Visualization Research Center (VISUS)

University of Stuttgart
Allmandring 19
D–70569 Stuttgart

Masterarbeit

A Novel NeRF-based Approach for Extracting Single Objects and Generating Synthetic Training Data for Grasp Prediction Models

Anas Mobasher

Course of Study:	Information Technology (INFOTECH)
Examiner:	Prof. Dr. Daniel Weiskopf
Supervisor:	Dr. Miroslav Gabriel (Bosch Research), Dr. Christoph Schulz, Sebastian Künzel, M.Sc.
Commenced:	February 22, 2023
Completed:	August 22, 2023

Abstract

One of the main challenges in robotic manipulation is to grasp previously unseen objects without prior knowledge. State-of-the-art methods rely on dedicated machine learning models which are trained on RGB-Depth (RGB-D) images and annotated labels to predict grasp poses in unstructured environments and for a wide range of previously unseen objects. Collecting a diverse and labeled dataset, however, can be time-consuming and costly. To overcome these challenges, we propose to use Neural Radiance Fields (NeRF) to generate RGB-D images and to combine these with a cutting-edge automatic-labeling approach to create data for training grasp prediction networks.

The main contribution of this thesis is a novel method for obtaining individual NeRFs for objects of interest and backgrounds. The method requires two input scenes: a complete scene containing an object of interest and the same scene but without the object. The steps of the method include training a NeRF on the background scene, aligning it with the object scene, combining it with another NeRF to be trained on the object scene, and joint optimization of both NeRFs with depth regularization loss added to NeRF loss. By applying this approach to various datasets, it is possible to create a library of trained object and background NeRFs. Arbitrary combinations of these NeRFs can then be used to generate novel scenes and render synthetic images for training detection networks.

In a comprehensive ablation study, we employ our approach to create four distinct datasets, apply an automatic labeling pipeline to them and use them to train corresponding grasp prediction networks. The results validate the viability of NeRF-generated data for training detection models, showcasing a performance nearly on par with real data. Furthermore, our approach unveils exciting potential for scalability by facilitating the generation of novel data.

Overall, this research advances the field of robotic manipulation by proving the potential of using NeRF-generated synthetic data and novel scenes to train robust grasp prediction models for real-world applications.

Contents

1 Introduction

1.1 Motivation

The precise manipulation and placement of objects play a crucial role in various domains, including factories, warehouses, industrial settings, and logistics operations. In factory or assembly settings, it is often the same objects that are grasped repeatedly, and this can be achieved by directly teaching the specific objects and their grasp points. However, when dealing with a wide variety of object types, model-free grasping becomes essential, especially in the logistics domain.

Robotic manipulation involves various tasks, and one of its core components is grasping objects accurately. Developing a reliable grasp model requires several challenges, mainly due to the diverse scenes, objects, and camera types encountered. These challenges introduce biases within the grasp model, restricting its effectiveness to specific environments and the objects it was trained on. These challenges become even more critical when the model is trained using synthetic data, such as simulations or image rendering engines, as this data inherently differs from real-world images.

To address this limitation, recent research has explored the utilization of machine learning techniques to enable model-free grasping prediction for unseen objects and environments. An AI-based grasp model refers to an approach that utilizes artificial intelligence, particularly neural networks, to predict grasp points for objects without relying on explicit models. This innovative approach shows promise in enhancing the versatility and adaptability of robotic grasping in real-world scenarios.

To train an AI-based grasp model capable of handling any object in any environment, a large dataset comprising RGB-D images and labels of pixel-wise grasp success probability is required. This dataset needs to include diverse objects with different configurations and various environmental factors, such as backgrounds and camera types. However, collecting such a large dataset is a time-consuming and labor-intensive process, and labeling such a dataset requires significant effort.

Neural rendering approaches currently have the potential of generating implicit representations of scenes by learning from RGB images of a static scene and corresponding camera positions, resulting in extremely realistic novel views images. However, we believe that by integrating virtual scene editing capabilities via NeRF, the true potential of NeRF approaches may be achieved. We can generate an infinite number of realistically created novel scenes by allowing the addition or modification of objects, adjusting their positions, and changing the surroundings. This ability for virtual scene editing utilizing NeRF offers up new avenues for dramatically upgrading various kinds of detection networks such as segmentation detection models and grasp models, thereby promising future advances. Through the generation of diverse and synthetic scenes, the detection model can be exposed to a wide range of object configurations, environmental variations, and camera types.

1.2 Objective and Contribution

A large dataset containing RGB-D images and pixel-wise grasp success probability labels is essential to train an AI-based grasp model capable of handling unseen objects in any environment. Collecting such a dataset and labeling it requires a huge effort and is time-consuming. Regarding the labeling effort, some recent methods are capable to generate the labels automatically, such as [SGK+23]. However, to obtain correctly labeled data, paired RGB-D images where each sample contains two RGB-D images: one of a scene, including objects of interest, to be labeled and the other of the same scene but without the objects, are required. These paired images must be sufficiently aligned for accurate labeling.

To collect such a large dataset, two approaches can be used. The first involves a stationary camera setup, providing the advantage of using one background image for labeling the dataset, but it is time-consuming as collecting each sample requires a new scene with changes in object orientations and configurations, i.e. one view per scene. The second approach is using a moving camera setup, i.e. wrest mounted camera, which allows for collecting multiple views of one scene, leading to a larger dataset, but it requires special setups for obtaining aligned background images, incurring significant costs. Additionally, variations in scenes can still impose time-consuming efforts.

To tackle these challenges, generating novel scenes using implicit representations, such as NeRF, has been proposed as a solution for the large dataset collection problem which is our main focus. NeRF allows the generation of realistic novel views, which can be leveraged in the training dataset and alleviate the limitations imposed by data collection challenges. By utilizing NeRF-generated synthetic data, we aim to improve the grasp model's performance and enable it to generalize to a broader range of objects and environments.

The primary objective of this thesis is to enhance the performance of the model-free grasp model. To achieve this, we will focus on four key objectives.

Firstly, we demonstrate that training a grasp prediction model using NeRF-generated data is possible and yields promising results. By leveraging the capabilities of NeRF to generate realistic scenes, we can train the grasp model on this data and evaluate its performance.

Secondly, we show that training the grasp model on newly rendered camera intrinsics and novel views using NeRF significantly improves its generalization to different camera types. This approach allows us to expand the robustness of the grasp prediction model to diverse image perspectives.

Thirdly, we develop a method for separating objects and backgrounds within a scene into two implicit representations using distinct NeRFs. This is the main contribution of this thesis. By creating separate implicit representations for objects and backgrounds, we can effectively isolate and manipulate these components individually. Other approaches for creating such representations require object masks or other prior knowledge. Utilizing rendering techniques with different NeRFs, we can generate novel synthetic scenes that feature various object configurations and backgrounds.

Extending the automatic labeling method [SGK+23] to work with datasets collected by a moving camera setup is another objective. By obtaining sufficiently aligned synthesized images of empty scenes as references, we ensure accurate labeling for the recorded dataset used for training the grasp prediction model.

Finally, we demonstrate that training the grasp model on this synthetic data, encompassing different camera types and a range of object configurations with varying backgrounds leads to significant improvements in its performance. By exposing the model to diverse and realistic synthetic scenes, we can enhance its ability to generalize and effectively grasp objects in a wide range of real-world scenarios.

The thesis is structured as follows. Chapter 2 will provide an overview of NeRF, showing a comprehensive understanding of its principles and capabilities. Additionally, the chapter will delve into the model-free grasping approach, providing a detailed explanation of its functioning and relevance in the context of robotic manipulation. Furthermore, the chapter will introduce the labeling algorithm employed for labeling the grasp model training dataset. Moving on to chapter 3, our proposed method for effectively separating objects and backgrounds within a scene using distinct NeRFs will be presented. This chapter will outline the technique developed to achieve this separation, highlighting the advantages and applications of this novel approach. In chapter 4, we will delve into the experimental setup and methodology used to evaluate our proposed method. The chapter will provide an in-depth explanation of the conducted experiments, including the dataset used, the evaluation metrics employed, and any necessary implementation details. Furthermore, the chapter will present the results obtained from these experiments and offer insights into the effectiveness and performance of our proposed method. In the last chapter, a brief conclusion of our work and the results will be provided, and future work will be presented. In general, this thesis aims to provide a comprehensive understanding of NeRF, model-free grasping, the proposed object-background separation method, and the experimental evaluation of our proposed method.

2 Background

This chapter provides a comprehensive understanding of NeRF and model-free grasping. It is split into three sections.

In the first section, we will present an in-depth overview of NeRF, covering its fundamental principles and discussing its capabilities. We will explain the underlying concepts of NeRF, including its representation of 3D scenes, coordinate sampling, and the process of volume rendering. Additionally, we will delve into the training process of NeRF. This section aims to provide readers with a solid foundation in understanding NeRF and its applications in computer vision and graphics.

The second section discusses approaches for model-free grasping, corresponding dataset labeling, and its relevance in the context of robotic manipulation. We will explain the functioning of model-free grasping, which enables grasping objects without relying on explicit models. Furthermore, we will provide a detailed explanation of the labeling algorithm employed for generating the grasp model training dataset. By addressing these aspects, we aim to provide readers with insights into the training process of the grasp model and the considerations necessary to create a labeled training dataset.

In the third section, we provide an overview of recent studies that concentrate on using NeRFs to create synthetic scenes. Furthermore, we delve into recent research on model-free grasping that revolves around predicting grasp poses without depending on explicit object models.

By dividing the chapter into these three sections, we aim to provide a comprehensive understanding of both NeRF and the model-free grasping approach and give an overview of the recent studies in these fields. This will enable readers to get the fundamental concepts and methods necessary for subsequent chapters and the overall understanding of the thesis.

2.1 Neural Radiance Fields (NeRF)

NeRF was initially introduced by Mildenhall et al. [MST+21] in 2020 as an innovative method for generating novel views of a scene. It has emerged as a major development in the field of computer vision. It uses neural networks to provide a powerful and novel technique for scene representation and the synthesis of novel points of view. NeRF models have been widely utilized in a variety of disciplines, including robotics, virtual reality, and augmented reality. The ability of NeRFs to learn implicit representations of scenes using neural networks provides its major feature. NeRF is capable of capturing fine details and changes in the appearance and geometry of a static scene from a collection of images. This enables the synthesis of realistic novel views from arbitrary camera positions, even for parts of the scene that have not been directly observed during training. NeRF offers the potential to generate accurate and detailed 3D reconstructions of complex environments.

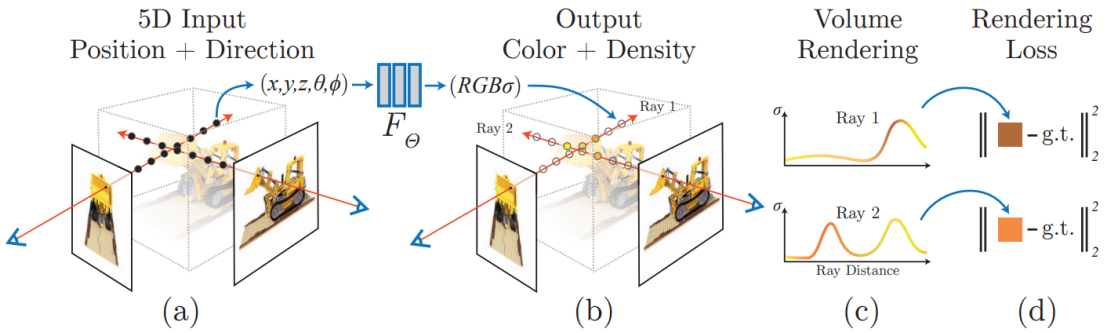


Figure 2.1: An overview of the NeRF training pipeline [MST+21]. (a) The process of selecting sampling points for individual pixels in an image that is to be synthesized. An MLP produces densities and colors at the selected sampling points (b). (c) Using the differentiable rendering function, pixel colors are calculated by integrating the colors and densities obtained from the sampling points along the associated camera rays. (d) The loss between the synthesized pixel colors and the ground truth pixel colors is minimized to train the MLP.

Using a set of RGB images, the target is synthesizing novel views from previously unseen camera poses. Additionally, it models the view-dependent color of occupied regions in the scene, capturing the appearance and visual characteristics. Figure 2.1 presents an overview of the NeRF pipeline. NeRF takes a set of N RGB images, denotes as $\mathcal{I}$, along with their corresponding camera intrinsics and extrinsics, denotes as $\mathcal{T}$, as an input. This set of images should contain only images of a static scene, denoted as $S = \{\mathcal{I}, \mathcal{T}\}$. In case camera intrinsics and extrinsics are not available, they can be estimated using structure-from-motion approaches such as COLMAP [SF16] or DSO [EKC17]. NeRF represents the scene as a radiance field, which is a volumetric density function capturing the shape and structure of the scene. Based on these camera intrinsics and extrinsics, rays are cast to generate the corresponding images. These rays are then sampled and transformed into points, each possessing position and direction attributes. Subsequently, these points are encoded to get embedded coordinates. The encoded position and direction information of the points are fed into a Multilayer Perceptron (MLP), which predicts their respective color and volume density. This information is then employed in volume rendering to reconstruct the estimated pixel color for each ray. Since the entire rendering is differentiable, the MLP can be trained by minimizing the difference between the estimated pixel color and the ground truth color. The subsequent subsections present the NeRF pipeline in more detail.

2.1.1 Radiance Field

A field is defined as a mathematical function that assigns each point in 3D space to a scalar or vector quantity. A radiance field F:

$$F : (\mathbf{x}, \mathbf{d}) \rightarrow (\mathbf{c}, \sigma) \tag{2.1}$$

is a special type of field which maps the position of a point, denoted by the vector $\mathbf{x} = (x, y, z)$, and its viewing directions, represented by the azimuth and elevation, and denoted by the vector $\mathbf{d} = (\theta, \phi)$, to a color $\mathbf{c} = (r, g, b)$ and a volume density σ. A Neural Radiance Field approximates the radiance field using an MLP. In practical applications, the viewing direction is typically expressed using a three-dimensional unit vector, $\mathbf{d} = (d_x, d_y, d_z)$.

The MLP is designed to approximate the radiance field of a scene. To ensure multi-view consistency, the calculation of the volume density is made independent of the input direction by the MLP. However, the color is calculated based on both the input position and direction. The reason for that is σ represents the density or opacity of the scene at a certain point, high sigma values mean solid region and low values mean empty or transparent region, indicating how much light is absorbed or scattered at that point, on the other hand, **c** represents the appearance of the scene at that point from the given viewing direction, which helps in handling transparent and occlusion of the scene making the final rendered image more realistic.

This approach is implemented in a pair of steps. In the first step, the input location **x** is fed to fully-connected layers with ReLU activations, each of which has 256 neurons. This phase generates the volume density σ as well as a high-dimensional 256-dimensional feature vector. In the second step, the feature vector is concatenated with the viewing direction **d** and then fed to an extra fully-connected layer. This layer consists of 128 neurons with ReLU activation that output the color **c**. Figure 2.2 illustrates the MLP architecture for a NeRF.

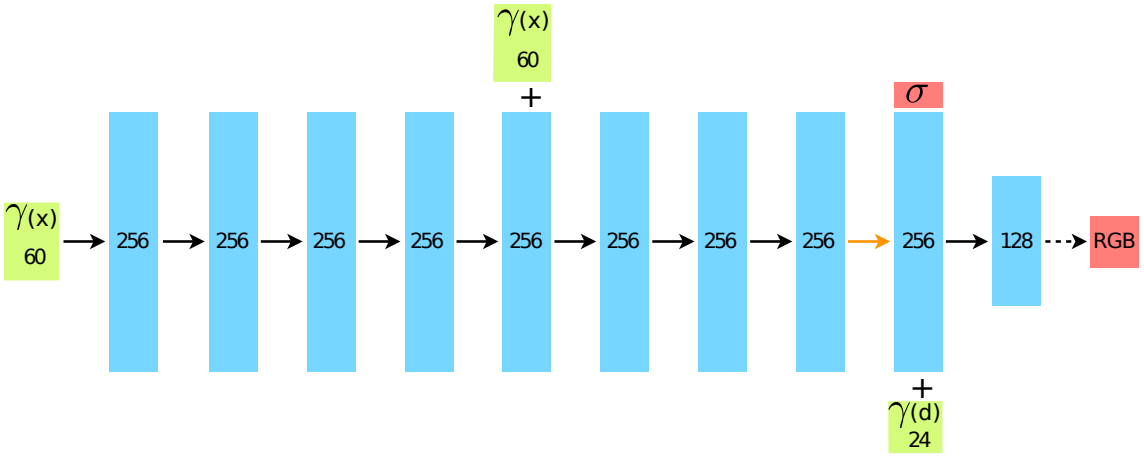


Figure 2.2: The NeRF MLP architecture [MST+21] is a fully-connected design with ReLU activations. The input position $\gamma(x)$ is encoded and passed through a series of eight fully-connected ReLU layers, each consisting of 256 neurons. To introduce a skip connection, the encoded input position $\gamma(x)$ is concatenated with the activation from the fifth layer. Following the eight layers, an additional layer outputs the volume density σ, which is rectified using a ReLU activation to ensure only positive values. This layer generates a 256-dimensional feature vector. The positional encoding of the input viewing direction is then concatenated with this feature vector, and the concatenated data is passed through another fully-connected ReLU layer, featuring 128 neurons. The final layer contains sigmoid activation that outputs the emitted RGB radiance at the specified position **x**, as viewed by the ray with direction **d**.

2.1.2 Volume Rendering

A classical volume rendering [KV84] is used to render the color of a ray from the outputs of the radiance field. The color of a ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$, where $\mathbf{o}$ is the camera position and $\mathbf{d}$ is the camera viewing direction, can be calculated by

$$C(\mathbf{r}) = \int_{t_0}^{t_1} T(t) \cdot \sigma(\mathbf{r}(t)) \cdot \mathbf{c}(\mathbf{r}(t), \mathbf{d}) \cdot dt, \tag{2.2}$$

where t_0 and t_1 are the near and far bounds of the ray, dt represents the infinitesimal distance covered by the ray during each integration step, σ is the volume density, and $\mathbf{c}$ is the RGB color. The accumulated transmittance $T(t)$ represents the probability of the ray traveling from t_0 to t without hitting any object and is given by

$$T(t) = \exp(-\int_{t_0}^{t} \sigma(\mathbf{r}(v)) \cdot dv). \tag{2.3}$$

The cumulative transmittance is used for calculating the ray's predicted depth. This involves accumulating the transmittance values along the path of the ray. We can calculate the predicted depth or distance traveled by the ray within the scene as calculated in *Vanilla NeRF* [YFB+21] by adding up the transmittance values encountered along the ray's path. The estimated depth is

$$D(\mathbf{r}) = \int_{t_0}^{t_1} T(t) \cdot \sigma(\mathbf{r}(t)) \cdot t \cdot dt. \tag{2.4}$$

To then render images for novel views, the rays of the corresponding pixels are traced in a similar way. The interval $[t_0, t_1]$ is divided into N evenly-spaced bins. Subsequently, one sample uniformly is selected from within each bin. This approach ensures that the samples are distributed evenly across the interval and reduces potential bias in the sampling process. These samples are used to estimate $C(\mathbf{r})$, so the approximated color $\hat{C}(\mathbf{r})$ will be

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i \alpha_i \mathbf{c}_i, \text{ where } T_i = \exp(-\sum_{j=1}^{i-1} \sigma_j \delta_j), \tag{2.5}$$

where the variable α_i represents the transparency obtained through traditional alpha composition and is calculated by $\alpha_i = 1 - \exp(-\sigma_i \delta_i)$. δ_i corresponds to the distance between two samples i and $i + 1$. By using this formulation, the transparency of each sample is determined based on the volume density and the distance traveled between consecutive samples.

Similar to equation 2.5, the expected depth can be approximated as follows

$$\hat{D}(\mathbf{r}) = \sum_{i=1}^{N} T_i \alpha_i t_i. \tag{2.6}$$

An alternative approach for depth calculation is introduced in [IAKG21]. It is a transparency-aware method and it sets the depth to the distance until the first sample, which has σ above a threshold h, along a ray. We will use this approach for depth calculation with $h = 20$. As Vanilla NeRF depth estimation might yield inaccurate depth values for transparent or semitransparent objects.

To enhance the modeling of high-resolution and complex scenes, two key improvements are implemented: positional encoding of the input coordinates and hierarchical sampling. These two improvements will be explained in more detail in the following sections.

2.1.3 Positional Encoding

Empirical observations indicate that directly applying the NeRF MLP to the input position vector $\mathbf{x}$, and the direction vector $\mathbf{d}$, results in suboptimal renderings. The MLP, by default, tends to learn low-frequency representations and struggles to effectively capture high-frequency variations in color and geometry. This phenomenon has been substantiated by [RBA+19], which demonstrates that deep networks tend to learn lower-frequency functions. Furthermore, the study highlights that encoding the inputs into a higher-dimensional space using high-frequency functions prior to feeding them into the network improves the model's ability to fit data with high-frequency variations.

Positional encoding of the input coordinates is implemented to assist the MLP in capturing high-frequency functions and representing fine-grained details in the scene. This encoding maps the input coordinates to a higher-dimensional space, enabling the network to better capture and model high-frequency variations in the scene's geometry and color. By incorporating positional encoding, NeRF becomes more effective in representing complex scenes with intricate structures and fine details.

To leverage that in the context of NeRF, the input $\mathbf{x}$ and $\mathbf{d}$ are encoded by applying a positional encoding function γ to each component of $\mathbf{x}$ and $\mathbf{d}$. $\gamma(\cdot)$ maps the input from the real numbers $\mathbb{R}$ to a higher-dimensional space, specifically $\mathbb{R}^{2P}$, where P represents the dimensionality of the encoding. $\gamma(\cdot)$ is given by

$$\gamma(x) = \left(\sin\left(2^0\pi x\right), \cos\left(2^0\pi x\right), \cdots, \sin\left(2^{P-1}\pi x\right), \cos\left(2^{P-1}\pi x\right)\right). \tag{2.7}$$

Based on the experiments conducted by [MST+21]. It was observed that using $P = 10$ for the position vector $\mathbf{x}$ and $P = 4$ for the direction vector $\mathbf{d}$ yields the best results and helps to improve the model's ability to capture and represent high-frequency variations.

2.1.4 Hierarchical volume sampling

To enhance the efficiency of rendering and capturing high-frequency details in the scene, a hierarchical sampling procedure is introduced. This method addresses the inefficiencies of traditional uniform sampling along rays, which may sample unnecessary regions with minimal contribution to the final rendered image. Hierarchical sampling optimizes the allocation of samples by focusing on regions that contain relevant visual content, resulting in improved rendering efficiency while preserving important details.

Uniform sampling points along rays can be inefficient, particularly in regions like free space that have little impact on the final color rendering. Conversely, sampling too few points in occluded regions can lead to lower-quality renderings. To overcome these challenges, researchers have proposed hierarchical representations, drawing inspiration from previous work like Levoy et al. [Lev90].

In the hierarchical sampling approach in the context of NeRF, two MLPs are utilized: one fine MLP, F_{Θ_f}, and one coarse MLP, F_{Θ_c}, where Θ_f and Θ_c are the trainable parameters of the fine and coarse MLP, respectively. Initially, N_c points are sampled using stratified sampling and passed through the coarse MLP. This provides a coarse understanding of the scene, which is then used to generate additional samples near the surface of the scene as described in the following.

To implement this approach, the alpha-composited color from the coarse MLP, denoted as $\hat{C}_c(r)$ in Equation 2.5, can be expressed as a weighted sum of all sampled colors $\mathbf{c}_i$ along the ray:

$$\hat{C}_c(\mathbf{r}) = \sum_{i=1}^{N_c} w_i \mathbf{c}_i, \tag{2.8}$$

where the weights w_i are the depth distribution and calculated as $w_i = T_i \alpha_i = T_i(1 - exp(-\sigma_i \delta_i))$.

Since the weights along a ray represent the depth distribution i.e. represent the occluded regions along the ray, so normalizing these weights as $\hat{w}_i = w_i / \sum_{j=1}^{N_c}(w_j)$, a piecewise-constant probability density function (PDF) is obtained along the ray which indicates occluded regions along the ray. This PDF guides the hierarchical sampling process, determining the distribution of additional samples to be taken along the ray. Regions with higher weights (indicating greater importance) will have more samples allocated to them, allowing for a more accurate representation of the scene and improving the rendering quality.

To refine the sampling strategy, we proceed with a second set of samples from the computed PDF along the ray. These additional samples, denoted as N_f, are obtained using the inversion sampling method. This method involves calculating the cumulative distribution function (CDF) from the PDF then generating uniform random variables between 0 and 1 and applying these values to the inverse of the CDF. By evaluating the fine MLP at the combined set of samples from both the coarse and fine networks, we obtain a more comprehensive understanding of the scene.

Using Equation 2.5, we compute the final rendered color of the ray, denoted as $\hat{C}_f(r)$, by incorporating all $N_c + N_f$ samples. This approach ensures that more samples are allocated to regions expected to contain visible content. The allocation of additional samples to these regions helps improve the accuracy and quality of the final rendered image.

There are alternative approaches that do not rely on hierarchical sampling. In the case of depth-supervised NeRF [DLZR22], depth information is utilized to guide the sampling strategy, particularly by placing more samples in regions closer to the scene. This approach aims to capture the geometry of the scene accurately by focusing sampling points on areas of higher importance.

Unlike the hierarchical sampling approach, depth-supervised NeRF employs a single MLP instead of using separate coarse and fine networks. By incorporating depth information into the sampling process, the model can prioritize regions based on their distance from the camera, allowing for a more effective allocation of samples.

This depth-guided sampling technique offers an alternative to hierarchical sampling and can provide high-quality renderings by emphasizing samples in areas that contribute significantly to the scene's geometry. Each approach has its advantages and considerations, and the choice between them depends on the specific requirements and characteristics of the rendering task at hand.

2.1.5 Rendering Loss

After rendering the color of each ray using the samples from both the coarse and fine sets, see Equation 2.5, a squared error between the rendered pixel colors and the true pixel colors is applied:

$$\mathcal{L}_c(\mathcal{I}, \mathcal{R}, \Theta) = \sum_{\mathbf{r}\in\mathcal{R}} \left[\left\| \hat{C}_c(\mathbf{r}, \Theta_c) - C_{g.t}(\mathbf{r}) \right\|_2^2 + \left\| \hat{C}_f(\mathbf{r}, \Theta_f) - C_{g.t}(\mathbf{r}) \right\|_2^2 \right], \tag{2.9}$$

where $\mathcal{R}$ is a batch of rays calculated from camera poses of scene S, $C_{g.t}(\mathbf{r})$ is the corresponding ground truth pixel color sampled from $\mathcal{I}$, $\hat{C}_c(\mathbf{r})$ is the color rendered using coarse MLP, $\hat{C}_f(\mathbf{r})$ is the color rendered using fine MLP.

By comparing the rendered colors to the ground truth, the loss function provides a measure of how well the neural network approximates the true scene. The goal is to minimize this squared error loss, which encourages the network to produce more accurate and visually appealing renderings. So we minimize the color loss by optimizing the NeRF MLP parameters Θ into corresponding images $\mathcal{I}$ and the corresponding view rays $\mathcal{R}$:

$$\Theta^* = \arg\min_{\Theta} \mathcal{L}_c(\mathcal{I}, \mathcal{R}, \Theta).$$

In regression problems, including image rendering it is common to use the squared error loss as it effectively penalizes larger deviations and allows for differentiable optimization. By minimizing this loss the network can learn to produce renderings that closely resemble the desired colors resulting in enhanced rendering quality.

2.2 Robotic grasping

Fully automating picking applications in logistical applications is a challenge, especially when handling a variety of unseen items, in different environments. For example, in warehouses, a typical pick-and-place system involves a robot equipped with a suction gripper, bins containing various goods provided by a conveyor belt, and an RGB-D camera positioned above the scene. Recently machine learning techniques have been developed to predict objects grasping without the need for teaching the objects and their grasp points via a CAD file. These techniques, such as [[SGK+23] [KBKH20] [NGC+23]], have shown results in effectively managing different types of unseen items in various environments.

2.2.1 Model-Free Graspablility Prediction

To accomplish the objective of predicting grasping positions for objects in the scene a two-step approach is applied.

In the first step, we estimate a grasping quality, denoted as $\hat{Q}$, at each pixel location in the scene. We take an input image C that has both RGB and depth information and has dimensions $\mathbb{R}^{h \times w \times 4}$, where h and w represent the height and width of the image respectively. The target is to estimate how good a grasp would be at each location. The output $\hat{Q}$ is a map with dimensions $\mathbb{R}^{h \times w}$. Each pixel in this map represents a quality value that indicates the probability of successfully grasping an object at that location.

In the second step, we apply post-processing techniques to obtain a collection of possible grasps represented as $g = (\vec{x}, \vec{v})$. Each possible grasp consists of a 3D position denoted by $\vec{x}$ and an orientation represented by $\vec{v}$. By examining the grasp quality map Q and considering certain criteria like setting a threshold for grasp quality, we can identify regions or points in the scene where successful grasps are likely. These regions or points correspond to grasps. By extracting information about their position and orientation, we obtain a set of feasible grasps g.

By following this two-step process and obtaining the feasible grasps (g), we can effectively predict and identify grasp poses that are likely to succeed in manipulating objects.

2.2.2 Grasp-Quality Prediction

To obtain $\hat{Q}$, a U-Net [RFB15] architecture with a ResNet-34 [HZRS16] is used. The input to the network is a 3-channel image I. The first channel, denoted as I_{Grey}, represents the grayscale image of the RGB channels in C. The second channel, denoted as I_{Depth}, contains the depth information. Lastly, the third channel, denoted as I_{STD}, represents the standard deviation of the surface normals. There are some preprocessing applied to the raw data which is C and I_{Depth} in order to obtain the channels of the network input.

The decision to use a single grayscale channel image I_{Grey} instead of a three-channel RGB image in the input image is motivated by several factors. First, it allows the network to retain important texture information while reducing the number of input channels. This can be beneficial in scenarios where texture plays a significant role in determining grasp quality. Additionally, using a grayscale channel helps prevent the network from overfitting to specific color patterns or backgrounds, ensuring more robust and generalizable grasp predictions. The inclusion of the standard deviation of surface normals I_{STD} as a third channel in the input image is motivated by the specific requirements of the application. Suction grippers, in particular, rely heavily on the local surface structure for successful grasping. On irregular surfaces with a high variation of surface normals, it is challenging to create a sealed vacuum with a suction cup. By including I_{STD} as an input channel, the network can take the local surface structure into account and enhance its grasp prediction for suction grippers. Including I_{Depth} as an input channel to the network offers multiple benefits. One of the main advantages is its ability to provide structural information about the scene, while I_{Grey} mainly captures appearance and texture details. Additionally, it allows the network to distinguish between background and objects based on their spatial arrangement. Hence, the network becomes capable of handling occlusions in complex environments. The network can use the depth map to identify object shapes and positions, which ultimately leads to more accurate and robust predictions.

To obtain the input channels, specially I_{STD} and I_{Grey}, required for the network, the RGB-D images undergo preprocessing steps. To calculate I_{Grey}, we use the luminance method. To calculate I_{STD}, the following steps are followed. Initially, I_{Depth} and the camera intrinsic matrix (K) are used to convert I_{Depth} into a point cloud using

$$I_{\text{Pcd}}(u, v) = K^{-1}(I_{\text{Depth}}(u, v)[u, v, 1]^{\text{T}}), \forall (u, v) \in h \times w. \tag{2.10}$$

Subsequently, the surface normal is obtained for each pixel individually. Afterward, for a small neighborhood, the standard deviation of the surface normals is calculated. Finally, the calculated standard deviation is normalized to a range between 0 and 1.

To tackle inaccuracies and undetectable depth regions frequently encountered in RGB-D images from 3D cameras, a pair of pre-processing steps is utilized. Firstly, missing or unreliable depth information areas are addressed by approximating depth values. Secondly, outlier pixels are filtered out to diminish noise and enhance the depth image's overall quality. These pre-processing steps enhance the accuracy and reliability of the depth information, leading to an improvement in I_{STD}.

Moving on to the U-Net, it takes the three-channel input image and performs convolutional operations to extract relevant features. The network's architecture allows for effective learning and representation of the grasp quality information at each pixel. The output of the network is a prediction of the grasp quality map $\hat{Q}$.

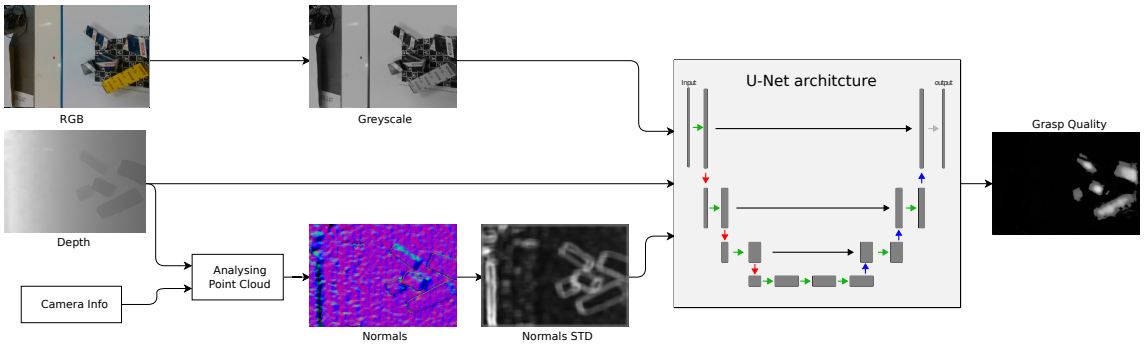


Figure 2.3: Overview of the grasp-quality prediction pipeline, showing examples of the three-channel U-Net input, their corresponding grasp quality, and the pre-processing steps.

Figure 2.3 illustrates the grasp-quality prediction pipeline, showing the pre-processing steps, the three-channel U-Net input, and the corresponding expected grasp-quality prediction.

2.2.3 Label Approximation and Supervised Training

To train a grasp-quality prediction network, a large dataset consisting of RGB-D images and their corresponding pixel-wise grasp quality ground truth, Q, is necessary. However, manually labeling such a vast dataset, which should contain a wide range of objects with various configurations and cluttered scenes, is a time-consuming and labor-intensive task. To address this challenge, the authors of [SGK+23] propose a method for approximating the labels and generating them automatically. By leveraging this label approximation method, we can generate approximate labels, L, for the grasp-quality prediction network training dataset, such that $L \approx Q$. Although these

approximations may not perfectly match Q, they provide a reasonable estimation that can still be useful for training the network. This label approximation method primarily focuses on simple geometries. The underlying concept is that a flatter surface tends to result in a higher grasp success rate. Therefore, the label approximation is based on the negative normalized standard deviation of the surface normals.

To incorporate this idea into the network, the label is calculated as $L_{STD} = 1 - I_{STD}$. This approach is consistent with utilizing of I_{STD} as one of the input channels for the network. Additionally, it is desirable for a grasp to be close to the center of mass of the object, as this promotes grasp stability. To incorporate this property into the label, the pixels on the graspable surface, as indicated by L_{STD}, are clustered. Another component called L_{dist} is introduced, representing the distance of each pixel to its respective cluster center.

One challenge of this method is that it may also detect the background, e.g. the bin, and non-object geometries since it only depends on surfaces. To overcome this, a background image, i.e. an image of the bin without any graspable objects, is recorded. By subtracting the depth image of the background from the depth of the original image, we can mask the non-object pixels of I_{STD} and use only the object pixels M_{ob}.

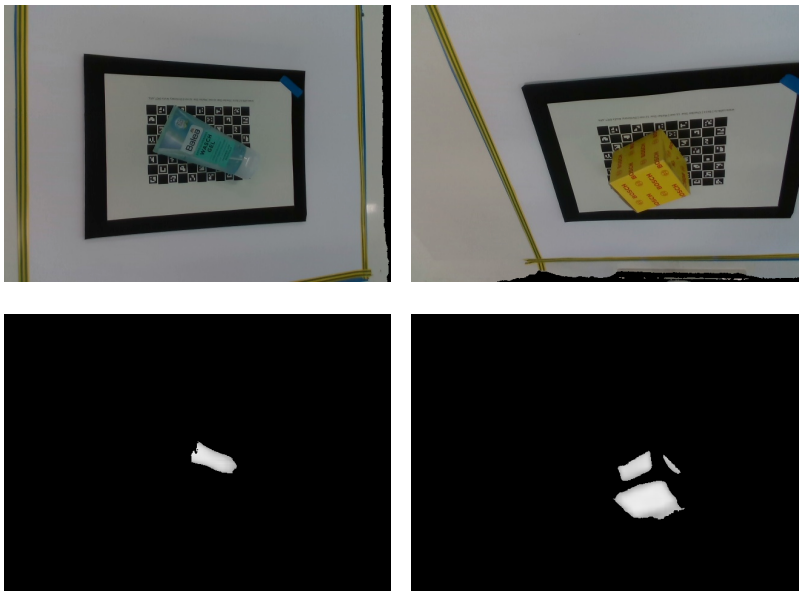


Figure 2.4: Two labeled examples are shown, where the RGB input images are displayed in the top row, and the corresponding approximated labels L are shown in the bottom row. The imperfections in the labels are due to inaccuracies in the depth information.

Figure 2.4 displays two examples of images labeled using our automatic labeling approach. It is essential to note that the labels may not be perfect in some cases due to invalid depth information.

The labels L_{STD} and L_{dist} are weighted by respective weights, w_{STD} and w_{dist}, to balance their influence on the training process.

$$L = \begin{cases} w_{\text{STD}} L_{\text{STD}} + w_{\text{dist}} L_{\text{dist}} & \text{where } M_{ob} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.11}$$

In our experiments, these weights are chosen to be equal to 0.5. Overall, this label approximation method provides a way to generate labels that capture the flatness of the surface and the proximity to the object's center of mass. It takes into account the object pixels while eliminating non-object pixels using a background subtraction technique. However, a critical requirement for this method's success is the availability of aligned background images within the dataset. These background images are essential for accurately defining the background pixels and object pixels, which can be a challenging aspect of the dataset collection process.

The final loss used for training the grasp-quality prediction network is

$$\mathcal{L}_{grasp} = w_{bg} \left\| (1 - M_{ob})(L - \hat{Q}) \right\|_2^2 + w_{ob} \left\| M_{ob}(L - \hat{Q}) \right\|_2^2, \tag{2.12}$$

where w_{bg} and w_{ob} are weights calculated individually for each image. These weights are determined by considering the background and foreground pixels within that particular image. In our experiments, we set w_{bg} to 0.5 and w_{ob} to 0.5.

2.3 Related Work

We divide this section into two subsections, the first subsection gives an overview of the related works to NeRFs for generating synthetic scenes and the second subsection provides some recent research on model-free grasping.

2.3.1 NeRFs for Generating Synthetic Scenes

In the field of synthetic scene rendering, various methods have been developed to support the creation and synthesis of virtual scenes and novel views. Some approaches, such as Broadhurst et al. [BDC01], Bleyer et al. [BRR11], and Seitz et al. [SCD+06], adapt traditional modeling and rendering pipelines to accommodate synthetic scenes and enable novel view synthesis. However, recent works have explored the use of NeRF for flexible scene manipulation, presenting promising advancements in this area.

Lazova et al. [LGO+23] suggest an approach that separates rendering from scene representation, enabling rapid multi-scene training, high-fidelity image synthesis, and manipulation. They show object duplication, removal, rigid and non-rigid transformations, and inter-scene object transfers, among other scene alterations. However, instead of implicit functions, they use voxel-based representations.

Yang et al. [YZX+21] provide a neural scene rendering framework that enables the creation of new synthetic scenes by combining the several NeRFs of different objects. For training, however, they require object masks, and they do not take into account lighting circumstances in generated scenes, such as shadows, and it employs a hybrid space embedding that combines voxelized representation with coordinate-based positional encoding. Another similar approach is [WTB+23]. They manage to train a NeRF representing only the object without the surrounding scene but also require a segmentation mask of the object in one of the images.

Additionally, Guo et al. [GFWF20] demonstrate an approach for combining objects into realistic images of dynamic scenes. Their method decomposes a scene into implicit object functions that are based on view and lighting circumstances, allowing object usage across scenes involving various objects, cameras, and lighting. Inter-object light transport effects are demonstrated through integration with volumetric ray tracing. Nonetheless, their approach is based on synthetic objects rather than real-world ones.

Stelzner et al. [SKK21] proposed an unsupervised object segmentation utilizing the relation between 3D scene geometry and 2D observations. Their usage of NeRFs allows them to parameterize arbitrary geometry and appearances, allowing them to learn without prior knowledge of object structures in the dataset. However, the method is limited to synthetic objects and scenes, lacking real-world scene applicability.

Lastly, Kosiorek et al. [KSZ+21] introduce NeRF-VAE, a geometry-aware scene generative model employing NeRF as a decoder within a Variational Autoencoder (VAE) framework. However, it allocates its capacity to a single scene, limiting per-scene expressivity.

In contrast to these existing methods, our approach avoids voxel-based representations and relies only on coordinate positional encoding and implicit functions. Additionally, it can decompose objects and backgrounds from scenes using two recordings, one recording of a scene with objects and another without objects, without requiring explicit object masks. Our method enables novel view synthesis with multi-object manipulation in generated scenes, offering a distinct advantage over the aforementioned works.

2.3.2 Model-Free Grasping

In recent studies on model-free grasping researchers have been focusing on using deep learning techniques, specifically Convolutional Neural Networks (CNNs) to predict the success of different grasping attempts. These methods aim to train CNN models on datasets so they can quickly and accurately determine the likelihood of a successful grasp based on depth images.

One example is Dex-Net [MLN+17], where the researchers have created a model called Grasp-Quality Convolutional Neural Network (GQ-CNN). This model takes depth images as input. Uses them to predict how likely a grasp will be successful based on the position, angle, and depth of a gripper relative to an RGB-D sensor. By training on datasets, the GQ-CNN can efficiently evaluate various grasping configurations enabling more effective robotic grasping in real-world settings.

In [ZSY+22], An object-agnostic grasping framework is used instead of depending on particular training data for each object encountered. It converts observations into actions by generating probability maps that show how possible various grasping actions are for each pixel. The algorithm then chooses the action that has the best chance of executing the grasp. Furthermore, it leverages a cross-domain image classification framework to identify the specified objects by comparing observed images with product images.

In [KFH+22], NeRFs are utilized to render new RGB-D images, which are then used as inputs to a grasping network. The NeRF training process occurs in real-time, but the rendered image quality is comparatively lower than other NeRF approaches. To address this limitation, the corresponding

grasping network is trained specifically and exclusively on this type of data. However, this specialization constrains its potential applications, making it less adaptable to diverse scenarios and objects.

3 Methodology

Our method aims to learn individual implicit representations of an object and its surrounding background using two NeRFs, and Object-NeRF, F_{ob}, Background-NeRF, F_{bg}, and their corresponding parameters are Θ_{ob} and Θ_{bg}, respectively. Applying our method to different datasets, allows us to combine multiple trained NeRFs during rendering to generate novel realistic synthetic scenes. Two recordings are required to achieve this goal: one recording of the scene without objects, denoted as S_{bg}, and another recording of the scene including objects of interest, denoted as S_{scn}. The scene S_{scn} comprises two sets, denoted as $\mathcal{I}_{scn}, \mathcal{T}_{scn}$, where $\mathcal{I}_{scn}$ is a collection of scene images, and $\mathcal{T}_{scn}$ represents the corresponding camera extrinsics and intrinsics. By utilizing the information from $\mathcal{T}_{scn}$, we cast rays that correspond to the scene images, resulting in a set of rays, denoted as $\mathcal{R}_{scn}$. Similarly, the same process is applied to the background scene S_{bg}, yielding sets $\mathcal{I}_{bg}, \mathcal{T}_{bg}$, and the associated rays, denoted as $\mathcal{R}_{bg}$.

In this chapter, we will first discuss how NeRFs are combined, and then give a detailed explanation of the training pipeline and its individual steps.

3.1 Combining NeRFs

Our approach revolves around the utilization of multiple NeRFs for rendering and generating realistic novel scenes. A synthetic scene is generated using a set of M NeRFs, denoted as $\mathcal{F} = \{F_1, ..., F_m, ..., F_M\}$, and the output densities and colors of each NeRF are σ_m and $\mathbf{c}_m$, respectively. Each NeRF contributes independently its own densities and colors along the ray which travels through the scene and they are combined, similar to the superposition principle [SKK21]. The accumulated transmittance $T(t)$ along the ray is computed by multiplying the transmittances $T_m(t)$ of each NeRF encountered which is determined by the probability of the ray not intersecting any object given by

$$T(t) = \prod_{m=1}^{M} T_m(t) = \exp\left(-\int_{t_0}^{t} \sum_{m=1}^{M} \sigma_m(\mathbf{r}(v))dv\right). \tag{3.1}$$

Equation 3.1 represents the probability of the ray not intersecting any object. The final volumetric density, denoted as $\sigma(x)$, is obtained by summing the densities of each NeRF. $\sigma(x)$ is calculated as follows:

$$\sigma(x) = \sum_{m=1}^{M} \sigma_m(x).$$

Furthermore, the probability that a specific NeRF m is responsible for the light reaching the camera from a given depth t is computed as

$$w(t, m) = \sigma_m(\mathbf{r}(t))T(t),$$

similar to equation 2.2.

The expected color from a set of NeRFs $\hat{C}_{\mathcal{F}}(\mathbf{r})$ is obtained by marginalizing over m and t. This is calculated by integrating the product of the weights $w(t, m)$ and the colors $\mathbf{c}_m(\mathbf{r}(t), \mathbf{d})$ over the depth range. It is calculated as follows:

$$\hat{C}_{\mathcal{F}}(\mathbf{r}) = \int_0^{\infty} \sum_{m=1}^{M} w(t, m)\mathbf{c}_m(\mathbf{r}(t), \mathbf{d})dt. \tag{3.2}$$

Similar to equation 2.5, the final approximated color of combining multiple NeRFs is

$$\hat{C}_{\mathcal{F}}(\mathbf{r}) = \sum_{i=0}^{N}\sum_{m=1}^{M} w_{im}\mathbf{c}_{im} = \sum_{i=0}^{N}\sum_{m=1}^{M} T_i\alpha_{im}\mathbf{c}_{im}, \text{ where } T_i = \exp(-\sum_{j=1}^{i-1}\sum_{m=1}^{M}\sigma_{mj}\delta_j). \tag{3.3}$$

To calculate the weights w for rendering using hierarchical sampling, we marginalize over the components m only, resulting in the depth distribution $w(t)$ given by the sum of the weights $w(t, m)$ over all NeRFs

$$w(t) = \sum_{m=1}^{M} w(t, m).$$

To train a combined NeRF on a scene, the used loss is similar to the loss in 2.12, but the expected color, in this case, is $\hat{C}_{\mathcal{F}}$ instead of $\hat{C}$. The loss is

$$\mathcal{L}_{c_{\mathcal{F}}}(\mathcal{I}, \mathcal{R}, \Theta_{\mathcal{F}}) = \sum_{\mathbf{r}\in\mathcal{R}} \left| \left\| \hat{C}_{\mathcal{F}_c}(\mathbf{r}, \Theta_{\mathcal{F}_c}) - C_{g.t}(\mathbf{r}) \right\|_2^2 + \left\| \hat{C}_{\mathcal{F}_f}(\mathbf{r}, \Theta_{\mathcal{F}_f}) - C_{g.t}(\mathbf{r}) \right\|_2^2 \right|, \tag{3.4}$$

where $\hat{C}_{\mathcal{F}_c}(\mathbf{r})$ is the rendered color by combining the coarse MLPs of $\mathcal{F}$. denoted as $\mathcal{F}_c$, using Equation 3.3, and $\hat{C}_{\mathcal{F}_f}(\mathbf{r})$ is the rendered color by combining the fine MLPs of $\mathcal{F}$, denoted as $\mathcal{F}_f$, using Equation 3.3.

3.2 Object-Background Separation using Distinct NeRFs

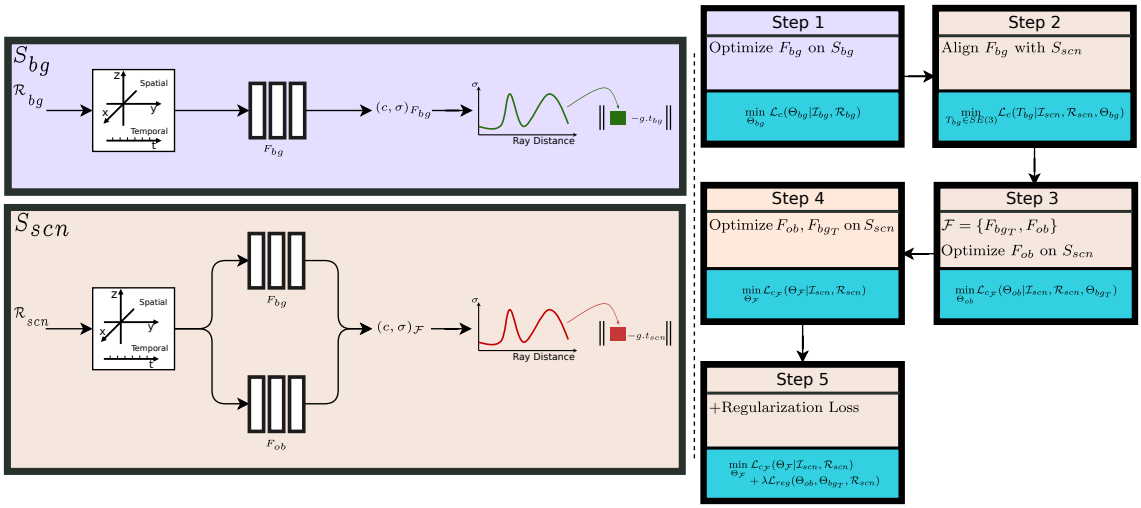
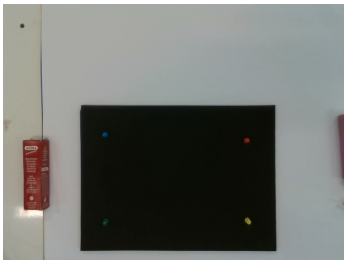


Figure 3.1: Overview of our object-background separation pipeline. Two scenes, a scene includes an object of interest S_{ob} and another scene includes the background only without the object S_{bg}, are given as inputs. The pipeline comprises the following steps: 1) A NeRF, referred to as Background-NeRF F_{bg}, is trained on the S_{bg} scene. 2) The trained F_{bg} is aligned with the scene S_{scn}. 3) The F_{bg} is combined with another NeRF, referred to as Object-NeRF F_{ob}, and optimization only for F_{ob} is conducted on S_{scn}. 4) Both NeRFs, F_{bg} and F_{ob}, are jointly optimized on S_{scn}. 5) The depth regularization loss is added to the color loss while optimizing both NeRFs.

The key idea behind our method is to train F_{bg} on S_{bg}, and then combine F_{bg} with F_{ob}, and train on S_{scn}. By the end of execution of the pipeline, F_{ob} and F_{bg} should represent the object only and the background only of S_{scn}, respectively. Figure 3.1 shows an overview of our pipeline. Our pipeline is divided into five steps:

- The first step involves training F_{bg} on S_{bg}, allowing it to capture the scene's background information.
- In the second step, we align the trained F_{bg} with S_{scn}.
- In the third step, we combine F_{bg} with F_{ob}, fixing the weights of F_{bg}, and training the weights of F_{ob} on S_{scn}. This enables F_{ob} to focus on modeling the object's appearance and shape within the scene.
- In the fourth step, we unfreeze the weights of both NeRFs, Θ_{ob} and Θ_{bg}, and train them jointly on S_{scn}.
- In the last step, we add a depth regularization loss to the final loss to learn F_{ob} on the object only.

In our pipeline, we utilize the same NeRF architecture, that was originally introduced in the Vanilla NeRF, in both NeRFs F_{bg} and F_{ob}.



(a) An image of S_{bg}.

(b) An image of S_{scn}.

Figure 3.2: Sample images of the test dataset.

As our pipeline involves multiple sequential steps and to demonstrate well each step, we will apply our pipeline on a test dataset showing the resulting scene mesh of the trained NeRF of each step. The mesh is generated by the Marching Cubes algorithm [LC98] using $\sigma > 20$. An example of the test dataset is depicted in Figure 3.2, including examples of both input scenes, S_{bg} and S_{scn}.

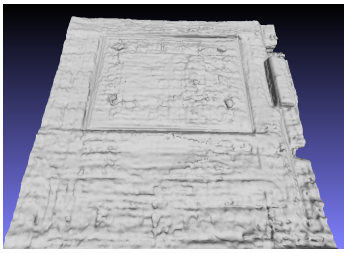
3.2.1 Background-NeRF Training

The initial step in our pipeline focuses on training F_{bg} on S_{bg} for a specified number of training steps s_1, where s_1 is a hyperparameter. This training aims to obtain an implicit representation of the scene without the object. We minimize the NeRF color loss by optimizing Θ_{bg} on S_{bg}. The optimization problem for the first step can be expressed as:

$$\min_{\Theta_{bg}} \mathcal{L}_c(\Theta_{bg} \mid \mathcal{I}_{bg}, \mathcal{R}_{bg}).$$

The objective is to obtain an implicit representation of the background only in order to be able to define the background from other scenes that share the same background.

(a) A rendered image by F_{bg}.



(b) The scene mesh of F_{bg}.

Figure 3.3: Results of the first step of our pipeline.

Figure 3.3 shows an example of an image rendered by F_{bg} and the corresponding scene mesh.

3.2.2 Offset Correction

The camera extrinsics of both scenes, S_{bg} and S_{scn}, are calculated based on two different reference frames, therefore, both scenes are not aligned. Hence, in the second step of our pipeline, we aim to align the trained F_{bg} with the dataset of the scene S_{scn}. Similar to iNeRF [YFB+21], we introduce a global transformation matrix, T_{bg}, that will be used only with F_{bg} to align its implicit scene with S_{scn}.

Unlike traditional NeRF approaches that optimize the MLP on the camera rays and the input pixels, in this step, we focus on optimizing the values of T_{bg}. After sampling the rays in $\mathcal{R}_{scn}$, we obtain samples that each is represented by its position and direction. These samples are transformed using T_{bg} before feeding to the positional encoder of F_{bg}. To achieve alignment, we use the same color loss function $\mathcal{L}_c$ that is used in NeRF. However, only T_{bg} is updated while keeping the parameters Θ_{bg} fixed unlike in standard NeRF training. We optimize T_{bg} to minimize $\mathcal{L}_c$. The optimization problem can be expressed as:

$$\hat{T}_{bg} = \underset{T_{bg} \in SE(3)}{\arg\min}\ \mathcal{L}_c(T_{bg} \mid \mathcal{I}_{scn}, \mathcal{R}_{scn}, \Theta_{bg}).$$

To ensure that the estimated offset correction matrix T_{bg} remains within the SE(3) manifold during optimization, the parameterization of T_{bg} is done using exponential coordinates [LP17]. This ensures that the estimated transformation matrix T_{bg} maintains its validity as a rigid body transformation. The initialization of the estimated global offset correction matrix T^0_{bg} is done by setting it to an initial matrix that lies within the SE(3) manifold, such as the identity matrix. During the optimization process, the offset correction matrix is represented as:

$$T_{bg} = exp([S]\theta)T^0_{bg},$$

$$\text{where } exp([S]\theta) = \begin{bmatrix} exp([\omega]\theta) & K(S,\theta) \\ 0 & 1 \end{bmatrix} \text{ and } S = [\omega, v]^T,$$

where S represents the screw axis with the magnitude θ and the 3×3 skew-symmetric matrix $[\omega]$, $K(S,\theta) = (I\theta + (1 - cos\theta)[\omega] + (\theta - sin\theta)[\omega]^2)v$ and $exp([\omega]\theta) = I + \sin\theta\,[\omega] + (1 - \cos\theta)[\omega]^2$ [LP17].

With the parameterization described, the objective is to solve for the optimal offset correction matrix from T^0_{bg},

$$\hat{S\theta} = \underset{S\theta \in \mathbb{R}^6}{\arg\min}\ \mathcal{L}_c(exp([S]\theta)T^0_{bg} \mid \mathcal{R}_{scn}, \mathcal{I}_{scn}, \Theta_{bg}).$$

We optimize the $\hat{T}_{bg}$ for a specific number of training steps s_2, where s_2 is a hyperparameter, so the alignment process iterates over multiple steps to refine the estimated offset correction matrix.

3.2.3 Object-NeRF Training

Up to this step, we have the two scenes, S_{bg} and S_{scn}, the trained F_{bg}, and the trained correction matrix T_{bg}. The purpose of T_{bg} is to align the implicit scene in F_{bg} with the scene S_{scn}. To simplify our notation, we treat the combination of F_{bg} and T_{bg} as a single entity, denoted as $F_{bg_T} = F_{bg}(T_{bg}(x))$ and its corresponding parameter weights $\Theta_{bg_T} = \{\Theta_{bg}, S\theta\}$. In essence, this signifies that the coordinates of F_{bg} are transformed using T_{bg}. It is important to note that the difference between F_{bg_T} and S_{scn} is ideally the object.

The goal of the subsequent step is to initialize the second NeRF, F_{ob}, and to train it to exclusively represent the object. To achieve this goal, we combine both NeRFs, F_{bg_T} and F_{ob}, into a combined NeRF, denoted as $\mathcal{F} = \{F_{bg_T}, F_{ob}\}$, and its corresponding parameter weights, $\Theta_{\mathcal{F}} = \{\Theta_{bg_T}, \Theta_{ob}\}$. Subsequently, we render pixel colors using the combined NeRF $\mathcal{F}$ and compare it to the target pixel obtained from S_{scn}. During this step, we solely update the weight parameters Θ_{ob} associated with F_{ob} and keep the parameters of F_{bg} and T_{bg} fixed.

Mathematically, the optimization process can be expressed as:

$$\min_{\Theta_{ob}} \mathcal{L}_{c_{\mathcal{F}}}(\Theta_{ob} \mid \mathcal{I}_{scn}, \mathcal{R}_{scn}, \Theta_{bg_T}).$$

By optimizing F_{ob} for a specified number of steps s_3, where s_3 is a hyperparameter, we allow F_{ob} to learn the differences between F_{bg_T} and S_{scn}.

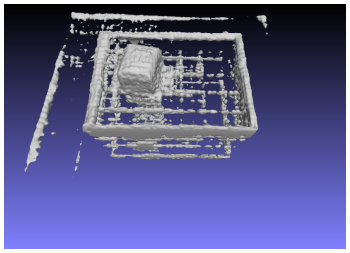


Figure 3.4: The scene mesh of F_{ob} at step 3.

At this point, F_{ob} has learned the differences between F_{bg_T} and S_{scn}, ideally representing only the object. However, the training of T_{bg} may not completely align F_{bg} and S_{bg} due to potential imperfections in F_{bg} and the objects that are in S_{scn} and not in F_{bg}. Consequently, this misalignment can introduce inaccuracies in T_{bg}. As a result, F_{ob} may inadvertently learn not only the object but also undesired portions, such as certain parts of the background. Figure 3.4 illustrate that F_{ob} learned all the differences between F_{bg_T} and S_{scn} including undesired parts of the background.

3.2.4 Joint Optimization

In this step, we aim to optimize both NeRFs, F_{bg_T} and F_{ob}. Since F_{bg_T} has already learned the background and F_{ob} has learned the object along with some background parts, our objective is to enable F_{bg_T} to fully adapt to the background of S_{scn} by incorporating the background parts learned by F_{ob}. On the other hand, F_{ob} should continue learning the object's features while forgetting background elements. Hence, our goal at this stage is to solve the following optimization problem:

$$\min_{\Theta_{\mathcal{F}}} \mathcal{L}_{c_{\mathcal{F}}}(\Theta_{\mathcal{F}} \mid \mathcal{I}_{scn}, \mathcal{R}_{scn}).$$

During this process, we optimize both F_{ob} and F_{bg_T} simultaneously for a specific number of steps s_4, which is an additional hyperparameter.

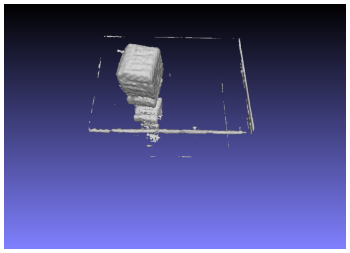


Figure 3.5: The scene mesh of F_{ob} at step 4.

At the end of this step, we achieved sufficient alignment between F_{bg_T} and S_{scn}, capturing all color changes between S_{scn} and S_{bg} in F_{ob}. However, it is possible that these color changes include non-object parts due to various factors such as variations in lighting conditions or minor shifts in non-object regions between the two input scenes recordings, resulting in a very small thickness layer in the implicit scene of F_{ob} directly over the surface of background. Figure 3.5 illustrate that F_{ob} learned all the differences between F_{bg_T} and S_{scn} including small parts of the background due to the mentioned factors.

3.2.5 Regularization Loss

To eliminate these non-object parts from F_{ob}, we introduce an additional regularization term to the loss, $\mathcal{L}_{reg}$, based on the depth, which is reconstructed by F_{ob} and F_{bg_T}:

$$\mathcal{L}_{reg}(\Theta_{ob}, \Theta_{bg_T}, \mathcal{R}_{scn}) = \frac{1}{N_{R_d}} \sum_{b \in R_d} \sigma_{b_{ob}},$$

where $R_d = \{r : \left|D_{ob,bg}(r) - D_{bg}(r)\right| < \epsilon, r \in \mathcal{R}_{scn}\}$, $\sigma_{b_{ob}}$ is the average σ value obtained by F_{ob} along the ray b, N_{R_d} is the total number of rays in R_d, $D_{ob,bg}$ and D_{bg} correspond to the reconstructed depth obtained using approach in [IAKG21] by NeRFs, combined NeRF $\{F_{bg_T}, F_{ob}\}$ and F_{bg_T}, respectively. ϵ is a hyperparameter that is usually set to $\epsilon = 0.001m$ in order to remove the small thickness layer, from the previous step, that contains non-object parts from F_{ob}.

In this step, we again optimize both NeRFs, F_{ob} and F_{bg_T} simultaneously for a fixed number of steps, s_5, which is an additional hyperparameter. The objective function to be minimized is given by:

$$\min_{\Theta_{\mathcal{F}}} \mathcal{L}_{c_{\mathcal{F}}}(\Theta_{\mathcal{F}} \mid \mathcal{I}_{scn}, \mathcal{R}_{scn}) + \lambda \mathcal{L}_{reg}(\Theta_{ob}, \Theta_{bg_T}, \mathcal{R}_{scn}),$$

where λ is a hyperparameter that represents the contribution of the depth regularization loss in the total loss.

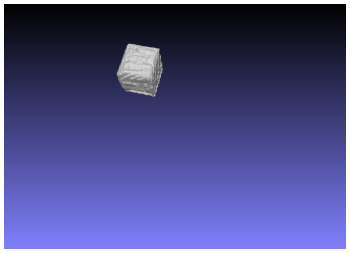


Figure 3.6: The scene mesh of F_{ob} after the final step.

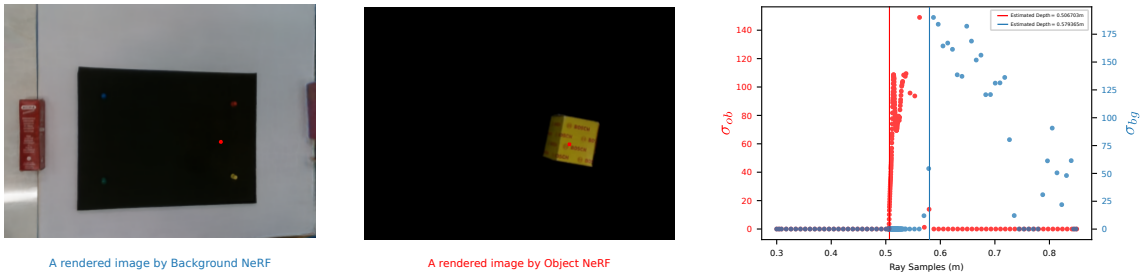


Figure 3.7: Examples of rendered images by Object-NeRF and Background-NeRF and σ distribution of both NeRFs along the ray that goes through the marked pixel.

After the final step of our method, we obtain a NeRF that represents the object only as shown in Figure 3.6. Figure 3.7 showcases examples of rendered images by F_{ob} and F_{bg_T}, and σ distribution of both NeRFs along the ray that goes through the marked pixel at the rendered images, illustrating that σ_{ob} has high values only on surface of the object and zero values on the background region. These NeRFs can be combined with various other NeRFs representing different objects to construct synthetic scenes with diverse object configurations. This process will be described in the subsequent section.

Moreover, as part of our method, we also obtain a trained F_{bg_T}. This NeRF exclusively represents the background of the scene and is aligned with S_{scn}. Utilizing this trained F_{bg_T}, we can generate aligned backgrounds for the recorded images in S_{scn}, which are useful for generating labels using the approach in 2.2.3 for the dataset of S_{scn} for the training of the grasp prediction network. This enables us to leverage synthetic data for labeling real data for training, further enhancing the grasp prediction network's performance and generalization capabilities.

3.3 Rendering of Composable Scenes

To generate scenes by combining NeRFs for objects and backgrounds, we follow a pipeline described in the work of Stelzner et al. (2021) [SKK21]. The process involves loading the desired trained NeRFs and combining them seamlessly.

In this approach, we first load the NeRF representation of the background, which captures the environment or scene without any objects. F_{bg} provides the foundational structure of the scene.

Next, we load the NeRFs corresponding to each object present in the scene. These Object-NeRFs, denoted as $F_{ob_1}, F_{ob_2}, ..., F_{ob_n}$, capture the appearance and geometry of individual objects. They contain information about the objects' shape and texture properties.

To place each object in the desired location within the scene, we use user-defined transformation matrices, $T_{ob_1}, T_{ob_2}, ..., T_{ob_n}$. These transformation matrices specify the position and orientation of each object. By applying the corresponding transformation matrix to the Object-NeRFs, we position the objects correctly within the scene.

Combining the NeRFs requires the F_{bg} to be combined with all transformed Object-NeRFs $F_{ob_i}(T_{ob_i}(x))$. This results in a combined set of NeRFs denoted as

$$\mathcal{F} = \{F_{bg}, F_{ob_1}(T_{ob_1}(x)), F_{ob_2}(T_{ob_2}(x)), ..., F_{ob_n}(T_{ob_n}(x))\}.$$

Each element in this set represents a component of the scene, including the background and the transformed objects.

Finally, we render images of the final scene using Equation 3.3. This process is typically explained in Section 3.1.

This pipeline allows for the creation of scenes consisting of a single background and multiple objects. Additionally, it provides flexibility in arranging objects within the scene, enabling the generation of scenes using the combined NeRF representations. It is also possible to generate larger-scale scenes by combining multiple backgrounds.

4 Experiments and Results

To assess the effectiveness of our proposed method, we perform two types of experiments:

1. Realism Evaluation: Here, the focus is on evaluating how realistic the generated scenes are compared to real scenes. The process involves generating new scenes using our method and comparing them with real scenes. Several metrics are used for the comparison. These metrics measure the similarity between the generated and the real images, and they quantify the realism of the generated scenes. Additionally, qualitative visual comparisons are also provided to assess the overall quality and realism of the generated scenes.
2. Grasp-Quality Prediction Network Evaluation: We aim to evaluate if NeRF-generated data can be used to train a grasp-quality prediction network. We conduct an ablation study to investigate how the performance of grasp-quality prediction networks is influenced by training on four NeRF-generated datasets. The performance of the grasp-quality prediction networks is then measured on a dedicated evaluation dataset which consists of real-world scenes. Comparing the performance of the networks on this dataset allows us to quantify the quality of the grasp prediction when trained on NeRF-generated data.

By conducting evaluations of both approaches, we can gain a comprehensive understanding of the strengths and limitations of the proposed method.

4.1 Realism Evaluation

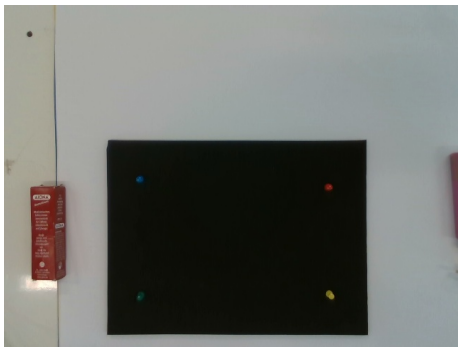
For a comprehensive evaluation of the effectiveness of the proposed pipeline in generating realistic scenes, we propose an evaluation methodology for assessing the realism of the generated scenes. It consists of the following steps:

- Three datasets are collected for evaluation. The first dataset includes recordings of the scene without objects. The second dataset includes recordings of the scene with an object of interest placed at a specific position. The third dataset comprises recordings of the scene with the same object, however, the object is located at a different position.
- Our pipeline is applied to one of the datasets containing the object with the dataset that doesn't contain the object in order to obtain separate NeRFs for the background and the object.
- Using the obtained Background-NeRF and Object-NeRF, a new scene is generated where the object is relocated to the position of the object in the other dataset and the background is adjusted.
- The generated scene is then compared with the ground truth real scene from the third dataset. Additionally, a qualitative comparison will also be performed to assess the overall realism and similarity between the generated and real scenes.

4.1.1 Datasets

(a) Sample image of Position A dataset.

(b) Sample image of Position B dataset.



(c) Sample image of Background Scene dataset.

Figure 4.1: Sample images of the datasets used in realism evaluation. Views of the scenes of Position A, Position B, and Background datasets are shown in a,b, and c, respectively.

Three datasets are used for the realism evaluation. These datasets are:

1. Background: This dataset consists of multiple RGB images of a static scene without any objects. Each image is accompanied by camera intrinsics and extrinsics that are estimated using DSO [EKC17] approach, which is one of the structure-from-motion techniques. An example of this dataset is shown in Figure 4.1c.
2. Position A: This dataset has the same structure as the Background dataset. It shows the same scene but with an object of interest placed at an arbitrary position, denoted by A, within the scene. Figure 4.1a shows an example of this dataset.
3. Position B: This dataset has the same structure as the Background dataset as well. It shows the same scene as the previous two datasets, but with the same object placed at a different position within the scene, denoted by B. An example of this dataset is provided in Figure 4.1b.

To collect these datasets, a RealSense D415 camera [23], with a resolution of 640×480, is mounted on a robot arm that follows a fixed trajectory around the scene. The camera records around 6000 images of different views. However, only 150 images are used for training, 10 images for validation, and 40 images for testing.

4.1.2 Experimental Setup

To assess the performance of our method, the evaluation is split into two sub-experiments:

- Exp1: As described in Section 3.2, we train on the Background and the Position A to obtain separate NeRFs of the background and the object. We then transform the obtained Object-NeRF to the pose of the object in Position B dataset, which is the target dataset in this experiment, and render new synthetic images with it. We compare the synthetic images to images rendered from a Vanilla NeRF trained on the Position B dataset.
- Exp2: To assess the robustness of our method against different object positions, we implemented this experiment. We do the same procedure as Exp1, however, we interchange between Position A and Position B datasets. Hence, our method is trained on the Background and Position B datasets. Then, the obtained Object-NeRF is transformed to the pose of the object in Position A dataset, which is the target dataset of this experiment, and the rendered images are compared with the images rendered from Vanilla NeRF trained on Position A dataset.

4.1.2.1 Scene Generation and Transformation

After applying our method, the obtained Object-NeRF should be transformed in such a way that the object is positioned at the same location as in the target dataset. To transform the Object-NeRF, we define a transformation matrix associated with the trained Object-NeRF. Additionally, the camera extrinsics in the three datasets are inconsistent due to differences in the reference frames of the three datasets. Consequently, the obtained Background-NeRF is not aligned with the background of the target dataset. To address this issue, we define another transformation matrix associated with the Background-NeRF to rectify this misalignment.

We divide the target dataset into two subsets: a training subset and an evaluation subset. Similar to Section 3.2.2, we obtain these transformation matrices by additionally training on the corresponding subset while keeping the parameters of the trained NeRFs fixed. This approach ensures accurate transformation of the object and alignment of the backgrounds in both experiments. The Vanilla NeRF is trained on the training subset as well. Subsequently, evaluation is performed on the evaluation subset.

4.1.2.2 Hyperparameters

In our method, each NeRF is structured with a coarse MLP employing 6 layers, each containing 256 neurons, while the fine MLP incorporates 8 layers, each comprising 256 neurons. The hyperparameters are configured as follows: the number of coarse samples N_c is set to 64, the number of fine samples N_f is set to 128, the near bound t_0 is set to 0.05m, and the far bound t_1 is defined as 0.8m.

Regarding the hardware and training aspect, our model is implemented on Pytorch framework, and it runs on an Nvidia V100 GPU with a batch size of 4096 and a learning rate of 0.0005. The number of training steps for each stage of our pipeline is as follows: s_1 is set to 80,000 training steps, s_2 is set to 8,000 training steps, s_3 is set to 12,000 training steps, s_4 is set to 15,000 training steps, and s_5 is set to 20,000 training steps.

For the training of the additional transformation matrices for aligning the background and accurately positioning the object, the additional training is implemented for 4,000 training steps with a batch size of 4096.

4.1.2.3 Evaluation Metrics

In NeRF literature, standard evaluation methods primarily rely on visual quality assessment metrics to benchmark the rendered images. Each viewpoint is expected to produce an image that shows the scene and its detail with sufficient accuracy. Content invariant metrics play a crucial role in determining whether an image contains noise that hinders it from resembling the ground truth image. These metrics aim to evaluate image quality either through pixel-wise similarity metrics like Peak Signal-to-Noise Ratio (PSNR) or structural similarity metrics such as Structural Similarity Index Measure (SSIM) or Learned Perceptual Image Patch Similarity (LPIPS).

PSNR is a measure commonly used in science and engineering to compare the level of the desired signal to the background noise level. It is defined as the ratio of signal power to noise power and is often expressed in decibels. The PSNR is calculated using the mean squared error (MSE) between the generated image and the ground truth image:

$$PSNR(x, y) = 10 \cdot \log_{10}(\mathrm{MAX}(x)^2) - 10 \cdot \log_{10}(\mathrm{MSE}(x, y)),$$

where $\mathrm{MAX}(x)$ is the maximum possible value of a pixel in an input image, e.g., 255 for 8-bit integer, and $\mathrm{MSE}(x, y)$ is the average of the squared differences between all color channels of the generated image x and target image y. However, PSNR doesn't always correlate well with human perception and may not capture perceptual differences.

As described in [WBSS04], SSIM provides a more comprehensive assessment of image quality beyond just pixel-wise differences, taking into account structural information and aims to mimic the human visual perception system's ability to identify structural information from a scene. The SSIM evaluates three metrics from an image: Luminance, Contrast, and Structure. Luminance is a measure of brightness. Contrast is the difference in color and brightness between pixels, and is calculated by computing the standard deviation of the luminance values. Structure captures the texture and patterns in the images, and is calculated based on the covariance in the luminance values. For a single image patch, the SSIM is given by:

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)},$$

where x and y are the generated and the target images, μ_x and μ_y are the mean luminance values, σ_x and σ_y are the standard deviations of x and y, σ_{xy} are the covariance between x and y, and $C_i = (K_i L)^2$, L is the dynamic range of the pixels (e.g., 255 for 8-bit integer), and $K_1 = 0.01$ and $K_2 = 0.03$ are constants chosen by the original authors. In an 11×11 circular symmetric Gaussian weighted window with weights w_i with a standard deviation of 1.5 and normalized to 1, the local statistics μ_0 and σ_0 are calculated. They are given by

$$\mu_x = \sum_i w_i x_i,$$

$$\sigma_x = (\sum_i w_i (x_i - \mu_x)^2)^{1/2},$$

$$\sigma_{xy} = \sum_i w_i (x_i - \mu_x)(y_i - \mu_y),$$

where x_i and y_i are pixels sampled from the generated and target images, respectively, to be compared. The patch-wise SSIM scores are averaged over the entire image in practice. SSIM can better account for perceptual differences and is often preferred over PSNR in evaluating visual quality.

As introduced in [ZIE+18], LPIPS is another objective metric that assesses the structural similarity of high-dimensional images with contextually dependent pixel values. Unlike previous metrics, LPIPS measures perceptual similarity rather than quality assessment. LPIPS is based on deep neural networks and operates by computing the feature similarities between image patches extracted from the generated and the target images. These features are typically obtained from pre-trained VGGNet. The LPIPS score is obtained by calculating a weighted pixel-wise MSE of feature maps over multiple layers. LPIPS is defined as

$$LPIPS(x, y) = \sum_l^L \frac{1}{H_l W_l} \sum_{h,w}^{H_l, W_l} ||w_l (x_{hw}^l - y_{hw}^l)||_2^2,$$

where $x_{h,w}^l$ and $y_{h,w}^l$ are the features of the generated and target images at pixel height h, pixel width w, and layer l of the pretrained network. H_l and W_l represent the height and width of the feature map at the corresponding layer. The original LPIPS paper utilized SqueezeNet[IHM+16], VGG[SZ14], and AlexNet[IHM+16] as feature extraction backbones with five layers in total. LPIPS takes into account high-level features learned by the neural network, making it a good choice for evaluating perceptual quality.

Employing these content invariant metrics enables us to make quantitative comparisons with ground truth images and, thus, to assess the quality and realism of the scenes generated by our approach.

4.1.3 Results

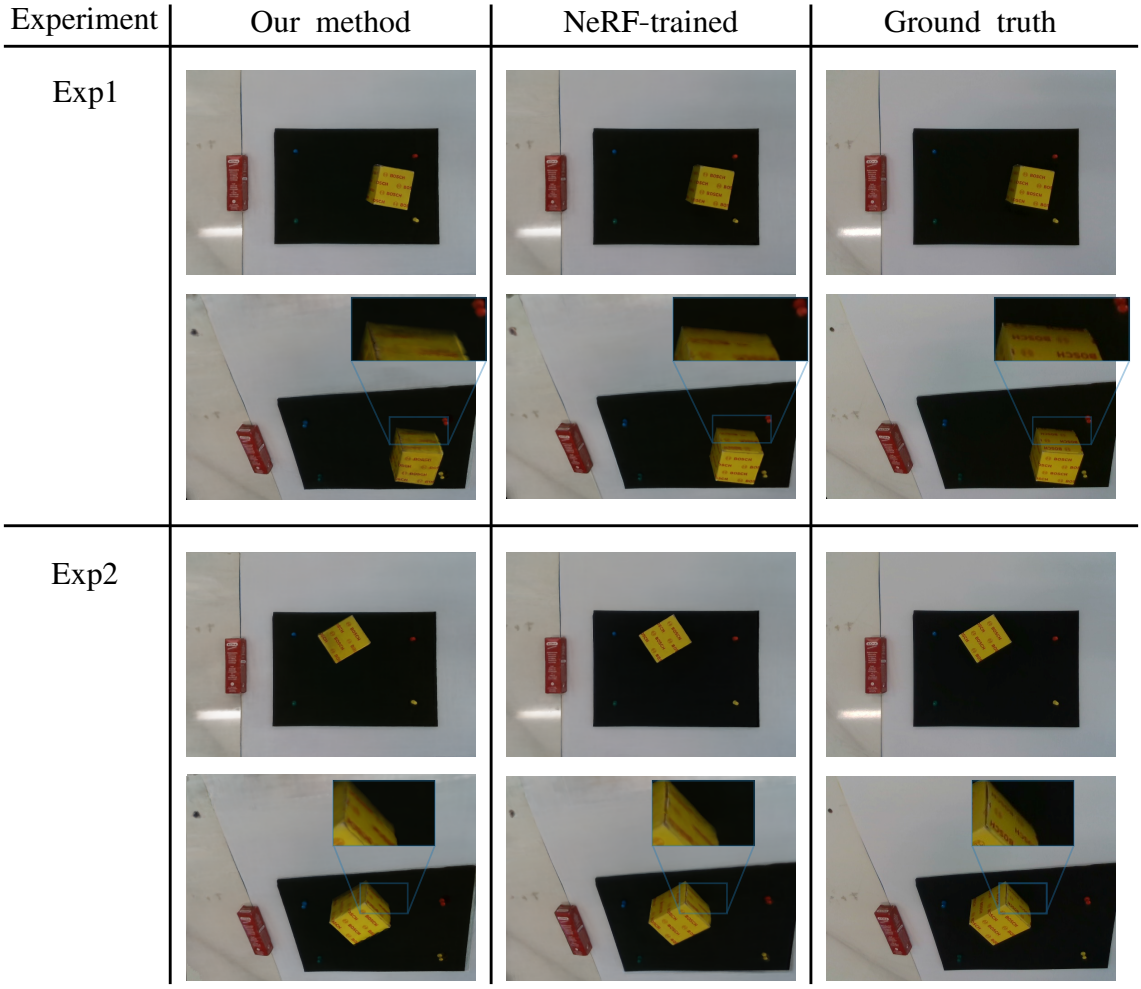


Figure 4.2: Qualitative comparison of rendered images by different methods. In Exp1, the scene of Position A, e.g., the ground truth scene from Exp2, along with the scene of Background is used to train our method. Subsequently, the object is repositioned to match its location in the ground truth scene of Exp1, e.g. the scene of Position B, resulting in the generation of a synthetic scene resembling the ground truth. Two rendered views of the synthetic scene are displayed under Our Method and Exp1. In Exp1 and NeRF-trained, the same views are used to render images from a Vanilla NeRF trained on the ground truth scene. The final column shows the corresponding ground truth images. The same process is replicated in Exp2, with the training scenes, however, Position A and Position B are switched, showing two rendered views from the final scene of Our Method, NeRF-trained, and the ground truth.

Our evaluation of the results will involve both qualitative and quantitative comparisons. In Figure 4.2, we provide a visual comparison between the generated scenes and the NeRF-trained scene, as well as the corresponding ground truth images for two different views for each experiment. Images

denoted by NeRF-trained in the figure are rendered by training a Vanilla NeRF on the training subset of the evaluation dataset. These comparisons are shown for two different views and for both experiments, Exp1 and Exp2. This qualitative analysis allows us to assess the visual realism and accuracy of the generated scenes compared to the ground truth.

Experiment	Method	PSNR↑ (dB)	SSIM↑	LPIPS↓
Exp1	Our method	27.04	0.9391	0.0945
	NeRF-trained	31.89	0.9611	0.0927
Exp2	Our method	25.34	0.9232	0.0972
	NeRF-trained	31.27	0.9571	0.0891

Table 4.1: Quantitative comparison between images rendered from the scene generated using our method and images rendered from a Vanilla NeRF trained on the corresponding ground truth scene.

Additionally, Table 4.1 presents quantitative comparisons between rendered images from the generated scene using our method, NeRF-trained, and the corresponding ground truth images. The quantitative metrics are averaged over 40 views, providing a comprehensive assessment of the effectiveness of our method.

4.1.4 Discussion

The provided comparison between our method and NeRF-generated results is primarily illustrative, aiming to show the means by which convergence toward NeRF-generated values can be achieved, although, from the qualitative comparison, it is clear that the object is accurately transformed to the target position in both experiments. It is noteworthy that both methods are trained on distinct datasets. Consequently, this section is dedicated to a discussion of the pivotal factors that could influence the results of our method. Addressing these factors holds the promise of more convergence of our results toward those obtained through NeRF-trained.

Firstly, regarding the results of our method in Exp1, in the second view, the side of the box that is under magnification from Our method appears to be less defined compared to the corresponding in the NeRF-trained. This issue could be due to that this side of the box is not covered well in the training subset of Position A dataset. In contrast, the training subset of the Position B dataset may provide more comprehensive coverage of this side. We can also realize that in Exp2 this side of the box is reconstructed better by our method than NeRF-trained.

Secondly, our method omits the integration of lighting and shadows during the scene generation process. However, due to the utilization of a black background in the experiments, this distinction is not clear.

A third factor focuses on the contact region between the background and the object, as shown in the qualitative comparison, which occasionally shows both elements, the object, and the background, as they are semi-transparent. This issue occurred due to the use of the combining NeRFs approach, which is based on the alpha blending method. Notably, this particular contact region encounters that

the background has a higher volume density compared to the object. Therefore, the object lacks the requisite volume density to effectively occlude the background, thus leading to the observed semi-transparent coexistence.

In summary, the combination of qualitative and quantitative comparisons allows us to demonstrate the effectiveness of our method in generating realistic scenes and its potential to enhance the training of grasp-quality prediction networks.

4.2 Grasp-Quality Prediction Network Evaluation

In this section, our primary objective is to demonstrate the feasibility of training a grasp-quality prediction network on data generated by NeRF in general, as well as on synthetic data generated using our method. Through a series of experiments, we will evaluate the performance of the grasp-quality prediction network using different training datasets. Our focus is on comparing the network's performance when training on real datasets versus different NeRF-generated synthetic datasets with novel views rendered using various camera intrinsics. Additionally, we will investigate the advantages of the synthetic data with views from novel scenes generated by our method.

As described in Section 2.2.3, the method applied for automatically annotating the images with grasp quality labels relies on subtracting a depth image of the background scene to identify objects. For this, the background image must be accurately aligned with the object image, which is, in practice, hard to achieve for setup with wrist-mounted cameras, e.g., moving camera setups. Our method outputs yields a sufficiently aligned background scene represented by F_{bg}, alongside the real scene S_{scn} and the synthetic scene obtained by combining F_{bg} and F_{ob}. This alignment enables the automatic labeling of the data and ensures a high quality dataset for training the grasp-quality prediction network.

To assess the generalization ability of the grasp-quality prediction network, we will consider two cases. Firstly, we will examine whether the network can effectively grasp unseen objects placed in a familiar environment, captured by a known camera, and positioned at a known location, referred to as similar scenes. Secondly, we will investigate whether the network can generalize to grasp both seen and unseen objects positioned in unseen environments, which are recorded using new camera types, representing entirely new scenes.

This evaluation will be structured as follows. We will first delve into the dataset used for training our proposed pipeline and present the outputs of our method, including the generation of novel views and novel synthetic scenes. These outputs will serve as the training data for the grasp-quality prediction network. Moving on, we will focus on the grasp-quality prediction network training datasets, which comprise the results obtained from our method. We will explain the different grasp-quality prediction networks that are trained on these datasets. Additionally, we will outline the evaluation dataset used to assess the performance of the grasp-quality prediction networks. To quantify the effectiveness of the models, we will employ various evaluation metrics which will be explained. Then we will showcase the final results, comparing the performance of the different grasp-quality prediction networks and highlighting the impact of our method on the grasp-quality prediction network's capabilities.

4.2.1 Methodology for Dataset Generation

For achieving robust and effective training of the grasp prediction model, it is essential to include a diverse set of objects in the dataset. This diversity ensures that the model becomes capable of detecting and generalizing to previously unseen objects. By exposing the model to various shapes, sizes, and appearances of objects during training, it can learn to extract meaningful features and patterns that are applicable to a wider range of objects. Thus, when presented with new, unseen objects during inference, the grasp prediction model will have a higher probability of

making accurate and successful grasp predictions. This ability to handle novel objects enhances the model's versatility and applicability in real-world scenarios, where encountering diverse objects is common.

4.2.1.1 Input Data

(a) Sample image of the scene with a yellow box.

(b) Sample image of the scene with a cyan tube.

(c) Sample image of the scene with a blue box.

(d) Sample image of the background Scene.

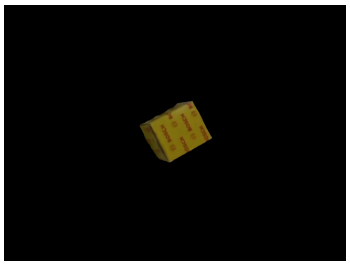
Figure 4.3: Samples of the training datasets, showing a sample of the background scene and three samples of different scenes.

The main objective of training our model in this evaluation is to generate synthetic data that will be used for training the grasp prediction model. To achieve this, we recorded various scenes, each containing a single object, for different objects. Additionally, we recorded an empty scene, i.g., a background scene. The structure of these datasets is similar to datasets in section 4.1.1 and the dataset collection setup as well. Specifically, we collected using a moving camera around the scene setup a total of 11 datasets, consisting of 10 scenes for 10 different objects, such as boxes, tubes, and other shapes. All of these scenes share a common background. We also recorded the background scene separately. Each recording comprises approximately 6000 images, along with their corresponding camera intrinsics and extrinsics, which are estimated using DSO [EKC17]. Figure 4.3 shows examples of a selection of datasets containing scenes with various objects, along with their corresponding background scene. For data preparation, we utilized from each collected dataset only 150 images for training, 10 images for validation, and 40 images for testing. These 40 test images were also employed for training the grasp model in the subsequent stages. The presence of a chessboard in the scenes is to obtain more precise camera extrinsics from DSO.

The implementation details and used hyperparameters are identical to those described in 4.1.2.2.

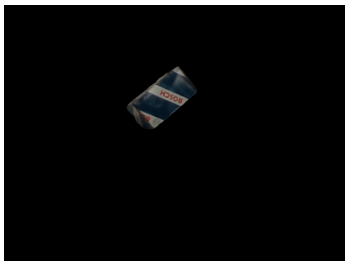
4.2.1.2 Methodology

We independently apply our method 3.2 to each of the object datasets, while using the same background scene for all of them. For each of them, we obtain two NeRFs: Object-NeRF and Background-NeRF. The Background-NeRF is accurately aligned with the corresponding dataset scene and utilized to generate aligned background images for the scene. This allows us to apply the automatic labeling approach explained in section 2.2.3 to obtain accurate labels for the data.



(a) Object-NeRF of a yellow box.

(b) Object-NeRF of a cyan tube.



(c) Object-NeRF of a blue box.

Figure 4.4: Rendered images generated by the trained object-NeRFs.

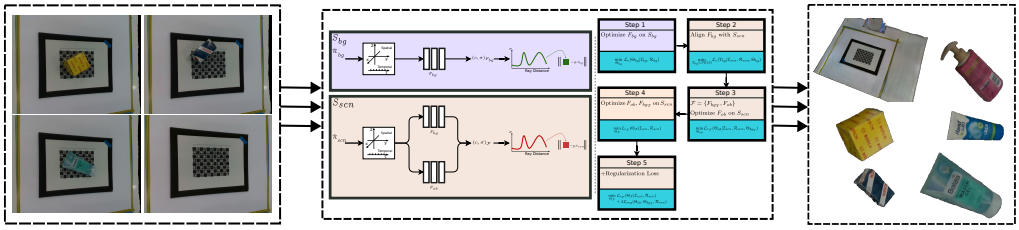
(a) Composing Object-NeRF and Background-NeRF of a yellow box.

(b) Composing Object-NeRF and Background-NeRF of a cyan tube.

(c) Composing Object-NeRF and Background-NeRF of a blue box.

Figure 4.5: Rendered images generated by composition of the trained Object-NeRFs and their corresponding Background-NeRFs.

The Object-NeRF, on the other hand, is combined with other Object-NeRFs and Background-NeRFs to generate synthetic scenes. Figure 4.4 presents a rendering of the trained Object-NeRF alone, where the images are rendered from the same pose as in Figure 4.3. By combining the Background-NeRF with the Object-NeRF of one experiment, we can generate novel views which are similar to views rendered by a trained Vanilla NeRF directly on the corresponding dataset. Figure 4.5 shows examples for composing the trained Object-NeRFs and correspondingly aligned Background-NeRFs.



a) Input Datasets

b) Our Proposed Pipeline for Object-Background Separation

c) Object/Background NeRF Library

Figure 4.6: The pipeline for obtaining a library of trained Object-/Background-NeRFs. The process begins with collecting the training dataset (a), which includes a background scene and scenes with objects. Our proposed pipeline for Object-Background separation is applied to obtain separate NeRFs for backgrounds and objects (b), resulting in a library of trained Object-/Background-NeRFs (c).

Figure 4.6 showcases the complete pipeline for obtaining a library of trained Object-/Background-NeRFs. The process starts with collecting the training dataset, which includes both a background scene dataset and a scene with an object dataset. Then our proposed method is applied to obtain separate NeRFs, one for the background and one for the object only. By applying our method to different datasets, we can create a library of trained Object-/Background-NeRFs, where each NeRF represents a specific object or a background. This library allows us to efficiently synthesize novel scenes with different combinations of objects and backgrounds.

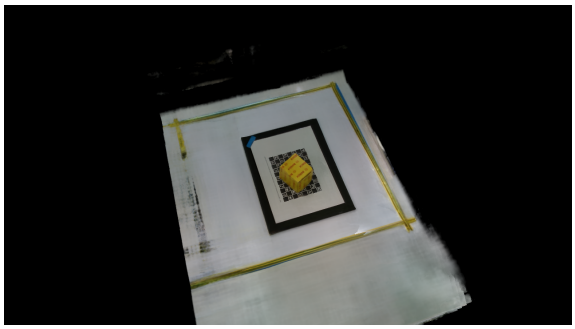
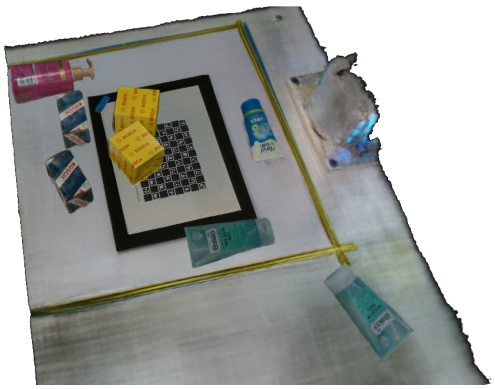


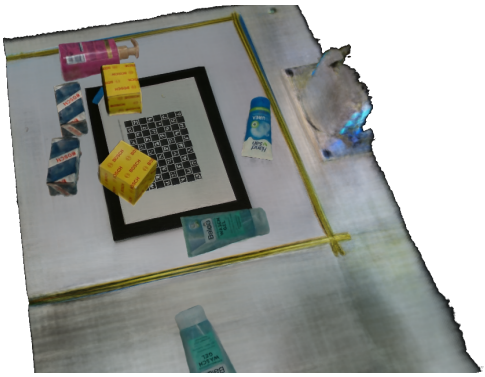
Figure 4.7: Novel view rendered using novel camera intrinsics

In addition, it is also possible to synthesize novel views by training a Vanilla NeRF directly on a scene. These synthesized views can then be utilized in training the grasp model. Figure 4.7 shows an illustrative example of rendering a novel view by combining the Background-NeRF and the Object-NeRF using novel camera intrinsics, specifically the intrinsics of RealSense D435 camera [22] with a resolution of 1280x720. This demonstrates the ability of our method, similar to Vanilla NeRF, to synthesize realistic images from different camera intrinsics.

a) Object/Background NeRF Library



b) Compose Scene



c) Move objects

d) Move Camera

Figure 4.8: Dynamic scene generation: The diagram showcases the process of creating a novel scene using the library of trained NeRFs. In (a), the library contains a collection of Background-/Object-NeRFs. By combining one Background-NeRF with 5 distinct Object-NeRFs, a composed scene with 9 NeRFs is generated (b). The position of each object can be adjusted individually through their respective transformation matrices, allowing for easy manipulation of object positions (c). Additionally, the camera position can be altered to synthesize different views of the scene (d). This dynamic approach enables us to create diverse scenes with varying object arrangements and camera perspectives.

Using a combination of the trained object-NeRFs along with one background-NeRF, and potentially repeating the usage of some of them, we can generate novel scenes. For each used NeRF, we define a desired transformation to place the objects at specific locations within the generated scene. By manipulating these transformation matrices, we can create dynamic scenes where the objects move within the scene. Figure 4 demonstrates how the library of trained NeRFs allows us to construct scenes with desired object configurations. We can manipulate the objects' positions within the scene and also adjust the camera extrinsic and intrinsic. This flexibility enables us to create various scenes with different arrangements and camera perspectives and dynamic scenes as well.

4.2.2 Training Dataset for Grasp Prediction Model

To evaluate the performance of the grasp-quality prediction network and its ability to be trained on NeRF-generated synthetic data, as well as the impact of augmenting data and generating novel scenes using our method, we create four dedicated datasets:

1. NeRF-Synth Dataset: This dataset is generated using a combination of Object-NeRFs and the corresponding Background-NeRF. It consists of novel views of the input scene. It is labeled by automatic labeling 2.2.3 using the images of this dataset and the corresponding background images rendered by Background-NeRF. Figure 4.5 illustrates examples of this dataset.
2. Real-Only Dataset: This dataset comprises real RGB-D images recorded by RealSense camera [23], that are corresponding to the generated images in NeRF-Synth Dataset, and it is not used in training our model. However, this data is labeled using the recorded images and the corresponding background images rendered by Background-NeRF as we lack aligned real background images. Examples of this dataset are shown in figures 4.3c, 4.3b and 4.3a
3. Camera-Augmented Dataset: This dataset includes the NeRF-Synth Dataset along with additional novel views rendered using different camera intrinsics. It is labeled similarly to the labeling of the NeRF-Synth Dataset. Figure 4.7 illustrate an example of the additional novel view.
4. Novel-Scene Dataset: This dataset comprises the Camera-Augmented Dataset and additional novel views of novel synthetic scenes generated using our method. It is labeled similarly to the labeling of the NeRF-Synth Dataset. Figure c shows an example of the additional novel views of novel synthetic scenes.

4.2.3 Experimental Setup

To train the grasp-quality prediction network and assess its performance, we first apply the automatic labeling approach 2.2.3 on the training dataset to generate labels. Then we train different instances of the network using randomly selected 400 samples from each of the aforementioned datasets:

1. NeRF-Synth GraspNet: Trained on the NeRF-Synth Dataset.
2. Camera-Augmented GraspNet: Trained on the Camera-Augmented Dataset.
3. Novel-Scene GraspNet: Trained on the Novel-Scene Dataset.
4. Real-Only GraspNet: Trained on the Real-Only Dataset.

The instances are trained for 30 epochs using an NVIDIA A1000 GPU, with a learning rate of 0.0001. Since the training dataset contains images with different resolutions, we resize the input to a fixed resolution of 1024×640 before feeding it to the network.

By comparing the performance of these trained GraspNets, we aim to evaluate the effectiveness of training on NeRF-generated synthetic data and the benefits of augmenting data with novel scenes generated using our method.

4.2.4 Evaluation Dataset

To finally evaluate the performance of the trained grasp-quality prediction networks and to assess their generalization capabilities, we collect two dedicated evaluation datasets:

- RealSense: This dataset comprises images collected using the same camera and setup as used for collecting the training dataset. It is specifically designed to evaluate the capabilities of the network to generalize to previously unseen objects within a similar environment to that used for collecting the training dataset. The scenes in this dataset include only separated objects, and the main objective is to assess the network's ability to grasp and recognize objects it has not encountered during training.
- Zivid: This dataset is collected using a different camera, specifically the Zivid 2 camera [Ziv], which provides a high resolution of 1920×1200. The scene in this dataset contains a completely different background and includes a larger number of objects. The scenes in this dataset are designed to resemble a typical bin-picking application, where objects are placed in a bin to be picked. The dataset encompasses both, seen and unseen objects with different configurations, such as separated, stacked, and overlapped objects. With this dataset we evaluate how well the networks generalize to new environments, as well as different cameras and backgrounds.

Figure 4.9: Example of evaluation datasets. The top-left image shows an example from the RealSense dataset, while the top-right image displays a simple evaluation sample from the Zivid dataset. The bottom-left image illustrates a typical bin containing overlapped random objects from the Zivid dataset, and the bottom-right image presents a special case where objects are stacked side by side from the Zivid dataset.

Figure 4.9 provides examples of the evaluation datasets. The top-left image showcases an example from the RealSense dataset. The top-right image presents a straightforward evaluation sample from the Zivid dataset. In the bottom-left image, we observe a typical bin filled with overlapped random objects from the Zivid dataset, simulating a real-world scenario in a logistics center. Lastly, the bottom-right image displays a special case from the Zivid dataset where objects are stacked side by side, reflecting a different object arrangement for evaluation purposes. These diverse evaluation datasets allow us to thoroughly assess the performance and generalization capabilities of our trained GraspNets.

Instead of the automatic labeling approach introduced in Section 2.2.3, we manually annotate the data of the evaluation sets with instance masks and grasp quality labels to avoid any inaccuracies in the data and the labels.

4.2.5 Evaluation Metrics

we evaluate the performance of the grasp prediction network based on the following metrics:

- Grasp Success, *Success*: The percentage of feasible grasps among the predicted grasps, where feasible grasps require non-zero grasp quality values. A successful grasp means that the predicted grasp is considered valid and likely to succeed.
- Grasp-Quality, *Quality*: The average ground-truth grasp quality of feasible grasps, which ranges between 0 and 1. This metric gives an indication of how well the predicted grasps align with the actual quality of the grasping action.
- Object clearing rate, *Object*: The percentage of objects for which grasps are detected. This metric counts the proportion of objects in the scene for which the grasp prediction network successfully detects valid grasps.

To achieve a high level of performance, it is essential for all three metrics to yield elevated values, with particular emphasis on grasp success rate and object clearing rate. Notably, these two metrics exhibit a relationship akin to a multiplication process; for instance, having a high grasp success rate coupled with a very low object clearing rate results in overall bad performance. To truly achieve exceptional performance, both grasp success rate and object clearing rate should concurrently demonstrate high values.

By considering these metrics, we can assess the overall performance and generalization of the grasp prediction network across different datasets and scenes.

4.2.6 Results

Evaluation Dataset	Model	Success↑	Quality↑	Objects↑
RealSense	NeRF-Synth GraspNet	74.28%	0.7330	59.04%
	Camera-Augmented GraspNet	80.00%	0.7667	60.24%
	Novel-Scene GraspNet	80.66%	0.8211	73.49%
	Real-Only GraspNet	85.19%	0.8188	74.69%
Zivid	NeRF-Synth GraspNet	92.40%	0.5258	38.12%
	Camera-Augmented GraspNet	73.15%	0.5908	71.87%
	Novel-Scene GraspNet	86.96%	0.6028	88.83%
	Real-Only GraspNet	91.15%	0.7506	92.5%

Table 4.2: Evaluation Results of Grasp Prediction Networks on the Evaluation Datasets

Table 4.2 summarizes the evaluation results for all four versions of the grasp prediction network. The Real-Only GraspNet demonstrates superior performance, achieving the best results among all the models. The second-best result is observed in the Novel-Scene GraspNet, which is based on our method. Additionally, the NeRF-Synth GraspNet performs well in seen environments, such as the RealSense dataset. However, its performance in unseen environments, like the Zivid dataset, exhibits a high grasp success rate but a low object clearing rate, leading to a bad performance overall. The augmentation of synthetic data with novel views and camera intrinsic rendering leads

to significant improvements, as seen in the Camera-Augmented GraspNet results. Furthermore, our approach of generating novel scenes and augmenting training data yields substantial performance enhancements, as demonstrated by the results of the Novel-Scene GraspNet when compared to the Camera-Augmented GraspNet outcomes. A notable performance gap is evident between the Real-Only GraspNet and the NeRF-Synth GraspNet in unseen environments.

Overall, the evaluation results confirm the effectiveness of our method in training the grasp prediction network using synthetic data and its ability to generalize to real-world scenarios when combined with novel views and camera intrinsic rendering. The results also highlight the significant improvement achieved by generating novel scenes and augmenting the data, demonstrating the potential of our approach in improving grasp prediction performance.

4.2.7 Discussion

As anticipated, the Real-Only GraspNet yields the best outcomes. This aligns with the expectation that training on genuine data surpasses the efficacy of synthetic data, which might not precisely replicate all intricacies of an authentic scene, particularly surface reconstruction. It's important to note that while this dataset incorporates real data, the labeling process involves synthesized background images to ensure alignment.

However, there is still a gap in performance between the NeRF-Synth GraspNet and the Real-Only GraspNet on the Zivid dataset. This discrepancy can be attributed to several factors. One of the primary factors is that our proposed pipeline relies on the Vanilla NeRF architecture, which exhibits some imperfections in surface reconstruction. This factor affects the accuracy of the NeRF depth-generated map, leading to challenges in the automatic labeling approach for grasp prediction. This problem is relatively mitigated when using depth maps from the camera compared to depth maps generated by NeRF. To address this challenge, one potential solution is to utilize a variant NeRF architecture, such as the depth-supervised NeRF proposed in [DLZR22]. This architecture incorporates camera depth information within the NeRF framework, enabling better surface reconstruction and potentially improving the accuracy of depth maps used for automatic labeling. Another approach could involve representing the surface as a signed distance function, as demonstrated in works like [WLL+21] and [WTB+23]. These techniques aim to enhance the reconstruction of surfaces, which could further enhance the performance of the grasp-quality prediction network.

Novel-Scene GraspNet shows the second-best result, as it exposes the network to multi-object scenes with a wider range of object configurations. However, this model can achieve better results as the generated synthetic scene used in the Novel-Scene Dataset only involves one dynamic scene where objects move along certain defined trajectories, but all objects are separated. This dataset does not encompass more complex scenarios, such as stacked objects or overlapped objects. Introducing such complex scenarios in the generated synthetic scenes and augmenting them with the grasp network training data could further improve the performance of the grasp prediction quality network.

The results demonstrate the potential of training the grasp prediction network on NeRF-generated synthetic data and the importance of augmenting novel scenes, views, and different camera intrinsics rendering to enhance generalization to new environments. However, further improvements can be achieved by employing more accurate NeRF architectures and generating synthetic scenes with a broader range of complex scenarios.

4.3 Real-world Grasping Experiment

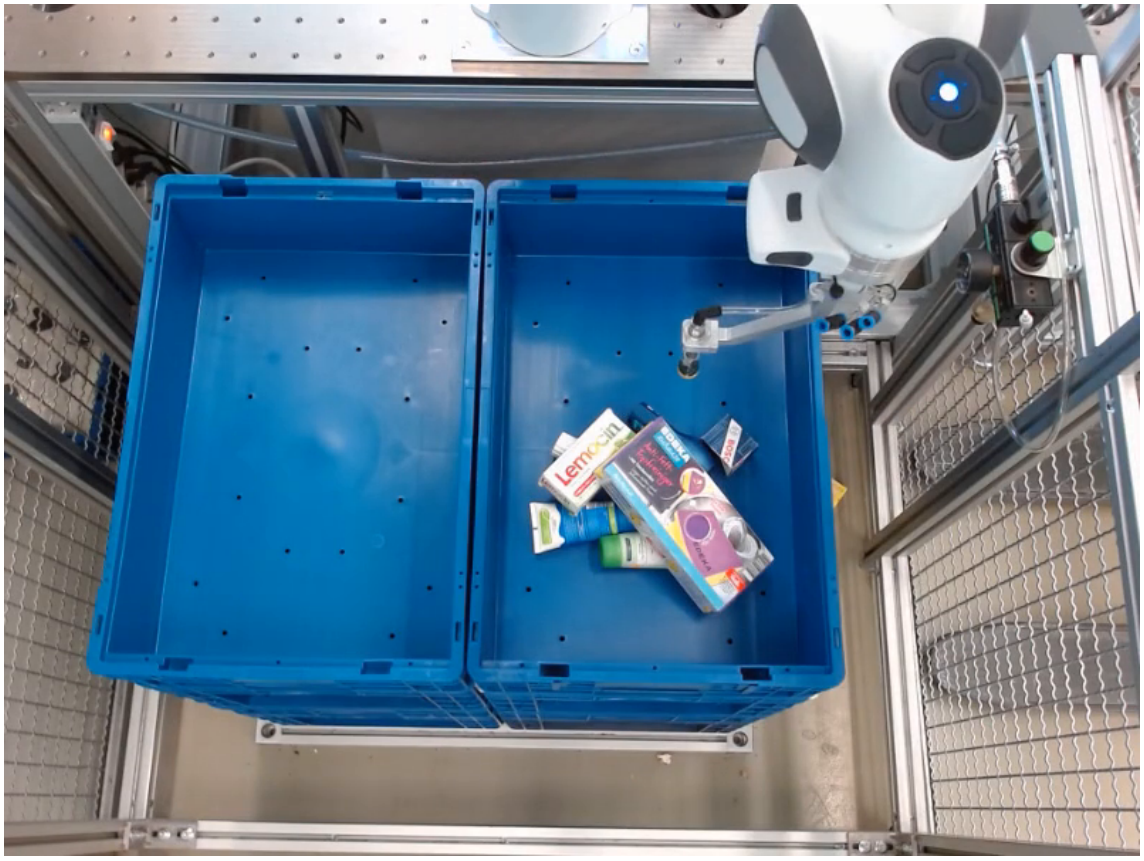


Figure 4.10: Bin picking system consisting of a robot arm equipped with a single cup suction gripper and bins containing random objects. The task is to pick objects from the right bit and place them in the left bin.

Finally, we apply the two most promising candidates of our trained grasping network, Real-Only GraspNet, and Novel-Scene GraspNet, in a real-world bin-picking application that simulates the commissioning processes in a logistic center, as shown in Figure 4.10. The setup comprises a Zivid one+ RGB-D camera [Ziv] positioned approximately 1.2 meters above the bins, along with a Franka Emika robot arm [Fra] which is equipped with a single cup suction gripper at the end effector for grasping objects. The objective is to pick objects from the right bin and drop them in the left bin.

To demonstrate the capabilities of the trained grasp networks, we utilize objects similar to those found in the grasp quality evaluation datasets. The experiment follows predefined sequences in which the system records a new image, and uses the trained grasping networks to determine the grasp candidates. It then performs the candidate with the highest probability of a successful grasp.

A video of the experiments is provided in the supplementary material of this thesis.

The results of the experiments show that all objects in the bin are successfully picked from the left bin and dropped in the right bin using both GraspNet. Additionally, after successfully emptying the right bin, both models give zero possible grasps. The results showcase the potential of using the trained model for robust and efficient grasp prediction in bin-picking applications.

5 Conclusion and Outlook

In this thesis, we introduced a novel approach for obtaining distinct NeRFs of objects and backgrounds using a dataset containing two scenes: one representing the background and the other representing both an object of interest and the background. The core idea is to start by training a NeRF on the background scene to implicitly represent the background in 3D space. We then align this implicit scene with the scene containing the object. Next, we fix the weight of the NeRF, introduce another NeRF, and combine both NeRFs and train on the scene containing the object then unfix the fixed weights and jointly train them on the scene containing the object. This process allows us to obtain a NeRF representing only the object and another one representing only the background but aligned with the object scene. We applied our proposed approach to different datasets, in order to create a library of trained NeRFs. These NeRFs can be combined to compose novel synthetic scenes.

Additionally, we extend the automatic labeling approach, described in Section 2.2.3, to work with datasets collected with a moving camera by utilizing the RGB-D images rendered by the NeRF representing the background for labeling the data.

Furthermore, we assessed our approach using two distinct methodologies. The first approach was designed to assess the realism of the generated scenes using our proposed method. Under this approach, we conducted two separate experiments on different datasets. The idea of this approach is to apply our proposed method to two datasets, a background scene dataset, and a complete scene, including an object of interest, dataset, to obtain separate NeRFs, one representing only the background and another one representing only the object. Then, another dataset of the same scene, but the object is positioned in a different position, is collected. Using our method, we generated a scene that is similar to the last collected scene, and images of both, the real and the generated scenes, are compared in order to evaluate the realism of our proposed method. By analyzing both qualitative and quantitative results, we showed that the generated scenes are realistic. In the second evaluation approach, we demonstrated that our proposed method has the potential to enhance the effectiveness of detection models, such as the grasp prediction model. From the outcomes of this approach, we showed that the grasp prediction model can be trained on NeRF-generated synthetic data and give reasonable results on real data. Additionally, we proved that including the generated synthetic scenes and novel views into the training process of the grasp prediction model significantly improves the performance of the grasp-quality prediction network by generalizing to new and diverse environments, making it more applicable in real-world scenarios.

Additionally, we demonstrated the practical applicability of the trained grasp-quality prediction network in a research setup of a bin-picking application. The experiment is conducted using a robotic cell equipped with RGB-D cameras and is a real-world test for the proposed approach.

The experiment's outcomes are remarkably encouraging, as the robot effectively performs the grasping task for all objects in the bin by utilizing the predicted feasible grasps provided by the trained grasp-quality prediction network. This successful execution exemplifies the approach's

potential in real-world scenarios, especially in cluttered environments where robotic systems must precisely and dependably grasp objects. This application has significant implications for automation and manufacturing processes.

In summary, this thesis proposes a novel NeRF-based synthetic scene generation approach and proves the effectiveness of using NeRF-generated synthetic data for training grasp-quality prediction networks. The work underscores the potential of using NeRF-generated data to improve the performance and generalization capabilities of detection models.

5.1 Future Work and Outlook

Due to the time constraints of this thesis and our primary focus on proving the concept of our proposed pipeline to obtain separate NeRFs for objects and demonstrate the feasibility of training detection networks on NeRF-generated data, several modifications and experiments are left for future endeavors. Examples of these modifications and experiments are

- Enhancing Depth Maps: An essential extension would involve integrating a NeRF architecture, that offers better surface reconstruction capabilities, into our pipeline. This improves the quality of depth maps used for generating grasp-quality labels, and would further improve the performance of the grasp prediction network.
- Realism Boost with Inverse Rendering Techniques: A potential extension is incorporating inverse rendering techniques, similar to [WSG+23]. This could introduce varying shadows and lighting effects to the synthetic scenes, improving the realism of the generated scenes, and further improving the grasp-quality prediction network's performance.
- Exploring Synthetically Complex Scenes: Experiments that could be explored involve generating more complex scenes, which, for example, contain overlapping or stacked objects. Adding images and labels from such scenes to the training dataset of the grasp network could improve its performance in cluttered and stacked scenarios with many objects.
- Including Object Collisions: Drop objects into the scene by considering gravity instead of placing them is another interesting work. This would require including collision checking for the objects and could make the scenes more realistic.
- Creating a Combined Representation of a Number of Objects of Interest: Our approach isn't restricted to extracting a single object; it's adaptable to scenes with multiple objects. In such cases, the trained object-NeRF would encompass all the objects present, excluding the background. To separate individual objects from the NeRF and determine which specific objects to display, a simple clustering approach like DBSCAN could be employed.
- Extending to Other Detection Models: It's worth assessing the performance improvement when our proposed method is used to generate novel scenes and NeRF-generated data for other detection models, such as segmentation models. This exploration could provide insights into the versatility and applicability of our approach across various detection tasks.

In conclusion, while we provide a strong foundation and promising results for NeRF-based synthetic scene generation and grasp-quality prediction, there are numerous avenues for further research and development that could advance the capabilities and applications of the proposed approach.

List of Figures

List of Tables

Bibliography

[22] *RealSense D435 Depth Camera*. Dec. 2022. URL: https://www.intelrealsense.com/depth-camera-d435/ (cit. on p. 46).

[23] *RealSense D415 Depth Camera*. Apr. 2023. URL: https://www.intelrealsense.com/depth-camera-d415/ (cit. on pp. 37, 48).

[BDC01] A. Broadhurst, T. W. Drummond, R. Cipolla. "A probabilistic framework for space carving". In: *Proceedings eighth IEEE international conference on computer vision. ICCV 2001*. Vol. 1. IEEE. 2001, pp. 388–393 (cit. on p. 21).

[BRR11] M. Bleyer, C. Rhemann, C. Rother. "Patchmatch stereo-stereo matching with slanted support windows." In: *Bmvc*. Vol. 11. 2011, pp. 1–11 (cit. on p. 21).

[DLZR22] K. Deng, A. Liu, J.-Y. Zhu, D. Ramanan. "Depth-supervised nerf: Fewer views and faster training for free". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 12882–12891 (cit. on pp. 16, 52).

[EKC17] J. Engel, V. Koltun, D. Cremers. "Direct sparse odometry". In: *IEEE transactions on pattern analysis and machine intelligence* 40.3 (2017), pp. 611–625 (cit. on pp. 12, 36, 44).

[Fra] Franka. *FRANKA RESEARCH 3*. URL: https://www.franka.de/research (cit. on p. 53).

[GFWF20] M. Guo, A. Fathi, J. Wu, T. Funkhouser. "Object-centric neural scene rendering". In: *arXiv preprint arXiv:2012.08503* (2020) (cit. on p. 22).

[HZRS16] K. He, X. Zhang, S. Ren, J. Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778 (cit. on p. 18).

[IAKG21] J. Ichnowski, Y. Avigal, J. Kerr, K. Goldberg. "Dex-NeRF: Using a neural radiance field to grasp transparent objects". In: *arXiv preprint arXiv:2110.14217* (2021) (cit. on pp. 15, 31).

[IHM+16] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, K. Keutzer. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size". In: *arXiv preprint arXiv:1602.07360* (2016) (cit. on p. 39).

[KBKH20] K. Kleeberger, R. Bormann, W. Kraus, M. F. Huber. "A survey on learning-based robotic grasping". In: *Current Robotics Reports* 1 (2020), pp. 239–249 (cit. on p. 17).

[KFH+22] J. Kerr, L. Fu, H. Huang, Y. Avigal, M. Tancik, J. Ichnowski, A. Kanazawa, K. Goldberg. "Evo-nerf: Evolving nerf for sequential robot grasping of transparent objects". In: *6th Annual Conference on Robot Learning*. 2022 (cit. on p. 22).

[KSZ+21] A. R. Kosiorek, H. Strathmann, D. Zoran, P. Moreno, R. Schneider, S. Mokrá, D. J. Rezende. "Nerf-vae: A geometry aware 3d scene generative model". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 5742–5752 (cit. on p. 22).

[KV84] J. T. Kajiya, B. P. Von Herzen. "Ray tracing volume densities". In: *ACM SIGGRAPH computer graphics* 18.3 (1984), pp. 165–174 (cit. on p. 14).

[LC98] W. E. Lorensen, H. E. Cline. "Marching cubes: A high resolution 3D surface construction algorithm". In: *Seminal graphics: pioneering efforts that shaped the field*. 1998, pp. 347–353 (cit. on p. 28).

[Lev90] M. Levoy. "Efficient ray tracing of volume data". In: *ACM Transactions on Graphics (TOG)* 9.3 (1990), pp. 245–261 (cit. on p. 16).

[LGO+23] V. Lazova, V. Guzov, K. Olszewski, S. Tulyakov, G. Pons-Moll. "Control-nerf: Editable feature volumes for scene rendering and manipulation". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2023, pp. 4340–4350 (cit. on p. 21).

[LP17] K. M. Lynch, F. C. Park. *Modern robotics*. Cambridge University Press, 2017 (cit. on p. 29).

[MLN+17] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, K. Goldberg. "Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics". In: *arXiv preprint arXiv:1703.09312* (2017) (cit. on p. 22).

[MST+21] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, R. Ng. "Nerf: Representing scenes as neural radiance fields for view synthesis". In: *Communications of the ACM* 65.1 (2021), pp. 99–106 (cit. on pp. 11–13, 15).

[NGC+23] R. Newbury, M. Gu, L. Chumbley, A. Mousavian, C. Eppner, J. Leitner, J. Bohg, A. Morales, T. Asfour, D. Kragic, et al. "Deep learning approaches to grasp synthesis: A review". In: *IEEE Transactions on Robotics* (2023) (cit. on p. 17).

[RBA+19] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio, A. Courville. "On the spectral bias of neural networks". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 5301–5310 (cit. on p. 15).

[RFB15] O. Ronneberger, P. Fischer, T. Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*. Springer. 2015, pp. 234–241 (cit. on p. 18).

[SCD+06] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, R. Szeliski. "A comparison and evaluation of multi-view stereo reconstruction algorithms". In: *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06)*. Vol. 1. IEEE. 2006, pp. 519–528 (cit. on p. 21).

[SF16] J. L. Schonberger, J.-M. Frahm. "Structure-from-motion revisited". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 4104–4113 (cit. on p. 12).

[SGK+23] P. Schillinger, M. Gabriel, A. Kuss, H. Ziesche, A. V. Ngo. “Model-free Grasping with Multi-Suction Cup Grippers for Robotic Bin Picking”. In: 2023 (cit. on pp. 8, 17, 19).

[SKK21] K. Stelzner, K. Kersting, A. R. Kosiorek. “Decomposing 3d scenes into objects via unsupervised volume segmentation”. In: *arXiv preprint arXiv:2104.01148* (2021) (cit. on pp. 22, 25, 33).

[SZ14] K. Simonyan, A. Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014) (cit. on p. 39).

[WBSS04] Z. Wang, A. C. Bovik, H. R. Sheikh, E. P. Simoncelli. “Image quality assessment: from error visibility to structural similarity”. In: *IEEE transactions on image processing* 13.4 (2004), pp. 600–612 (cit. on p. 38).

[WLL+21] P. Wang, L. Liu, Y. Liu, C. Theobalt, T. Komura, W. Wang. “Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction”. In: *arXiv preprint arXiv:2106.10689* (2021) (cit. on p. 52).

[WSG+23] Z. Wang, T. Shen, J. Gao, S. Huang, J. Munkberg, J. Hasselgren, Z. Gojcic, W. Chen, S. Fidler. “Neural Fields meet Explicit Geometric Representations for Inverse Rendering of Urban Scenes”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 8370–8380 (cit. on p. 56).

[WTB+23] B. Wen, J. Tremblay, V. Blukis, S. Tyree, T. Müller, A. Evans, D. Fox, J. Kautz, S. Birchfield. “BundleSDF: Neural 6-DoF Tracking and 3D Reconstruction of Unknown Objects”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 606–617 (cit. on pp. 21, 52).

[YFB+21] L. Yen-Chen, P. Florence, J. T. Barron, A. Rodriguez, P. Isola, T.-Y. Lin. “inerf: Inverting neural radiance fields for pose estimation”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 1323–1330 (cit. on pp. 14, 29).

[YZX+21] B. Yang, Y. Zhang, Y. Xu, Y. Li, H. Zhou, H. Bao, G. Zhang, Z. Cui. “Learning object-compositional neural radiance field for editable scene rendering”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 13779–13788 (cit. on p. 21).

[ZIE+18] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, O. Wang. “The unreasonable effectiveness of deep features as a perceptual metric”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 586–595 (cit. on p. 39).

[Ziv] Zivid. *Zivid two industrial 3D camera*. URL: https://www.zivid.com/zivid-2 (cit. on pp. 49, 53).

[ZSY+22] A. Zeng, S. Song, K.-T. Yu, E. Donlon, F. R. Hogan, M. Bauza, D. Ma, O. Taylor, M. Liu, E. Romo, et al. “Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching”. In: *The International Journal of Robotics Research* 41.7 (2022), pp. 690–705 (cit. on p. 22).

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature