3D Gaussian Splatting with Diffusion Priors

Fredrik Ekholm
fwe21@cam.ac.uk

Guillem Tarrach
gg457@cam.ac.uk

Jonah Miller
jm2399@cam.ac.uk

Abstract

We propose and evaluate a method for regularising 3D Gaussian Splatting (3DGS) [1] scenes using a denoising diffusion model for RGBD images. Our goal is to increase the accuracy and performance of 3DGS when using a small (around 3 or 9) set of training images. Our method follows closely that of DiffusioNeRF [2] - where Neural Radiance Fields [3] are combined with with a generative diffusion model that serves as a learnt scene prior.

1 Introduction and Motivation

1.1 Introduction to the problem

3D modelling is the computational construction of a 3D scene. It can often be done from a finite collection of images leading to far-reaching applications. 3D modelling in medicine, for example, has become a very active field. 3D ultrasounds have allowed doctors to understand and diagnose abnormalities in fetal development through gaining a much more comprehensive view of the organs and systems involved [4]. Compared to using multiple 2D ultrasounds and mentally composing them, such a system saves on procedure time and increases diagnosing accuracy. The mechanism behind this is a scanner which logs the precise orientation and geometry of rapidly taken 2D images [5]. This can then form an accurate 3D model, as it has images from all necessary angles.

The capturing, rendering, and synthesising of these images together has a high computational cost due to the sheer amount of images needed to describe an entire system. For this reason, applying machine learning techniques to this problem can be very fruitful. It allows the model to be generated on significantly fewer images, with the unseen views being inferred by a machine learning algorithm. This will have knock on effects such as reducing the cost for the mechanical scanning device, as it won't need the precision needed for rapid-fire picture taking.

This is a microcosm for the field, and displays how much potential there is for machine learning methods to revolutionise 3D modelling in terms of efficiency, accuracy and cost. The process for generating novel views is referred to as 3D view synthesis - the construction of unseen views of a scene give a fixed number of real views.

Ever since their introduction in 2020, Neural Radiance Fields (NeRFs) [3] have been a popular method as the core of 3D reconstruction algorithms. They have given rise to a multitude of work on fit-for-purpose reconstruction models as well as generative modelling.

NeRF generates a fully connected deep neural network, which when queried with a position in the scene and a view angle, outputs the volume density and emitted radiance of the view. Sampling along a camera ray allows the use of a differentiable renderer to reconstruct camera images. The network can then be trained with gradient descent to overfit on these training images and in turn create a synthetic, novel view of the unseen.

However, the novel views will often have some errors. This arises from issues such as poor lighting, variable apertures or reflective surfaces giving different perspectives from the training images taken near to each other. The symptoms of this are displayed as generated unphysical structures or blurred

objects seen from the novel views. This issue is amplified with fewer training images, where objects are often not placed in the right location.

Generative modelling can be used to regularise the novel views by introducing prior statistics about the scene. These statistics relate to aspects such as the geometry and the colour of a scene. The approach involves adding the probabilistic prior to the general photometric loss function in the NeRF's MLP. This helps alleviate the error the neural network generates from using an insufficiently detailed training set.

1.2 Background and related work

3D Gaussian Splatting (3DGS), proposed by Kerbl et al. [1] in 2023, demonstrates "state-of-the-art visual quality" by representing a scene as a synthesis of 3D anisotropic Gaussians. These Gaussians are then optimised through an iterative process using an adaptive density control function and a differentiable renderer. The resulting training times are significantly faster than NeRF methods such as Mip-NeRF 360 [6]. This is because 3DGS does not use an MLP, which has a high time complexity from the sampled camera rays' views being rendered. Moreover, this time save allows 3DGS to be the only model with real-time rendering of unbounded scenes.

PixelNeRF [7] aims to build the radiance field from a few training images. It takes a small number of input images of a bounded scene and computes a convolutional grid aligned with the image pixels. This is then used as the scene prior for a NeRF network allowing more optimised volumetric density and colour to be outputted. It displays significantly better results than non-specialised NeRF models trained on few images. However, it has only been trained and tested on simple, bounded scenes rather than more complicated or unbound scenes. It also uses a simple scene prior based on the input images, not more intricate methods as displayed below. We use this to demonstrate the potential for combining 3DGS and a scene prior to allow unbounded scene generation from few images. We also aim to show the improvement our method gives from bounded scene generation as we compare our results to pixelNeRF.

Denoising Diffusion models (DDM) were first introduced in 2020 by Ho et al. [8]. They use a Markov chain to reduce noise from a random pixel sample until it matches the target image. They show comparable – if not increased – quality in their output compared to other generative models as well as being more efficient to train. They are not necessarily better than normalising flow [9] models (such as those used in RegNeRF [10]) at generating log-likelihoods, but they are often more straightforward to work with.

We have chosen to base our approach on DiffusioNeRF, introduced by Wynn et al. [2] in 2023. It is a method for regularising the radiance fields of a NeRF using a DDM pretrained on RGBD patches as a scene prior. Novel views generated by the radiance field have an estimate of the gradient of the log likelihood of their occurrence assigned to them by the DDM. This gradient is then fed into the loss function to allow better colour and geometry estimates going forward.

1.3 Overview of the idea

Our goal is to adapt the regularisation techniques introduced in DiffusioNeRF to the setting of 3DGS. Our motivation stems from a desire to combine the significantly faster training and rendering times of 3DGS with the accuracy of regularised NeRF models.

In more concrete terms, we use a DDM to enforce learnt priors on the scene reconstructed with 3DGS. We also introduce additional regularisation methods. We show that these regularisation methods help improve the accuracy of 3DGS for reconstructing both bounded and unbounded scenes from a small number of images.

2 Methods

2.1 Baseline algorithm

We build our work on 3DGS radiance fields [1]. In this section, we give an overview of this algorithm. The input to the algorithm is a set of images of a single static scene. Structure-from-Motion (SfM) [11] is used to create a sparse point-cloud and to calibrate the cameras. The latter is to estimate the

image parameters. These are both intrinsic (such as focal length) and extrinsic (such as position and direction).

In 3DGS, the scene is represented as a union of 3D Gaussians, whose parameters –position, opacity and anisotropic covariance, as well as spherical harmonics coefficients, which account for directional differences– are optimised during training. These are initialised to be centered at the points in the above point-cloud. At each training iteration, a camera is chosen and a differentiable rasteriser renders the scene from that camera. The rendered image is then compared to the original image and the parameters of the Gaussians are updated by gradient descent to minimise the L_1 difference between them.

The algorithm also involves an adaptative control of the number of Gaussians. Every 100 iterations after a certain number of warm-up iterations, all Gaussians that are essentially transparent are removed and a densification process occurs. Some areas of the scene may be missing geometric detail, while in others, the areas that some Gaussians cover may be too large. The densification process is designed to solve these problems by dividing the Gaussians in these regions into multiple ones, as illustrated in Figure 1. In the original algorithm, every 3000 iterations all the opacities are set to be close to 0. Gradient descent will increase the opacity back for Gaussians that contribute to reconstructing the scene, while all others will be removed. However, as is explained in Section 2.3.3 below, for a small number of images this results in significantly worse results. Thus, we have decided to omit this step in the baseline.

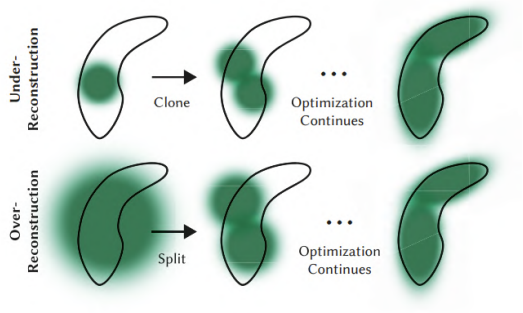

Figure 1: Illustration of the Gaussain densification process (image reproduced from [1]).

2.2 Algorithm improvements

We have experimented with several regularisation methods for 3DGS. The outline of our approach is summarised in Figure 2. The main regularisation method consists of using a DDM to enforce scene priors on the reconstructed scene. We start by rendering the reconstructed scene from a novel angle. We then feed the rendered image into a pretrained DDM, and modify the parameters of the Gaussians in the scene in the direction suggested by the DDM. When the scene is reconstructed from a small number of images, several artifacts will appear, such as objects not having the correct size or being in the right location. The DDM should detect these non-sensical artifacts in the novel views, and try to remove or alter them as appropriate. We have used additional regularisation methods that we will motivate and discuss in more detail below.

2.3 Implementation details

We modify the open source implementation of 3DGS from the original paper[1], adding the code for DDM-regularisation from the DiffusioNeRF implementation [2] and other regularisation methods, that we describe below. Our implementation can be found at `https://github.com/Guillem-Fredrik/gaussian-splatting-with-priors`.

[1] `https://github.com/graphdeco-inria/Gaussian-splatting`
[2] `https://github.com/nianticlabs/diffusionerf`

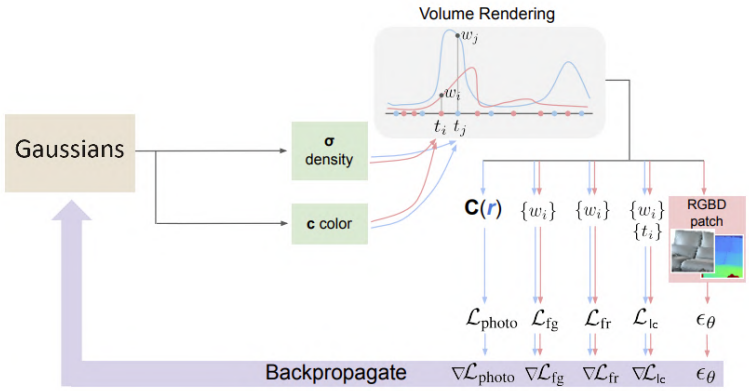

Figure 2: Outline of the approach (image modified from [2]).

2.3.1 Denoising Diffusion Model

We use the pre-trained RGBD diffusion model published as part of DiffusionNeRF [2]. As explained in Section 2.2, we perturb a training camera, render the scene from the resulting camera and input the outcome to the diffusion model. The DDM outputs the predicted noise in the rendered image, and we modify the parameters of the Gaussians in the direction of such noise. This is equivalent to adding the following loss term:

$$L_{DDM} = -\sum_{p \in \text{Pixels}} \mathbf{x}_p \cdot \epsilon_p(\mathbf{x}),$$

where $\mathbf{x}$ is the rendered RGBD image and $\epsilon_p(\mathbf{x})$ is the predicted RGBD noise at pixel p. For computing gradients, we consider $\epsilon_p(\mathbf{x})$ as a constant, so that

$$\nabla_{\mathbf{x}_p} L_{DDM} = -\epsilon_p(\mathbf{x}).$$

In DiffusionNeRF [2], the cameras are perturbed by translating and rotating the cameras by amounts sampled from uniform distributions with predetermined ranges. We found that this did not give good results for some scenes due to their different sizes. Instead, we propose the following method for perturbing the cameras: After a warm-up period, every 100 iterations we compute a weighted average of the depths in the reconstructed scene for each camera. When perturbing a camera, we then simply rotate around the point in the direction that the camera is pointing with the computed depth, again by an angle sampled from a uniform distribution.

2.3.2 Depth Rendering

The differentiable rendered from the original 3DGS implementation only supports rendering RGB images. As the DDM model takes an RGBD-image as input, we need to implement depth rendering. It is not obvious how to define depth when the scene is represented volumetrically with non-opaque volumes. DiffusionNeRF [2] calculates depth as the average distance to camera weighted by visibility, similarly to how colours are calculated. We choose to use the same method, in hope to improve compatibility with the DiffusionNeRF DDM model. To simplify implementation we assume a constant depth for each Gaussian that is equal to the distance from its center to the camera. We then simply treat depth as an additional colour channel and proceed with the normal rendering procedure.

2.3.3 Opacity reset

An important step in the adaptative control of the number of Gaussians in 3DGS, as described in Section 2.1, is resetting the opacities of the Gaussians at a regular interval. When the number of images is small, we have observed that this gives significantly worse results. Some of the Gaussians in the foreground tend to get repurposed to represent objects in the background. For this reason, we have omitted this step in our experiments.

2.3.4 Additional proposed regularisation methods

We refer to the original loss function of 3DGS as the *photometric loss*. We propose additional loss functions to acts as regularisation:

Foreground Regularisation To prevent the model from leaving low-opacity holes in the scene and relying on the background for correct colour, we want the total opacities σ_p of each pixel to be close to 1. To achieve this, we add

$$L_{fg} = \sum_{p \in \text{Pixels}} (\sigma_p - 1)^2$$

σ_p is calculated by adding the opacity of each Gaussian as an additional colour channel before rendering.

Frustum regularisation When training with a small number of images, the reconstructed scene will occasionally contain a copy of some of the objects in front of each training camera. While the diffusion model helps avoid this, additional regularisation can help it achieve this in less iterations. For this reason, we have added frustum regularisation following DiffusioNeRF. This penalises Gaussians for appearing in less than two of the training camera frustums. More explicitly, the frustum regularisation loss is defined as

$$L_{frustum} = \sum_{i \in \text{Gaussians}} w_i \cdot 1_{F_i < 2},$$

where w_i is the opacity of the i-th Gaussian and F_i is the number of training frustums containing its center. We note that it is possible for the center of a Gaussian to lie outside of a frustum while filling a large portion of the corresponding camera. However, the adaptative control of the number of Gaussians in the baseline 3DGS algorithm should prevent this. Additionally, frustum regularisation will also prevent fake copies of the whole scene being reconstructed in front of each training camera which could be undetectable to the DDM.

Lenticular regularisation In our early experiments, we observed that Gaussians tended to become flat in the direction of one of the training cameras. This made it invisible to a subset of the cameras. Similar to a lenticular lens, this allowed objects in the reconstructed scene to display different images when viewed at different angles. This resulted in reconstructed objects not being in the correct positions. To prevent this, we propose adding the following regularisation term:

$$L_{lc,1} = \sum_{j \in \text{TrainCamera}} \sum_{i \in \text{Gaussians}} w_i \left(\log(\varepsilon + A_i^2) - \log(\varepsilon + A_{i,j}^2) \right),$$

where $A_{i,j}$ is the area of the i-th Gaussian's orthogonal projection to the plane perpendicular to the direction of the j-th camera, and A_i is the area of the i-th Gaussian's orthogonal projection to the plane that maximises it.

We found that, when including this, the Gaussians tended instead to become very thin which resulted in similar problems. To prevent this, we considered

$$L_{lc,2} = \sum_{j \in \text{TrainCamera}} \sum_{i \in \text{Gaussians}} w_i \left(\log(\varepsilon + B_i^2) - \log(\varepsilon + B_{i,j}^2) \right),$$

where now $B_{i,j}$ is the length of the i-th Gaussian's orthogonal projection to the line in the direction to the direction of the j-th camera, and B_i is the length of the i-th Gaussian's orthogonal projection to the line that maximises it.

We defined lenticular loss as $L_{lc} = L_{lc,1} + L_{lc,2}$. We observed qualitatively that it tended to make the Gaussians more round in general, which was helpful in the beginning to prevent the phenomena above, but it resulted in worse results later on. For this reason it was given a small weighting in our experiments.

2.4 Data pipelines

A 3D reconstruction dataset typically consists of a set of around 30 to 300 images of a scene, together with camera calibration parameters and sparse 3D point cloud as estimated by COLMAP [12, 13]. The images are split into train and test sets. We aim to evaluate our approach using fewer images, and thus have to modify the datasets slightly. We ignore the original train-test split and instead manually select either 3 or 9 images to use for training and a separate set of images for testing. As the point cloud is calculated from all available images, we filter it based on what points are visible from the chosen training images. This more accurately simulates what initialisation information could be reconstructed from the training images. However, we use the provided camera calibration parameters which are calculated using all images. In practice we found that the parameters can be estimated accurately enough even with few images.

2.5 Training and testing procedures

Our implementation builds on that of 3DGS and we use the same training logic. The implementation is mainly written in PyTorch, with the differentiable Gaussian renderer written in CUDA for both forward and backward passes. We use an Adam optimiser with different learning rates for the different parameters of the Gaussians. The learning rate for the positions of the Gaussians follows an exponential decay schedule, while the learning rate for the other parameters are constant. The weights for the different proposed regularisation terms follow a linear schedule.

We used the scenes listed in Section 3.1 for training and testing during the development of the project. We selected a set of 3 or 9 images in the datasets to use for training and a set of the remaining images for testing.

For each scene, we ran 10 tests with the hyperparameters for the weights and different regularisation losses randomly generated from distributions that we deemed were sensible from earlier experiments. We present our scenes from the setup that gave the highest PSNR score compared to the ground truth. In order for this to be computationally feasible with our available resources, we trained each scene for 10,000 iterations. The 3DGS state-of-the-art results came from using 30,000 iterations.[1].

3 Experiments and evaluation

3.1 Datasets

We have used the following datasets:

- DTU [14], a set of real indoor bounded scenes used in pixelNeRF.
- LLFF [15], a collection of synthetic and real images depicting indoor scenes.
- Tanks&Temples [16], a group of indoor and outdoor environments.
- Mip-NeRF 360 [6], which depicts both indoor and outdoor scenes.

In particular, we evaluated our model on the following scenes.

- 19 scans from the DTU dataset. We used 9 training images for each. We separated these 19 scans into two sets. The first contains a general set of DTU images and form a baseline of the DTU results we expect. The second contains the set of DTU images that both pixelNeRF, DiffusioNeRF as well as others form their results on. This is the dataset we use to compare our results to theirs. The full dataset has 15 images, however we could only test on 9 due to a COLMAP reconstruction failure.
- `bonsai_3` and `bonsai_9` from the Mip-NeRF 360 [6] dataset, depicting an indoor scene with a LEGO bonsai in the middle and a bike and musical instruments in the background. `bonsai_3` uses 3 training images from similar angles and `bonsai_9` uses 9 images from different angles.
- `truck` from the Tanks&Temples dataset, depicting a truck in a street. This is an unbounded scene and we used 3 training images.

Additionally, we have created our own custom dataset, with the following scenes, each of which consists of 3 training images:

- `guillem`, depicting one of the authors of the project.
- `chairs`, depicting an indoor scene with two chairs stacked on top of each other.
- `rooftops`, depicting rooftops in Cambridge. This is an unbounded scene.

3.2 Qualitative results

Overall, the our methods perform better than the baseline 3DGS algorithm, but not significantly so. Let us discuss in more detail the observed effects of regularisation.

3.2.1 Better geometry reconstruction

Our regularisation methods result in more accurate locations for the objects in the scene, especially for the scenes with 3 training images. See Figure 3 for an example. This is especially noticeable when comparing depth maps for the baseline and regularised reconstructions, as shown for example in Figure 4.

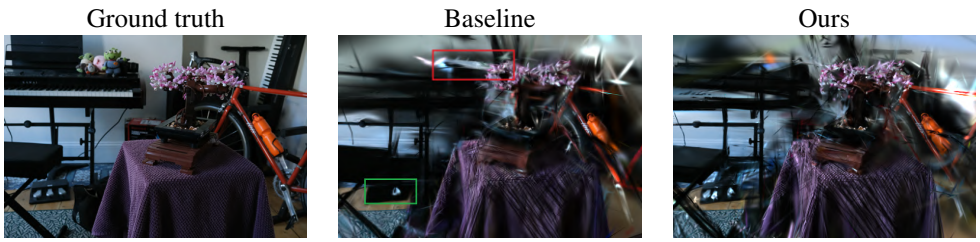

Figure 3: Example test image for `bonsai_3`. In the baseline reconstruction, the keys of the keyboard are in the wrong location (red rectangle). Moreover, some of the pedals are missing (green rectangle). Analysing the scene with an interactive 3D viewer shows that the missing pedals are under the floor and not visible from this view. These errors are not present in our reconstruction.

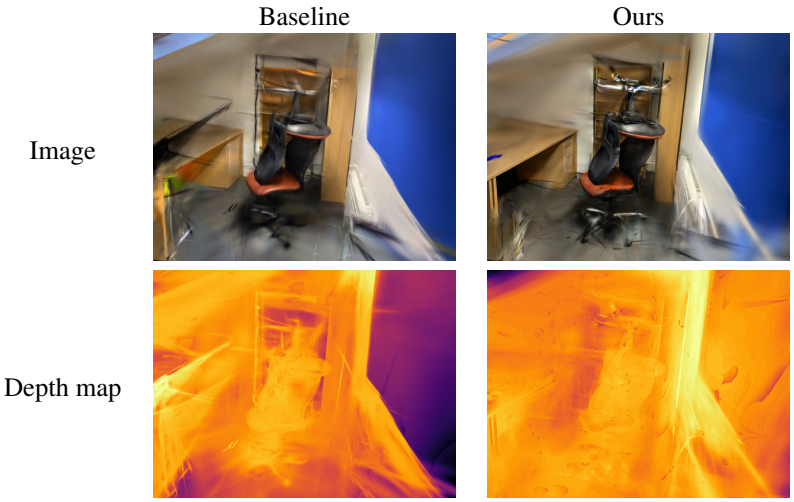

Figure 4: Example test image reconstructions for `chairs` and their corresponding depth maps. The floor, the table on the left, the bookshelf on the back and the blue board on the right all have more realistic depth in our reconstruction (right) than the baseline (left). However, the part of the wall surrounding the blue board as well as the top right corner still do not have the correct depth.

3.2.2 Smoothing of irregular surfaces

In the baseline, many surfaces are not reconstructed properly. Instead, they are composed of many Gaussians facing different directions other than the surface normals, but which happen to align when viewed from one of the training cameras. Our regularisation methods help remedy this and result in smoother surfaces with less high-frequency noise, as shown in Figure 5. Despite this, they are often still significantly more irregular than they should be.

Figure 5: The effect of our regularisation methods on a test image for `truck`. One can observe that the regularised image (right) is less noisy than the baseline reconstruction image (left).

3.2.3 Artifacts

Our regularisation helps remove some reconstruction artifacts. On the other hand, it sometimes introduces additional artifacts in the form of very wide semi-transparent Gaussians, possibly as a way to circumvent the diffusion loss. Examples of these phenomena can be observed in Figure 6

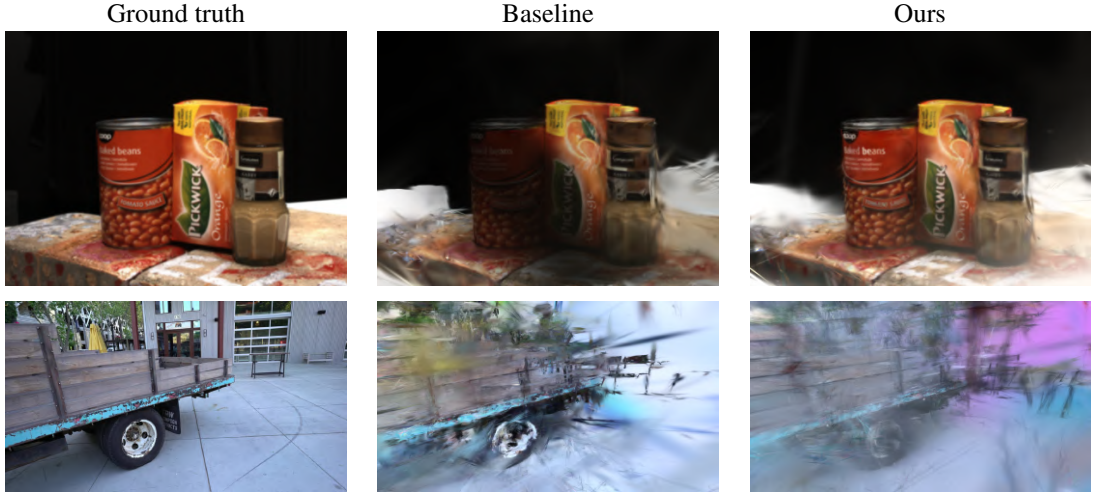

Figure 6: Examples of artifacts being removed (top) or added (bottom) by our regularisation. In the bottom right reconstruction, some Gaussians have appeared in front of the camera.

3.2.4 No big-scale changes

Most of the benefits of our regularisation methods are achieved through small changes to the Gaussians' parameters. When bigger-scale changes are necessary, such if the reconstruction of an object is far from its ground-truth location, our methods proved ineffective. We believe this is because, whilst moving the object to the correct location, it would then appear in other training views. This then leads to a poorer reconstruction from the latter training view. See Figure 7 for an example. Moreover, increasing the regularisation weight in order to circumvent this leads to unstable training that does not converge.

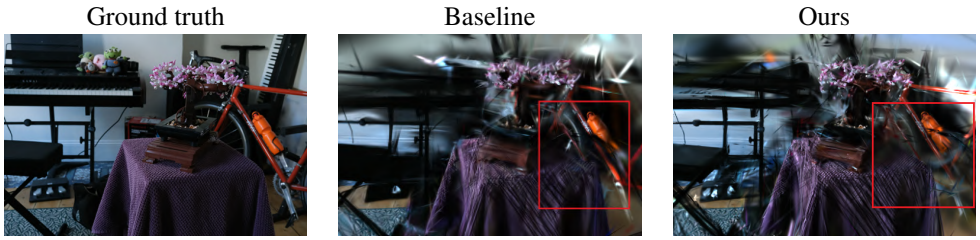

Figure 7: Example test image for `bonsai_3`. If one observes closely both reconstructions, one can find the back half of the bike next to the bonsai and in front of the front half of the bike. The back half is angled so that it is invisible to all but one training camera, and moving it into its correct location would require it to appear in the other cameras.

3.2.5 Difficulty with points not visible from training cameras

Most of the objects that are not reconstructed in the right location, both in the baseline and with our proposed regularisations, are those that appear in only one of the training images. For example, in `bonsai_3`, the front half appeared in all three training images and was reconstructed properly. However, the back half appeared in a single training image, and was not reconstructed in the right location as seen in Figure 7.

3.2.6 Sensitivity to hyperparameters

We found our results to be highly dependant on the choice of hyperparameters for the regularisation methods. In Section 4.2, we hypothesise reasons for this. Moreover, a good choice of hyperparameters for one scene does not always translate to a good choice for a different scene.

3.3 Quantitative results

A summary of our quantitative results for our general test sets are displayed in Table 1. .

Scene	# training images	Baseline PSNR	Our PSNR
`truck`	3	12.99	13.64
`room`	3	21.23	23.07
`bonsai_3`	3	14.83	14.93
`bonsai_9`	9	15.37	17.27
`guillem`	3	13.82	15.48
`chairs`	3	14.27	15.14
`rooftop`	3	13.73	17.50
average		15.17	16.72

Table 1: Table of quantitative results for general scans.

Scene	# training images	Baseline PSNR	Our PSNR
`DTU scan 1`	9	20.12	20.59
`DTU scan 2`	9	21.11	21.44
`DTU scan 4`	9	20.25	22.07
`DTU scan 5`	9	19.38	20.85
`DTU scan 7`	9	19.02	22.82
`DTU scan 9`	9	20.58	23.72
`DTU scan 11`	9	21.12	21.53
`DTU scan 12`	9	21.82	23.62
`DTU scan 14`	9	19.81	23.00
`DTU scan 15`	9	22.24	24.10
average		20.86	22.37

Table 2: Table of quantitative results for the first DTU set

Scene	# training images	Baseline PSNR	Our PSNR
DTU scan 21	9	20.45	21.30
DTU scan 31	9	21.06	21.61
DTU scan 34	9	20.92	21.69
DTU scan 38	9	22.64	23.67
DTU scan 40	9	18.74	21.83
DTU scan 41	9	19.19	20.56
DTU scan 45	9	20.85	22.44
DTU scan 55	9	23.86	27.00
DTU scan 82	9	18.03	23.08
average		20.64	22.58

Table 3: Table of quantitative results for the second DTU set

3.4 Comparison to state-of-the-art

We have compared our method to the state-of-the-art for the test DTU scans, and the results are summarised in Table 4.

Method	# training images	PSNR
mip-NeRF* [17]	9	23.58
DietNeRF* [18]	9	23.83
PixelNeRF* [7]	9	21.83
MVSNeRF* [19]	9	22.22
RegNeRF* [10]	9	24.93
DiffusioNeRF* [2]	9	25.18
3DGS [1]	9	20.64
Ours	9	22.58

Table 4: Table of results comparison with SOTA methods.
*These methods test on all 15 DTU test scans, while we test only on 9.

3.4.1 Comparison to 3DGS

In all the scenes that we tested, our methods showed an increase PSNR over 3DGS with both 3 and 9 training images. This increase tended to be larger on unbounded or more complicated scenes.

3.4.2 Comparison to NeRF methods

As shown in Table 4, our method outperforms PixelNeRF and MVSNeRF, but lags far behind DiffusioNeRF and RegNeRF. We note, however, that our training times are significantly faster. For reference, a scene took DiffusioNeRF approximately 30 minutes to train on a Nvidia A100 GPU, however it only takes us approximately 7 minutes on a Nvidia 4090 GPU. Moreover, we believe that further hyperparameter optimisation would have narrowed the PSNR gap between our reconstructions and the state-of-the-art.

4 Conclusion and future directions

4.1 Conclusions

In this project, we have explored the application of a method for regularising radiance fields with a diffusion model to 3D Gaussian Splatting. We have trained and evaluated this method on several few-view scenes. In doing both of these, we have successfully completed the core goals we set for the project. Additionally, we have completed our extension goals: we have proposed different regularisation methods and approaches to improve on the base method, as well as captured our own dataset and evaluated our methods on it. The method has shown PSNR improvements over the baseline in every scene we tested, but overall the difference was smaller than expected. Furthermore, we are still far away from the best NeRF methods in terms of PSNR.

4.2 Discussion of limitations

As our method closely follows the one from DiffusionNeRF, we expected it to perform similarly. In our experiments it performed worse, and we propose two main reasons explaining the difference in results:

Bias towards high frequency geometry Compared to NeRF, 3DGS is more likely to learn a geometry with sharp, high-frequency features due to its discrete representation of separate Gaussians. In particular, 3DGS can easily represent a very thin surface that is barely visible from some views but fully visible from others. We thus hypothesise that 3DGS is more likely to learn incorrect geometries than NeRF when using few images, and that the learnt scenes are harder to continuously move into more correct geometries. This is because the discrete high-frequency representation can require large movement of scene parameters to improve the geometry (see Section 3.2.4) .

Sensitivity to hyperparameters We found that if the hyperparameters are chosen poorly, the diffusion model will fail to contribute at all to the PSNR of the scene, and sometimes reduce the PSNR by 10 to 15 points.

Some of the important parameters are:

- **Diffusion Weight**: The diffusion model is predicting the gradient of the log prior probability. At every diffusion step we are updating the scene in the direction of this gradient in order to increase the prior probability of the scene. As usual with gradient descent, the step size has to be small enough for the gradient to give a good approximation of the optimal update. However, with a small diffusion weight the regularising effect might not be large enough to make a difference.
- **Ratio between diffusion weight and photo loss**: When the photometric loss is much higher than the diffusion loss, the diffusion model will fail to have significant impact. In particular, the scene will likely quickly arrive at a low photometric loss but with incorrect geometry. At this point a large movement is needed from the diffusion model, in which case the diffusion regularisation is ineffective (see Section 3.2.4). However, when diffusion loss is higher than photometric loss, the training becomes unstable withe the diffusion model pushing the scene to geometries not matching the training images.
- **Perturbation Strength**: If we render a patch for the diffusion model that is very far from showing the correct geometry, the diffusion model will provide an update moving the scene towards some plausible geometry which is not necessarily the correct one. When the patch is almost correct, the diffusion model should hopefully provide an update towards the correct geometry. Thus the perturbation strength can't be too high in the beginning of training, when only views close to the training views are likely to look correct. However, a low perturbation strength will not regularise the scene for the full range of novel views. Therefore, both the initial value and the increase over time of the perturbaton strength is important.
- **Depth scale**: Different scenes have different coordinate systems and thus overall scales. The diffusion model will only give useful gradients in the range of depths it was trained with. The exact method of depth scaling when training the diffusion model from DiffusionNeRF is not described. We scale each depth image to have an average depth of d, where d is a hyper parameter. For all reported results, we set $d = 10$, which we found was a value at which the diffusion model did show significant bias towards any direction (moving objects closer or further away).

As there are many high impact hyper-parameters, it could be the case that we simply haven't found the best combination of hyper-parameters yet, and that they have been chosen more carefully in the DiffusionNeRF paper. We also expect hyper-parameters to matter less when running for more iterations.

4.3 Future directions

In the future, we would like to improve on our methods with the following changes:

- **Improved DDM**: The DDM we use is taken directly from DiffusioNeRF. It is trained on RGBD sampled patches from the hypersim dataset [20]. This dataset focuses on "holistic

indoor scene understanding". Therefore, it's less relevant for our work on outdoor scenes. It also may not be powerful enough for working with Gaussians without further adaptions.

- **Hyperparameter optimisation**: With greater time, we could run more hyperparameter searches to find the optimal setup. This would also allow us to gain a better understanding of how much each hyperparameter impacts the scene and how they interact with other parameters. This would also significantly raise the PSNR above the baseline 3DGS, hinted at by the infrequent large gains we see in Table 1.
- **Iterations**: Due to time constraints each scene was run for 10,000 iterations. The state-of-the-art from 3DGS uses 30,000 iterations. Optimising the diffusion over this time period should improve results by allowing for lower learning rates and diffusion weights, increasing stability. Furthermore, we can try and optimise the relationship between the adaptive density control and diffusion. This may involve iteratively diffusing and densifying with varying weights.
- **Improved regularisation methods**: Whilst we have had moderate success introducing regularisation method, we believe more can be done. Lenticular regularisation showed promise but ultimately did not contribute enough and should be improved. We would also like to implement regularisation methods to remove the most frequent artifacts encountered in our reconstructed scenes that we did not have time to target.

References

[1] Bernhard Kerbl et al. "3D Gaussian Splatting for Real-Time Radiance Field Rendering". In: *ACM Transactions on Graphics* (2023).

[2] Jamie Wynn and Daniyar Turmukhambetov. "DiffusioNeRF: Regularizing Neural Radiance Fields with Denoising Diffusion Models". In: *CVF Conference on Computer Vision and Pattern Recognition* (2020).

[3] Ben Mildenhall et al. "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis". In: *Neural Fields in Visual Computing* (2020).

[4] Sean Kitson. *A Dive into the World of 3D Medical Imaging*. URL: `https://openmedscience.com/a-dive-into-the-world-of-3d-medical-imaging/`.

[5] Aaron Fenster, Grace Parraga, and Jeff Bax. "Three-dimensional ultrasound scanning". In: *National Library of Medicine* (2011).

[6] Jonathan T. Barron et al. "Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields". In: *CVPR* (2022).

[7] Alex Yu et al. "pixelNeRF: Neural Radiance Fields from One or Few Images". In: 2021.

[8] Jonathan Ho, Ajay Jain, and Pieter Abbeel. "Denoising Diffusion Probabilistic Models". In: *NeurIPS* (2020).

[9] Ivan Kobyzev, Simon J.D. Prince a d, and Marcus A. Brubaker. "Normalizing Flows: An Introduction and Review of Current Methods". In: *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE* (2020).

[10] Michael Niemeyer et al. "RegNeRF: Regularizing Neural Radiance Fields for View Synthesis from Sparse Inputs". In: *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. 2022.

[11] Noah Snavely, Steven M. Seitz, and Richard Szeliski. "Photo Tourism: Exploring Photo Collections in 3D". In: *ACM Trans. Graph.* 25.3 (2006), 835–846. ISSN: 0730-0301. DOI: `10.1145/1141911.1141964`. URL: `https://doi.org/10.1145/1141911.1141964`.

[12] Johannes Lutz Schönberger and Jan-Michael Frahm. "Structure-from-Motion Revisited". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[13] Johannes Lutz Schönberger et al. "Pixelwise View Selection for Unstructured Multi-View Stereo". In: *European Conference on Computer Vision (ECCV)*. 2016.

[14] Rasmus Jensen et al. "Large scale multi-view stereopsis evaluation". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2014, pp. 406–413.

[15] Ben Mildenhall et al. "Local Light Field Fusion: Practical View Synthesis with Prescriptive Sampling Guidelines". In: *ACM Trans. Graph.* 38.4 (2019). ISSN: 0730-0301. DOI: `10.1145/3306346.3322980`. URL: `https://doi.org/10.1145/3306346.3322980`.

[16] Arno Knapitsch et al. “Tanks and Temples: Benchmarking Large-Scale Scene Reconstruction”. In: *ACM Transactions on Graphics* 36.4 (2017).

[17] Jonathan T. Barron et al. “Mip-NeRF: A Multiscale Representation for Anti-Aliasing Neural Radiance Fields”. In: *ICCV* (2021).

[18] Ajay Jain, Matthew Tancik, and Pieter Abbeel. “Putting NeRF on a Diet: Semantically Consistent Few-Shot View Synthesis”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021, pp. 5885–5894.

[19] Anpei Chen et al. “Mvsnerf: Fast generalizable radiance field reconstruction from multi-view stereo”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 14124–14133.

[20] Mike Roberts et al. “Hypersim: A Photorealistic Synthetic Dataset for Holistic Indoor Scene Understanding”. In: *International Conference on Computer Vision (ICCV) 2021*. 2021.

A Appendix. Distribution of work

A.1 Implementation

Most of the tasks were worked on by everyone, and discussed as a group while doing so. However, we list for each of the group members their primary responsibilities:

- Fredrik implemented the data pipeline, depth renderer, foreground regularisation and an interactive 3D viewer.
- Guillem implemented the DDM, lenticular and frustum regularisation losses.
- Jonah implemented the testing and evaluation frameworks and conducted the tests.

A.2 Report

All sections were revised, discussed and edited by everyone. We list the main contributions of each authors to the original draft of the report:

- Section 1 was written by Jonah.
- Section 2 was written equally by Fredrik and Guillem.
- Section 3 was written primarily by Guillem, with some parts written by Jonah.
- Section 4 was written equally by Jonah and Fredrik, with some parts written by Guillem.

B Appendix. Test images

We include testing images for some the different datasets we used. We selected for each scene 3 test images with significant differences between the baseline and our reconstruction.

Figure 8: 3 selected test images for `bonsai_3`

Figure 9: 3 selected test images for `bonsai_9`

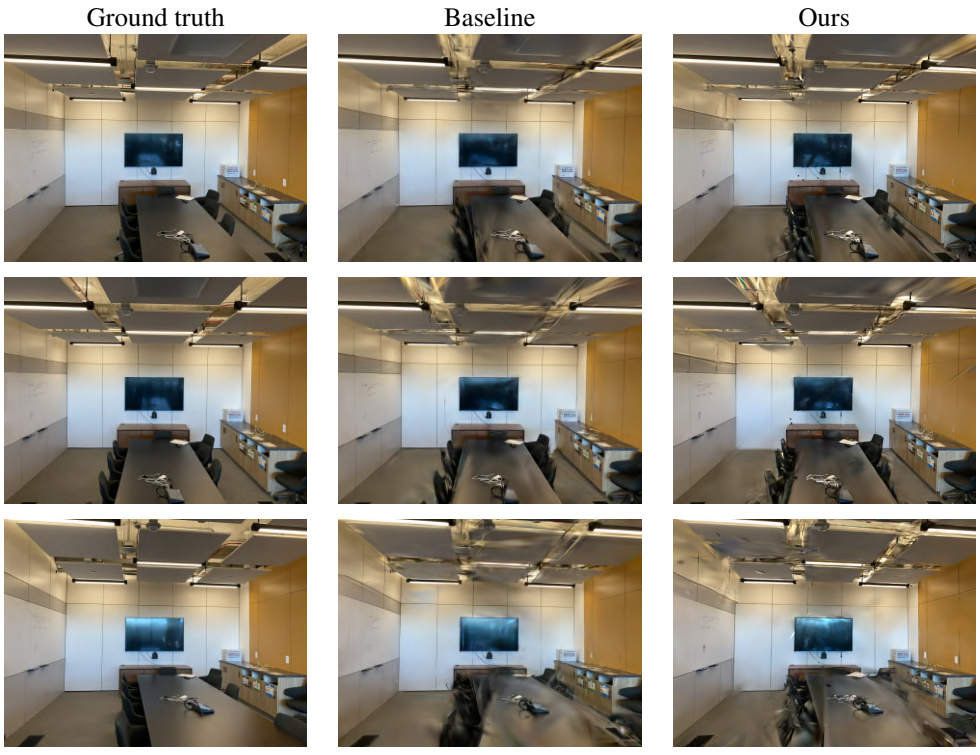

Figure 10: 3 selected test images for `room`

Figure 11: 3 selected test images for `truck`

Figure 12: 3 selected test images for `guillem`

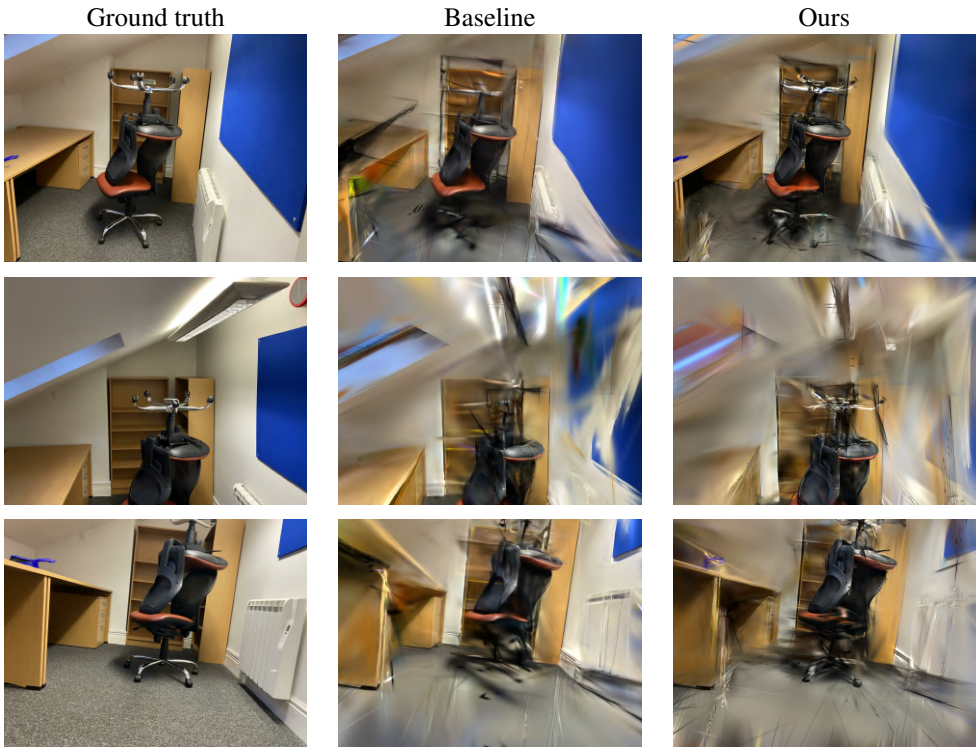

Figure 13: 3 selected test images for `chairs`

Figure 14: 3 selected test images for `rooftop`

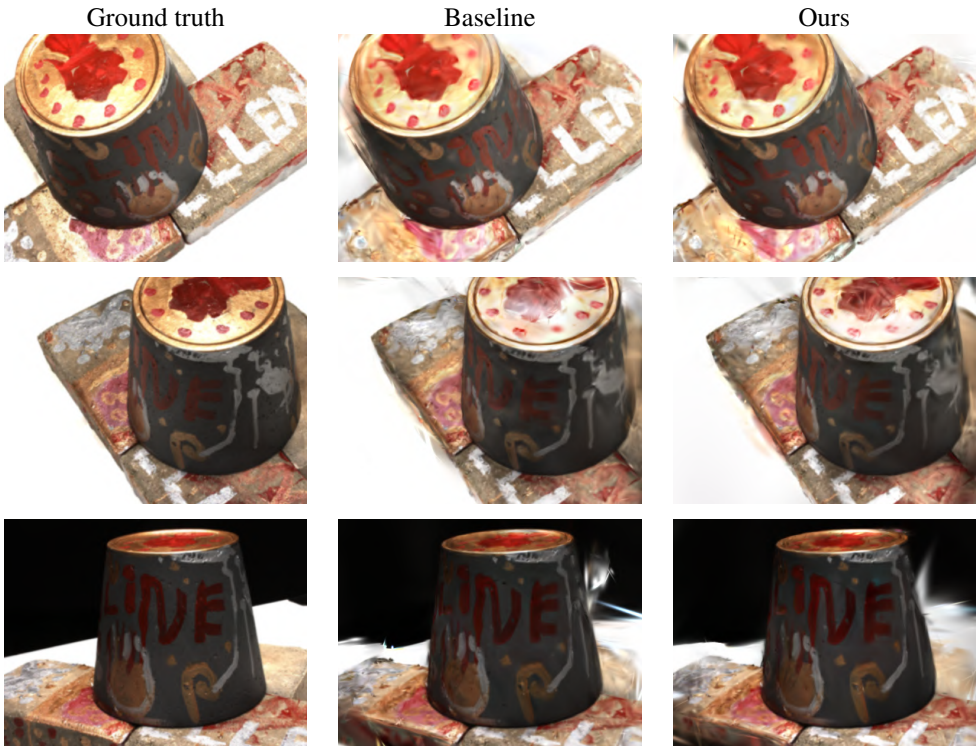

Figure 15: 3 selected test images `dtuscan1`

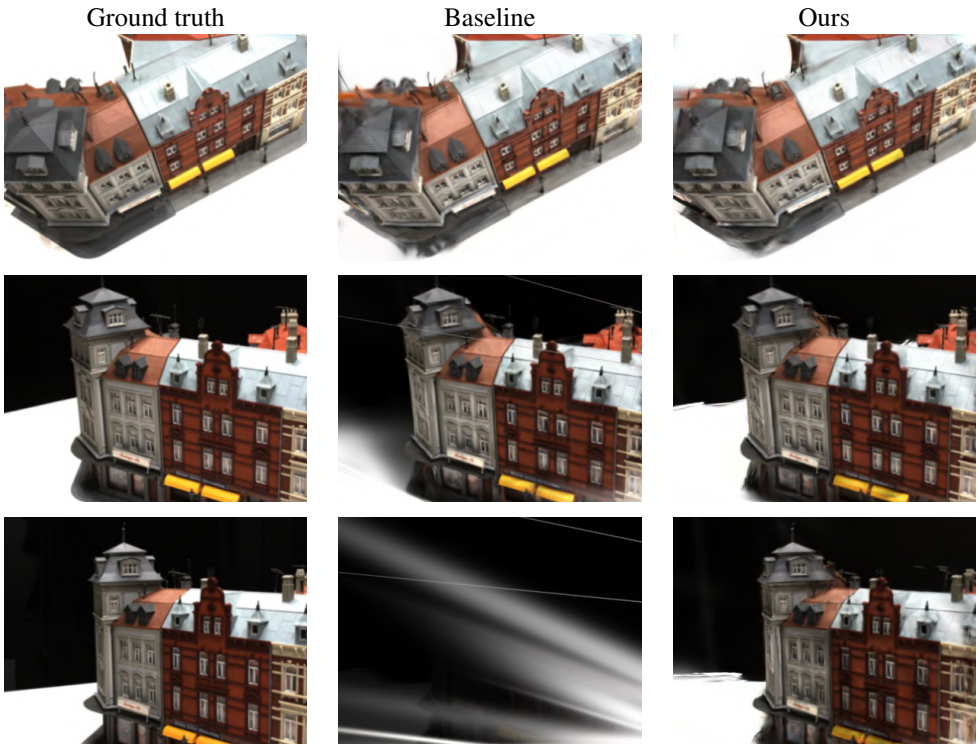

Figure 16: 3 selected test images `dtuscan9`

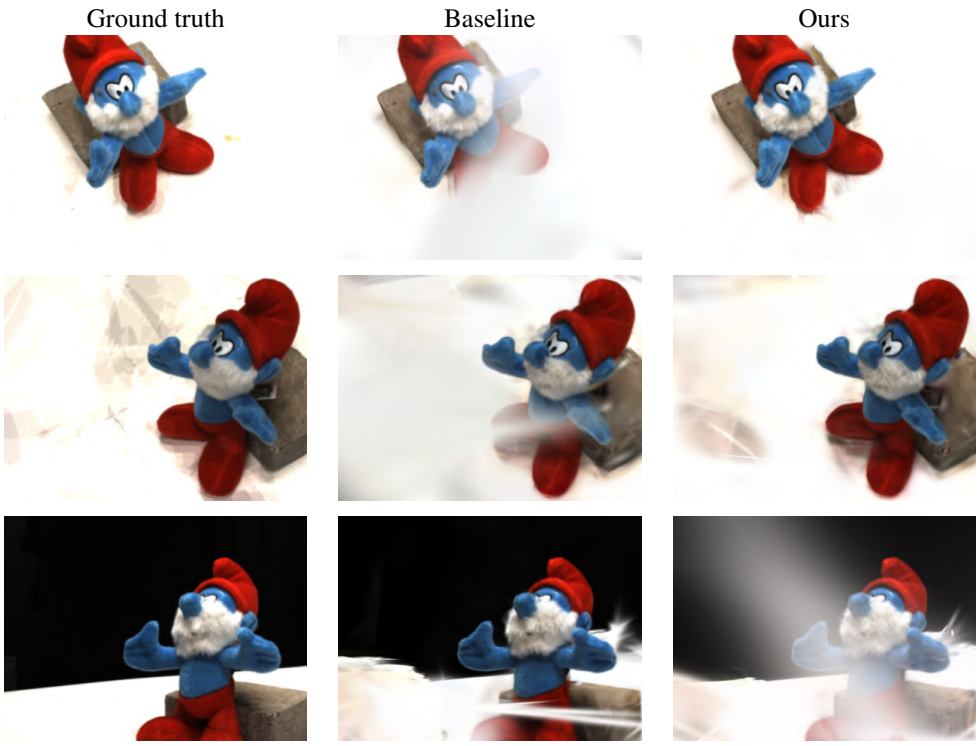

Figure 17: 3 selected test images `dtuscan82`