}^{2\times 2}$.

We first compute the gradient contribution of 2D mean μ' to camera coordinates $t \in \mathbb{R}^4$, and of 2D covariance Σ' to 3D covariance Σ and camera coordinates t. Note that both μ' and Σ' contribute to the gradient with respect to t :

$$\frac{\partial \mathcal{L}}{\partial t_i} = \frac{\partial \mathcal{L}_{\mu'}}{\partial t_i} + \frac{\partial \mathcal{L}_{\Sigma'}}{\partial t_i} = \frac{\partial \mathcal{L}}{\partial \mu'} \frac{\partial \mu'}{\partial t_i} + \left\langle \frac{\partial \mathcal{L}}{\partial \Sigma'}, \frac{\partial \Sigma'}{\partial t_i} \right\rangle \tag{32}$$

For 2D mean μ', we have the contribution to the gradient of t as:

$$\frac{\partial \mathcal{L}_{\mu'}}{\partial t} = \frac{1}{2} P^\top \begin{bmatrix} w/t_w & 0 & 0 & -w \cdot t_x/t_w^2 \\ 0 & h/t_w & 0 & -w \cdot t_y/t_w^2 \end{bmatrix}^\top \frac{\partial \mathcal{L}}{\partial \mu'}. \tag{33}$$

The 2D covariance Σ' contributes to the gradients of Σ and t. where $\Sigma' = T\Sigma T^\top$. The contribution to Σ is straightforward. Letting $G = \frac{\partial \mathcal{L}}{\partial \Sigma'}$, we have

$$\begin{aligned}\partial \mathcal{L}_{\Sigma'} &= \left\langle G, \partial \Sigma' \right\rangle \\ &= \left\langle G, (\partial T)\Sigma T^\top + T(\partial \Sigma)T^\top + T\Sigma\left(\partial T^\top\right)\right\rangle \\ &= \left\langle G T \Sigma^\top, \partial T\right\rangle + \left\langle T^\top G T, \partial \Sigma\right\rangle + \left\langle G^\top T \Sigma, \partial T\right\rangle \\ &= \left\langle G T \Sigma^\top + G^\top T \Sigma, \partial T\right\rangle + \left\langle T^\top G T, \partial \Sigma\right\rangle.\end{aligned} \tag{34}$$

We read out the gradient with respect to $\Sigma \in \mathbb{R}^{3\times 3}$ as

$$\frac{\partial \mathcal{L}}{\partial \Sigma} = T^\top \frac{\partial \mathcal{L}}{\partial \Sigma'} T. \tag{35}$$

We continue to propagate gradients through $T = JR_{\text{cw}} \in \mathbb{R}^{2\times 3}$ for $J \in \mathbb{R}^{2\times 3}$:

$$\partial \mathcal{L} = \left\langle \frac{\partial \mathcal{L}}{\partial T}, (\partial J) R_{\text{cw}} \right\rangle = \left\langle \frac{\partial \mathcal{L}}{\partial T} R_{\text{cw}}^\top, \partial J \right\rangle, \quad \text{where } \frac{\partial \mathcal{L}}{\partial T} = \frac{\partial \mathcal{L}}{\partial \Sigma'} T \Sigma^\top + \frac{\partial \mathcal{L}}{\partial \Sigma'} T \Sigma. \tag{36}$$

We continue propagating through J for camera coordinates $t \in \mathbb{R}^4$ for the contribution through Σ' to the gradients of t :

$$\frac{\partial J}{\partial t_x} = \begin{bmatrix} 0 & 0 & -f_x/t_z^2 \\ 0 & 0 & 0 \end{bmatrix}, \quad \frac{\partial J}{\partial t_y} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -f_y/t_z^2 \end{bmatrix}, \tag{37}$$

$$\frac{\partial J}{\partial t_z} = \begin{bmatrix} -f_x/t_z^2 & 0 & 2f_x t_x/t_z^3 \\ 0 & -f_y/t_z^2 & 2f_y t_y/t_z^3 \end{bmatrix}, \quad \frac{\partial J}{\partial t_w} = \mathbf{0}^{2\times 3}. \tag{38}$$

We can now sum the two gradients $\frac{\partial \mathcal{L}_{\mu'}}{\partial t}$ and $\frac{\partial \mathcal{L}_{\Gamma'}}{\partial t}$ into $G = \frac{\partial \mathcal{L}}{\partial t}$, and compute the full gradients with respect to the 3D mean μ and the view matrix T_{cw}. We have that $t = T_{\text{cw}} q$, where $q = \begin{bmatrix}\mu & 1\end{bmatrix}^\top$.

$$\begin{aligned}\partial \mathcal{L} &= \langle G, \partial t \rangle = \langle G, \partial \left(T_{\text{cw}} q\right)\rangle \\ &= \left\langle G q^\top, \partial T_{\text{cw}} \right\rangle + \left\langle T_{\text{cw}}^\top G, \partial q \right\rangle.\end{aligned} \tag{39}$$

The gradients with respect to T_{cw} and μ are then

$$\frac{\partial \mathcal{L}}{\partial T_{\text{cw}}} = \frac{\partial \mathcal{L}}{\partial t} q^\top \in \mathbb{R}^{4\times 4}, \quad \frac{\partial \mathcal{L}}{\partial \mu} = R_{\text{cw}}^\top \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial t_x} & \frac{\partial \mathcal{L}}{\partial t_y} & \frac{\partial \mathcal{L}}{\partial t_z} \end{bmatrix}^\top \in \mathbb{R}^3 \tag{40}$$

D.4 Scale and rotation gradients

Now we have $\Sigma = MM^\top$ and $\frac{\partial \mathcal{L}}{\partial \Sigma}$. Letting $G = \frac{\partial \mathcal{L}}{\partial \Sigma}$, we have

$$\begin{aligned}\partial \mathcal{L} &= \langle G, \partial \Sigma \rangle \\ &= \left\langle G, (\partial M) M^\top + M\left(\partial M^\top\right)\right\rangle \\ &= \left\langle G M + G^\top M, \partial M \right\rangle\end{aligned} \tag{41}$$

which gives us

$$\frac{\partial \mathcal{L}}{\partial M} = \frac{\partial \mathcal{L}}{\partial \Sigma} M + \frac{\partial \mathcal{L}}{\partial \Sigma}^{\top} M \tag{42}$$

Now we have $M = RS$, with $G = \frac{\partial \mathcal{L}}{\partial M}$ as

$$\begin{aligned} \partial \mathcal{L} &= \langle G, \partial M \rangle \\ &= \langle G, (\partial R) S \rangle + \langle G, R(\partial S) \rangle \\ &= \left\langle G S^{\top}, \partial R \right\rangle + \left\langle R^{\top} G, \partial S \right\rangle \end{aligned} \tag{43}$$

which gives us

$$\frac{\partial \mathcal{L}}{\partial R} = \frac{\partial L}{\partial M} S^{\top}, \quad \frac{\partial \mathcal{L}}{\partial S} = R^{\top} \frac{\partial L}{\partial M}. \tag{44}$$

The Jacobians of the rotation matrix R wrt the quaternion parameters $q = (w, x, y, z)$ are

$$\frac{\partial R}{\partial w} = 2 \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}, \frac{\partial R}{\partial x} = 2 \begin{bmatrix} 0 & y & z \\ y & -2x & -w \\ z & w & -2x \end{bmatrix}, \tag{45}$$

$$\frac{\partial R}{\partial y} = 2 \begin{bmatrix} -2y & x & w \\ x & 0 & z \\ -w & z & -2y \end{bmatrix}, \frac{\partial R}{\partial z} = 2 \begin{bmatrix} -2z & -w & x \\ w & -2z & y \\ x & y & 0 \end{bmatrix}. \tag{46}$$

The Jacobians of the scale matrix S with respect to the scale parameters $s = (s_x, s_y, s_z)$ are

$$\frac{\partial S}{\partial s_j} = \delta_{ij} \tag{47}$$

whichs selects the corresponding diagonal element of $\frac{\partial \mathcal{L}}{\partial S}$.

Appendix E. Data Conventions

Various conventions are used within the `gsplat` library. We briefly outline the most important ones.

E.0.1 Rotation matrix representation

Similar to the original work by Kerbl et al., we represent a Gaussian rotation by a four dimensional quaternion $q = (w, x, y, z)$ with the Hamilton convention such that the $SO(3) \in \mathbb{R}^{3\times3}$ rotation matrix is given by

$$R = \begin{bmatrix} 1 - 2\left(y^2 + z^2\right) & 2\left(xy - wz\right) & 2\left(xz + wy\right) \\ 2\left(xy + wz\right) & 1 - 2\left(x^2 + z^2\right) & 2\left(yz - wx\right) \\ 2\left(xz - wy\right) & 2\left(yz + wx\right) & 1 - 2\left(x^2 + y^2\right) \end{bmatrix}. \tag{48}$$

E.0.2 Pixel Coordinates

Conversion to discrete pixel coordinates $\boldsymbol{p} = (p_i, p_j) \in \mathbb{Z}^+$ from continuous image coordinates $\boldsymbol{\mu'} = (\mu'_x, \mu'_y) \in \mathbb{R}^+$ assumes that a pixel's center is located at the center of a

box of area 1. This gives the following relation between pixel space, image space, and 3D coordinates $\boldsymbol{t} = (t_x, t_y, t_z)$:

$$\begin{aligned} p_i + 0.5 &= \mu'_x = f_x \cdot t_x / t_z + c_x \\ p_j + 0.5 &= \mu'_y = f_y \cdot t_y / t_z + c_y \end{aligned} \tag{49}$$

where (f_x, f_y, c_x, c_y) are the pinhole camera intrinsics.