NPM3D Project: 3D Gaussian Splatting

Clément Weinreich - Master MVA

March 2024

1 Introduction

The objective of novel view synthesis is to create unseen perspectives of a 3D scene or object by interpolating between a collection of images with predefined camera parameters. Introduced by Mildenhall et al. 2021, Neural Radiance Fields (NeRF) have gained widespread popularity as a method for representing 3D scenes. They achieve high-quality visual results by using neural networks trained through differentiable volume rendering techniques. However, these approaches are associated with significant training and rendering expenses. Addressing these challenges, Kerbl et al. 2023 proposed 3D Gaussian Splatting (3DGS), a different approach for 3D scene representation. The method utilizes anisotropic 3D Gaussians to form a non-structured representation of radiance fields. In contrast to the slow volume rendering techniques used in NeRFs, this employs a fast differentiable tile-based rasterization algorithm which allows α-blending of anisotropic splats to project the points on the 2D plane for rendering views. In this project, the 3DGS method is analyzed in depth from a conceptual and experimental point of view. The original 3D Gaussian Splatting method is presented in Section 2. Section 3 highlights the effects of various hyperparameters across different datasets. Section 4 introduces some enhancements to the methods, while Section 5 examines limitations and compares them with other recent advancements in the literature. The report concludes with final thoughts in Section 6.

2 Method

3D Gaussian Splatting consists of a Gaussian point cloud representation in 3D space. Each Gaussian g is characterized by its mean 3D position vector $\mu_g \in \mathbb{R}^3$, and its anisotropic covariance matrix Σ_g. The covariance is parametrized by a scaling vector $s_g \in \mathbb{R}^3$ and a quaternion $q_g \in \mathbb{R}^4$ encoding the rotation. Additionally, each Gaussian includes an opacity value $o_g \in [0, 1]$, and a set of spherical harmonics (SH) coefficients $\mathcal{C}_g$ that defines the color, which is particularly convenient for view-dependent appearance. Hence, in the world coordinate system, a 3D Gaussian function can be expressed as

$$G_{\mu_g,\Sigma_g}(x) = \exp\left(-\frac{1}{2}(x-\mu_g)^T\Sigma_g^{-1}(x-\mu_g)\right)$$

where Σ_g is built from q_g and s_g.

To render an image from a set of Gaussians and a given viewpoint, the method relies on a rasterizer. This rasterizer splats the 3D Gaussians into 2D Gaussians that are parallel to the image plane. First, they perform frustum culling where they only keep the 3D Gaussians that intersect the view frustum, i.e. that can be observed from the given viewpoint and position. They also reject Gaussians that are too close to the near plane and far outside the view frustum. Then they project the 3D Gaussians in 2D which involves an affine transformation of the 3D Gaussians from the world coordinate to the camera coordinate system, and finally to the image coordinate using an affine local approximation. The visible 2D Gaussians are then sorted by depth and composited from front to back to construct the output image. To achieve this, the screen space is divided into tiles of size 16×16. For each Gaussian, they compute the axis-aligned bounding box, which encloses the 99% confidence ellipse (equivalent to 3 sigma) of each 2D projected covariance. The Gaussian is included in a tile bin if its bounding box intersects with the tile. Given that a projected Gaussian may cover several tiles, they replicate each Gaussian according to the number of tiles they overlap and assign an identifier for the relevant tile so each Gaussian is associated with a key combining the tile identifier and the view space depth. Then they sort the Gaussians based on these keys using the GPU Radix sort method to get a list of Gaussians sorted by depth for each tile. The rasterization is performed in parallel for each tile, they get the associated sorted Gaussians and for all pixels in the tile, the colors and opacity values are accumulated by traversing the list front to back (closest to farthest). If we denote the color at pixel i by C_i, and let j index the N Gaussians involved in that pixel, the blending is performed as:

$$C_i = \sum_{j \leq N} c_j \alpha_j \prod_{k=1}^{j-1} (1 - \alpha_k), \text{ with } \alpha_j = o_j G_{\mu'_j, \Sigma'_j}(x'_i)$$

where c_j is the RGB color obtained by evaluating the SH coefficients $\mathcal{C}_j$, μ' is the 2D Gaussian center, Σ' its 2D Covariance and x' the pixel center coordinates. When a pixel reaches target saturation (full opacity, α goes to 1) the processing of the pixel stops, and when all pixels have saturated in a tile, the processing of the tile is done.

Using a Structure from Motion (SfM) algorithm like COLMAP, they initialize the scene with a set of isotropic 3D Gaussians mapped onto the sparse point cloud generated from input images and their camera positions. The isotropic covariance is initialized with the mean distance to the closest 3 other Gaussians, and the opacity of every Gaussian is set to 0.1. Optimization of the Gaussian's attributes is performed using Stochastic Gradient Descent as the rendering process is fully differentiable. For each view (image and camera parameters) drawn from the training dataset, the corresponding image is synthesized (projection of the Gaussians and rasterization) with the forward process explained above. The reconstruction loss that we minimize is

$$\mathcal{L} = (1 - \lambda)\mathcal{L}_1 + \lambda \mathcal{L}_{D-SSIM}$$

where $\mathcal{L}_1$ is the L_1 difference between the rendered image and the ground truth, and $\mathcal{L}_{D-SSIM}$ is the Data Structural Similarity Index.

Due to the ambiguities of 3D to 2D projection, geometry may be incorrectly placed. Moreover, the method strongly relies on the redundancy of the input images to initialize the 3D Gaussian point cloud. Thus, the optimization process needs to be able to create, destroy, and move geometry if it has been incorrectly positioned. For that purpose, the optimization process is interleaved with adaptive density control (pruning or densification) where they periodically add or remove 3D Gaussians from the scene. The pruning operation consists of removing Gaussians that are too transparent, i.e., $o_g \leq \epsilon_o$. The densification process addresses under-reconstructed and over-reconstructed regions that can both be detected as they produce large view-space positional gradients. In the under-reconstruction case where the region is missing geometric features, such as a small Gaussian within a wider geometry, the Gaussian is cloned and moved in the direction of the positional gradient. In the case of over-reconstruction where there is a large Gaussian in a high variance region, the Gaussian is split into two smaller Gaussians positioned on sampled points from the original Gaussian.

Especially in unseen regions, the method is particularly sensible to floaters, i.e. isolated small part of the volumetric representation that is not integrated into the main structure. For this issue, they periodically set the opacity o_g close to 0. The optimization process will increase the opacity for relevant Gaussians, while others will be removed by the pruning step. Another issue that can occur is the extreme elongation of some Gaussians, resulting in very large Gaussians polluting the scene. To cope with these anomalies they periodically remove very large Gaussians in the worldspace and viewspace.

In the end, Gaussian Splatting generally produces scenes with millions of Gaussians. The choice of a 3D Gaussian primitive allows to preserve the properties of volumetric rendering for optimization while directly allowing fast splat-based rasterization.

3 Experiments

3.1 Experimental setup

To better understand the importance of different components of the method, I conducted some experiments where I varied some of the hyperparameters or directly changed/removed some components. I have tested the method on different datasets that already contain the SfM output data, and selected three for these experiments. From the Tanks&Temples dataset, I used the scenes "train" (301 images) and "truck" (251 images) that are provided by the authors of Kerbl et al., 2023. These two scenes are quite challenging as they consist of large unbounded outdoor environments. The last scene I used is "reindeer", a custom dataset of 749 images captured by Alexandre Carlier and available on reshot.ai. Compared to the two others, this scene is easier as it only consists of a single object with a white background and a lot of redundant points of view in the dataset.

To perform these experiments, I used the Google Cloud Platform and Kaggle. As the outdoor scenes are highly demanding in GPU memory, I trained using images of lower resolution (divided by two) which allowed me to test the method faster in different settings. Following the Gaussian Splatting paper, I trained on 30K iterations and used every 8th photo of the dataset for the test set. For a meaningful comparison, the results are evaluated using the standard PSNR (Peak Signal to Noise Ratio), SSIM (Structural Similarity), and LPIPS (Learned Perceptual Image Patch Similarity) metrics. All of the numerical and qualitative results presented in this report are computed on the test set. On a T4 GPU with 16GB VRAM, the training time is close to 40 minutes for the truck and train scenes and requires about 20 minutes for the reindeer scene. To visualize the impact of changing the hyperparameters, all the results are compared to the baseline run on each dataset with the default hyperparameters shown in Table 1. Qualitative results of the baseline obtained on the 3 datasets are also available in Appendix A. Moreover, all the raw results of the experiments are available in Appendix B. The code used to perform the experiments is built on top of the original 3DGS GitHub repository, and can be accessed on GitHub: `https://github.com/Clement-W/gaussian-splatting-experiments`.

Hyperparameter/Component	**Default Value**
Loss function	$0.8\mathcal{L}_1 + 0.2\mathcal{L}_{D-SSIM}$
Spherical harmonics degree	3
Number of iterations for densification start	500
Number of iterations for densification stop	15,000
Threshold for densification based on 2D positional gradient	0.0002
Densification interval	Every 100 iterations
Opacity reset interval	Every 3000 iterations

Table 1: Default hyperparameters and components of the baseline used in all the following experiments.

3.2 Loss function

The initial loss function combines an L1 and an SSIM term, which is not discussed in the original paper. To study the impact of the components involved in the loss function, I tested different combinations:

- $\mathcal{L} = \mathcal{L}_{D-SSIM}$: SSIM is designed to measure the perceptual difference between two images by considering luminance, contrast, and structure. It emphasizes structural information and texture similarity over pixel-wise accuracy, which can be more in line with human perception. This term is useful to ensure that fine details are preserved.
- $\mathcal{L} = \mathcal{L}_1$: The L1 loss penalizes the absolute difference between the predicted and actual values. The L1 difference tends to produce sparser solutions, which in the context of image reconstruction could translate to images that might have sharper edges, but less smoothness in homogeneous regions. This term is useful when the goal is to preserve sharp transitions such as edges in a scene, without overly smoothing them out.
- $\mathcal{L} = 0.8\mathcal{L}_2 + 0.2\mathcal{L}_{D-SSIM}$: Combine the SSIM term with the L2 difference instead of L1. The L2 difference squares the differences between predicted and actual values. Due to its quadratic

nature, it can result in smoother gradients and transitions within the image. Since the optimization penalizes large deviations heavily, it encourages a more averaged outcome. This term can be beneficial for achieving an overall smoothness across the scene.

- $\mathcal{L} = 0.8\mathcal{L}_{huber} + 0.2\mathcal{L}_{D-SSIM}$: Combine the SSIM term with the Huber loss instead of L1. The Huber loss is a smooth L1 loss. It behaves similarly to the L2 loss for small errors and like the L1 loss for large errors. This term combines the strengths of L1 and L2, it is less sensitive to outliers than L2 and keeps the smoothness in homogeneous regions that L1 might lose.

As the metrics obtained on the 3 scenes have different scales, and as we are interested in the change induced by modifying the components of the methods, the numerical results are presented as the percentage change compared to the baseline that has been described in subsection 3.1. The percentage change directly indicates a loss or a gain in performance for the given metric and changed component, which facilitates comparisons. The results of these experiments are displayed in Figure 1.

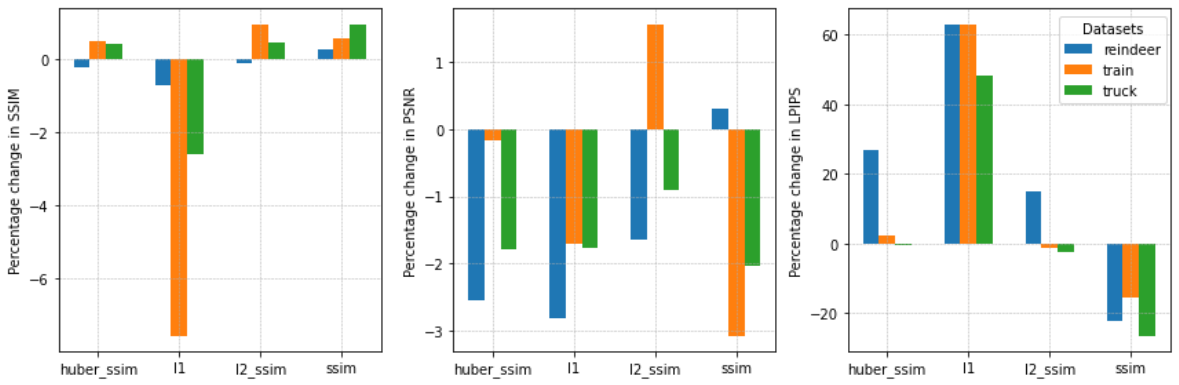


Figure 1: Percentage change in SSIM↑ (left), PSNR↑ (middle) and LPIPS↓(right) compared to the metrics computed on the baseline when changing the loss function. The baseline uses the loss function $\mathcal{L} = 0.8\mathcal{L}_1 + 0.2\mathcal{L}_{D-SSIM}$. Each color corresponds to a different dataset, and the methods tested (huber_ssim, l1, l2_ssim, and ssim) are spaced on the horizontal axis. The percentage indicates a loss (if negative) or a gain (if positive) of that percentage for the given metric and the method in question.

Let's first analyze how using the single components $\mathcal{L}_1$ or $\mathcal{L}_{D-SSIM}$ impacts the results. Using only the L1 loss hurts the results according to the 3 metrics, especially for the SSIM and LPIPS. Visually, only using the L1 loss smooths the reconstructed images, it does not capture all the high-frequency details and average the information as shown in Figure 2 (top). The green boxes highlight the parts that contain high-frequency details that have been smoothed by the L1 loss. This is even more pronounced in areas that are least seen in the dataset, or in specular effects. Reflections and visibility through the window of the car in Figure 2 (bottom) are not well captured by the L1 loss, it results in an averaged reflection (which is still approximately accurate in terms of color). As expected, using only the SSIM term in the loss improves the final evaluation according to the SSIM and LPIPS metrics. However as the optimization was focused on structural similarity and not a pixel-wise difference, the PSNR dropped drastically. It still produces very appealing results, fine details of structure and contrast of the textures are learned as shown in Figure 2 (top). The visibility through the window in Figure 2 (bottom) is well reconstructed, although some noise might have been captured. This better perceptual quality comes at the cost of highly increasing the number of Gaussians in the scene, hence increasing the memory cost and training/rendering time. Looking at Appendix B.1, using only the SSIM term in the loss multiply by 3 the final number of Gaussians on average. To accurately capture and reproduce the complex textures and structural nuances of the scene, the optimization process may end up introducing a larger number of Gaussians, where each Gaussian contributes to modeling finer details to satisfy the SSIM's emphasis on structural similarity.

This first analysis demonstrates the benefits of having a balance between L1 and SSIM within the loss function. The L1 component provides a foundation for pixel-wise accuracy and general smoothness. This can help to anchor the optimization process and provide a stable baseline to minimize a coarse

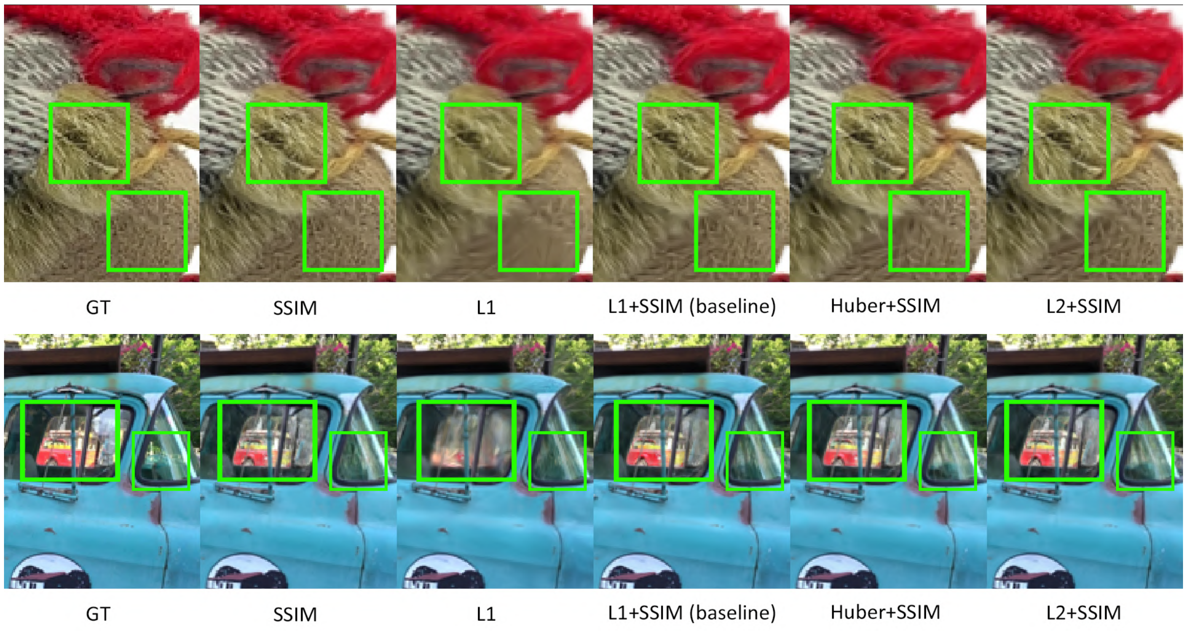


Figure 2: Close-up on synthesized images of the reindeer (top) and truck (bottom) scenes when changing the loss function for the optimization and compared with the ground truth (GT).

error. Then, the SSIM component enriches this foundation by emphasizing the importance of perceptual quality, including texture, contrast, and other structural details. The SSIM term guides the optimization towards solutions that are not only accurate in a pixel-wise sense (with the L1 term), but also maintain the visual integrity of the scene. This combination also allows to keep a relatively small number of Gaussians across the scene while keeping the perceptual details brought by the SSIM. The reconstructed images in Figure 2 (bottom) show that we obtain more details when combining those terms compared to only using the SSIM for optimizing the scene, especially for the windscreen of the truck.

To complete this analysis, I swapped the main L1 term of the loss by Huber, and the L2 difference. Overall the results are very similar or even less good according to the scene. When switching to the L2 difference, we obtain mixed results. Due to the quadratic term, it heavily minimizes large pixel differences which can result in a smoother basis than L1. The fine details lost by this over-smoothing effect are however restored thanks to the SSIM term. The effectiveness of this combination depends heavily on the specific characteristics of each scene, including the distribution of errors, the presence of noise, and the complexity of textures and structures. Finally when swapping the L1 term of the loss with the Huber loss or "smoothed L1 difference", we observe similar results. The similarity between these approaches can be observed in the numerical results of Figure 1, and qualitatively in Figure 2.

It is important to note that the results mainly depend on the scene complexity. As we observe here, the quantitative and qualitative results differ between the outdoor scenes and the single object with a white background. This prevents us from drawing strong conclusions but still helps to better understand the different components and improve our intuition of the inner workings of the optimization process.

3.3 Spherical Harmonics

The authors of Kerbl et al., 2023 employ spherical harmonics to represent the color, which is a common practice when it is necessary to represent view-dependent appearance. They chose a degree of 3 resulting in 16 SH coefficients to optimize for the color. Increasing the number of coefficients inevitably improves the representation of high-frequency details. However, as each coefficient is an RGB float

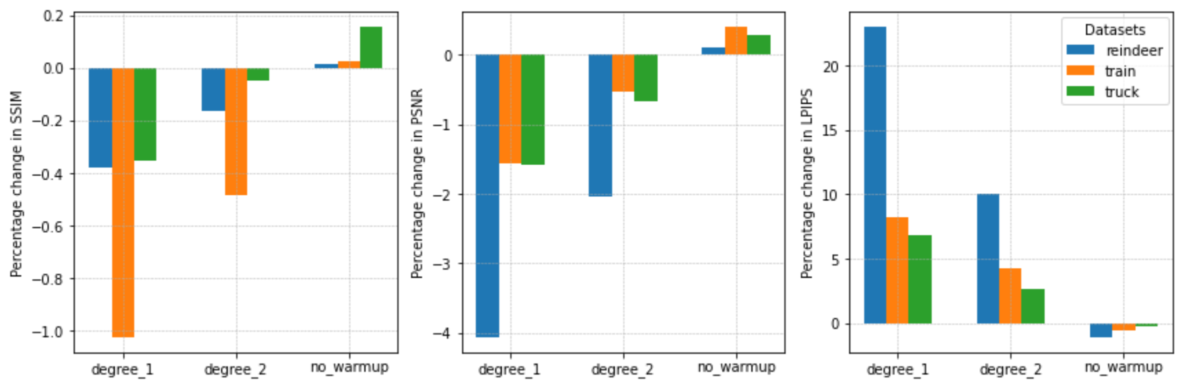


Figure 3: Percentage change in SSIM↑ (left), PSNR↑ (middle) and LPIPS↓(right) compared to the baseline when changing the SH degree or when removing the warm-up of the degrees at the beginning of the training. The baseline uses a degree of 3.

triplet, it remains particularly expensive to store. With a degree of 3, the color coefficients already account for more than 80% (48/54) of the dimensions of the full attribute vector of a Gaussian. To optimize these coefficients, the authors start to optimize the zero-order component (base color). Then they increase the degree every 1000 iterations until it reaches the target, which is 3 here. This choice is motivated by the fact that SH coefficient optimization is sensitive to the lack of angular information. For typical captures with a central object, the optimization works well. However, if the capture has missing angular regions, values for the zero-order component of the SH can be incorrectly produced by optimization. The selected datasets all fall in the case of "central object" scenes. To see the impact of this SH coefficients warm-up on these scenes, I have removed it from the optimization, thus directly optimizing the four bands of the SH. To visualize the impact of having lower SH degrees, I also tested with degrees of 1 and 2. Numerical results are shown in Figure 3.

We first observe the expected behavior of lowering the SH degree on all scenes: the results are getting worse according to the SSIM, PSNR, and LPIPS. In practice, we do not observe a significant difference across the scenes, but the close-up on the reindeer scene in Figure 4 allows us to see the slight improvement of having a higher degree. Regarding the warm-up of the SH coefficients, we observe that it is not beneficial for these scenes. It is even slightly better to directly optimize all the SH bands. This could also be partially due to the experimental setup, where we train on lower-resolution images (divided by 2) which could simplify the representation of the high frequencies in the scenes, and thus neglect the need for this procedure.

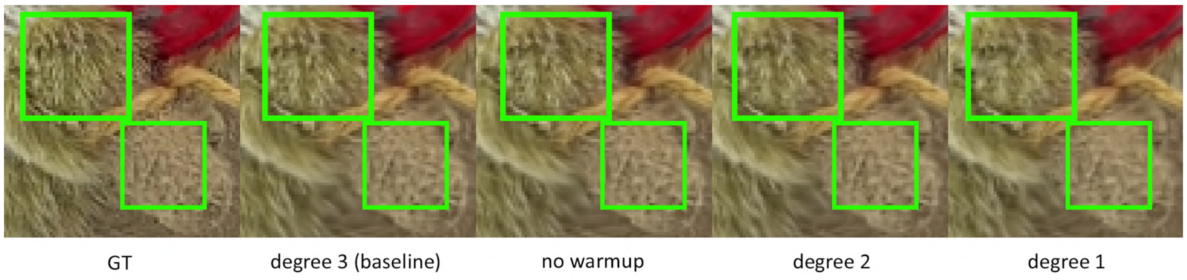
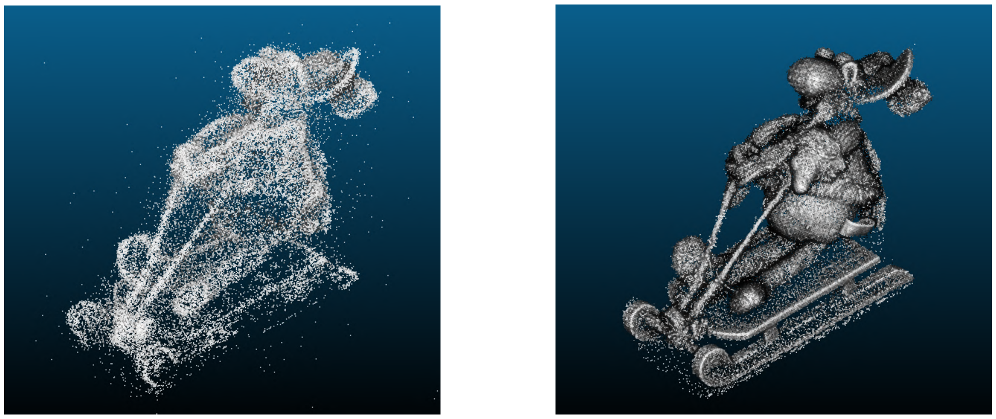


Figure 4: Close-up on a synthesized image of the reindeer scene using different SH degrees or by removing the warm-up of the degrees at the beginning of the training.

3.4 Densification process

3.4.1 Overall performance of the densification

Using the default hyperparameters, the method succeeds in changing the geometry of the scene between the input data and the optimized points. As shown in Figure 5, the densification process removed the useless points and added geometric features where needed. 3DGS managed to add and destroy geometry to obtain a “clean” and dense point cloud of the scene. We notice that many outliers in Figure 5a have been removed as they are no longer present on the final optimized point cloud in Figure 5b. Moreover, some very precise geometric features have been added such as the tag visible on the left-hand side of the reindeer. This first visualization shows the ability of the densification process to move geometry. The next experiments focus on specific aspects of the densification process.



(a) (b)

Figure 5: Comparison of the point cloud obtained by SfM on the reindeer dataset before optimizing with 66 290 points (5a), and point cloud obtained after optimization of the scene with 3D Gaussian Splatting resulting in 211 537 points (5b).

3.4.2 Duration of the densification process

In the baseline, the densification process starts at 500 iterations and stops at 15K iterations, leaving 15K additional iterations for the model to refine the Gaussians without directly adding or removing them. To understand how the optimization behaves regarding the duration of the densification, I tested different times to stop the process: 10K iterations, 20K iterations, and 25K iterations. I also tried to let the densification continue until the end of the training at 30K iterations. The numerical results are displayed on Figure 6.

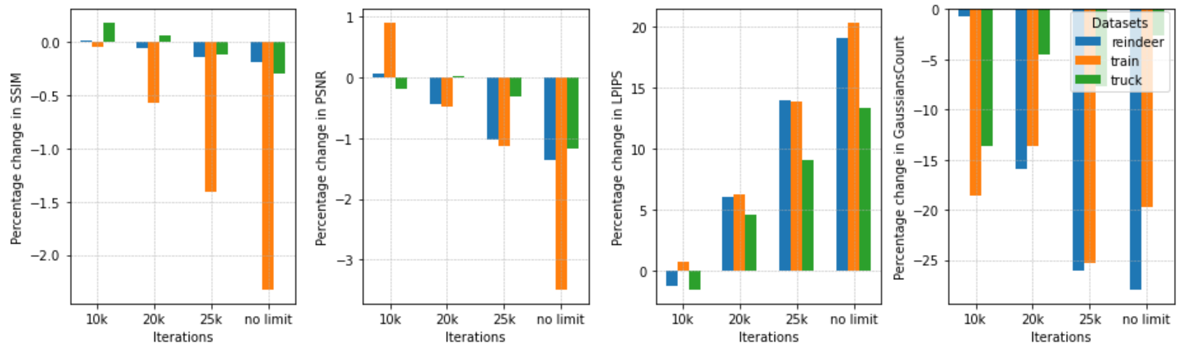
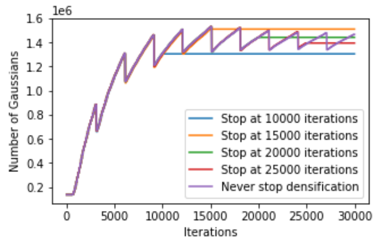
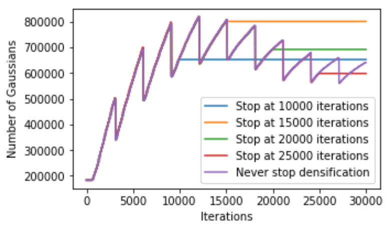


Figure 6: Percentage change in SSIM↑ (first), PSNR↑ (second), LPIPS↓(third) and in the number of Gaussians (fourth) compared to the baseline when changing the iteration at which the densification stops, or not limiting the densification (no limit). The baseline densifies until 15K iterations.

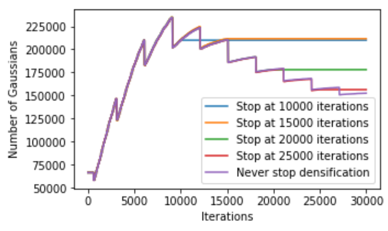
First, we observe that no matter when we stop the training, after 15K iterations the results are getting worse in terms of PSNR, SSIM, and LPIPS. We also notice that the number of Gaussians is decreasing compared to stopping at 15K iterations which can be surprising at first. However, when tracking the number of Gaussians along the optimization, we notice that these numbers reach a maximum between 10K and 15K iterations, and then remain stable or slowly decrease. This behavior is shown in Figure 7. We observe the pruning step that periodically removes transparent Gaussians until a stable number is



(a) Truck dataset



(b) Train dataset



(c) Reindeer dataset

Figure 7: Evolution of the number of Gaussians when stopping the densification at different intervals during optimization. A similar behavior is observed across the different datasets, between 10K and 15K iterations the number of Gaussians reaches a maximum, and then decreases or remains stable.

found. Then, the number of Gaussians oscillates which only brings instability to the optimization. This can also be seen when monitoring the PSNR which does not increase when the densification continues after 20K iterations. The instability prevents the model from refining the Gaussian's properties as the geometry is changing periodically. This shows that the optimization of 3DGS contains different steps. The first step is dedicated to moving the geometry across the scene. Once the scene becomes sufficiently populated, its structure becomes almost fixed (although the position of the Gaussians continues to be refined through SGD), and the rest of the iterations are used to refine the other attributes of the Gaussians. In practice, the PSNR continues to increase after the densification which demonstrates that the other attributes are optimized and benefit from the stability of not being disturbed by the addition or deletion of points.

3.4.3 Densification threshold

A Gaussian is subject to the densification process if the magnitude of its 2D positional gradient is greater than a certain threshold set to 0.0002 in the baseline. Intuitively, this corresponds to regions that are not yet well reconstructed, and the optimization with SGD tries to move the Gaussians to correct it. In these experiments, I change this threshold to visualize the impact of making the algorithm more or less tolerant to decide if a Gaussian needs to be cloned or split. I tested threshold values of 0.0001 (more strict), 0.0003, and 0.0005 (more tolerant). The numerical results are displayed in Figure 8. As one might expect, the number of Gaussians explodes when the threshold is very

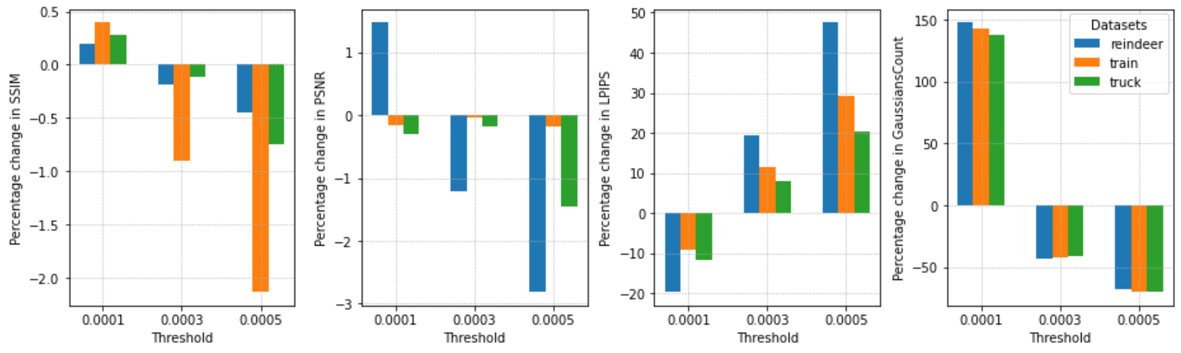
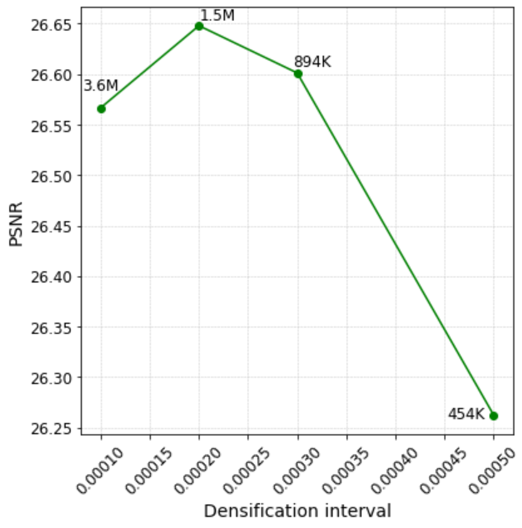
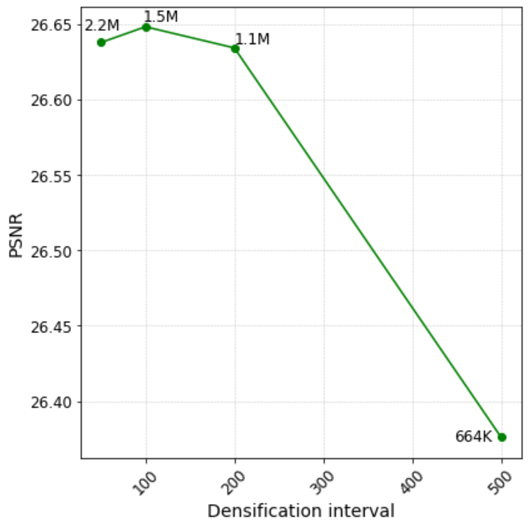


Figure 8: Percentage change in SSIM↑ (first), PSNR↑ (second), LPIPS↓(third) and in the number of Gaussians (fourth) compared to the baseline when changing the threshold value that triggers densification. The baseline uses a threshold of 0.0002.

small. This is expected as it is easier for the magnitude of the 2D positional gradient to be above the threshold, so more points are often cloned or split. For certain scenes, having a lower threshold (0.0001) slightly improves the results, at the cost of having more than twice the number of Gaussians of the Baseline. Even though having a lower threshold could improve the results, it is not worth comparing to the number of Gaussians introduced as shown in Figure 9a. Smaller thresholds lead to



(a)



(b)

Figure 9: Threshold value (9a) and densification interval (9b) versus PSNR for the truck dataset, along with the associated number of Gaussians on each point.

fewer Gaussians while overall, the PSNR remains at the same level. At some point, there is a knee point on the curve where the PSNR rapidly decreases as the number of Gaussians is not enough to represent all the geometric features. Hence, the optimal threshold value must lie right before that point. We observe this behavior in Figure 10 where increasing the threshold prevents us from having specific geometric details on the bench.

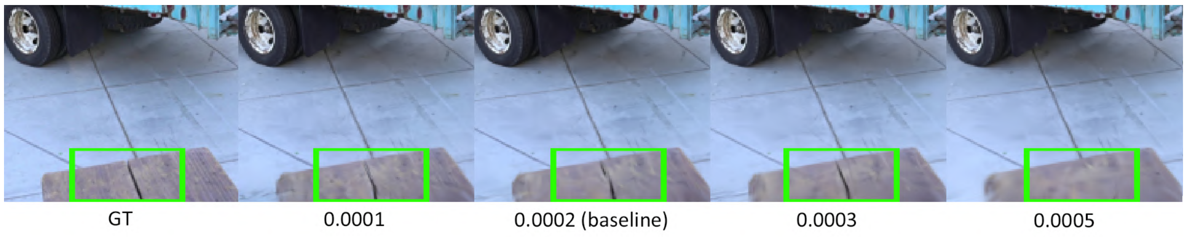


Figure 10: Close-up on a synthesized image of the truck scene using different threshold values to trigger densification.

3.4.4 Densification interval

An important factor that controls how the scene is populated is the densification interval. In the baseline, the densification process is triggered every 100 iterations. These experiments are meant to study the impact of having more or less frequent densifications in the optimization process. I tested to change the interval to 50, 200, and 500 iterations. The numerical results are displayed in Figure 11. Overall, we observe a similar behavior to the densification threshold hyperparameter where increasing the densification frequency also implies an unreasonable increase in the number of Gaussians. The SSIM and LPIPS show a similar trend, but the PSNR provides mixed results, as the densification interval has a different impact according to the complexity of the scene. Looking at Figure 9b, we effectively observe a comparable behavior to the threshold hyperparameter. One can observe a knee

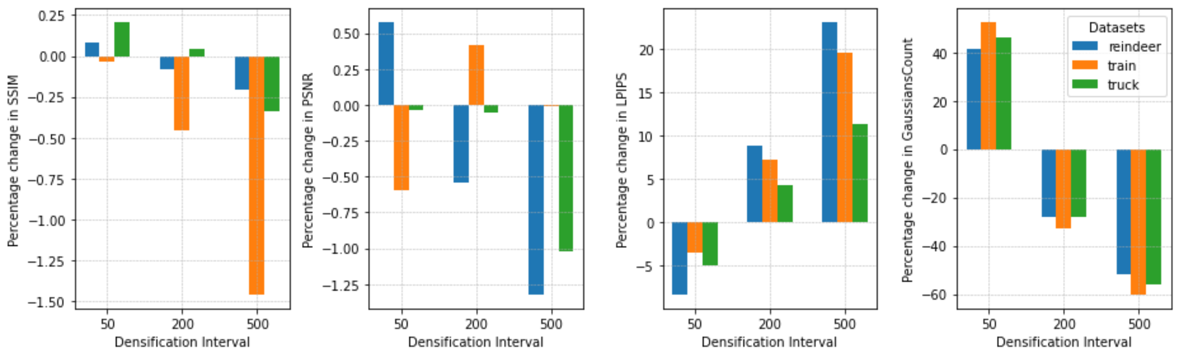


Figure 11: Percentage change in SSIM↑ (first), PSNR↑ (second), LPIPS↓(third) and in the number of Gaussians (fourth) compared to the baseline which triggers the densification process every 100 iterations.

point from which the PSNR decreases drastically. Before that knee point, between an interval of 50 and 200 iterations, the PSNR is approximately equivalent, except that the number of Gaussians increases as the interval decreases. Visually, it is noticeable that the less we densify, the fewer Gaussians we have, leading to a loss of details at all levels. Figure 12 demonstrates that details are lost on objects both very close and very far away from the viewpoint. Again, the optimal densification interval must be found right before the knee point, where the trade-off between the number of Gaussians and reconstruction quality is no longer profitable.

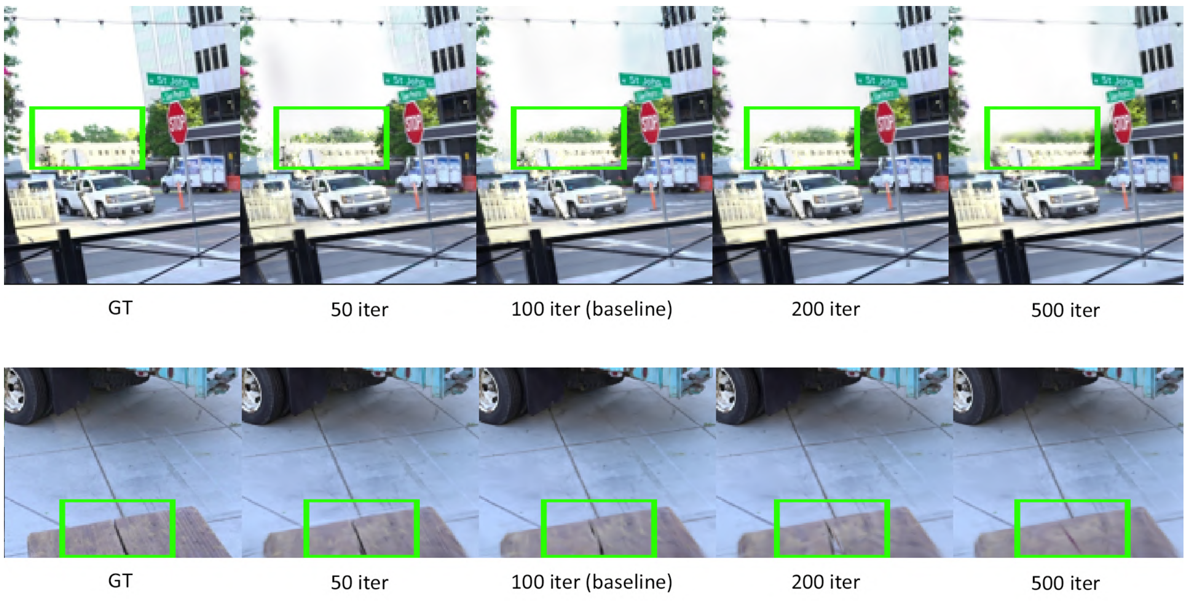


Figure 12: Close-up on two areas of synthesized images of the truck scene when changing the densification interval. The top image shows a faraway object in the background, and the bottom one shows an object close to the viewpoint.

3.4.5 Opacity reset interval

Finally, the pruning step is effective because every 3000 iterations, the opacity of all the Gaussians reset to 0.01, so the optimization with SGD increases the opacity of the relevant Gaussians while the others are pruned. To understand the effect of this component, I tested intervals of 2K and 10K to reset the opacity. I also tested to remove this component from the optimization to better understand its importance. Numerical results are available in Figure 13. First, resetting the opacity more often

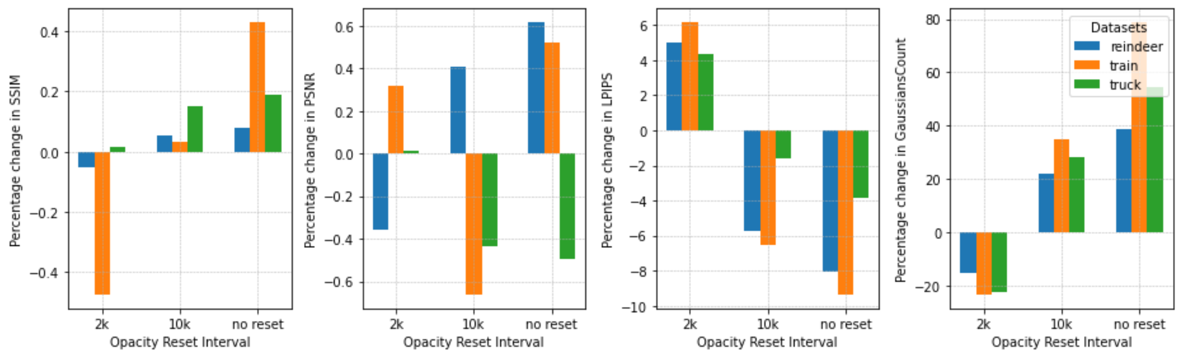


Figure 13: Percentage change in SSIM↑ (first), PSNR↑ (second), LPIPS↓(third) and in the number of Gaussians (fourth) compared to the baseline when changing the opacity reset interval. The baseline reset the opacity every 3000 iterations.

(every 2K iterations) leads to fewer Gaussians in the long term, as more are removed by the pruning step. The impact on the SSIM and PSNR is low (less than 0.5% difference), while the LPIPS drops by about 20%. When resetting the opacity less often (every 10K iterations), the results are slightly better due to a higher number of Gaussians (+30%). Finally, if the opacity is never subject to a reset, the results are also slightly better at the cost of having more Gaussians. Hence, this reset interval must be tuned accordingly to remove the useless Gausians while maximizing the quality. Visually, one might notice on Figure 14 (top) that resetting the opacity regularly allows for the removal of floaters that pollute the scene. However areas containing high frequencies such as in Figure 14 (bottom) are better reconstructed when there are more Gaussians, so when the opacity is less often reset. These two examples demonstrate the trade-off in removing the Gaussians regularly and keeping high-frequency details with a high enough number of Gaussians.

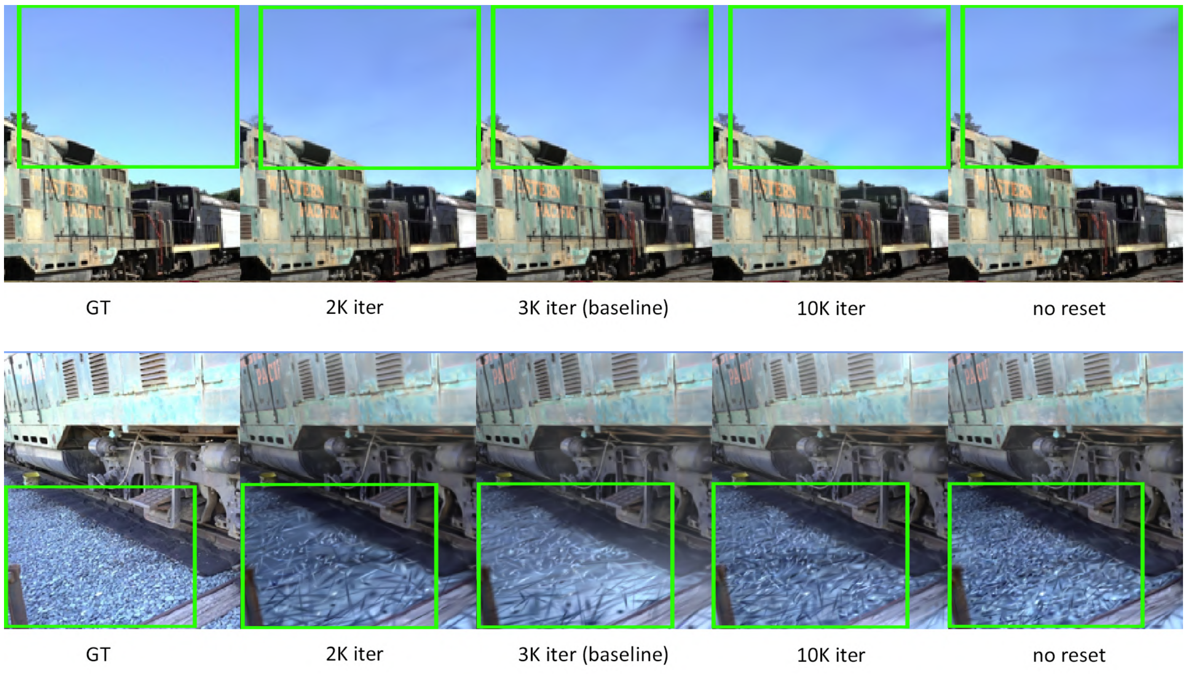


Figure 14: Close-up on two areas of the synthesized image from the train scene when changing the opacity reset interval. The top image shows a part of the sky where floaters are more visible when the opacity is not enough subject to a reset. The bottom image shows a part of the ground containing high frequencies, which benefits from having more Gaussians, hence when opacity is less often subject to a reset.

4 Add regularization strategies to 3DGS

These experiments allow us to better visualize how the method is affected by the variations of some hyperparameters. This opens the door to simple solutions that could overcome some limitations of the method. As observed in the previous experiments, even though anisotropic Gaussians can capture complex geometries, they can lead to unwanted visual artifacts. For example, large 3D Gaussians in regions that contain view-dependent appearance can cause popping artifacts where some elements instantly appear or vanish. Thus an interesting direction is to improve the optimization process to allow for a finer control over the spatial distribution and the management of the Gaussians. For this purpose, regularization strategies can be incorporated within the optimization procedure. In the next subsections, I present some regularization strategies I have implemented, along with other ideas that have not been tested.

4.1 Description and motivations

4.1.1 Edge Preservation Regularization

This strategy aims to maintain the integrity of edges in the synthesized images, which is crucial for perceptual quality and the overall accuracy of the geometry of the scene. The goal is to ensure that the edges in the synthesized images align with those in the target images and encourage optimization to preserve sharp transitions and avoid blurring or displacing important edges in the scene. Edge preservation can be encouraged by penalizing differences in edge maps obtained by an edge detection operator. The edge map can be easily computed with Sobel filters: $D = \sqrt{D_x^2 + D_y^2}$ where D_x and D_y are respectively approximations of the horizontal and vertical derivative on each point, computed by convolving the image with the Sobel derivative filters. The regularization term can be formulated as the mean absolute difference between the edge maps:

$$R_{edge} = \frac{1}{|P|} \sum_{p \in P} |D_{synth}(p) - D_{target}(p)|$$

where p index the pixels P of the image. In practice, this method is not computationally expensive as convolving is very fast on GPU, and edge maps can be pre-computed for target images. Thus, only the computation of the edge map of the synthesized images must be done during optimization.

4.1.2 Smoothness Regularization

In uniform areas of images, it is important to keep a homogeneous appearance without noise or artifacts. For this purpose, I added a smoothness regularization to help the reconstructed image exhibit a natural appearance in regions that should be visually smooth or homogeneous. This regularization can be effective in areas that do not contain significant texture or sharp edges, where noise or artifacts can be distracting and degrade the perceptive quality of the image. The Laplacian operator is a good candidate to achieve this. When applied to an image, the Laplacian will emphasize areas of sharp discontinuity, making it useful to enhance fine details. Due to its derivative nature, the Laplacian is sensitive to noise which can also make discontinuities like noise more prominent. So including the 2-norm of the Laplacian in the loss, the optimization is encouraged to find solutions where the second-order derivatives of the image intensity are minimized, which discourages rapid intensity changes and promotes smoothness in the reconstructed image. The regularization term can be formulated as:

$$R_{smooth} = ||\Delta I_{synth}||_2$$

However, this requires a careful balance to prevent over-smoothing of areas that must contain high-intensity changes (edges or fine details). This is why this regularization strategy can be counterbalanced with the edge regularization term that focuses on detail preservation. These two terms have a complementary objective: smoothness regularization minimizes unnecessary variations to promote visual coherence, and edge regularization focuses on preserving sharp edges and textures, which are critical for perceptual quality.

4.1.3 Variance Regularization

An issue which is discussed in the paper is that the method can create elongated artifacts. When the optimization creates large Gaussians, one can observe popping artifacts, which tend to happen in regions with view-dependent appearance. To mitigate this issue I added a regularization term that penalizes the mean of the maximum scaling coefficients of the 3-dimensional scaling vector. The primary purpose is to prevent Gaussians from becoming too large or too spread out in one direction. This term also encourages the Gaussians to remain compact and ensure that the scene is represented with a more detailed and precise arrangement of Gaussians. This regularization strategy can be formulated as

$$R_{var} = \frac{1}{N}\sum_{j=1}^{N} \max\left(s_j\right)$$

where the $\max\left(s_j\right)$ gets the largest coefficients in s_j. However, this term must be carefully handled as giving too much importance to it could prevent the model from capturing necessary spatial variations. It is also expected that this regularization implies the addition of more Gaussians in the scene, which must be controlled to avoid an unnecessary memory increase.

4.1.4 Binary Opacity Regularization

When navigating in a scene represented with 3DGS, one might notice ambiguous semi-transparent Gaussians leading to visual artifacts such as ghosting or blurring. To tackle this issue, an idea would be to reduce the uncertainty of each Gaussian's contribution to the scene and encourage each Gaussian to decisively contribute (be fully opaque) or not contribute (be fully transparent) to the final image, rather than existing in a semi-transparent state that might result in visual artifacts. For this, we can employ a binary entropy penalization of the opacity values of the Gaussians and express the regularization term with

$$R_{opacity} = \frac{1}{N}\sum_{j=1}^{N} H(o_j), \text{ where } H(p) = -p\log(p) - (1-p)\log(1-p).$$

This term encourages Gaussians to be either fully involved in the scene representation (opacity close to 1) or not at all (opacity close to 0).

4.1.5 Other ideas

An interesting idea would be to introduce a regularization term that enforces smoothness in the distribution and the features of 3D Gaussians across the scene. This spatial regularization term could be designed to target the consistency and gradual variation of the Gaussians in space. Encouraging similar Gaussians to have similar properties when they are close to each other in the scene could mitigate popping artifacts and enhance consistency in unseen regions. With this strategy, the Gaussians are not regularized from the image space as done with R_{edge} and R_{smooth} but directly in the world coordinate space. The spatial coherence can be enforced by targeting the distribution (position smoothness) and the orientation smoothness of the Gaussians. The position term is

$$R_{pos} = \sum_{j}\sum_{k\in N(g_j)} w_{ik}||\mu_j - \mu_k||^2$$

where $N(g_j)$ denotes the set of indices of neighboring Gaussians of g_j and $w_{ik} = \exp\left(-\eta||\mu_j - \mu_k||^2\right)$ is a weight that decreases with the distance between g_j and g_k, controlled by the decay rate η. Similarly for orientation smoothness, we can encourage adjacent Gaussians to have similar orientations, measured by the angle difference between the quaternions:

$$R_{rot} = \sum_{j}\sum_{k\in N(g_j)} w_{ik}\cos^{-1}(2\langle q_i, q_k\rangle^2 - 1).$$

The overall spatial regularization term would combine these two components. This method is however costly to implement as it requires finding the neighboring points of every point (based on spatial proximity or with k-nearest neighbors) while the positions are changing at each iteration. To lighten the computational cost, the neighbors can periodically be computed and stored to avoid the computation at every iteration, but it remains non-negligible. I did not test this regularization, but it would be interesting to see if it helps to improve the coherence and reduce artifacts, especially in regions not directly constrained by observed data. This idea of regularizing the position and rotation could be extended to a global feature regularization where close Gaussians should exhibit close properties in shapes and colors. It is however challenging to identify which Gaussians should be considered as representing similar features or regions.

4.2 Experimenting with the regularization terms

4.2.1 Finding the right time interval

Applying the 4 previous regularization terms throughout the training process is challenging, especially in a context that involves a dynamic procedure of densification. The strategy for when to apply these regularization terms can significantly impact the optimization effectiveness and efficiency. Thus, after manually finding the appropriate weight for each regularization term, I tried to apply each term individually at different time intervals to understand their impact.

For the edge regularization, I noticed that the appropriate interval was during the densification process, i.e., between 500 and 15K iterations. During densification, geometry is added or removed from the scene, hence it can be expected that penalizing sharp edges helps the method to place relevant Gaussians at appropriate locations. Applying this regularization after the densification also helps the model (although not as much), but increases the positional gradients. My interpretation is that the optimization is thus trying to displace the Gaussians to better fit the edges, at the expense of the other features that cannot be refined as precisely as when it is quasi-motionless.

For the smoothness regularization, applying it at every interval (pre/post-densification or during the whole optimization) was fruitful in terms of PSNR, LPIPS, and SSIM. Even without the edge regularization to counter the potential over-smoothing effect, penalizing the Laplacian seems to help the model to represent the scene. Hence, I placed it during almost all the training, i.e., between 500 and 30K iterations.

For the variance regularization, I found it better to apply it during the whole training or just after the densification process. As this penalization directly targets the Gaussian's scaling which is optimized during the whole training, it is sound that it works well at different intervals. For the next experiments, I applied it only on the post-densification interval.

For the opacity regularization, I have obtained mixed results. Applying it to the whole training or only at the beginning drastically impedes the results, with a lot of splotchy artifacts across the scene. Applying it in the post-densification phase leads to similar results to the baseline, with a slight increase in SSIM.

4.2.2 Finding the right combination with appropriate weights

The previous experiments allowed me to greatly reduce the number of possibilities to find how these regulation terms can be combined. Naturally, it is also necessary to adjust the weights of each term since the different regularization strategies interact with each other. Therefore, I experimented with various combinations using grid search and then progressively refined by tweaking the weights, ultimately narrowing it down to a shortlist of four combinations:

- $R = \lambda_{opacity} R_{opacity} + \lambda_{edge} R_{edge}$
- $R = \lambda_{var} R_{var} + \lambda_{edge} R_{edge} + \lambda_{smooth} R_{smooth}$
- $R = \lambda_{edge} R_{edge}$
- $R = \lambda_{smooth} R_{smooth}$

It is interesting to note that R_{edge} and R_{smooth} produces good results alone. The two other combinations consist of mixing the opacity and edge regularization, and the other one consists in combining all the regularization terms except the opacity. Finally, I launched several experiments on the 3 datasets with different lambdas to obtain a final combination that produces equivalent or better results on all the datasets. This combination is

$$R = \lambda_{var} R_{var} + \lambda_{edge} R_{edge} + \lambda_{smooth} R_{smooth}$$

with $\lambda_{var} = 10^{-4}$, $\lambda_{edge} = 5 \times 10^{-3}$ and $\lambda_{smooth} = 10^{-4}$.

4.3 Results

The numerical results presented in Table 2 are averaged over 3 runs on each dataset.

Method	PSNR ↑	SSIM ↑	LPIPS ↓	Nb points
Train scene				
Baseline	22.468 ± 0.068	0.8598 ± 0.0011	**0.1325 ± 0.0007**	798 068 ± 2539
w/ Reg.	**22.768 ± 0.093**	**0.8602 ± 0.0010**	0.1339 ± 0.0005	765 952 ± 3769
Truck scene				
Baseline	26.670 ± 0.016	0.9239 ± 0.0002	**0.0760 ± 0.0002**	1 529 402 ± 3351
w/ Reg.	**26.743 ± 0.030**	**0.9242 ± 0.0002**	0.0769 ± 0.0004	1 419 210 ± 9702
Reindeer scene				
Baseline	**39.247 ± 0.004**	**0.98856 ± 0.00001**	**0.01225 ± 0.000007**	209 933 ± 655
w/ Reg.	39.246 ± 0.002	0.98855 ± 0.00001	0.01233 ± 0.000012	208 046 ± 957

Table 2: Average results obtained on the 3 datasets with the baseline, and when adding the regularization. The standard deviation is displayed on the right-hand side of the average value. The best results are highlighted in bold for each metric and dataset.

To account for variability the standard deviation is also given. These results show the *slight* improvement of the regularization strategy to the base method. Overall, the results are better in terms of PSNR and SSIM on the outdoor scenes. The largest difference is on the train dataset which benefits from an increase of 0.3 in PSNR when using the regularization strategy. However, the results do not improve when using regularization on the reindeer dataset. This could be due to the simplicity of this scene which does not necessitate regularization to be learned correctly. However as the outdoor scenes are more complex and more subject to artifacts issues discussed previously, it is more interesting to see the impact. On each dataset, we can also notice that the number of Gaussians decreases when the regularization is used (even though the results are at least similar). This outcome is very interesting, especially for the Truck scene where 7% of the Gaussians are removed while slightly improving the PSNR. This effectively shows that the scene contains Gaussians that are not necessarily useful or not well optimized and that it is possible to have a more compact representation with the same number of Gaussians.

Visually, one can observe some improvements in the truck and train scenes. Figure 15 (top) shows that the geometry of the bench that is not well captured by the baseline can be accurately placed using the regularization strategy. In the previous experiments focused on the densification interval and densification threshold in subsection 3.4, we observed that it was necessary to add more Gaussians to the scene to capture this geometric feature. Here, with fewer Gaussians the edge on the middle of the bench is present and not distorted. Another example is displayed in Figure 15 (bottom), where we observe that the background hills are better reconstructed using regularization. There are more structures preserved in the hills with trees that are accurately placed.

4.4 Critical analysis of the approach

In practice, it is hard to verify the hypothesis made when designing the regularization strategies. The results are not significant enough to validate the approaches or demonstrate that the regularization improves the performance of the method. More careful experiments on datasets of several types should

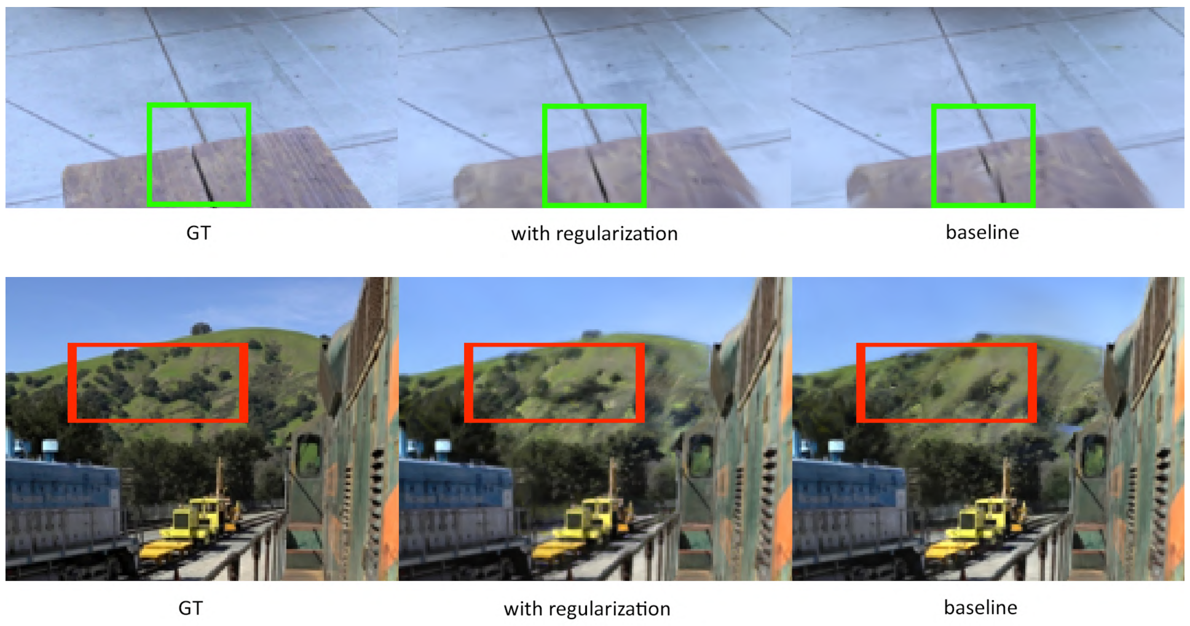


Figure 15: Close-up on synthesized images from the truck scene (top) and the train scene (bottom) when using the baseline method and when adding regularization to it. The regularization terms allow to keep more geometric features than the baseline while slightly reducing the number of Gaussians needed to represent the scene.

be carried out to assess the results. Given the time and computational power I had, the choice for the combinations and weights has probably not been studied enough and would require a more rigorous analysis. In particular, a more detailed analysis of the impact of each individual regularization term would help to better validate the hypothesis on which the regularization ideas rely. However, the choices that have been made are based on experiments conducted on the same 3 datasets and result in a slight improvement of the results for the outdoor scene while reducing the number of Gaussians, which still shows that regularizing 3DGS has a clear potential to improve the results and overcome some current limitations.

5 Limitations of 3DGS with current solutions

This section aims to highlight the main issues encountered with the 3DGS method and propose solutions to these problems given the very active literature on the subject.

3DGS struggles to model specular components within scenes. This primarily comes from the limited ability of low-order spherical harmonics to capture high-frequency information, which is often required in these scenarios. This makes it challenging for 3DGS to model scenes with reflection and specular components as we could observe in the different results presented in this report. This issue is starting to be tackled, a recent work of Yang et al., 2024 introduced the use of anisotropic spherical Gaussians appearance field instead of SH to model the view-dependent appearance of each 3D Gaussian. Their method improves the ability of 3DGS to model scenes with specular and anisotropic components without increasing the number of 3D Gaussians.

As stated previously, 3DGS tends to produce scenes with millions of Gaussians which is very costly to store in memory and increases the training/rendering time. The GPU runtime memory requirements during training and rendering require almost 20GB of GPU VRAM for several high-resolution scenes, which is impractical. This issue is the reason why I trained on lower-resolution images, which remains demanding in terms of memory, especially for the Truck scene. This limitation is often criticized in 3DGS and a growing number of publications is targeting this issue. Girish et al., 2023 proposed a technique that uses quantized embeddings to reduce the memory storage of the attributes. Their approach aims at compressing the color and rotation attributes via a latent quantization framework and quantizing the opacity attribute. Furthermore, Fan et al., 2024, proposed a method to identify Gaussians that are less significant in contributing to the scene reconstruction, and adopts a pruning and recovery process that effectively reduces redundancy in Gaussian counts while preserving visual effects. They also employ distillation and pseudo-view augmentation to distill spherical harmonics to a lower degree, allowing the transfer of important information while maintaining the scene's appearance. Finally, the same authors also propose to quantize all attributes with VecTree quantization to reduce the bandwidth representation with minimal accuracy loss. Their method allows them to achieve an average compression rate of 15x while boosting the rendering time.

Another common limitation of 3DGS is the floating artifacts from Gaussians which cannot be removed during the optimization. Standard training of 3DGS proceeds by computing the loss over the full image resolution, which results in a complex loss landscape as the Gaussians are forced to fit the fine features of the scene early in the training (resulting in overfitting). As the SfM initialization is sparse and the attributes are initialized with rough estimates, the optimization can be suboptimal, resulting in these floating artifacts. To counter this issue, many contributions including Yang et al., 2024 and Girish et al., 2023 proposed a coarse-to-fine strategy by initially rendering at a small scene resolution and gradually increasing the size until reaching the target resolution. This enables 3DGS to learn global information from the images in the early stages of training, and therefore reduces overfitting to local areas of the training images, and eliminates a significant number of floaters in the novel views.

The quality of the novel views with 3DGS heavily depends on the amount of overlapping images in the training dataset. Uncommon points of view produce a coarse appearance with visible Gaussians as seen in Figure 14 (bottom). Sampling strategies in the dataset could be studied to overcome these issues. This could however lead to overfitting resulting in visual artifacts. Chung et al., 2024 proposed a depth-guided optimization strategy that enables optimization of the scene with very few images, while mitigating the overfitting issue. The overfitting problem is managed by incorporating an early stop strategy when their depth-guided loss starts to rise. Moreover, they achieve stability by applying a smoothness constraint that ensures that neighbor 3D points have similar depths. This method allows to obtain stable results in a few-shot image setting.

The previous issue mentioned the over-reconstruction problem that often arises when training 3DGS. Over-reconstruction happens during densification where high-variance image regions are covered by a few large Gaussians only which leads to blur and artifacts in the rendered image (Figure 14, bottom). Zhang et al., 2024 notice that the over-reconstruction phenomena can be detected by the discrepancy between the frequency spectrum of the rendered image and the ground truth. Hence, they introduce a frequency annealing technique to achieve progressive frequency regularization from low-frequency to high-frequency signals. This strategy complements the pixel-level L1 loss, mitigates the over-reconstruction, and greatly improves the densification process.

Finally, the whole method relies on SfM to initialize the scene with a sparse point cloud and rough information of color and position. This initialization allows to produce a coarse approximation of the underlying distribution necessary to represent the scene as observed Figure 5. When performing benchmarks, we typically use datasets that already contain the output of the SfM algorithm (such as COLMAP). In a real-world application, it is however necessary to run the SfM algorithm on the images, which can take a few hours when having hundreds of images. Hence, even though training 3DGS to represent a scene takes about 40 minutes, the whole process necessitates several hours. It remains possible to train 3DGS on a randomly initialized point cloud. In their ablation study, the authors suggest to uniformly sample points within a cube of size three times the extent of the input camera's bounding box to initialize the scene. This results in a loss of 4-5 dB in PSNR in general. To remove the need for this costly initialization, Jung et al., 2024 proposed to initialize the scene with random sparse Gaussians with large variance and progressively apply a low-pass filter control. This effectively guides the Gaussians to learn a coarse approximation of the scene. This eliminates the need to run SfM to train 3DGS, which is particularly convenient for practical applications.

6 Conclusion

This project was dedicated to a comprehensive analysis of the 3D Gaussian Splatting method. The additional experiments highlighted the importance of the different components of the method, while emphasizing trade-offs that must be considered to improve image quality without increasing memory costs. In response to some of the observed limitations, I proposed extensions through regularization strategies. Across the three datasets considered, these strategies led to slight improvements while reducing the number of Gaussians needed to represent the scene. Finally, I discussed various limitations of 3DGS, and provided recent advances in the literature that have started to alleviate these issues.

Working on 3D Gaussian Splatting was very enriching, especially exploring the literature that demonstrates the popularity of the field (more than 60 papers related to Gaussian Splatting for CVPR 2024). Even though some of the results obtained from the experiments I have carried out were expected, performing the experiments and spending time to examine the qualitative and quantitative results helped me to really understand the method and its inner workings. Moreover, the project allowed me to grasp the original codebase of 3DGS, in particular by adding features for the benchmarks. Given the time limitations, I was only able to implement my initial thoughts on regularization. However, I would have been interested in exploring some of the concepts mentioned in the method's limitations to fully assess their effects.

References

J. Chung, J. Oh, and K. M. Lee. Depth-regularized optimization for 3d gaussian splatting in few-shot images, 2024.

Z. Fan, K. Wang, K. Wen, Z. Zhu, D. Xu, and Z. Wang. Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps, 2024.

S. Girish, K. Gupta, and A. Shrivastava. Eagles: Efficient accelerated 3d gaussians with lightweight encodings, 2023.

J. Jung, J. Han, H. An, J. Kang, S. Park, and S. Kim. Relaxing accurate initialization constraint for 3d gaussian splatting, 2024.

B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), 2023.

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. *Communications of the ACM*, 65(1): 99–106, 2021.

Z. Yang, X. Gao, Y. Sun, Y. Huang, X. Lyu, W. Zhou, S. Jiao, X. Qi, and X. Jin. Spec-gaussian: Anisotropic view-dependent appearance for 3d gaussian splatting, 2024.

J. Zhang, F. Zhan, M. Xu, S. Lu, and E. Xing. Fregs: 3d gaussian splatting with progressive frequency regularization, 2024.

A Qualitative results

As most of the results presented in the report consist of zoomed close-ups on specific parts of the synthesized images, this appendix provides multiple qualitative results of the baseline on the 3 datasets.

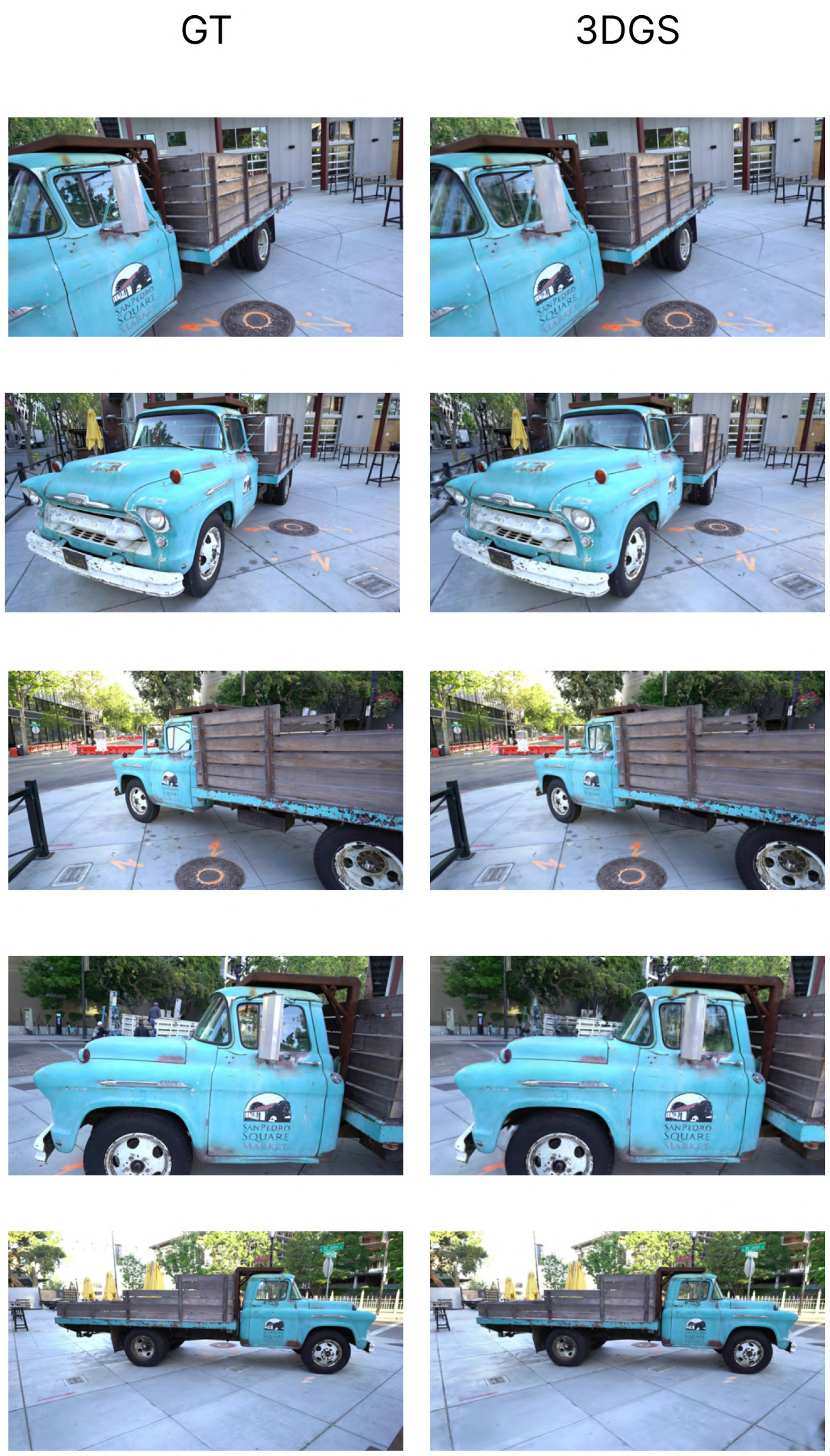


Figure 16: Comparing five different views of the truck scene coming from the test set, with the ground truth on the left and the synthesized image from the baseline 3DGS on the right.

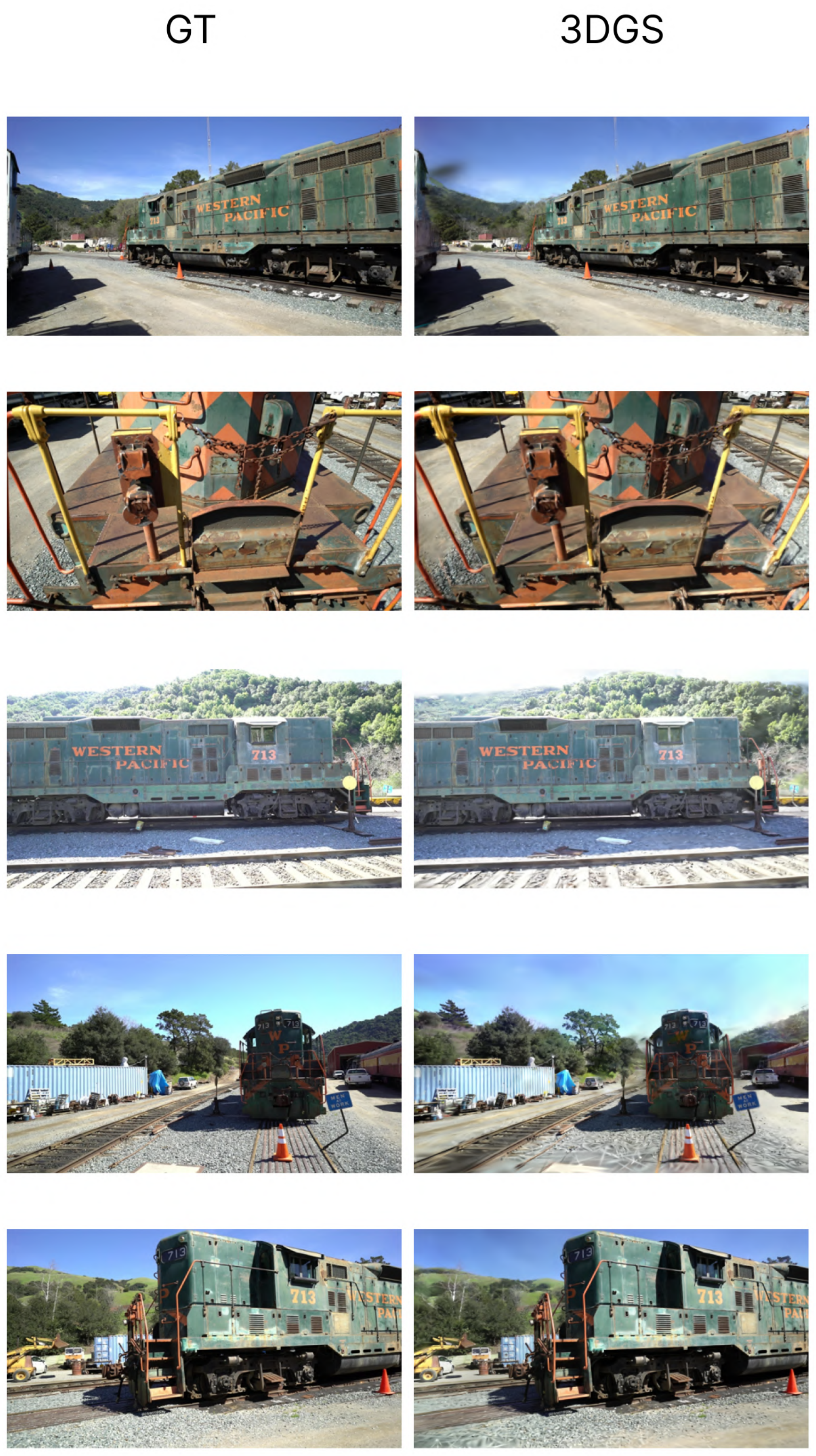


Figure 17: Comparing five different views of the train scene coming from the test set, with the ground truth on the left and the synthesized image from the baseline 3DGS on the right.

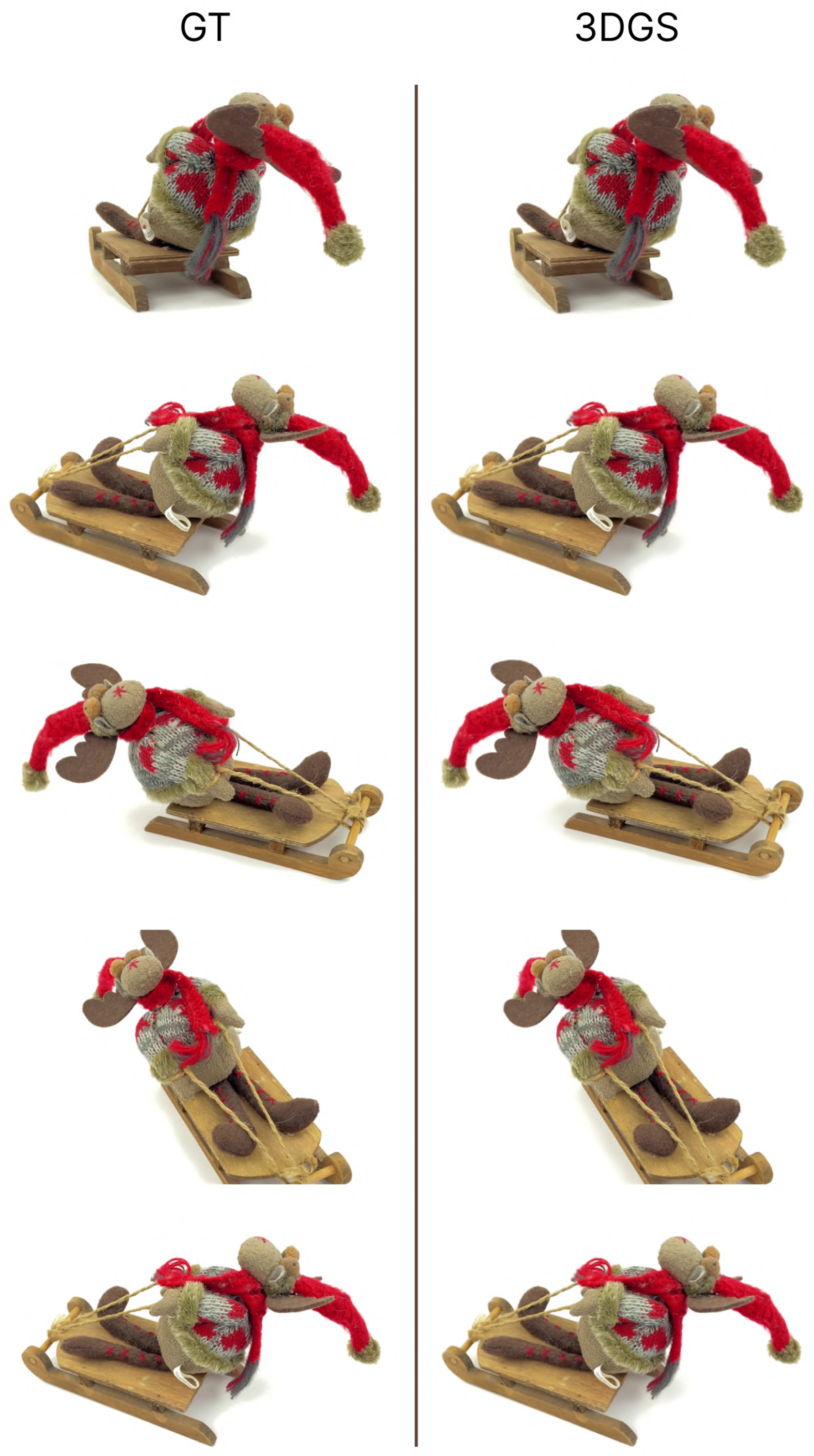


Figure 18: Comparing five different views of the reindeer scene coming from the test set, with the ground truth on the left and the synthesized image from the baseline 3DGS on the right.

B Raw results from experiments

B.1 Experiments on the loss

Experiment	PSNR ↑	SSIM ↑	LPIPS ↓	Nb of Gaussians
(L1 + SSIM) baseline	**26.647985**	0.922565	0.076046	1 510 861
Huber + SSIM	26.172070	0.926276	0.075745	743 446
L1	26.174578	0.898513	0.112726	774 592
L2 + SSIM	26.404264	0.926753	0.074196	841 686
SSIM	26.107460	**0.931388**	**0.055982**	4 277 740

Table 3: Results for the loss experiments on the truck scene

Experiment	PSNR ↑	SSIM ↑	LPIPS ↓	Nb of Gaussians
(L1 + SSIM) baseline	22.544100	0.861008	0.131800	800 873
Huber + SSIM	22.508022	0.865237	0.134692	441 708
L1	22.158302	0.795998	0.214835	336 520
L2 + SSIM	**22.897047**	0.869103	0.130015	506 307
SSIM	21.850454	**0.865966**	**0.111654**	2 995 382

Table 4: Results for the loss experiments on the train scene

Experiment	PSNR ↑	SSIM ↑	LPIPS ↓	Nb of Gaussians
(L1 + SSIM) baseline	39.243313	0.988582	0.012221	211 537
Huber + SSIM	38.242168	0.986442	0.015482	93 632
L1	38.135418	0.981649	0.019886	98 310
L2 + SSIM	38.595188	0.987414	0.014079	117 689
SSIM	**39.365574**	**0.991093**	**0.009524**	560 093

Table 5: Results for the loss experiments on the reindeer scene

B.2 Experiments on the spherical harmonics

Experiment	PSNR ↑	SSIM ↑	LPIPS ↓	Nb of Gaussians
degree 3 (baseline)	26.647985	0.922565	0.076046	1 510 861
degree 3 (no warm-up)	**26.724007**	**0.923987**	**0.075848**	1 539 067
degree 2	26.471998	0.922122	0.078079	1 529 672
degree 1	26.227222	0.919289	0.081218	1 538 324

Table 6: Results for the spherical harmonics experiments on the truck scene

Experiment	PSNR ↑	SSIM ↑	LPIPS ↓	Nb of Gaussians
degree 3 (baseline)	22.544100	0.861008	0.131800	800 873
degree 3 (no warm-up)	**22.633924**	**0.861242**	**0.131022**	807 335
degree 2	22.426823	0.856821	0.137416	812 406
degree 1	22.190079	0.852201	0.142635	800 192

Table 7: Results for the spherical harmonics experiments on the train scene

Experiment	PSNR ↑	SSIM ↑	LPIPS ↓	Nb of Gaussians
degree 3 (baseline)	39.243313	0.988582	0.012221	211 537
degree 3 (no warm-up)	**39.283443**	**0.988707**	**0.012093**	212 117
degree 2	38.440880	0.986975	0.013453	216 937
degree 1	37.650005	0.984843	0.015026	219 719

Table 8: Results for the spherical harmonics experiments on the reindeer scene

B.2.1 Experiments on the densification duration

Experiment	PSNR ↑	SSIM ↑	LPIPS ↓	Nb of Gaussians
10K	26.595974	**0.924219**	**0.074808**	1 305 413
15K (baseline)	26.647985	0.922565	0.076046	1 510 861
20K	**26.654917**	0.923078	0.079533	1 442 306
25K	26.563126	0.921424	0.082907	1 395 042
30K (no limit)	26.333935	0.919807	0.086161	1 470 825

Table 9: Results for the duration of the densification experiments on the truck scene

Experiment	PSNR ↑	SSIM ↑	LPIPS ↓	Nb of Gaussians
10K	**22.747423**	0.860630	0.132781	652 434
15K (baseline)	22.544100	**0.861008**	**0.131800**	800 873
20K	22.434734	0.856038	0.139983	691 706
25K	22.290501	0.848872	0.150131	598 076
30K (no limit)	21.755592	0.840955	0.158640	643 227

Table 10: Results for the duration of the densification experiments on the train scene

Experiment	PSNR ↑	SSIM ↑	LPIPS ↓	Nb of Gaussians
10K	**39.270580**	**0.988677**	**0.012070**	209954
15K (baseline)	39.243313	0.988582	0.012221	211 537
20K	39.071365	0.987949	0.012960	177 994
25K	38.842945	0.987169	0.013929	156 299
30K (no limit)	38.707073	0.986700	0.014556	152 463

Table 11: Results for the duration of the densification experiments on the reindeer scene

B.2.2 Experiments on the gradient threshold for densification

Experiment	PSNR ↑	SSIM ↑	LPIPS ↓	Nb of Gaussians
0.0001	26.566711	**0.925132**	**0.067002**	3 593 807
0.0002 (baseline)	**26.647985**	0.922565	0.076046	1 510 861
0.0003	26.601210	0.921538	0.082096	894 745
0.0005	26.262461	0.915628	0.091398	454 455

Table 12: Results for the gradient threshold experiments on the truck scene

Experiment	PSNR ↑	SSIM ↑	LPIPS ↓	Nb of Gaussians
0.0001	22.508377	**0.864403**	**0.119631**	1 944 437
0.0002 (baseline)	**22.544100**	0.861008	0.131800	800 873
0.0003	22.533855	0.853205	0.146825	462 560
0.0005	22.505777	0.842627	0.170406	240 328

Table 13: Results for the gradient threshold experiments on the train scene

Experiment	PSNR ↑	SSIM ↑	LPIPS ↓	Nb of Gaussians
0.0001	**39.821888**	**0.990576**	**0.009809**	523 716
0.0002 (baseline)	39.243313	0.988582	0.012221	211 537
0.0003	38.765541	0.986687	0.014602	120 970
0.0005	38.138741	0.984098	0.018026	67 496

Table 14: Results for the gradient threshold experiments on the reindeer scene

B.2.3 Experiments on the densification interval

Experiment	PSNR ↑	SSIM ↑	LPIPS ↓	Nb of Gaussians
50	26.637577	**0.924460**	**0.072283**	2 205 865
100 (baseline)	**26.647985**	0.922565	0.076046	1 510 861
200	26.633980	0.922971	0.079324	1 084 112
500	26.376322	0.919454	0.084631	664 314

Table 15: Results for the densification interval experiments on the truck scene

Experiment	PSNR ↑	SSIM ↑	LPIPS ↓	Nb of Gaussians
50	22.409729	0.860704	**0.127242**	1 220 841
100 (baseline)	22.544100	**0.861008**	0.131800	800 873
200	**22.637451**	0.857110	0.141265	536 244
500	22.540956	0.848448	0.157706	318 474

Table 16: Results for the densification interval experiments on the train scene

Experiment	PSNR ↑	SSIM ↑	LPIPS ↓	Nb of Gaussians
50	**39.468884**	**0.989395**	**0.011201**	299 452
100 (baseline)	39.243313	0.988582	0.012221	211 537
200	39.030697	0.987740	0.013309	152 046
500	38.723915	0.986513	0.015045	102 228

Table 17: Results for the densification interval experiments on the reindeer scene

B.2.4 Experiments on the opacity reset interval

Experiment	PSNR ↑	SSIM ↑	LPIPS ↓	Nb of Gaussians
2K	**26.652332**	0.922726	0.079380	1 174 379
3K (baseline)	26.647985	0.922565	0.076046	1 510 861
10K	26.532238	0.923979	0.074855	1 933 962
30K (no reset)	26.516413	**0.924296**	**0.073123**	2 336 817

Table 18: Results for the opacity reset interval experiments on the truck scene

Experiment	PSNR ↑	SSIM ↑	LPIPS ↓	Nb of Gaussians
2K	22.616619	0.856919	0.139926	613 514
3K (baseline)	22.544100	0.861008	0.131800	800 873
10K	22.394600	0.861288	0.123190	1 079 217
30K (no reset)	**22.662132**	**0.864709**	**0.119443**	1 432 080

Table 19: Results for the opacity reset interval experiments on the train scene

Experiment	PSNR ↑	SSIM ↑	LPIPS ↓	Nb of Gaussians
2K	39.104179	0.988086	0.012831	178 894
3K (baseline)	39.243313	0.988582	0.012221	211 537
10K	39.404606	0.989112	0.011520	258 303
30K (no reset)	**39.485565**	**0.989375**	**0.011238**	293 716

Table 20: Results for the opacity reset interval experiments on the reindeer scene