ETH zürich

Computer Vision and Geometry Lab

Optimizing 3D Object-Wise Representations

Master Thesis

Christian Bohn

Advisors:
Cathrin Elich, Martin Oswald
Supervisor:
Prof. Marc Pollefeys

Contents

1 Introduction **3**

2 Related Work **4**

3 Method **5**
3.1 Model Architecture 5
3.2 Render-and-Compare Optimization 8
3.2.1 Abstract Overview of the Optimization Process 8
3.2.2 Implementation Details 8
3.2.3 Optimization of Latents Initialized as ϵ-Vectors 10
3.2.4 Optimization with Anti-Aliasing 11
3.3 Loss Functions 12
3.3.1 Overview: Reconstruction Losses, Regularizers, Ground Loss, Intersection Loss 12
3.3.2 Silhouette Reconstruction Losses 13
3.3.3 Loss Weights and Smoothing Kernel Schedules 17
3.4 Supervised Training with Optimized Latents 19

4 Data, Configurations, and Evaluation Metrics **20**
4.1 The Clevr Dataset 20
4.2 Reconstruction Metrics 21
4.3 Model Variants to be Compared 25
4.4 Render-and-Compare Configurations for Experiments 26

5 Experiments and Discussion **27**
5.1 Comparison of the Models' Behavior during Training 27
5.2 Evaluation Results from Direct Model Outputs 30
5.3 Evaluation Results from Render-and-Compare Outputs 33
5.4 Evaluation Results from Render-and-Compare Outputs with Anti-Aliasing . . 36
5.5 Evaluation Results from Render-and-Compare Outputs with Shape Updates . 38
5.6 Limitation to Scenes with Previously Known Number of Objects 40
5.7 Runtime Discussion 42

6 Concluding Remarks and Further Work **43**

1 Introduction

For humans, semantically decomposing a scene into the objects in it and understanding the geometry and texture of each object is a simple and effortless task. Here, we seek to train a neural network model in a similar way to how humans learn this task as infants, by *self-supervision*, i.e., no one tells them the true parameters of the objects to learn from and instead they refine their mental images of objects by comparing them to their observations. Thus, this training approach can in some sense be called natural.

In the area of deep learning for object-wise scene decompositions, self-supervised learning has recently gained high attention. The task of decomposing an input scene from an image into descriptions for each of the objects visible in it lends itself nicely to the self-supervised learning approach. Another advantage of this approach is that datasets to train on are readily available or very cheap to produce, as they do not require annotation, as in supervised learning.

In this work, we present an optimization module based on the self-supervised model architecture introduced in [Elich et al., 2021]. This deep learning model can be thought of as an autoencoder architecture. It decomposes an image of the input scene into latent vectors describing its constituent objects in terms of their shape, texture, and pose. These latent vectors are then passed to a differentiable renderer that reconstructs them back into a rendering of the input scene. Ideally, this rendering matches the input image as closely as possible.

Our module that we add to this model implements the *Render-and-Compare Optimization*, similar to what was introduced in [Li et al., 2019] and [Beker et al., 2020a]. The main idea behind this optimization is to iteratively improve the object-wise description and reconstruction of a scene. To that end, for each object in the scene, we take its latent vector produced by the model as an initialization and iteratively update it in the following manner: We first render the image corresponding to the latent vector. Then, from that rendering, we can define a *reconstruction loss* between it and the input image. We then compute the gradient of that loss w.r.t. the latent vector and perform a gradient descent step based on it. This results in an updated vector that is likely to describe the object better. With this updated version, we then continue in another iteration of that process if the reconstruction loss has not converged yet.

We evaluate the results of that optimization process on a version of the synthetic CLEVR dataset.

The significant contributions of this work are:
1) The implementation of the Render-and-Compare optimization described above.
and
2) The introduction of some advanced losses into the training procedure to address specific shortcomings of the neural network model which the Render-and-Compare optimization was not or not sufficiently able to overcome on its own.

2 Related Work

Self supervised learning of scene decompositions and object-wise shape- and pose-estimation has been demonstrated in the following, significant works: In [Nguyen-Phuoc et al., 2020] and [Liao et al., 2019] images of multi-object scenes are generated from object-wise descriptions, but they lack the capacity to encode scenes. A method for 3D pose estimation in multi-object scenes with known object geometries is presented in [Periyasamy et al., 2019]. In [Greff et al., 2020], [Locatello et al., 2020], and [Burgess et al., 2019] methods are shown to segment, encode, and reconstruct the visible objects in the input scene images in the pixel domain.

In our implementation, the object-wise geometries are represented as signed distance functions predicted by a DeepSDF model, as introduced in [Park et al., 2019]. Similar to that approach in the usage of deep learning of continuous object representation are
[Sitzmann et al., 2020], [Xu et al., 2019], [Mescheder et al., 2019], [Chen and Zhang, 2019] and [Oechsle et al., 2019]. Alternatively, neural networks can also be used to express object shapes in discrete representations such as voxel grids ([Tulsiani et al., 2017], [Kar et al., 2017], [Wu et al., 2017], [Gadelha et al., 2016], [Qi et al., 2016], [Rezende et al., 2018], [Choy et al., 2016], [Shin et al., 2018], [Xie et al., 2019]), point clouds ([Qi et al., 2017], [Achlioptas et al., 2018]), and meshes ([Groueix et al., 2018]).

Differentiable rendering has been introduced in many recent works. Here a distinction needs to be made between fully deep learning based approaches using generative neural networks such as in [Niemeyer and Geiger, 2021] and [Nguyen-Phuoc et al., 2020] and more traditional explicit rendering pipelines implemented in a differentiable way, as shown in [Tulsiani et al., 2017], [Nguyen-Phuoc et al., 2019], [Gadelha et al., 2016], [Richardson et al., 2017], and [Sitzmann et al., 2020]. The rendering method demonstrated in [Sitzmann et al., 2020] is based on signed distance functions, similar to the differentiable renderer implementation used in this work.

In [Li et al., 2019] an iterative optimization process to estimate an object's pose is demonstrated. Differing from our approach, they do not update the pose descriptions directly via gradient descent, but train a neural network that produces these updates for the input image and the object rendering based on the current pose. However, they assume the geometry of the object is known, which in our case is not given.
A Render-and-Compare optimization process similar to what is presented in this work is used in [Beker et al., 2020b]. That work differs from ours in its limitation to a very specific set of objects detected and represented, namely cars in street scenes. The CLEVR dataset [Johnson et al., 2017] used in our work covers a wider range of possible objects and provides less context that helps with object reconstruction.
This work directly builds on the model presented in [Elich et al., 2021] which is detailed in the subsequent Chapter.

3 Method

The architecture of the model we use throughout this work is explained in detail in Chapter 3.1. In Chapter 3.2, we discuss the Render-and-Compare optimization method to refine predictions from the model, and in Chapter 3.3 we explain the refined loss functions that address some issues with the original variant of the model.

3.1 Model Architecture

The neural network architecture used in this work is unchanged from the one in [Elich et al., 2021]. In the broadest sense, it can be thought of as an autoencoder architecture: For some number N of objects present in an image, this model encodes an input RGB image into N latent vectors, each describing the shape, texture and pose of an object visible in the image. These latent vectors are subsequently passed into a differentiable renderer that produces a reconstructed image of the scene depicted in the input. The aim is to produce an output image that matches the input image as closely as possible. This neural network is trained using RGB input images and ground-truth depth-maps.
In the subsequent paragraphs, we discuss both the encoder- and decoder-components of this architecture in detail, followed by a description of its self-supervised training procedure.

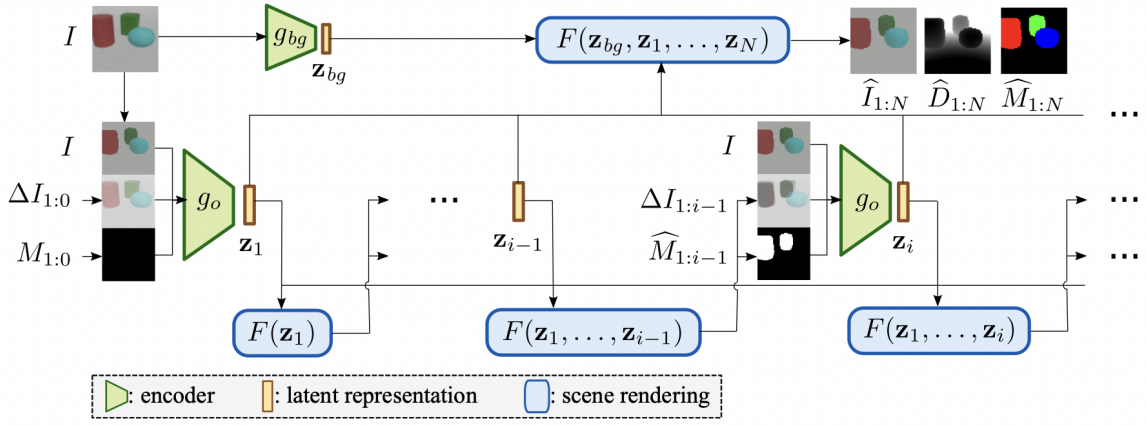

Figure 1: **Model Architecture overview.** (Figure taken from [Elich et al., 2021]). In this sequential architecture, for each object, the object encoder network g_o receives the input image I, a mask M based on the objects already reconstructed, as well as the difference-image ΔI. From these inputs, it outputs the latent vector $\mathbf{z}$ of the next object to be represented. Taking these latent vectors as input, the decoder F, based on a differentiable renderer, is used to produce renderings and masks of the individual objects and also to reconstruct the entire image, given $\mathbf{z}_{bg}$, the latent representation of the image background.

Scene Encoding.
All objects $i \in \{1, ..., N\}$ are encoded sequentially by the same object encoder g_o. In an *object slot* of the model, we encode an object i in the scene depicted in the input image

I into its latent representation $\mathbf{z}_i = g_o(I, \Delta I_{1:i-1}, \hat{M}_{1:i-1})$. Here, $\hat{M}_{1:i-1}$ denotes the occlusion mask of all already reconstructed objects $\{1, ..., i-1\}$, and $\Delta I_{1:i-1}$ refers to the difference image between the input and the combined rendering of all objects reconstructed so far. These inputs are computed from the output of the previous slot as follows: Denoting the scene decoding detailed in the next paragraph by F, and given the latent vectors for the background and the object set $\{1, ..., i-1\}$, the output of the slot for object $i-1$ is $(\hat{I}_{1:i-1}, \hat{D}_{1:i-1}, \hat{M}_{1:i-1}) = F(\mathbf{z}_{bg}, \mathbf{z}_1, ..., \mathbf{z}_{i-1})$. Here we refer by $\hat{I}_{1:i-1}$ to the predicted composition image, by $\hat{D}_{1:i-1}$ to the depth map, and by $\hat{M}_{1:i-1}$ to the occlusion mask showing all objects up to object $i-1$. From that we generate the difference image as $\Delta I_{1:i-1} = I - \hat{I}_{1:i-1}$. The uniform background color $\mathbf{z}_{bg} \in \mathbb{R}^3$ is regressed by the background encoder $g_{bg}(I) = \mathbf{z}_{bg}$.

The latent vector $\mathbf{z}_i$ for object i is comprised of shape, texture, and extrinsics latent encodings $\mathbf{z}_{i,sh}$, $\mathbf{z}_{i,tex}$, and $\mathbf{z}_{i,ext}$. The latents $\mathbf{z}_{i,sh}$ and $\mathbf{z}_{i,tex}$ are input parameters to the shape- and texture-decoder networks Φ and Ψ. The shape decoding network Φ is a DeepSDF model, regressing signed-distance-function values based on a shape latent parameter for query points, as introduced in [Park et al., 2019]. The extrinsics latent vector $\mathbf{z}_{i,ext} = (x_i, y_i, z_i, \cos(\theta_i), \sin(\theta_i), s_i)$ directly contains the object's position $\mathbf{p}_i = (x_i, y_i, z_i)$ in world coordinates, rotation $\theta_i = \arctan(\cos(\theta_i), \sin(\theta_i))$, and the objects scale $s_i \in [s_{min}, s_{max}]$. We assume all objects are placed upright on the known ground plane, such that the only variable rotation is around the vertical axis, parametrized by θ_i.

Both the object- and background encoders g_o and g_{bg} are convolutional neural networks, consisting of several convolutional layers of kernel size (3, 3) and strides of (1, 1), each followed by ReLU activations and (2, 2) max-pooling. Then the final encodings are obtained through fully-connected layers. Both the shape- and texture-decoders are fully-connected neural networks, similar to [Park et al., 2019].

Scene Decoding.
From an object's latent vector $\mathbf{z}_i$ produced by the encoder, the fully differentiable object-wise renderer f of our model computes for each of these vectors its reconstruction $(\hat{I}_i, \hat{D}_i, \hat{M}_i) = f(\mathbf{z}_i)$, consisting of its predicted image $\hat{I}_i$, depth $\hat{D}_i$, and mask $\hat{M}_i$. For each pixel $\mathbf{u} \in \mathbb{R}^2$ in image coordinates, its depth is determined through raycasting into the SDF defined by $\Phi(\mathbf{z}_{i,sh}, \cdot)$. As in [Wang et al., 2020], we approximate the first SDF zero-crossing $\mathbf{x}(\mathbf{u})$ (in world coordinates) by sampling in uniform steps along the ray between a minimum and maximum depth. Then the depth $\hat{D}_i(\mathbf{u})$ is given by the z-coordinate of $\mathbf{x}(\mathbf{u})$ if a zero crossing was found for pixel $\mathbf{u}$ and is set to a large constant otherwise. The occlusion mask $\hat{M}_i$ is set to 1 if for a pixel a zero-crossing was found and to 0 otherwise. Similarly to the shape decoder Φ, the texture decoder Ψ is parametrized by the texture latent vector $\mathbf{z}_{i,tex}$ and outputs an RGB color value for an input query point. To color the pixels and obtain the object rendering $\hat{I}_i$, for each pixel $\mathbf{u}$ where $\hat{M}_i(\mathbf{u}) = 1$, we query $\Psi(\mathbf{z}_{i,tex}, \cdot)$ at the equivalent point of $\mathbf{x}(\mathbf{u})$ in object coordinates.

We then combine these object-wise renderings $(\hat{I}_i, \hat{D}_i, \hat{M}_i) = f(\mathbf{z}_i)$, along with the background color $\mathbf{z}_{bg}$ regressed by the background encoder into $(\hat{I}_{1:n}, \hat{D}_{1:n}, \hat{M}_{1:n}) = F(\mathbf{z}_{bg}, \mathbf{z}_1, ..., \mathbf{z}_n)$, for $n \leq N$ through z-buffering: For each pixel $\mathbf{u}$, the combined results $\hat{I}_{1:n}, \hat{D}_{1:n}, \hat{M}_{1:n}$ are set to

the corresponding values of the object-wise results for the object with the smallest depth at **u**.

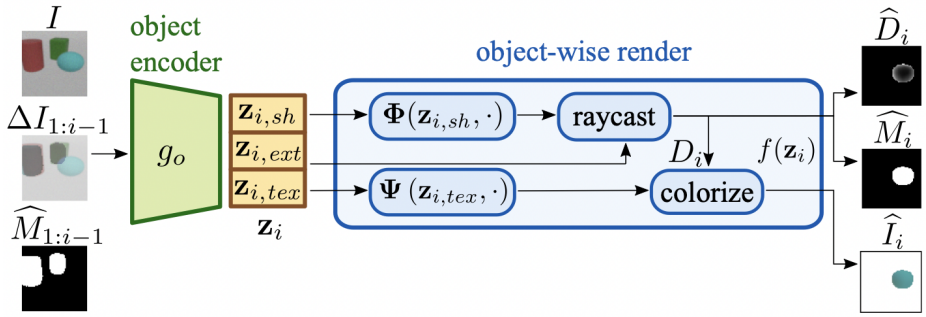

Figure 2: **Encoding and rendering of an object in a single slot of the model.** (Figure taken from [Elich et al., 2021]). The input image, difference image and combined mask based on the previously reconstructed objects are passed into the object encoder g_o. Its output latent vector $\mathbf{z}_i$ describing the next object is comprised of the extrinsics $\mathbf{z}_{i,ext}$, shape $\mathbf{z}_{i,sh}$, and texture latents $\mathbf{z}_{i,tex}$. The latter two of those latents parametrize the shape- and texture-decoders Φ and Ψ. From the output of Φ and the extrinsics latent, the object depth and mask are raycast. The colors of the pixels are obtained by sampling Ψ at the raycast positions.

Training.
The training procedure for this model is split into two stages: First, as in [Park et al., 2019], we train the DeepSDF shape decoder network Φ in a supervised manner on query points in a volume and a dataset of meshes similar to the objects to be reconstructed later. Then, in a second stage, this pre-trained DeepSDF model is used to learn the encoding and decoding tasks of the entire model. The exact loss functions used during training are detailed in Chapters 3.3.1 and 3.3.2.

3.2 Render-and-Compare Optimization

Taking the latent scene representations from the Deep Learning model detailed in Chapter 3.1 as initializations, the aim of the Render-and-Compare optimization is to make small adjustments to these scene representations that improve the accuracy of the reconstruction.

In 3.2.1, a theoretical overview of the process is provided, while 3.2.2 dives into the implementation details. Chapter 3.2.3 shows an alternative way to perform this optimization.

3.2.1 Abstract Overview of the Optimization Process

For each object i in the input scene, this optimization process manipulates the latent vector z_i of that object in the following manner: The object's latent vector is initialized, either by taking the output of the model's encoder as outlined in Chapter 3.1 or by sampling it randomly from a uniform distribution around the zero-vector (the motivation and reasoning behind this approach is discussed in Chapter 3.2.3).

Once this initial latent vector $z_i^{(0)}$ is obtained, the Render-and-Compare optimization updates it until either the loss term converges or up to a maximum number of iterations. In each iteration T, this heuristic performs the following three steps:

1) *The Render Step*: Here, the latent vector $z_i^{(T)}$ is passed through the model's decoding- and rendering-pipeline in order to obtain the pictorial expressions of it, based on which the reconstruction losses can be computed.
2) *The Compare Step*: Here, the object rendering produced in the first step is compared to the input image by means of computing the reconstruction losses: To that end, the remaining, not yet optimized objects $i+1, i+2, ...$ are also rendered via simple forward propagation steps through the Neural Network. Finally all object renderings are combined into the final output image by means of Depth-Ordering of the objects and the reconstruction loss is computed.
3) *The Update Step*: The reconstruction loss L computed in the Compare Step is passed to the optimizer which computes $\frac{\partial L}{\partial z_i^{(T)}}$ and performs a Gradient Descent step to obtain the updated latent vector $z_i^{(T+1)}$.

Upon conclusion of this optimization process, the final, optimized latent vectors are stored alongside their reconstructed images.

3.2.2 Implementation Details

In detail, the Render-and-Compare optimization process is the following:

First, the neural network model whose outputs are to be optimized, is loaded and its weights are restored. After this setup phase, the optimization of the scenes in the dataset provided begins. Every scene in the dataset is optimized in the following way.

Each input scene in the dataset contains the input RGB image of the scene, alongside the ground-truth depth map and instance masks. In the Render-and-Compare optimization, we

only utilize the RGB input image and assume the ground-truth depth- and mask-data is unavailable. The optimization parameters define the weights for the different losses needed to compute the total loss that we seek to minimize in this optimization. Additionally they also contain the standard deviation defining the amount of Gaussian blur to apply in the computation of the reconstruction loss. This blur parameter is set to zero during the normal optimization process. The exact losses and their weights used here are detailed in Chapters 3.3.1, and 4.4.

The batch size for the optimization is set to one, i.e., each scene is processed one after the other. This is necessary since each object in each scene needs its individual number of Gradient Descent optimization steps until its loss converges.

In the scene-wise optimization, lists are defined that hold the latent vectors, object masks, and difference images for the object slots that have already been optimized. Then, for each object slot, the contents of these list are combined to provide the object encoder with the correct inputs to be able to generate the initialization of the latent vector for the current object. This initial latent vector is computed by a forward pass for the slot input through the object encoder. The vector is then passed as input to the object-wise optimization method which performs gradient descent updates on the latent vector until convergence and then returns it. Finally, that resulting optimized latent is rendered to generate the object mask and difference image for this slot that are then appended to the list mentioned above. These updated lists are subsequently used as input for the next slot.

The object-wise optimization method starts by rendering the object defined by the initial latent for the current slot. It also predicts and renders the objects in the remaining subsequent slots based on the rendering results for the current object with a forward pass through the network. These object renders are all necessary since the combined RGB prediction for the entire scene is required to compute the loss value for the current latent vector. Based on this loss value, the gradient w.r.t. the current latent vector is computed and a gradient descent step is performed to obtain the updated latent vector. In the gradient descent steps the shape-, texture- and extrinsics-parts of the latent vector are updated with their individual learning rates. Starting from this updated latent, the current object is rendered again and the subsequent objects are predicted to generate the next loss value. This process is repeated until either a maximum number of iterations or until the difference between the previous and the current loss value is less than some small parameter. At this point, we consider the loss converged and the method returns the final version of the latent vector.

This method offers the options of optimizing the shape-, texture-, and extrinsics-parts of the latent vectors individually and sequentially. However, we found this to be inefficient and unnecessary.
Furthermore, it is also possible to define a set of different smoothing kernel sizes to be applied in the loss computation. In that case, each object is optimized until convergence for the first kernel size. Then that resulting latent vector is taken as input for the optimization with the second kernel size. That process is repeated for all sizes in that set. This is sensible for weak models that struggle with precise object placement and sizing and allows to decrease the

amount of smoothing over the optimization, similarly to how the model is trained. Except for the situation described in Chapter 3.2.3, however, this was not used.

3.2.3 Optimization of Latents Initialized as ϵ-Vectors

Since the model used originally often failed to represent even large and obvious objects, we experimented with optimization of latent vectors initialized around zero.

If the model's object encoder failed to represent an object, it would often place it completely outside the camera's view, leading to a situation where such objects could not be optimized, as they would not contribute to the reconstruction loss. An object with a latent vector initialized closely around zero would be visible to the camera and could thus be optimized. Therefore, in this variant of the Render-and-Compare optimization, we use such a vector as initialization for an object's latent vector instead of the encoder output.
As an exact zero-vector would not lead to valid gradient, the entries of these vectors were initialized as ϵ, where ϵ is sampled uniformly at random from $[-1 \cdot 10^{-10}, 1 \cdot 10^{-10}]$. Whenever the total loss for the initialization of a slot exceeded a loss threshold, such a vector would be optimized instead of the model output for that slot. In this case, we also make use of the possibility to define a set of several smoothing kernel sizes mentioned in Chapter 3.2.2. For each of these smoothing intensities, we optimize the latent vector of the current slot and use that optimized vector as input for the optimization with the next kernel size. In our implementation we used three such kernel sizes with decreasing smoothing intensity.

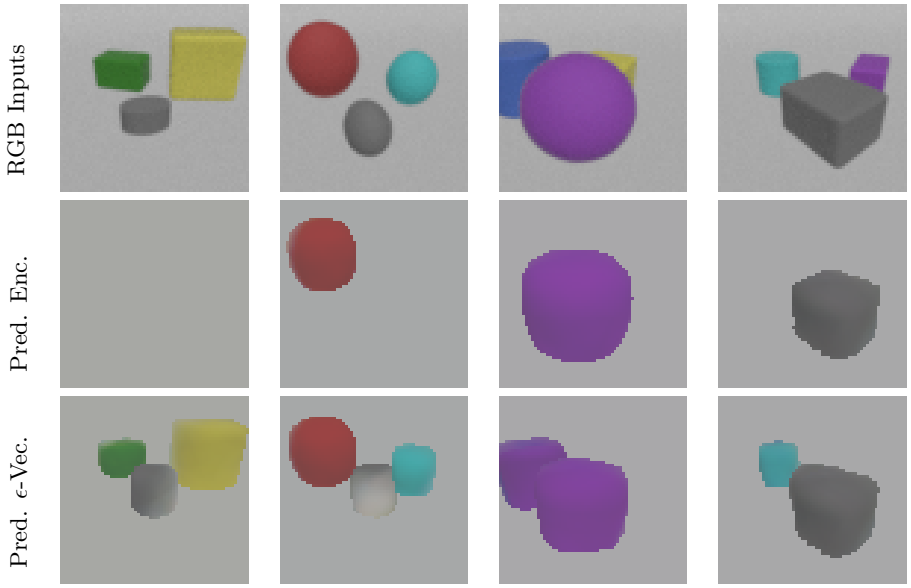

Figure 3: **Qualitative results on test set.** Each row shows the RGB input, the optimization result with encoder output as initialization, and the optimization result with an ϵ-vector as initialization. The much improved object detection performance of the optimization from ϵ-vectors can be clearly seen, but the reconstruction quality of the newly detected objects is poor.

As can be seen in Figure 3, the Render-and-Compare optimization from ϵ-vectors is able to detect significantly more objects. However, the reconstructions of the newly detected objects are quite rough: In many cases it represents the objects' scales, positions, and colors correctly, but despite the fact that in this case the learning rate for the shape updates is nonzero (0.01) and no shape regularization is applied, most objects are only reconstructed as mean shapes. In the first two scenes, it can also be seen how this method leads to artifacts: In both cases, there is a gray mean shape object in the image center. Most likely, this is the representation of a ϵ-latent vector which did not get matched to a ground truth object correctly and was only optimized to blend into the scene background.

While this method did lead to some improvement in object detection, it was only able to correctly reconstruct the basic aspects of object position, color and size. Shape, rotation, and texture details were only reconstructed very poorly, if at all. This made it quite apparent that for good optimization results, an encoder with reliable object detection and precise object reconstruction is absolutely necessary. The extent to which the Render-and-Compare optimization can improve a reconstruction is considerable, as will become apparent in the evaluation in Chapter 5, but also clearly has its limits, as demonstrated here.

Therefore, an improved model, especially in terms of object detection, was required. The additions made to the model in order to achieve that, are laid out in Chapters 3.3.2 and 3.3.3.

3.2.4 Optimization with Anti-Aliasing

Since the input images in the dataset are rendered with anti-aliasing, we sought to leverage that additional information by incorporating anti-aliasing into the model's rendering pipeline, as well. In particular, we hoped that this method would lead to slightly refined object placement and rotation as the reconstructions can now match the input image more closely. Additionally it was clear that this method would yield significantly improved RGB reconstructions and smoother object edges.

This method implements anti-aliasing in the following way. The differentiable renderer now outputs twice the image-, depth-map-, and mask-size, i.e., (128, 128) instead of (64, 64) as before. The combined higher-resolution images showing the depth-ordering of the reconstructed objects are then sub-sampled to produce images of size (64, 64) with 2x anti-aliasing, the new model output. The same procedure is also applied to the generated depth-maps and instance masks.

Since internally, this leads to 4 times as many pixels to be rendered, it also slows down the back-propagation through the network by roughly a factor of 4. Therefore, we decided not to train a completely new model with this method, as the training time would have been unfeasably long. We did however apply it in the Render-and-Compare pipeline, where the performance degradation was still acceptable. The results of this method can be seen in Chapter 5.4.

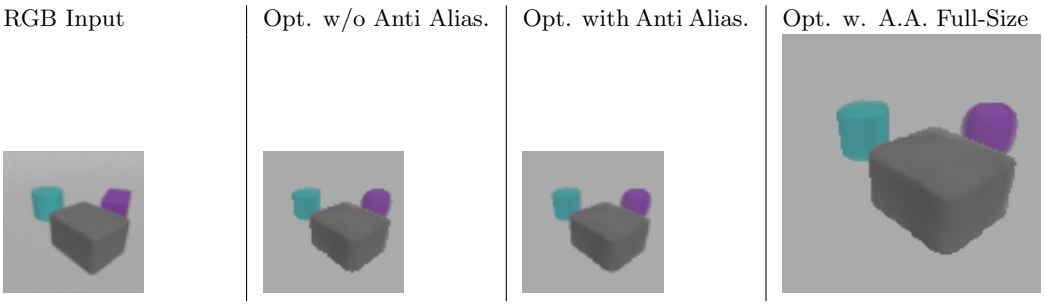

Figure 4: **Qualitative results of optimization with anti-aliasing on test set.** An example RGB input and the model's reconstruction without and with anti-aliasing are shown, as well as the internal higher-resolution full-size rendering. The effect of anti-aliasing can be seen well around the objects' edges.

3.3 Loss Functions

Chapters 3.3.1 and 3.3.2 describe the loss functions used in the different model configurations, Chapter 3.3.3, explains the fine-tuned loss weight schedules required to achieve the results presented later.

3.3.1 Overview: Reconstruction Losses, Regularizers, Ground Loss, Intersection Loss

The original self-supervised model, as presented in [Elich et al., 2021], used the following four sub-losses, whose weighted sum then defined the overall total loss function of the model:

Let $\hat{I}_{1:N}, \hat{D}_{1:N}, \hat{N}_{1:N}$ denote the predicted RGB image, depth map, and normal map for the combined scenes from all N object slots of the model. Furthermore, let I_{gt}, D_{gt}, N_{gt}, denote the ground-truth image, depth, and normal map, let Ω be the set of image pixels, and $G(\cdot)$ refer to the Gaussian Smoothing operation with the appropriate standard deviation for the current training epoch:

- **RGB reconstruction loss:** This is an L2-loss defined between the predicted and the ground-truth RGB images, after the same Gaussian smoothing has beed applied to both. The loss formula is the following:

$$L_{RGB} = \frac{1}{|\Omega|} \sum_{\mathbf{u} \in \Omega} \left\| G(\hat{I}_{1:N})(\mathbf{u}) - G(I_{gt})(\mathbf{u}) \right\|_2^2 \tag{1}$$

- **Depth reconstruction loss:** This L1-loss is defined between the smoothed predicted and ground-truth depth maps. The loss formula is the following:

$$L_D = \frac{1}{|\Omega|} \sum_{\mathbf{u} \in \Omega} \left\| G(\hat{D}_{1:N})(\mathbf{u}) - G(D_{gt})(\mathbf{u}) \right\|_1 \tag{2}$$

- **Ground loss:** This loss seeks to penalize objects intersecting the ground plane. It achieves this goal through the following, simplified measure: For each object i in the

scene, let $\mathbf{c}_i$ denote the coordinate of its center, and let $\mathbf{c}'_i$ refer to the projection of this center coordinate onto the ground plane. Furthermore, $c_{i,z}$ denotes the z-coordinate of object i's center, i.e., its height above the ground plane, and $\Phi(\mathbf{z}_{i,sh}, \mathbf{o}(\mathbf{c}'_i, \mathbf{z}_{i,ext}))$ refers to the value of the SDF defined by the shape latent $\mathbf{z}_{i,sh}$ evaluated at $\mathbf{o}(\mathbf{c}'_i, \mathbf{z}_{i,ext})$, the mapping of $\mathbf{c}'_i$ from world coordinates into object coordinates, based on object i's position latent vector $\mathbf{z}_{i,ext}$. Then, the ground loss is defined as:

$$L_{gr} = \frac{1}{N} \sum_{i=1}^{N} \Big(\max(0, -c_{i,z}) + \max(0, -\Phi(\mathbf{z}_{i,sh}, \mathbf{o}(\mathbf{c}'_i, \mathbf{z}_{i,ext}))) \Big) \tag{3}$$

- **Object regularization loss:** This regularizer penalizes large vector entries in the shape and texture latents $\mathbf{z}_{sh}$ and $\mathbf{z}_{tex}$ and thus forces the reconstructed shapes towards a mean shape and texture:

$$L_{reg} = \sum_{i=1}^{N} \|\mathbf{z}_{i,sh}\|_2^2 + \|\mathbf{z}_{i,tex}\|_2^2 \tag{4}$$

- **Normal reconstruction loss:** This is an L2-loss defined over the normals belonging to all pixels in the image. It differs from the RGB- and depth reconstruction losses in its lack of Gaussian Smoothing. We do not apply smoothing in this loss since smoothed normals would lead to incorrect shape reconstructions, particularly around the objects' edges. Since the pixel-wise normals $N(\mathbf{u})$ are neither a direct output of the model nor part of the ground-truth data, they are computed as $N(x, y) = (\frac{\partial D(x,y)}{\partial x}, \frac{\partial D(x,y)}{\partial y})^\top$, the directional derivatives of the values in the depth map. The loss formula is the following:

$$L_N = \frac{1}{|\Omega|} \sum_{\mathbf{u} \in \Omega} \left\| \hat{N}_{1:N}(\mathbf{u}) - N_{gt}(\mathbf{u}) \right\|_2^2 \tag{5}$$

- **Object intersection loss:** This loss attempts to measure in a simplified manner if a pair of objects in the scene intersect each other. To that end, for each pair (i, i') of objects, a number C of coordinates along the connecting line between their centers are evaluated. If the objects do not intersect in the vicinity of this line, then for each point $\mathbf{p}$ on the line it must hold that $\Phi(\mathbf{z}_{i,sh}, \mathbf{o}(\mathbf{p}, \mathbf{z}_{i,ext})) + \Phi(\mathbf{z}_{i',sh}, \mathbf{o}(\mathbf{p}, \mathbf{z}_{i',ext})) \geq 0$. If that is not the case, then $\mathbf{p}$ is either inside both objects or farther inside one object than it is outside of the other. This leads to the following formula for this loss:

$$L_{int} = \frac{1}{C \frac{N(N-1)}{2}} \sum_{(i,i')} \sum_{\mathbf{p} \in l_{i,i'}} \max\Big(0, -\Phi(\mathbf{z}_{i,sh}, \mathbf{o}(\mathbf{p}, \mathbf{z}_{i,ext})) - \Phi(\mathbf{z}_{i',sh}, \mathbf{o}(\mathbf{p}, \mathbf{z}_{i',ext}))\Big) \tag{6}$$

, where $l_{i,i'}$ refers to the set of 10 points along the line between the centers of objects i and i' to be evaluated.

3.3.2 Silhouette Reconstruction Losses

As mentioned in Chapter 3.2.3, the original model trained only with the four loss functions detailed in Chapter 3.3.1 exhibited very unreliable object detection, in particular for smaller

objects (in terms of their pixel count). To improve this aspect of the network, we added a novel kind of loss to the overall loss function of the model, which leads it to focus on the outer edges / silhouettes of the objects in RGB-, depth-, and later normal reconstruction, the image regions most critical to reliable object reconstruction. Hereafter these losses are referred to as *silhouette losses*.

To achieve this focus on the silhouette regions of the objects, for each input image, a mask is needed, indicating which pixels of the input belong to the critical silhouette region. In the first version of the implementation of these losses, this mask was obtained in the following way: First, we would apply edge detection on the ground-truth depth map using a Sobel edge detection convolution kernel. Then the binary mask was computed by thresholding the resulting activation map. In order to capture some more context around these edges, we applied a (3, 3) dilation to this map. Although in this case with the uniformly gray background, the edge detection would have worked better by applying it to the RGB image rather than the depth map, we decided to compute the mask based on the depth data, as that would be robust also to more irregular background textures.
With a threshold of 5, this first method of extracting the edges from the depth map led to sets of edge pixels like the one shown in Figure 5. It can be seen that this first method is able to capture almost the entire silhouettes of the objects, except for their edges close to the ground, since there the depth values of the objects are almost the same as the depth of the ground plane.

In order to capture the entire silhouettes of the objects, we additionally also extracted edges from the normal maps derived from the depth data as their directional derivatives. Here, the regions of the silhouettes that previously led to little activation of the edge detection kernel, produce a large response. With a threshold of 5 applied to the normal edges, we were able to obtain sets of edge pixels like the one shown in the lower row in Figure 5. In that raw edge set from the ground-truth normal map, there is one edge we needed to remove as it does not belong to an object: It is located where the ground plane meets the set of pixels with the clipped maximum depth value. We eliminated this unwanted edge by removing all pixels with depths close to that maximum depth value from the edge set. Finally we combine the edges from depth and normal data by taking the pixel-wise maximum of both sets and obtain the final silhouette mask from that combined set with a (3, 3) dilation.
In Figure 5 it becomes apparent that beyond the silhouettes of the objects, this method for computing the masks also captures hard edges within the objects, particularly within cuboids. While this goes beyond the required set of pixels for improved object detection, it is actually desirable since an exact depth reconstruction in these high-curvature regions of the objects implies a good shape reconstruction.

Since we want the impact of the silhouette losses to be roughly equal regardless of the number of pixels in the silhouette mask for a given input, we finally normalize the mask by dividing its entries by the number of pixels in the mask.

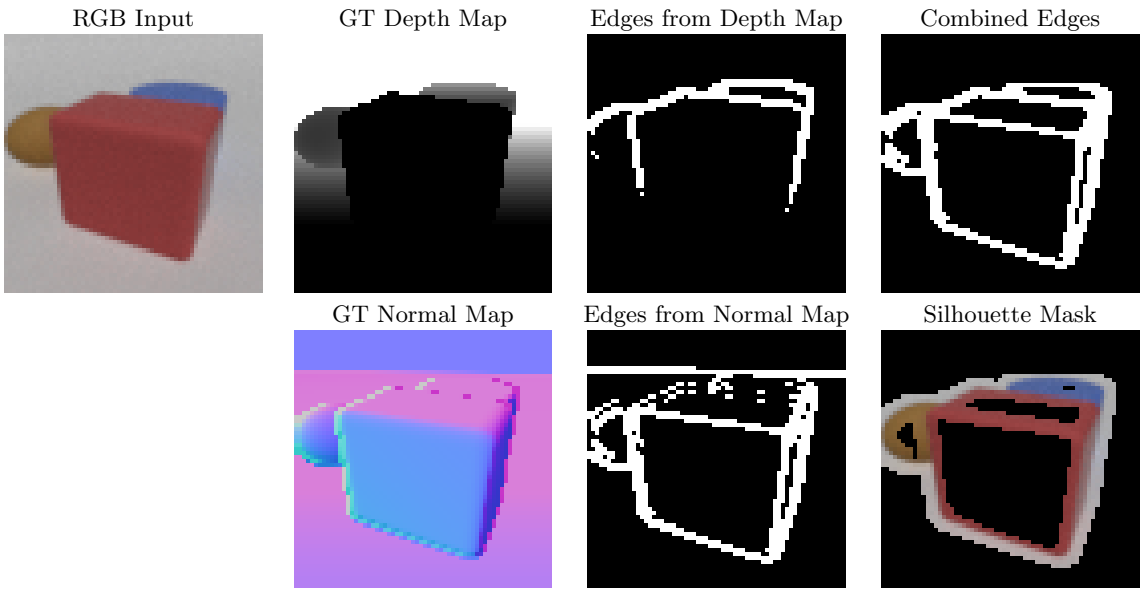

Figure 5: **Silhouette mask implementation.** The edges extracted from the ground-truth depth and normal maps are combined into the set of edge pixels that after dilation defines the silhouette mask. The final silhouette mask is shown applied to the input RGB image.

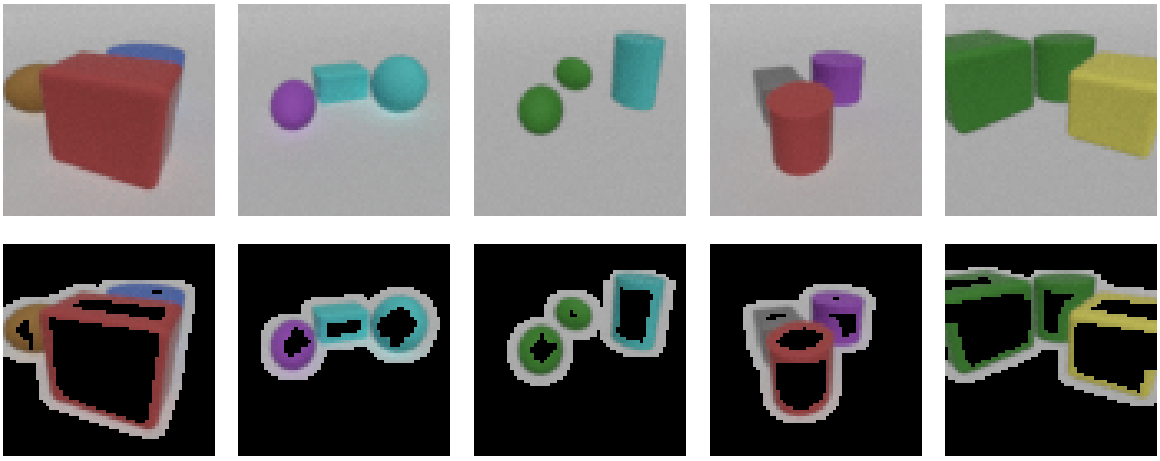

Figure 6: **Examples of silhouette masks.** Top row: input images; Bottom row: application of corresponding silhouette masks

Based on the three reconstruction losses detailed in Chapter 3.3.1, we introduced these three new silhouette losses as additional parts of the model's overall loss function:
Let $M(\mathbf{u})$ refer to the binary silhouette mask before normalization at pixel position $\mathbf{u}$, and let $s(M) = \sum_{\mathbf{u}} M(\mathbf{u})$, i.e., the number of silhouette pixels in the image. Then the exact formulae for the silhouette losses are the following:

- **RGB silhouette reconstruction loss:**

$$L_{RGB,sil} = \frac{1}{|\Omega|} \sum_{\mathbf{u} \in \Omega} \left\| \frac{1}{s(M)} M(\mathbf{u}) \Big(G(\hat{I}_{1:N})(\mathbf{u}) - G(I_{gt})(\mathbf{u}) \Big) \right\|_2^2 \tag{7}$$

- **Depth silhouette reconstruction loss:**

$$L_{D,sil} = \frac{1}{|\Omega|} \sum_{\mathbf{u} \in \Omega} \left\| \frac{1}{s(M)} M(\mathbf{u}) \Big(G(\hat{D}_{1:N})(\mathbf{u}) - G(D_{gt})(\mathbf{u}) \Big) \right\|_1 \tag{8}$$

- **Normal silhouette reconstruction loss:**

$$L_{N,sil} = \frac{1}{|\Omega|} \sum_{\mathbf{u} \in \Omega} \left\| \frac{1}{s(M)} M(\mathbf{u}) \Big(\hat{N}_{1:N}(\mathbf{u}) - N_{gt}(\mathbf{u}) \Big) \right\|_2^2 \tag{9}$$

The motivation for these new losses is the fact that previously, the simple RGB and depth losses did not penalize wrong, or missing reconstructions of small objects enough. This led to many instances where large objects (in terms of their number of pixels in the image) would be reconstructed faithfully, whereas smaller objects would often simply not be reconstructed at all. The introduction of silhouette losses addresses this issue, as the number of silhouette pixels is roughly linear in the diameter of an object, while its overall number of pixels is quadratic. This can be seen well in Figure 6, where only a small fraction of the pixels belonging to large objects is part of the silhouette masks, while almost all pixels of the smaller objects are also silhouette pixels. Furthermore, the normalization by the number of silhouette pixels present in an image additionally levels the impact of an object's size. Since larger objects still lead to more silhouette pixels compared to smaller ones, this has the effect that for scenes containing only small objects, the impact of the silhouette loss is approximately the same as for scenes with large objects.

3.3.3 Loss Weights and Smoothing Kernel Schedules

For the different losses detailed in Chapters 3.3.1 and 3.3.2, we found empirically these loss weights / loss weight schedules to work well:

Subsequently, let e refer to the number of the current training epoch.

- RGB Loss Weight:

$$\lambda_{RGB} = 1.0 \tag{10}$$

- RGB-Silhouette Loss Weight:

$$\lambda_{RGB,sil} = 4000000.0 \tag{11}$$

- Depth Loss Weight:

$$\lambda_{D} = 0.1 \tag{12}$$

- Depth-Silhouette Loss Weight:

$$\lambda_{D,sil} = 50.0 \tag{13}$$

- Normal Loss Weight:

$$\lambda_N = \begin{cases} 0.0 & \text{if } e < 150 \\ 5.0 \cdot \frac{e-150}{50} & \text{if } 150 \leq e < 200 \\ 5.0 & \text{otherwise} \end{cases} \tag{14}$$

- Normal-Silhouette Loss Weight:

$$\lambda_{N,sil} = \begin{cases} 0.0 & \text{if } e < 150 \\ 10000000.0 \cdot \frac{e-150}{50} & \text{if } 150 \leq e < 200 \\ 10000000.0 & \text{otherwise} \end{cases} \tag{15}$$

- Ground Loss Weight:

$$\lambda_{gr} = 0.01 \tag{16}$$

- Intersection Loss Weight:

$$\lambda_{int} = \begin{cases} 0.0 & \text{if } e < 150 \\ 0.001 \cdot \frac{e-150}{50} & \text{if } 150 \leq e < 200 \\ 0.001 & \text{otherwise} \end{cases} \tag{17}$$

- Object Regularization Loss Weight:

$$\lambda_{reg} = \begin{cases} 0.025 - \frac{e}{400} \cdot 0.0225 & \text{if } e < 400 \\ 0.0025 & \text{otherwise} \end{cases} \tag{18}$$

For the six reconstruction losses on RGB, depth, and normals, the loss weights were chosen so that these losses all have roughly the same impact on the total loss. The silhouette losses need larger magnitude weights to compensate for the on average small normalization fractions $\frac{1}{s(M)}$ in these loss functions. Among the silhouette losses, for the depth-silhouette loss, a smaller magnitude weight is sufficient as it is an L1-loss while the other two are L2-losses.

The training-epoch-dependent loss weight schedules for the two normal losses are necessary due to the lack of Gaussian Smoothing in these loss functions. If the final weights for these losses were applied from the beginning of training, it would lead to results like the ones shown in Figure 7.

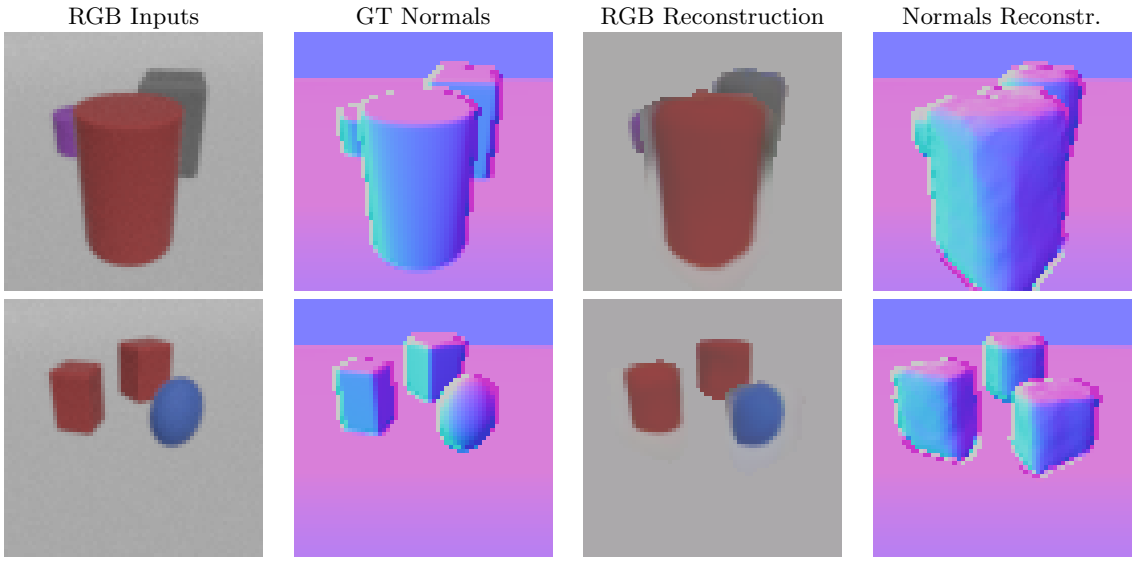

Figure 7: **Qualitative results on test set from model trained without variable normal loss weights.** Both the ground-truth and predicted images and normal maps are shown.

From the qualitative results in Figure 7, it becomes clear that in this case, the model failed to learn the relationship between the smoothed RGB and depth images, and the unblurred normal maps. That has the effect that the network predicts object normals that have little connection to the actual shapes of the objects, and at the same time tries to minimize the RGB losses by adapting the textures to replicate the input image.

The reasoning behind the gradual increase of the intersection loss weight is the fact that the strongly blurred input images during the early epochs of training will very likely lead to some objects intersecting as the model is not yet able to position and scale them precisely. We do not penalize this yet since during this early training phase, the model should focus on learning to detect objects reliably, rather than reconstructing them precisely. To that end, the intersection loss is not required.

The motivation for the regularization loss schedule is a similar one to the intersection loss: During early training, the model should only learn to detect objects well, while the exact

shape and texture is not yet relevant. Crucially, it prevents the early model from generating shapes that lie outside the realm of shapes the SDF network was trained on. The decreased regularization weight later during training allows the model to predict shapes and textures that deviate more strongly from the mean.

The schedule for the standard deviation in the Gaussian Smoothing kernel is:

$$\sigma_G = \begin{cases} \frac{16}{3} - \frac{e}{200} \cdot (\frac{16}{3} - 0.5) & \text{if } e < 200 \\ 0.5 & \text{otherwise} \end{cases} \tag{19}$$

Similarly to the intersection and regularization weight schedules, the application of this decreasingly strong Gaussian Smoothing allows the network to focus on the necessary basics first, before learning the more advanced tasks: With a strong blur, the focus is on the tasks of object detection as well as size-, position-, and object-color-reconstruction, whereas later during training, with only minimally smoothed images, the model can learn to represent exact shapes, texture details and object orientations. The ablation study in [Elich et al., 2021] shows that training the model in this coarse-to-fine manner is absolutely essential: Without smoothing, the model is barely able to produce meaningful results at all.

3.4 Supervised Training with Optimized Latents

Since the Render-and-Compare optimization is very computationally expensive compared to a simple forward pass through a Neural Network and the time until convergence of the optimization is not constant, we devised the following procedure:

We would utilize the Render-and-Compare process to generate a dataset of optimized *Pseudo-Ground-Truth Latent Vectors*, which could in turn be used to train the encoder of the network. With this method, we hoped to be able to transfer at least some of the improvement the optimization achieved over the plain model into the model itself.

In detail, the method was as follows:
Since the encoder of the model works on single objects rather than entire scenes, we created a dataset consisting of input-output pairs like the following. As input we would pass the encoder the same input as during the usual operation of the model: The input image and a difference image and segmentation mask indicating the objects already encoded. As pseudo-ground-truth output we would provide the optimized latent vector of the next object to encode.
We generated this dataset based on the training set of our model. Therefore, in this case, in order to produce the highest-quality pseudo-ground-truth latent vectors possible, it is necessary to utilize the ground-truth depth maps in the Render-and-Compare optimization, as well. To this end, in the optimization we used the same losses as during training with their weights set to the same values as in the last epoch of training.

Unfortunately, in practice, we were unable to achieve an improved performance of the model with this approach. We do believe, however, that theoretically, this method is sound and the issue is with our implementation, rather than it being unviable.

4 Data, Configurations, and Evaluation Metrics

4.1 The Clevr Dataset

We evaluate our model on a variant of the CLEVR dataset [Johnson et al., 2017]. This dataset contains renderings of simple objects arranged in random poses on a ground plane in 3D-space.

Utilizing the dataset generation implementation for CLEVR, we produced a dataset of 11000 scenes. Each of those scenes contains three objects, although in same instances, complete occlusions of objects are possible, such that in those cases only one or two objects are actually visible. The objects themselves all belong to one of three classes of shapes: ellipsoids, cylinders, and cuboids. In each scene, the objects are arranged at random positions and with random rotations on a ground plane, with the limitations that no two objects intersect each other and that all objects lie within the camera's view.

For each such scene, the dataset contains the RGB rendering of the scene and the following ground-truth data: The depth map of the scene, an instance mask and the pose for each object within it, as well as as the 3D geometry for the entire scene in an OBJ-file. The images, depth maps, and instance masks all have a resolution of (64, 64). The OBJ-file is needed to compute the voxel sets for the 3D shape evaluation.
We used a training/validation/test split of the dataset of 9000 scenes for training and 1000 scenes each for validation and testing.

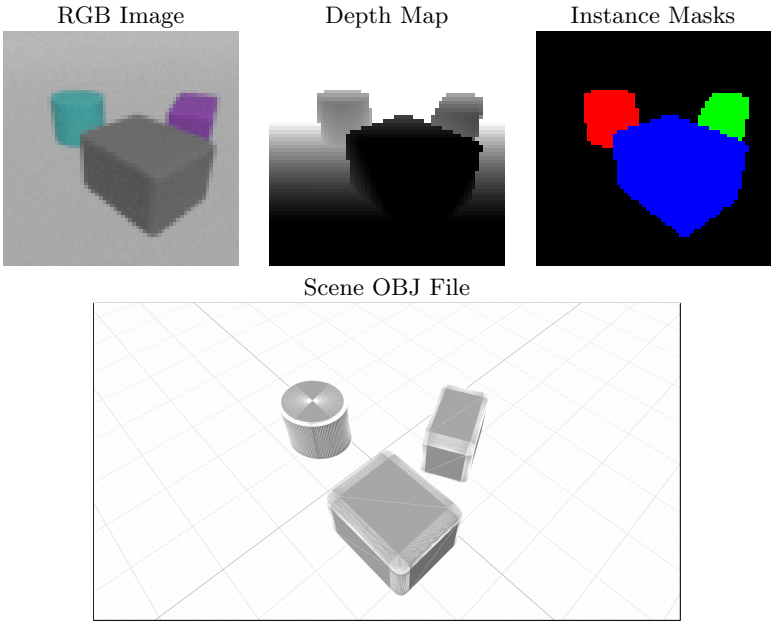

Figure 8: **Contents for example scene in our variant of the CLEVR dataset.**

4.2 Reconstruction Metrics

In Chapters 5.2 to 5.5, we report the following reconstruction metrics. Unless otherwise specified, these are the same as in [Elich et al., 2021].

Instance Reconstruction Metrics
For each predicted object i in a scene with instance mask $\hat{M}_i$, we consider it correctly detected, i.e., a *True Positive (TP)* if in that scene there exists a ground-truth instance mask $M_{gt,j}$ for an object j such that the Intersection-over-Union of the pixel sets $IoU(\hat{M}_i, M_{gt,j}) \geq \tau$ for a threshold τ. If no ground-truth mask fulfills this property for a predicted object, we call it a *False Positive (FP)*. Ground-truth objects for which none of the reconstructions satisfies the property above are considered *False Negatives (FN)*. As mentioned in Chapter 4.1, not all objects in a scene are necessarily visible. Hence, we only consider objects whose masks contain at least 25 pixels. For a scene with predicted and ground-truth masks $(\hat{M}_{1:N}, M_{gt})$, let the number of True Positives be denoted by $TP_\tau = TP_\tau(\hat{M}_{1:N}, M_{gt})$, the number of False Positives by $FP_\tau = FP_\tau(\hat{M}_{1:N}, M_{gt})$, and the number of False Negatives by $FN_\tau = FN_\tau(\hat{M}_{1:N}, M_{gt})$. Furthermore, we refer to the set of images over which we evaluate these metrics as $\mathcal{I}$ and define $\mathcal{T} := \{0.5, 0.55, ..., 0.95\}$.

- **Average Precision.**
  Here, we report the following two metrics:

$$AP_{0.5} = \frac{1}{|\mathcal{I}|} \sum_{(\hat{M}_{1:N}, M_{gt})} Prec_{0.5} = \frac{1}{|\mathcal{I}|} \sum_{(\hat{M}_{1:N}, M_{gt})} \frac{TP_{0.5}}{TP_{0.5} + FP_{0.5}} \tag{20}$$

$$mAP = \frac{1}{|\mathcal{T}|} \sum_{\tau \in \mathcal{T}} AP_\tau \tag{21}$$

- **Average Recall:**

$$AR_{0.5} = \frac{1}{|\mathcal{I}|} \sum_{(\hat{M}_{1:N}, M_{gt})} Rec_{0.5} = \frac{1}{|\mathcal{I}|} \sum_{(\hat{M}_{1:N}, M_{gt})} \frac{TP_{0.5}}{TP_{0.5} + FN_{0.5}} \tag{22}$$

- **F1-Score:**

$$F1_{0.5} = \frac{1}{|\mathcal{I}|} \sum_{(\hat{M}_{1:N}, M_{gt})} 2\frac{Prec_{0.5} \cdot Rec_{0.5}}{Prec_{0.5} + Rec_{0.5}} \tag{23}$$

In the interest of improved legibility, the subsequent metrics are all defined over single scenes, not over the entire dataset. The values reported for these are the averages over all scenes in the test set.

Image Reconstruction Metrics
For these metrics, we define $\Delta I(\mathbf{u}) = \hat{I}_{1:N}(\mathbf{u}) - I_{gt}(\mathbf{u})$ and L as the dynamic range of valid pixel intensities.

- **Root Mean Squared Error:**

$$RMSE(\hat{I}_{1:N}, I_{gt}) = \sqrt{MSE(\hat{I}_{1:N}, I_{gt})} = \sqrt{\frac{1}{|\Omega|} \sum_{\mathbf{u} \in \Omega} \|\Delta I(\mathbf{u})\|_2^2} \tag{24}$$

- **Peak Signal-to-Noise Ratio:**

$$PSNR(\hat{I}_{1:N}, I_{gt}) = 10 \log_{10} \frac{L^2}{MSE(\hat{I}_{1:N}, I_{gt})} \tag{25}$$

- **Structural Similarity Index:**
  For the explanation of the SSIM metric we refer to [Wang et al., 2004].

Depth Reconstruction Metrics
In the following, let $\Delta D(\mathbf{u}) = \hat{D}_{1:N}(\mathbf{u}) - D_{gt}(\mathbf{u})$.

- **Root Mean Squared Error:**

$$RMSE(\hat{D}_{1:N}, D_{gt}) = \sqrt{\frac{1}{|\Omega|} \sum_{\mathbf{u} \in \Omega} \|\Delta D(\mathbf{u})\|_2^2} \tag{26}$$

- **Absolute Relative Difference:**

$$AbsRD(\hat{D}_{1:N}, D_{gt}) = \frac{1}{|\Omega|} \sum_{\mathbf{u} \in \Omega} \frac{|\Delta D(\mathbf{u})|}{D_{gt}(\mathbf{u})} \tag{27}$$

- **Squared Relative Difference:**

$$SqRD(\hat{D}_{1:N}, D_{gt}) = \frac{1}{|\Omega|} \sum_{\mathbf{u} \in \Omega} \frac{\|\Delta D(\mathbf{u})\|_2^2}{D_{gt}(\mathbf{u})} \tag{28}$$

3D Reconstruction Metrics

- **Position Error:**
  Here, for each predicted object i with the position of its center $\mathbf{p}_i$, let $\mathbf{p}_{gt,j}$ denote the position of the closest ground-truth object. The position error is only computed over the set of True Positives, i.e., the set of predicted objects for which we have found a ground-truth match. By assigning the objects in a greedy manner, we make sure each ground-truth object is assigned only once. We denote the set of found position matches $(\mathbf{p}_i, \mathbf{p}_{gt,j})$ in a scene by P.

$$Err_{pos}(P) = \frac{1}{|P|} \sum_{(\mathbf{p}_i, \mathbf{p}_{gt,j}) \in P} \sqrt{\|\mathbf{p}_i - \mathbf{p}_{gt,j}\|_2^2} \tag{29}$$

- **Rotation Error:**

$$Err_{rot}(P) = \text{median}_{(\mathbf{p}_i, \mathbf{p}_{gt,j}) \in P} \left[\frac{360}{2\pi} \Delta r_{i,j} \right] \tag{30}$$

where $\Delta r_{i,j} = \min\left(|\theta_i - \theta_{gt,j}|, 2\pi - |\theta_i - \theta_{gt,j}|\right)$. All $\Delta r_{i,j}$ take the symmetry of the objects in the dataset into account, i.e., for some rotation θ, we consider $\theta + i \cdot \frac{\pi}{2}$ equivalent, for all $i \in \mathbb{N}$.

Beyond those metrics, we also report these novel metrics that evaluate the 3D geometry of the reconstructions:

- **3D IoU for entire scenes:**
  Here, we utilize the 3D information about the objects in the scenes from the OBJ-files stored with each scene in the dataset. We use the point clouds in these files, as well as the predicted SDFs from the shape decoder Φ to produce voxel sets for both the ground-truth scene and the predicted one. From the reconstruction, it is straightforward to compute this voxel set: If for one of the objects in the scene, the object-coordinate associated with the world coordinate of that sample point leads to an SDF-value ≤ 0, then we consider the voxel for that sample point to be occupied. For the ground-truth data, we first set all voxels occupied which contain a vertex of the point cloud inside them. These initial voxel sets only contain the shells of the objects. However, conveniently, since all objects in the CLEVR dataset are convex, it is easy to obtain the complete, filled voxel sets. These voxel sets are computed over the bounding box $[-3.0 : 3.0, -3.0 : 3.0, -1.0 : 2.0]$. The size of the voxel sets is (120, 120, 60). An example voxel set when evaluating over an entire scene can be seen in Figure 9.

  From these predicted and ground-truth voxel sets, we compute the Intersection-over-Union. For one scene, let $\hat{V}_{1:N}$ and V_{gt} denote the predicted and ground-truth voxel sets. Then, $I(\hat{V}_{1:N}, V_{gt})$ is the intersection and $U(\hat{V}_{1:N}, V_{gt})$ is the union voxel set. Furthermore, for a voxel set V, let $|V|$ refer to the number of occupied voxels. Then the formula for this metric is:

$$3DIoU_{scene}(\hat{V}_{1:N}, V_{gt}) = \frac{|I(\hat{V}_{1:N}, V_{gt})|}{|U(\hat{V}_{1:N}, V_{gt})|} \tag{31}$$

- **3D IoU for single objects:**
  Here, we match each predicted object to one of the ground-truth objects in the scene in a greedy manner, similarly to the procedure above for the position error. Objects for which no close match in terms of distance between the object center positions is found, we consider unmatched. For each match of objects, we compute the voxel sets for the reconstructed and the ground truth object in the object coordinate system with the respective rotations applied. In each scene, for each of these pairs of single-object voxel sets, we compute the 3D-IoU, analogously to Equation 31. To obtain the values reported in Chapter 5, we average these single-object 3D-IoU values over all valid pairs of reconstruction and ground-truth object in the dataset.
  Additionally, for this metric, we also report the total number of matched objects in

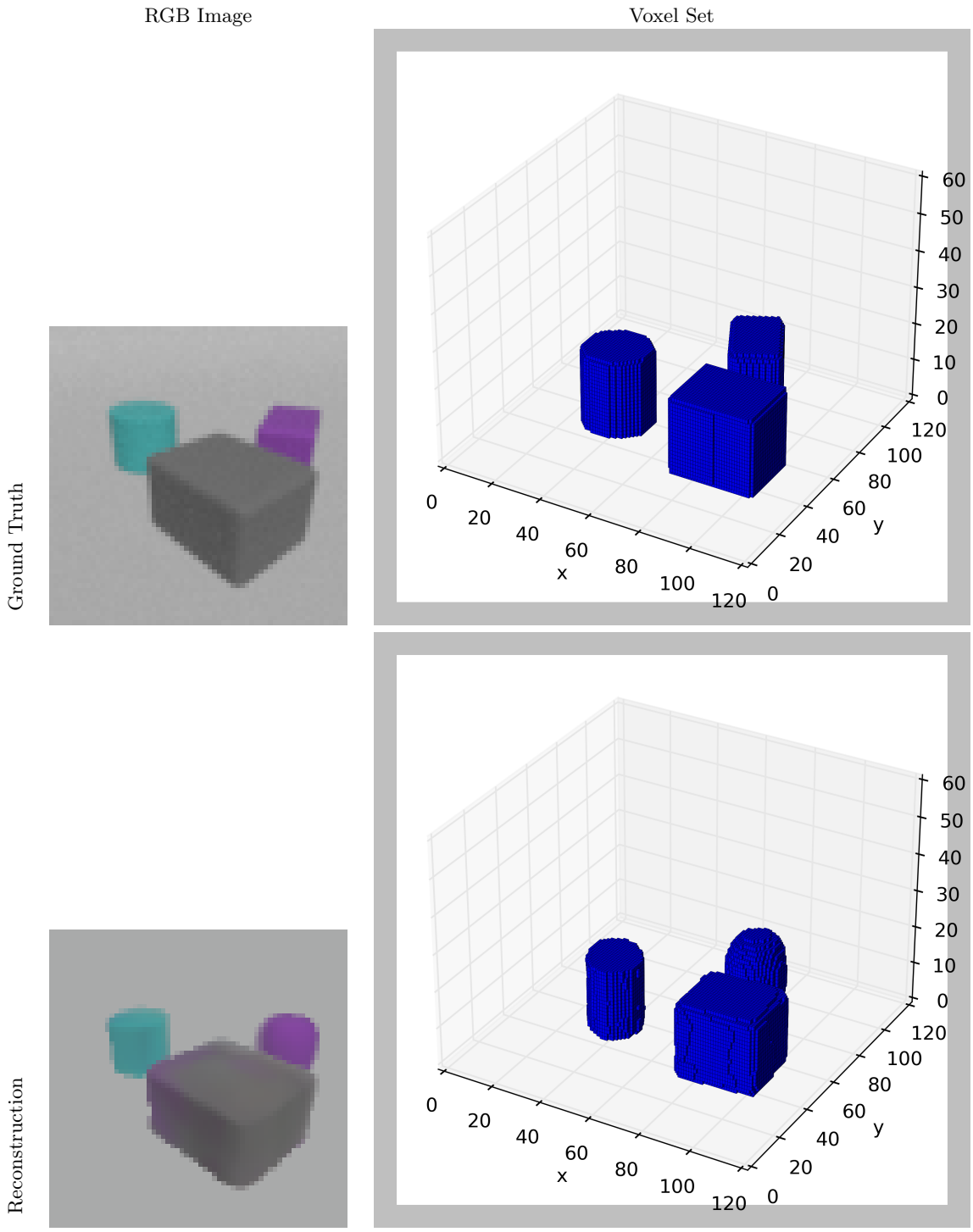

Figure 9: **3D voxel sets for example scene.** Here, the ground-truth and the predicted voxel set for an input image can be seen.

the dataset the metric is computed over. For 1000 scenes with three objects each, the maximum number of matched objects possible would be 3000. In practice, though, even for a perfect model, this number will be lower as not all objects in the dataset are visible in the images. Reporting this additional number adds crucial context, since for instance, a model with a poor recall could focus on reconstructing the easy objects very well and miss the remaining ones completely. This would lead to a high 3D-IoU value, even though the model could not be considered very capable.
Compared to the 3D-IoU over an entire scene, the advantage of this metric is the fact that position errors do not influence it. That makes it more sensitive to the quality of the shape, rotation, and scale reconstruction.

4.3 Model Variants to be Compared

All model variants presented here use the same model architecture described in Chapter 3.1. The learning rate during training of all variants was 0.0001, the batch size was 8 scenes. The weights for the losses used to compute the total loss are the same for each model as defined in Chapter 3.3.3. The model variants differ in the subsets of losses they use.

The three model variants to be compared are the following:

- *Base Model*:
  The model parameters for this version are the same as the ones used in [Elich et al., 2021]. The function for its total loss is:

$$L_{total,base} = \lambda_{RGB} L_{RGB} + \lambda_D L_D + \lambda_{gr} L_{gr} + \lambda_{reg} L_{reg} \tag{32}$$

  with the losses and their weights as defined in Chapter 3. This model was trained for 400 epochs since that was sufficient for loss convergence.

- *Intermediate Model*:
  This variant adds the silhouette losses for image and depth reconstruction to the total loss function:

$$L_{total,interm.} = \lambda_{RGB} L_{RGB} + \lambda_{RGB,sil} L_{RGB,sil} + \lambda_D L_D + \lambda_{D,sil} L_{D,sil} + \lambda_{gr} L_{gr} + \lambda_{reg} L_{reg} \tag{33}$$

  Here, 400 training epochs were sufficient, as well.

- *Final Model*:
  Here we additionally added the normal loss, the silhouette-normal loss, as well as the intersection loss. The total loss for the final model is defined as:

$$L_{total,final} = \lambda_{RGB} L_{RGB} + \lambda_{RGB,sil} L_{RGB,sil} + \lambda_D L_D + \lambda_{D,sil} L_{D,sil} + \lambda_N L_N + \lambda_{N,sil} L_{N,sil} + \lambda_{gr} L_{gr} + \lambda_{int} L_{int} + \lambda_{reg} L_{reg} \tag{34}$$

  This model was trained for 800 epochs, as the loss function could not be considered converged after 400 epochs of training.

Each of those model variants was trained five times with the same configurations.

4.4 Render-and-Compare Configurations for Experiments

In the following chapter, we evaluate three different configurations of the Render-and-Compare optimization. Here, we introduce the parameter values of those configurations.

The parameters that remain constant between the different configurations are the following: The maximum number of gradient descent steps we perform if the loss does not converge before is set to 50. The minimum difference between the previous and the current loss value to continue the iteration is 10^{-10}. The learning rate for the updates to the texture and extrinsics parts of the latent vectors are 0.001 and 0.01, respectively.
Since at test time we cannot use the ground-truth data from the dataset, the function for the total loss in the optimization can only be comprised of those losses that do not rely on ground-truth data. These are the RGB-reconstruction loss, as well as the intersection-, ground-, and regularization-losses. In that same order, the weights applied to these losses are 1.0, 0.001, 0.1, and 0.0025 respectively. We do not apply Gaussian smoothing during the optimization.
The three configurations differ in these aspects:

- *Base Configuration:*
  Here, the learning rate for the gradient descent updates to the shape part of the latent vector is zero, i.e., the shapes are not updated. The internal renderings are performed at the same resolution as the inputs, i.e., (64, 64).

- *Configuration with Anti-Aliasing:*
  Similarly to the base configuration, this configuration does not update the shape latent vectors of the objects. Internally, the reconstructions are rendered at twice the size, i.e., at a resolution of (128, 128). The loss is computed based on the anti-aliased down-sampling of these internal renders.

- *Configuration with Shape-Updates:*
  This configuration is the same as the base configuration up to the learning rate for the updates to the shape latent: Instead of zero, this learning rate is 0.01 in this case.

5 Experiments and Discussion

For the model variants and Render-and-Compare configurations detailed in Chapters 4.3 and 4.4, here we report and discuss the evaluation metrics defined in Chapter 4.2 on our CLEVR test set detailed in Chapter 4.1. Furthermore, we also discuss the training behavior of different model variants and address some shortcomings of the model.

5.1 Comparison of the Models' Behavior during Training

The original and final models exhibit some interesting differences and similarities in their behavior during training. These can be seen particularly well in their loss graphs for L_{RGB} and L_D, but a similar development can also be observed for the other reconstruction losses.

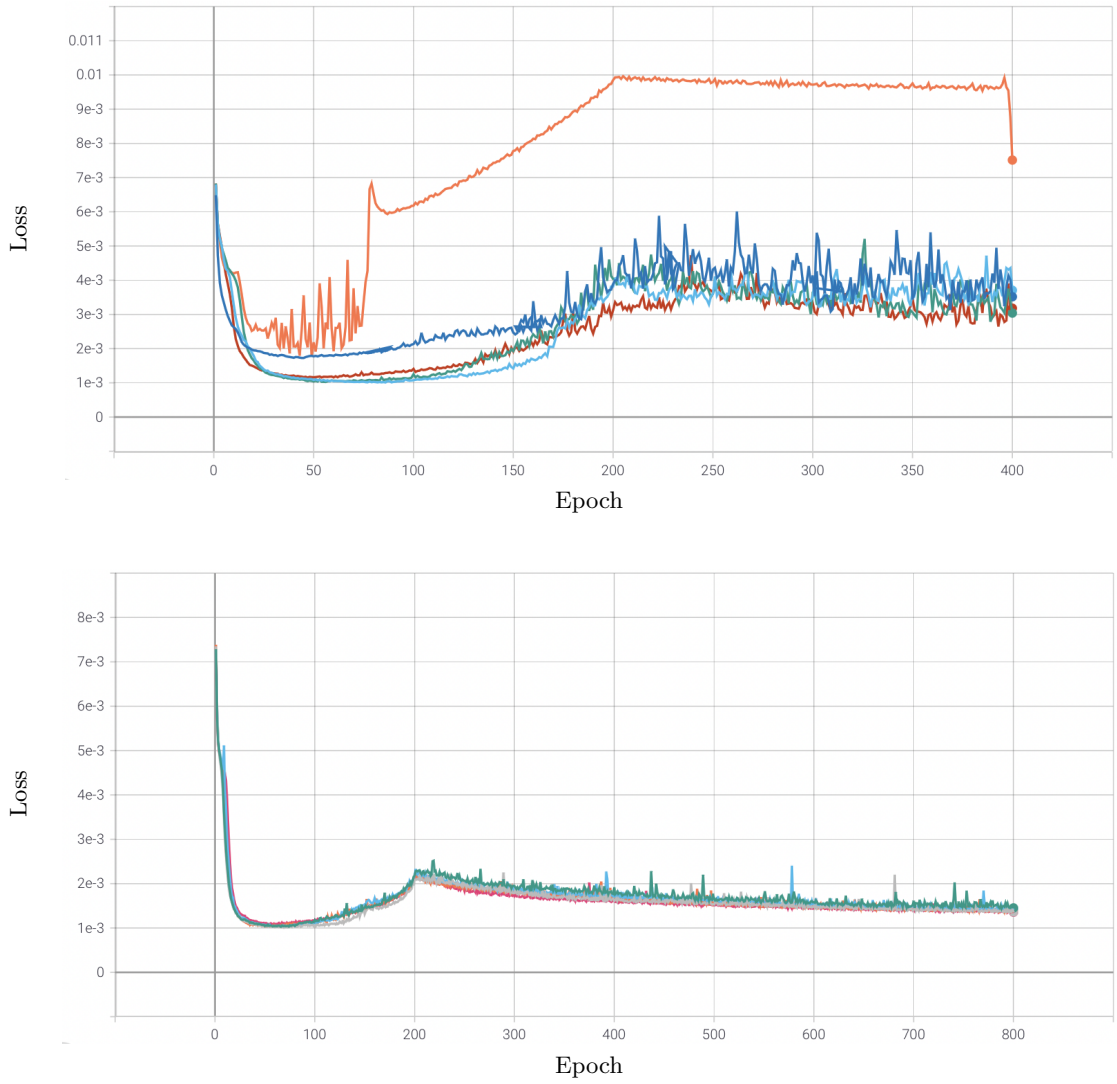

Figure 10: **RGB reconstruction loss over epochs of training for the original (top) and final (bottom) model over 5 different training runs**

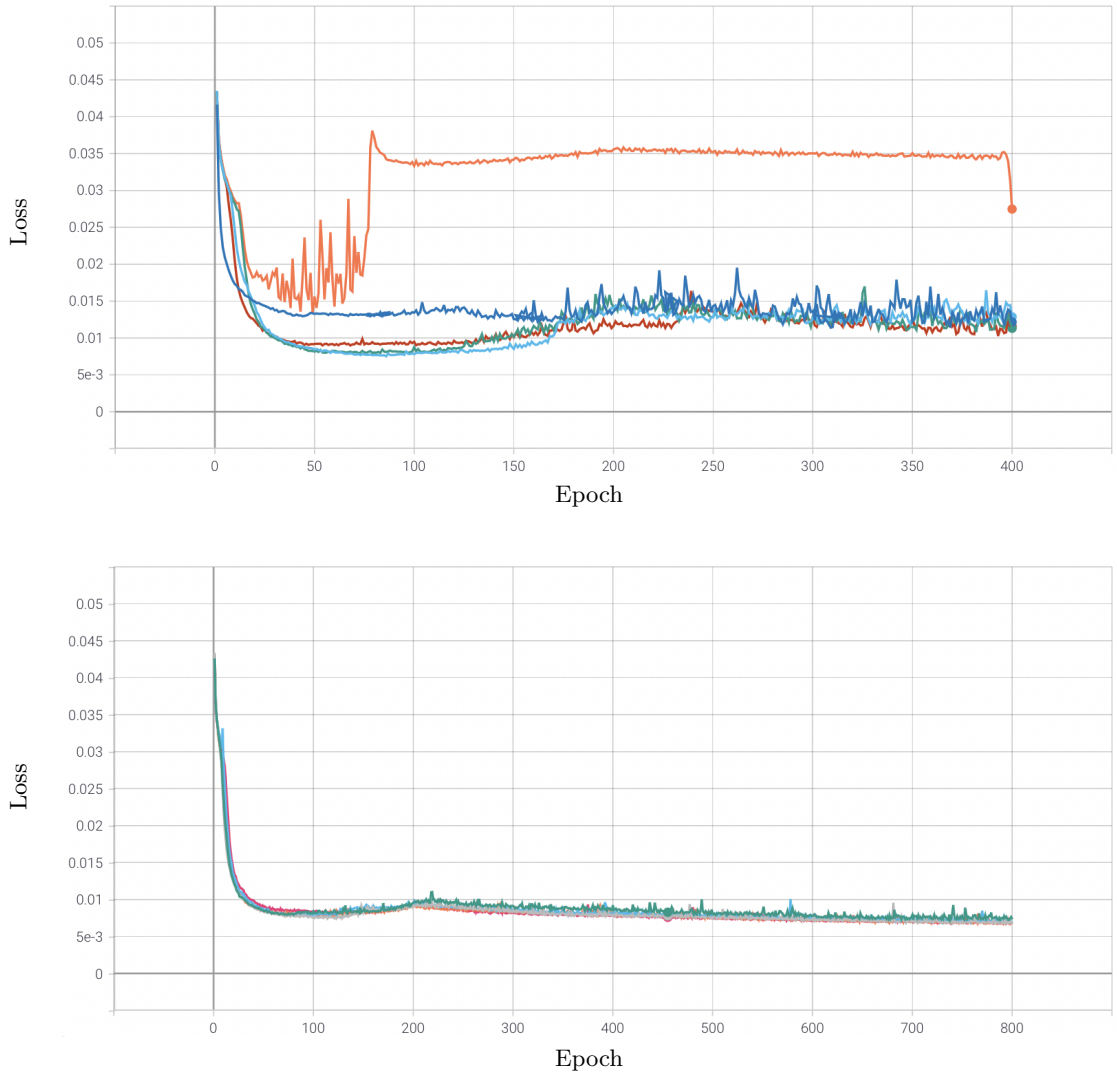

Figure 11: **Depth reconstruction loss over epochs of training for the original (top) and final (bottom) model over 5 different training runs**

What all training runs of both models for both losses shown in Figures 10 and 11 share, is the following behavior in the loss graphs: Roughly during the first 50 epochs of training, we can observe a sharp decrease of the losses. This is due to the completely untrained model starting to learn the basic tasks of object detection, positioning, scaling, and coloring. Then, however, over the following epochs until epoch 200, a sometimes extreme, sometimes only minor increase of the losses can be seen: The reason for that is the gradual decrease of the standard deviation in the Gaussian Smoothing kernel. This effects the images / depth maps over which the losses are computed to become clearer and less blurred, which renders the reconstruction tasks increasingly harder. The models can not quite compensate this increase in difficulty with their improving capabilities, thus leading to increases in loss. Next, in all training runs, the 200th epoch forms an inflection point: Here, the loss values suddenly cease to grow and transition into a phase of gradual decrease for the rest of training. The reason

for that is that starting from epoch 200, the amount of Gaussian Smoothing applied is no longer decreasing but constant at a very small value. This allows the improvements in model capabilities to once again become apparent in the loss graphs, as the task's difficulty is no longer simultaneously raised.

A significant difference between the models visible in these graphs is in the stability of the loss values both between different training runs of the same model and between temporally adjacent training epochs in the same run: While the original model displays a large variability and instability, the final model is remarkably more stable. Since the most significant difference between these models is the addition of silhouette losses, this improvement in stability can most likely be attributed to the improved object detection afforded by the silhouette losses: Where the final model likely already is able to detect objects reliably after the first phase of training roughly up until epoch 50, the original model's detection performance is much more fragile over the entire training duration: Without the strong training signal provided by the silhouette losses, object detection is quite unstable, making it sensitive to weight initialization and prone to getting stuck in local minima. This sensitivity to weight initializations explains the large variation between training runs, while the model's propensity to being caught in a local minimum goes to explain the instability between epochs of the same run.

The training behavior of the intermediate model variant was very similar to the final model variant, therefore it is not shown.
Finally, these graphs show that for the original model, 400 epochs of training were quite sufficient, while the final model needed twice as many epochs to achieve a state that can be considered converged. On a single Nvidia GTX 1080 GPU, training the first two models took roughly 5 days each, while the final model with twice as many epochs, needed about 10 days of training time.

5.2 Evaluation Results from Direct Model Outputs

Here, we report the evaluation results produced from the three model variants directly. For those results, no Render-and-Compare optimization has been performed.

	mAP	$AP_{0.5}$	$AR_{0.5}$	$F1_{0.5}$
Base model	0.6353 ± 0.0991	0.8689 ± 0.1047	0.7172 ± 0.0992	0.7664 ± 0.1006
Intermediate model	$\mathbf{0.7318 \pm 0.0185}$	$\mathbf{0.9640 \pm 0.0125}$	0.9358 ± 0.0066	$\mathbf{0.9458 \pm 0.0074}$
Final model	0.7046 ± 0.0068	0.9411 ± 0.0052	$\mathbf{0.9494 \pm 0.0014}$	0.9432 ± 0.0034

Table 1: **Mean and standard deviation of instance reconstruction metrics for direct model outputs over 5 training runs.**

	RGB RMSE	RGB PSNR	RGB SSIM
Base model	0.0665 ± 0.0092	23.9521 ± 1.1776	0.8759 ± 0.0237
Intermediate model	0.0494 ± 0.0013	26.3766 ± 0.2236	0.9181 ± 0.0035
Final model	$\mathbf{0.0460 \pm 0.0003}$	$\mathbf{27.0148 \pm 0.0501}$	$\mathbf{0.9253 \pm 0.0008}$

Table 2: **Mean and standard deviation of RGB reconstruction metrics for direct model outputs over 5 training runs.**

	Depth RMSE	Depth Abs.Rel.Diff	Depth Squ.Rel.Diff
Base model	0.7711 ± 0.1223	0.0437 ± 0.0075	0.1140 ± 0.0370
Intermediate model	$\mathbf{0.6078 \pm 0.0129}$	$\mathbf{0.0335 \pm 0.0006}$	0.0700 ± 0.0044
Final model	0.6185 ± 0.0028	0.0338 ± 0.0002	$\mathbf{0.0677 \pm 0.0013}$

Table 3: **Mean and standard deviation of depth reconstruction metrics for direct model outputs over 5 training runs.**

	Pos.Err	Rot.Err	Mean 3D IoU	Mean 3D IoU Single Objs.
Base model	0.1655 ± 0.0244	18.3005 ± 0.5121	0.5124 ± 0.0763	$0.7046 \pm 0.0291/2141 \pm 224$
Interm. model	$\mathbf{0.1494 \pm 0.0087}$	19.8006 ± 0.6319	$\mathbf{0.6287 \pm 0.0065}$	$\mathbf{0.7224 \pm 0.0021/2748 \pm 23}$
Final model	0.1722 ± 0.0041	$\mathbf{16.1424 \pm 0.3433}$	0.6078 ± 0.0059	$0.6863 \pm 0.0038/2806 \pm 8$

Table 4: **Mean and standard deviation of 3D reconstruction metrics for direct model outputs over 5 training runs.** For the mean 3D IoU for single objects, we additionally report the number of successfully detected objects out of the 3000 objects in the test set.

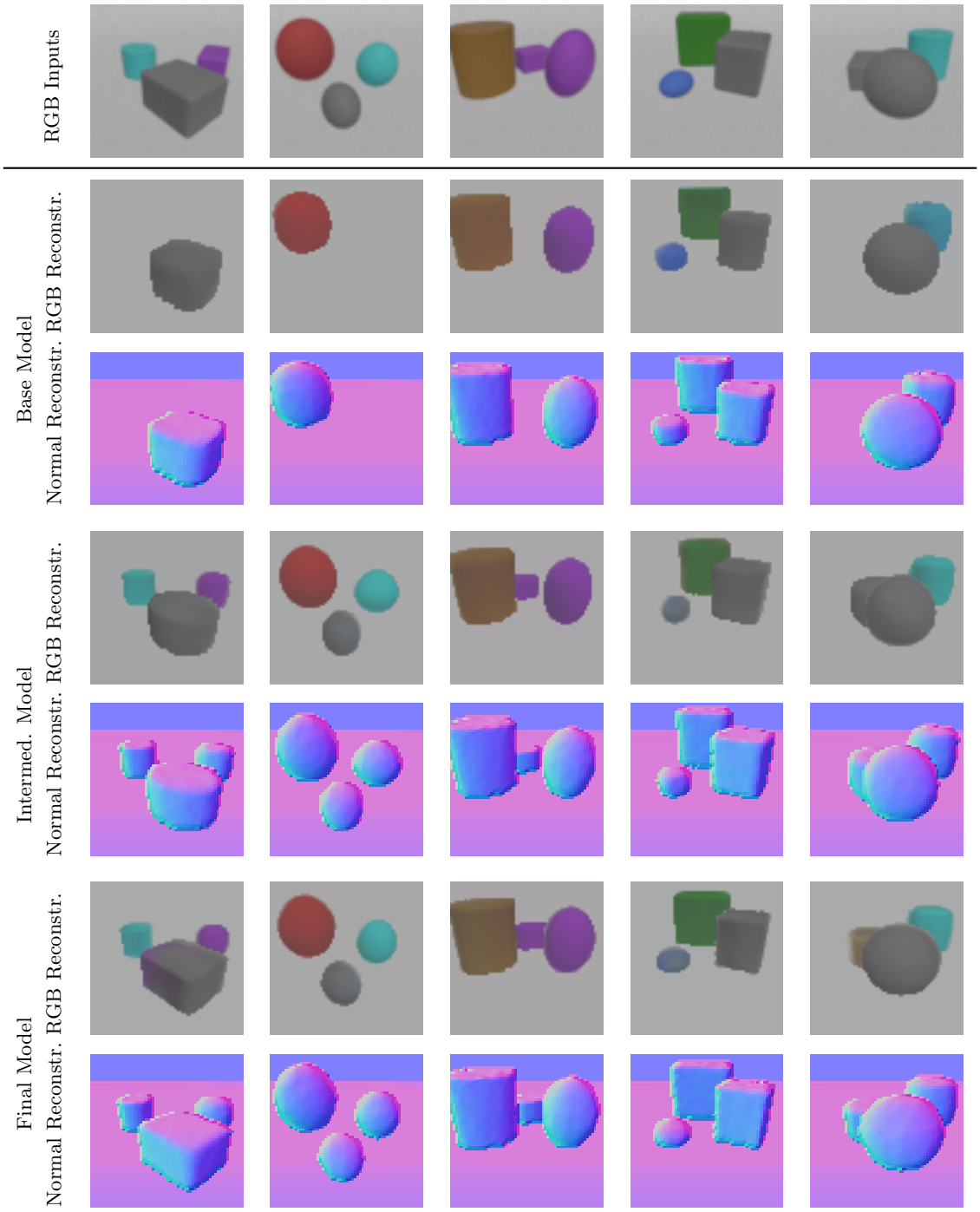

Figure 12: **Examples of inputs and reconstructions from the different models.** For each model both the predicted RGB reconstruction and the predicted normal map is shown. Between the first two models a significant improvement in the number of objects detected is visible. For the final model, the improvements in shape reconstruction compared to the previous two can be seen well.

While the intermediate model trained with the silhouette loss improved significantly over the model trained without it in terms of object detection, it performs worse in terms of the precise shape reconstruction. An obvious example of this can be seen in the leftmost column in Figure 12. This can also be seen in its deteriorated rotation error values compared to the base model variant: Due to the worse shape reconstruction, it is also more difficult for the model to precisely orient these less precise shapes. This issue is most likely due to the fact that the depth-silhouette-loss places too much emphasis on the silhouettes of the objects and thus de-emphasizes the correct depth reconstruction within the objects. The same would most likely be true for RGB if the objects were not uniformly colored: The silhouette regions would be reconstructed well (and by extension, object detection performance would be good), but the texture reconstruction within the objects would suffer. Overall, when trained simultaneously, good object detection and precise object reconstruction are somewhat at odds with one another: With a high silhouette loss weight, objects are detected reliably, but the reconstruction quality of the objects interiors deteriorates, and vice-versa for a low silhouette loss.

The addition of the normal losses managed to overcome this issue, however only at the expense of longer training time: While for the first two models 400 epochs of training sufficed to achieve loss convergence, the model with normal losses required 800 epochs. The addition of the normal losses in the final model did not lead to significant improvements in the evaluation metrics, but the representative examples in Figure 12 clearly show an improved shape reconstruction over the intermediate model.

What all these evaluation metrics clearly show is a tremendous decrease in their standard deviations over the the different training runs of the same models. This is again due to the large instability of the original model that was already discussed in Chapter 5.1 and the improvement in that regard afforded by the silhouette losses introduced in the later models.

For the 3D reconstruction metrics, the opposite trend to the previous metrics can be observed: While the other metrics improve significantly for the more sophisticated model variants, conversely, some of these results deteriorate slightly. This can be attributed to the fact that the much higher recall rates for the more advanced models lead to the models attempting to represent objects that are more challenging to detect and correctly reconstruct, especially highly occluded objects. Since the position- and rotation-errors and the 3D IoU values for single objects are computed only over the correctly detected objects, this leads to a relative disadvantage for the stronger models, and thus to slightly worse results.

This evaluation also serves as an ablation study on the impact of the silhouette- and normal-losses, the impact of the remaining losses has already been studied in [Elich et al., 2021].

What can be seen quite well in Figure 12 is that all models, even the base model, have learned some basic shading capability, even though no light source is modeled anywhere in the rendering pipeline: The models simply adapt the object textures to imitate the shading in the input images.

5.3 Evaluation Results from Render-and-Compare Outputs

Here, we compare the evaluation results for each of the model variants to the results of Render-and-Compare optimization without anti-aliasing or shape updates based on those model variants.

	mAP	$AP_{0.5}$	$AR_{0.5}$	$F1_{0.5}$
Base model	0.6353 ± 0.0991	0.8689 ± 0.1047	0.7172 ± 0.0992	0.7664 ± 0.1006
+ Render & Compare	0.6758 ± 0.1032	0.8841 ± 0.0947	0.7303 ± 0.0886	0.7803 ± 0.0901
Intermediate model	0.7318 ± 0.0185	0.9640 ± 0.0125	0.9358 ± 0.0066	0.9458 ± 0.0074
+ Render & Compare	$\mathbf{0.7683 \pm 0.0161}$	$\mathbf{0.9702 \pm 0.0121}$	0.9425 ± 0.0066	0.9522 ± 0.0074
Final model	0.7046 ± 0.0068	0.9411 ± 0.0052	0.9494 ± 0.0014	0.9432 ± 0.0034
+ Render & Compare	0.7467 ± 0.0076	0.9532 ± 0.0054	$\mathbf{0.9597 \pm 0.0016}$	$\mathbf{0.9546 \pm 0.0033}$

Table 5: **Mean and standard deviation of instance reconstruction metrics for optimized model outputs over 5 training runs.**

	RGB RMSE	RGB PSNR	RGB SSIM
Base model	0.0665 ± 0.0092	23.9521 ± 1.1776	0.8759 ± 0.0237
+ Render & Compare	0.0630 ± 0.0089	24.4896 ± 1.2154	0.8869 ± 0.0225
Intermediate model	0.0494 ± 0.0013	26.3766 ± 0.2236	0.9181 ± 0.0035
+ Render & Compare	0.0461 ± 0.0011	26.9829 ± 0.1981	0.9278 ± 0.0026
Final model	0.0460 ± 0.0003	27.0148 ± 0.0501	0.9253 ± 0.0008
+ Render & Compare	$\mathbf{0.0418 \pm 0.0004}$	$\mathbf{27.8322 \pm 0.0704}$	$\mathbf{0.9375 \pm 0.0009}$

Table 6: **Mean and standard deviation of RGB reconstruction metrics for optimized model outputs over 5 training runs.**

	Depth RMSE	Depth Abs.Rel.Diff	Depth Squ.Rel.Diff
Base model	0.7711 ± 0.1223	0.0437 ± 0.0075	0.1140 ± 0.0370
+ Render & Compare	0.7184 ± 0.1243	0.0412 ± 0.0073	0.1011 ± 0.0358
Intermediate model	0.6078 ± 0.0129	0.0335 ± 0.0006	0.0700 ± 0.0044
+ Render & Compare	$\mathbf{0.5600 \pm 0.0099}$	$\mathbf{0.0316 \pm 0.0004}$	0.0602 ± 0.0034
Final model	0.6185 ± 0.0028	0.0338 ± 0.0002	0.0677 ± 0.0013
+ Render & Compare	0.5728 ± 0.0055	0.0320 ± 0.0002	$\mathbf{0.0575 \pm 0.0010}$

Table 7: **Mean and standard deviation of depth reconstruction metrics for optimized model outputs over 5 training runs.**

	Pos.Err	Rot.Err	Mean 3D IoU	Mean 3D IoU Single Objs.
Base model	0.1655 ± 0.0244	18.3005 ± 0.5121	0.5124 ± 0.0763	$0.7046 \pm 0.0291/2141 \pm 224$
+ R. & C.	0.1622 ± 0.0294	18.3595 ± 0.4269	0.5189 ± 0.0737	$0.7032 \pm 0.0316/2153 \pm 204$
Interm. model	0.1494 ± 0.0087	19.8006 ± 0.6319	0.6287 ± 0.0065	$\mathbf{0.7224 \pm 0.0021/2748 \pm 23}$
+ R. & C.	$\mathbf{0.1429 \pm 0.0082}$	19.4878 ± 0.5100	$\mathbf{0.6351 \pm 0.0068}$	$\mathbf{0.7223 \pm 0.0021/2750 \pm 24}$
Final model	0.1722 ± 0.0041	16.1424 ± 0.3433	0.6078 ± 0.0059	$0.6863 \pm 0.0038/2806 \pm 8$
+ R. & C.	0.1677 ± 0.0041	$\mathbf{16.0672 \pm 0.4958}$	0.6107 ± 0.0051	$0.6866 \pm 0.0041/2817 \pm 11$

Table 8: **Mean and standard deviation of 3D reconstruction metrics for optimized model outputs over 5 training runs.**

Tables 5 to 8 show improvements from the Render-and-Compare optimization for all models across almost all metrics. For every metric, the best result overall is one achieved with Render-and-Compare optimization. To some extent, this would be expected as the losses minimized during optimization and the metrics shown here are correlated.

As detailed in Chapter 3.2.3, the Render-and-Compare optimization initialized with encoder outputs is unable to recover objects that have not been detected and represented by the model. Hence, the improvements in the instance reconstruction metrics are most likely due to some edge cases whose IoU-values previously classified them as False Positives. In those cases a small improvement in object pose or scale is sufficient to turn them into True Positives, leading to an improvement in all instance reconstruction metrics.

The most significant improvements with the optimization can be observed in the RGB and depth reconstruction metrics. This can be easily explained. We optimize image reconstruction loss directly, which leads immediately to better RGB metrics. Furthermore, in most cases, optimizing an object's extrinsics to improve its RGB appearance, also leads to improved depth reconstruction. The same holds for the position- and rotation error, although there, the improvements are less pronounced.

The minor improvement in the 3D voxel IoU results for entire scenes is due to the improved object placement achieved by the optimization. Since during optimization we do not apply shape updates here, the changes in the object-wise voxel IoU values can be due only to updated rotations and scaling of the objects. The reason for these results remaining roughly constant is most likely the fact that the improvements in rotation error are too small to influence those IoU results at the chosen voxel resolution.

In the qualitative results visible in Figure 13, most obviously, the refinements of the object textures can be seen. Improved pose reconstructions are visible as well, but the extent of those is more subtle.

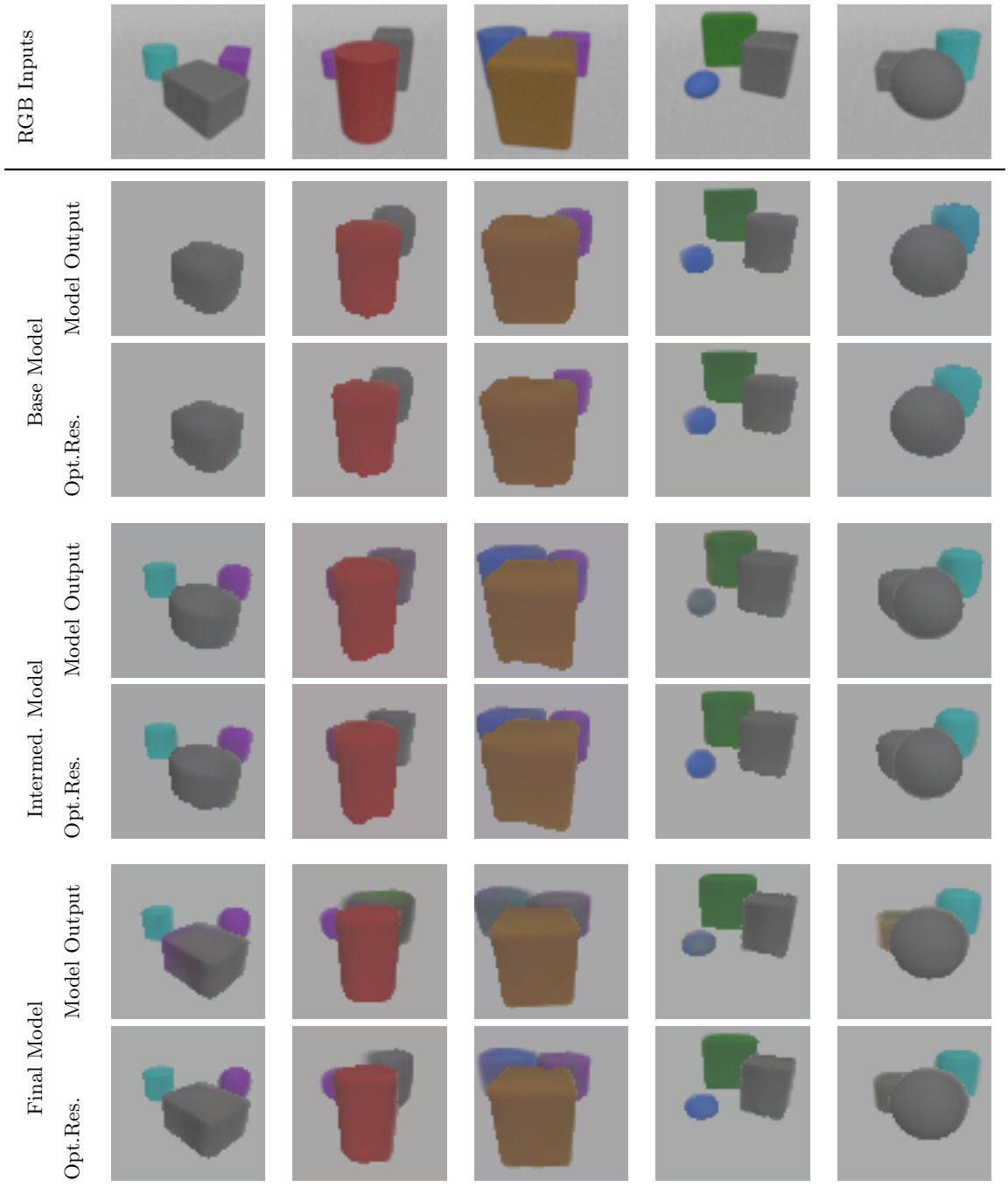

Figure 13: **Examples for Render-and-Compare optimization.** For each of the three models, the direct model output and the optimization result is shown. The most obvious improvements from the optimization visible here are the correction of texture reconstruction errors. In the second and third scene there are also improvements in pose reconstruction visible, however, these tend to be much more subtle.

5.4 Evaluation Results from Render-and-Compare Outputs with Anti-Aliasing

Here, we compare the evaluation results from the Render-and-Compare configuration with anti-aliasing to the configuration without, along with the direct model output of the final model variant.

	mAP	$AP_{0.5}$	$AR_{0.5}$	$F1_{0.5}$
Final model	0.7046 ± 0.0068	0.9411 ± 0.0052	0.9494 ± 0.0014	0.9432 ± 0.0034
+ Render & Compare	$\mathbf{0.7467 \pm 0.0076}$	0.9532 ± 0.0054	0.9597 ± 0.0016	0.9546 ± 0.0033
+ Anti-Aliasing	0.7400 ± 0.0070	$\mathbf{0.9549 \pm 0.0053}$	$\mathbf{0.9620 \pm 0.0016}$	$\mathbf{0.9566 \pm 0.0030}$

Table 9: **Mean and standard deviation of instance reconstruction metrics for optimized model outputs with anti-aliasing over 5 training runs.**

	RGB RMSE	RGB PSNR	RGB SSIM
Final model	0.0460 ± 0.0003	27.0148 ± 0.0501	0.9253 ± 0.0008
+ Render & Compare	0.0418 ± 0.0004	27.8322 ± 0.0704	0.9375 ± 0.0009
+ Anti-Aliasing	$\mathbf{0.0388 \pm 0.0003}$	$\mathbf{28.5089 \pm 0.0658}$	$\mathbf{0.9471 \pm 0.0008}$

Table 10: **Mean and standard deviation of RGB reconstruction metrics for optimized model outputs with anti-aliasing over 5 training runs.**

	Depth RMSE	Depth Abs.Rel.Diff	Depth Squ.Rel.Diff
Final model	0.6185 ± 0.0028	0.0338 ± 0.0002	0.0677 ± 0.0013
+ Render & Compare	0.5728 ± 0.0055	0.0320 ± 0.0002	0.0575 ± 0.0010
+ Anti-Aliasing	$\mathbf{0.5179 \pm 0.0042}$	$\mathbf{0.0315 \pm 0.0002}$	$\mathbf{0.0474 \pm 0.0007}$

Table 11: **Mean and standard deviation of depth reconstruction metrics for optimized model outputs with anti-aliasing over 5 training runs.**

	Pos.Err	Rot.Err	Mean 3D IoU	Mean 3D IoU Single Objs.
Final model	0.1722 ± 0.0041	16.1424 ± 0.3433	0.6078 ± 0.0059	$0.6863 \pm 0.0038/2806 \pm 8$
+ R. & C.	$\mathbf{0.1677 \pm 0.0041}$	$\mathbf{16.0672 \pm 0.4958}$	0.6107 ± 0.0051	$0.6866 \pm 0.0041/2817 \pm 11$
+ Anti-Alias.	0.1692 ± 0.0040	16.1621 ± 0.4059	$\mathbf{0.6112 \pm 0.0048}$	$\mathbf{0.6866 \pm 0.0041/2823 \pm 13}$

Table 12: **Mean and standard deviation of 3D reconstruction metrics for optimized model outputs with anti-aliasing over 5 training runs.**

As can be seen from these results, none of the 3D reconstruction metrics improved significantly with the addition of anti-aliasing. Thus, the Anti-Aliasing method was not able to produce more precise predictions in terms of positional error and 3D-IoU, as we hoped for and suspected in Chapter 3.2.4.

This method did, however, lead to large improvements in both the RGB and depth reconstruction metrics. This was to be expected as the model is now able to match its input much more closely than before, particularly around the edges of the objects. That can also be seen

Figure 14: **Examples for Render-and-Compare optimization with anti-aliasing.** For each input RGB image, the direct model output, the Render-and-Compare optimization result, and the optimization result with anti-aliasing is shown. The impact of the application of anti-aliasing is particularly visible around the objects' edges which are much smoother compared to the results without anti-aliasing.

well in the example results in Figure 14.

The refinements in instance reconstruction are due to the fact that anti-aliasing is also applied to the instance masks. With the pixel IoU implementation used, this leads to slightly higher IoU values and thus, as explained in Chapter 5.3 to improved instance reconstruction metrics.

5.5 Evaluation Results from Render-and-Compare Outputs with Shape Updates

In this chapter, we present the results of our experiments with shape updates in the Render-and-Compare optimization.

	mAP	$AP_{0.5}$	$AR_{0.5}$	$F1_{0.5}$
Final model	0.7046 ± 0.0068	0.9411 ± 0.0052	0.9494 ± 0.0014	0.9432 ± 0.0034
+ Render & Compare	$\mathbf{0.7467 \pm 0.0076}$	$\mathbf{0.9532 \pm 0.0054}$	$\mathbf{0.9597 \pm 0.0016}$	$\mathbf{0.9546 \pm 0.0033}$
+ Shape Updates	0.7062 ± 0.0089	0.9429 ± 0.0066	0.9540 ± 0.0028	0.9463 ± 0.0046

Table 13: **Mean and standard deviation of instance reconstruction metrics for optimized model outputs with shape updates over 5 training runs.**

	RGB RMSE	RGB PSNR	RGB SSIM
Final model	0.0460 ± 0.0003	27.0148 ± 0.0501	0.9253 ± 0.0008
+ Render & Compare	$\mathbf{0.0418 \pm 0.0004}$	$\mathbf{27.8322 \pm 0.0704}$	$\mathbf{0.9375 \pm 0.0009}$
+ Shape Updates	0.0447 ± 0.0005	27.2300 ± 0.0791	0.9283 ± 0.0014

Table 14: **Mean and standard deviation of RGB reconstruction metrics for optimized model outputs with shape updates over 5 training runs.**

	Depth RMSE	Depth Abs.Rel.Diff	Depth Squ.Rel.Diff
Final model	0.6185 ± 0.0028	0.0338 ± 0.0002	0.0677 ± 0.0013
+ Render & Compare	$\mathbf{0.5728 \pm 0.0055}$	$\mathbf{0.0320 \pm 0.0002}$	$\mathbf{0.0575 \pm 0.0010}$
+ Shape Updates	0.6140 ± 0.0072	0.0339 ± 0.0003	0.0716 ± 0.0025

Table 15: **Mean and standard deviation of depth reconstruction metrics for optimized model outputs with shape updates over 5 training runs.**

	Pos.Err	Rot.Err	Mean 3D IoU	Mean 3D IoU Single Objs.
Final model	0.1722 ± 0.0041	16.1424 ± 0.3433	0.6078 ± 0.0059	$0.6863 \pm 0.0038/2806 \pm 8$
+ R. & C.	0.1677 ± 0.0041	$\mathbf{16.0672 \pm 0.4958}$	$\mathbf{0.6107 \pm 0.0051}$	$0.6866 \pm 0.0041/2817 \pm 11$
+ Shape Upd.	$\mathbf{0.1676 \pm 0.0050}$	16.3451 ± 0.4385	0.6060 ± 0.0052	$\mathbf{0.6896 \pm 0.0029/2808.0 \pm 14}$

Table 16: **Mean and standard deviation of 3D reconstruction metrics for optimized model outputs with shape updates over 5 training runs.**

As can be seen in Tables 13 to 16 and qualitatively in Figure 15, the Render-and-Compare optimization with shape updates performs slightly worse across almost all metrics compared to the optimization without shape updates. This can be explained by the following insights:

If a shape update from one shape category to another is required, for instance to turn a cylinder into a cube, it is likely that during the optimization's traversal of the shape space an intermediate object configuration will be encountered that leads to a higher loss value. That higher loss value causes the optimization to abort in a suboptimal state. This can be understood easily with the subsequent example: If a cylinder needed to be turned into a cube for correct reconstruction, then the cube's rotation would very likely be wrong by a large margin

as so far, for the cylinder, the orientation was irrelevant. In that case the cylinder would have a lower loss value than the incorrectly oriented cube. What this example illustrates is the fact that the shape space is very prone to local minima that are in most cases impossible to escape with gradient descent. For tasks like object positioning, scaling, and coloring, this issue exists only to a much smaller extent, if at all, as for those the loss function is roughly convex.

Beyond that, since the optimization only sees the input RGB image and no depth data, correcting shape reconstruction mistakes of the model is probably only possible with lighting estimation and modeling. In that case, the RGB reconstruction loss might lead the optimization to be able to reconstruct shapes based on light and shadows on the objects. Without lighting estimation, however, the results suggest that improving shape reconstruction is too complex to be directly updated through gradient descent.

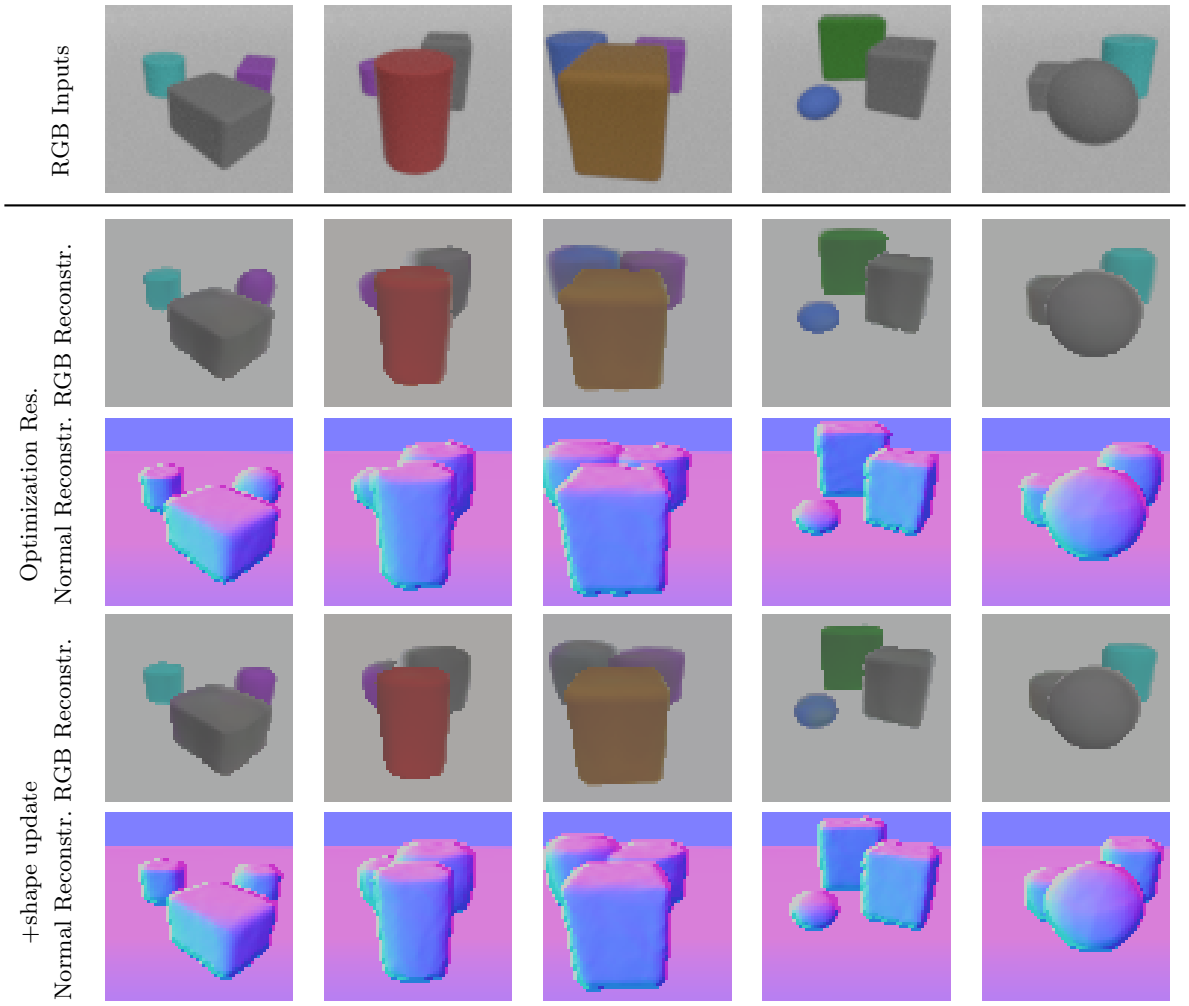

Figure 15: **Examples for Render-and-Compare optimization with shape-updates.** For each input RGB image, the Render-and-Compare optimization result with and without shape-updates is shown. For the purple object in the leftmost scene, a slight improvement in its shape can be seen. However, for this method, the drawbacks outweigh the benefits: Especially for texture reconstruction there is a small, but noticeable deterioration visible.

5.6 Limitation to Scenes with Previously Known Number of Objects

Figure 16 shows some instances where fewer objects than the number of slots predicted are visible in the scene. Furthermore, it shows how the final model reconstructs these scenes.

In the two leftmost columns, a typical failure case in this situation is visible: Since the object mask predicted in the first slot does not cover an object (the purple one in both cases) completely, and no other object remains to be reconstructed, the model often makes the mistake of representing one object as multiple ones arranged behind each other. In the central column there is a highly occluded third object visible, but the model fails to detect it and instead represents the turqoise one twice.

The two rightmost columns show cases where the model handles this case more gracefully: Here the object in the last slot is either a tiny, gray one that blends into the background, and does not impact depth loss significantly, or the object is moved out of the camera's view entirely.
However, also this behavior, while preferable to the one discussed before, is not ideal.

These issues are all due to the inflexible prior assumption of a certain fixed number of objects in the input scene. As an attempt to overcome these problems, a post-processing procedure was implemented to remove these superfluous objects after the model's prediction: This method would remove objects from the final output, whose pixels are occluded to a large percentage by another object of similar average color that is closer in terms of the L2-distance to the object than some threshold. In practice, we found a distance between the objects of less than 2.0, an L2-distance between the average colors of the objects of less than 0.3 and a fraction of more than 80% of the object's pixels occluded to be good criteria. If all those criteria were met, an object would be removed from the prediction.
In a qualitative inspection, we found that overall this lead to a better handling of this case of fewer objects than slots, but it would also sometimes lead to objects being erroneously removed, which is even more undesirable.

Therefore, ideally, a future version of this model would have the capability to predict the number of objects in a scene and stop once no more objects are to be reconstructed.

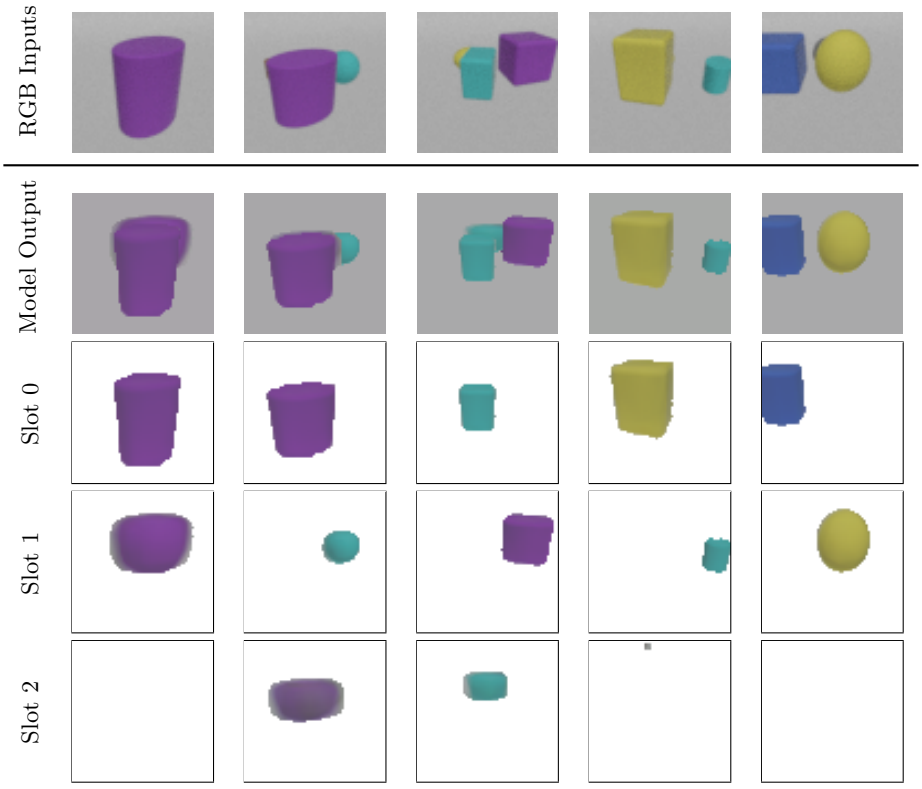

Figure 16: **Cases with more slots than visible objects in the scene.** For each RGB input image, the depth-ordered combined result and the contents of the individual slots predicted by the final model are shown. The failure cases in the left three columns show situations where one object is reconstructed as multiple ones of similar color arranged behind each other. In the remaining columns, examples can be seen where the situation of more slots than objects in the scene is handled more gracefully.

5.7 Runtime Discussion

While on a modern CPU a single forward pass through the model takes about 500ms to 1000ms for a batch of 8 scenes, optimizing a single scene with our Render-and-Compare implementation can take between 2s and 20s, depending on the quality of the objects' initializations. On a GPU this runtime difference will likely be even more pronounced.

Thus, in terms of runtime, a Render-and-Compare optimization is not competitive with a direct model prediction. For practical applications, we believe that this optimization is best suited to creating a dataset with pseudo-ground-truth latents, as described in Chapter 3.4. This dataset could then be used in a supervised training setting and would ideally transfer some of the improvements over the model the optimization is based on into the new model.

6 Concluding Remarks and Further Work

In this work we presented 1) the Render-and-Compare optimization method to iteratively refine object poses, shapes, and textures beyond what the model would predict itself and 2) the notion of silhouette losses and their relevance to reliable object detection. Beyond that, we refined the model further with fine-tuned loss weight schedules and the addition of normal losses.

For these contributions, the results of our ablation studies in the evaluation show significant improvements across most of the evaluation metrics.

The most obvious limitations of the model and the Render-and-Compare optimization are the lack of lighting estimation and modeling: By estimating the light source in the scene and by shading the rendered objects with it, a significant improvement in reconstruction quality particularly of the objects' shapes might be possible.
Furthermore, the inflexible assumption of a fixed number of objects in a scene leads to problems in cases where more or fewer objects are visible. To overcome this limitation, a convolutional deep learning model could be devised that regresses the number of objects visible in an input image.

We believe that, even as the reconstruction quality of such self-supervised scene representation networks improves in the future, a Render-and-Compare optimization as presented here still has its merits, since in many cases it is able to overcome some of the inaccuracies a model may display in its outputs.

References

[Achlioptas et al., 2018] Achlioptas, P., Diamanti, O., Mitliagkas, I., and Guibas, L. (2018). Learning representations and generative models for 3D point clouds. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 40–49. PMLR.

[Beker et al., 2020a] Beker, D., Kato, H., Morariu, M. A., Ando, T., Matsuoka, T., Kehl, W., and Gaidon, A. (2020a). Monocular differentiable rendering for self-supervised 3d object detection.

[Beker et al., 2020b] Beker, D., Kato, H., Morariu, M. A., Ando, T., Matsuoka, T., Kehl, W., and Gaidon, A. (2020b). Monocular differentiable rendering for self-supervised 3d object detection.

[Burgess et al., 2019] Burgess, C. P., Matthey, L., Watters, N., Kabra, R., Higgins, I., Botvinick, M., and Lerchner, A. (2019). Monet: Unsupervised scene decomposition and representation. *CoRR*, abs/1901.11390.

[Chen and Zhang, 2019] Chen, Z. and Zhang, H. (2019). Learning implicit fields for generative shape modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Choy et al., 2016] Choy, C. B., Xu, D., Gwak, J., Chen, K., and Savarese, S. (2016). 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction.

[Elich et al., 2021] Elich, C., Oswald, M. R., Pollefeys, M., and Stueckler, J. (2021). Semi-supervised learning of multi-object 3d scene representations.

[Gadelha et al., 2016] Gadelha, M., Maji, S., and Wang, R. (2016). 3d shape induction from 2d views of multiple objects.

[Greff et al., 2020] Greff, K., Kaufman, R. L., Kabra, R., Watters, N., Burgess, C., Zoran, D., Matthey, L., Botvinick, M., and Lerchner, A. (2020). Multi-object representation learning with iterative variational inference.

[Groueix et al., 2018] Groueix, T., Fisher, M., Kim, V. G., Russell, B. C., and Aubry, M. (2018). A papier-mch approach to learning 3d surface generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Johnson et al., 2017] Johnson, J., Hariharan, B., Maaten, L. V. D., Fei-Fei, L., Zitnick, C. L., and Girshick, R. B. (2017). Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1988–1997.

[Kar et al., 2017] Kar, A., Hne, C., and Malik, J. (2017). Learning a multi-view stereo machine.

[Li et al., 2019] Li, Y., Wang, G., Ji, X., Xiang, Y., and Fox, D. (2019). Deepim: Deep iterative matching for 6d pose estimation. *International Journal of Computer Vision*, 128(3):657?678.

[Liao et al., 2019] Liao, Y., Schwarz, K., Mescheder, L. M., and Geiger, A. (2019). Towards unsupervised learning of generative models for 3d controllable image synthesis. *CoRR*, abs/1912.05237.

[Locatello et al., 2020] Locatello, F., Weissenborn, D., Unterthiner, T., Mahendran, A., Heigold, G., Uszkoreit, J., Dosovitskiy, A., and Kipf, T. (2020). Object-centric learning with slot attention.

[Mescheder et al., 2019] Mescheder, L., Oechsle, M., Niemeyer, M., Nowozin, S., and Geiger, A. (2019). Occupancy networks: Learning 3d reconstruction in function space.

[Nguyen-Phuoc et al., 2019] Nguyen-Phuoc, T., Li, C., Balaban, S., and Yang, Y.-L. (2019). Rendernet: A deep convolutional network for differentiable rendering from 3d shapes.

[Nguyen-Phuoc et al., 2020] Nguyen-Phuoc, T., Richardt, C., Mai, L., Yang, Y.-L., and Mitra, N. (2020). Blockgan: Learning 3d object-aware scene representations from unlabelled images.

[Niemeyer and Geiger, 2021] Niemeyer, M. and Geiger, A. (2021). Giraffe: Representing scenes as compositional generative neural feature fields. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.

[Oechsle et al., 2019] Oechsle, M., Mescheder, L., Niemeyer, M., Strauss, T., and Geiger, A. (2019). Texture fields: Learning texture representations in function space. In *International Conference on Computer Vision*.

[Park et al., 2019] Park, J. J., Florence, P., Straub, J., Newcombe, R. A., and Lovegrove, S. (2019). Deepsdf: Learning continuous signed distance functions for shape representation. *CoRR*, abs/1901.05103.

[Periyasamy et al., 2019] Periyasamy, A. S., Schwarz, M., and Behnke, S. (2019). Refining 6d object pose predictions using abstract render-and-compare.

[Qi et al., 2017] Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017). Pointnet: Deep learning on point sets for 3d classification and segmentation.

[Qi et al., 2016] Qi, C. R., Su, H., Niessner, M., Dai, A., Yan, M., and Guibas, L. J. (2016). Volumetric and multi-view cnns for object classification on 3d data.

[Rezende et al., 2018] Rezende, D. J., Eslami, S. M. A., Mohamed, S., Battaglia, P., Jaderberg, M., and Heess, N. (2018). Unsupervised learning of 3d structure from images.

[Richardson et al., 2017] Richardson, E., Sela, M., Or-El, R., and Kimmel, R. (2017). Learning detailed face reconstruction from a single image.

[Shin et al., 2018] Shin, D., Fowlkes, C. C., and Hoiem, D. (2018). Pixels, voxels, and views: A study of shape representations for single view 3d object shape prediction.

[Sitzmann et al., 2020] Sitzmann, V., Zollhfer, M., and Wetzstein, G. (2020). Scene representation networks: Continuous 3d-structure-aware neural scene representations.

[Tulsiani et al., 2017] Tulsiani, S., Zhou, T., Efros, A. A., and Malik, J. (2017). Multi-view supervision for single-view reconstruction via differentiable ray consistency.

[Wang et al., 2020] Wang, R., Yang, N., Stueckler, J., and Cremers, D. (2020). Directshape: Direct photometric alignment of shape priors for visual vehicle pose and shape estimation.

[Wang et al., 2004] Wang, Z., Bovik, A., Sheikh, H., and Simoncelli, E. (2004). Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612.

[Wu et al., 2017] Wu, J., Zhang, C., Xue, T., Freeman, W. T., and Tenenbaum, J. B. (2017). Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling.

[Xie et al., 2019] Xie, H., Yao, H., Sun, X., Zhou, S., and Zhang, S. (2019). Pix2vox: Context-aware 3d reconstruction from single and multi-view images. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*.

[Xu et al., 2019] Xu, Q., Wang, W., Ceylan, D., Mech, R., and Neumann, U. (2019). Disn: Deep implicit surface network for high-quality single-view 3d reconstruction.

List of Tables

1 **Mean and standard deviation of instance reconstruction metrics for direct model outputs over 5 training runs.** 30
2 **Mean and standard deviation of RGB reconstruction metrics for direct model outputs over 5 training runs.** 30
3 **Mean and standard deviation of depth reconstruction metrics for direct model outputs over 5 training runs.** 30
4 **Mean and standard deviation of 3D reconstruction metrics for direct model outputs over 5 training runs.** For the mean 3D IoU for single objects, we additionally report the number of successfully detected objects out of the 3000 objects in the test set. 30
5 **Mean and standard deviation of instance reconstruction metrics for optimized model outputs over 5 training runs.** 33
6 **Mean and standard deviation of RGB reconstruction metrics for optimized model outputs over 5 training runs.** 33
7 **Mean and standard deviation of depth reconstruction metrics for optimized model outputs over 5 training runs.** 33
8 **Mean and standard deviation of 3D reconstruction metrics for optimized model outputs over 5 training runs.** 34
9 **Mean and standard deviation of instance reconstruction metrics for optimized model outputs with anti-aliasing over 5 training runs.** . . 36
10 **Mean and standard deviation of RGB reconstruction metrics for optimized model outputs with anti-aliasing over 5 training runs.** . . 36
11 **Mean and standard deviation of depth reconstruction metrics for optimized model outputs with anti-aliasing over 5 training runs.** . . 36
12 **Mean and standard deviation of 3D reconstruction metrics for optimized model outputs with anti-aliasing over 5 training runs.** 36
13 **Mean and standard deviation of instance reconstruction metrics for optimized model outputs with shape updates over 5 training runs.** . 38
14 **Mean and standard deviation of RGB reconstruction metrics for optimized model outputs with shape updates over 5 training runs.** . 38
15 **Mean and standard deviation of depth reconstruction metrics for optimized model outputs with shape updates over 5 training runs.** . 38
16 **Mean and standard deviation of 3D reconstruction metrics for optimized model outputs with shape updates over 5 training runs.** . . . 38

List of Figures

1 **Model Architecture overview.** (Figure taken from [Elich et al., 2021]). In this sequential architecture, for each object, the object encoder network g_o receives the input image I, a mask M based on the objects already reconstructed, as well as the difference-image ΔI. From these inputs, it outputs the latent vector $\mathbf{z}$ of the next object to be represented. Taking these latent vectors as input, the decoder F, based on a differentiable renderer, is used to produce renderings and masks of the individual objects and also to reconstruct the entire image, given $\mathbf{z}_{bg}$, the latent representation of the image background. 5

2 **Encoding and rendering of an object in a single slot of the model.** (Figure taken from [Elich et al., 2021]). The input image, difference image and combined mask based on the previously reconstructed objects are passed into the object encoder g_o. Its output latent vector $\mathbf{z}_i$ describing the next object is comprised of the extrinsics $\mathbf{z}_{i,ext}$, shape $\mathbf{z}_{i,sh}$, and texture latents $\mathbf{z}_{i,tex}$. The latter two of those latents parametrize the shape- and texture-decoders Φ and Ψ. From the output of Φ and the extrinsics latent, the object depth and mask are raycast. The colors of the pixels are obtained by sampling Ψ at the raycast positions. 7

3 **Qualitative results on test set.** Each row shows the RGB input, the optimization result with encoder output as initialization, and the optimization result with an ϵ-vector as initialization. The much improved object detection performance of the optimization from ϵ-vectors can be clearly seen, but the reconstruction quality of the newly detected objects is poor. 10

4 **Qualitative results of optimization with anti-aliasing on test set.** An example RGB input and the model's reconstruction without and with anti-aliasing are shown, as well as the internal higher-resolution full-size rendering. The effect of anti-aliasing can be seen well around the objects' edges. 12

5 **Silhouette mask implementation.** The edges extracted from the ground-truth depth and normal maps are combined into the set of edge pixels that after dilation defines the silhouette mask. The final silhouette mask is shown applied to the input RGB image. 15

6 **Examples of silhouette masks.** Top row: input images; Bottom row: application of corresponding silhouette masks 15

7 **Qualitative results on test set from model trained without variable normal loss weights.** Both the ground-truth and predicted images and normal maps are shown. 18

8 **Contents for example scene in our variant of the CLEVR dataset.** 20

9 **3D voxel sets for example scene.** Here, the ground-truth and the predicted voxel set for an input image can be seen. 24

10 **RGB reconstruction loss over epochs of training for the original (top) and final (bottom) model over 5 different training runs** 27

11 **Depth reconstruction loss over epochs of training for the original (top) and final (bottom) model over 5 different training runs** 28

12 **Examples of inputs and reconstructions from the different models.** For each model both the predicted RGB reconstruction and the predicted normal map is shown. Between the first two models a significant improvement in the number of objects detected is visible. For the final model, the improvements in shape reconstruction compared to the previous two can be seen well. 31
13 **Examples for Render-and-Compare optimization.** For each of the three models, the direct model output and the optimization result is shown. The most obvious improvements from the optimization visible here are the correction of texture reconstruction errors. In the second and third scene there are also improvements in pose reconstruction visible, however, these tend to be much more subtle. 35
14 **Examples for Render-and-Compare optimization with anti-aliasing.** For each input RGB image, the direct model output, the Render-and-Compare optimization result, and the optimization result with anti-aliasing is shown. The impact of the application of anti-aliasing is particularly visible around the objects' edges which are much smoother compared to the results without anti-aliasing. 37
15 **Examples for Render-and-Compare optimization with shape-updates.** For each input RGB image, the Render-and-Compare optimization result with and without shape-updates is shown. For the purple object in the leftmost scene, a slight improvement in its shape can be seen. However, for this method, the drawbacks outweigh the benefits: Especially for texture reconstruction there is a small, but noticeable deterioration visible. 40
16 **Cases with more slots than visible objects in the scene.** For each RGB input image, the depth-ordered combined result and the contents of the individual slots predicted by the final model are shown. The failure cases in the left three columns show situations where one object is reconstructed as multiple ones of similar color arranged behind each other. In the remaining columns, examples can be seen where the situation of more slots than objects in the scene is handled more gracefully. 42

The implementation of the work presented here can be found at https://gitlab.inf.ethz.ch/celich/3dmultiobj_optimize

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Optimizing 3D Object-Wise Representations

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Bohn	Christian

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zürich, 16.08.2021	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.