IMPERIAL

Moving Objects in 3D Point Clouds

MEng Final Year Project Report

Author

Anlan Qiu

CID: 02033960

Supervised by

Prof. Krystian Mikolajczyk

A Thesis submitted in fulfillment of requirements for the degree of
Master of Science in Electronic and Information Engineering

Department of Electrical and Electronic Engineering
Imperial College London

2025

Abstract

Abstract

Currently, there exists no 3D editing tool that allows user-controlled relocation of objects within a 3D point cloud to new positions. This report introduces a pipeline that allows object repositioning in 3D point clouds by first removing the object from its current location, inpainting the region exposed by the removed object, and then inserting the removed object at a new location in the point cloud. Furthermore, the pipeline supports the operations of only removing the object, or replacing the removed object with a different object. To set up evaluation metrics for these operations, a testing procedure is also developed which allows the quality of object repositioning, object removal and object replacement to be compared to a ground-truth measure.

Declaration of Originality

I hereby declare that the work presented in this thesis is my own unless otherwise stated. To the best of my knowledge the work is original and ideas developed in collaboration with others have been appropriately referenced.

I acknowledge the use of ChatGPT o4-mini-high and GPT-4o to obtain information on specific topics, improve the quality of my English and to generate images for this report.

Copyright Declaration

Acknowledgments

I would first like to express my deepest gratitude to my parents. My mother's unwavering support and encouragement has been a constant source of strength, and my father's wise advice and guidance have shaped both my academic and personal growth.

I am profoundly grateful to Professor Krystian Mikolajczyk, PhD candidates Ye Mao and Junpeng Jing, and the entire Matchlab team for welcoming me into the world of research. Over the past year, they have provided me with invaluable opportunities to explore the fields of Machine Learning and Computer Vision, challenged me with an engaging research topic, and supported me as I dove deeper into my studies than ever before. Their mentorship has not only enhanced my technical skills but also nurtured my development as a thoughtful and resourceful engineer.

Finally, I extend my sincere thanks to the Imperial community—my friends, the dedicated staff members who assisted me whenever I needed, and the many new experiences that have enriched my journey. Each encounter and opportunity has helped me grow both personally and professionally, and I am truly appreciative of the supportive environment that has surrounded me.

Contents

List of Acronyms

List of Figures

1 Introduction

Contents

1.1 Point Clouds

Point clouds serve as the primary data representation for 3D scenes—and are the focus of this project's manipulations. In this subsection, we will explain what a point cloud is and explore its key applications.

Point clouds are collections of discrete points in three-dimensional space that together form a digital sampling of real-world surfaces and structures, as illustrated in Fig. 1.1. Each point carries 3D spatial coordinates (and often color or intensity values), enabling the reconstruction of complex 3D scenes without the need for continuous mesh geometry.

1

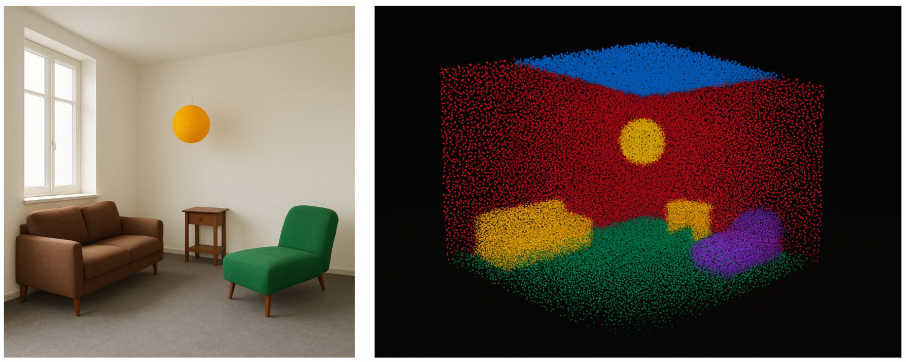

Figure 1.1: An illustration of how a real scene (left image) can be represented by a point cloud (right image).

Generating Point Clouds Point clouds can be acquired through a variety of techniques. LiDAR systems [1] use active laser scanning to sample millions of distance measurements directly from surfaces. Stereo vision [2] employs two or more cameras to capture a scene from slightly different angles and derive depth by matching image features. Structure-from-Motion (SfM) [3] takes this further by processing large sets of overlapping photographs—often gathered with a handheld camera or drone—and automatically estimating camera positions and the 3D coordinates of key points. In this way, SfM treats any collection of multi-view images as input for generating a detailed point cloud.

Alternative Format Instead of just a cloud of separate points in space, a mesh connects those points to form a complete surface. Meshes are useful because they look like real objects on a screen—you can paint colors on them, make them move naturally, or use them in simulations. To make a mesh from a point cloud, points are linked with its neighbors in a careful pattern, so that all the gaps are filled in and you end up with a solid model that can be used in games, movies, or design work. A mesh can be seen in Fig. 1.2.

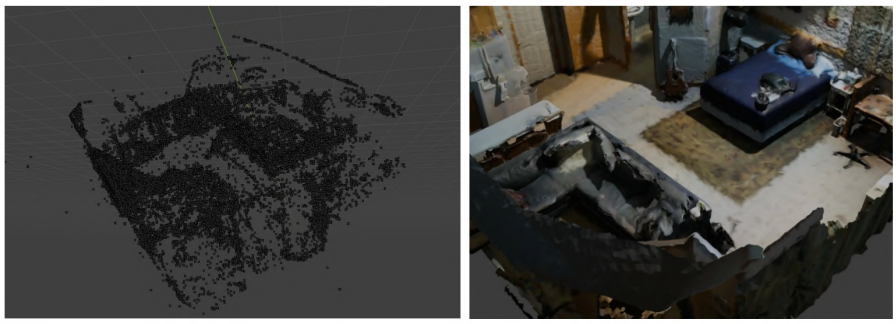

Figure 1.2: The left image shows a point cloud, which can ne turned into the mesh shown on the right through estimating surface normals. The scene is taken from the ScanNet [4] dataset

Applications Point clouds serve as the foundational data for many applications: in surveying and mapping, they enable accurate terrain and infrastructure models; in autonomous vehicles, they provide spatial awareness for navigation and obstacle avoidance; in architecture and construction, they facilitate as-built documentation and renovation planning; and in entertainment and virtual reality, they underpin realistic environment reconstruction and motion capture. Because point clouds are inherently flexible and scalable, they are also widely used in robotics, cultural heritage preservation, and quality inspection—wherever precise three-dimensional measurements are essential.

Scene Editing Because point clouds faithfully encode the shape and appearance of objects, labeling each point into different object classes, or segmenting the point cloud, as shown in the right image in Fig. 1.1, unlocks powerful editing capabilities: objects can be removed, and even inserted into a new location, making point clouds not only powerful for scene representation and simulation, but also a tool for 3D scene manipulation and augmentation.

1.2 Motivation

Fig. 1.3 shows three different ways of editing objects within a scene. Scene editing, and specifically moving objects in a 3D scene is valuable for several key reasons:

Applications Moving objects within a point cloud offers many practical applications: it lets you generate new obstacle layouts for robotics simulations, create fresh environments in video games, and visualize alternative furniture or fixture arrangements in architecture and interior design. Without a dedicated digital pipeline for object repositioning, making such edits would require physically moving the object in the real world and then re-running the entire point-cloud capture process—an approach that is not only time-consuming but often infeasible if you no longer have access to the original capture site. This underscores how a digital editing pipeline can save significant time and effort. Just as it's commonplace to remove or adjust unwanted elements in a 2D photo, we aim to build an analogous tool that makes object-level edits in 3D point clouds equally straightforward.

A Lack of Existing 3D Datasets In modern computer vision—whether tackling 2D tasks like image classification, semantic segmentation, or generative modeling, or extending these tasks

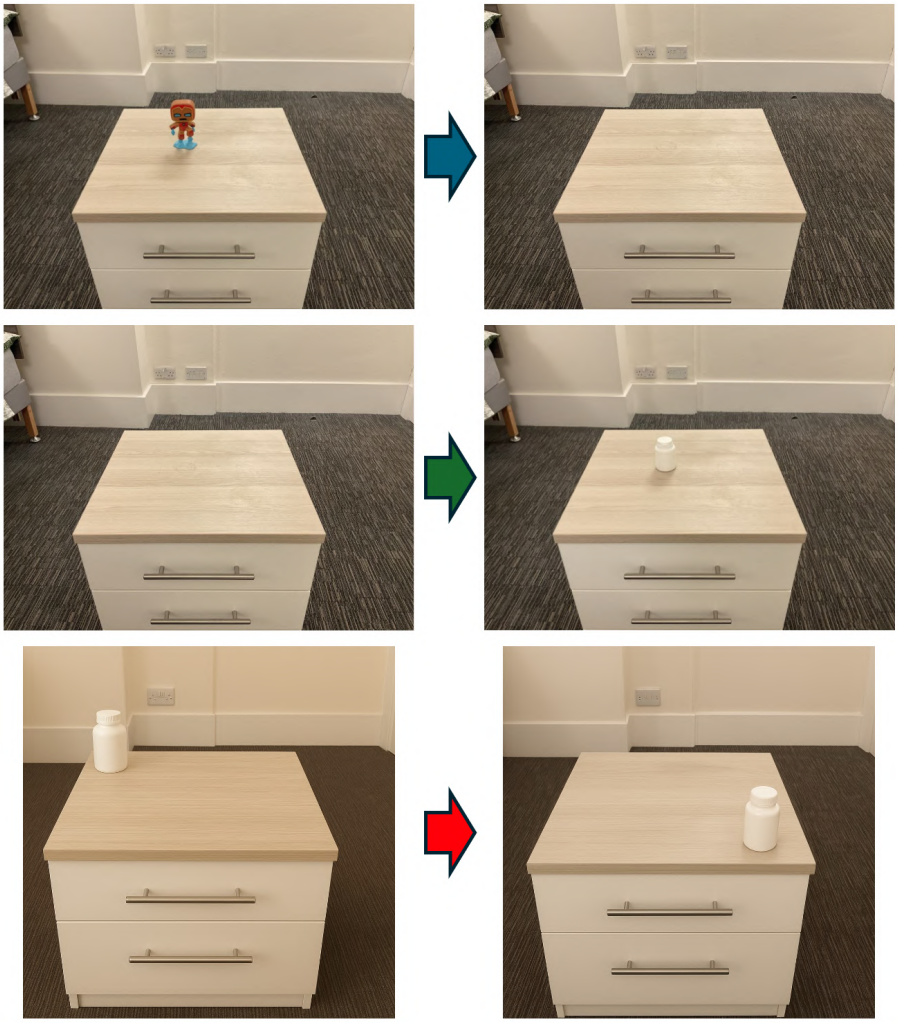

Figure 1.3: Examples of the basic scene editing operations. The top row shows an object being removed from a scene, and the bottom row shows an object being inserted intio a scene. Finally, the bottom row shows object repositioning. This is a composition of the first two operations: removing the object in the first location and adding it back into the second location.

into 3D—state-of-the-art approaches almost invariably rely on machine learning, which in turn demands vast amounts of training data. In the 2D realm, large-scale datasets such as ImageNet [5] and COCO [6] have fueled breakthroughs like deep residual networks for classification [7], Mask R-CNN for instance segmentation [8], and GAN-based methods for image synthesis [9]. By contrast, acquiring and annotating 3D point clouds remains significantly more challenging, so widely used datasets like ScanNet [4] and Matterport3D [10] are orders of magnitude smaller than their 2D counterparts. An object-repositioning tool effectively serves as a 3D data augmentation tool—it generates new, diverse point-cloud scenes without recapturing physical environments, thereby supplying additional training examples that can accelerate the development of next-generation 3D algorithms.

Hypothetical Reasoning Hypo3D [11] defines 3D hypothetical reasoning as the task where a 3D vision-language model (VLM) [12] is given an initial point cloud—say, of a bedroom—along with a described change, such as "remove the lamp from the desk." The model must then answer questions (e.g., "How many utility items remain in the room?") by combining knowledge of the original scene with the hypothetical alteration. Currently, VLMs perform well below human level on this task because they must mentally simulate scene changes rather than observe them directly. By using our object-repositioning pipeline to generate a new point cloud that reflects the described modification, the VLM can reason over an explicit, updated 3D representation instead of attempting to imagine the change—an approach that should significantly boost hypothetical reasoning accuracy.

1.3 Challenges

In Motivation, we have explained that object repositioning is the composition of object removal and object insertion. However, after an object has been removed from a point cloud, it exposes a region that used to be below the object, which could have an irregular texture and geometry. This presents the challenge of inpainting this region: making the area consistent in geometry and texture with the rest of the scene. Hence, there are now three components to object repositioning: object removal, inpainting and object insertion, which all come with their own challenges.

1.3.1 Object Removal

Removing an object from a 3D point cloud hinges on accurately segmenting each point by object. Once segmentation is reliable, removal is just deleting the relevant points. In practice, however, segmentation is challenging for several reasons:

Irregular, unstructured data Point clouds are unordered (x, y, z) samples without a uniform grid. Unlike 2D images—where a U-Net [13] can slide a convolutional kernel—3D methods (PointNet [14], PointNet++ [15], graph networks [16]) must explicitly construct local neighborhoods or learn to aggregate irregular points, adding design and computational complexity.

Variable density and occlusions Real-world scans often have uneven sampling—walls may be dense while furniture is sparse—and occluded or missing regions where the sensor couldn't see. A

model must handle all these patterns, requiring diverse training data to cover every scenario.

noise and registration artifacts Depth sensors introduce measurement noise, and multi-view scans need precise alignment. The result is jitter, outliers, and stitching artifacts. Segmentation models must tolerate this messiness, which further increases the need for well-curated, annotated examples.

Limited, costly annotations Labeling every point in a dense scan (millions of points) is far more labor-intensive than annotating 2D images. As a result, 3D datasets remain much smaller and less diverse than their 2D counterparts. Even semi-automatic tools can't fully close this gap, and inconsistent labeling (due to noisy or incomplete surfaces) further complicates learning.

These factors—irregular representation, uneven sampling, noise, occlusions, and high annotation overhead—mean that without a robust, data-efficient segmentation pipeline, object removal will produce artifacts: leftover "ghost" points or accidental deletion of neighboring geometry.

1.3.2 Inpainting

In 3D, point clouds lack the regular grid structure of images, so when an object is removed, the surrounding points may be unevenly distributed or missing entirely. Inferring a smooth, plausible surface requires estimating local geometry—normals, curvature, and neighborhood relationships—from sparse, irregular samples. Without an explicit mesh, even small errors in these estimates can create visible artifacts like bumps or gaps.

Maintaining semantic consistency is also crucial. If you remove a chair from a room, the inpainting algorithm must recognize the shape and context (e.g., floor plane, nearby walls) to reconstruct the missing volume as an empty floor or extend adjacent surfaces appropriately. Failing to understand scene layout can lead to nonsensical geometry—such as floating fragments or misaligned planes—that breaks realism.

When point clouds include color (RGB), deleted regions leave no direct color cues. Algorithms must predict plausible textures and lighting for newly generated points, blending smoothly with surrounding colors. Sensor noise and varying illumination in real scans make accurate color inference even more challenging, as mismatches are easily noticeable in the final render.

Finally, scalability and robustness are significant concerns. Real-world scans often contain millions of points with unpredictable noise and occlusions. An inpainting method that relies on heavy graph computations or dense surface fitting can become prohibitively slow, especially when processing large indoor or outdoor scenes. Balancing computational efficiency with high-quality reconstruction remains an open problem in 3D hole filling.

1.3.3 Object Insertion

Even with perfect segmentation, transplanting an object—whether inserting a new model or moving an existing one—into a point cloud poses nontrivial challenges. First, the added or relocated object must align correctly within the scene's geometry. Determining the proper pose (translation, rotation, and scale) requires understanding the local surface normals and underlying floor or table plane. Small misalignments can lead to "floating" or "sunk" appearances, where the object either hovers above the surface or intersects it, breaking physical plausibility.

Next, collision avoidance and scene integration are essential. A moved sofa might collide with nearby chairs or walls if its new position isn't checked against existing points. Since point clouds lack an explicit mesh, collision detection often depends on approximating object and scene boundaries from sparse samples. Without a watertight representation, even simple overlap checks can miss subtle intersections, resulting in interpenetrating geometry that looks unnatural when rendered or meshed.

Maintaining semantic and contextual consistency is also critical. Inserting a lamp on a table demands recognition that the tabletop is a flat, elevated surface, not part of the floor or a decorative rug. If the algorithm misinterprets the scene layout—confusing a low shelf for a floor—it could place the lamp where it makes no sense. Effective repositioning therefore relies on high-level scene understanding to ensure objects occupy semantically valid locations (e.g., chairs near tables, artwork on walls).

When objects come with color, the color blending and density adaptation stage becomes tricky. Point density and color distribution around the insertion point often differ from those of the object. Naively pasting the object's points can create abrupt density transitions or visible seams in color. To avoid this, algorithms must either downsample or upsample nearby points and adjust per-point color attributes, which is complicated by sensor noise and varying illumination. Any mismatch in point density or shading will be obvious once the scene is visualized.

1.4 Scope

Repositioning objects within a 3D point cloud can be interpreted in many ways, since point clouds come in different formats, represent a vast array of possible scenes, and contain countless object categories. To narrow the scope of this project, we make the following clarifications:

1.4.1 Types of input

We will be using SfM as the input to our pipeline. Images will be taken from two types of scenes: Real world scenes, and multi-view renders of meshes.

1.4.2 Type of scenes

We support both indoor and outdoor environments. Moreover, scenes may originate from real-world image captures (e.g., photographs of a living room or a street corner) or from multi-view renders of meshes that were themselves reconstructed from point-cloud datasets. In all cases, the goal is a semantically coherent, photorealistic 3D scene.

1.4.3 Definition of "objects"

When referring to "objects" within a scene, we mean smaller, movable entities—furniture, lamps, vehicles, or other standalone items. Structural elements (walls, floors, ceilings) and large terrain features (ground planes, building shells) are not treated as movable objects in this work.

By restricting our inputs and the kinds of objects to be manipulated, we ensure a consistent testing environment and address point-cloud editing methods on data that can reliably be meshed, visualized, and semantically parsed.

1.5 Project Objectives

1. Develop modules that allow object removal, inpainting and object insertion on a point cloud.

2. Combine these modules to form a point cloud editing tool that allows for object repositioning.

3. Theoretically, qualitatively and quantitatively evaluate current tools that perform object removal, inpainting and object insertion on point clouds.

Contribution Currently, there doesn't exist a tool that performs object repositioning. If the first two objectives of this project are met, the developed pipeline will be a novel method. Even if the first two objectives are not fully met, the third objective, evaluation, will allow an understanding of the challenges preventing the full piepline to be achieved, and the current state of the tools that aim to achieve the pipeline.

1.6 Report Structure

The purpose of each of the subsequent sections, and how they contribute to the project objectives, will now be discussed:

1. Background: this section provides the existing tools which can be utilised to build the final pipeline.
2. Design and Analysis: this section provides the skeleton and architecture of the pipeline, and also begins the evaluation objective: a theoretical comparison of some of the existing tools introduced in the Background section.
3. Implementation: this section goes into detail on the pipeline architecture proposed in Design and Analysis, providing detailed guidance on how to build the pipeline.
4. Evaluation: this is the main section for achieving the third objective of evaluation. Each of the modules of the architecture introduced in Design and Analysis is qualitatively and quantitatively evaluated.
5. Conclusion: to what extent the project objectives have been achieved is discussed. Based on the work of this report, interesting topics for future works are identified.

1

2 Background

Contents

As mentioned in Chapter 1, moving an object in a 3D point cloud involves removing the object from its original location, inpainting the area left underneath the object, and adding the object in a new desired location. There already exist methods that handle parts of this pipeline. This project involves composing these different methods together into a complete object moving pipeline, as well as incorporating other techniques/projects that enhance the performance of the original projects. Hence, the background covered will be quite broad.

2.1 Generating Point Clouds

Generating accurate 3D point clouds is a foundational step for many computer vision and robotics applications, providing a detailed spatial representation of real-world scenes. One of the most common approaches is multi-view photogrammetry, or Structure-from-Motion (SfM). In SfM, a set of overlapping RGB images—captured from different viewpoints around a scene—are processed to detect and match feature points (e.g., using SIFT [17] or SURF [18]), as shown in Fig. 2.1.

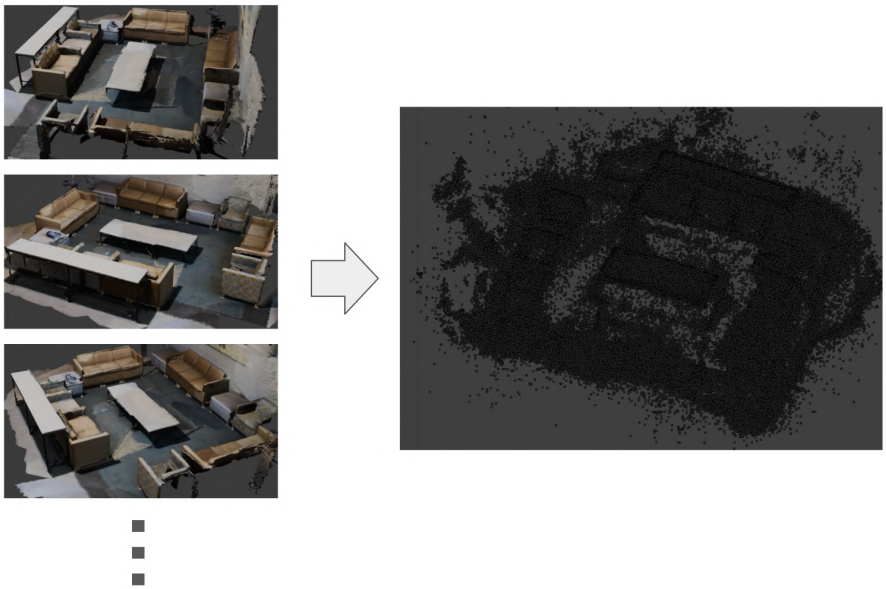

Figure 2.1: Multi-view images taken in a scene, and fed into SfM pipelines such as Colmap to eventually form point clouds. The scene is taken from the ScanNet [4] dataset

After establishing correspondences, a bundle-adjustment routine [19] jointly refines camera poses and triangulates an initial sparse point set. A subsequent Multi-View Stereo (MVS) [20] stage then densifies the reconstruction by estimating depth for each pixel via pixel-wise matching and fusing multiple views (often implemented end-to-end in open-source systems like COLMAP [21]). The result is a dense, colorized point cloud that can capture fine textures and geometric detail, provided that the input images offer sufficient coverage, consistent lighting, and abundant surface features.

Stereo vision is another widely used technique, particularly when real-time or near-real-time performance is required. In a calibrated stereo rig—with two cameras separated by a known baseline—depth is recovered by computing disparities along corresponding epipolar lines. Algorithms such as block matching or more advanced semi-global matching [22] (and increasingly deep-learning–based methods) yield a dense disparity map, which is then reprojected into 3D coordinates per pixel. Stereo systems can operate at video frame rates, making them suitable for dynamic scenarios (e.g., mobile robotics or driver-assistance systems). However, depth accuracy decreases with distance (as disparity diminishes), and regions lacking texture, or containing repetitive patterns, often produce noisy or incomplete reconstructions.

LiDAR (Light Detection and Ranging) offers a more direct approach: by emitting laser pulses and measuring their time of flight (or phase shift), LiDAR sensors can generate accurate, high-density point clouds, often with per-point intensity or reflectance values. Scanners range from station-

ary, tripod-mounted units—ideal for architectural surveys—to mobile or vehicle-mounted systems used in autonomous driving. Because LiDAR is less dependent on ambient lighting or surface texture, it excels at capturing large outdoor environments or indoor spaces with minimal visual features. That said, raw LiDAR data typically require significant preprocessing to remove spurious points, calibrate multiple sweeps, and mitigate multipath artifacts. Tools like dustr [23] —a library designed for denoising and outlier removal in point-cloud datasets—play a crucial role in cleaning LiDAR scans before downstream tasks such as segmentation or mesh reconstruction.

Beyond these primary methods, several specialized techniques offer unique advantages for particular use cases. Structured light scanners [24] project coded patterns (stripes or grids) onto an object and analyze pattern deformation to infer depth, achieving submillimeter accuracy for small-object digitization in controlled, indoor settings. Time-of-Flight (ToF) cameras [25], which modulate infrared light and measure phase delays, can capture depth at video rates, though their accuracy drops off beyond a few meters, and they are susceptible to multipath interference. Emerging hybrid approaches—such as RGB-D SLAM frameworks [26]—combine sequential RGB-D frames to build consistent, large-scale reconstructions in real time, while photometric stereo can estimate fine surface normals under varying illumination before integrating normals into a point cloud. Each method involves trade-offs in cost, resolution, speed, and environmental constraints, but by selecting—and often combining—the right techniques, practitioners can generate the dense, high-fidelity point clouds necessary for tasks ranging from virtual reality to autonomous navigation.

2.2 Rendering Point Clouds

In recent years, innovative rendering techniques such as Neural Radiance Fields (NeRF) [27] and Gaussian Splatting [28] have emerged, revolutionizing the way 3D scenes are represented and rendered. NeRF uses a neural network to model the volumetric density and color of a scene, enabling highly photorealistic rendering from sparse input views. However, NeRF is computationally intensive, requiring significant time and resources for training and rendering. Gaussian Splatting, on the other hand, offers a faster alternative by representing scenes with a set of 3D Gaussian distributions, which can be efficiently optimized and rendered. This approach not only reduces computational overhead but also provides high-quality results comparable to or exceeding NeRF, particularly in terms of rendering speed.

Both techniques enable differentiable rendering, a technique that allows gradients to be back-

propagated through the rendering process. This capability facilitates tasks such as 3D reconstruction, identity encoding, and inpainting by integrating rendering directly into the optimization pipeline. Differentiable rendering, powered by these advancements, has expanded the applications of 3D vision and graphics in areas like augmented reality, robotics, and content creation, marking a significant step forward in bridging the gap between visual realism and computational efficiency.

The two projects that are most similar to the objective of this thesis are Gaussian Grouping [29] and Gaussian Editor [30], both of which utilize Gaussian Splatting for their 3D scene representation and rendering techniques. Both projects employ similar segmentation and inpainting pipelines, but Gaussian Grouping offers a more accessible codebase, so it will be more focused on in this report.

2.3 3D Segmentation

In 3D point-cloud segmentation, early geometry-based approaches often relied on local surface properties: region growing methods expand clusters from seed points based on normal and curvature consistency, while graph-based clustering builds a connectivity graph over points and partitions it via spectral or minimum-cut algorithms. These techniques excel in clean, uniformly sampled scans but can falter when facing noise, holes, or varying point densities.

With the rise of deep learning, purely 3D networks such as PointNet++, KPConv [31], and RandLA-Net [32] have achieved impressive results by learning hierarchical or kernelized features directly on raw coordinates. PointNet++ captures local context through nested set abstraction, KPConv defines deformable convolutional kernels in 3D space, and RandLA-Net emphasizes efficient, random sampling for large scenes. Yet all must grapple with the irregular, unstructured nature of point clouds and depend heavily on extensive, high-quality 3D annotations—which are costly to acquire.

Gaussian grouping offers a hybrid alternative: the point cloud is rendered into multiple 2D views, each segmented by state-of-the-art image networks, then points are clustered in 3D according to a Gaussian-weighted consensus of those 2D labels. By leveraging the robustness and rich context of 2D vision models—such as precise edge detection and texture cues—this method naturally handles noise and uneven sampling without the cubic memory overhead of volumetric grids. As a result, Gaussian grouping provides semantically coherent, photorealistic segmentations that align closely with both geometry and appearance, making it an attractive complement to purely 3D techniques.

2.4 Inpainting Methods

Figure 2.2: The process of inpainting is illustrated above. An inpainting mask is supplied to select an area of the original image to be modified. The modification is to be predicted using the un-inpainted region of the image.

The process of 2D inpainting can be explained in Fig. 2.2, where an undesirable region is modified to a plausible alternative. The same concept applies in 3D, where an undesirable region of a point cloud must be modified.

In 3D point-cloud inpainting, purely volumetric approaches first convert the incomplete data into a voxel grid and then apply 3D convolutional networks—such as 3D-Unet [13] variants or generative adversarial models like 3D-RecGAN [33]—to predict occupancy or signed-distance values in missing cells. Geometry-driven methods, on the other hand, interpolate local surface patches directly on the mesh or point cloud, using Poisson reconstruction [34] or moving least-squares [35] to smoothly bridge voids. More recent point-based networks (e.g., PCN [36], TopNet [37], SnowflakeNet [38], PMPNet [39]) operate on raw coordinates, learning an end-to-end mapping from sparse, partial inputs to dense, completed outputs by encoding global shape priors and decoding fine-scale detail. Implicit-field techniques (Occupancy Networks [40], DeepSDF [41]) represent shapes as continuous functions that can be queried at arbitrary resolution, offering high fidelity at the cost of per-scene optimization and complex mesh extraction. Although these methods can produce plausible completions, they often struggle with memory constraints (in voxels), unstructured input (in points), and limited 3D training data.

By contrast, 2D-assisted inpainting leverages mature image-based models to handle large missing regions with photorealistic detail. The incomplete point cloud is rendered into multiple views—depth, normals, or RGB—and passed through state-of-the-art image inpainting networks (e.g., partial-conv [42], EdgeConnect [43], LaMa [44]) that excel at synthesizing textures, edges, and shading at high resolution without cubic memory growth. These filled-in views are then projected back onto the 3D data and fused—using depth consistency or confidence weighting—to guide or supervise a final 3D completion step. Because each inpainted image is by definition a realistic rendering of the scene, this hybrid pipeline guarantees natural texture and lighting cues, enriches the 3D model with rich, view-consistent supervision, and circumvents many of the scalability and resolution challenges inherent to purely 3D methods.

2.5 Diffusion Models

Diffusion models have quickly become a leading approach in generative modeling, thanks to their simple two-stage process of incrementally corrupting data with noise and training a neural network to reverse that corruption and recover realistic samples. These methods were popularized by [45], which demonstrated that a U-Net denoiser could generate high-fidelity images. Building on this momentum, OpenAI's DALL · E [46] adopted a diffusion-based decoder conditioned on CLIP embeddings [47], achieving more coherent and diverse outputs than its autoregressive predecessor. Beyond text or label conditioning, ControlNet enhances pretrained diffusion models with spatial controls—such as edge maps, depth cues, and human pose inputs—by integrating additional trainable modules, thus enabling precise structural guidance in image synthesis. Unlike adversarial methods, diffusion models optimize a clear likelihood-based objective, leading to stable training dynamics and a rich diversity of samples. The latest breakthrough, Flux.1 [48], represents the current state-of-the-art in in-context text-to-image generation and editing, allowing seamless iterative modifications of images guided by both text and visual inputs. Flux.1 also offers an inpainting model. Recent innovations in latent diffusion and accelerated sampling have further broadened their applicability to domains such as speech synthesis, molecular generation, and video production, cementing diffusion frameworks as versatile tools for future AI-driven creativity.

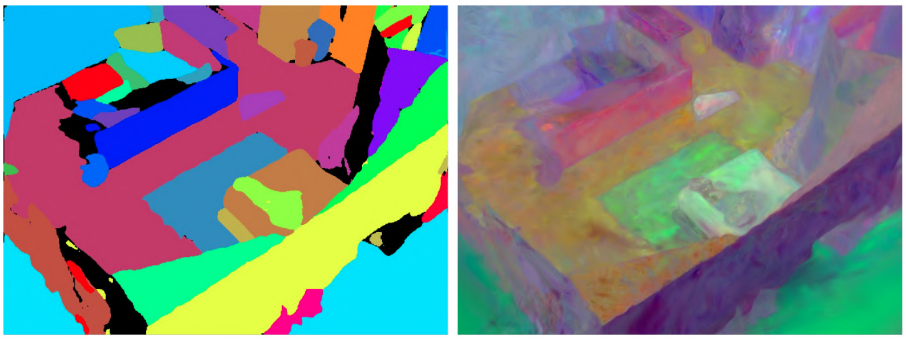

Figure 2.3: Segmentation images from Gaussian Grouping: the left image shows segmentation in 2D, whereas the right image is a visualisation of 3D segmentation, or identity encodings. The 2D segmentations are used to train the 3D segmentations. The 3D point segmentation is not a simple label. Each point is given a probability for each of the possible classes. The right image is a PCA representation of this.

2.6 Gaussian Grouping

The process Gaussian Grouping uses to segment objects in a 3D point cloud will be described. First, a point cloud and an RGB frame sequence of the corresponding scene are taken as input. 2D segmentation using [49] is performed on the frames to identify objects in each view, and an object tracking model [50] is applied to ensure consistent mask IDs across the segmented 2D frames, resulting in coherent multi-view segmentation. These segmented frames are then used as supervision to generate a Gaussian splat representation of the point cloud, where the scene is represented as a set of Gaussian distributions. Identity encodings, as show in the right image of Fig. 2.3, are added to each Gaussian to retain object-specific information. Finally, differential rendering is performed, leveraging the multi-view segmented frames to refine the Gaussian splat representation. This process produces a 3D segmentation of the Gaussian splat, effectively segmenting objects in the original 3D point cloud.

Once 3D segmentation is achieved, object removal becomes a straightforward process. To remove an object, all Gaussians with the identity encoding corresponding to that object are simply eliminated from the representation.

After object removal, a dark hole will be left in the area that used to be covered by the removed object. The process Gaussian Grouping [29] uses to inpaint the hole left by object removal will be described. After the object removal step, renders of the Gaussian splat with the object removed are generated from the original camera positions. These 2D renders are then inpainted using a 2D inpainting method [51], which requires the construction of 2D masks. The masks are generated using an object detection model [52], and associated with a tracking model [50], ensuring

consistent identification of the removed object's hole (inpainting mask) across views. Once the 2D inpainted frames are obtained, they serve as supervision for the 3D inpainting process. Differential rendering is employed to propagate the 2D inpainting results into the Gaussian splat representation, completing the reconstruction of the removed areas in 3D space. A problem with this pipeline is that using [50] and [52] for object-tracking often fails in scenes that have any sort of complexity.

2.7 Object Insertion

2.7.1 SIGNeRF

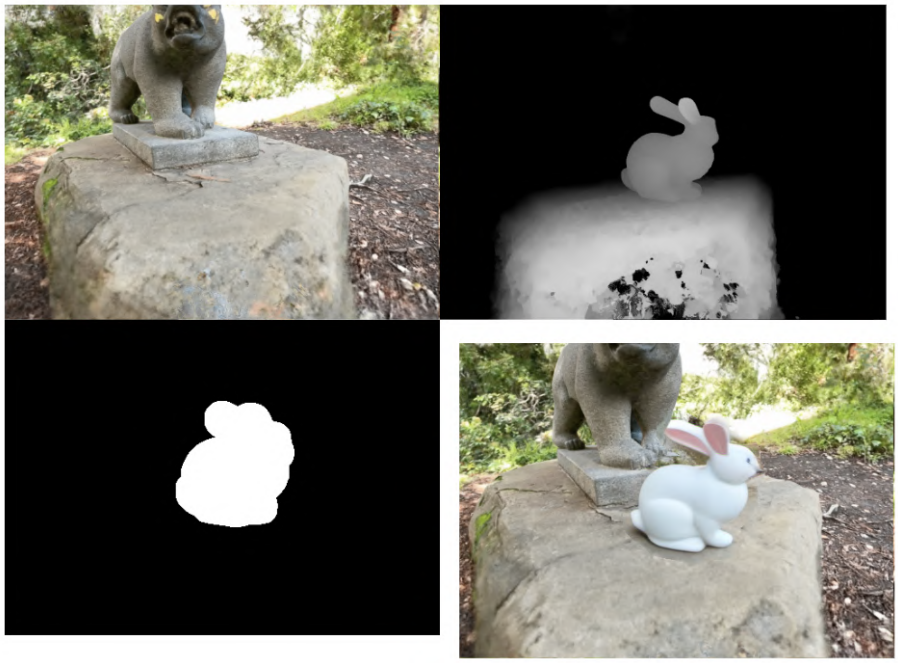

Figure 2.4: ControlNet allows further conditioning of the inpainting on one extra piece of information, such as a segmentation map or depth map. The above images show inpaining conditioned on a depth map using ControlNet. The scene is taken from the dataset introduced in [53].

SIGNeRF [54] achieves object insertion by leveraging a pre-trained NeRF representation of the original scene in conjunction with a depth-conditioned image diffusion model [45] (ControlNet [55]) to generate multi-view edits, which are then used to refine the NeRF. The key insight is that requesting a grid of views from ControlNet, conditioned on depth maps, inherently enforces 3D consistency across the inserted object's appearance. First, a proxy mesh of the new object (e.g., a CAD model or simplified geometry) is placed at the desired location within the scene. This can be done through a method such as Instant-nsr-pl [56].Camera poses are chosen around this proxy to form a small "reference sheet"—typically five to seven views—each accompanied by its

depth map extracted from the original NeRF. ControlNet then inpaints these reference images, as explained in Fig. 2.4, inserting the object with correct occlusion boundaries and lighting, producing a multi-view grid that specifies how the scene should look with the new object.

Next, SIGNeRF uses this edited reference sheet to update the entire original image set. Rather than processing each view independently, the method projects the edited reference images into pixel-aligned space for all other camera poses, creating an "edited image collection" that remains coherent with the reference sheet's geometry. This step ensures that every original viewpoint is adjusted consistently, avoiding mismatches that would otherwise arise if individual views were edited in isolation. By applying the same depth-conditioned diffusion process to each view—guided by both the reference sheet and the proxy mesh—SIGNeRF generates a full set of edited training images that reflect the object's insertion from all angles.

Finally, the original NeRF is fine-tuned using this new image collection in a single optimization pass. Because the edited views are already 3D-consistent, SIGNeRF can update the radiance and density fields to incorporate the new object without requiring iterative back-and-forth between image- and 3D-space. In cases where minor inconsistencies remain, an optional second iteration—where a fresh reference sheet is generated from the updated NeRF—can further refine the results. The outcome is a NeRF that renders the scene with the inserted object seamlessly integrated: correct occlusions, shadows, and lighting, all without manual mesh editing inside NeRF.

2

3

Design and Analysis

Contents

3.1 Architectures

With the project split into three sections: object removal, inpainting and object insertion, the pipeline of each of these stages will be explained in detail. Each pipeline explanation will include a series of different files, representing the data we have at each stage, and the tools used to transition from one set of data to another. The files will be represented with boxes, and the transitions with tools will be represented with arrows.

Fig. 3.1 shows the pipeline for the first stage of the project: object removal. The input to this pipeline, which is also the input for the entire project, is a sequence of multi-view images representing a scene. Colmap is then used to take these images and to produce a point cloud representation of the scene. At the same time, SAM and DEVA are used to perform multi-view consistent object segmentation on the multi-view image sequence. Next, both the point cloud and the segmentations are fed into the Gaussian Grouping training script. Here, the multi-view images

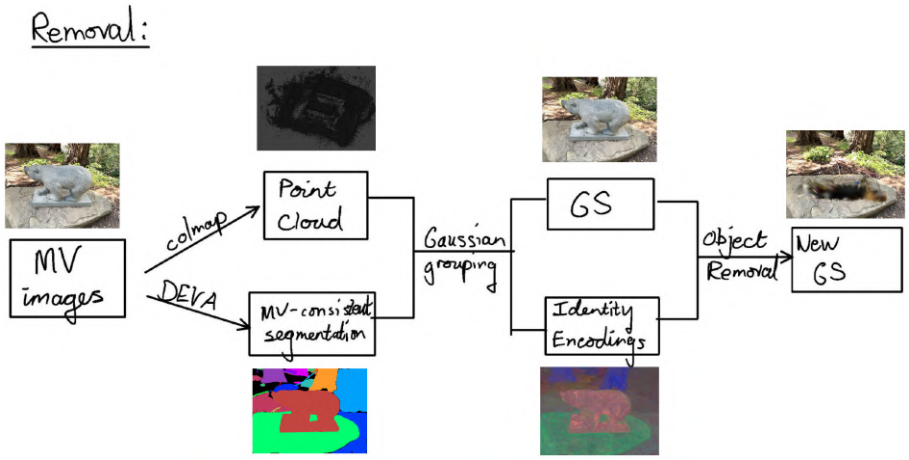

Figure 3.1: The removal pipeline starts off with the multi-view image sequence. After obtaining a point cloud through Colmap and 2D segmentations via DEVA, we can use these to train a Gaussian splat that is 3D segmented via Gaussian Grouping. Once this is trained, object removal is trivial due to the 3D segmentation.

and the point cloud are used to train the normal Gaussian splat. Also having the segmentations, the identity encodings, or the 3D point gaussians segmentations, is also trained. Both the idenitity encodings and the gaussian properties are properties of each point. Hence, with the point cloud segmented in 3D, object removal is carried out, producing a new point cloud/gaussian splat without the removed points.

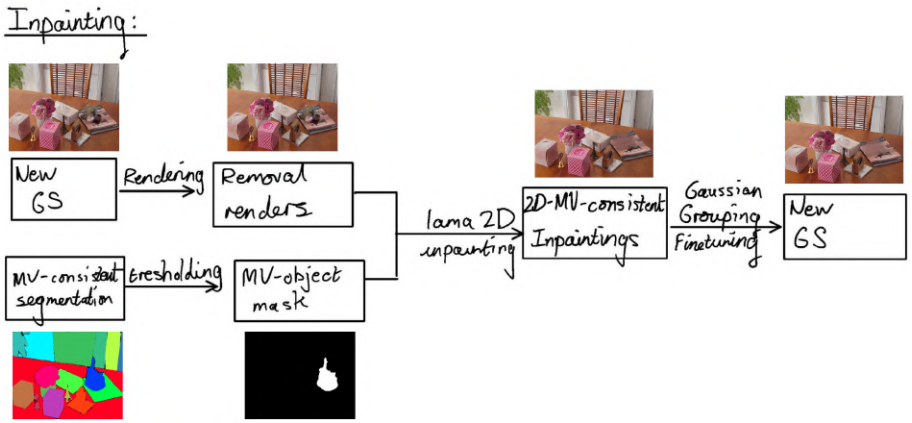

Figure 3.2: Inpainting masks (that are the removed object masks) along with renders of the gaussian splat after removal are used to produce 2D inpaintings. These 2D inpainings are then used to finetune the 3D gaussian splat, so that the 3D gaussian splat then also becomes inpainted.

In the inpainting pipeline, Fig. 3.2, two subprocesses must initially happen. Firstly, from the gaussian splat after object removal, renders of the gaussian splat from the initial camera poses are extracted. Secondly, we need to obtain object-masks. These are produced by performing thresholding on the multi-view consistent segmentations from the removal pipeline. Each render will have a corresponding object-mask. With these render and mask pairs, 2D inpainting can

happen with Lama. For each pair, the object mask region, which contains the hole region in the render, will instruct Lama as to the region to be inpainted. Inpainting in 3D will now happen, which involves the tuning the gaussian parameters to make renders of the gaussian splat as similar to the inpaintings from the same camera angles as possible. Hence, the 2D inpaintings are used to finetune the gaussian splat, to produce a gaussian splat with the hole region inpainted.

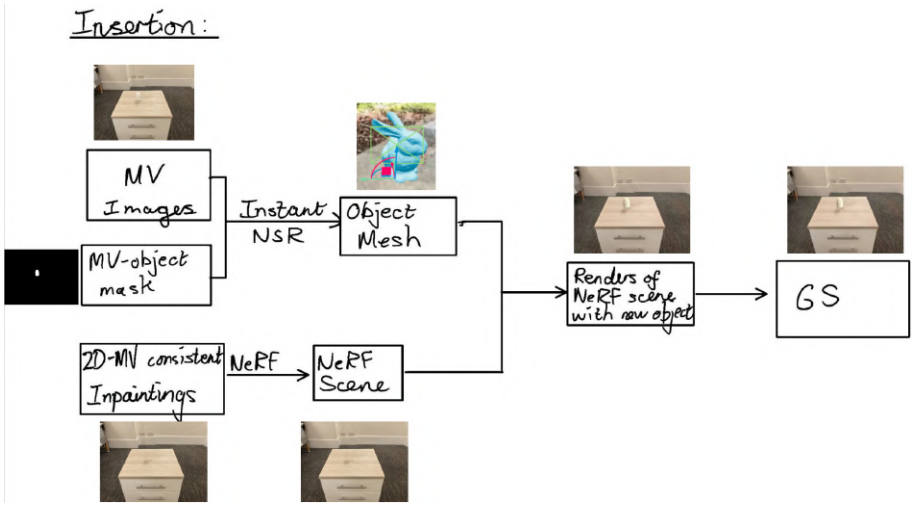

Figure 3.3: A NeRF scene is initially trained through the inpainting renders. SIGNeRF then produces images of a new object within the scene, which are used to finetune the NeRF scene so that the new object is then incorporated.

In the final stage of object insertion, Fig. 3.3, we use the inpainted gaussian splat renders to train a NeRF scene, and insert the removed object into the NeRF scene at a new location compared to before. We then take renders of the NeRF scene to train an entirely new gaussian splat. This is because the NeRF scene is not a point cloud, whereas the gaussian splat is. This gaussian splat represents the output of the entire project pipeline. To insert an object into the NeRF scene that is trained, we need to produce some renders of the object in the scene from a certain camera pose. To do this, we use inpainting with a diffusion model and ControlNet. This process requires a NeRF scene render, an inpainting mask (object mask), and a depth map for ControlNet conditioning. To obtain the inpainting mask and the depth map, we insert a mesh into the NeRF scene, as shown in the pipeline in Fig. 3.3.

3.2 Design Choices

3.2.1 3D Segmentation

Purely 3D segmentation approaches—such as region growing, graph-based clustering, or end-to-end deep nets like PointNet++, KPConv, and RandLA-Net—operate directly on raw point coordinates and often rely on local geometric features (normals, curvature) or learned point-wise embeddings. While these methods can capture fine spatial structure, they frequently struggle with noisy scans, nonuniform sampling densities, and the limited availability of large, richly annotated 3D datasets. They also tend to be computationally intensive, since every point must be compared or aggregated in 3D space.

Gaussian grouping takes a different tack by projecting the point cloud into multiple 2D views, running high-accuracy 2D segmentation on each image, and then clustering points in 3D based on a Gaussian-weighted agreement of those segmentations. This hybrid strategy inherits the strengths of mature 2D networks—including robust edge localization, texture awareness, and access to vast labeled image corpora—while still working directly on the original 3D data. Because each point's label emerges from a consensus of photorealistic renderings, the method is naturally more resilient to sensor noise and uneven sampling.

By definition, Gaussian grouping delivers more realistic, semantically coherent segmentations: it leverages the same cues a human would use in images (color gradients, object silhouettes) yet projects them into 3D. It avoids the heavy memory footprint and sparse-data pitfalls of volumetric techniques, and it sidesteps the generalization limits of purely 3D learners by piggybacking on the maturity of 2D vision models. In practice, this often yields higher accuracy, faster convergence in training, and segmentations that align perfectly with both geometry and appearance—making Gaussian grouping a preferred choice for point-cloud segmentation.

3.2.2 General Inpainting Method

In this project, 3D inpainting is completed through inpainting 2D renders of the 3D scene, and using these 2D inpaintings to train a new 3D point cloud. This has been chosen over purely 3D inpainting methods(e.g. Poisson surface reconstruction, volumetric CNNs (e.g., 3D-Unet), point-based networks (PCN, SnowflakeNet), and implicit-field models (DeepSDF, Occupancy Networks)),

which have advanced hole filling but still face trade-offs. Voxel grids become prohibitively large at high resolution; point-cloud networks struggle with sparse, unstructured data; and implicit fields often demand costly per-scene optimization. Moreover, limited 3D datasets constrain their ability to generalize complex, arbitrary hole shapes.

The 2D-driven approach used in this project sidesteps these issues. By rendering the incomplete point cloud into multi-view images, you leverage mature image-inpainting networks that handle large missing regions, produce fine detail, and work at flexible resolutions without cubic memory growth. Crucially, because these inpainted views are actual renderings of the scene, they guarantee photorealism by definition—textures, shading, and edges appear naturally as they would in real captures. Fusing these realistically filled images back into 3D provides your model with rich, high-fidelity supervision, yielding reconstructions that are both semantically plausible and visually exact.

3.2.3 Retraining vs Finetuning

In the Gaussian Grouping project, removal and 3D inpainting was achieved by using the 2D inpaintings to finetune the gaussian splat. However, in this project, the 2D inpaintings have been used to train a completely new gaussian splat from scratch. While this is less efficient and more time-consuming, experiments on using the method shown in Gaussian Grouping show that finetuning the gaussian splat leads to artifacts in the resulting renders of the new gaussian splat. Although the finetuning process aims to minimise the difference between the 2D inpaintings and renders of the gaussian splat, often time renders from these methods look unrealistic for the human world, as shown in Fig. 3.4.

Figure 3.4: The two example images above show the poor results of the finetuned Gaussian Splat. Some of the gaussians after fine tuning make the scene look unrealistic, and the renders of the scene are significantly worse than the 2D inpainting results.

3.2.4 2D Inpainting

Inpainting Mask Generation

DINO+DEVA Object Tracking is the method for obtaining the inpainting mask in Gaussian Grouping. However, this method is designed for tracking regular real world objects, as opposed to artificially created hole regions exposed by object removal. If the hole is not obviously defined with clear contours in a frame, the method will not be able to detect the hole on that frame, or it might wrongly detect a different object. Missing the object means there will be no inpainting mask produced for that frame, which will lead to the render at that frame not being rendered, and the wrong object being identified will lead to the wrong region in the corresponding render being inpainted. Furthermore, in the initial multi-view image sequeunce is rendered from a mesh, as is one of the input types of this project, normal objects are often poorly tracked, as shown in Fig. 3.5.

Figure 3.5: Object detection results when trying to detect the bed. The object with the highest probability of being the prompt is accepted as the mask, but the actual bed is not the object with the highest porbability.

Using the object mask of the removed object as the inpainting mask is instead the preferred method of this project. The advantage over using DINO+DEVA Object Tracking is that we can consistently a mask covering the hole region for every frame. The object mask is obtained from the multi-view segmentation image. The colour representing the removed object is turned white,

and the remaining image turned black, to produce the mask. The object segmentation results are reliable, and hence trusted to produce reliable inpainting masks.

The problem with using the object mask is that the exposed hole is a smaller area than the object mask region. Hence, the object mask covers some area that does not need inpainting.

Diffusion Based Methods + ControlNet

Despite diffusion based inpainting producing results that look realistic and consistent with the rest of the image, the inpainting sometimes contains a new image in the inpainting area, as shown in Fig. 3.6.

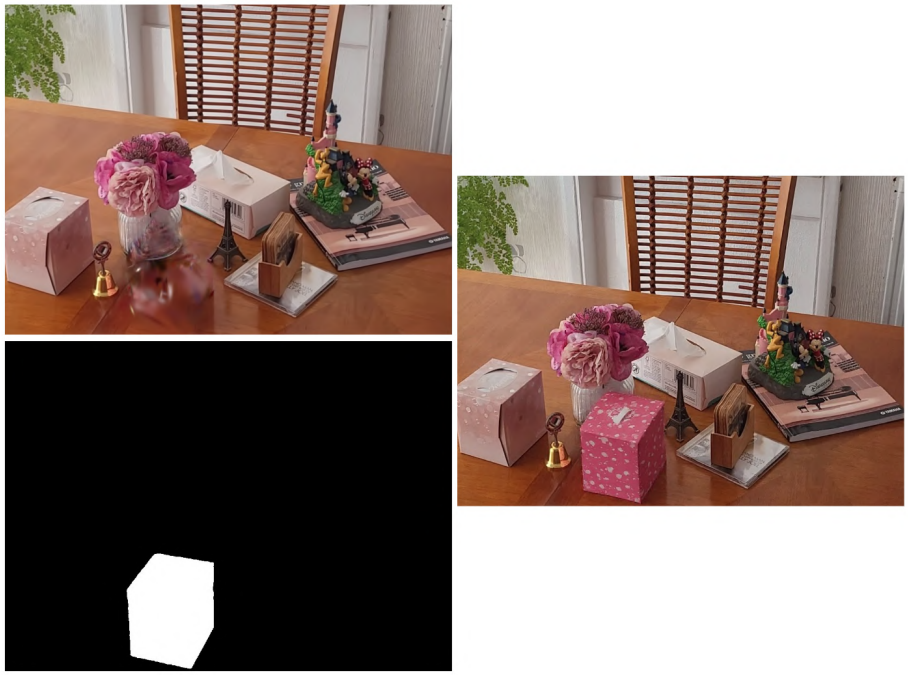

Figure 3.6: Result of diffusion inpainting with the FLUX.1-Fill-dev inpainting model. The original image and the inpainting mask are shown on the left, with the result on the right. An object has been generated in the inpainting region, which is not the desired outcome.

ControlNet can be used to prevent the problem of new objects being inpainted. This can be done by using ControlNet to condition the inpainting on a segmentation that is consistent with the scene after inpainting, as shown in Fig. 3.7.

The problem is that there is no universal way to transform the segmentation map after removal to the segmentation map of the desired inpainting. In Fig. 3.8 the transformation is done through manually painting blue onto the original segmentation map. This is a more straight forward case due to the exposed hole region only being on top of the carpet in this case, so the hole region can

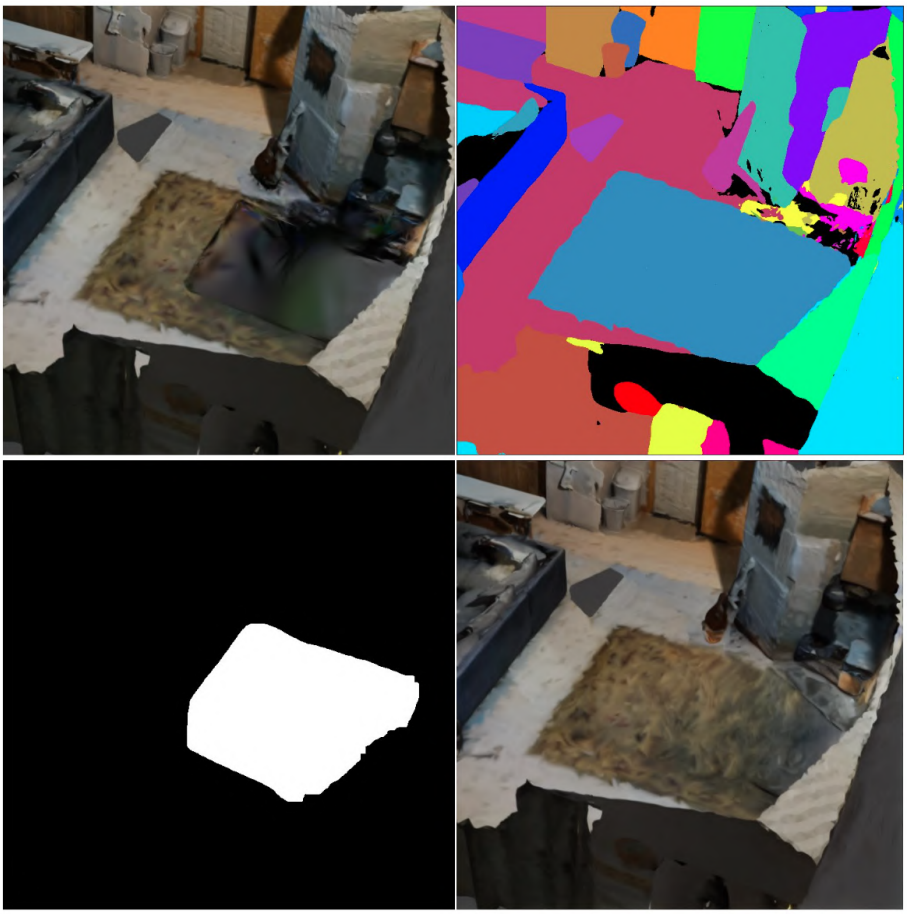

Figure 3.7: Result of diffusion inpainting with ControlNet conditioned on a segmentation map. The result, in the bottom right is of a good quality. However, this method is hard to achieve due to the difficulty of obtaining the segmentation mask in the top right.

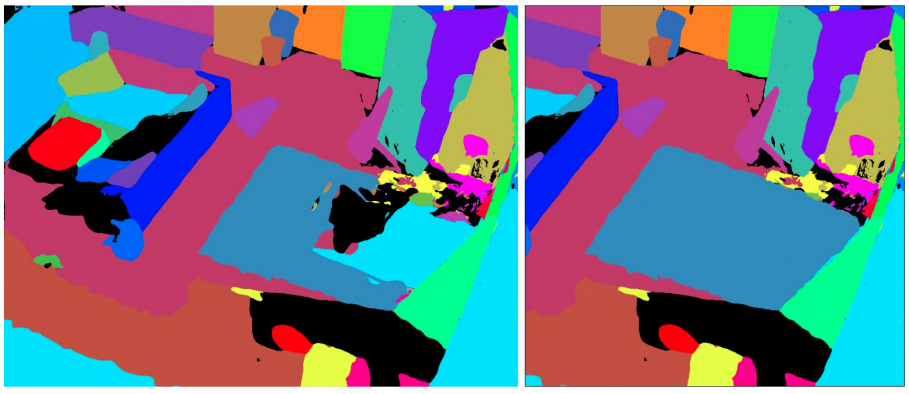

Figure 3.8: The left image shows the segmentation result on renders of the Gaussian Splat after removal. The right image is the desired segmentation map, which has been manually produced through a painting software. It is infeasible to manually paint each segmentation map from each camera angle.

fully be painted dark blue. In other cases the bed may sit on top of several different objects.

3.2.5 DALL-E

DALL-E is able to do inpainting without an inpainting mask. It is conditioned on a text prompt, and the degree to which the results follow the promptis very high. The problem is that DALL-E must change the image to its own representation before any editing operations are made, as shown in Fig. 3.9. This not only changes the style of the scene completely, but also changes some of the details within the image outside of the area to be inpainted.

Figure 3.9: Result of diffusion inpainting with DALL-E. Even though the inpainted content is very accurate, DALL-E has changed the style of the images.

Lama

The chosen method of inpainting in this project is Lama, which is also the method used in Gaussian Grouping. Despite not producing as realistic results as diffusion models, Lama fully uses its surrounding region to predict the inpainted area, so an unexpected new object won't appear. Furthermore, the image after inpainting still keeps its original style, as opposed to DALL-E. The results are realistic for simple cases, however when there are multiple different textures around the inpainting area, the inpainted region becomes an unrealistic mix of its surrounding textures, as shown in Fig. X. Nonetheless, given the shortcomings of the other methods, this is the best option.

Figure 3.10: Result of diffusion inpainting with Lama. The castle from the top left has been inpainted. Although the inpainting results are reasonable, the cover of the book has not been realistically predicted.

4 Implementation

Contents

4.1 Scene Capture

4.1.1 Real World Images

To prepare multi-view inputs for COLMAP, capture a continuous sequence of high-quality photographs that fully surrounds your scene of interest. Modern smartphone cameras typically offer sufficient resolution—as long as your shots are sharp, well-exposed, and clearly resolve scene details, COLMAP will be able to extract reliable feature matches. Below are some points to consider for taking a satisfactory set of images:

4

Figure 4.1: To obtain the initial multi-view images as input to the pipeline, take a range of images around the scene that is desired for the point cloud. The original scene in this image is taken from [53].

Complete Angular Coverage Perform at least one full 360° orbit around the central object, keeping the camera pointed at it throughout. For greater reconstruction fidelity—especially when rendering from novel viewpoints—consider additional revolutions at varying heights and trajectories (e.g., raised or dipped camera angles).

Frame Spacing and Overlap Ensure consecutive frames overlap by roughly 60–80% of the scene content. Too little overlap makes feature matching harder; too much yields little new information. Frames extracted at 0.5–1 m intervals around the scene typically suffice. Videos are not recommended, as the high fps often adds frames without introducing a noticeable improvement in point cloud reconstruction.

Trade-Offs: Quantity vs. Processing Time More images generally improve point-cloud density and novel-view rendering quality—but they also increase COLMAP's reconstruction time and downstream training costs for Gaussian Splatting or NeRF. Aim for 50–150 well-overlapped images for a typical tabletop-sized scene; scale up or down according to scene complexity and available compute.

By following these principles—comprehensive orbiting, controlled overlap, and mindful framing—you'll produce a robust image set that maximizes COLMAP's reconstruction accuracy without incurring unnecessary processing overhead.

4.1.2 Point Cloud Renders

To generate a new multi-view image set for object repositioning, we render the original point-cloud mesh in Blender rather than re-photographing the scene. In practice:

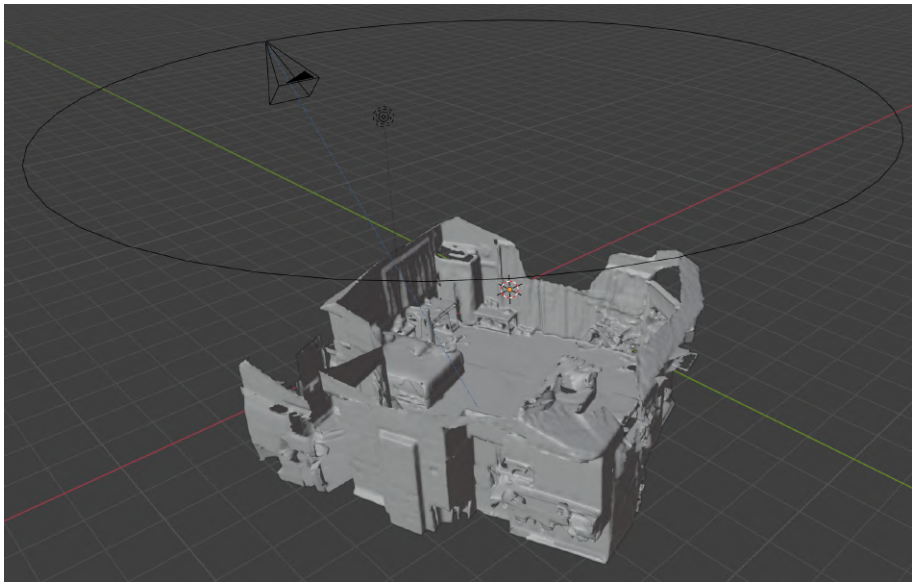

Figure 4.2: To obtain the initial multi-view images as input to the pipeline, take a range of images around the scene that is desired for the point cloud.

Import and Mesh Conversion Load your point-cloud into Blender and convert it to a watertight mesh (e.g., via Poisson surface reconstruction). Apply any necessary cleanup or smoothing to ensure consistent shading.

Camera Path Setup Create a circular (or custom-shaped) camera rig that orbits the object at a fixed radius. Position the path slightly above the object's midline to expose both top and side features.

Image Sampling Configure the path to complete one 360° revolution. Sample 150 evenly spaced frames along the path—this yields sufficient overlap (60–80%) while keeping render times manageable. Use consistent focal length and aperture settings to match your later real-world captures.

Rendering Considerations Enable ambient occlusion and soft shadows to enhance depth cues, but avoid overly complex lighting that slows down batch renders. Render at the same resolution you intend to process in COLMAP (e.g., 2,000 × 1,500 px) to eliminate extra rescaling steps.

By following these steps—mesh preparation, controlled camera motion, and judicious frame spacing—you'll obtain a synthetic image sequence that integrates seamlessly with your real-world captures, enabling accurate COLMAP reconstruction and object repositioning.

4.2 Multi-View Segmentation

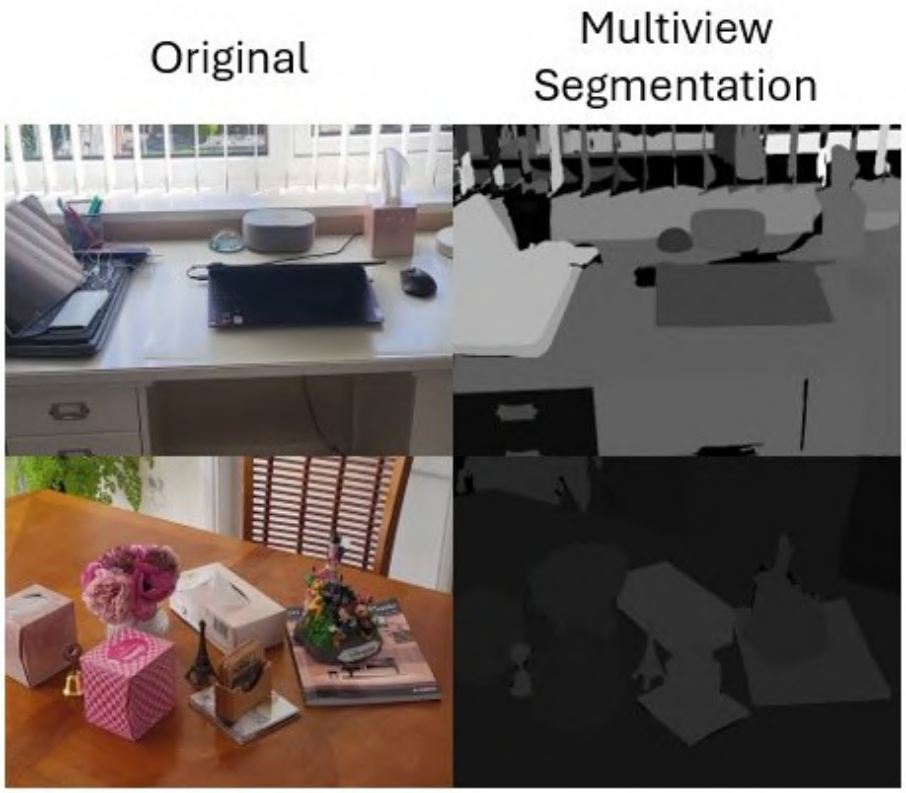

Figure 4.3: The results of using DEVA for multiview segmentation. The top row shows an inaccurate segmentation whereas the bottom row is accurate.

DEVA is used in this project for multi-view consistent segmentation. Fig. 4.3 shows a poor result of multi-view segmentation by DEVA in the top row, and an accurate result in the bottom row.

In the top row, transparent windows, dynamic lighting, reflective screens, and thin wires each undermine DEVA's core assumption of consistent, view-invariant appearance, leading to unreliable segmentation. Transparent surfaces distort and blend background textures differently at each viewpoint, preventing feature matching from correctly associating window regions across views. Variable shadows and highlights introduce non-geometric intensity changes that DEVA may misinterpret as object boundaries or background clutter. Highly specular displays and screens produce view-dependent glare and color shifts, breaking the photo-consistency DEVA relies upon to propagate labels. Finally, slender wires often fall below the algorithm's matching threshold, yielding sparse or incorrect correspondences that cause label "drop-out" and fragmentation. Together, these factors disrupt DEVA's multi-view constraints, resulting in noisy, incomplete, or erroneous

segmentations.

By contrast, the bottom rows present much simpler scenes, featuring well-defined object boundaries and far fewer complications from challenging lighting, reflections, or thin, hard-to-capture elements.

When selecting a scene for image capture, the scene must be carefully assessed in terms of its complexity, and any elements—like specular surfaces, transparent materials, or subtle "thin" structures—that could undermine DEVA's multi-view segmentation accuracy must be identified. Without accurate segmentation, certain objects cannot be removed in the following stage, the inpainting mask cannot be identified for inpainting, and an accurate mesh cannot be constructed for object insertion. Hence, multi-view segmentation is crucial for the success of the project's pipeline.

4.2.1 SAM+DEVA settings

There are two settings available to tune when performing consistent multi-view segmentation with SAM and DEVA: box threshold and NMS (Non-Maximum Suppression) threshold.

The box threshold is the minimum confidence score that a candidate region proposal (or "box") must have before SAM attempts to generate a mask for it; raising this threshold (e.g. toward 0.3–0.5) reduces spurious, low-confidence boxes but risks missing small or faint objects, while lowering it (toward 0.1–0.2) captures more potential objects at the cost of more noise. The NMS threshold is an intersection-over-union (IoU) cutoff used to merge or discard overlapping mask hypotheses—setting it low (e.g. 0.3) aggressively suppresses overlapping candidates to yield one mask per object, whereas a higher value (e.g. 0.5–0.6) allows more overlap and preserves multiple hypotheses for later fusion. In practice, you tune the box threshold to balance recall versus precision in your scene's object scales and textures, then adjust NMS to control whether you want strictly one mask per object or more redundant views for DEVA's multi-view consistency step.

In the dining table scene in the bottom row of Fig. 4.3, due to the well-contrasted objects on a plain background, both thresholds were tightened to cut out nearly all noise. The box threshold up was set to 0.35, so only strong, unambiguous region proposals (e.g. your distinct objects) got masked. An NMS threshold of 0.4–0.5 was chosen.

4.3 Removal

4.3.1 Obtaining Object ID

From the previous step, multi-view segmentation assigns each object an ID, which corresponds to the greyscale value in Fig. 4.3. The ID of the object to be removed must be specified in a removal configuration file before removal, and must first be identified.

The process of indentifying the object ID is as following scripts:

1. highlight_pixel.py takes an image and image pixel coordinates as input, and outputs the image with the given pixel highlighted. This script is used iteratively to find a pixel of the object to be removed, as shown in Fig. 4.4.

2. get_greyscale_val.py takes an image and image pixel coordinates as input, and outputs the greyscale value of the specified pixel on the image.

Figure 4.4: The process of highlighting a pixel of the selected object to obtain its object ID, through its greyscale value.

4.3.2 Removal Settings

After segmentation in 3D has occurred on the point cloud, each point is labeled with a length-256 vector of probabilities. Eacch value represents the probability of this point belonging to one of the 256 classes. Typically, the probabilities of most of the classes will be small bar one class. Hence, when performing removal of an object class, a threshold must be set. All gaussians in the scnee with probability of the specified class greater than the specified probability will be removed.

4.4 Inpainting

Inpainting involves two steps: inpainting the 2D removal renders to produce 2D inpaintings, and using these 2D inpaintings to finetune the gaussian splat, achieving inpainting in 3D.

To inpaint all the removal renders, an inpainting mask is required for each render. This can be obtained from the multi-view segmentation results. For each removal render, the corresponding multi-view segmentation, from the same camera angle, can be retrieved. This segmentation should then be thresholded to create a mask: the pixels belonging to the removed object are set to white, and all other pixels are set to black. The Lama inpainting tool then takes each (removal render, inpainting mask) pair and produces a 2D inpainting.

For inpainting the gaussian splat in 3D, the inpainting masks and the 2D inpaintings need to be provided. The inpainting masks are there so that the gaussian splat finetuning/training process knows which gaussians to focus on. A configuration file must be specified for 3D inpainting, which among other settings allows the number of finetune iterations and the weighting of LPIPS (Learned Perceptual Image Patch Similarity) loss compared to L2 loss for finetuning.

4.5 NeRF Training and Insertion

For object insertion there are four steps: training a NeRF scene using the inpainting renders on nerfstudio, creating a mesh file of the object to be inserted, using SIGNeRF to generate images of the inserted object, by placing the mesh into the NeRF scene, and using these images to finetune the NeRF scene, and finally taking renders of the final NeRF scene to create a new point cloud with the inserted object, via Gaussian splatting.

4.5.1 Training the initial NeRF scene

Firstly, the initial inpainting renders are inputted to Colmap again to initiate a point cloud. Note that the same Colmap output from Gaussian Grouping cannot be used here, as nerfstudio expects a slightly different format. Next, the point cloud and the renders are used to train the NeRF scene using nerfstudio's 'nerfacto' command. The training process can be viewed in an interactive web viewer.

4.5.2 Generating a mesh of the object to be inserted

The mesh is generated using Instant-nsr. Instant-nsr takes the original input to this project's pipeline, the multi-view image sequence, as well as the 2D inpainting masks as input. The image sequence is again inputted to Colmap to produce a point cloud. A training process can then be initalised which extracts a mesh consistent with the area specified by the inpainting masks.

4.5.3 SIGNeRF

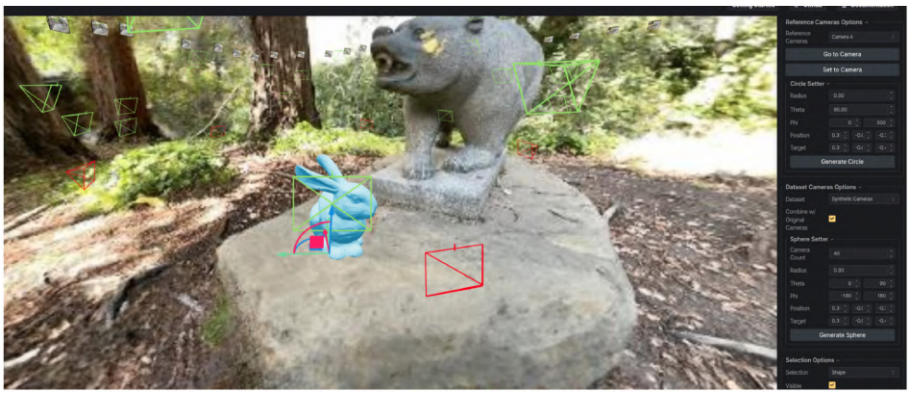

Figure 4.5: The interative viewer in SIGNeRF where mesh insertion and diffusion settings are chosen.

4.1 Reference vs. Dataset Cameras In our pipeline, dataset cameras refer to the original viewpoints used to train the base NeRF (e.g., those exported from COLMAP and loaded into Nerfstudio). These cameras provide the photometric and geometric priors that anchor the scene's implicit representation. By contrast, reference cameras are a new set of virtual viewpoints that we place interactively using 'ns-train signerf'. Each reference camera is defined by its own extrinsic parameters within the trained NeRF coordinate frame and serves as a target from which we will render novel views for object insertion.

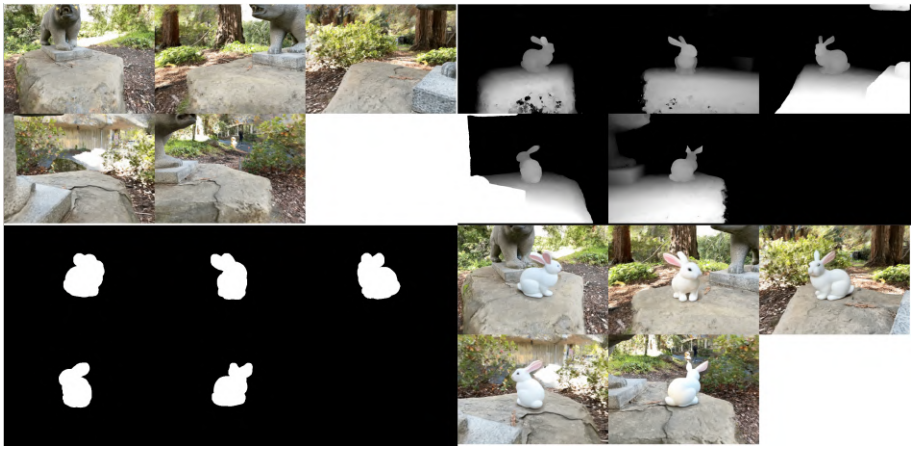

Figure 4.6: The diffusion process SIGNeRF uses for multi-view image consistency. Reference generations guide the new generations.

4.2 Camera Placement and Rendering Once the scene is loaded in Nerfstudio's SIGNeRF viewer, as shown in Fig. 4.5, we add a series of reference cameras at desired positions around the region of interest. For each reference camera, we render an RGB image and its corresponding depth map (or object silhouette) using the NeRF's differentiable renderer. These renders form the basis for both mask-based mesh placement and subsequent inpainting.

4.3 Mesh Alignment and Silhouette Extraction The 3D mesh of the object to be inserted is first aligned to the NeRF coordinate system—either by manual registration or by optimizing a rigid transformation to match roughly the scene scale and orientation. We then project this mesh into each reference camera to generate per-view binary masks. These masks precisely delineate where the mesh would appear in each render, ensuring that the inpainting step knows exactly which pixels should contain the new object.

4.4 Diffusion-Model Inpainting Masked reference renders are passed to a pretrained diffusion-based inpainting model (e.g. Stable Diffusion with depth guidance). The model fills each mask region with a photorealistic appearance of the target object, conditioned on the surrounding context and the depth map to maintain correct occlusions and lighting cues. Because each view is inpainted independently, this stage produces a set of novel images showing the object seamlessly integrated from multiple angles. The diffusion process can be seen in Fig. 4.6.

4.5 Multi-View Consistency via SIGNeRF Optimization To reconcile any subtle discrepancies across independently inpainted views, SIGNeRF performs a final optimization: it augments the original scene's signed distance field with an additional implicit field for the inserted object. Using the inpainted images as supervision, it minimizes photometric and silhouette losses across both

the dataset and reference cameras. The result is a unified NeRF that renders the newly inserted object consistently and coherently from any viewpoint, achieving true multi-view consistency.

4.5.4 Getting the final point cloud

Unlike a NeRF—which encodes scene appearance in a fully implicit volumetric field without any discrete points—a Gaussian Splat representation produces an explicit point-based model. To recover a point cloud for our augmented scene, we therefore re-apply the Gaussian-splat extraction pipeline introduced earlier. Specifically, we feed a set of novel-view renders from the final NeRF (with the inserted object) into the same Gaussian splatting algorithm used originally.

5 Evaluation

Contents

5.1 Introduction

Basic Tests

1. Removal: We remove an object from a scene, and compare its renders with real ground-truth images where the object has actually been removed.

2. Repositioning: After the object has been removed, we add the same object back into the scene at the exact same location. We then compare its renders with the ground truth images of the original scene before removal.

3. Replacement: After the object has been removed, we add a different object back into the scene at the exact same location. We then compare its renders with the ground truth images of the a scene taken with the same background but with the inserted object rather than the original object.

While repositioning technically is not moving the object to a new position, adding the object back in the same position allows for easier obtainment of ground truth, and is still an accurate measure of insertion ability. Also, object replacement has also been added as an operation to test insertion ability, giving some insight in to the effects of adding novel objects, rather than the one native to the scene.

Different Methods to Compare using Tests *Problem statement 1: Comprehensively evaluate the strengths and limitations of the current SOTA methods in removal, inpainting and insertion thorugh theoretical, qualitative and quantitative analysis. Compare with other methods when applicable.*

In line with the problem statement, this section tests the SOTA methods in inpainting, Lama, and insertion, SIGNeRF, that have previously been justified theoretically in Section 3. Removal (in terms of how fully points are removed from the point cloud) is not separately evaluated, as it belongs together with inpainting, and the results they obtain when perform together is more representative of object removal.

As mentioned, the strengths and limitations of the methods is to be tested, as well as comparison to other methods when feasible. Hence, for (removal+inpainting) we will compare the results of different inpainting methods, whereas for insertion, where we have settled on the SIGNeRF method, only the SIGNeRF method will be evaluated, however, different components in the SIGNeRF pipeline will be specifically tested, to comprehensively determine the strengths of SIGNeRF. The specific tests that will be carried out for each of removal/inpainting and insertion are the following:

Removal:

1. Retraining vs Finetuning Gaussian Splat with 2D inpaintings
2. Lama Inpainting
3. FLUX (Diffusion Model) Inpainting
4. DALL-E Inpainting

Insertion:

1. Inserting by only adding a mesh

2. SIGNeRF insertion without text prompt
3. SIGNeRF insertion with poor reference cameras
4. Full SIGNeRF pipeline
5. Comparison of full SIGNeRF with different diffusion models.

How tests are evaluated Furthermore, for insertion, we will test two operations: object repositioning and object replacement.

For each operation, we will perform both quantitative and qualitative evaluation. Quantitative analysis will involve calculating PSNR and SSIM between a render after the operation has been completed, and a ground truth image. Qualitative evaluation will involve evaluating images. In terms of the difficulty of the operations, a variety of difficulties from easy to difficult will be considered across the different operations.

The procedure for obtaining the experimental data, including the pipeline inputs and ground truth images, is described below.

5.2 Experimental setup

For each operation, we will evaluate two different scenes quantitatively.

What types of images to capture needs to be considered before evaluation can occur. The image sequences we take need to be sufficient to evaluate all three operation types, as well as providing ground truth images for quantitative evaluation. Let us consider how images will be taken for each operation type.

1. Object Removal: This is the most straight forward case. We will take two sets of multi-view image sequences, representing the two scenes. For each scene, at each cameras pose, we take two images: one with the object to be removed, and one without. This way, at the end we get two sets of multi-view images per scene: one with the object and one without. The set of images with the object will represent the scene to be operated on, and the set of images without the object will act as the ground truth for removal. Once the object has been removed from the point cloud of the first sequence, renders after removal of the scene will be compared to the ground truth in the second sequence with the same camera pose.

2. Object Replacement: The two scenes complement each other well. The object to replace the original object in the first scene will be the object from the second scene, and vice versa. Hence, given the background stays the same between scenes and the only difference is the object to be removed, the sequence with the object in the first set will act as the ground truth for the second scene, and vice versa.

3. Object Repositioning: for each image sequence/scene, after we have removed the object, we will insert the object back in the exact same place. Hence, for each scene, the original image sequence will act as the ground truth.

Observing the above statements, it can be seen that, if the first scene and the second scene are taken with the exact same backgrounds but with a different object to be removed, only three image sequences need to be taken: a sequence with the first object, a sequence with the second object, and a sequence with neither object. The sequence with neither object as the ground truth for object removal *both* for both scenes, the sequences with an objects act as the ground truth for each other's object replacement, and each sequence acts as the ground truth for its own scene's object repositioning. Some images from each of the sequences is shown in Fig. 5.1.

Object Choice For the two scenes, the background remained the same, however different objects were used to differentiate between the scenes. The white chewing gum bottle was chosen as the simple scene, whereas the Iron Man figurine was chosen as the more difficult scene. A contrast in difficulty is important to gauge the extent of the pipeline's abilities. Here is why the two objects differ in difficulty:

1. Due to the simpler structure of the bottle, in removal/inpainting the inpainting mask will be simpler, leading to a potentially easier inpainting task. Furthermore, in insertion, with a simpler 3D structure, the depth conditioning for the diffusion process will be simpler. On the other hand, the more complicated structure of the figurine should lead to more difficult removal/inpainting and insertion processes.

2. The figurine is marginally taller than the bottle, meaning that in most multi-view input images of the scene the figurine crosses the threshold of the table it sits on. On the other hand, it can be seen that the bottle is almost always fully surrounded by the wooden texture in 2D photos. This is important because when the segmentation mask of the object is fully surrounded by one texture, it makes the inpainting process much simpler.

Due to the two objects being of similar size and perimeter length, it also allows SSIM and PSNR comparisons between the two scenes to be more meaningful.

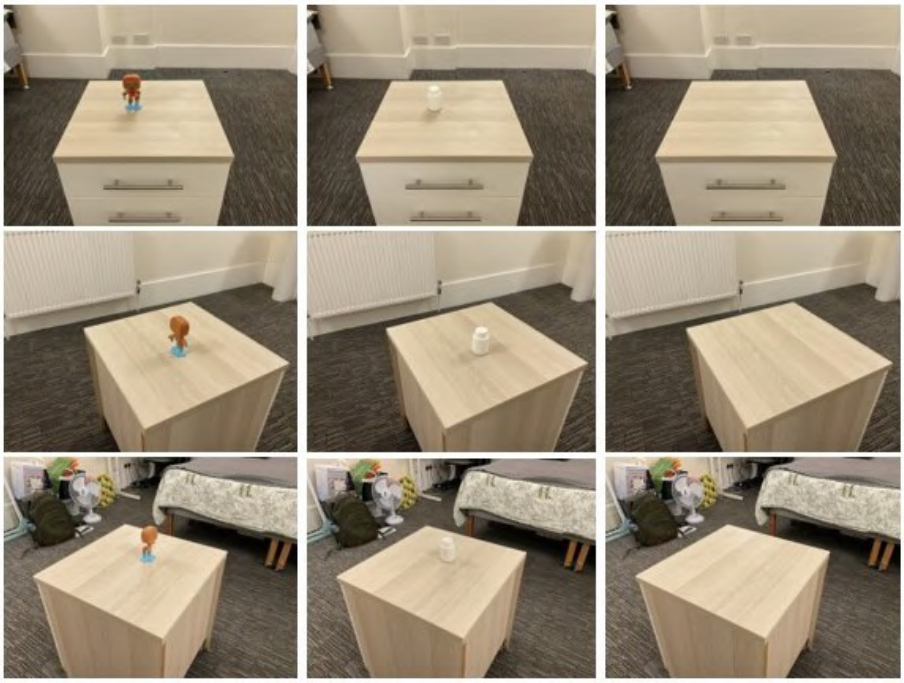

Figure 5.1: 59 images were taken for each of the three scenes: the scene with the figurine, the scene with the bottle, and the scene without an object. 3 images have been chosen from each scene for illustrative purposes.

5.3 Results

5.3.1 Removal/Inpainting

Lama Here we evaluate the performance of Lama, the model used by Gaussian Grouping for inpainting.

	SSIM		PSNR (dB)	
	Capsule	Iron Man	Capsule	Iron Man
0◦	0.8924	0.8210	28.10	26.72
180°	0.8766	0.8212	27.00	25.96

Table 5.1: SSIM and PSNR values of Lama inpainting compared to the ground truth scene without an object in it.

Qualitatively, from Fig. 5.2, it can be seen that Lama performs relatively well for both the bottle and figurine scenes: the object has been fully removed, and the inapainted texture preserves the table and carpet textures in both cases, in a realistic manner. The only noticeable difference

Figure 5.2: Lama inpainting compared to the ground truth scene without an object in it.

between the inpainting renders and the ground truth images is the presence of the shadows in the inpainting renders. This is because after the object was initially removed, the shadows still remained. Some shadows persisted around the inpainting mask, which also causes the inpainted region to have shadows. Since the figurine has a slightly larger area due to its height, it casts a greater shadown than the bottle.

Finetuning vs Retraining In Fig. 5.3 renders of the Gaussian Splat that has been finetuned with the 2D inpainting results are shown.

Figure 5.3: 3D inpainted Gaussian Splat renders compared to the ground truth scene without an object in it.

Looking closely at Fig. 5.3 it can be seen that some Gaussians artifacts, which haven't been finetuned to where they should be, make the scene look unrealistic. This is potentially due to the

	SSIM		PSNR	
	Capsule	Iron Man	Capsule	Iron Man
0◦	0.8613	0.7821	27.57	25.57
180°	0.8492	0.7781	26.61	25.27

Table 5.2: SSIM and PSNR values of the inpainted 3D Gaussian Splat after finetuning compared to the ground truth scene without an object in it.

suggested number of finetune iterations in Gaussian Grouping not being enough to train a realistic scene. The finetuning time compared to the time required to retrain a new Gaussian Splat is a lot less, which could explain the above results.

FLUX Here we evaluate FLUX, which is currently the state-of-the-art diffusion model for inpainting.

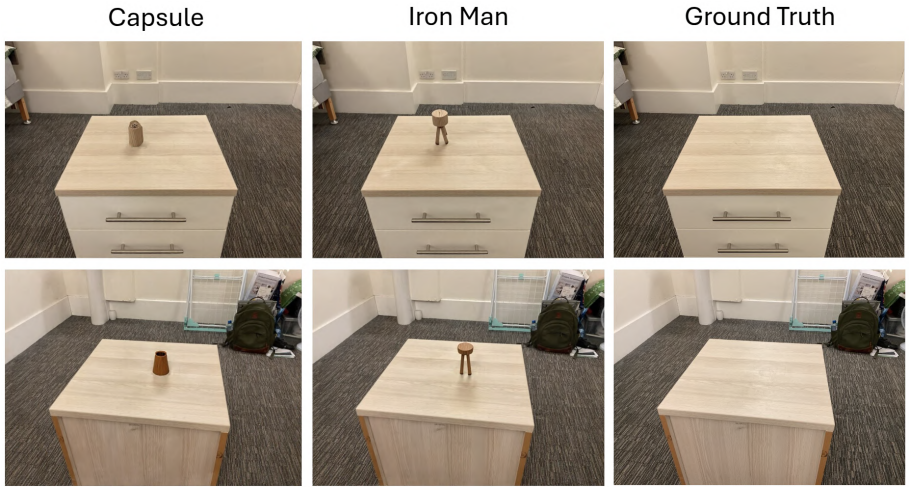

Figure 5.4: FLUX inpainting compared to the ground truth scene without an object in it.

	SSIM		PSNR	
	Capsule	Iron Man	Capsule	Iron Man
0◦	0.5265	0.5226	19.79	19.61
180°	0.4915	0.4747	19.26	19.42

Table 5.3: SSIM and PSNR values of FLUX inpainting compared to the ground truth scene without an object in it.

Figure 5.5: The resulting generations after applying the diffusion inputs specified in Tab. 5.4.

From Fig. 5.4 it can be seen that inpainting with the FLUX diffusion model doesn't quite work, as rather than continuing the pattern of the surface in the inpainting area, FLUX adds a

Attempt	Prompt	Guidance Scale
1	NO PROMPT	0
2	No object is inserted	30
3	No object is inserted	50
4	A wooden-textured table surface	50

Table 5.4: Attempted diffusion settings to achieve an inpainting with no object added. Unfortunately, all attempts still resulted in an object addition.

new object. The problem is likely due to the shape of the inpainting mask provided to FLUX. For example in the figurine scene, looking at the inpainting mask, it is clear that the mask resembles some sort of humanoid figure. FLUX uses the inpainting mask shape information, leading to the inaccurate results.

Attempts were made to try and stop an object from being generated from FLUX inpainting. The results and strategy can be seen in Fig. 5.5 and Tab. 5.4.

Figure 5.6: DALL-E inpainting compared to the ground truth scene without an object in it.

DALL-E From Fig. 5.6 it can be seen that for DALL-E, while the quality of the object removal is high, the overall style of the results looks slightly different to that of the real world images in the ground truth. DALL-E generations are said to be ”hyperreal” and from a ”dream world”, with colours being slightly more vivid than real images. Furthermore, from the bottom middle image in Fig. 5.6 it can be seen that the green backpack has been removed in this DALL-E generation, showing that the objects of a scene are not always fully preserved. The field-of-view of the DALL-E inpaintings also differs from the original images, making quantitative evaluation using SSIM and PSNR less meaningful.

5.3.2 Insertion

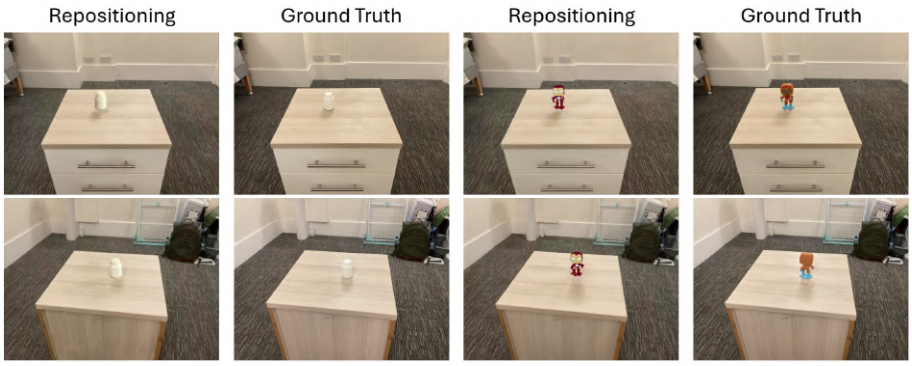

Figure 5.7: Repositioning results compared to the ground truth scene with the actual object.

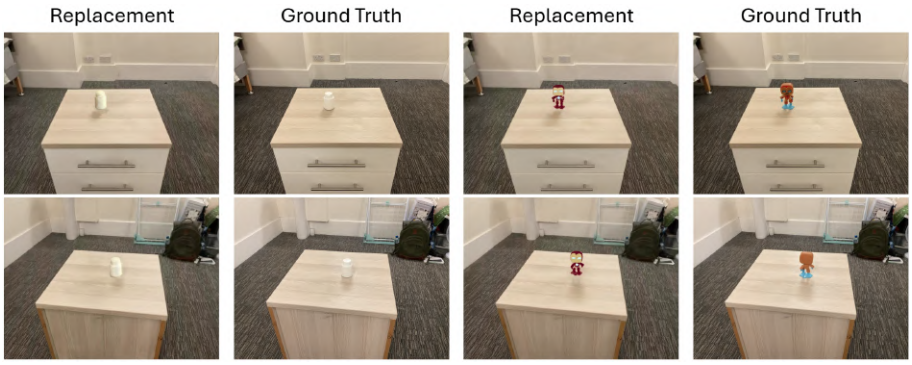

Figure 5.8: Replacement results compared to the ground truth scene with the actual object.

	SSIM		PSNR	
	Capsule	Iron Man	Capsule	Iron Man
0◦	0.5681	0.5603	21.35	20.70
180°	0.5343	0.4895	20.78	20.20

Table 5.5: SSIM and PSNR values of repositioning compared to the ground truth scene with the actual object.

The SSIM and PSNR table for replacement has been added to the Appendix, in Tab. A.1

5.4 Further Qualitative Evaluation

Fig. 5.9 introduces a simple case of object removal, whereas Fig. 5.10, 5.11 and 5.12 introduce more complex scenes with greater level of occlusions.

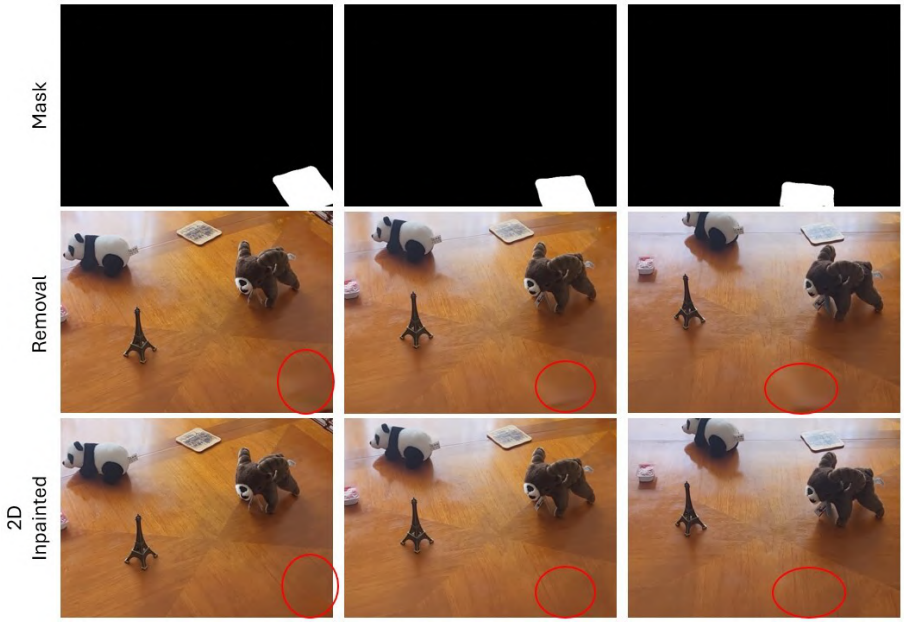

Figure 5.9: A simple inpainting case, where Lama performs extremely well.

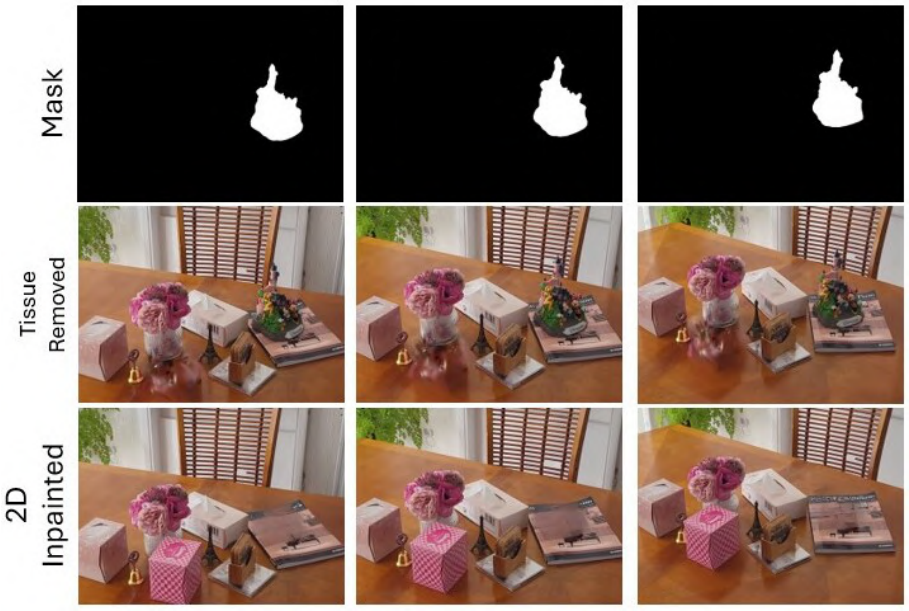

Figure 5.10: The castle has been inpainted in this scenario. While the results are acceptable, the cover of the book is not well predicted.

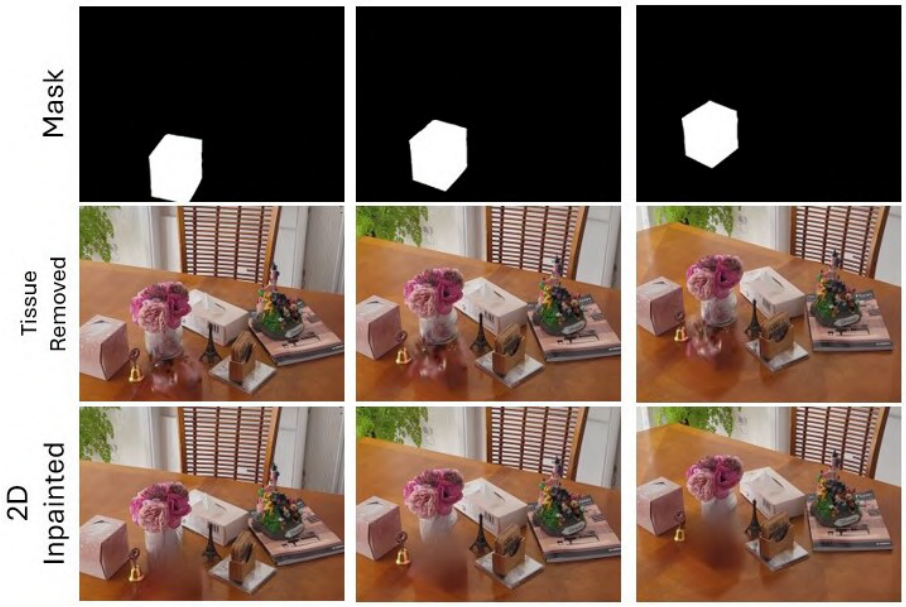

Figure 5.11: The tissue inpainting case shows that Lama performs poorly. The inpainted region doesn not show clear separation between the vase and wooden table texture.

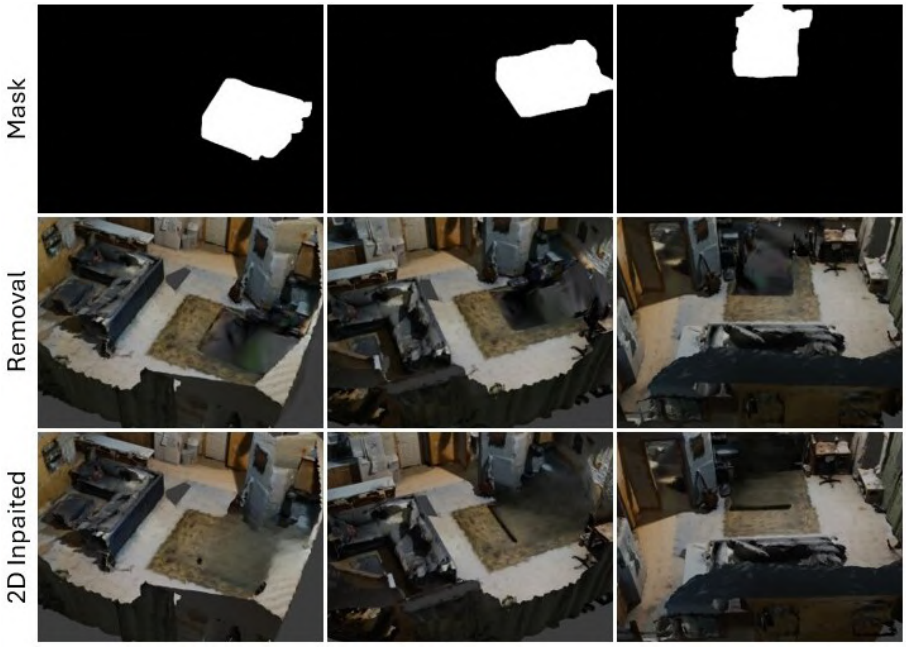

Figure 5.12: A mesh rendering inpainting case.

5

6

Reflections

Contents

6.1 Communication

Competency: *The student can communicate effectively on complex engineering matters with technical and non-technical audiences, evaluating the effectiveness of the methods used.*

To ensure that the project can be effectively communicated to both technical and non-technical audiences, multiple forms of communication will be developed, including a project report, demonstration and code repository.

For non-technical audiences, the project demonstration will include a brief introduction, covering the purpose and motivations behind the project, as well as the main results. This is an effective way of communicating to non-technical audiences because of two reasons: firstly, some of the results of this project, such as comparisons of the original and the final image, where an object has been moved, are very intuitive and easy to appreciate for anyone, and secondly, the demonstration will contain slightly less technical explanation.

For technical audiences, the demonstration can still be great way to gain technical understanding, by asking questions after the presentation. In addition, extra implementation, testing and

background information can be obtained through the full report. If technical audiences want to go a step further by trying out the pipeline to replicate results, the code repository will offer comprehensive instructions on setting up the pipeline.

6.2 Environmental and Social impact

Comptency: *The student can evaluate the environmental and societal impact of solutions to complex problems (to include the entire life-cycle of a product or process) and minimise adverse impacts.*

The main environmental impact of this project is the energy consumption from training and inference using GPUs. The CO_2 emission calculator proposed in [57] will be used to estimate the CO_2 emitted due to energy production. We will consider this separately for the process of completing the project, as well as the process of a potential user using the proposed pipeline. Each run of the pipeline takes approximately 3 hours of GPU usage. Throughout the process of completing the project, the pipeline was run approximately 20 times, leading to a total of 60 hours of GPU usage. An L40S GPU was used. Hence:

$$72\,\mathrm{W} \times 60\,\mathrm{h} = 4.32\,\mathrm{kWh} \times 0.432\,\frac{\mathrm{kg\ CO_2}}{\mathrm{kWh}} = 1.87\,\mathrm{kg\ CO_2} \tag{6.1}$$

$$62\,\mathrm{W} \times 3\,\mathrm{h} = 0.22\,\mathrm{kWh} \times 0.432\,\frac{\mathrm{kg\ CO_2}}{\mathrm{kWh}} = 0.1\,\mathrm{kg\ CO_2} \tag{6.2}$$

To minimize the project's environmental footprint, an energy-efficient L40S GPU was selected, and each inference task was preceded by a review of its expected benefit to ensure only necessary computations were performed. Notably, reducing the number of images captured in the original scene led to a measurable decrease in training time. Future work could investigate the smallest image set required to maintain acceptable performance, and hence reduce training time and environmental footprint.

One societal concern related to this project is the potential misuse of the developed techniques to create deepfake point clouds, where objects or scenes could be manipulated in ways that mislead or deceive users. Such misuse could have serious implications in fields like forensics, autonomous systems, or architectural visualization. To mitigate this risk, the work will clearly document its intended purpose and limitations and encourage transparency when applying the methods.

Conclusions and future directions

When evaluating the overall success of the project, the extent to which the functional and non-functional objectives were achieved should be considered.

Functional Requirements From the introduction we specified the following functional requirements:

1. Explore methods of removing all the points belonging to an object within a point cloud.
2. Explore ways of covering up the exposed region left after the removal in a realistic manner.
3. Explore methods of inserting the removed object back in a new position.
4. Combining the systems systems produced in the first three requirements to build an overall pipeline for object repositioning in point clouds.

It is easy to evaluate the degree of success of these modules, as there exists a ground truth to be compared with in both qualitative and quantitative evaluation.

Object removal and inpainting were evaluated together in the object removal part of evaluation, and, for simple cases, qualatatively, the results looked almost identical from a real world removal, and quantitatively scores similarly showed to be good. However, complex evaluation cases with more object occlusions showed poorer results. Due to not having an accurate object mask, and the limitations of the inpainting module, while the objectg was clearly removed, the inpainted cover up looked unrealistic.

Object insertion also showed success for simple cases, where, while it could be determined the added object was computationally added, and not real, the shape and style followed closely with the intended object to be added, resulting in the evaluation performing well quantatively. However, when complex objects were added, while the object could still be seen to be the original object, finer details of the object were often lost or misrepresented.

Having tested each stage separately, and also testing the entinre pipeline's performance, it could be seen that no extra error was added due to combining the modules. Hence, for the fourth

requirement, the overall pipeline worked well in cases where removal and insertion worked well. The errors within each stage accumulated for the final pipeline.

Despite the pipeline showing poorer results in more complicated scenes, the functionality of the project can be considered successful. This is due to the novel nature of the project: no projects has achieved the repositioning of objects to a good degree before. Furthermore, the limitations within each stage are limitations of the state-of-the-art. The objective was to explore the most promising methods of achieving each module, and, if the result of a module was not ideal, several methods were explained and the module chosen was justified over other methods.

Non-Functional Requirements and Motivation In the motivation, it was mentioned that the project aimed to create a general purpose 3D scene editing tool, that is fit for tasks such as 3D data augmentation, and generating hypothetical scenes. This has now been fulfilled, as our expeirments show scene edits completed by the pipeline, perhaps with slightly poorer results in complex scenes. The motivation and the non-functional reuqirements are closely linked. The non functional requirements include:

1. Usability
2. Accessibility
3. Speed

These non-functional requirements are key to the motivation of creating a general purpose 3D scene editing tool. Usability allows for a better user experience, and allows the user to fully actualise their motivations when using the pipeline, a tool should not take too long to use making speed important, and the tool should be accessible average users, so should not require overly excessive compute reuqirements.

Usability: Compared to black-box tools, there isn't much uncertainty to the process. The user can select which object is to be removed and the new position of the object. The user even has supervision of the style of the object to be insered: a preview of the appearance of the object can be quicly generated before committing to it. This gives the user huge flexibility in how they want their scene to be edited. Making the process automated would be even better for user experience.

Speed: With the recommended input frames and resolution, the removal process takes around 70 minutes, and insertion 50 minutes. The overall time taken to edit the scene is slightly long

compared to general purpose editing tools in 2D, but this is the due to having to train a both Gaussian Splat and NeRF representations of the scene.

Compute: the system can run on XYZ.

Some of the limitations in terms of non-functional requirements will be addressed in the next section.

Future Work

1. While Hypothetical Reasoning was one of the primary motivations of the project, due to limited time, the ScanNet scenes modifications performed in ScanNet were not carried out using our pipeline. It would have been interesting to evaluate the results of VLMs using using the outputs of our pipeline compared to the original results of Hypo3D.
2. For a 3D scene editing tool to be general purpose, the technical details need to be automated, with the user only describing the change they want to make. A way of making the system fully automated would be to have the user specify a text prompt of the scene edit they want to make, and this and the multi-view image seuqence input being the only things the user needs to input for the whole pipeline to function. Hence, the following parts of the pipeline need automation: selection of the object to be removed, choice of inpainting method, object and training camera placements in insertion, diffusion settings etc. Visual grounding could be used to make the 2D segmentation semantic for removal.
3. One of the limitations of the project is currently that the inserted object does not maintain the same style as the original object. Perhaps the SIGNeRF pipeline could be updated so that the diffusion process is conditioned on images of the original object.
4. As seen in the Iron Man insertion example in Fig. 5.7, Iron Man has his facial features on the back of his head as well. This is due to front and back of a figure having the same depth profile. Future work could consider addressing this issue by allowing different text prompts for different angles of generation.
5. Currently, in the object insertion pipeline, the inserted object is made consistent with the scene surroundings, for example, shadows will be cast on the object due to scene lighting. This is due to inpainting with the diffusion model. However, the effect of the the inserted object on the surrounding scene has not been considered. This could be an area of futre research in field of graphics.

A

Appendix

	SSIM		PSNR	
	Capsule	Iron Man	Capsule	Iron Man
0◦	0.5594	0.5578	21.32	20.73
180°	0.4667	0.4875	20.65	20.25

Table A.1: SSIM and PSNR values of replacement compared to the ground truth scene with the actual object.

A

Bibliography

[1] A. Wehr and U. Lohr, "Airborne laser scanning—an introduction and overview," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 54, no. 2, pp. 68–82, 1999, ISSN: 0924-2716. DOI: https://doi.org/10.1016/S0924-2716(99)00011-8. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0924271699000118.

[2] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International Journal of Computer Vision*, vol. 47, no. 1-3, pp. 7–42, 2002.

[3] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. New York, NY, USA: Cambridge University Press, 2003, ISBN: 0521540518.

[4] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner, "Scannet: Richly-annotated 3d reconstructions of indoor scenes," in *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[6] T.-Y. Lin, M. Maire, S. Belongie, *et al.*, *Microsoft coco: Common objects in context*, 2015. arXiv: 1405.0312 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1405.0312.

[7] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, 2015. arXiv: 1512.03385 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1512.03385.

[8] K. He, G. Gkioxari, P. Dollár, and R. Girshick, *Mask r-cnn*, 2018. arXiv: 1703.06870 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1703.06870.

[9] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, *et al.*, *Generative adversarial networks*, 2014. arXiv: 1406.2661 [stat.ML]. [Online]. Available: https://arxiv.org/abs/1406.2661.

[10] A. Chang, A. Dai, T. Funkhouser, *et al.*, "Matterport3d: Learning from rgb-d data in indoor environments," *International Conference on 3D Vision (3DV)*, 2017.

[11] Y. Mao, W. Luo, J. Jing, A. Qiu, and K. Mikolajczyk, "Hypo3d: Exploring hypothetical reasoning in 3d," *arXiv preprint arXiv:2502.00954*, 2025.

[12] C. Zhu, T. Wang, W. Zhang, J. Pang, and X. Liu, *Llava-3d: A simple yet effective pathway to empowering lmms with 3d-awareness*, 2025. arXiv: 2409.18125 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2409.18125.

[13] Ö. Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger, "3d u-net: Learning dense volumetric segmentation from sparse annotation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2016*, ser. Lecture Notes in Computer Science, vol. 9901, Springer, 2016, pp. 424–432.

[14] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, *Pointnet: Deep learning on point sets for 3d classification and segmentation*, 2017. arXiv: 1612.00593 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1612.00593.

[15] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, *Pointnet++: Deep hierarchical feature learning on point sets in a metric space*, 2017. arXiv: 1706.02413 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1706.02413.

[16] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, *Dynamic graph cnn for learning on point clouds*, 2019. arXiv: 1801.07829 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1801.07829.

[17] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[18] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded up robust features," in *Proceedings of the 9th European Conference on Computer Vision (ECCV)*, ser. Lecture Notes in Computer Science, vol. 3951, Heidelberg, Germany: Springer, 2006, pp. 404–417.

[19] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle adjustment—a modern synthesis," in *Vision Algorithms: Theory and Practice*, ser. Lecture Notes in Computer Science, vol. 1883, Berlin, Heidelberg: Springer, 2000, pp. 298–372.

[20] Y. Furukawa and J. Ponce, "Accurate, dense, and robust multi-view stereopsis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 8, pp. 1362–1376, 2010.

[21] J. L. Schönberger and J.-M. Frahm, "Structure-from-Motion Revisited," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[22] H. Hirschmüller, "Accurate and efficient stereo processing by semi-global matching and mutual information," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005, pp. 807–814. DOI: 10.1109/CVPR.2005.177.

[23] dustr Development Team, *dustr: A library for denoising and outlier removal in point-cloud datasets*, `https://github.com/your-org/dustr`, 2025.

[24] J. Geng, "Structured-light 3d surface imaging: A review," *Optics and Lasers in Engineering*, vol. 49, no. 2, pp. 144–156, 2011.

[25] R. Lange and P. Seitz, "Solid-state time-of-flight range camera," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2001, pp. 354–361. DOI: `10.1109/CVPR.2001.990484`.

[26] R. A. Newcombe, S. Izadi, O. Hilliges, *et al.*, "Kinectfusion: Real-time dense surface mapping and tracking," in *Proceedings of the IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, IEEE, 2011, pp. 127–136. DOI: `10.1109/ISMAR.2011.6092378`.

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

[28] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, *3d gaussian splatting for real-time radiance field rendering*, 2023. arXiv: `2308.04079 [cs.GR]`. [Online]. Available: `https://arxiv.org/abs/2308.04079`.

[29] M. Ye, M. Danelljan, F. Yu, and L. Ke, "Gaussian grouping: Segment and edit anything in 3d scenes," *arXiv preprint arXiv:2312.00732*, 2023.

[30] Y. Chen, Z. Chen, C. Zhang, *et al.*, *Gaussianeditor: Swift and controllable 3d editing with gaussian splatting*, 2023. arXiv: `2311.14521 [cs.CV]`.

[31] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas, *Kpconv: Flexible and deformable convolution for point clouds*, 2019. arXiv: `1904.08889 [cs.CV]`. [Online]. Available: `https://arxiv.org/abs/1904.08889`.

[32] Q. Hu, B. Yang, L. Xie, *et al.*, *Randla-net: Efficient semantic segmentation of large-scale point clouds*, 2020. arXiv: `1911.11236 [cs.CV]`. [Online]. Available: `https://arxiv.org/abs/1911.11236`.

[33] J. Yang, B. Gao, J. Wang, and X. Tong, "3d-recgan: 3d shape reconstruction from a single depth view with adversarial learning," in *Proceedings of the International Conference on Computer Vision Workshops (ICCVW)*, 2017, pp. 1–9.

[34] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *Proceedings of the Symposium on Geometry Processing*, Eurographics Association, 2006, pp. 61–70.

[35] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva, "Point set surfaces," in *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, ACM, 2003, pp. 21–28.

[36] W. Yuan, T. Khot, D. Held, C. Mertz, M. Hebert, and S. Savarese, "Pcn: Point completion network," in *Proceedings of the International Conference on 3D Vision (3DV)*, IEEE, 2018, pp. 728–737.

[37] R. Tchapmi, C. Russell, L. Chang, K. Bousmalis, A. Hertzmann, and K. Fatahalian, "Topnet: A topology adaptive graph neural network for point cloud completion," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 330–339.

[38] Y. Li, M.-M. Cheng, X. Hu, S.-C. Cheung, and J. Jia, "Snowflakenet: Point cloud completion by snowflake point deconvolution with skip-transformer," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 815–824.

[39] J. Jiang, Y. Lv, Z. Zhang, Y. Han, Z. Lin, and C. Davies, "Pmpnet: Point-merge-projection network for 3d point cloud completion," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 915–925.

[40] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger, "Occupancy networks: Learning 3d reconstruction in function space," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 4455–4465.

[41] J. J. Park, P. R. Florence, J. Straub, R. A. Newcombe, and S. Lovegrove, "Deepsdf: Learning continuous signed distance functions for shape representation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 165–174.

[42] G. Liu, F. A. Reda, K. J. Shih, T.-C. Wang, A. Tao, and B. Catanzaro, "Image inpainting for irregular holes using partial convolutions," in *Proceedings of the European Conference on Computer Vision (ECCV)*, Springer, 2018, pp. 85–100.

[43] K. Nazeri, E. Ng, and M. Ebrahimi, "Edgeconnect: Generative image inpainting with adversarial edge learning," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 815–824.

[44] R. Suvorov, Q. Yuan, T. Czekel, and J. Kautz, "Resolution-robust large mask inpainting with fourier convolutions," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 14 705–14 715.

[45] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," in *Proc. NeurIPS*, 2020.

[46] A. Ramesh, M. Pavlov, G. Goh, *et al.*, "Zero-shot text-to-image generation," *arXiv preprint arXiv:2102.12092*, 2021. [Online]. Available: https://arxiv.org/abs/2102.12092.

[47] A. Radford, J. W. Kim, C. Hallacy, *et al.*, "Learning transferable visual models from natural language supervision," *arXiv preprint arXiv:2103.00020*, 2021.

[48] B. F. Labs, *Flux*, https://github.com/black-forest-labs/flux, 2024.

[49] A. Kirillov, E. Mintun, N. Ravi, *et al.*, *Segment anything*, 2023. arXiv: 2304.02643 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2304.02643.

[50] H. K. Cheng, S. W. Oh, B. Price, A. Schwing, and J.-Y. Lee, *Tracking anything with decoupled video segmentation*, 2023. arXiv: 2309.03903 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2309.03903.

[51] R. Suvorov, E. Logacheva, A. Mashikhin, *et al.*, "Resolution-robust large mask inpainting with fourier convolutions," *arXiv preprint arXiv:2109.07161*, 2021.

[52] S. Liu, Z. Zeng, T. Ren, *et al.*, *Grounding dino: Marrying dino with grounded pre-training for open-set object detection*, 2024. arXiv: 2303.05499 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2303.05499.

[53] A. Haque, M. Tancik, A. Efros, A. Holynski, and A. Kanazawa, "Instruct-nerf2nerf: Editing 3d scenes with instructions," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023.

[54] J.-N. Dihlmann, A. Engelhardt, and H. Lensch, *Signerf: Scene integrated generation for neural radiance fields*, 2024. arXiv: 2401.01647 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2401.01647.

[55] S. Zhang, W. Bao, Z. Wang, *et al.*, "Controlnet: Adding conditional control to text-to-image diffusion models," in *Proc. CVPR*, 2023.

[56] Y.-C. Guo, *Instant neural surface reconstruction*, https://github.com/bennyguo/instant-nsr-pl, 2022.

[57] A. Lacoste, A. Luccioni, V. Schmidt, and T. Dandres, "Quantifying the carbon emissions of machine learning," *arXiv preprint arXiv:1910.09700*, 2019.