Patch-Based Inpainting of 3D Gaussian Splatting

A.M. Pardoel

Delft University of Technology

TU Delft

Patch-Based Inpainting of 3D Gaussian Splatting

by

A.M. Pardoel

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Friday October 18, 2024 at 1:30 PM.

Student number:	4729609	
Project duration:	December 2023 – October 2024	
Thesis committee:	Prof. dr. E. Eisemann,	TU Delft, thesis advisor
	P. Kellnhofer,	TU Delft, daily supervisor
	L. Cruz,	TU Delft

Cover: 3D Gaussian Splatting render of the *bicycle* scene of the Mip-NeRF 360 dataset

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Abstract

Image inpainting is a problem that has been well studied over the last decades. In contrast, for 3D reconstructions such as neural radiance fields (NeRFs), work in this area is still limited. Most existing 3D inpainting methods follow a similar approach: they perform image inpainting on the training images and use the inpainted images for further training of the 3D model. Due to inconsistencies in the different inpaintings of the images, the 3D inpainting often becomes blurry. With the advent of 3D Gaussian Splatting (3DGS), we identify a new opportunity for 3D inpainting. As 3DGS is more explicit in nature than NeRF, we can manipulate the 3D Gaussians directly rather than relying on image inpainting. Based on that key idea, we propose a method that works similar to the PatchMatch image inpainting algorithm. We first construct a nearest-neighbour field (NNF) by searching for nearest-neighbour patches throughout the scene that look similar to the area we want to inpaint. After constructing the NNF we copy the contents of the nearest-neighbour patches to the inpainting region and blend them together to obtain the inpainting result. In our experiments we found that our method performs well in terms of texture synthesis but struggles with structure synthesis, similar to the original PatchMatch algorithm. In cases where only texture synthesis is required to inpaint the area our method is able to provide good results, although in some cases pre-processing of the scene is necessary, as we found that better quality inputs (e.g. the scene itself, the surface mesh underlying the scene, and precise masks) drastically improve the results of our method. Moreover, some parameters of the algorithm are highly scene-dependent and by tailoring them to the scene we can further enhance the performance of the algorithm. Besides introducing a 3D inpainting method that directly manipulates the scene contents, our work offers valuable new insights into 3DGS editing in general.

Contents

1 Introduction

In image editing we often encounter situations where missing information needs to be filled in. For example, to remove objects or artefacts from photos we need to complete the image after removing certain pixels. Moreover, in contexts such as art conservation or medical imaging we often have to deal with incomplete data. We refer to this problem as image inpainting, which has been well-studied over the last few decades. The problem definition is simple: given an image and a mask, how do we fill in the masked pixels in the masked image such that we get a plausible result? An example of this is shown in Fig. 1.1.

With the advent of neural radiance fields (NeRFs) [Mildenhall et al. 2021], the problem arose of how to perform similar inpainting tasks for these 3D reconstructions, often referred to in the literature as multi-view inpainting. The most common technique to perform such a multi-view inpainting is to render the scene from different viewpoints (often the training viewpoints from the dataset for sake of simplicity) and inpaint the rendered images using some image inpainting method. The inpainted images are then used to further train the NeRF. However, this approach poses one major limitation: since image inpainting can produce a wide range of plausible results, there is no way to guarantee that the inpainted images from different viewpoints will be coherent in their inpainted regions, hence this method cannot achieve true multi-view consistency. The easiest way to mitigate this issue is to use a perceptual loss term

(a) Original image **(b)** Inpainting mask **(c)** Inpainted image

Figure 1.1: Example of image inpainting by Ogawa and Haseyama [2013].

(a) Original **(b)** Inpainting with blurry artefact

Figure 1.2: Example of blurry artefact of multi-view inpainting produced by the method of Yin et al. [2023]. (a) One of the original training images and (b) a render after performing multi-view inpainting to remove the box.

when training the NeRF based on the inpainted images. By not purely averaging over the pixel values but instead capturing some more high-level features, this tries to ensure that even if the images do not contain similar content, some of the structures will still be preserved to generate a more realistic output. In practice, however, this often still results in blurry artefacts in the scene due to averaging of the inpainted images, as shown in Fig. 1.2.

Moreover, the black-box nature of NeRFs makes it difficult to perform inpainting directly in the 3D space, rather than through training images. However, the more explicit nature of the recently emerged 3D Gaussian Splatting (3DGS) [Kerbl et al. 2023] offers new opportunities to perform multi-view inpainting by manipulating the 3D content directly. Other works have explored inpainting of 3DGS scenes using similar techniques as for NeRFs, thus suffering from the same limitations instead of leveraging the explicit nature of the 3D Gaussians. In this work, we aim to utilise the 3D Gaussians to generate higher-resolution textures in the inpainting areas and hereby avoid the blurry artefacts produced by the current state-of-the-art.

In order to achieve this, we port a traditional image inpainting algorithm to the 3D setting, as these methods method rely on direct manipulation of pixel content in contrast to more modern methods that use deep learning. Moreover, with traditional methods we can understand what the algorithm does and apply the same concepts in 3D space. Although learning-based methods for images have been around for some time now and provide high-quality results, deep learning in 3D is still at an early stage, with current methods focusing mostly on classification, segmentation, and completion tasks for point clouds. Works such as by X. Yu; Tang, et al. [2022] and Ma et al. [2022] offer impressive results by leveraging transformers and multi-layer perceptrons respectively. However, due to the more complex nature of 3DGS compared to a simple point cloud we do not consider these methods suitable for 3D Gaussians at this time and we decided to use a traditional non-learning-based method in our research.

The algorithm we choose to port to 3D is the classic PatchMatch [Barnes; Shechtman; Finkelstein, et al. 2009] algorithm. Similar to PatchMatch, we construct a nearest-neighbour field (NNF) that matches parts inside the inpainting area to similar parts elsewhere in the scene. The NNF is then used to copy Gaussians from other parts of the scene to areas inside the inpainting mask. Lastly, we perform optimisation to ensure that the inpainted area looks as much as possible as a weighted blend of the patches where the inserted Gaussians are copied from, since copying the Gaussians without this optimisation results in artefacts.

To summarise, our work provides the following contributions:

- An analysis of different inpainting techniques and their applicability to 3DGS.
- An inpainting method for 3DGS scenes that directly manipulates the 3D Gaussians rather than relying on image inpainting.
- An evaluation of our proposed 3D inpainting method, showcasing the strengths and weaknesses of the algorithm.

2

Related work

This chapter briefly discusses existing inpainting methods for images and 3D reconstructions. For 3D inpainting we distinguish between multi-view inpainting methods based on 2D image inpainting and true 3D inpainting methods where the 3D content is manipulated directly.

2.1. 3D representations

For the 3D inpainting task there are different 3D representations that one might consider. More traditionally, point clouds and voxels are common 3D representations, both of which can be captured using specialised equipment. Rather than using specialised equipment, one can also use a set of images to create a 3D reconstruction. This process generally consists of two steps: applying Structure from Motion (SfM) to extract camera positions from the images and then applying multi-view stereo (MVS) techniques to extract a representation of the 3D shape. SfM was first introduced by Longuet-Higgins [1981], presenting a direct method to compute the relative orientation of two views when at least eight pairs of corresponding points between the views are given. Currently, COLMAP [Schonberger and Frahm 2016] is the SfM technique commonly used in academia as it is one of the most robust open-source ones out there. The first to introduce the concept of MVS were Seitz and Dyer [1999], seeking to reconstruct a dense 3D model from multiple images taken from different viewpoints. It is common to use an MVS pipeline to extract a mesh from the sparse SfM point cloud. COLMAP provides an open-source library integrating both SfM and MVS into a single pipeline, which also enables the output of a mesh. Moreover, the last few years 3D reconstruction methods aiming to achieve photorealistic novel view synthesis have gained traction, the most prominent ones being NeRFs [Mildenhall et al. 2021] and 3DGS [Kerbl et al. 2023]. These methods take a set of training images as input and often use COLMAP to estimate the camera parameters of those images in order to train a photorealistic 3D model of the scene. Instead of using meshes or points clouds for the 3D model, these methods introduce new 3D representations based on storing the information in a neural network (NeRF) or modelling 3D Gaussians to fit the contents of the scene (3DGS).

2.2. Image inpainting methods

Many different overviews exist of the field of image inpainting [Jam et al. 2021; Barcelos et al. 2024; Qin et al. 2021; H. Xiang et al. 2023]. This section discusses the most relevant inpainting approaches we identified based on these literature reviews and their applicability to 3DGS.

Image inpainting techniques can be split into two categories: traditional and deep learning-based techniques, both of which are discussed in more detail in this section.

2.2.1. Traditional techniques

The most important categories we identified within traditional techniques are diffusion-based and exemplar-based, each of which is discussed below.

Diffusion-based methods These methods work based on diffusion of information to reconstruct the missing region. Bertalmio et al. [2000] first introduced this, using anisotropic diffusion to propagate information along the isophotes (level lines where the light intensity is the same). They use partial differential equations (PDEs) for the diffusion process, propagating information from surrounding areas into the void. Other works build on this foundation by adjusting the PDEs to enhance the diffusion process [Richard and Chang 2001; Tschumperlé 2006]. More recently, Sridevi and Srinivas Kumar [2019] proposed the use of fractional-order nonlinear diffusion models, attaining a good trade-off between edge preservation and smoothness.

Diffusion-based methods generally work well for smaller gaps and struggle with larger missing regions. In the context of 3DGS we sometimes encounter large areas with artefacts, therefore these methods are probably not the most suitable.

Exemplar-based methods These methods iteratively search for patches or pixels in the image resembling the damaged region. Efros and Leung [1999] pioneered work in this direction by proposing a method using Markov random field modelling to fill in the missing region pixel by pixel. Their method is able to replicate textural patterns, though it struggles with more complex structures. Later, Criminisi et al. [2004] proposed a method that works patch by patch rather than pixel per pixel, which in turn inspired many other works. Their approach prioritises patches inside the inpainting area by using heuristics such as the number of neighbouring pixels already filled in. The algorithm proceeds to find the most similar patch outside the inpainting area for the patch with the highest priority. Empty pixels inside the highest-priority patch are then filled in by copying the contents of the similar patch. This process is repeated patch by patch, gradually completing the image. Instead of filling in the hole patch by patch, Barnes; Shechtman; Finkelstein, et al. [2009] propose PatchMatch, which uses a randomised search algorithm to match all patches inside the inpainting area to a similar patch outside the inpainting area simultaneously. Pixels inside the inpainting area are then filled in by a process called “patch voting”, which combines the similar patches of neighbouring pixels instead of filling in the pixels patch by patch.

The replication of patches ensures the maintenance of structural and textural coherence in these exemplar-based methods, though the iterative approach can be computationally expensive. Given their higher-quality results and intuitive approach, they could be suitable for 3D Gaussians as well, copying Gaussians from other parts of the scene to the inpainting area. Since current multi-view inpainting techniques are not fast enough to work in real-time either, as discussed later in this chapter, the drawback of the iterative approach being more expensive does not necessarily pose a problem.

Comparing two of the most influential exemplar-based methods, PatchMatch [Barnes; Shechtman; Finkelstein, et al. 2009] is generally more efficient, scalable, and versatile than the method proposed by Criminisi et al. [2004]. However, the strength of PatchMatch lies more in texture synthesis than structure synthesis, while the method from Criminisi et al. is generally better at preserving structure. Since none of the two methods clearly outperforms the other in all scenarios, we deem both good candidates for adaptation to 3DGS. In the end, a hybrid approach between the two might work better than each of the methods separately. However, simply adapting one of the methods should provide insight into the applicability of these methods to 3DGS in general. Because the higher efficiency of PatchMatch might be beneficial with the additional overhead posed by working on 3D Gaussians instead of pixels, we pick PatchMatch as the best choice for our work.

2.2.2. Deep learning-based techniques

Deep learning-based techniques can be divided into four categories: convolutional neural networks-based, generative adversarial networks-based, transformer-based, and diffusion-based methods. As their naming indicates, the methods differ in the type of network architecture they use. It should be noted that all latter categories often make use of convolutional neural networks as well, as part of their broader architecture.

Convolutional neural networks-based methods The first work to use a convolutional neural network (CNN) for the task of image inpainting is proposed by Jain and Seung [2008]. In their work they train a CNN with over 15,000 parameters using backpropagation to learn to remove Gaussian noise

from images. Xie et al. [2012] improve on this by combining sparse encoding and deep networks pre-trained with denoising auto-encoder. Their method is able to solve more challenging tasks, such as removing superimposed text from an image rather than only Gaussian noise. One of the most prominent more recent methods in this category is the work by H. Liu et al. [2020]. They use a mutual encoder-decoder CNN for joint recovery of structure and texture, rather than recovering both in two separate stages as done by Ren et al. [2019]. Because the recent trend in deep learning-based methods has been to use networks that fall into either of the next three categories instead of purely using CNNs, the work by H. Liu et al. [2020] is still one of the most relevant works in this category today.

Generative adversarial networks-based methods Pathak et al. [2016] first introduced generative adversarial networks (GANs) to image inpainting, by having a "context encoder" generator network that generates the inpainting and a discriminator network that ensures the inpainting looks realistic and coherent. They show that using this adversarial approach enhances the visual quality of the inpaintings. Work by J. Yu et al. [2019] shows how a different type of convolution can be used in the generator network to further improve results. They introduce the concept of gated convolutions, which are able to dynamically determine the importance of features. The gating mechanism is learned together with the rest of the network and during inpainting the gates ensure that relevant features are emphasised while irrelevant ones are suppressed. One of the most prominent recent works using GANs for image inpainting is EdgeConnect [Nazeri et al. 2019], where they split the inpainting problem into two separate problems: structure prediction and image completion. Their model first predicts the structure of the missing region in the form of an edge map, which is then used to guide the inpainting process. Both stages use their own GAN. Another method using a different type of convolution and one of the most prominent recent works in image inpainting is LaMa [Suvorov et al. 2022], which employs fast Fourier convolutions. Because of the use of the frequency domain it excels at inpainting periodic structures. While not strictly operating as a standard GAN that uses a discriminator network to judge the entire output at once, LaMa uses a patch-based adversarial loss as part of the total loss function. Therefore, we still consider it to be a GAN-based method.

Transformer-based methods J. Yu et al. [2018] pioneered the use of contextual attention for image inpainting. Although their work does not use a transformer network, it inspired many other works leveraging the attention mechanism for inpainting and eventually the use of transformers [Quan et al. 2024]. Wan et al. [2021] and Y. Yu et al. [2021] both employ a transformer network, generating multiple plausible inpainting results. The latter achieves superior performance by proposing a novel bidirectional and autoregressive transformer. In contrast, Zheng et al. [2022] focus on improving the fidelity of a single inpainting rather than generating multiple outputs. They achieve this by first generating a coarse inpainting using a transformer network and then propose a refinement network using a novel attention-aware layer to turn the coarse inpainting into a higher-quality output.

Diffusion-based methods Not to be confused with the traditional diffusion-based methods, these diffusion models form a class of generative AI. Sohl-Dickstein et al. [2015] already applied early diffusion models to inpainting, however, diffusion models for image generation as we know them today were first introduced by Ho et al. [2020], improving on the theoretical foundation from Sohl-Dickstein et al. [2015] to make the models more practical. The most prominent work using diffusion models specifically for image inpainting is RePaint [Lugmayr et al. 2022], proposing several improvements to make the model perform better on this specific task. Moreover, the most renowned diffusion model for image synthesis is currently Stable Diffusion [Rombach et al. 2022], which introduced latent diffusion models and focuses on a broader set of tasks than inpainting exclusively.

When considering the applicability of deep learning to 3DGS, we have to note that deep learning depends on a vast amount of training data being available to train the network. As images are easy to capture and many image datasets already exist, this enabled researchers to train these deep learning models for image inpainting. However, we face numerous issues when we think about applying deep learning methods to 3DGS. It is not immediately clear how to feed a 3DGS scene to a neural network. The scenes are not as structured as images, since they do not follow a regular grid structure similar to the pixels of an image. With the number of 3D Gaussians in a scene typically ranging from 1-5 million [Kerbl et al. 2023] and each Gaussian having numerous different properties, the scenes are also

made up of a lot more data than the small images that networks are generally trained on. Moreover, capturing an image is easy, but capturing a 3DGS requires capturing multiple images and training the scene for a non-negligible amount of time, with varying results in terms of scene quality. Therefore, it is cumbersome to generate a vast collection of scenes, especially one vast enough to train a deep learning network with. Taking all of the above into account it comes as no surprise that big datasets of 3DGS scenes do not exist yet.

If one were to research application of deep learning methods on 3DGS scenes, it would make sense to focus on a very specific topic of interest that makes it easy to acquire many scenes to train the network with. After showing that the general approach works for this specific topic, one could extend the network by feeding it a wider range of data after acquiring more varying scenes. However, taking current limitations into account, we look back to traditional methods instead for application to 3DGS.

2.3. Multi-view inpainting methods

For multi-view inpainting we can distinguish between methods for NeRFs and 3DGS. However, state-of-the-art methods for both follow essentially the same approach [Mirzaei; Aumentado-Armstrong; Derpanis, et al. 2023; Yin et al. 2023; J. Wang et al. 2024; Ye et al. 2023; J. Huang and H. Yu 2023]: given images and image masks, they employ an image inpainting algorithm and use the inpainted images to further train the scene, resulting in an inpainted 3D scene. The major problem with this approach is that the 2D image inpaintings can be wildly inconsistent, introducing artefacts to the 3D scene when training on the inpainted images. To improve consistency and appearance of the geometry inside the masked region, some of the methods implement depth supervision by inpainting the depth maps as well and/or use a perceptual loss metric during training. Notably, the image inpainting method most commonly used for similar purposes and also the one used in all of these methods is LaMa [Suvorov et al. 2022]. Moreover, the perceptual loss metric usually employed is LPIPS [Zhang et al. 2018].

Since these methods all follow a very similar approach, they all suffer from the same limitation as well. The depth supervision and perceptual loss help mitigate the issue of inconsistently inpainted images, however, the state-of-the-art methods are still prone to blurry artefacts. Mirzaei; Aumentado-Armstrong; Brubaker, et al. [2023] try to overcome this limitation by using only a single inpainted reference view and monocular depth estimators to back-project the inpainted view to the correct 3D positions. However, their method still faces limitations: it falls back to a diffuse prior in the case of isolated masked regions lacking surrounding context and exact depth alignment remains difficult.

Contemporary with our work, others have introduced diffusion priors to guide the inpainting process. While technically not researching inpainting techniques but instead aiming to improve NeRF training to reduce floaters, some of the first papers to use diffusion priors in this context were DiffusioNeRF [Wynn and Turmukhambetov 2023] and Nerfbusters [Warburg et al. 2023]. Introducing this technique to multi-view inpainting was done by Inpaint3D [Prabhu et al. 2023], MVIP-NeRF [Chen et al. 2024], and MALD-NeRF [Lin et al. 2024] for NeRFs, while RefFusion [Mirzaei; De Lutio, et al. 2024] and InFusion [Z. Liu et al. 2024] did the same for 3DGS. Without going into detail, the key idea of these methods is that since the images generated by the diffusion model depend on some initial noise (the diffusion prior), we can tweak this noise in a smart way to obtain more view-consistent inpaintings. With the exception of Inpaint3D, all of these diffusion-based inpainting methods use latent diffusion models [Rombach et al. 2022]. The methods deliver very high-quality results with detailed 3D-consistent inpaintings.

NeRFiller [Weber et al. 2024] also uses diffusion, but rather than using the diffusion prior they identify a useful phenomenon in diffusion models: denoising images tiled in a grid results in more consistent multi-view inpaintings than inpainting them independently. However, while the technique encourages more consistent image inpaintings, this is not guaranteed.

Taking another approach, M. Wang et al. [2024] aim to enhance consistency using uncertainty estimation. When training the inpainted NeRF, the uncertainty affects the weights used in the loss function. Uncertainty estimation has been used in other methods related to NeRFs, but this is the first work to use it for NeRF inpainting. However, their work still has its limitations, being dependent on LaMa image inpainting and showing flaws in handling shadows.

2.4. True 3D inpainting methods

In the context of NeRFs and 3DGS, virtually no work has been published on true 3D inpainting methods that explicitly work on the 3D contents of the scene. Therefore, we look to the related fields of point cloud completion and voxel inpainting.

Hoppe et al. [1992] were the first to propose a systematic method for surface reconstruction of unorganised point clouds and with this they laid the foundation for many other works in this direction. Their method uses a signed distance field to represent the surface. Pauly et al. [2005] detect geometric patterns from the point cloud and match these against an example database for shape completion. H. Huang et al. [2013] propose a method that aims to detect and preserve important edge structures.

An interesting work on point cloud completion inspired by traditional image inpainting techniques is 3D-PatchMatch, introduced by Cai et al. [2015]. Their work draws inspiration from the algorithms presented by Barnes; Shechtman; Finkelstein, et al. [2009] and Criminisi et al. [2004]. In their method a nearest-neighbour field is constructed over all the boundary patches, after which they are prioritised using a prioritisation function similar to Criminisi et al. Based on their priority the boundary patches are filled in patch by patch and the boundary is updated. The process continues until the entire hole is filled in. In contrast, our work does not use boundary patches, but fills in the entire hole at once, more similar to the original PatchMatch algorithm. Moreover, because the 3D Gaussians are inherently different from simple points, their method is not directly applicable to 3DGS. For example, the Gaussians can cover larger areas of the scene, requiring a different inpainting approach. In the context of 3DGS we also have to consider view dependence, which Cai et al. did not need to take into account for point clouds.

In contrast to these traditional methods, more recent methods for point cloud completion generally use a deep learning-based approach. Laying the groundwork for deep learning on point clouds, PointNet [Qi et al. 2017] introduced a network for 3D classification and segmentation. Yuan et al. [2018] were the first to propose a network specifically for point cloud completion. More recently, approaches using geometry-aware transformer networks were introduced by PoinTr [X. Yu; Rao, et al. 2021] and SnowflakeNet [P. Xiang et al. 2021]. Most of these deep learning-based techniques directly take the point cloud as input, instead of working on images as we saw for multi-view inpainting.

Given some of the adequate results achieved with these methods, it seems likely that in the future deep learning will be possible on 3DGS scenes as well. However, the current limitations (e.g. lack of datasets and large number of properties/points) still pose too big a challenge.

Lastly, an approach that especially finds its use in the medical imaging field is voxel inpainting. Also in this field methods such as the one by Torrado-Carvajal et al. [2021] exist that are based on traditional inpainting algorithms while others such as Kang et al. [2021] and Wei et al. [2023] use deep learning.

3

Preliminaries

This chapter provides a more detailed background on two papers: the original PatchMatch paper [Barnes; Shechtman; Finkelstein, et al. 2009] and the original 3D Gaussian Splatting paper [Kerbl et al. 2023]. It is important to have a grasp of these papers in order to better understand our work, which builds on concepts introduced by these papers. This chapter aims to cover only the relevant concepts of these papers in relation to our work. For more details the reader is referred to the original papers.

3.1. PatchMatch

Barnes, Connelly; Shechtman, Eli; Finkelstein, Adam, and Goldman, Dan B [2009]. "PatchMatch: A randomized correspondence algorithm for structural image editing". In: *ACM Trans. Graph.* 28.3, p. 24

The original PatchMatch paper describes an algorithm that can be used for an array of image editing tasks, one of which being image inpainting. The key idea of the algorithm is that given two images A and B, for every patch in A we can find a similar patch in B. This concept is called a nearest-neighbour field (NNF) and the paper shows how it can be used to perform various image editing tasks. A schematic illustration of the NNF is shown in Fig. 3.1.

More formally, the NNF is defined as a function $f : A \to \mathbb{R}^2$, which maps every patch coordinate (e.g. the pixel at the centre of a patch) in image A to a 2D offset. For any patch coordinate $\boldsymbol{a}$ in image A, its nearest neighbour $\boldsymbol{b}$ in image B is given by $\boldsymbol{b} = \boldsymbol{a} + f(\boldsymbol{a})$. The mapping minimises some patch distance metric $D(\boldsymbol{a}, \boldsymbol{b})$ and the patches are of a fixed size, $N \times N$, where N is a parameter of the algorithm.

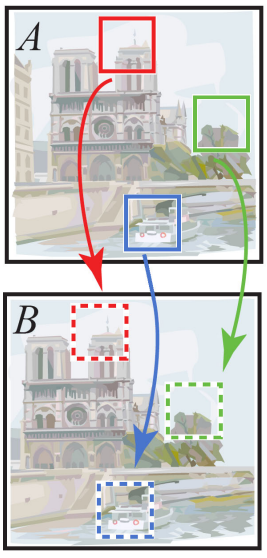


Figure 3.1: Schematic illustration of NNF [Barnes; Shechtman; Finkelstein, et al. 2009].

NNF construction The naive approach to construct such an NNF is using brute force search. However, this is very expensive. Therefore, the paper introduces a novel approach to efficiently compute an approximate NNF. The algorithm has three main components, illustrated in Fig. 3.3. The NNF is initially filled with random offsets and then updated iteratively. During an iteration of the algorithm, all possible patches in A are visited in scan order (from left to right, top to bottom). Every patch undergoes propagation and random search, each trying to improve the nearest neighbour of the patch. That is, to find a patch in B that is more similar than the best one we found so far, according to our distance metric D. When visiting a patch we perform both propagation and random search before moving on to the next patch.

For propagation we consider the patches located one pixel up or left, e.g. the adjacent patches we have already visited in this iteration. The underlying as-

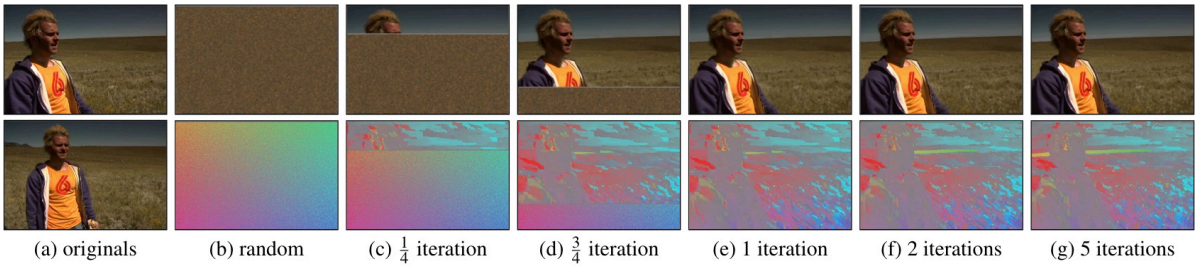


Figure 3.2: Illustration of convergence of the approximate NNF algorithm from Barnes; Shechtman; Finkelstein, et al. [2009]. "(a) The top image is reconstructed using only patches from the bottom image. (b) above: the reconstruction by the patch "voting" ..., below: a random initial offset field, with magnitude visualized as saturation and angle visualized as hue. (c) 1/4 of the way through the first iteration, high-quality offsets have been propagated in the region above the current scan line (denoted with the horizontal bar). (d) 3/4 of the way through the first iteration. (e) First iteration complete. (f) Two iterations. (g) After 5 iterations, almost all patches have stopped changing. The tiny orange flowers only find good correspondences in the later iterations." [Barnes; Shechtman; Finkelstein, et al. 2009]

sumption is that the offsets are likely to be the same. We try both two offsets and update the NNF accordingly. To ensure that information is propagated in both direction we visit patches in reverse scan order during even iterations.

After propagation we perform random search. The idea is to improve the nearest neighbour by testing a series of random candidates at a decreasing distance from our current best match. We start with some large search window and keep halving the search window until we reach a window of below 1 pixel.

The algorithm finds an approximate solution in only a small number of iterations. An example of this is shown in Fig. 3.2.

Inpainting We will now discuss how this approximate NNF algorithm can be used to perform image inpainting given an image A and a mask M. Instead of finding the correspondences between two images A and B, we construct an NNF that maps every patch inside the masked area of A to its nearest-neighbour patch outside the masked area. Once we have this NNF, we perform "patch voting" to blend the nearest-neighbour patches together inside the masked area. Intuitively, for every patch inside the inpainting mask we copy the pixel contents of its nearest-neighbour patch and paste it at the patch location inside the mask. Because the patches overlap we take the average of the pasted pixel values at every pixel in order to construct a new image. Instead of simply taking the average value we can also take a weighted average based on the patch distances given by our NNF. This lets us prioritise patches for which a better match was found. Fig. 3.5 shows an example of the image

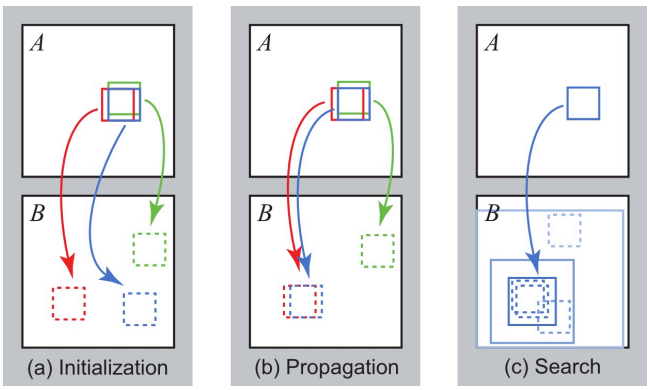
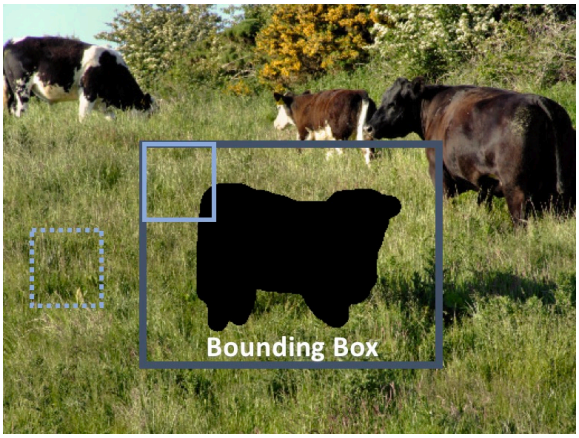


Figure 3.3: Phases of NNF construction [Barnes; Shechtman; Finkelstein, et al. 2009].

Figure 3.4: Pyramid of the multi-scale approach [Lee et al. 2018].

(a) Original



(b) Masked

(c) Result without multi-scale

(d) Result with multi-scale

Figure 3.5: Example of image inpainting using the PatchMatch algorithm, taken from Lee et al. [2018]. (a) The original image, (b) the masked image and an example that shows how a patch inside the inpainting mask is matched to a patch outside the mask, (c) the inpainted image after performing "patch voting" without the multi-scale approach, and (d) the inpainted image using the multi-scale approach.

inpainting process. It should be noted that the entire PatchMatch algorithm works in an expectation maximisation (EM) fashion, where the result is iteratively improved. In every EM iteration we first construct an NNF and then use that to construct a new image by means of patch voting. This process iteratively minimises the summed distance of the patches.

The approach described so far still faces two major issues: the inpainting hole is initially empty so it is not trivial to find nearest-neighbour patches and the result contains blurry artefacts due to incoherent nearest-neighbour patches. To avoid these issues a multi-scale "gradual scaling" process is employed where we start with a low resolution copy of the image and gradually resize it. After every scaling, we perform a few EM iterations until we reach the full resolution. As Fig. 3.4 shows, at the coarsest level a patch covers a very large part of the inpainting hole. Intuitively, because a patch at the coarsest level is so large, every patch inside the inpainting mask will also contain something outside the mask, giving us an indication of what to look for when searching for its nearest neighbour. This solves the first issue with our simpler approach. At the same time, by incorporating more of the global information of the image we ensure that the algorithm does not get stuck in local minima and we end up with more coherent nearest-neighbour patches, which solves the second issue as well. Fig. 3.6 shows the intermediate result of the multi-scale approach at every resolution. By comparing Fig. 3.5c and 3.5d it becomes clear that the multi-scale approach produces sharper textures than the approach without.

The pseudocode of the complete algorithm is presented in Alg. 1.

Figure 3.6: Intermediate results of PatchMatch inpainting with multi-scale approach [Philippeau 2010].

Algorithm 1 PatchMatch pseudocode.

```
for l = 1, ..., L do                          ▷ For every resolution, starting at the coarsest level
    for i = 1, ..., N_EM do                   ▷ Perform N_EM EM iterations
        for j = 1, ..., N_NNF do              ▷ Construct NNF in N_NNF iterations
            for every patch in (reverse) scan order do
                propagation()
                random_search()
            end for
        end for
        patch_voting()                        ▷ Use NNF to fill hole
    end for
end for
```

3.2. 3D Gaussian Splatting

Kerbl, Bernhard; Kopanas, Georgios; Leimkühler, Thomas, and Drettakis, George [2023]. "3D Gaussian Splatting for Real-Time Radiance Field Rendering". In: *ACM Trans. Graph.* 42.4, pp. 139–1

3D Gaussian Splatting is a 3D reconstruction method, taking a set of images and camera positions and constructing a 3D model based on this input allowing it to render the scene from new viewpoints, a process known as novel view synthesis. Another prominent 3D reconstruction method that has been around for a few years are neural radiance fields (NeRFs) [Mildenhall et al. 2021], which essentially try to capture all the scene information in a neural network. With NeRFs, views are rendered through a process called ray marching, shooting a ray from every pixel into the scene and sampling it at certain intervals to see what is visible in the scene. In contrast, with 3DGS no neural network is involved. Moreover, rather than ray marching 3DGS is more like rasterisation, rendering new views by projecting 3D Gaussians onto the camera view. One can imagine these 3D Gaussians as blobs sitting in space, as depicted in Fig. 3.7. The entire scene is made up of them and they each have a position (mean), covariance matrix (scaling and rotation), opacity, and colour. The colour is defined using spherical harmonics, allowing for view-dependent colours. It is important to note that this idea is not entirely novel, as similar 2D splatting methods have been around for years. However, the paper leverages modern GPUs and fast GPU sorting algorithms to apply these concepts in 3D and achieve real-time rendering. Besides introducing 3D Gaussians as a novel high-quality 3D representation, the main contributions of the paper are an optimisation method to construct high-quality representations of captured scenes and a fast, differentiable rendering approach for the GPU.

The proposed optimisation method requires one additional input besides the images and camera positions, namely a point cloud resembling the scene. Fortunately, the Structure-from-Motion (SfM) technique often used to extract camera positions from a set of images also produces a sparse point cloud. This point cloud is used to initialise the Gaussians based on the points' positions and colours. After initialisation of the scene, the images and camera positions are used to train the model in an iterative fashion. In every iteration we take a camera and its corresponding ground-truth image and we render the scene from the camera. After rendering we can compute the loss against the ground-truth image and backpropagate the gradients in order to update the properties of the Gaussians. Moreover, we also use the gradients to decide if Gaussians need to be cloned or split to better fit the contents of the scene, as shown by the schematic illustration in Fig. 3.8. Lastly, Gaussians with a very low opacity (e.g. that are essentially transparent) are pruned from the scene. The entire optimisation pipeline is depicted in Fig. 3.9.

To allow fast training and rendering, the paper proposes a differentiable tile rasteriser. The rasteriser

Figure 3.7: 3DGS scene with the 3D Gaussians rendered as ellipsoids [Ebert 2023].

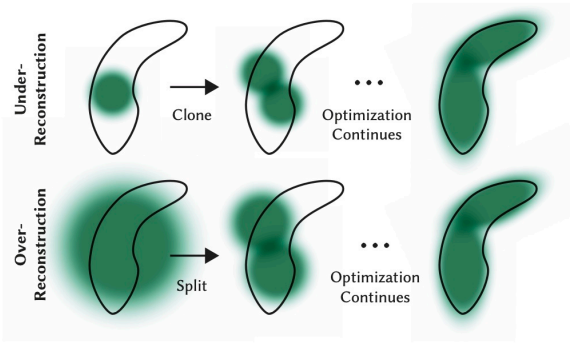


Figure 3.8: Adaptive densification scheme of 3D Gaussian optimisation [Kerbl et al. 2023].

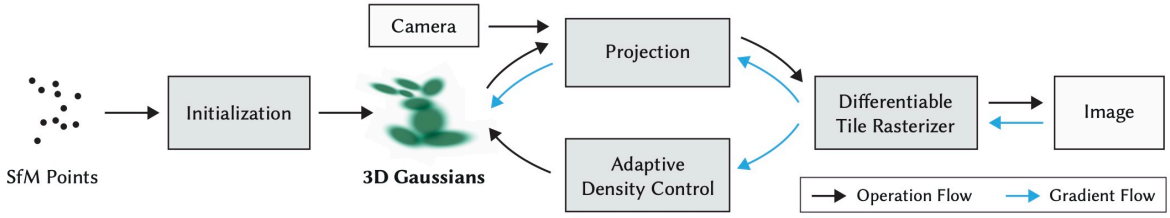


Figure 3.9: Optimisation method of 3D Gaussian Splatting scenes [Kerbl et al. 2023].

needs to be differentiable in order to support the gradient calculation necessary for training and the tiles serve as an acceleration structure to limit the Gaussians considered per pixel. The rasteriser first divides the screen into 16×16 tiles and then the Gaussians are culled against the view frustum and each tile. Each Gaussian is instantiated for every tile it overlaps and using a single fast GPU radix sort all instances are sorted by their view space depth and tile ID. Gaussians are not further sorted per pixel and their blending order is determined only by this initial sorting. Therefore, in some cases the α-blending is approximate. However, in practice this rarely affects the output noticeably while it greatly enhances performance. After sorting, for every pixel the list of Gaussians belonging to its tile is traversed in front-to-back order such that early termination can be used once an accumulated target saturation α for the pixel is reached.

4

Method

This chapter gives an in-depth explanation of our proposed method for inpainting 3DGS scenes. The method is inspired by PatchMatch [Barnes; Shechtman; Finkelstein, et al. 2009] and applies this algorithm in the context of 3DGS rather than images.

4.1. Overview

Our method takes the following inputs: a set of 3D Gaussians G representing the 3DGS scene, a surface mesh S underlying the scene, an inpainting mask M_I marking the part of the scene to inpaint, and a global mask M_G marking the part of the scene to restrict the nearest-neighbour search to. The output of our method is a modified set of Gaussians G', where the region defined by the inpainting mask is inpainted. A mask is defined as a function $M : \mathbb{R}^3 \to \{0, 1\}$ which takes a value of 1 if the input is inside the mask and 0 otherwise. In practice this means that the masks can be specified in any way that allows us to determine if a 3D point is inside the mask, e.g. if $M(\boldsymbol{p}) = 1$ for some point $\boldsymbol{p}$. For example, we can use a 3D shape as mask or image masks with their corresponding camera positions.

Since obtaining a surface mesh underlying the scene is not a trivial task, we use the method from SuGaR [Guédon and Lepetit 2024] to extract a mesh from the scene. Their algorithm first trains the scene based on a set of training images while using a regularisation term encouraging the Gaussians to be more surface-aligned. After performing a fixed number of training iterations, they use Poisson surface reconstruction [Kazhdan et al. 2006] to extract a mesh from the Gaussians.

Similarly to PatchMatch, our method starts by constructing an NNF over the scene. We first sample points along the surface S to discretise the search space for nearest-neighbour pairs and then match each point inside the inpainting mask to its nearest neighbour outside the mask. After constructing the NNF, we use this mapping to fill the inpainting hole by copying Gaussians from every nearest neighbour to the corresponding point inside the inpainting mask. We then perform optimisation to ensure that the copied Gaussians fit together well and blend in better with the rest of the scene at the border of the inpainting region. The process of constructing the NNF and inpainting the masked area is repeated multiple times at different resolutions to obtain the final result. The rest of this chapter will provide an in-depth explanation of all these steps.

4.2. Nearest-neighbour search

Analogous to PatchMatch, we aim to define an NNF to find similar patches efficiently. This section will highlight the key differences between our method and the original PatchMatch algorithm. The general algorithm remains the same as described by the pseudocode in Alg. 1.

4.2.1. Domain

Remember that originally the NNF is defined as a function $f : A \to \mathbb{R}^2$ of offsets over all possible patch coordinates (locations of patch centres) in image A. It is not immediately clear how this translates to

the 3DGS context, since the Gaussians are unstructured and thus do not follow a regular grid structure like the pixels of an image. Therefore, we propose a different domain in the context of 3DGS. The two main requirements for the domain are that it is discrete and that patches are approximately uniformly spaced out. This can be seen as a relaxation of the regular grid structure: clearly the two requirements hold for a regular grid, but the grid imposes more constraints on the topology. Since the Gaussians usually represent a surface, we obtain a domain satisfying the two conditions by sampling points along the surface S. This process consists of two steps:

1. **Masking:** we remove the vertices from S that are inside the inpainting mask M_I and perform Poisson surface reconstruction to fill the hole we created. We end up with two separate surface meshes S_{in}, S_0, inside and outside the inpainting mask respectively. We then define the surface mesh S_{out} as S_0 limited to the vertices inside the global mask M_G. Moreover, to mask the actual contents of the 3DGS scene we remove any Gaussians with a mean inside of M_I to obtain the masked set of Gaussians $G_0 = \{g \in G | M_I(\boldsymbol{\mu}_g) = 0\}$, where $\boldsymbol{\mu}_g$ denotes the mean (e.g. position) of g.
2. **Point sampling:** after obtaining S_{in}, S_{out} we use Poisson disc sampling as described by Yuksel [2015] to sample points along the surfaces, since this method should give us approximately evenly spaced out points. The method uses a greedy sample elimination algorithm to pick a subset of the desired size with Poisson disc property. We sample points on S_{in} and S_{out} separately, such that we get two sets of points P_{in}, P_{out} respectively. An example of this is shown in Fig. 4.1. We introduce a parameter γ for the desired average number of Gaussians per point and use that to compute the number of points to sample on the surface. Let A_0, A_{in} be the surface area of S_0, S_{in} respectively. With the point density $\rho = \frac{|G_0|}{A_0 \gamma}$, we define the number of points to sample on S_{in} to be $K_{in} = \lfloor \rho A_{in} \rfloor = |P_{in}|$ and similarly for K_{out}.

We will refer to the combined set of points as $P = P_{in} \cup P_{out}$. Furthermore, we modify the NNF definition to be $f : P_{in} \to \mathbb{R}^3$, such that every point in the inpainting region is associated with a 3D offset and we define a function $\mathrm{Snap}(\boldsymbol{x}) : \mathbb{R}^3 \to P = \mathrm{argmin}_{\boldsymbol{p} \in P} \|\boldsymbol{p} - \boldsymbol{x}\|$ which gives us the point $\boldsymbol{p} \in P$ with the smallest Euclidean distance to a 3D coordinate $\boldsymbol{x}$. Snap_{in} and Snap_{out} are defined similarly over P_{in}, P_{out} respectively. For a point $\boldsymbol{p} \in P_{in}$, its nearest neighbour $\boldsymbol{q} \in P_{out}$ is then given by $\mathrm{Snap}_{out}(\boldsymbol{p} + f(\boldsymbol{p}))$.

4.2.2. Patches

Now that we have our patch coordinates P, we will discuss how patches can be compared in order to determine the nearest-neighbour patches. In the image context, a patch is simply the $N \times N$ patch of pixels around the patch coordinate. In order to fill in missing parts of the surface, our goal is to ensure that the surface looks similar when we compare patches. Since the Gaussians are view- and order-dependent, the easiest way to determine what the surface actually looks like is to render it to an image.

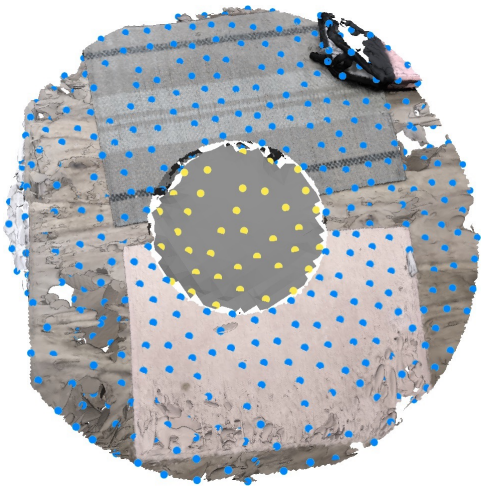


Figure 4.1: Example of points sets P_{in} (yellow), P_{out} (blue) sampled on the surfaces S_{in}, S_{out} respectively.

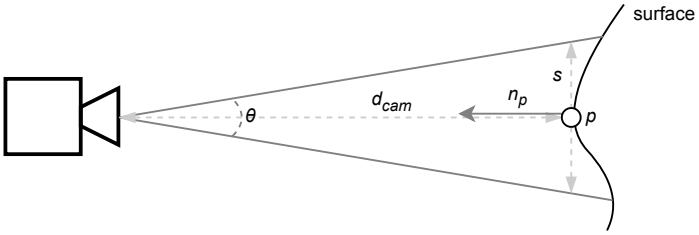


Figure 4.2: Illustration of the relationship between the camera distance d_{cam} and field of view θ to render an area of size s in world space around point $\boldsymbol{p} \in P$ with surface normal $\boldsymbol{n_p}$.

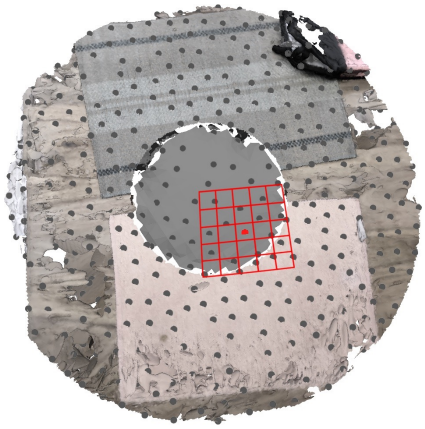


Figure 4.3: Example of a 5×5 patch at the red point $\boldsymbol{p} \in P$.

As we are only interested in a small part of the surface when examining a patch, we only render the part of the surface around the patch coordinate $\boldsymbol{p} \in P$. We use the patch renders to compare patches similarly to the pixel patches in PatchMatch using some distance metric D. In our experiments, we use the mean squared pixel-wise ΔE colour difference [Sharma et al. 2005] as distance metric.

To render the patch at $\boldsymbol{p}$, we render an image looking at $\boldsymbol{p}$ from its normal direction $\boldsymbol{n_p}$ and assuming an up-vector $\boldsymbol{u} = (0\ 1\ 0)^T$. As illustrated in Fig. 4.2, we set the camera distance d_{cam} according to the desired field of view θ and the size s of the area around $\boldsymbol{p}$ that we want to render: $\tan(\frac{\theta}{2}) = \frac{s}{2} \cdot d_{cam}^{-1}$. In the image context, one pixel is always the distance between two adjacent patch coordinates. We define a "pixel size" s_{px} in world space similarly by taking the mean of the distances from every $\boldsymbol{p} \in P_{in}$ to its k closest adjacent points in P_{in}, where k is a parameter of the algorithm. In our experiments we use $k = 4$, analogous to the image context where each pixel has four adjacent pixels. Following this analogy, these k points are also the ones we will use for propagation later. Using the world space "pixel size", we then define $s = N \cdot s_{px}$. Fig. 4.3 shows an example of a 5×5 patch around a point $\boldsymbol{p}$, where the side of the square is equal to s.

Moreover, in our experiments we set $\theta = 1°$, such that the perspective projection approximates an orthogonal projection. We wish to use an orthogonal projection as not to let perspective distort the view. However, as the standard 3DGS implementation only provides a perspective projection, we use this approximation. The 3DGS implementation also does not support frustum culling. Therefore, we restrict the Gaussians for rendering to $\{g \in G' | \left\|\boldsymbol{\mu}_g - \boldsymbol{c}\right\| > z_{near}\}$, where G' denotes the current state of the scene (including intermediate inpainting results), $\boldsymbol{c}$ the camera position, and $z_{near} = d_{cam} - \frac{s}{2}$.

In the case of rendering a patch at $\boldsymbol{p} \in P_{out}$, we mask any pixels that do not directly correspond to S_{out}, allowing us to penalise patches close to the inpainting border or the edge of the scene. We achieve this by rendering the depth buffers Z_{in}, Z_{out} of S_{in}, S_{out} respectively, and mask any pixels for which $Z_{in} > Z_{out}$. When computing the pixel-wise distances according to the distance metric D, we set the distance to a masked pixel to the maximum possible distance allowed by the metric.

Lastly, the original 3DGS implementation is sensitive to aliasing, producing artefacts especially when rendering at low resolutions. Because of this, if we were to render a patch of size N to an $N\times N$ image, the resulting image would be a poor representation of the actual scene contents, as N is typically a low value. To alleviate this issue, we introduce a parameter $\phi \in \mathbb{Z}^+$ and obtain the patch rendering resolution $R = N \cdot \phi$. In our experiments we used $\phi = 2$, meaning that for patch size $N = 5$ we would render a patch as a 10×10 image.

4.2.3. Iteration

The two main operations to construct the NNF are propagation and random search. As in PatchMatch, the operations are interleaved at the patch level, meaning that when visiting a point we first perform propagation and consecutively random search before moving onto the next point. We will now discuss both operations in more detail. To simplify updating the NNF we define an operator which returns

the offset $\boldsymbol{p} - \boldsymbol{q}$ if $\boldsymbol{q}$ is an improvement over the current nearest neighbour of $\boldsymbol{p}$, or the current value otherwise:

$$\mathrm{Update}_f(\boldsymbol{p}, \boldsymbol{q}) = \left(\underset{\boldsymbol{x} \in \{\boldsymbol{q}, \boldsymbol{p}+f(\boldsymbol{p})\}}{\mathrm{argmin}} D(\boldsymbol{p}, \boldsymbol{x}) \right) - \boldsymbol{p}$$

This allows us to easily test a candidate point and update the NNF accordingly.

Propagation

We visit points in order of ascending x-coordinate to ensure we sweep over all points in one direction. Moreover, in even iterations we visit points in reverse order to propagate information both ways. For every point $\boldsymbol{p} \in P_{in}$, we consider its k closest adjacent points in P_{in} for propagation (e.g. the same points as for determining s_{px}). Of those k adjacent points, we only take the ones we already visited during this iteration. For every visited adjacent point $\boldsymbol{a}$, we obtain the point $\boldsymbol{q} = \mathrm{Snap}_{out}(\boldsymbol{p} + f(\boldsymbol{a}))$, e.g. the point outside the inpainting mask closest to the offset point $\boldsymbol{p} + f(\boldsymbol{a})$. For each candidate point $\boldsymbol{q}$, we attempt to improve the NNF by setting $f(\boldsymbol{p}) \leftarrow \mathrm{Update}_f(\boldsymbol{p}, \boldsymbol{q})$. After testing all candidates for propagation, we move on to random search.

Random search

We attempt to find a better nearest neighbour for $\boldsymbol{p}$ by testing a series of random candidates surrounding its current nearest neighbour $\hat{\boldsymbol{q}} = \boldsymbol{p} + f(\boldsymbol{p})$. The random candidates are chosen at an exponentially decreasing distance from $\hat{\boldsymbol{q}}$. We define a search radius $r_i = w\alpha^i$, where w is a large maximum search radius and α a fixed ratio between search radii. We choose w to be the maximum Euclidean distance between any two points $\boldsymbol{p}, \boldsymbol{q} \in P_{out}$ and $\alpha = 0.5$. For every search radius r_i, we pick a random point $\boldsymbol{q}_i \in P_{out}$ within a distance of r_i from $\hat{\boldsymbol{q}}$. Again, we attempt to improve the NNF by setting $f(\boldsymbol{p}) \leftarrow \mathrm{Update}_f(\boldsymbol{p}, \boldsymbol{q}_i)$. We test candidates for $i = 0, 1, 2, ...$ until there are no other points than $\hat{\boldsymbol{q}}$ itself within a distance of r_i from $\hat{\boldsymbol{q}}$, after which we move on to propagation for the next point $\boldsymbol{p} \in P_{in}$.

Lastly, after performing half of the iterations to construct the NNF we try to merge with a prior NNF if available. This ensures that as we go through multiple EM iterations we propagate information across iterations and lower the chances of getting stuck in local minima. Merging the current NNF f with a prior NNF g is done as follows: for every $\boldsymbol{p} \in P_{in,f}$, we find the point $\hat{\boldsymbol{p}} = \mathrm{Snap}_{in,g}(\boldsymbol{p})$. Let $\boldsymbol{q} = \mathrm{Snap}_{out,f}(\boldsymbol{p} + g(\hat{\boldsymbol{p}}))$, then we try to improve the NNF by setting $f(\boldsymbol{p}) \leftarrow \mathrm{Update}_f(\boldsymbol{p}, \boldsymbol{q})$.

4.3. Inpainting

After constructing the NNF, we fill the inpainting hole by copying patches of Gaussians. For every point $\boldsymbol{p} \in P_{in}$ and its nearest neighbour $\boldsymbol{q} \in P_{out}$, we copy the Gaussians from $\boldsymbol{q}$ to $\boldsymbol{p}$ in order to obtain the modified set of Gaussians $G' = G_0 \cup G^*$, with G^* the copied Gaussians. More specifically, we copy the Gaussians inside a cube with centre $\boldsymbol{q}$ and side $2s_{px}$, where one of the axes of the cube is the normal vector $\boldsymbol{n_q}$ and one is given by $\frac{\boldsymbol{n_q} \times \boldsymbol{u}}{\|\boldsymbol{n_q} \times \boldsymbol{u}\|}$, with $\boldsymbol{u}$ our assumed up-vector. Notably, this is different from the image version of PatchMatch, where we copy the entire patch contents. Whereas the pixel values in the image context can be accumulated when overlapping patches, this is more difficult with the 3D Gaussians and by copying the entire patch contents we would massively increase the density of the Gaussians in the inpainting region. We also found that using side $s = Ns_{px}$ (e.g. the entire patch contents) caused the Gaussians not to be aligned to the surface while side s_{px} caused gaps in the inpainting hole after copying Gaussians. Therefore, side $2s_{px}$ offers a balanced alternative, as demonstrated by the comparison in Fig. 4.4. To account for differences in orientation between the surface normals $\boldsymbol{n_p}$ and $\boldsymbol{n_q}$, we first rotate the Gaussians around $\boldsymbol{q}$ before translating them by $\boldsymbol{p} - \boldsymbol{q}$. This rotation is given by rotating from $\boldsymbol{n_q}$ to $\boldsymbol{n_p}$ under our assumed up-vector $\boldsymbol{u}$. Mathematically, it is expressed as $\boldsymbol{R} = \boldsymbol{W}\boldsymbol{V}^{-1}$, where $\boldsymbol{W} = \begin{pmatrix} \boldsymbol{w}_1 & \boldsymbol{w}_2 & \boldsymbol{w}_3 \end{pmatrix}$ given $\boldsymbol{w}_1 = \boldsymbol{n_p}$, $\boldsymbol{w}_2 = \frac{\boldsymbol{w}_1 \times \boldsymbol{u}}{\|\boldsymbol{w}_1 \times \boldsymbol{u}\|}$, and $\boldsymbol{w}_3 = \boldsymbol{w}_1 \times \boldsymbol{w}_2$. $\boldsymbol{V}$ is defined similarly for $\boldsymbol{n_q}$ instead of $\boldsymbol{n_p}$.

Simply copying the patches of Gaussians causes artefacts, for example because the underlying surfaces from the different patches do not exactly line up or because overlapping patches get intertwined and become noisy. To overcome this problem, we propose an optimisation phase similar to the standard training of a 3DGS scene. For every point $\boldsymbol{p} \in P_{in}$ and its nearest neighbour $\boldsymbol{q} \in P_{out}$, we render

(a) Side s_{px} **(b)** Side $2s_{px}$ **(c)** Side Ns_{px}

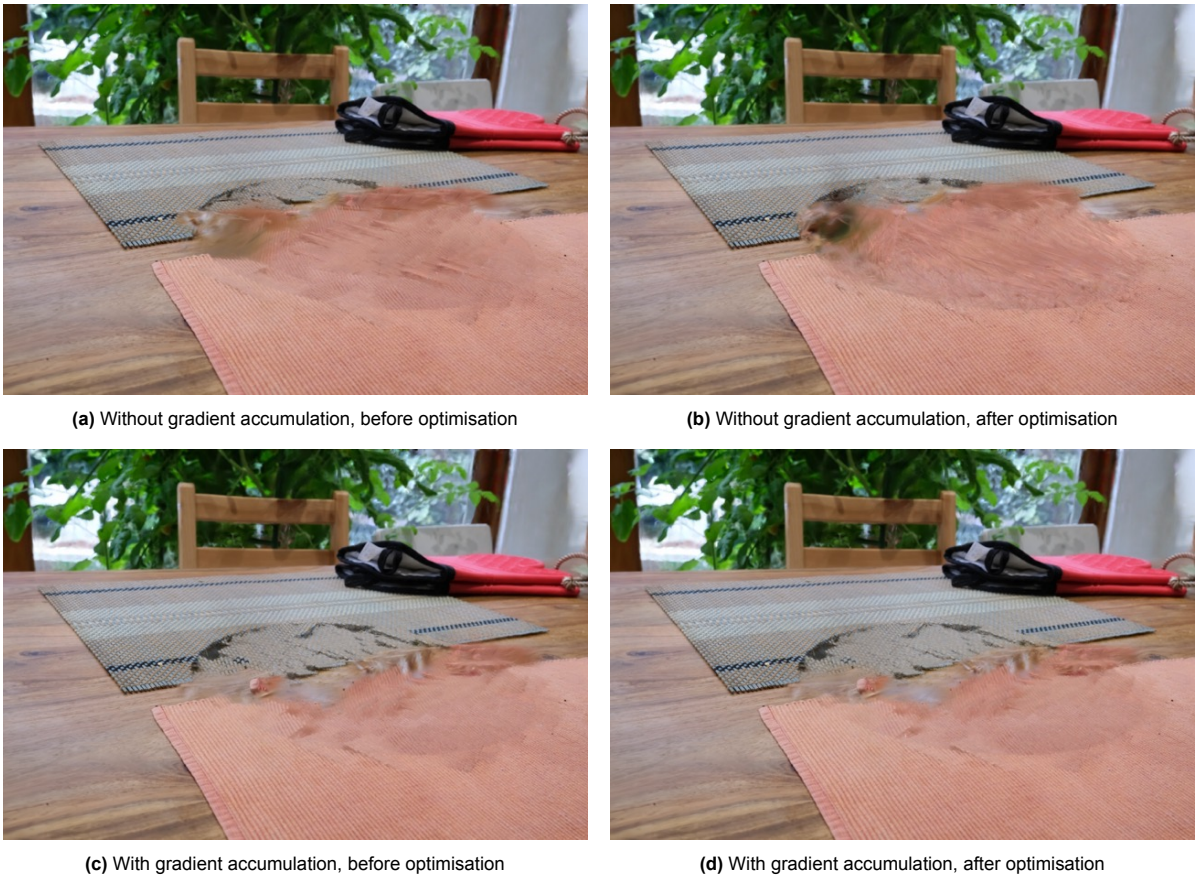
Figure 4.4: Comparison of different cube sizes for copying Gaussians. A larger cube means we copy more Gaussians per patch from the area outside the mask to inside the mask. The top row shows renders of the scene immediately after copying the Gaussians. The bottom row shows renders from a top view after optimisation.

the patch at $\boldsymbol{q}$ to an $R\times R$ image $I_{\boldsymbol{q}}$, which we use as a ground-truth training image for the patch at $\boldsymbol{p}$. Then we render the patch at $\boldsymbol{p}$ to an $R\times R$ image $I_{\boldsymbol{p}}$ and compute the L1 loss between $I_{\boldsymbol{p}}$ and $I_{\boldsymbol{q}}$, in order to calculate the gradients of the Gaussians we inserted into the inpainting hole. Moreover, to prioritise patches for which we found better neighbours, we multiply the L1 loss by the squared patch similarity $S(\boldsymbol{p},\boldsymbol{q}) = 1 - D(\boldsymbol{p},\boldsymbol{q})$ under the assumption that $D \in [0,1]$. As we do not want the new Gaussians to affect the area outside the inpainting area, for every point $\boldsymbol{p} \in P_{in}$ we also perform the same operation for some point outside but close to the inpainting mask M_I. We refer to one full sweep of all K_{in} points in P_{in} and the K_{in} chosen points from P_{out} as one optimisation iteration. In our experiments we perform 25 optimisation iterations during the optimisation phase. To ensure stable convergence of the optimisation, we average the gradients of the Gaussians after an iteration and only then perform backpropagation to modify the Gaussians. As shown in Fig. 4.5, without this gradient accumulation our optimisation actually achieves the opposite of our desired effect: the solution starts diverging instead of converging and the quality of the inpainted region deteriorates. We also zero out the gradients of the Gaussians in G_0 such that they are not affected. Similar to the standard 3DGS training, we densify and prune Gaussians every 10 iterations. Finally, for the last EM iteration we render patches at a higher resolution during the optimisation phase in order to ensure that we do not introduce artefacts by overlooking details. To achieve this, we use a resolution of $8R$ instead of R for the last iteration.

4.4. Multi-scale approach

For the same reasons as in the image context (e.g. to take global information into account and to have an initial inpainting guess) it makes sense to employ a multi-scale approach in the 3D setting. We downsample our representation of the scene by sampling fewer points along the surface. We obtain the number of levels in the hierarchy $L = \lfloor \log_4 K_{in} \rfloor$, such that every coarser level corresponds to a 4 times decrease in the number of points, analogous to downsampling an image to half its resolution and getting 4 times less pixels. By this definition, the coarsest level contains 4 to 15 points inside the inpainting mask M_I. We extend our definition of K_{in} (and similarly K_{out}) as follows: $K_{in,l} = \lfloor 4^{-l} \rho A_{in} \rfloor$ for level $l \in [0..L]$.

Moreover, when inpainting the coarser levels we do not copy the Gaussians as described before. This



(a) Without gradient accumulation, before optimisation

(b) Without gradient accumulation, after optimisation

(c) With gradient accumulation, before optimisation

(d) With gradient accumulation, after optimisation

Figure 4.5: Effect of gradient accumulation during the optimisation phase. Note that the images before optimisation are different from each other because the gradient accumulation affected the intermediate inpainting results during previous iterations.

would introduce unintended details to the finer level, e.g. details that were not visible when rendered at the lower resolution but become visible when rendering at a higher resolution. Instead we perform inpainting rendering the nearest-neighbour patches and inserting Gaussians into the inpainting area based on the pixel values from those renders. For instance, given a point $\boldsymbol{p} \in P_{in,l}$ and its nearest neighbour $\boldsymbol{q} \in P_{out,l}$ for some $l > 0$, we render the patch at $\boldsymbol{q}$ to an image $I_{\boldsymbol{q}}$. For all except the last EM iteration of the level, $I_{\boldsymbol{q}}$ has a resolution of $R \times R$. In the last EM iteration of the level, the resolution is doubled to $2R \times 2R$, as a way of upsampling to the next finer level and introducing more detail in the inpainting region. After rendering $I_{\boldsymbol{q}}$ we insert ψ^2 Gaussians per pixel, where ψ is a parameter of the algorithm. We found that sometimes only inserting a single Gaussian per pixel restricted the optimisation phase and inserting multiple Gaussians offered the algorithm a degree of freedom when blending overlapping patches together. Therefore, we used $\psi = 2$ in our experiments. All Gaussians inserted for a patch are grid-wise aligned on the tangent plane at $\boldsymbol{p}$ and the ψ^2 Gaussians corresponding to a pixel are each assigned the colour of the pixel and a position that matches the pixel. Every Gaussian has the same scaling parameters, such that each group of ψ^2 Gaussians corresponding to a pixel effectively covers that pixel projected into world space. Lastly, we remove any new Gaussians outside the inpainting mask, such that we do not unintentionally alter the content outside the inpainting area. Table 4.1 provides a summary of the different dimensions in world space and rendering resolutions used throughout the algorithm.

Table 4.1: Summary of the different dimensions in world space and rendering resolutions used throughout the algorithm.

	World space	**Rendering resolution**
Compute NNF error	$(Ns_{px}) \times (Ns_{px})$	$R \times R$
Insert Gaussians (coarser levels)	$(Ns_{px}) \times (Ns_{px})$	$R \times R$
Copy Gaussians (finest level)	$(2s_{px}) \times (2s_{px}) \times (2s_{px})$	n/a
Optimisation	$(Ns_{px}) \times (Ns_{px})$	$R \times R$, if not last EM iteration of the level $2R \times 2R$, if last EM iteration of coarser level $8R \times 8R$, if last EM iteration of finest level

5 Results

This chapter discusses all the experiments we ran in order to evaluate the effectiveness of our proposed method. We study how the algorithm performs in various different settings and provide educational examples such as a comparison with image PatchMatch and use of a ground-truth mesh underlying the scene to gain more insight into the behaviour and limitations of the method.

5.1. Implementation

We implemented our method in Python, making use of PyTorch and the vanilla 3DGS code provided by Kerbl et al. [2023]. The versions we used are Python 3.9, PyTorch 2.2.0, and CUDA 11.8. All our experiments are run on the DAIC HPC cluster from TU Delft, using an A40 GPU with 48GB VRAM. For most scenes, significantly less resources can be used to run the algorithm. However, we found that some parts of our code are not optimised for minimum memory usage and caused an out-of-memory error for large scenes. To run our experiments on those scenes we simply moved some data from the GPU to the CPU and used more RAM. In our experiments, by default we performed 2 EM iterations per level of the multi-scale hierarchy with 25 NNF iterations. Moreover, in all experiments except an ablation experiment for the patch size we used a patch size of $N = 3$. The complete source code is available at: `https://github.com/adriaanpardoel/gs-patchmatch`.

5.2. Datasets

For a quantitative evaluation of our method we used the SPIn-NeRF [Mirzaei; Aumentado-Armstrong; Derpanis, et al. 2023] dataset, which consists of forward-facing scenes and includes image masks and ground-truth images where the object to be inpainted is removed. For qualitative evaluation we used six scenes (bicycle, bonsai, garden, kitchen, room, stump) from the Mip-NeRF 360 [Barron et al. 2022] dataset, which contains 360-degree scenes. We also used the bear statue scene from the Instruct-NeRF2NeRF [Haque et al. 2023] dataset and the Lego scene from the synthetic NeRF [Mildenhall et al. 2021] Blender dataset. Lastly we used meshes from Sketchfab[1] and Poly Haven[2] to analyse our algorithm in a more synthetic scenario. All datasets used except the SPIn-NeRF dataset do not contain inpainting masks, therefore, we defined our own masks for those scenes.

5.3. Quantitative results

This section presents our quantitative results on the SPIn-NeRF dataset. It is good to be aware, however, that even though the dataset contains "ground-truth" images where they removed the object in real life, there is not a single best solution, as many plausible inpaintings exist. Moreover, different metrics are used to compare images in order to evaluate the inpainting result, but it is debatable in how far these metrics truly represent the quality of the inpainting. Keeping this in mind, we still think it is valuable to give a quantitative evaluation for a quick first comparison to other methods.

[1] `https://sketchfab.com/`
[2] `https://polyhaven.com/`

Table 5.1: Comparison of LPIPS scores for SPIn-NeRF and our method. The result for SPIn-NeRF is taken from the original paper.

Method	LPIPS↓
SPIn-NeRF	**0.4662**
Ours	0.5088

Table 5.2: Quantitative evaluation of our method compared to Point'n Move. The results for Point'n Move are taken from the original paper.

Metric	Method	2	3	4	7	10	12	book	trash	mean
PSNR↑	Point'n Move	**18.48**	**18.04**	**20.88**	**21.40**	**19.75**	**16.62**	**22.28**	21.18	**19.83**
	Ours	17.80	13.85	18.71	18.20	17.95	13.30	21.62	**22.46**	17.99
FID↓	Point'n Move	**53.60**	**36.39**	**51.78**	**22.48**	**21.67**	**26.23**	**81.68**	**28.84**	**40.33**
	Ours	86.31	172.86	98.43	143.00	69.24	64.97	101.70	49.28	98.22
LPIPS↓	Point'n Move	0.4544	**0.2217**	0.3229	0.2858	0.2264	0.3352	0.2147	0.2301	0.2864
	Ours	**0.2619**	0.2622	**0.2763**	**0.2686**	**0.2225**	**0.3305**	**0.1828**	**0.1751**	**0.2475**

For our results we ran the algorithm on the SPIn-NeRF dataset with the number of Gaussians per point $\gamma = 250$. With these parameters we found that the algorithm was able to generate plausible inpaintings to a certain extent. Moreover, we did not use any global mask for these scenes, as the meshes we extracted from the scenes contain relatively little noise. The results are visualised in Fig. 5.13.

Possible baselines to consider for comparison are the methods discussed in Chapter 2 as the current state-of-the-art: SPIn-NeRF [Mirzaei; Aumentado-Armstrong; Derpanis, et al. 2023], OR-NeRF [Yin et al. 2023], GaussianEditor [J. Wang et al. 2024], Gaussian Grouping [Ye et al. 2023], and Point'n Move [J. Huang and H. Yu 2023]. We carefully inspected each paper to enable a fair comparison, ensuring we retrieve our metrics in the same manner as the paper we compare to.

SPIn-NeRF provides the FID and LPIPS scores of their inpainting algorithm on their own dataset. To give a quantitative comparison to their results, we compute the LPIPS score in the same manner by cropping the images to the bounding box of the provided inpainting mask and extending each side by 10% in every direction. However, they do not precisely describe how their FID score is calculated, therefore we do not consider this in our comparison. The LPIPS scores of our method compared to SPIn-NeRF are presented in Table 5.1. We see that based on the LPIPS score SPIn-NeRF outperforms our method.

OR-NeRF features a quantitative comparison to SPIn-NeRF, however, their reported numbers differ from the ones reported in SPIn-NeRF, indicating they computed the metrics in a different manner. Unfortunately, the paper does not explicitly mention how their metrics were calculated, therefore we cannot make a fair comparison to their numbers.

Interestingly, Point'n Move contains a quantitative comparison of their method against SPIn-NeRF and OR-NeRF, based on the numbers reported in the OR-NeRF supplement. On closer examination, it seems likely that they used yet another manner of computing the metrics, based on the much lower LPIPS numbers reported compared to the other methods. The paper mentions that they use the entire images for calculation of the metrics, therefore we compute our metrics similarly to compare our method with Point'n Move. This way, even though their comparison to the other methods might not be a fair one, we can still make a fair comparison of our method with theirs. These results are presented in Table 5.2. It should be noted that scene 1 and 9 from the SPIn-NeRF dataset are excluded from their results. We see that Point'n Move outperforms our method in terms of PSNR and FID, while ours outperforms theirs in terms of LPIPS. Our hypothesis is that since our inpainting is more noisy and the Point'n Move inpainting more "smooth" and averaged out, the peak pixel-wise error of our method is likely higher, resulting in the inferior PSNR score. As for the inferior FID score, we expect this is because the inpaintings generated by Point'n Move result in more realistic images, while our patch-based approach results in more discrepancies caused by a lack of structure in the inpainted area. In contrast, since our method copies other parts of the scene directly and replicates the texture relatively well, we likely obtain a good LPIPS score when comparing the entire images, as the inpainted region looks similar to other parts of the image.

While GaussianEditor and Gaussian Grouping also perform inpainting, their focus lies more on other editing tasks. Therefore, the papers do not provide quantitative results for the inpainting process, hence we do not provide a comparison to their methods here.

5.4. Qualitative results

This section contains our qualitative results. We do not only consider the final inpainting results but also the intermediate results of different parts of the algorithm in order to see how each part performs in practice.

First of all, the section features a qualitative comparison to numerous state-of-the-art methods based on our results on the SPIn-NeRF dataset. Moreover, in order to test our method on different datasets we also performed inpainting on the Mip-NeRF 360 dataset using custom masks defined by 3D spheres, as the dataset does not provide inpainting masks and the spheres are easy to define. Fig. 5.14 shows the original scene contents and the inpainting masks we defined, each designed to remove an object from the scene (e.g. the bicycle and the bench, the bonsai tree, the table and the patio, the Lego, the slippers, and the tree stump). For the global masks, we used the same spheres as the inpainting masks, except with a larger radius (2-4 times larger, varying per scene). This limits the inpainting to content from its direct surroundings and disregards all the noise farther away in the scene. The figure also contains the results of our inpainting with the number of Gaussians per point $\gamma = 25$. Later in this section we discuss how the patch size N and the number of Gaussians per point γ affect the results.

The last experiments in this section explore ways to improve the overall inpainting results of the algorithm, such as preprocessing the mesh extracted from the scene. To this end, we use different scenes for those experiments that specifically allow us to test certain scenarios.

5.4.1. Comparison to state-of-the-art

Fig. 5.1 shows how our method performs compared to numerous state-of-the-art methods. A quick look at these results shows that our methods is not quite on par with the current state-of-the-art and offers less visually pleasing inpainting results. The rest of this chapter will dissect the algorithm to analyse which parts perform well and which cause the relatively poor results compared to other methods.

5.4.2. Surface extraction

Since our method relies on SuGaR for surface extraction of the scene, the results of the SuGaR mesh extraction heavily affect the surface meshes produced by the first part of the algorithm. As can be seen in Fig. 5.2a, SuGaR can reconstruct detailed parts of the scene quite well but struggles with generating a smooth surface for the surrounding area. We observe that, probably also due to the shape of the Gaussians, the surface often becomes “blobby”. While this does not have to be a problem for our algorithm it does make it harder to find good nearest neighbours, as more patches will be considered that do not represent any meaningful part of the scene. In Fig. 5.2c we see that using Poisson reconstruction to fill the inpainting hole gives good results: the hole is filled and we end up

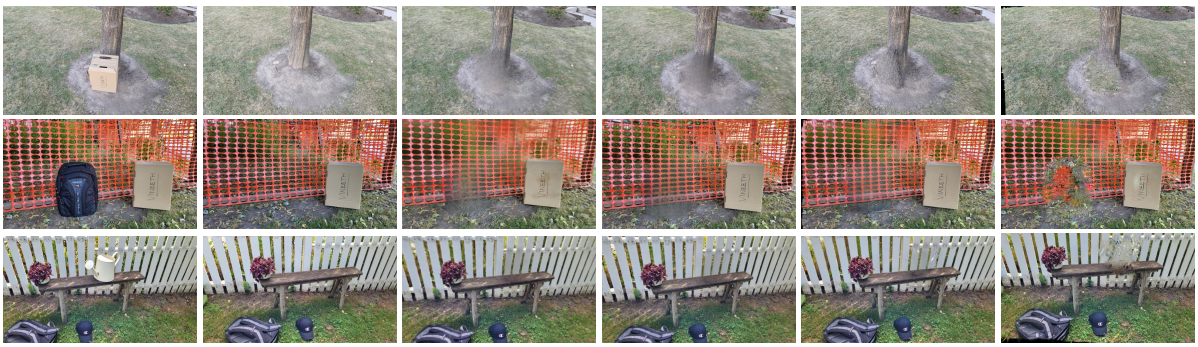
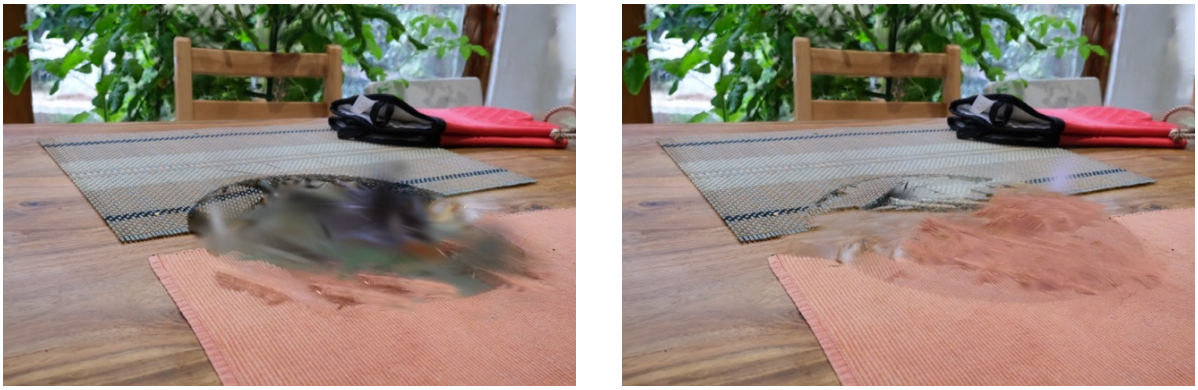


Figure 5.1: Comparison of our method to different state-of-the-art methods. Except for our own results, images are taken from the Point’n Move paper [J. Huang and H. Yu 2023].

(a) SuGaR **(b)** Masked **(c)** With Poisson reconstruction **(d)** Globally masked

Figure 5.2: Results of the surface extraction phase for the Mip-NeRF 360 kitchen scene. (a) The SuGaR mesh, (b) after removing the vertices inside the inpainting mask with the goal to remove the Lego from the scene, (c) after filling the hole with Poisson surface reconstruction, and (d) after removing vertices outside the global mask.



(a) Without global mask **(b)** With global mask

Figure 5.3: Effect of the global mask on the inpainting result for the Mip-NeRF 360 kitchen scene (with the number of Gaussians per point $\gamma = 40$). To show the effect better, we only performed 10 NNF iterations instead of the default 25.

with a smooth surface representation of the inpainting area. To get rid of the noise introduced by the SuGaR mesh and guide the nearest-neighbour search, it can be useful to define a global mask such that the algorithm will find better nearest neighbours faster. This results in a shorter execution time, as we can generally use less NNF iterations to achieve similar-quality inpainting results, but also enhances inpainting quality by excluding poor candidates from the nearest-neighbour search. Fig. 5.2d shows how a global mask is applied to the mesh. In this example we used spheres for the inpainting mask and the global mask, as we did in all of our experiments on the Mip-NeRF 360 scenes. The effect of the global mask on the inpainting result is shown in Fig. 5.3, clearly demonstrating a large difference.

5.4.3. Nearest-neighbour field

As the nearest-neighbour pairs in Fig. 5.4 demonstrate, the algorithm is able to find similar patches in the scene well. The patch renders within a pair generally look very similar and we can even see that some of the pairs feature structural elements that are well matched. This is exactly how we would expect the algorithm to behave.

Moreover, Fig. 5.5a shows that the algorithm is able to find coherent nearest neighbours to a certain degree, e.g. groups of nearby patches inside the inpainting mask are mapped to groups of nearby patches outside the mask. Fig. 5.5b substantiates this by showing that the directions of the mapping are clustered quite consistently.

Looking closely at the arrows in Fig. 5.5a, we also observe that sometimes multiple patches inside the mask are mapped to the same target patch. This can cause artefacts in the inpainting result where a patch is clearly repeated multiple times.

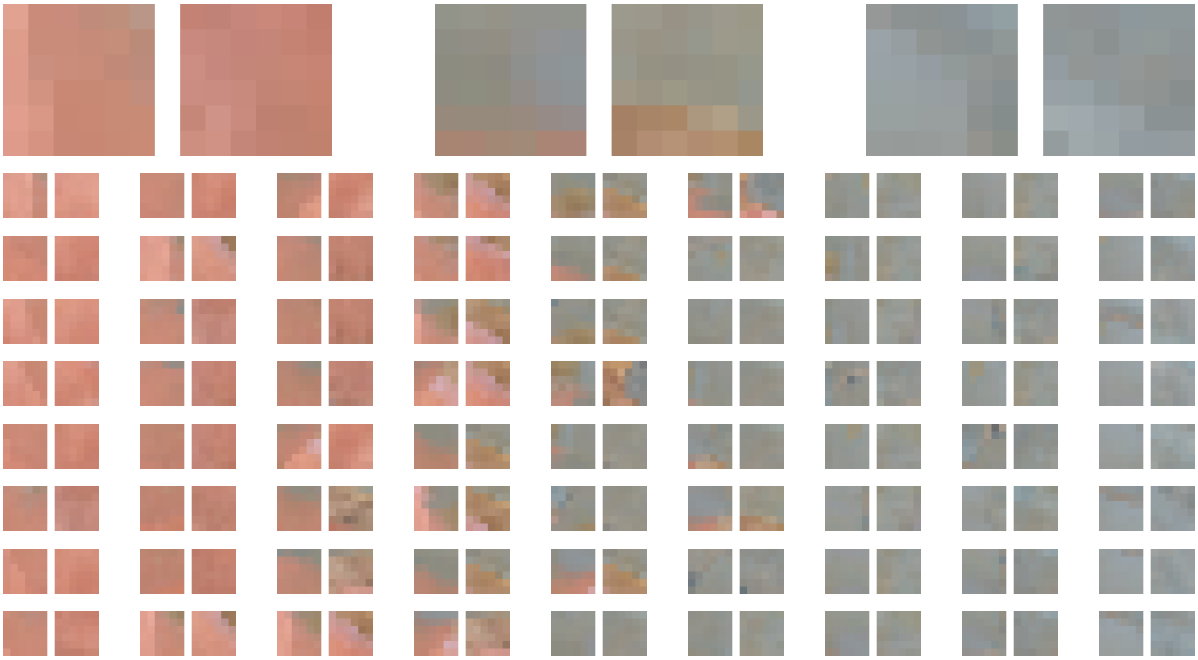
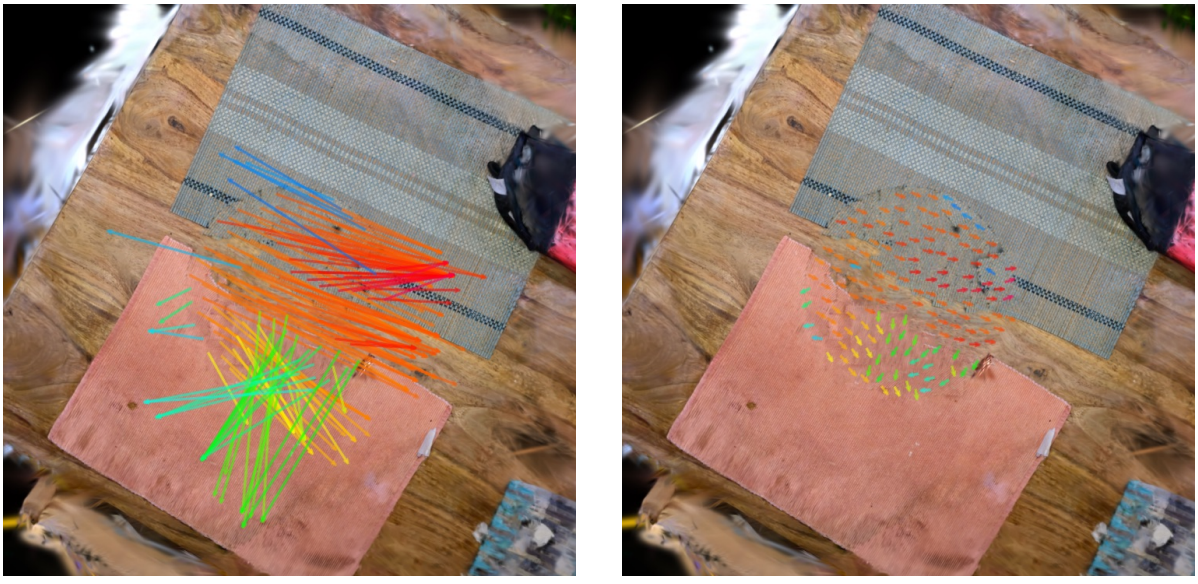


Figure 5.4: $R \times R$ renders of NNF patches during inpainting of the Mip-NeRF 360 kitchen scene ($N = 3, \phi = 2, R = 6$). Each pair of images consists of a patch inside the inpainting mask and its nearest-neighbour patch outside the mask. To make the figure more compact, we show only half of the actual patches of this NNF. The patches are ordered by hue. The three pairs at the top are magnified for ease of inspection.

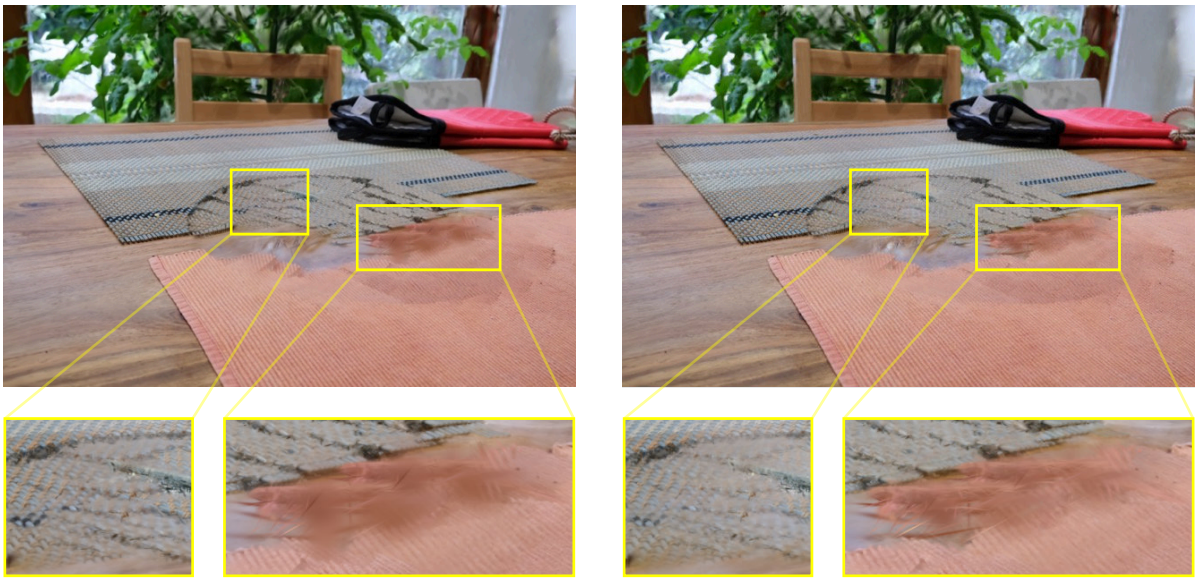


(a) **(b)**

Figure 5.5: Results of an NNF construction during inpainting of the Mip-NeRF 360 kitchen scene. The colours of the arrows represent their 2D direction as viewed in the images. (a) The mapping given by the NNF and (b) a vector field representation of the NNF.

5.4.4. Optimisation phase

Fig. 5.6 shows the inpainting results in the last iteration before and after the optimisation phase. As we can see, the optimisation does not have a huge impact on the structure and texture synthesis of the algorithm, but rather changes the overall appearance of the inpainted region slightly to make patches blend better together. On closer inspection of the NNF before and after optimisation, as depicted in Fig. 5.7, we see that the algorithm ensures that in some cases the patches inside the inpainting region look more similar to their nearest neighbour after optimisation, while in other cases they look more blurred in order to improve the blending with other patches.



(a) Before **(b)** After

Figure 5.6: Inpainting results in the last iteration before and after optimisation.

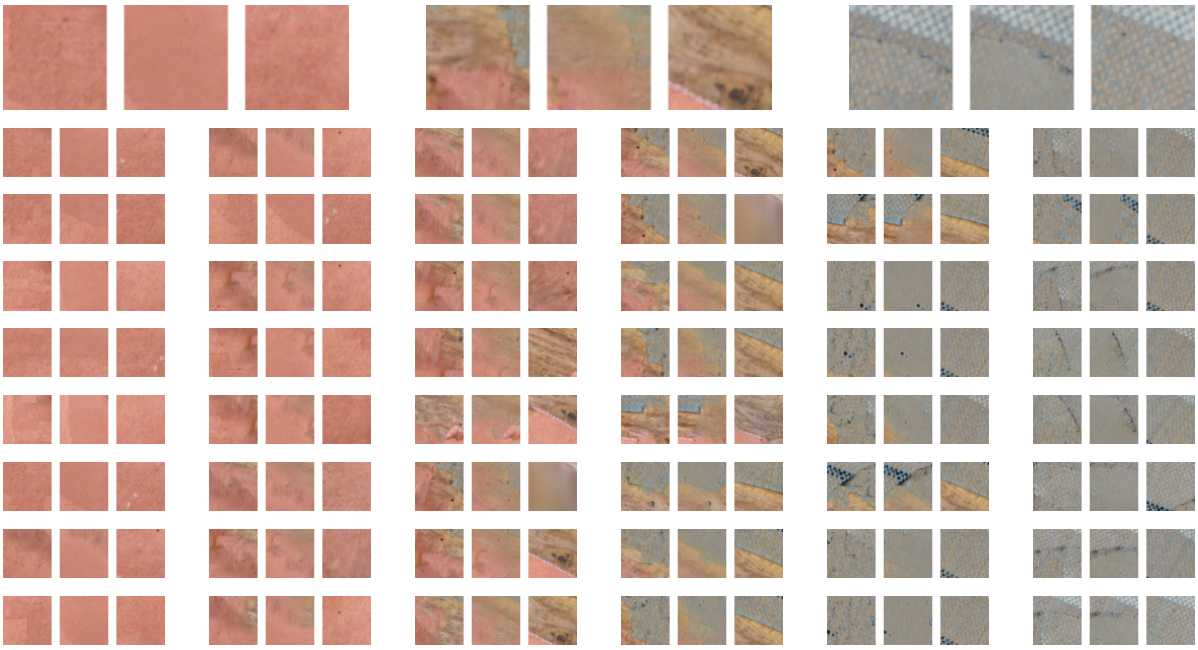


Figure 5.7: $8R \times 8R$ renders of NNF patches during inpainting of the Mip-NeRF 360 kitchen scene, before and after the optimisation phase ($N = 3, \phi = 2, R = 6$). Each triplet consists of (from left to right): a patch at $\boldsymbol{p} \in P_{in}$ before optimisation, the same patch after optimisation, and the patch at its nearest neighbour $\boldsymbol{q} \in P_{out}$. To make the figure more compact, we show only one third of the actual patches of this NNF. The patches are ordered by hue of the nearest-neighbour patch. The triplets in the top row are magnified for ease of inspection.

5.4.5. Ablations

To study the effect of different parameters of our algorithm, we ran numerous ablations. Specifically, we experimented with the number of Gaussians per point γ and the patch size N.

Number of Gaussians per point

We found that a good setting for the number of Gaussians per point γ is highly scene-dependent. To study the effect of the parameter, we ran an ablation experiment on the Mip-NeRF 360 dataset keeping all other parameters constant while varying the number of Gaussians per point $\gamma \in \{25, 50, 100\}$. This value affects the number of points sampled in the finest level of the hierarchy, thereby also affecting the number of levels in the hierarchy and the number of points in each level. The inpainting results for the different values of γ are shown in Fig. 5.15.

From the results we see that for larger values of γ the algorithm struggles more with generating a plausible structure. This is likely because larger patches of Gaussians are copied per point, and as patches contain their own structures they fit together less well. This effect is clearly visible in the room scene, where parts of the legs of a chair are unintentionally copied due to the larger patches for copying.

Moreover, the garden scene shows how γ affects the texture synthesis. For smaller values of γ we see that the inpainted texture contains more small repetitions, while for larger values we get fewer repetitions, although the repeated areas are larger. Depending on the texture to be inpainted, the parameter can be adjusted to generate the desired results.

Patch size

To study the effect of the patch size N, we ran an ablation experiment on the Mip-NeRF 360 dataset keeping all other parameters constant, with the number of Gaussians per point set to $\gamma = 25$, while varying the patch size $N \in \{3, 5, 7\}$. This change in N affects the NNF search, the inserted Gaussians at coarser levels of the hierarchy, and the optimisation loss. It does not influence the window size for copying Gaussians in the finest level of the hierarchy. The inpainting results for the different patch sizes are shown in Fig. 5.16.

Although in some instances a larger patch size can provide better results, such as for the room scene, we see that often in the early stages of the algorithm it causes too much global information to be taken into account. We see that in some scenes, such as the bicycle and kitchen scenes, the border of the inpainting area becomes more defined and clearly stands out. In contrast, for the room scene we see that with the small patch size too little global information is taken into account and at the coarsest level the patches at the centre of the inpainting area do not extend beyond that area. As the algorithm has no information about what to put there, the patch is mostly black and gets matched with some dark patch outside the inpainting area, resulting in the arbitrary inpainting result we see in the render.

From these ablations we conclude that there needs to be a balance between the number of Gaussians per point γ and the patch size N to result in the desired inpainting result. As good values for these parameters vary highly per scene, one needs to try different settings to obtain a good inpainting.

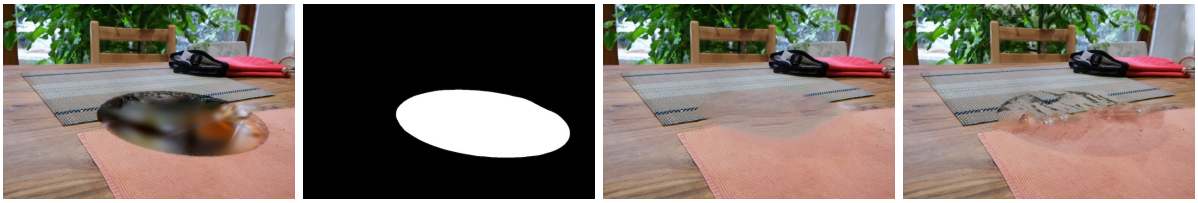
5.4.6. Depth

While we deal with mostly flat inpainting regions given the masks we defined for the Mip-NeRF 360 dataset, our results on the SPIn-NeRF dataset contain some more 3D inpainting regions. Looking at the depth maps in Fig. 5.13, we see that the performance of our method in this regard highly varies per scene. The largest factors affecting this are probably the quality of the SuGaR mesh and the Poisson reconstruction of the inpainting hole. We observe that the SuGaR mesh often contains some noise, resulting in artefacts in the inpainting. An example of this is visible in the depth map the scene with the tree (second row), where the dark red noise is caused by a floater in the mesh. In other scenes we see that our method struggles with the object removal based on the provided inpainting masks from the dataset, resulting in artefacts that can be seen in the depth maps of the scenes with the stone bench (first row) and the camping chair (third row from the bottom), where remainders of the removed object are still visible. For some scenes, such as the one with the book (second row from the bottom), we also see smooth depth maps that mostly meet our expectations.

5.4.7. Image PatchMatch comparison

Since our algorithm is based on the image PatchMatch algorithm, it also suffers from similar limitations. To analyse the similarities and differences between the two we compare them by inpainting one of the views in 2D using an open-source PatchMatch implementation[3], as shown in Fig. 5.8. We can see that the algorithms, despite giving quite a different-looking output, suffer from similar issues. As known about PatchMatch and Fig. 5.8c verifies, we see that the algorithm performs well for texture synthesis but performs poorly for structure synthesis. For both our algorithm and image PatchMatch we see that parts of the placemats are extended and look similar to the rest of the placemat, but both algorithms struggle to complete the borders of the placemats. Since we expect the 3D inpainting task to be more difficult than image inpainting, it is not a big surprise that our algorithm struggles to complete these types of scenes well. The major difference that we see between the image version and the 3D version is that the image version is able to synthesise more coherent areas, whereas our algorithm shows clearer signs of overlapping patches and looks more “messy”. In the image version, although the structure of the placemats is still not replicated well, there is a more coherent texture in the middle.

For a more complete analysis, we also ran the image PatchMatch algorithm on test views from the SPIn-NeRF dataset. These results are shown in Fig. 5.9. It should be noted that the output heavily depends on the chosen patch size and we only show the best result in the figure. Moreover, we found that a good choice for the patch size widely varies between different scenes. While the inpainted images generally look better than our 3D inpainting results, we again see that for most scenes the PatchMatch algorithm struggles with completing structures well. Since the quality of the output greatly depends on the parameters of the algorithm, it might be the case that our algorithm is also able to deliver better results when tweaking the parameters for each scene individually.



(a) Image **(b)** Mask **(c)** PatchMatch **(d)** Ours

Figure 5.8: Comparison of our method with PatchMatch image inpainting. (a) The image used to inpaint with PatchMatch, (b) the mask used to inpaint with PatchMatch, (c) the PatchMatch image inpainting result (with patch size 3), and (d) our 3D inpainting result (with the number of Gaussians per point $\gamma = 25$).

Figure 5.9: Results of PatchMatch image inpainting on the SPIn-NeRF dataset. For every scene we tried patch sizes 3, 5, 7, 9, 11, 13, 15, 17, and 19. We manually selected the output image we deemed most realistic.

[3] `https://github.com/mauwii/PyPatchMatch`

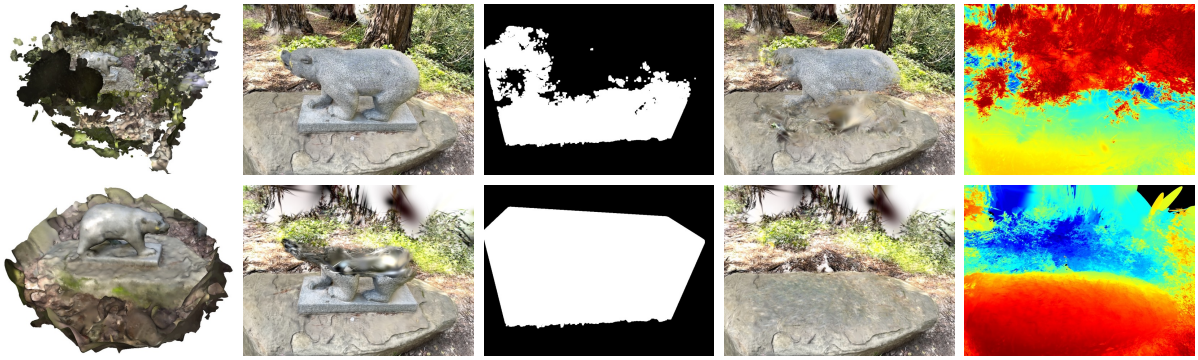
5.4.8. Scene preprocessing

In previous experiments we saw that the surface mesh extracted from a scene heavily influences the performance of our algorithm. Therefore, we explore in how far preprocessing of the scene can improve the output. We ran this experiment on the bear statue scene from the Instruct-NeRF2NeRF [Haque et al. 2023] dataset, which clearly highlights the effect that preprocessing can have. For this scene we defined quite precise masks using 3D boxes. As for the preprocessing of the scene, we manually perform the following three steps using the SuperSplat[4] editor for the 3D Gaussians and MeshLab[5] for the surface mesh:

1. We remove noise from the 3D Gaussian representation.
2. We remove noise from the mesh.
3. We apply Laplacian smoothing to the mesh.

As shown in Fig. 5.10, preprocessing the scene can dramatically improve the results of our algorithm. In the experiment for the scene without preprocessing we set the number of Gaussians per point $\gamma = 50$ while for the preprocessed scene we chose $\gamma = 600$. The large difference in the setting for γ is because during the preprocessing we trimmed the mesh much more than the Gaussians, resulting in a higher average number of Gaussians per surface area.

Preprocessing step 1 and 2 are essentially equivalent to applying a more precise global mask to the scene. Moreover, step 1 is necessary in this case because the trained 3D Gaussians contain inaccuracies: some Gaussians that show up as part of the bear statue are actually placed elsewhere in 3D space. Because of that, when we run our algorithm without preprocessing, we see that our inpainting mask still leaves parts of the statue in place. Step 3 is performed to ensure that our surface better represents the actual contents of the scene and the surface normals are more accurate, which is necessary to render patches from their correct direction. The surface extracted by SuGaR is often noisy and uneven, although we often expect a more flat, smooth output. Since all preprocessing steps taken essentially boil down to removing inaccuracies from the 3D Gaussians and the surface mesh, we argue that as code for 3DGS training and surface extraction improves, the performance of our algorithm will automatically improve with it.



(a) Mesh representation **(b)** 3DGS representation **(c)** Mask (projected) **(d)** Inpainting result **(e)** Depth map

Figure 5.10: Results of our algorithm with and without preprocessing of the scene. The top row is without preprocessing ($\gamma = 50$), the bottom row with preprocessing ($\gamma = 600$).

[4] https://playcanvas.com/products/supersplat
[5] https://www.meshlab.net/

5.4.9. Ground-truth mesh

As we saw in previous experiments, the results of our algorithm are heavily impacted by the mesh extracted by SuGaR. To gauge how much this affects the algorithm we created two 3DGS scenes based on meshes, such that we already have a ground-truth mesh and do not need to run SuGaR. In order to train 3DGS scenes based on the meshes, we used the open-source Blender add-on BlenderNeRF[6]. We obtained two different meshes to perform experiments on: *St. Luke's United Church in Belledune* (created by Air Digital Photogrammetry under CC Attribution licence) from Sketchfab[7] and *Coast Rocks 03* (created by Rico Cilliers and Rob Tuytel under CC0 licence) from Poly Haven[8].

Fig. 5.11 shows the results we were able to obtain on these scenes. For *St. Luke's United Church in Belledune* we set $\gamma = 75$ and for *Coast Rocks 03* we set $\gamma = 100$. As one can see, our algorithm performs far better on these scenes than on most of the standard datasets we saw before. Notably, these results were produced without need of a global mask. The high quality of the mesh thus clearly makes a difference. Moreover, as the Gaussians were trained using the mesh, the Gaussians are more regularised (e.g. surface-aligned). This ensures that less noise is introduced when we copy Gaussians from elsewhere in the scene. We conclude that our patch-based algorithm performs best when the Gaussians are well aligned to the surface and the surface is approximated well by a mesh.

(a) Original scene **(b)** Mask (projected) **(c)** Masked scene **(d)** Inpainting result **(e)** Depth map

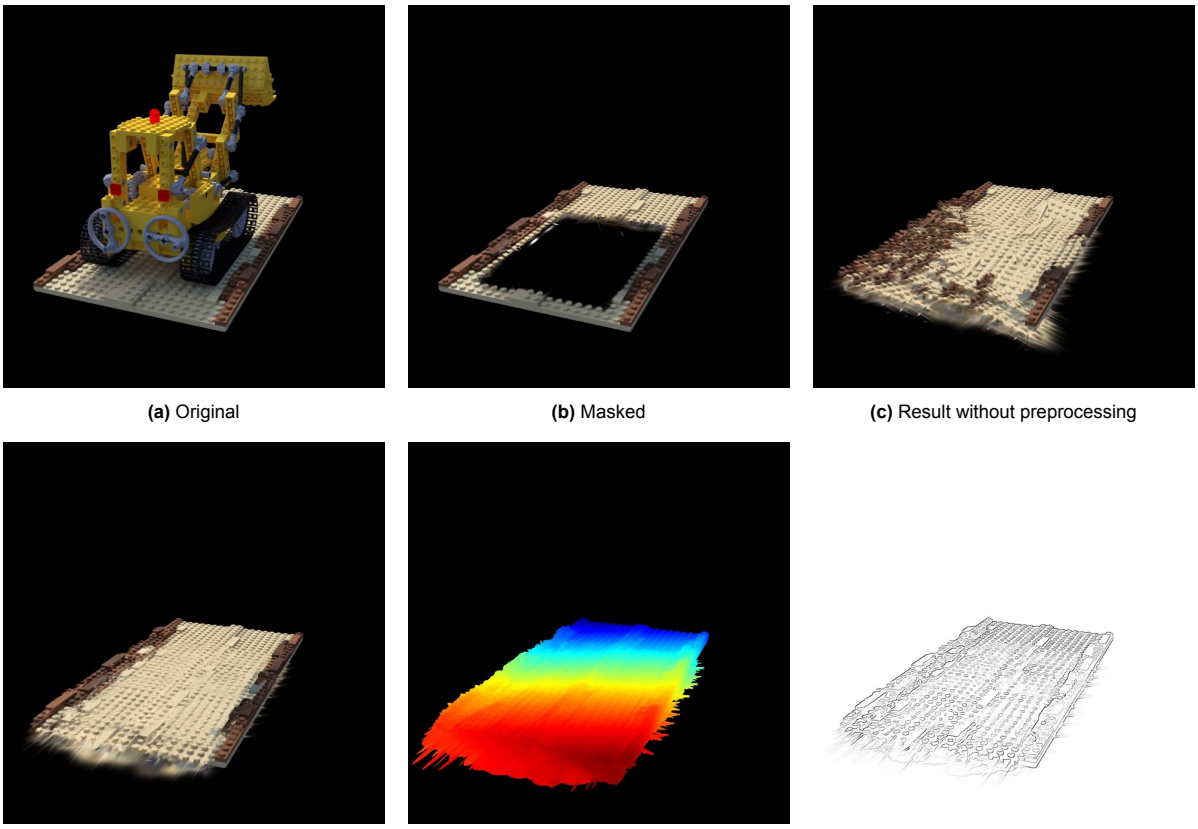
Figure 5.11: Results of our algorithm on scenes based on ground-truth meshes.

5.4.10. 3D texture synthesis

Lastly, we consider the ability of the algorithm to inpaint 3D textures. Our idea was that the Poisson reconstruction will provide a smooth surface to tell the algorithm which area needs to be inpainted and by copying groups of Gaussians from elsewhere in the scene we will preserve the 3D texture that the Gaussians represent.

As can be seen in the bottom row of Fig. 5.11, the inpainted region contains rocks similar to the other parts of the scene. This shows that our algorithm is capable of synthesising 3D texture rather well, although this case is still limited in the sense that it is based on one of the cases with a ground-truth mesh underlying the scene. Therefore, we conduct one more experiment, namely on the Lego scene from the synthetic NeRF [Mildenhall et al. 2021] Blender dataset. Although this is a synthetic scene, we do not use a ground-truth mesh but instead rely on SuGaR to extract the mesh from the scene, in line with our proposed method. We chose this scene as the regularity of the Lego bricks forms a challenging case. In contrast, the structure of the rocks we discussed before is not as regular, generally making it an easier case for inpainting, since the inpainting region is allowed to contain more randomness. The results on the Lego scene are shown in Fig. 5.12. We ran the experiments with $\gamma = 1100$. As can be seen in Fig. 5.12c, the algorithm struggles completing the surface well. However, by setting all the vertex normals of the extracted mesh to the vector pointing up from the Lego plane, we ensure that the surface is completed more flatly and the Gaussians are copied in their expected orientation. As shown in Fig. 5.12d, this gives better results and the 3D texture of the Lego bricks is replicated well.

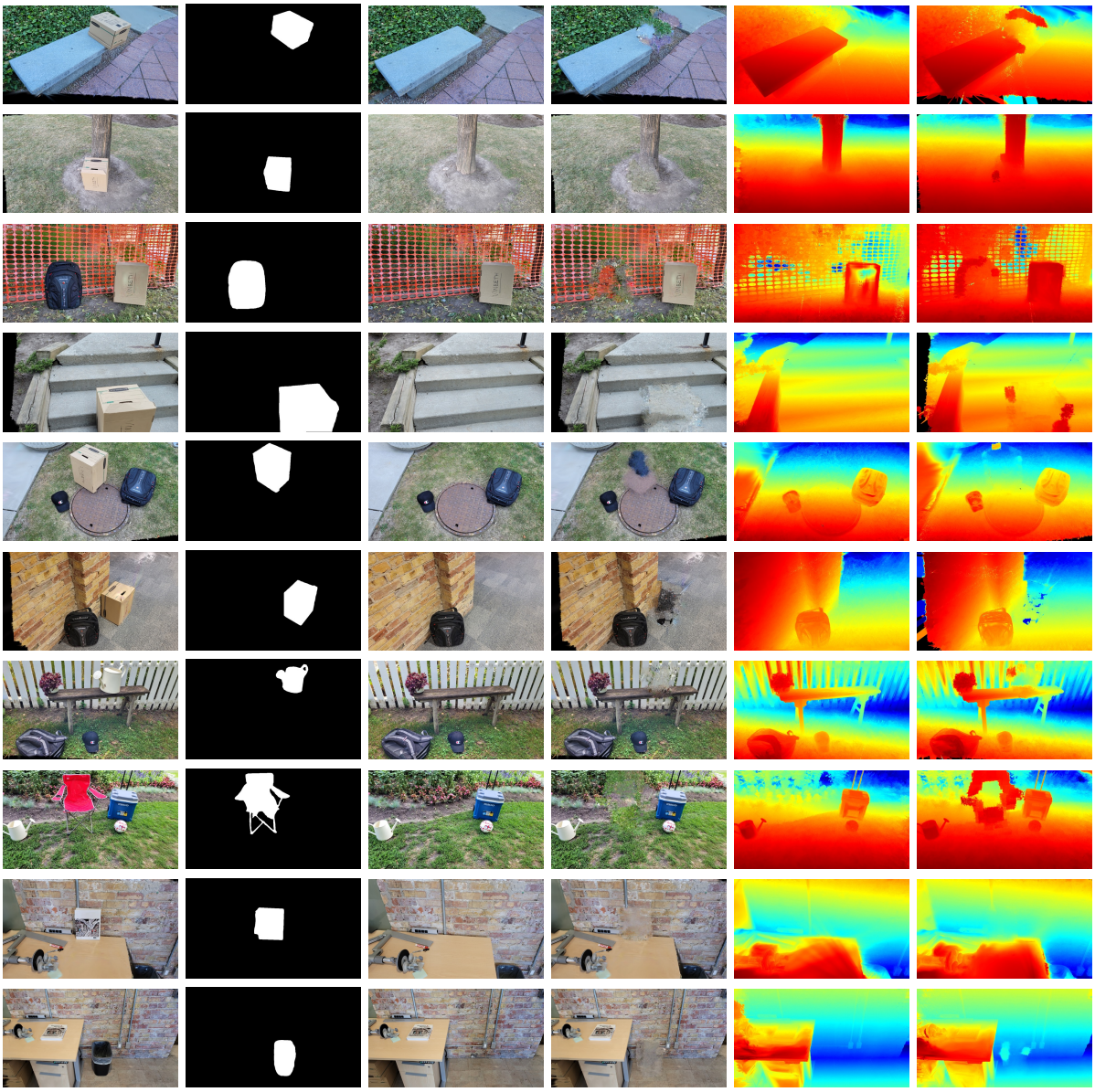
[6] https://github.com/maximeraafat/BlenderNeRF
[7] https://sketchfab.com/3d-models/st-lukes-united-church-in-belledune-8eae8eb9425b44cfb2287b016056e745
[8] https://polyhaven.com/a/coast_rocks_03



(a) Original **(b)** Masked **(c)** Result without preprocessing

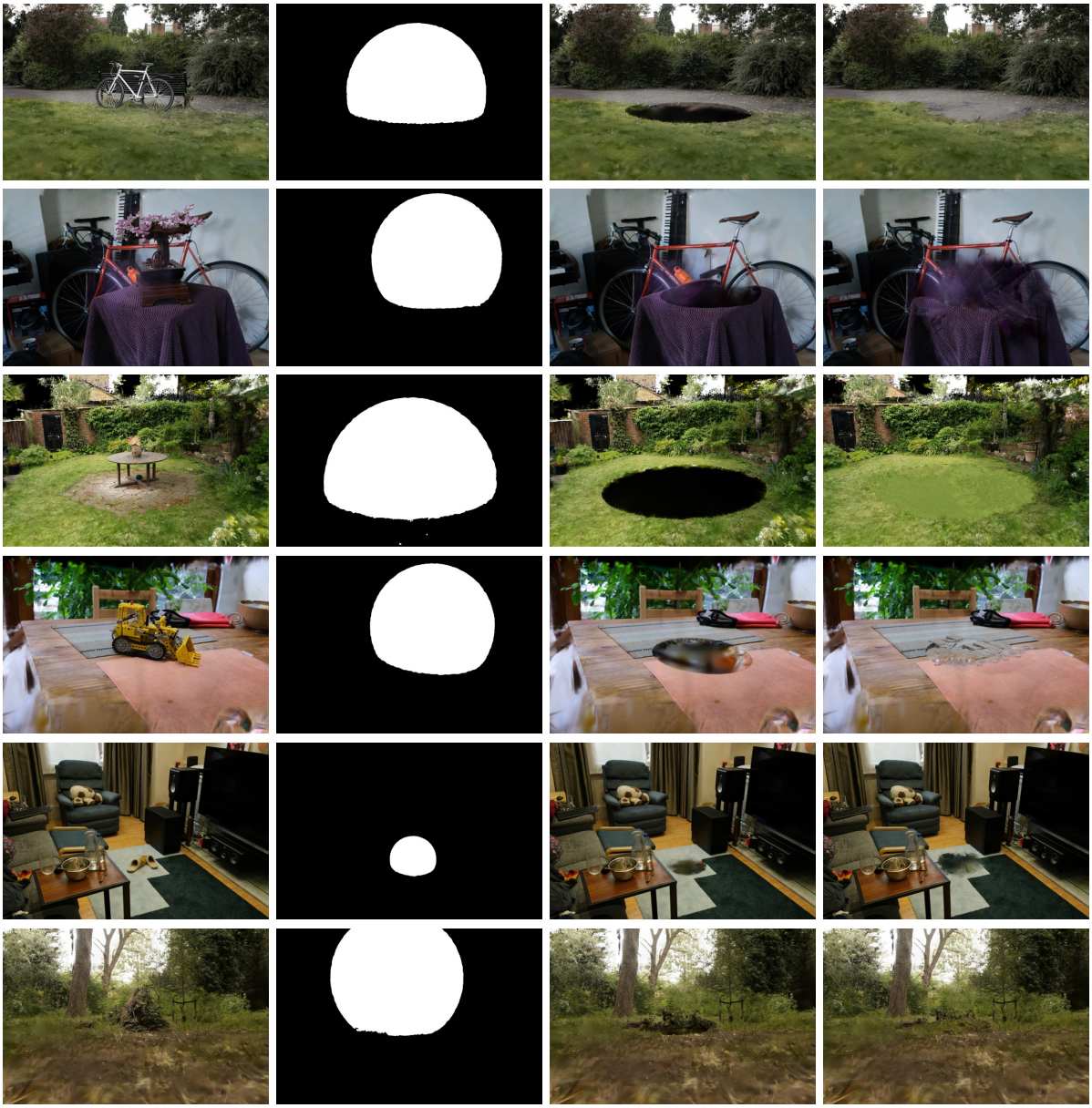
(d) Result with preprocessing **(e)** Depth map **(f)** Edges

Figure 5.12: Results of our algorithm on the Lego scene from the synthetic NeRF [Mildenhall et al. 2021] Blender dataset (with the number of Gaussians per point $\gamma = 1100$).



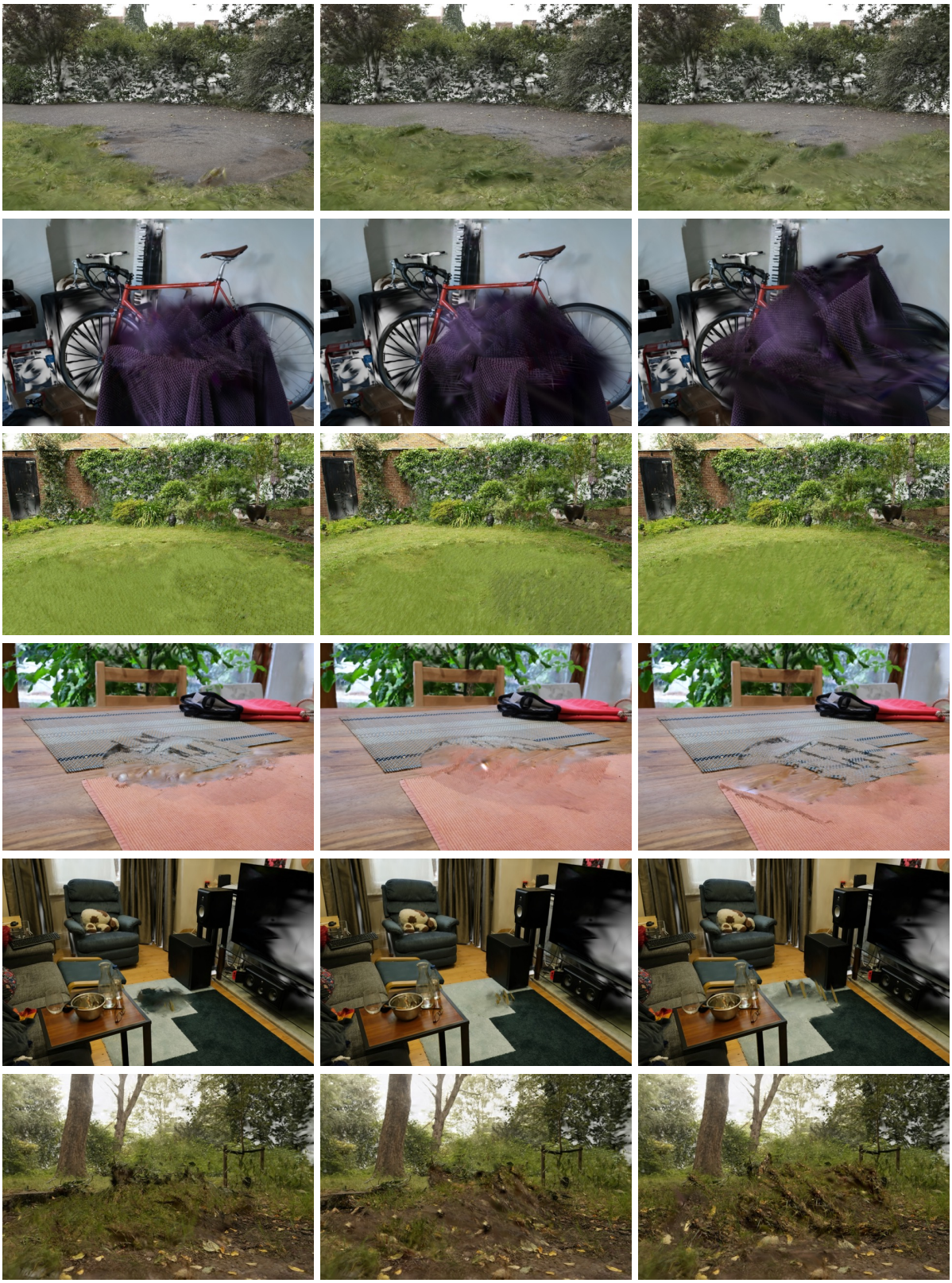
(a) Original **(b)** Mask **(c)** Ground truth **(d)** Ours **(e)** Ground-truth depth map **(f)** Ours depth map

Figure 5.13: Results on SPIn-NeRF dataset with the number of Gaussians per point $\gamma = 250$. (a) The original 3DGS scene rendered from one of the test views, (b) projection of the 3D inpainting mask generated by SPIn-NeRF's multi-view segmentation, (c) the ground-truth image with the object physically removed from the scene, (d) our inpainting result, (e) a depth map of the ground truth, and (f) a depth map of our inpainted scene.



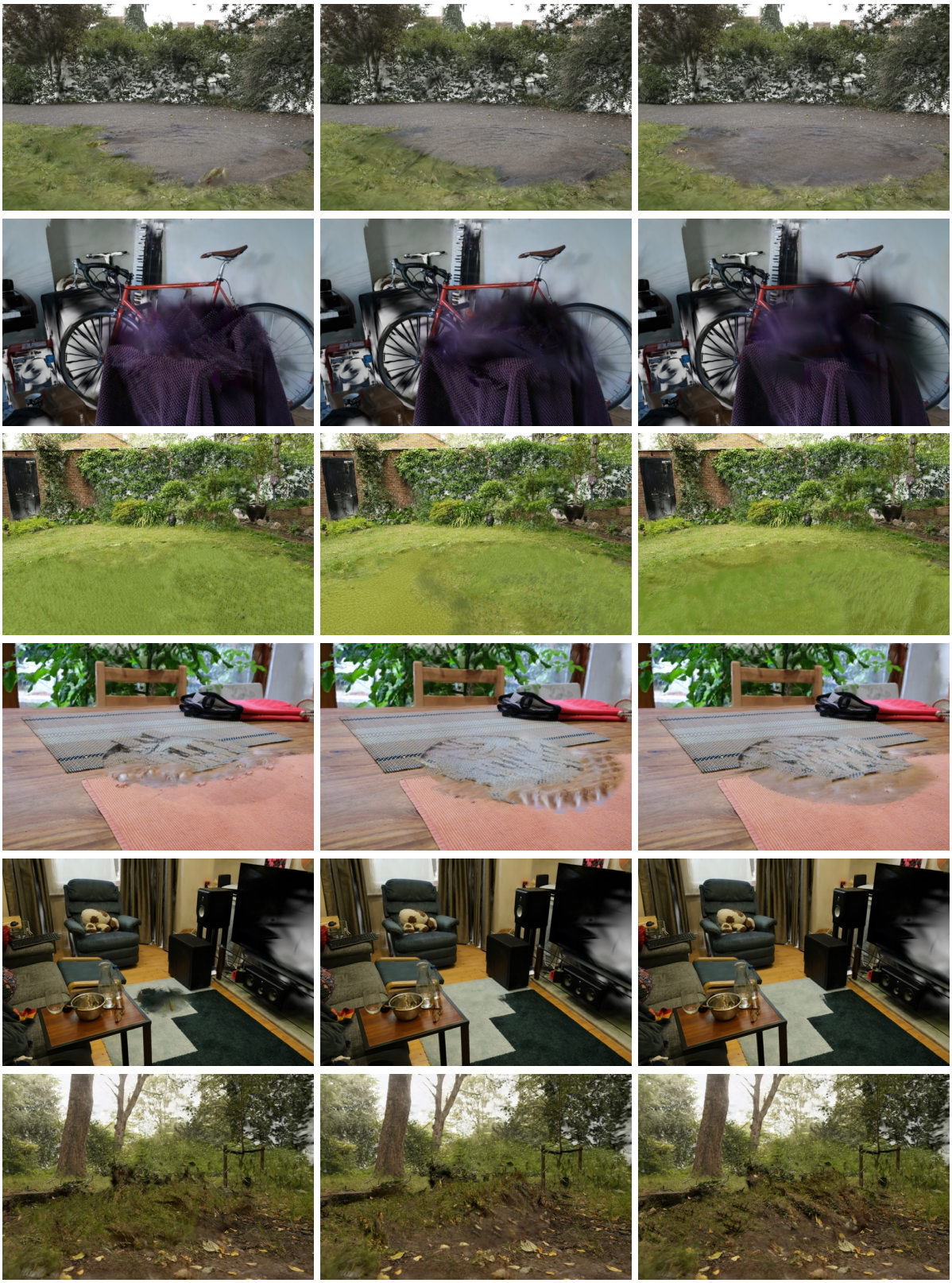
(a) Original scene **(b)** Inpainting mask (projected) **(c)** Masked scene **(d)** Inpainted scene

Figure 5.14: Our results on the Mip-NeRF 360 dataset. (a) The original scene, (b) a projection of the spherical mask we defined inside the scene, (c) the masked scene, and (d) our inpainting result with the number of Gaussians per point $\gamma = 25$.



$\gamma = 25$ $\gamma = 50$ $\gamma = 100$

Figure 5.15: Comparison of results for different values of the number of Gaussians per point γ.



$N = 3$ $N = 5$ $N = 7$

Figure 5.16: Comparison of results for different values of the patch size N (with the number of Gaussians per point $\gamma = 25$).

6 Discussion

This chapter gives an overview of how we arrived at our final method through different experiments and discusses the limitations of the method. Moreover, we identify several opportunities for future work, including patch rotation and combinations with other methods.

6.1. Process

This section highlights some of the different approaches we experimented with to perform 3D inpainting.

6.1.1. 2D prototype

In order to determine whether a patch-based algorithm for 3DGS inpainting could work, we first created a 2D prototype based on an open-source 2D Gaussians splatting implementation[1]. Instead of training a 3D scene based on multiple viewpoints, the 2D version is trained on a single image and the 2D Gaussians are fitted to resemble the training image. For the 2D prototype, we used an open-source PatchMatch implementation[2] to construct an NNF based on a masked image and then use the NNF to copy patches of Gaussians to the inpainting area. The results of our 2D prototype are shown in Fig. 6.1. While the implementation applies the same principles as 3DGS, the blending order of overlapping Gaussians is undefined since there is no depth in the 2D setting. In the 2D implementation the colour values of overlapping Gaussians are simply added together to compute the final pixel colour. As can be seen in Fig. 6.1d, this results in artefacts, making some parts of the inpainted area appear brighter than they should. Since these artefacts are caused by a known limitation of the 2D Gaussian splatting implementation which is not present in the 3D version, we saw potential for a patch-based inpainting algorithm to provide solid results in the 3D setting.

(a) Original **(b)** Masked **(c)** PatchMatch result **(d)** 2D prototype result

Figure 6.1: 2D prototype performing patch-based inpainting of a 2D Gaussian splatting “scene”.

[1] https://github.com/OutofAi/2D-Gaussian-Splatting
[2] https://github.com/mauwii/PyPatchMatch

(a) Original 3D scene

(b) Cropped

Figure 6.2: "2D" 3DGS scene.

6.1.2. Voxel-based approach

After creating a promising 2D prototype we moved onto a slightly more 3D setting, such that we could evaluate the feasibility of a patch-based inpainting algorithm with the actual 3DGS code. We created a "2D" 3DGS model by cutting a canvas painting out of an actual 3D scene of an art studio[3], as shown in Fig. 6.2. Technically this painting is a 3D scene, however, since it is mostly flat and we always look at it from the same direction, this simplifies some of the challenges we face in the 3D context, such as normal estimation.

We initially opted for a voxel-based approach in our algorithm, as this is the most natural translation from 2D to 3D, allowing us to apply PatchMatch in the exact same way as the original algorithm only with an extra dimension. For the painting scene this meant we could use an $N \times N \times 1$ voxel grid and we could render all (patches of) voxels from the same direction, e.g. the direction looking at the painting. Most details of the voxel-based algorithm are similar to our final method:

- Patches are rendered to images such that they can be compared pixel-wise in order to construct the NNF (the difference here being that as we move to actual 3D voxel patches, we render every voxel to a 1×1 image to get a voxel colour value).
- After construction of the NNF, Gaussians are copied and then optimised by performing training iterations based on patch renders inside the inpainting mask and their ground-truth nearest-neighbour patch renders.
- A multi-scale approach is enabled by inserting Gaussians inside the inpainting mask with their colours based on the pixel values of the nearest-neighbour patch renders.

Fig. 6.3 shows how our voxel-based algorithm is able to remove a person from the painting. Looking closely at the results we see that the algorithm generates a plausible inpainting and that the optimisation phase ensures a more natural blend between the copied Gaussians and their surroundings.

When we moved on to actual 3D scenes with our voxel-based approach we encountered numerous challenges. For testing our algorithm we first created a simple scene in Blender consisting of a textured cube. It was immediately noticeable that running the algorithm on the 3D voxel grid (e.g. $N \times N \times N$ voxels) drastically slowed down the execution, making it difficult to test and debug our code. Therefore, we first profiled the code and found that one of the main bottlenecks was determining which Gaussians to render for a given voxel. We implemented numerous optimisations such as a vectorised box-ellipsoid intersection algorithm and an octree acceleration structure to make this process faster and reduce the bottleneck. In our experiments on the textured cube scene we were able to speed up execution time of the algorithm by about 90% with these improvements, allowing us to actually test and debug the algorithm within reasonable time. When carefully inspecting the workings of the algorithm step by step, the larger issue with the voxel-based approach became apparent: inside the cube (e.g. behind the surface), some "random" Gaussians are present, resulting in voxel values that are not particularly well-defined. These values result in oddities once we start comparing 3D voxel patches and eventually cause poor nearest-neighbour matches. Since 3DGS scenes often represent surfaces, we expected that this issue would occur for most other scenes as well. Therefore, we considered an approach based

[3] `https://poly.cam/capture/17b53e95-96bf-42a3-9a20-b4eb06fb0a84`

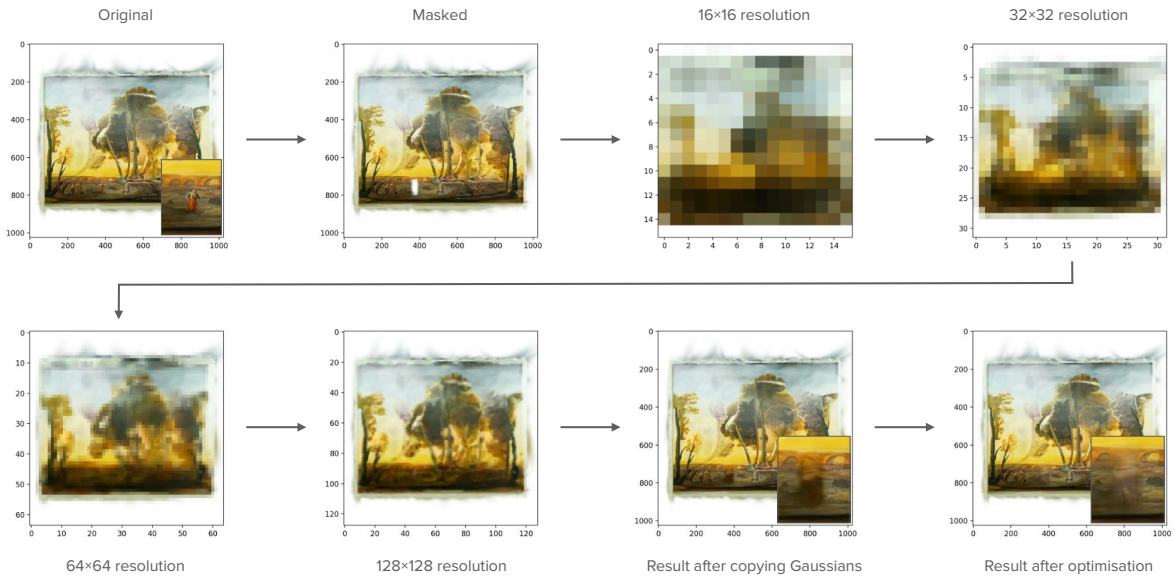


Figure 6.3: Inpainting process for removal of a person from the “2D” 3DGS painting scene. Zoom in to see the magnified inpainting area at the bottom-right of the plots of the original and the results.

on surface extraction to be more reliable than the voxel-based approach. This idea resulted in the final method presented in Chapter 4.

6.2. Limitations and future work

From our results we see that generally the NNF is quite well constructed and the algorithm is able to identify similar-looking parts of the scene. One of the biggest limitations in the patch matching is that we always assume the same up-vector, restricting the search space for nearest-neighbour candidates. A possible improvement would be to allow rotation of patches, such that we could also identify matches when they are oriented differently throughout the scene. This might also help to ensure that patches align better when copying Gaussians based on the NNF. Similar to the generalised PatchMatch algorithm by Barnes; Shechtman; Goldman, et al. [2010], the NNF search space could be extended to include not only an offset but also a rotation. The propagation and random search of our algorithm could be extended in a similar way as theirs in order to support the rotation.

Moreover, as our method heavily depends on SuGaR for mesh extraction, it also suffers from the same limitations. Although SuGaR is able to extract high-quality meshes for some 3DGS scenes, it is not guaranteed that the meshes will be of good quality and it often requires the user to tweak the parameters of the algorithm to get better results. During our experiments we saw that the meshes often contain a lot of noise, making the use of a global mask in our method a necessity rather than an optional improvement. We also saw that for scenes based on meshes the algorithm performed far better, even without a global mask. To improve our results on other scenes, better mesh extraction algorithms for 3DGS are needed. Alternatively, the user can use the SuGaR mesh as a starting point and manually refine the mesh to achieve better results with our method.

Furthermore, since we rely on Poisson reconstruction to fill the hole in the mesh, our method has a limited capacity to recognise bigger structures in the scene. The Poisson reconstruction often produces a rather smooth surface, making it harder to inpaint parts of scenes with sharper edges well. Despite struggling with these larger 3D structures, the algorithm is capable of inpainting non-planar details.

Moreover, as seen in the results, similar to PatchMatch the algorithm performs decently for texture synthesis but struggles with structure synthesis. This problem could be mitigated by using an inpainting prior before running our algorithm, such that the structure of the inpainting is guided mostly by the prior. One of the previously mentioned methods that use LaMa image inpainting to inpaint 3DGS scenes [J. Wang et al. 2024; Ye et al. 2023; J. Huang and H. Yu 2023] could be employed first as the inpainting

prior, after which some of the blurry artefacts could be fixed by our method, providing a better texture.

Lastly, it would be interesting to see if our previously discussed voxel-based approach could perform better when combined with SuGaR's hybrid representation combining a triangle mesh with 3D Gaussians. In the hybrid representation Gaussians are placed strictly inside triangles of the refined mesh, e.g. on the extracted surface. Since this should ensure that no "random" Gaussians are present behind the surface, which greatly hindered our voxel-based approach, it could be an opportunity for this approach to provide improved results.

7

Conclusion

In our work we have shown that a patch-based algorithm for 3D inpainting is feasible. However, similar to the PatchMatch image inpainting algorithm that our method is inspired by, it struggles with more advanced structural inpainting tasks, even though it is able to replicate textures relatively well.

The main problem that we identified with multi-view inpainting based on image inpainting from different viewpoints was that the inpaintings could be wildly different, resulting in blurry artefacts within the inpainted region. We aimed to avoid this problem by taking an entirely different approach to multi-view inpainting, operating directly on the 3D content of the scene. While our method succeeds in causing less blurry artefacts, it often introduces different artefacts, due to the inherent limitations of patch-based inpainting methods. Recent methods based on diffusion priors try to solve the problem in a different way by guiding the image inpaintings to be more consistent across viewpoints. Generally, these methods heavily reduce the blurry artefacts without introducing other types of artefacts and are therefore likely to be a more reliable alternative for multi-view inpainting.

Nevertheless, our work offers new insights into the direction of inpainting 3DGS scenes. For example, through our experiments we discovered the different challenges and opportunities of a voxel-based inpainting approach and a surface-based approach. These findings provide a deeper understanding of 3DGS editing in general.

References

Barcelos, Iany Macedo; Rabelo, Taís Bruno; Bernardini, Flávia; Monteiro, Rodrigo Salvador, and Fernandes, Leandro AF (2024). “From past to present: A tertiary investigation of twenty-four years of image inpainting”. In: *Computers & Graphics*, p. 104010.

Barnes, Connelly; Shechtman, Eli; Finkelstein, Adam, and Goldman, Dan B (2009). “PatchMatch: A randomized correspondence algorithm for structural image editing”. In: *ACM Trans. Graph.* 28.3, p. 24.

Barnes, Connelly; Shechtman, Eli; Goldman, Dan B, and Finkelstein, Adam (2010). “The generalized patchmatch correspondence algorithm”. In: *Computer Vision–ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part III 11*. Springer, pp. 29–43.

Barron, Jonathan T; Mildenhall, Ben; Verbin, Dor; Srinivasan, Pratul P, and Hedman, Peter (2022). “Mip-nerf 360: Unbounded anti-aliased neural radiance fields”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 5470–5479.

Bertalmio, Marcelo; Sapiro, Guillermo; Caselles, Vincent, and Ballester, Coloma (2000). “Image inpainting”. In: *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pp. 417–424.

Cai, Zhipeng; Wang, Cheng; Wen, Chenglu, and Li, Jonathan (2015). “3D-PatchMatch: An optimization algorithm for point cloud completion”. In: *2015 2nd IEEE International Conference on Spatial Data Mining and Geographical Knowledge Services (ICSDM)*. IEEE, pp. 157–161.

Chen, Honghua; Loy, Chen Change, and Pan, Xingang (2024). “MVIP-NeRF: Multi-view 3D Inpainting on NeRF Scenes via Diffusion Prior”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5344–5353.

Criminisi, Antonio; Pérez, Patrick, and Toyama, Kentaro (2004). “Region filling and object removal by exemplar-based image inpainting”. In: *IEEE Transactions on image processing* 13.9, pp. 1200–1212.

Ebert, Dylan (Sept. 2023). *Introduction to 3D Gaussian Splatting*. `https://huggingface.co/blog/gaussian-splatting`. (Accessed on 07/27/2024).

Efros, Alexei A and Leung, Thomas K (1999). “Texture synthesis by non-parametric sampling”. In: *Proceedings of the seventh IEEE international conference on computer vision*. Vol. 2. IEEE, pp. 1033–1038.

Guédon, Antoine and Lepetit, Vincent (2024). “Sugar: Surface-aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh rendering”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5354–5363.

Haque, Ayaan; Tancik, Matthew; Efros, Alexei A; Holynski, Aleksander, and Kanazawa, Angjoo (2023). “Instruct-nerf2nerf: Editing 3d scenes with instructions”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 19740–19750.

Ho, Jonathan; Jain, Ajay, and Abbeel, Pieter (2020). “Denoising diffusion probabilistic models”. In: *Advances in neural information processing systems* 33, pp. 6840–6851.

Hoppe, Hugues; DeRose, Tony; Duchamp, Tom; McDonald, John, and Stuetzle, Werner (1992). “Surface reconstruction from unorganized points”. In: *Proceedings of the 19th annual conference on computer graphics and interactive techniques*, pp. 71–78.

Huang, Hui; Wu, Shihao; Gong, Minglun; Cohen-Or, Daniel; Ascher, Uri, and Zhang, Hao (2013). “Edge-aware point set resampling”. In: *ACM transactions on graphics (TOG)* 32.1, pp. 1–12.

Huang, Jiajun and Yu, Hongchuan (2023). “Point’n Move: Interactive Scene Object Manipulation on Gaussian Splatting Radiance Fields”. In: *arXiv preprint arXiv:2311.16737*.

Jain, Viren and Seung, Sebastian (2008). “Natural image denoising with convolutional networks”. In: *Advances in neural information processing systems* 21.

Jam, Jireh; Kendrick, Connah; Walker, Kevin; Drouard, Vincent; Hsu, Jison Gee-Sern, and Yap, Moi Hoon (2021). “A comprehensive review of past and present image inpainting methods”. In: *Computer vision and image understanding* 203, p. 103147.

Kang, Seung Kwan; Shin, Seong A; Seo, Seongho; Byun, Min Soo; Lee, Dong Young; Kim, Yu Kyeong; Lee, Dong Soo, and Lee, Jae Sung (2021). “Deep learning-Based 3D inpainting of brain MR images”. In: *Scientific reports* 11.1, p. 1673.

Kazhdan, Michael; Bolitho, Matthew, and Hoppe, Hugues (2006). “Poisson surface reconstruction”. In: *Proceedings of the fourth Eurographics symposium on Geometry processing*. Vol. 7. 4.

Kerbl, Bernhard; Kopanas, Georgios; Leimkühler, Thomas, and Drettakis, George (2023). “3D Gaussian Splatting for Real-Time Radiance Field Rendering”. In: *ACM Trans. Graph.* 42.4, pp. 139–1.

Lee, Wen-Fu; Hsieh, Yuan-Ting, and Eryilmaz, Zubeyr Furkan (May 2018). *Interactive PatchMatch-Based Image Completion*. `https://wenfulee.github.io/CS-766-Computer-Vision/`. (Accessed on 07/25/2024).

Lin, Chieh Hubert; Kim, Changil; Huang, Jia-Bin; Li, Qinbo; Ma, Chih-Yao; Kopf, Johannes; Yang, Ming-Hsuan, and Tseng, Hung-Yu (2024). “Taming Latent Diffusion Model for Neural Radiance Field Inpainting”. In: *arXiv preprint arXiv:2404.09995*.

Liu, Hongyu; Jiang, Bin; Song, Yibing; Huang, Wei, and Yang, Chao (2020). “Rethinking image inpainting via a mutual encoder-decoder with feature equalizations”. In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*. Springer, pp. 725–741.

Liu, Zhiheng; Ouyang, Hao; Wang, Qiuyu; Cheng, Ka Leong; Xiao, Jie; Zhu, Kai; Xue, Nan; Liu, Yu; Shen, Yujun, and Cao, Yang (2024). “InFusion: Inpainting 3D Gaussians via Learning Depth Completion from Diffusion Prior”. In: *arXiv preprint arXiv:2404.11613*.

Longuet-Higgins, H Christopher (1981). “A computer algorithm for reconstructing a scene from two projections”. In: *Nature* 293.5828, pp. 133–135.

Lugmayr, Andreas; Danelljan, Martin; Romero, Andres; Yu, Fisher; Timofte, Radu, and Van Gool, Luc (2022). “Repaint: Inpainting using denoising diffusion probabilistic models”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11461–11471.

Ma, Xu; Qin, Can; You, Haoxuan; Ran, Haoxi, and Fu, Yun (2022). “Rethinking network design and local geometry in point cloud: A simple residual MLP framework”. In: *arXiv preprint arXiv:2202.07123*.

Mildenhall, Ben; Srinivasan, Pratul P; Tancik, Matthew; Barron, Jonathan T; Ramamoorthi, Ravi, and Ng, Ren (2021). “Nerf: Representing scenes as neural radiance fields for view synthesis”. In: *Communications of the ACM* 65.1, pp. 99–106.

Mirzaei, Ashkan; Aumentado-Armstrong, Tristan; Brubaker, Marcus A; Kelly, Jonathan; Levinshtein, Alex; Derpanis, Konstantinos G, and Gilitschenski, Igor (2023). “Reference-guided controllable inpainting of neural radiance fields”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 17815–17825.

Mirzaei, Ashkan; Aumentado-Armstrong, Tristan; Derpanis, Konstantinos G; Kelly, Jonathan; Brubaker, Marcus A; Gilitschenski, Igor, and Levinshtein, Alex (2023). “SPIn-NeRF: Multiview segmentation and perceptual inpainting with neural radiance fields”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 20669–20679.

Mirzaei, Ashkan; De Lutio, Riccardo; Kim, Seung Wook; Acuna, David; Kelly, Jonathan; Fidler, Sanja; Gilitschenski, Igor, and Gojcic, Zan (2024). “RefFusion: Reference Adapted Diffusion Models for 3D Scene Inpainting”. In: *arXiv preprint arXiv:2404.10765*.

Nazeri, Kamyar; Ng, Eric; Joseph, Tony; Qureshi, Faisal, and Ebrahimi, Mehran (2019). “Edgeconnect: Structure guided image inpainting using edge prediction”. In: *Proceedings of the IEEE/CVF international conference on computer vision workshops*.

Ogawa, Takahiro and Haseyama, Miki (2013). “Image inpainting based on sparse representations with a perceptual metric”. In: *EURASIP Journal on Advances in Signal Processing* 2013, pp. 1–26.

Pathak, Deepak; Krahenbuhl, Philipp; Donahue, Jeff; Darrell, Trevor, and Efros, Alexei A (2016). “Context encoders: Feature learning by inpainting”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2536–2544.

Pauly, Mark; Mitra, Niloy J; Giesen, Joachim; Gross, Markus H, and Guibas, Leonidas J (2005). “Example-based 3d scan completion”. In: *Symposium on geometry processing*, pp. 23–32.

Philippeau, Xavier (July 2010). *[Java] PatchMatch (Inpainting avec texture) - Contribuez*. https://www.developpez.net/forums/d947804/general-developpement/algorithme-mathematiques/contribuez/java-patchmatch-inpainting-texture/. (Accessed on 07/26/2024).

Prabhu, Kira; Wu, Jane; Tsai, Lynn; Hedman, Peter; Goldman, Dan B; Poole, Ben, and Broxton, Michael (2023). "Inpaint3D: 3D Scene Content Generation using 2D Inpainting Diffusion". In: *arXiv preprint arXiv:2312.03869*.

Qi, Charles R; Su, Hao; Mo, Kaichun, and Guibas, Leonidas J (2017). "Pointnet: Deep learning on point sets for 3d classification and segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 652–660.

Qin, Zhen; Zeng, Qingliang; Zong, Yixin, and Xu, Fan (2021). "Image inpainting based on deep learning: A review". In: *Displays* 69, p. 102028.

Quan, Weize; Chen, Jiaxi; Liu, Yanli; Yan, Dong-Ming, and Wonka, Peter (2024). "Deep learning-based image and video inpainting: A survey". In: *International Journal of Computer Vision* 132.7, pp. 2367–2400.

Ren, Yurui; Yu, Xiaoming; Zhang, Ruonan; Li, Thomas H; Liu, Shan, and Li, Ge (2019). "Structureflow: Image inpainting via structure-aware appearance flow". In: *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 181–190.

Richard, MMOBB and Chang, MYS (2001). "Fast digital image inpainting". In: *Appeared in the Proceedings of the International Conference on Visualization, Imaging and Image Processing (VIIP 2001), Marbella, Spain*, pp. 106–107.

Rombach, Robin; Blattmann, Andreas; Lorenz, Dominik; Esser, Patrick, and Ommer, Björn (2022). "High-resolution image synthesis with latent diffusion models". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695.

Schonberger, Johannes L and Frahm, Jan-Michael (2016). "Structure-from-motion revisited". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4104–4113.

Seitz, Steven M and Dyer, Charles R (1999). "Photorealistic scene reconstruction by voxel coloring". In: *International journal of computer vision* 35, pp. 151–173.

Sharma, Gaurav; Wu, Wencheng, and Dalal, Edul N (2005). "The CIEDE2000 color-difference formula: Implementation notes, supplementary test data, and mathematical observations". In: *Color Research & Application: Endorsed by Inter-Society Color Council, The Colour Group (Great Britain), Canadian Society for Color, Color Science Association of Japan, Dutch Society for the Study of Color, The Swedish Colour Centre Foundation, Colour Society of Australia, Centre Français de la Couleur* 30.1, pp. 21–30.

Sohl-Dickstein, Jascha; Weiss, Eric; Maheswaranathan, Niru, and Ganguli, Surya (2015). "Deep unsupervised learning using nonequilibrium thermodynamics". In: *International conference on machine learning*. PMLR, pp. 2256–2265.

Sridevi, G and Srinivas Kumar, S (2019). "Image inpainting based on fractional-order nonlinear diffusion for image reconstruction". In: *Circuits, Systems, and Signal Processing* 38, pp. 3802–3817.

Suvorov, Roman; Logacheva, Elizaveta; Mashikhin, Anton; Remizova, Anastasia; Ashukha, Arsenii; Silvestrov, Aleksei; Kong, Naejin; Goka, Harshith; Park, Kiwoong, and Lempitsky, Victor (2022). "Resolution-robust large mask inpainting with fourier convolutions". In: *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pp. 2149–2159.

Torrado-Carvajal, Angel; Albrecht, Daniel S; Lee, Jeungchan; Andronesi, Ovidiu C; Ratai, Eva-Maria; Napadow, Vitaly, and Loggia, Marco L (2021). "Inpainting as a technique for estimation of missing voxels in brain imaging". In: *Annals of biomedical engineering* 49, pp. 345–353.

Tschumperlé, David (2006). "Fast anisotropic smoothing of multi-valued images using curvature-preserving PDE's". In: *International Journal of Computer Vision* 68, pp. 65–82.

Wan, Ziyu; Zhang, Jingbo; Chen, Dongdong, and Liao, Jing (2021). "High-fidelity pluralistic image completion with transformers". In: *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 4692–4701.

Wang, Junjie; Fang, Jiemin; Zhang, Xiaopeng; Xie, Lingxi, and Tian, Qi (2024). "Gaussianeditor: Editing 3d gaussians delicately with text instructions". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 20902–20911.

Wang, Meng; Yu, Qinkang, and Liu, Haipeng (2024). "Three-Dimensional-Consistent Scene Inpainting via Uncertainty-Aware Neural Radiance Field". In: *Electronics* 13.2, p. 448.

Warburg, Frederik; Weber, Ethan; Tancik, Matthew; Holynski, Aleksander, and Kanazawa, Angjoo (2023). “Nerfbusters: Removing ghostly artifacts from casually captured nerfs”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 18120–18130.

Weber, Ethan; Holynski, Aleksander; Jampani, Varun; Saxena, Saurabh; Snavely, Noah; Kar, Abhishek, and Kanazawa, Angjoo (2024). “Nerfiller: Completing scenes via generative 3d inpainting”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 20731–20741.

Wei, Tzu-Ti; Kuo, Chin; Tseng, Yu-Chee, and Chen, Jen-Jee (2023). “Mpvf: 4d medical image inpainting by multi-pyramid voxel flows”. In: *IEEE Journal of Biomedical and Health Informatics*.

Wynn, Jamie and Turmukhambetov, Daniyar (2023). “Diffusionerf: Regularizing neural radiance fields with denoising diffusion models”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4180–4189.

Xiang, Hanyu; Zou, Qin; Nawaz, Muhammad Ali; Huang, Xianfeng; Zhang, Fan, and Yu, Hongkai (2023). “Deep learning for image inpainting: A survey”. In: *Pattern Recognition* 134, p. 109046.

Xiang, Peng; Wen, Xin; Liu, Yu-Shen; Cao, Yan-Pei; Wan, Pengfei; Zheng, Wen, and Han, Zhizhong (2021). “Snowflakenet: Point cloud completion by snowflake point deconvolution with skip-transformer”. In: *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 5499–5509.

Xie, Junyuan; Xu, Linli, and Chen, Enhong (2012). “Image denoising and inpainting with deep neural networks”. In: *Advances in neural information processing systems* 25.

Ye, Mingqiao; Danelljan, Martin; Yu, Fisher, and Ke, Lei (2023). “Gaussian grouping: Segment and edit anything in 3d scenes”. In: *arXiv preprint arXiv:2312.00732*.

Yin, Youtan; Fu, Zhoujie; Yang, Fan, and Lin, Guosheng (2023). “Or-nerf: Object removing from 3d scenes guided by multiview segmentation with neural radiance fields”. In: *arXiv preprint arXiv:2305.10503*.

Yu, Jiahui; Lin, Zhe; Yang, Jimei; Shen, Xiaohui; Lu, Xin, and Huang, Thomas S (2018). “Generative image inpainting with contextual attention”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5505–5514.

— (2019). “Free-form image inpainting with gated convolution”. In: *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 4471–4480.

Yu, Xumin; Rao, Yongming; Wang, Ziyi; Liu, Zuyan; Lu, Jiwen, and Zhou, Jie (2021). “Pointr: Diverse point cloud completion with geometry-aware transformers”. In: *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 12498–12507.

Yu, Xumin; Tang, Lulu; Rao, Yongming; Huang, Tiejun; Zhou, Jie, and Lu, Jiwen (2022). “Point-bert: Pre-training 3d point cloud transformers with masked point modeling”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 19313–19322.

Yu, Yingchen; Zhan, Fangneng; Wu, Rongliang; Pan, Jianxiong; Cui, Kaiwen; Lu, Shijian; Ma, Feiying; Xie, Xuansong, and Miao, Chunyan (2021). “Diverse image inpainting with bidirectional and autoregressive transformers”. In: *Proceedings of the 29th ACM International Conference on Multimedia*, pp. 69–78.

Yuan, Wentao; Khot, Tejas; Held, David; Mertz, Christoph, and Hebert, Martial (2018). “Pcn: Point completion network”. In: *2018 international conference on 3D vision (3DV)*. IEEE, pp. 728–737.

Yuksel, Cem (2015). “Sample elimination for generating poisson disk sample sets”. In: *Computer Graphics Forum*. Vol. 34. 2. Wiley Online Library, pp. 25–32.

Zhang, Richard; Isola, Phillip; Efros, Alexei A; Shechtman, Eli, and Wang, Oliver (2018). “The unreasonable effectiveness of deep features as a perceptual metric”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 586–595.

Zheng, Chuanxia; Cham, Tat-Jen; Cai, Jianfei, and Phung, Dinh (2022). “Bridging global context interactions for high-fidelity image completion”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11512–11522.