Regularizing 3D Gaussian Splatting for Sparse Input

Quanhong Liu, and Vanessa Yu

Abstract—Radiance field techniques have gained popularity for novel view synthesis. Among these, 3D Gaussian Splatting (3DGS) stands out for its quality and speed. However, 3DGS encounters difficulties when handling sparse input data, resulting in overfitting and unwanted artifacts in the generated images. To address this limitation, we explore the integration of geometry regularization techniques inspired by RegNeRF. Our approach involves randomly sampling unobserved viewpoints based on input camera positions and rotations, followed by rendering depth images and calculating a depth smoothness loss. This new loss metric is then combined with the standard loss used in 3DGS to create a comprehensive loss function. Despite the effectiveness of our regularization approach in scenes with gradual depth variations and bounded structures, challenges remain. Specifically, our method may struggle to handle rapid depth changes within localized regions and distant background objects. In conclusion, our project demonstrates promising regularization techniques for 3DGS but also highlights areas where further refinement is needed, particularly when dealing with scenes featuring abrupt depth variations and distant background elements.

Index Terms—3D Reconstruction, Novel View Synthesis, Radiance Fields

1 Introduction

Methods associated with radiance fields have become increasingly popular for the purpose of synthesizing novel views. The objective is to produce images of a scene or object from perspectives that were not included in the initial set of views used for model training.

Recently, 3D Gaussian Splatting (3DGS) [1] has emerged as a notable technique, drawing considerable interest in various fields. This method excels in providing high-quality results, fast reconstruction speeds, and the capability for real-time rendering. It conceptualizes a 3D environment by deploying many 3d Gaussians, and uses a rasterizer for rendering. While 3DGS stands out as an effective method for creating novel views in a scene, it struggles with handling sparse input data. In situations where it is provided with a limited number of input views, 3DGS tends to overfit to these specific images. This overfitting leads to a failure in accurately capturing the broader, global structure of the scene. Consequently, this might result in the appearance of floating artifacts in the scene, detracting from the overall quality and realism.

One approach to avoid overfitting is by regularization. However, the difference in the model structure and rendering method makes the existing regularization techniques for NeRF not easily applicable to 3DGS. To address this problem, we want to investigate a way to incorperate regularization technique to 3DGS for handling spare inputs. Drawing inspiration from RegNeRF, we first randomly sample unobserved viewpoint cameras based on input cameras' positions and rotations for each iteration. Next, similar to how a pixel's color is rendered in 3DGS, we render a depth image. This involves determining the expected depth for each pixel as seen by the unobserved viewpoint camera. With the depth image in hand, we proceed to calculate a depth smoothness loss. This new loss metric is then combined with the standard loss used in 3DGS and form our final loss.

We have also discovered that certain optimization techniques employed in 3DGS negatively affect the model's performance when dealing with sparse input.
To summarize, our main contributions are the following:

- We modified the optimization process in 3DGS for better handling sparse inputs.
- We proposed a novel Gaussian depth calculation method for 3DGS.
- A geometry regularizer for depth maps rendered from unobserved viewpoints, which improves scene geometry and avoid overfitting.

2 Related Work

2.1 Radiance Fields

Neural radiance field (NeRF) [2] is an emerging technique for synthesizing realistic 3D scenes using a large Multi-Layer Perceptron (MLP) model. The model optimizes a continuous 5D neural radiance field representation of a scene from a set of input images. Despite its success, this method has several limitations. The original NeRF suffers from its large computational expense. InstantNGP [3] tackles this issue by adopting a multi-scale grid to replace positional encoding. In Mip-NeRF [4], the traditional method of point-based ray tracing is substituted with cone tracing to address issues of sampling and aliasing. To further extend on that, Mip-NeRF360 [5] addresses the problem of scalability and unbounded scenes in the original NeRF.

3DGS tackles challenges related to speed (for both training and rendering), scalability, and handling unbounded scenes. Unlike the NeRF related techniques [2], [3], [4], [5]

- *Quanhong Liu and Vanessa Yu are with the Department of Computer Science, University of Toronto, Canada.*
E-mail: {quanhong, vyu}@cs.toronto.edu

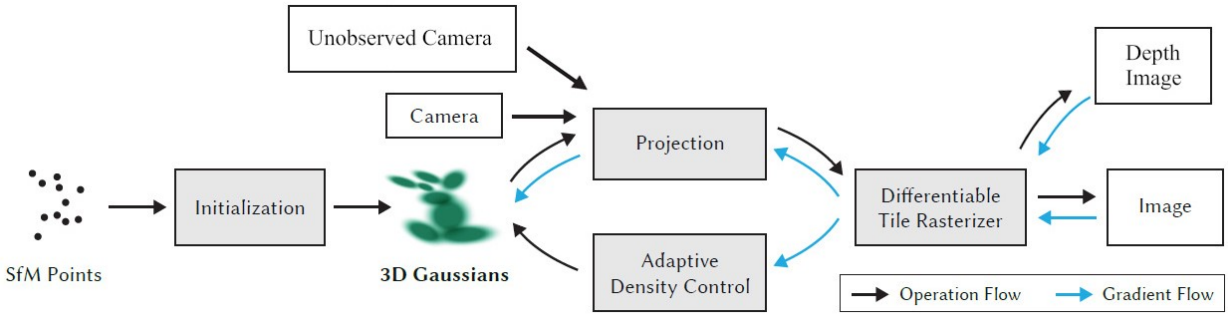


Fig. 1. model overview

which train neural networks to map points or volumes to color and volume density and then perform volume rendering by ray casting, Gaussian Splatting trains a set of anisotropic 3D Gaussians and uses a rasterizer for Gaussians for fast rendering.

2.2 Sparse Input Novel-View Synthesis

One approach to acheive Sparse Input Novel-View Synthesis is by regularizing appearance and geometry in novel views. Previous works in this direction include DS-NeRF [6] and DietNeRF [7] and RegNeRF [8]. RegNeRF proposed a regularization suitable for NeRF based models for better geometry and color permanence with fewer input views provided. By regularizing the geometry and appearance of patches rendered from unobserved viewpoints and using a normalizing flow model to regularize the color of unobserved viewpoints, RegNeRF obtained some outstanding results. Since RegNeRF shows that regularizing scene geometry has a much bigger impact on the performance than appearance regularization, we are going to focus on the geometry regularizer in our project as well.

3 Proposed Method

We build our model based on 3DGS code base. In the initialization phase, like RegNeRF, we randomly sample unobserved viewpoint cameras based on the input cameras. In each training iteration, in addition to rendering an input camera for the color image, we also render an unobserved camera but render for depth image. We then calculate the geometry loss based on the depth image rendered for the unobserved camera and add the geometry loss to the total loss. The overview of the whole model is illustrated in Fig. 1.

3.1 Sampling Camera

In the initialization phase, we sample unobserved viewpoint cameras to be used for depth image rendering in training iterations. Like in RegNeRF, we need to sample a position and a rotation for each camera. We define the space of all possible camera locations as the bounding box of all input camera positions

$$\mathcal{S}_t = \{t \in \mathbb{R}^3 | \mathbf{t}_{min} \leq \mathbf{t} \leq \mathbf{t}_{max}\}$$

Fig. 2. Example sampled cameras

where $\mathbf{t}_{min}$ and $\mathbf{t}_{max}$ are the elementwise minimum and maximum values of all input camera positions, respectively.

For the sample space of camera rotations, We make the assumption that all cameras are approximately centered on one focal point within the scene. We establish a shared "up" axis, denoted as $\bar{\mathbf{p}}_u$, by calculating the normalized average of the up axes from all input cameras. Subsequently, we determine a mean focal point, denoted as $\bar{\mathbf{p}}_f$, by addressing a least-squares problem to identify the 3D point that minimizes the squared distance to the optical axes of all input cameras. Additionally, we introduce random perturbations to the focal point prior to the computation of the camera rotation matrix. Hence, the set of all possible camera rotations, given the position $\mathbf{t}$, is

$$\mathcal{S}_R|\mathbf{t} = \{\mathbf{R}(\bar{\mathbf{p}}_u, \bar{\mathbf{p}}_f + \epsilon, \mathbf{t}) | \epsilon \sim \mathcal{N}(0, 0.125)\}$$

Here, $\mathbf{R}(\cdot,\cdot,\cdot)$ represents the resulting camera rotation matrix, and ϵ represents a minor perturbation introduced to the focal point.
Hence we obtain our randomly sampled camera as:

$$\mathcal{S}_P = \{[\mathbf{R}|\mathbf{t}] | \mathbf{R} \sim \mathcal{S}_R, \mathbf{t} \sim \mathcal{S}_t\}$$

See Fig. 2 for sampled camera examples.

3.2 Depth Calculation:

Gaussian Depth Calculation In the rasterization process, we calculate the depth of a Gaussian when rendered to a pixel by first transforming the Gaussian from world space to ray space using EWA splatting [9] and then computing the expectation of the Gaussian conditional to the pixel location. To transform a 3D Gaussian G with mean μ and covariance

matrix $\mathbf{\Sigma}$ from world space to ray space with camera specifications (view matrix W, focal length f_x and f_y), we first transform the Gaussian's mean to get its mean $\mathbf{t}$ in camera space:

$$\mathbf{t} = \mathbf{W}\mu$$

Then we compute the affine Jacobian as:

$$\mathbf{J} = \begin{pmatrix} f_x/\mathbf{t_z} & 0 & -f_x t_x/t_z^2 \\ 0 & f_y/\mathbf{t_z} & -f_y t_y/t_z^2 \\ \mathbf{t_x}/\mathbf{l} & \mathbf{t_y}/\mathbf{l} & \mathbf{t_z}/\mathbf{l} \end{pmatrix}$$
$$\text{where } l = \|\mathbf{t}\|$$

Then we compute the Gaussian's covariance matrix V in ray space as:

$$\mathbf{V} = \mathbf{JW\Sigma W^T J^T}$$

In this ray space, the x and y coordinates represent the location on the camera screen, and the z coordinate represents the distance to the camera. 3DGS already uses EWA splatting to obtain the 2D covariance on the screen $\mathbf{V_{2D}}$ by skipping the third row and column of $\mathbf{V}$. We utilize $\mathbf{k} = (\mathbf{V_{3,1}}, \mathbf{V_{3,2}})$ (row vector) to calculate the mean of the conditional distribution of z given x and y in ray space as:

$$\mathbf{E}(z \mid x, y) = \mu_z + \mathbf{k}\mathbf{V_{2D}^{-1}}((\mathbf{x}, \mathbf{y})^\mathbf{T} - \mu)$$

We use this mean as our depth of the Gaussian respect to a pixel, so

$$d = \mathbf{E}(z \mid x, y)$$

Expected Depth Calculation We calculate our expected depth for a given pixel similar to how 3DGS renders color for a given pixel. 3DGS renders an image by rasterizing the gaussians through α-blending. For an ordered set of Gaussian that covers the pixel, we have:

$$C = \sum_{i \in N} c_i \alpha_i T_i$$
$$where\ T_i = \prod_{j=1}^{i-1}(1 - \alpha_j)$$

where C is the color of the pixel, c_i is the color of the i^{th} Gaussian computed using the sphere harmonics features and the viewing direction, and α_i is the opacity of the i^{th} Gaussian rendered to the pixel's location.
Hence, similar to the above equations, we calculate our expected depth as:

$$D = \sum_{i \in N} d_i \alpha_i T_i$$

where D is the expected depth of the pixel, d_i is our calculated depth of the i^{th} Gaussian.

3.3 Loss

Depth Smoothness Loss We formulate our depth smoothness loss as:

$$\mathcal{L}_{DS}(\theta, \mathcal{C}_r) = \sum_{\mathbf{r} \in \mathcal{C}_r} \frac{1}{WH} \sum_{i,j} (\hat{d}_\theta(\mathbf{r}_{ij}) - \hat{d}_\theta(\mathbf{r}_{i+1j}))^2 + (\hat{d}_\theta(\mathbf{r}_{ij}) - \hat{d}_\theta(\mathbf{r}_{ij+1}))^2$$

where W, H is the width and height of the image respectively, $\hat{d}_\theta(\mathbf{r}_{i+1j})$ is the expected depth at pixel i, j; θ is the model parameter and $\mathcal{C}_r$ is the random sampled unobserved camera views. Note that, for pixels that have no Gaussian contribution to them, we do not include them when calculating the geometry loss.

Original Loss in 3DGS The orginal loss in 3DGS is:

$$\mathcal{L}(\theta, \mathcal{C}_i) = \mathcal{L}_1 + \mathcal{L}_{D-SSIM}$$

It's a $l1$ loss combined with a D-SSIM term, where $\mathcal{C}_i$ is the input camera views.

Total loss The total loss we optimize in each iteration is:

$$\mathcal{L}_{total} = \mathcal{L} + \lambda_D \mathcal{L}_{DS}$$

where λ_D is a hyperparameter.

3.4 Camera Extent Change

When training the models (both 3DGS and our model) in sparse input, we discovered that the original code of pruning Gaussians that are too large has a dramatic negative effect on the results. We figured it's because the camera extent, which determines the pruning threshold, is computed by the distance between input cameras which is usually too small in sparse input. After all, Colmap cannot generate a point cloud for a dataset that is both sparse and scattered. Therefore, we utilized the average focus point we calculated when generating random cameras, and computed the maximum distance between it and the cameras, and the maximum between it and the original camera extent to get the new camera extent. With this change, we make the 3DGS capable of handling concentrated cameras.

3.5 Training Details

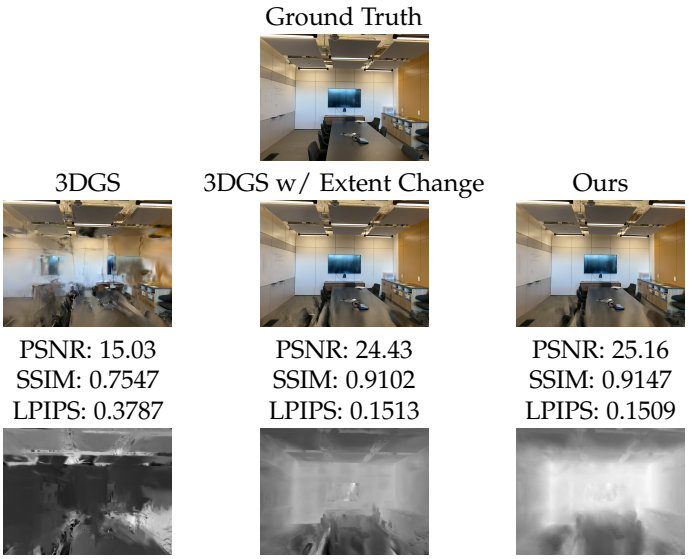
When training our model, we use a λ_D of 10^{-2}. We take 3 images as training data and train for 7000 iterations. We use 2 images as testing data. We train on one NVIDIA RTX 4070 Ti GPU.

4 Experimental Results

We show three sets of results to demonstrate the regularization ability of our model and its limitations.

4.1 Room

4.1.1 test results



From the above results we can tell that our extent change method and the regularizer works quite well on the Room scene. Our model is able to acheive the best performance among all three. From Fig. 3, we can tell that the original 3DGS is able to acheive about an equal performance till iteration 3000. Our extent change method is able to fix that.

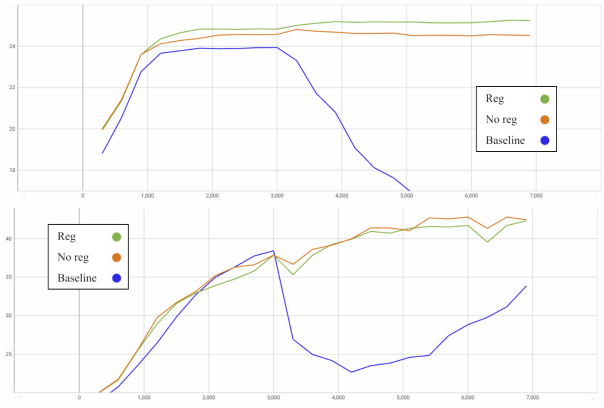


Fig. 3. Room - (up) test psnr, (bottom) train psnr vs. iteration plot

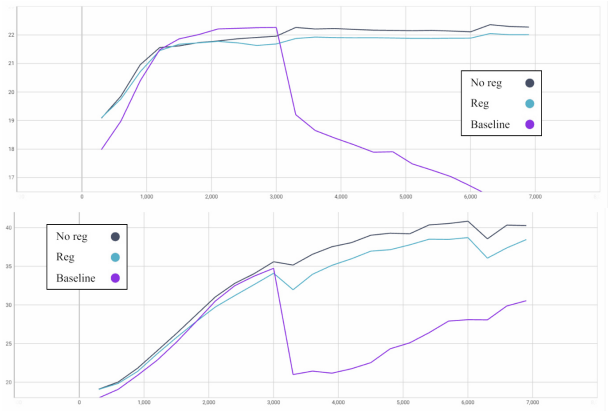
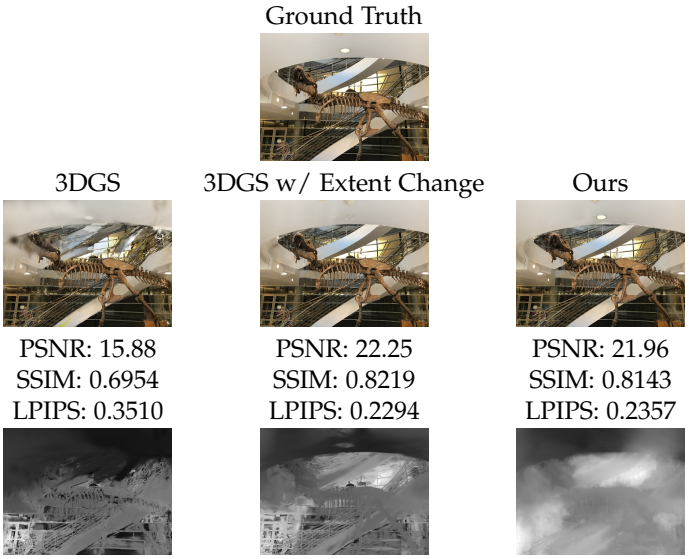


Fig. 4. Trex - (up) test psnr, (bottom) train psnr vs. iteration plot

4.2 Trex

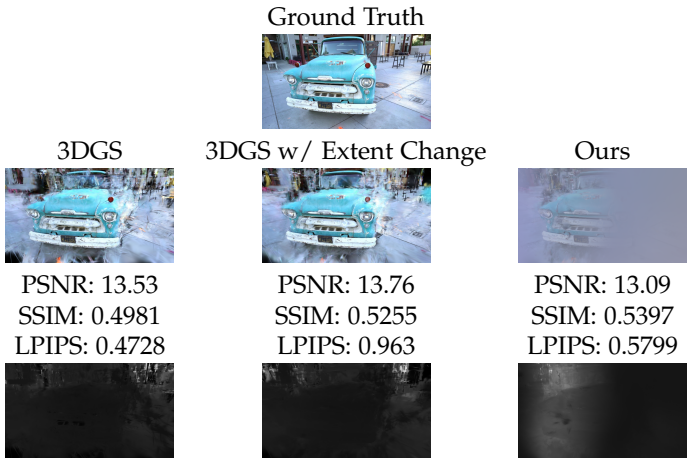
4.2.1 test results



Our extent change still has an effect on the above result, but our geometry regularizer fails probably because the scene has a lot of high-frequency domains, such as the skeleton and the stairs, which our geometry regularizer cannot maintain.

4.3 Truck

4.3.1 test results



From the above result, we can see that both the extent change and geometry regularizer do not work anymore. This is probably due to the unbounded nature of this scene.

5 Conclusion

We investigated a way to incorporate geometry regularization in RegNeRF to 3DGS for handling spare inputs. Our key insight is that our regularizer is effective in scenarios where scenes are reasonably confined and exhibit gradual

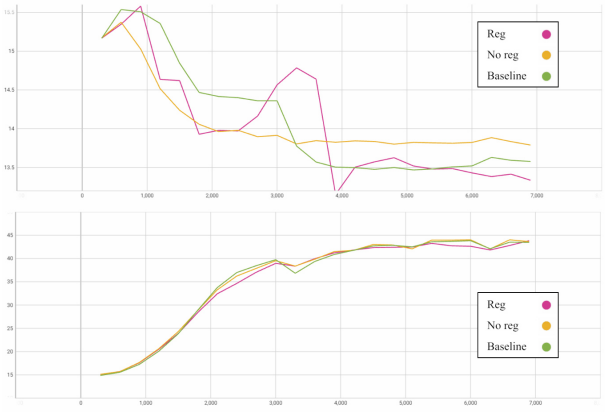


Fig. 5. Truck - (up) test psnr, (bottom) train psnr vs. iteration plot

depth variations. However, it falls short in enhancing model performance when the input image contains abrupt depth changes in localized regions, as our depth smoothness loss cannot effectively capture such variations. Additionally, our regularizer is less effective when the background is significantly distant from the central object. In this situation, our regularizer tends to concentrate all elements toward the center of the scene and fails to produce a realistic outcome.

6 Limitations and Future Work

In this project, rendering an additional camera in each iteration to generate depth image is quite computationally heavy. We believe that investigating pre-trained models designed for efficient and accurate depth estimation could be a promising direction for further exploration. Another limitation to consider is our utilization of the depth smoothness loss. This particular loss may not be the most suitable option when the input image exhibits rapid depth variations within a confined local area. Moreover, integrating the color regularization technique from RegNeRF [8] and exploring other regularization methods could potentially result in improved performance.

References

[1] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering," *ACM Transactions on Graphics*, vol. 42, no. 4, July 2023. [Online]. Available: https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/

[2] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," *CoRR*, vol. abs/2003.08934, 2020. [Online]. Available: https://arxiv.org/abs/2003.08934

[3] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *ACM Trans. Graph.*, vol. 41, no. 4, pp. 102:1–102:15, Jul. 2022. [Online]. Available: https://doi.org/10.1145/3528223.3530127

[4] J. T. Barron, B. Mildenhall, M. Tancik, P. Hedman, R. Martin-Brualla, and P. P. Srinivasan, "Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields," *ICCV*, 2021.

[5] J. T. Barron, B. Mildenhall, D. Verbin, P. P. Srinivasan, and P. Hedman, "Mip-nerf 360: Unbounded anti-aliased neural radiance fields," *CVPR*, 2022.

[6] K. Deng, A. Liu, J. Zhu, and D. Ramanan, "Depth-supervised nerf: Fewer views and faster training for free," *CoRR*, vol. abs/2107.02791, 2021. [Online]. Available: https://arxiv.org/abs/2107.02791

[7] A. Jain, M. Tancik, and P. Abbeel, "Putting nerf on a diet: Semantically consistent few-shot view synthesis," *CoRR*, vol. abs/2104.00677, 2021. [Online]. Available: https://arxiv.org/abs/2104.00677

[8] M. Niemeyer, J. T. Barron, B. Mildenhall, M. S. M. Sajjadi, A. Geiger, and N. Radwan, "Regnerf: Regularizing neural radiance fields for view synthesis from sparse inputs," *CoRR*, vol. abs/2112.00724, 2021. [Online]. Available: https://arxiv.org/abs/2112.00724

[9] M. Zwicker, H. Pfister, J. van Baar, and M. H. Gross.