ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

ReconRLA: Point-based large-scale Surface Reconstruction from Point Clouds

Bachelor Thesis CVG Lab
2020/21

Nicolas Muntwyler

Advisor: Sandro Lombardi
Supervisor: Prof. Dr. Marc Pollefeys

Abstract

3D shape reconstruction from point clouds is a crucial part in many computer graphics and vision applications like autonomous driving or video games. While classical methods suffer from large computational efforts or memory foot prints for detailed reconstructions, recent approaches use deep neural nets to implicitly represent the input shape, which enables them to generate possibly infinite resolution meshes. Although these approaches return state-of-the-art results on synthetic data sets, most do not scale to larger scenes. They either fail to produce accurate representations or become computationally intractable for larger point clouds. Implicit based methods that do scale use fast hierarchical grid structures, which have complex implementations and additional overhead. In this work we present a pipeline that operates directly on point clouds, which makes our method scalable and efficient. Further it is a lightweight and simple architecture that can be easily extended. Our approach gives good qualitative and quantitative results on synthetic datasets and generalizes to large-scale scenes. Lastly we show that our neural net generates comparable results to state-of-the-art reconstructions on noisy real-world data.

Acknowledgments

I would like to thank my advisor Sandro Lombardi for his helpful guidance, encouragement and endless optimism. Additionally I would like to extend my gratitude towards my family and friends for their constant mental support.

Contents

List of Figures

List of Tables

1 Introduction

In computer vision and computer graphics large-scale 3D surface reconstructions is still an unsolved problem and a multitude of distinct approaches exist with different advantages. While some methods have range or depth scans as their input, we focus on the setting where we are given a point cloud. Our task is to use the information from the point cloud to reconstruct the underlying object/scene. In the past classical methods were the front runners. However with the rise of deep neural nets recent learning-based methods outperform the older approaches. A big advantage of neural nets is their ability to implicitly represent the reconstructions [5, 6, 7, 8]. By querying the model for SDF [5, 7] or occupancy [8] values we can reconstruct shapes of possible infinite resolution, without becoming computationally intractable. Recent implicit-based methods [4, 9] achieve state-of-the-art for qualitative and quantitative results on synthetic datasets like ShapeNet [1]. However it becomes more difficult for a network to accurately reconstruct larger scenes, since it implicitly has to represent much more information. While pioneering work used only one global latent code [5] that led to loss of detail and over-smoothing, recent methods employ a hierarchical grid-based structure in order to represent larger scenes more accurately. Although there exist methods that find fast implementations for these grid-based approaches, their performance is lastly dependent on the grid resolution. Additionally they are often very complex and need to store additional grid information.

In this thesis we present a grid-free scalable simple reconstruction pipeline that directly operates on the input point cloud. To directly apply a deep neural net on point clouds was first introduced by PointNet [10] for the task of 3D shape segmentation. Recently RandLA [11] was proposed, a large-scale efficient semantic segmentation network for point clouds, which inspired us to apply their concepts to shape reconstructions. We present **ReconRLA**, a point-based neural network for **Recon**struction tasks, that is inspired by **R**and**LA**. Like RandLA, it is a lightweight and scalable architecture. We use ReconRLA to assign arbitrary query points a latent code. With this encoding we then use an implicit decoder to regress SDF values from which we can then construct a mesh by using Marching cubes. This pipeline is based on the work of Lombardi *et al.* [4] and is presented in detail in Section 3.1.

In summary our contributions include:

- We propose ReconRLA; A scalable, lightweight, learning-based and simple neural net that directly operates on the input point cloud and is easy to extend.
- By integrating ReconRLA as the encoder into our reconstruction pipeline we present a method that is scalable, grid-free and generalizes to large-scale real-world point clouds.
- With experiments we show qualitative and quantitative results that are on par with current state-of-the-art methods.

2 Related Work

Surface reconstruction has been studied in a plethora of works and a variety of different approaches have been introduced. In the following we give a small survey on some of the most important methods, but focus on implicit-based learning methods, since our model falls into this category. A small summary is presented in Table 2.1.

Method	Space	Out	Any Scale (Chunking)	Architecture	Latent Code	point-based (grid-free)
DeepSDF [5]	3D	Occ		AD	Global	
OccNet [8]	3D	Occ		ED	Global	
Chibane *et al.* [12]	3D	Occ		Grid ED	Local	
SSRNet [13]	2D	Occ	✓	U-Net ED	Local	
DeepLS. [9]	3D	SDF		Grid AD	Local	
Lombardi *et al.* [4]	3D	SDF	✓	U-Net VED	Local	
Ours	**3D**	**SDF**	✓	**U-Net (V)ED**	**Local**	✓

Table 2.1: Related work overview

2.1 Voxel-based Methods

Curless and Levoy [14] pioneered to use weighted sums of truncated signed distance functions (TSDF) within regular voxel grids. Further improvements counter outliers and noise [15, 16]. In order to close larger holes, surface regularization using L1-regularization [17] or Poisson Surface Reconstruction methods [18, 19] have been introduced. While these methods achieve great results, they have high computational resource demands and therefore don't scale well to larger scenes.

As in many other research areas, the rise of machine learning promised to produce new learning-based methods that may outperform classical approaches. With regression forests Ladicky *et al.* [20] was able to construct high resolution meshes in milliseconds. While this approach can be scaled to arbitrary large point clouds by splitting the space into a set of regions and then solve for each subproblem individually, global context information will be lost in the process. Methods [21, 22, 23] that try to process the whole input at once often apply an hierarchical volumetric grid and achieve qualitative results even from noisy and missing data. However due to costly volumetric convolutional operators these methods are slow for larger scenes and have a big memory footprint. Even with an advanced octree-based approach [24] that increases the voxel-grid resolution, the reconstructed shapes still seem blocky.

2.2 Probabilistic Methods

While synthetic data is usually noise-free, real-world point clouds often contain outliers and holes. Probabilistic approaches account for noise. Most promising, PSDF Fusion [25] explicitly keeps track of bidirectional sensor noise in order to tackle direction dependent noise. While these methods can be efficient, they don't adapt well if the error distributions differs from the often assumed gaussian noise distribution.

2.3 Implicit-based Methods

The disadvantage of most previous methods is the fixed resolution due to discretization of the voxel grid. Achieving a more detailed representation infers higher memory usage and computational effort. The idea for implicit-based methods is to use a deep network that learns to predict signed distance function (SDF) [4, 5, 9] or occupancy [8] values at arbitrary query points. A much higher (potentially infinitely higher) resolution can be achieved without a large increase in memory usage since the deep neural net has learnt a continuous field of SDF or occupancy values, that can be queried at arbitrary points. The challenge lies in implicitly representing the input point cloud without losing detail-information. The pioneering work of DeepSDF [5] could not represent fine details presumably due to the usage of only one global latent code. Since then many follow-up approaches focused on improving the implicit representation such that even high frequency features can be captured. While Sitzmann *et al.* [26] use periodic sinus activation functions, Amos *et al.* [27] improves the reconstruction quality through a better formulation of the loss function. Chibane *et al.* [12] extract with an hierarchical grid local and global latent codes, which are then used to make occupancy predictions in a decoding step. Similarly DeepLS [9] and Jiang *et al.* [28] store a grid of independent latent codes and show that their method achieves higher accuracy in reconstruction results than the traditional SDF fusion techniques. Finally Lombardi *et al.* [4] uses a permutohedral lattice paired with sparse convolutions in order to process the hierarchical features sparsely and efficient. In contrast to DeepLS [9] and Jiang *et al.* [28], who use an autoencoder architecture, Lombardi *et al.* [4] employ a variational encoder decoder structure, which enables them to directly process point clouds without having to first optimize a large amount of latent codes. Lastly Lombardi *et al.* [4] incorporate chunking of point clouds, similar to Mi *et al.* [13], which allows them to process arbitrary large scenes.
Although all of the mentioned scalable methods [4, 9, 28] achieve amazing results, they are all grid-based, where the input points are splatted onto grid points. The reconstruction performance is therefore dependent on the grid resolution. The smaller each grid cell, the more detailed is the reconstruction but the larger is the computational effort. Instead of a grid-based approach it would be advantageous to operate directly on the point cloud without having to store additional structure information. This could solve current memory problems and potentially be faster. To the best of our knowledge there exist no scalable learning-based reconstruction methods that operate directly on point clouds. However in the field of point cloud segmentation PointNet [10] pioneered a neural net that runs directly on point clouds. Since PointNet does not capture local structures many recent works [29, 30, 31, 32] introduced sophisticated neural modules that learn per-point local features. While these methods achieve astonishing results, they do not scale to large point clouds. RandLA [11] proposed a lightweight scalable approach. Instead of complex point selection they use random sampling

and introduce a local feature aggregation module. RandLA is up to 200x faster than other large-scale approaches like SPG [33]. Originally RandLA is only used for shape segmentation but we will use its core ideas to create ReconRLA, a scalable, point-based reconstruction method. Apart from being grid-free, our approach is much simpler than other scalable methods that rely on complex implementations (*e.g.* permutohedral lattice), which are difficult to further optimize.

3 Methods

3.1 Pipeline

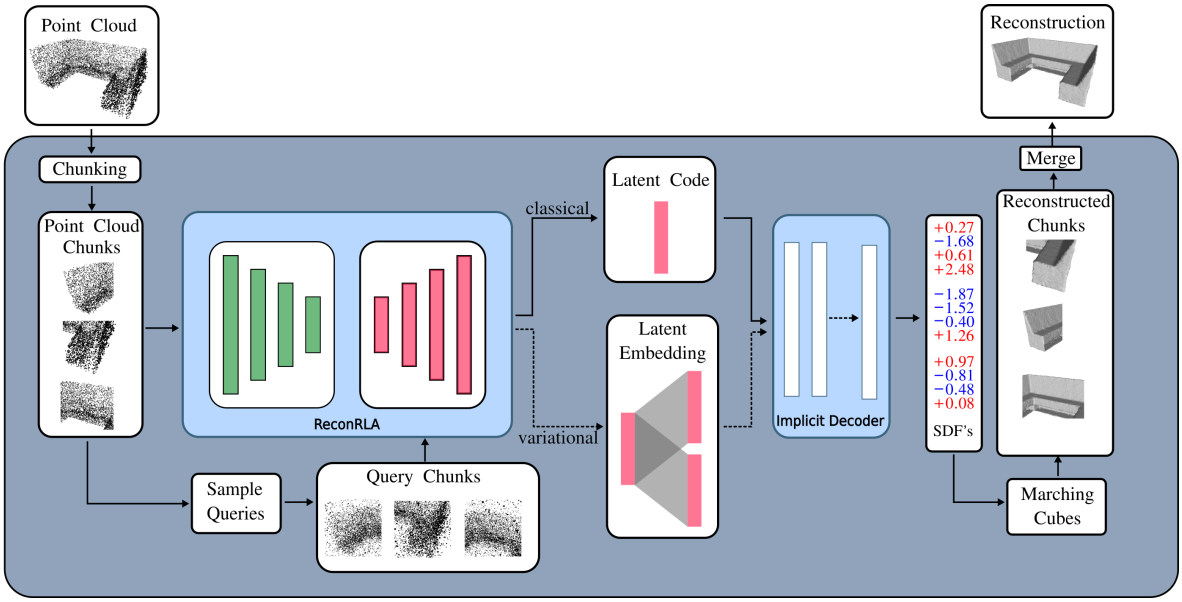


Figure 3.1: Given an input point cloud our pipeline produces a mesh-based reconstruction. We first chunk the point cloud in an octree fashion until each chunk has less than a threshold of points. We then process each chunk separately. The core idea is to generate query points for which we predict a SDF value. With the SDF values we can then use Marching cubes to extract the surface. In order to accurately predict the SDF value we first produce a latent code with ReconRLA by extracting hierarchical features from the input points, which are then pooled to find an informative feature vector for each query point. We then input the latent code into an implicit decoder that returns us a predicted SDF value.

We are using a very similar pipeline 3.1 as presented in Lombardi *et al.* [4]. The input of our pipeline is an oriented point cloud $\mathbf{P} = \{(\mathbf{p}_i, \mathbf{n}_i) \mid i \in \{1, ..., N\}\}$ where $\mathbf{p}_i \in \mathbb{R}^3$ are the point coordinates and $\mathbf{n}_i \in \mathbb{R}^3$ the normals. Additionally we have a set $\mathbf{Q}$ of query points $\{\mathbf{q}_j \in \mathbb{R}^3 \mid j \in \{1, ..., M\}\}$ for which we wish to regress a signed distance function (*SDF*) value. The SDF value of query point $\mathbf{q}_j$ determines the distance to the closest surface point. The sign indicates whether $\mathbf{q}_j$ lies inside (negative) or outside (positive) of the surface. Figure 3.2 gives a visualization. Notice that at the zero-level lies the surface. In order to predict the SDF values we employ an encoder-decoder network structure in which we use ReconRLA as the encoder and the implicit decoder from Lombardi *et al.* [4] as the decoder. The goal of our encoder is to find a latent code for each query point. A more detailed explanation of our

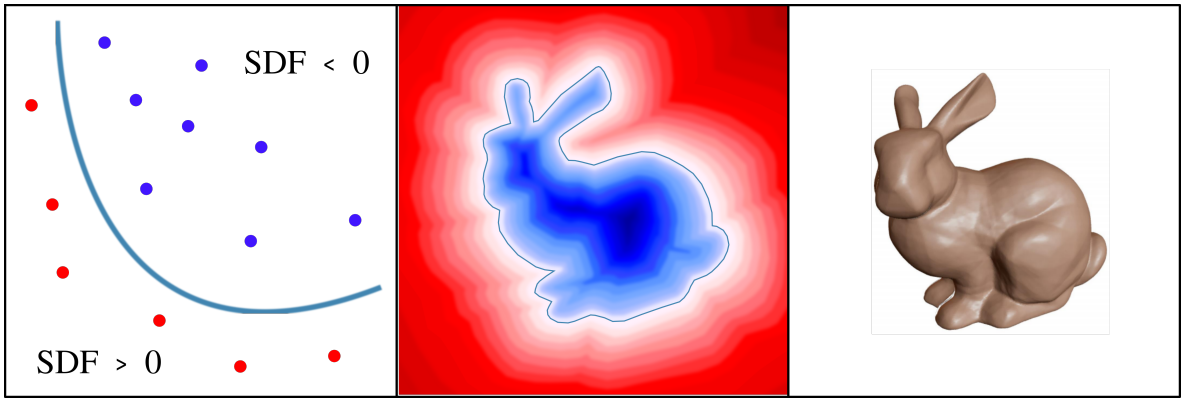


Figure 3.2: (Left): Visualization of the implicit surface. Blue points lie inside the bunny and have therefore a negative SDF value. Red points lie outside and are therefore positive. The surface lies where SDF = 0. (Middle): 2D cross-section of the learnt SDF field. (Right): Depiction of the 3D Stanford Bunny.

encoder is presented in Section 3.2. However the core idea is to use an hierarchical feature map to extract 3D features from the input point cloud at progressively coarser scales, which we then use to assign each query point an encoding. This gives us a crucial advantage to reconstruct finer details and larger scenes more accurately in comparison to methods that use only one latent code [5, 8].
While the encoder first produces a latent code for each query point, the decoder then uses the latent code to predict the SDF values. Ideally we would query infinitely many points to get a continuous field of SDF values with an infinite resolution zero-level iso-surface. Since this is computationally intractable we limit us to a number of query points. After we have predicted all SDF values we use Marching cubes [34] to generate a mesh at the zero-level of the signed distance function that represents our surface. During training we use point clouds that can be directly processed by this pipeline. However at inference time we can encounter point-clouds that are too large. Like Mi *et al.* [13] we divide too large point clouds into Kchunks $\mathbf{P}_k \subset \mathbf{P}$ with $k \in \{1, ..., K\}$ and process them individually. At the end we merge the reconstructed meshes of each chunk to get our final reconstruction.

3.2 ReconRLA Overview

In this section we explain how we use ReconRLA (Figure 3.3) as our encoder. ReconRLA is an altered version of RandLA [11] that we adapt into our pipeline. As Figure 3.3 depicts, ReconRLA has an encoder-decoder structure. During the encoding layers we construct an hierarchical feature map from the input point cloud. While in the first layers we generate fine grained features that represent detailed local neighborhood information, we progressively store more and more coarse grained features in the following encoding layers. This way we have generated features for different scales, which enables us to implicitly represent larger scenes and more details than we could have by using only one global latent code. In order to process point clouds of potentially millions of points we downsample the points in each encoding layer. This is reasonable since the more coarse grained our features vectors get, the less feature vectors we need to describe the whole scene. On the other hand we need many

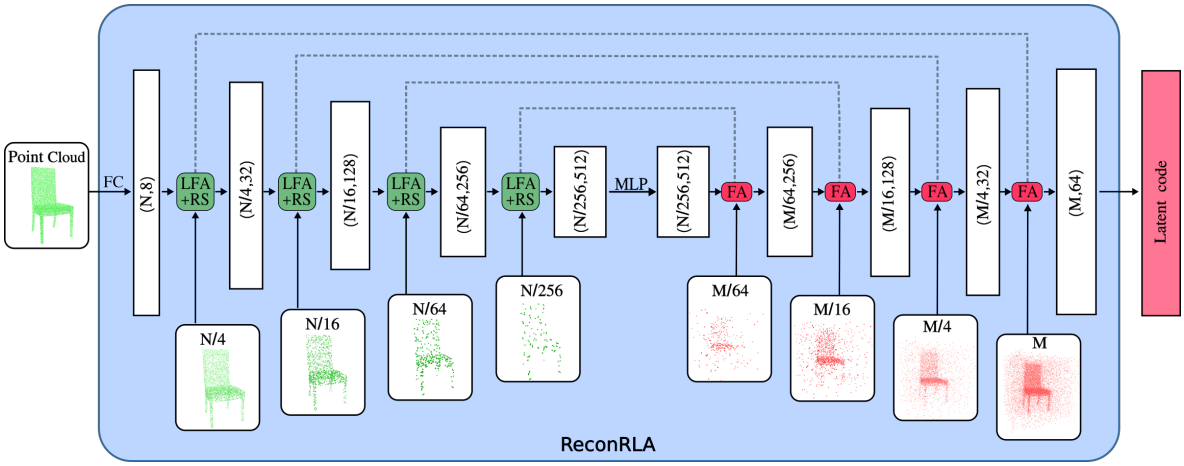


Figure 3.3: ReconRLA network architecture. The first layer is a fully connected linear layer. Afterwards follow four encoding layer to build up the hierarchical feature map. In each layer we have a local feature aggregation module (LFA) and random sampling (RS). After the MLP in the middle of the network follow four decoding layer in which we progressively pool and aggregate features from the hierarchical feature map for each query point (FA module). In the end we output a latent code for each query point.

feature vectors to represent all the different local neighbourhoods. In summary we build an hierarchical feature map with progressively less feature vectors the larger the scale of the neighbourhood. We can then use the hierarchical feature map to assign each query point an encoding over the spawn of our decoding layers. The idea is to pool features from each scale of our hierarchical feature map and then aggregate the pooled feature vectors at our given query location. We assume that feature vectors of points that are close in the Euclidean space to our query point contain all the needed information about $\mathbf{q}$'s neighbourhood to reliably predict the correct SDF value. Since we pool from different scales, each query gets global context information from coarse grained feature vectors and detailed local neighbourhood information from the fine grained feature vectors. In contrast to the encoding layers we reverse the sampling process. Instead of beginning with all queries we only start with a few and progressively upsample in order to make our approach scalable to millions of query points.

3.2.1 Encoding Layer

Each encoding layer produces one layer of our hierarchical feature map. While the first encoding layer produces very fine grained feature vectors for small details, the last encoding layer generates large-scale features that represent global scene information. In order to make our downsampling efficient for large point clouds we employ random sampling instead of more sophisticated but expensive sampling techniques. Since key features could be lost during the random sampling process we use a local feature aggregation module as presented in RandLA [11] and explained in the subsection 3.2.3. In summary each encoding layer performs a local feature aggregation module (LFA) and random sampling.

3.2.2 Decoding Layer

In each decoding layer we assign the given query points of that layer a feature vector by pooling encodings from the hierarchical feature map (skip connection) and combining them with the features, that we found in the previous layer. We abstract this pooling and aggregation step with a feature aggregation (FA) module. There exist a multitude of different design choices. We present a naive implementation in Section 3.3, that led to bad reconstruction results, and a more involved version in Section 3.4, which we are currently using.

3.2.3 Local Feature aggregation Module (LFA)

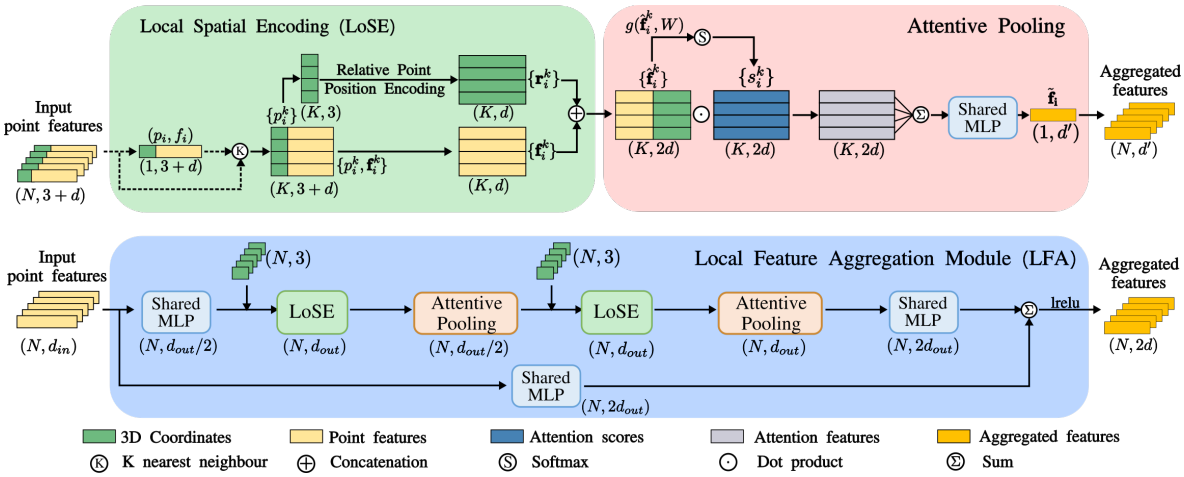


Figure 3.4: Local feature aggregation module. The local spatial encoding first extracts features from the local neighbourhood and then the attentive pooling weights each feature vectors importance and aggregates the encodings accordingly. By chaining these components together twice we have created our local feature aggregation module (LFA).

The main goal of the LFA module (Figure 3.4) is to give each point an informative feature vector. At the start every point $\mathbf{p}_i$ knows only about his spatial location and nothing about its surrounding. In order to give $\mathbf{p}_i$ an informative encoding, we need to increase the respective field in order to have a better understanding about the local neighbourhood. This additional information should gives us clues to where we are in the scene (*e.g.* Inside/Outside of the object, far away from the surface *etc.*). Additionally by storing key features of $\mathbf{p}_i$'s surrounding neighbours, we are keeping valuable information even when neighbouring points are lost due to the random sampling. Since we apply a LFA module in every encoding layer, we progressively increase the respective field of our points. In order to achieve this goal the LFA module employs two key neural units: Local spatial encoding (LoSE) and Attentive Pooling, which are chained together to increase the respective field. In Figure 3.5 we visualize the two-times (K^2) increase of the respective field.

The following paragraphs explains the design of the LoSE and attentive pooling module in more detail. At the beginning we have a point cloud $\mathbf{P}_k$ where every point $\mathbf{p}_i$ has an assigned feature vector $\mathbf{f}_i$. We then apply the LFA module to each point in parallel. The local

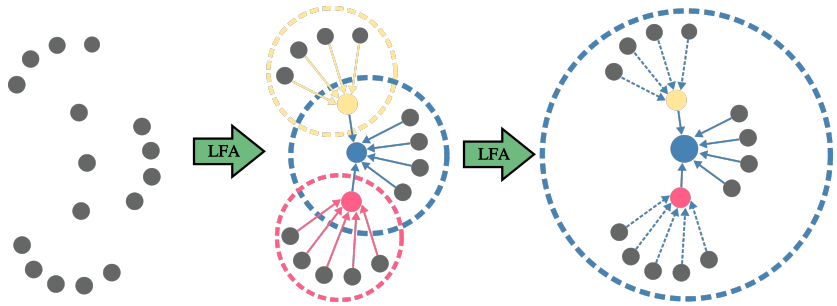


Figure 3.5: Illustration of the increase of the respective field (dotted circle) of each point through the application of the LFA module. In the last blue dotted circle the blue point has aggregated the features of all his surrounding points, which gives an informative feature vector about this point's local neighbourhood.

spatial encoding first gathers the K nearest neighbours and then appends each neighbour an additional feature vector $\mathbf{r}_i^k$ (relative point position encoding) that encodes the relative spatial locations of each neighbour.

$$\mathbf{r}_i^k = MLP(\mathbf{p}_i \oplus \mathbf{p}_i^k \oplus (\mathbf{p}_i - \mathbf{p}_i^k) \oplus \|\mathbf{p}_i - \mathbf{p}_i^k\|) \tag{3.1}$$

Where $\mathbf{p}_i$ and $\mathbf{p}_i^k$ are the coordinates, $\oplus$ denotes concatenation and the $\|\cdot\|$ represents the Euclidean distance.
By embedding the spatial locations of the neighbouring points, we allow the corresponding feature vectors to be aware of their relative spatial location to our point $\mathbf{p}_i$. This gives us information about the whole neighbourhood of our point. We now want to use all of this information to create a new feature vector for $\mathbf{p}_i$. Theoretically we could just use a common aggregation like Sum or Max-pooling. However we want to give the network a chance to decide for its own which neighbours features are most important. Therefore we apply the attentive pooling module. By applying an MLP $(g(\hat{\mathbf{f}}_i^k, \mathbf{W}))$ and a soft-max we are producing attention scores, that should represent the importance of each neighbours feature vector. After Multiplying the attention scores with the original feature vectors we can now take the sum over the encodings as our aggregation method. As a result we now have for each point a new feature vector $\tilde{\mathbf{f}}_i$ that is a combination of the K nearest neighbours and has increased the respective field of $\mathbf{p}_i$ to its K nearest neighbours. We can repeat this to further increase the respective field. This time we gather feature vectors, which are the result of the previous aggregation step.

3.3 Naive FA-Module

In this section we describe a naive design for the feature aggregation (FA) module that we employ in each decoding layer D_i with $i \in \{1, ..., k\}$ where k denotes the number of decoding layers. Figure 3.6 gives a visualization of the design. In order to find an encoding for a query $\mathbf{q}$ in layer D_i we use the information from the hierarchical feature map (skip connection) SC and the feature encodings of the points from the previous layer D_{i-1} which we name LB (Layer Before). Under the assumption that points close to each other in space have a similar encoding, a naive implementation is to aggregate the encoding $\mathbf{f}_q^{LB}$ of the nearest neighbour

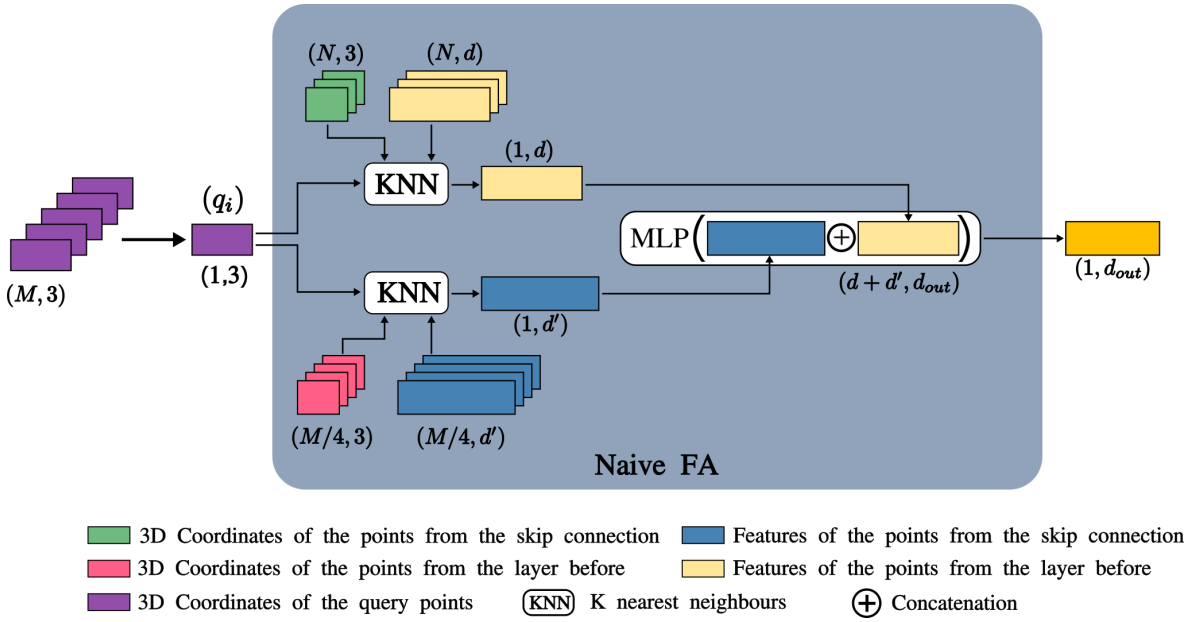


Figure 3.6: Naive FA module. For decoding layer D_i and query $\mathbf{q}$ we take the feature encoding of the point closest to $\mathbf{q}$. We do this both for the Skip connection points and the points from the layer before D_{i-1}. A simply aggregation by concatenation and MLP returns us the aggregated feature vector for $\mathbf{q}$.

$\mathbf{p}_q^{LB}$ of LB and the encoding $\mathbf{f}_q^{SC}$ of the nearest neighbour $\mathbf{p}_q^{SC}$ of SC with respect to query point $\mathbf{q}$. A zero loss aggregation of the two feature vectors is to just concatenate them. Since this would give us a high dimensional vector we reduce the dimensionality by applying an MLP. The following equation summarizes this process:

$$\mathbf{f}_q = MLP(\mathbf{f}_q^{LB} \oplus \mathbf{f}_q^{SC}) \tag{3.2}$$

Here $\mathbf{f}_q$ denotes the resulting feature vector for query point $\mathbf{q}$.
We apply this FA module in all decoding layers. However notice that in D_1 (The decoding layer with the most down sampled query point cloud) LB does not hold feature encodings of query points but from points from the input point cloud.

3.3.1 Reconstruction Results

As one can see in Figure 3.7 this approach does not produce qualitative results. Large parts of the surfaces are not reconstructed.

3.3.2 Reason for Failure

In each decoding layer D_i we pool the feature encoding of the nearest neighbour of LB and of SC. The nearest neighbours feature vectors are the only random variable in our naive FA module since we proceed by a deterministic concatenation and MLP application (Equation 3.2). Therefore queries with the same nearest neighbour from LB and SC end up with an identical feature encoding, which results in the same SDF value. Observe that this can happen to queries inside the surface, which should have a negative SDF value, and queries outside

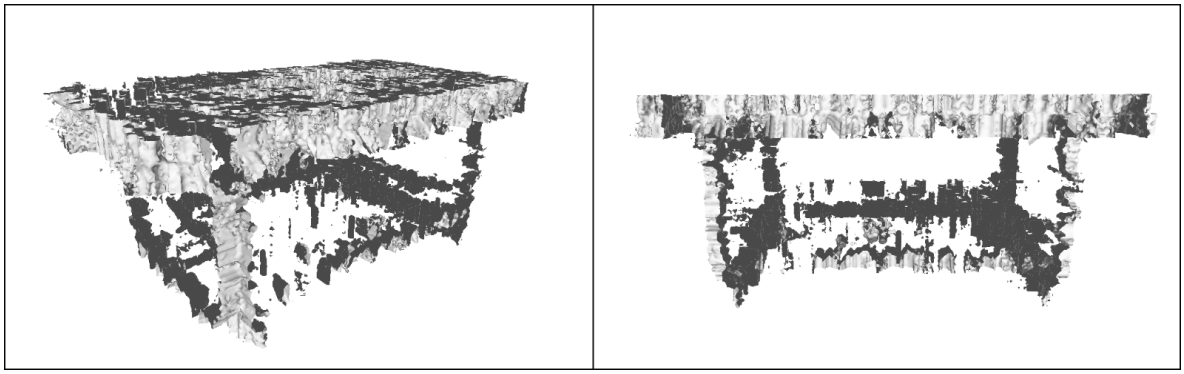


Figure 3.7: Reconstruction results of a table with the naive implementation of the FA module. Observe that large parts are not reconstructed.

the surface, which should have a positive SDF value. Figure 3.8 gives a visualization of this setting. We argue that if query point $\mathbf{q}_1$ lies on the outside of the surface and $\mathbf{q}_2$ mirrored on

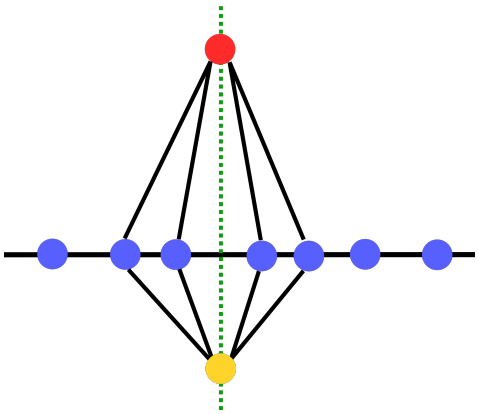


Figure 3.8: Visualization for Failure of the naive FA implementation. The red query lies outside of the shape and would therefore have a positive SDF value. The yellow query lies inside and should get a negative value. However since they have the same neighbours, they will end up with the same encoding, which results in the same SDF value.

the surface on the inside, then it is often the case that $\mathbf{q}_1$ and $\mathbf{q}_2$ have the same nearest point cloud neighbour and therefore the same feature encoding. This is very harmful since $\mathbf{q}_1$ and $\mathbf{q}_2$ should have opposite values and our model can not find a zero-level, which results in no surface reconstruction. We believe that this explains the large holes in the reconstructions. Further the probability of two queries having the same neighbours is increased in smooth flat areas while for very detailed regions the likelihood drops. Therefore we can see some surface reconstruction at edges and corners.
While this problem is most prominent in the first decoding layer it will not disappear for the other layers. Additionally, problematic encodings carry over through all layers. In the end we find ourselves with many encodings that are identical even for queries from different sides of the surface.

3.4 ReconRLA FA module

In this section we present the design of our current FA module. In the previous Section 3.3 we showed that a naive implementation does not produce acceptable reconstruction results. In order to fix the problem we need to ensure that distinct query points end up with different encodings. Our first intuition was to pool multiple nearest neighbours instead of just one. Therefore we take K nearest neighbours from LB and K from SC. Hence we have 2K encodings of points that are close in space to our query. In order to only have one feature vector for our query points we could average the 2K encodings. Initially we thought that this should fix the problem. However if we take a closer look at the same example from before (Figure 3.8) we realise that this does not fix the issue. Queries that are mirrored on the surface will still have a high chance to have the same neighbours and therefore the same encoding. To fix this problem we can't use simple interpolation. Even weighted interpolation based on the distance to the query point could still be troublesome since the distance of mirror queries is the same.

Our approach's core idea is to reuse the principle of the local feature aggregation module from the encoding layers. Given a query $\mathbf{q}$ we want to aggregate the features of $\mathbf{q}'s$ nearest neighbours together with their spatial information and then use an attentive pooling to weight the importance of each point's feature vector in order to produce a final encoding through an aggregations step. In the following we describe the detailed implementation of our FA module. Figure 3.9 gives a visualization without the attentive pooling. In Figure 3.10 (Original FA) we depict an overview of the whole FA module.

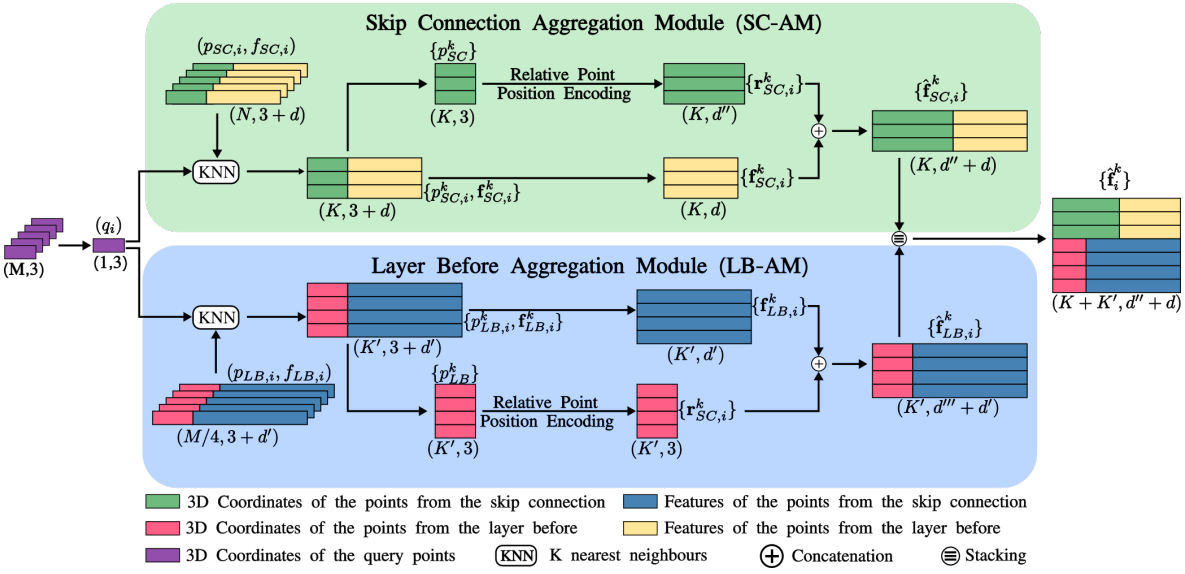


Figure 3.9: ReconRLA FA module. We pool features both from the skip connection and from the previous layer. We use the relative point position encoding to add more spatial information of our queries location relative to the corresponding points. After stacking the feature vectors we apply an attentive pooling module, which is left out in this visualization for better readability. We end up with a single aggregated feature vector for our query.

Given is the query point cloud $\mathbf{Q}$. All queries are processed simultaneously however we will explain it for a particular query point $\mathbf{q}_i$. We know that $\mathbf{q}_i$ lies somewhere in the Euclidean space. Similarly all points from the layer before $\mathbf{p}^k_{LB}$ already have an encoding and are also somewhere in this space. The same holds for the points from the skip connection $\mathbf{p}^k_{SC}$. The idea is now to use the encoding of the closest points in space from $\mathbf{q}_i$. We find the K nearest points of the skip connection SC points and the K' nearest from the point of the layer before LB. Notice that K and K' can be different. We append a relative point position encoding to the found feature vectors and stack them in order to apply attentive pooling (As presented in Section 3.2.3). However the feature vector from the skip connection has a different dimension than the ones from the layer before. There exist multiple solutions to this problem. We chose to use the MLP of the relative point position encoding as a gateway to have the same dimension. The feature vector from a point of the layer before $\mathbf{f}^k_{LB}$ is much larger than the feature vector of a point from the skip connection $\mathbf{f}^k_{SC}$. However for both points we will create a relative point position encoding $\mathbf{r}^k_{LB}$ and $\mathbf{r}^k_{SC}$. By choosing a different sized MLP we can create the relative point position encoding in a way that:

$$|\mathbf{r}^k_{LB}| + |\mathbf{f}^k_{LB}| = |\mathbf{r}^k_{SC}| + |\mathbf{f}^k_{SC}| \tag{3.3}$$

Here $|\cdot|$ denotes the dimensionality of the given vector.
We concatenate the relative point position encoding with the feature vector in order to get $\hat{\mathbf{f}}^k_{LB}$ and $\hat{\mathbf{f}}^k_{SC}$. Due to our relative point position encoding we now achieved equal dimensionality of $\hat{\mathbf{f}}^k_{LB}$ and $\hat{\mathbf{f}}^k_{SC}$, which allows us to stack them. This gives us K+K' feature vectors that describe the surrounding neighbourhood of $\mathbf{q}_i$. As a last step we apply the in Section 3.2.3 presented attentive pooling in order to aggregate the different feature vectors to one final encoding.

With this new FA-module we can ensure that distinct queries receive different latent codes and our shape reconstruction significantly improved. We present qualitative and quantitative results in chapter 4.

3.5 2x Aggregation FA

In the LFA module of the encoding layers we applied the Local Spatial Encoding (LoSE) and attentive pooling two times (Figure 3.4). This design choice increases the respective field of each point to K^2 points. Naturally we can also apply this idea to our FA module in the decoding layers. We visualize this architecture in Figure 3.10. After the attentive pooling from the first feature aggregation we have found an encoding for all of our query points. Now that we have found an encoding for all our query points of the current decoding layer, we can apply another feature aggregation step by chaining together a Local Spatial Encoding and an attentive pooling module. This results in an increase of the respective field of each query point but also increases the computational effort.
How this FA module with two aggregation and pooling steps compares to our original version is presented in Section 4.5 of chapter 4.

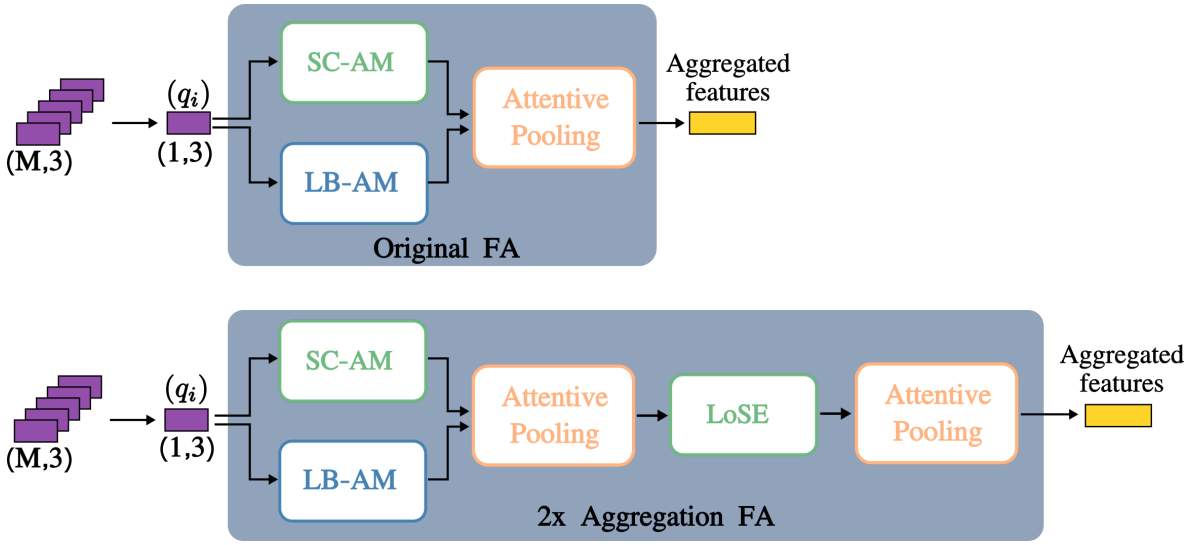


Figure 3.10: Visualization of a 2x aggregation module. SC-AM: Skip Connection Aggregation Module. LB-AM: Layer Before Aggregation Module. After the first attentive pooling we have found for every query point an aggregated feature vector. In order to further increase the respective field we perform another local feature aggregation (LoSE) chained with an attentive pooling.

3.6 Variational Approach

ReconRLA acts as an encoder of the query point cloud and produces latent code, which is then used in the implicit decoder to infer per-point SDF values. This encoding decoding structure can be transformed into a variational encoder decoder (VED). We input a query $\mathbf{q}_i$ and point cloud $\mathbf{P}_k$ and output a predicted SDF value $\hat{s}_i$. Let the learnt network be $f_{\phi,\theta,\theta'}$ where ϕ and θ represent the parameters of encoder enc_ϕ and decoder dec_θ.

$$f_{\phi,\theta,\theta'}(\mathbf{q}_i, \mathbf{P}_k) = \hat{s}_i \tag{3.4}$$

Notice that in contrast to a variational autoencoder (VAE) our input and output are different. However the same theories and formulations of VAE can be applied to VED with minor changes. The reconstruction error (loss function) is defined as the similarity of the true (s_i) and predicted ($\hat{s}_i$) SDF value. In the following we describe how we model the prior, likelihood and posterior distribution of our VED. Figure 3.11 depicts a visualization. For our VED we model the prior of the latent code as a standard multivariate gaussian.

$$\text{Prior: } p_{\theta'}(z) = N(0, I) \tag{3.5}$$

The dimensionality is a hyperparameter, which we choose to be 64.
For the likelihood we choose a multivariate Gaussian with the parameters μ_θ and covariance matrix Σ_θ, which are determined by our decoder dec_θ.

$$\text{Likelihood: } p_\theta(s_i \mid z) = N(\mu_z, \Sigma_z) \tag{3.6}$$

Since this likelihood function is a complicated neural net we can't analytically calculate the posterior $p_\phi(z|x)$, which should be our encoding step. Instead we model the encoder as

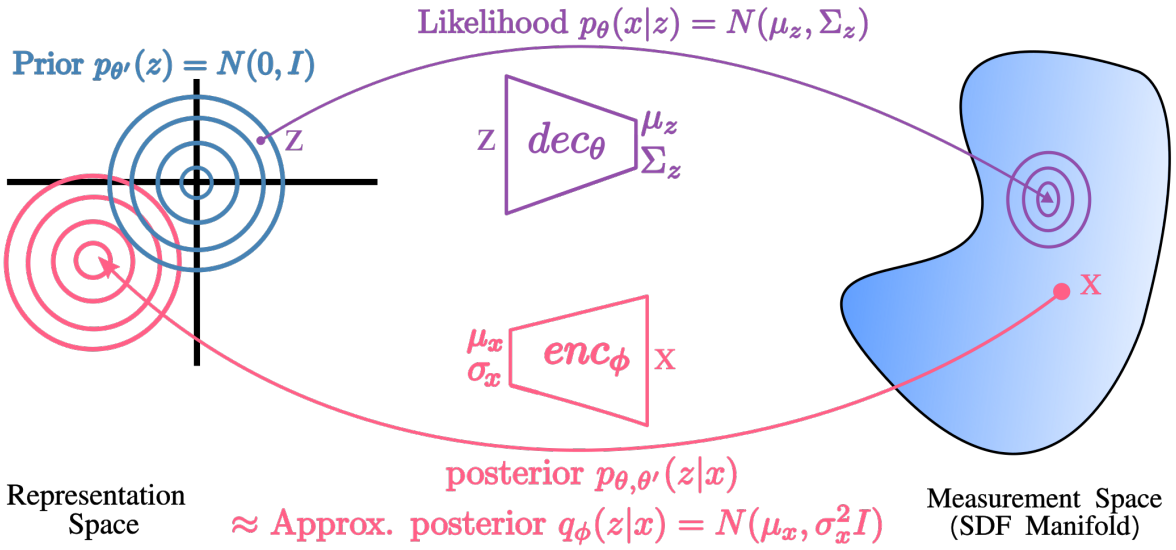


Figure 3.11: For each query sample $\mathbf{x}$ in the measurement space we try to find a meaningful feature vector z_i (latent code) in the representation space, which we can then decode to get our predicted SDF value.

another neural net that approximates the true posterior. Since prior and likelihood are both Gaussian's we know that the posterior will also be a Gaussian. The distribution parameters μ_ϕ and σ_ϕ of our approximate posterior are calculated by our encoder. We assume a diagonal Covariance matrix, in order to keep the dimensionality linear.

$$\text{Approx. Posterior: } q_\phi(z \mid \mathbf{q}_i, \mathbf{P}_k) = N(\mu_i, \sigma_i) \tag{3.7}$$

3.6.1 Training our VED

Following Kingma and Welling [35] we optimize $p_{\theta,\theta'}(s_i)$ with respect to our parameters over the whole training set.

$$arg\max_{\theta,\theta',\phi} \prod_{i=1}^{M} p_{\theta,\theta'}(s_i) \tag{3.8}$$

$$arg\max_{\theta,\theta',\phi} \sum_{i=1}^{M} \log\left(p_{\theta,\theta'}(s_i)\right) \tag{3.9}$$

Unfortunately $p_{\theta,\theta'}(s_i)$ is intractable due to the integral over z.

$$p_{\theta,\theta'}(s_i) = \int p_{\theta,\theta'}(s_i|z) p_{\theta,\theta'}(z) d_z \tag{3.10}$$

However we can derive an evidence lower bound (ELBO) that we can maximize instead.

$$arg\max_{\theta,\theta',\phi} \sum_{i=1}^{M} E_{z\sim q_\phi(\cdot|\mathbf{q}_i,\mathbf{P}_k)} \left[\log\left(\frac{p_{\theta,\theta'}(s_i, z)}{p_{\theta,\theta'}(z|s_i)}\right)\right] \tag{3.11}$$

$$arg\max_{\theta,\theta',\phi} \sum_{i=1}^{M} E_{z\sim \mathbf{q}_\phi(\cdot|\mathbf{q}_i,\mathbf{P}_k)} \left[\log\left(\frac{p_{\theta,\theta'}(s_i,z)}{p_{\theta,\theta'}(z|s_i)} \right) \cdot \frac{q_\phi(z|\mathbf{q}_i,\mathbf{P}_k)}{q_\phi(z|\mathbf{q}_i,\mathbf{P}_k)} \right] \tag{3.12}$$

$$arg\max_{\theta,\theta',\phi} \sum_{i=1}^{M} \underbrace{E_{z\sim q_\phi(\cdot|\mathbf{q}_i,\mathbf{P}_k)} \left[\log\left(\frac{p_{\theta,\theta'}(s_i,z)}{q_\phi(z|\mathbf{q}_i,\mathbf{P}_k)} \right) \right]}_{elbo_{\theta,\theta',\phi}(s_i)} + \underbrace{E_{z\sim q_\phi(\cdot|\mathbf{q}_i,\mathbf{P}_k)} \left[\log\left(\frac{q_\phi(z|\mathbf{q}_i,\mathbf{P}_k)}{p_{\theta,\theta'}(z|s_i)} \right) \right]}_{KL\left(q_\phi(z|\mathbf{q}_i,\mathbf{P}_k)\|p_{\theta,\theta'}(z|s_i)\right)\geq 0} \tag{3.13}$$

The Kullback-Leiber (KL) divergence is always greater than zero. We maximize the ELBO:

$$elbo_{\theta,\theta',\phi}(s_i) = \underbrace{E_{z\sim q_\phi(z|\mathbf{q}_i,\mathbf{P}_k)} \left[\log p_\theta(s_i|z)\right]}_{\text{Infomax}} + \underbrace{E_{z\sim q_\phi(\cdot|s_i)} \left[\log\left(\frac{p_{\theta'}(z)}{q_\phi(z|s_i)} \right) \right]}_{-KL\left(q_\phi(z|\mathbf{q}_i,\mathbf{P}_k)\|p_{\theta'}(z)\right)} \tag{3.14}$$

$$arg\max_{\theta,\theta',\phi} \sum_{i=1}^{M} E_{z\sim q_\phi(z|\mathbf{q}_i,\mathbf{P}_k)} \left[\log p_\theta(s_i|z)\right] - KL\left(q_\phi(z|\mathbf{q}_i,\mathbf{P}_k)\|p_{\theta'}(z)\right) \tag{3.15}$$

The second term ensures that our approximate posterior $q_\phi(z|\mathbf{q}_i,\mathbf{P}_k)$ is consistent with our prior $p_{\theta'}(z)$. Graphically this corresponds to that the two distributions are on top of each other. The first term is the expected reconstruction error, hence a measure of the similarity of the true SDF value s_i and our predicted SDF value $\hat{s}_i$.

$$p_{\theta,\theta'}(s_i|z) = e^{-\mathcal{L}(\hat{s}_i,s_i)} \tag{3.16}$$

Here $\mathcal{L}(\hat{s}_i,s_i)$ denotes the loss function. We take the absolute difference between our predicted value $\hat{s}_i = f_{\phi,\theta,\theta'}(\mathbf{q}_i,\mathbf{P}_k)$ and the ground truth s_i, but truncate the SDF value into the range $[-\delta,\delta]$ with $\text{clamp}(s,\delta) = \min(\delta \max(-\delta,s))$.

$$\mathcal{L}(\hat{s}_i,s_i) = |\text{clamp}(\hat{s}_i,\delta) - \text{clamp}(s_i,\delta)| \tag{3.17}$$

With the loss function plugged in we get:

$$arg\max_{\theta,\theta',\phi} \sum_{i=1}^{M} -\mathcal{L}(f_{\phi,\theta,\theta'}(\mathbf{q}_i,\mathbf{P}_k),s_i) - KL\left(q_\phi(z|\mathbf{q}_i,\mathbf{P}_k)\|p_{\theta'}(z)\right) \tag{3.18}$$

Instead of maximizing we can minimize the negative. We get our final objective function:

$$arg\min_{\theta,\theta',\phi} \sum_{i=1}^{M} \mathcal{L}(f_{\phi,\theta,\theta'}(\mathbf{q}_i,\mathbf{P}_k),s_i) + KL\left(q_\phi(z|\mathbf{q}_i,\mathbf{P}_k)\|p_{\theta'}(z)\right) \tag{3.19}$$

3.6.2 Latent space for SDF values

When using a VED we regularize the latent space, such that similar inputs will result in similar encodings and vice versa. For a given query $\mathbf{q}_1$ and point cloud $\mathbf{P}$ we get an encoding z_1. If we now change the input a little (e.g. shift the query point a bit) we expect to have a similar encoding z_2. Remember that for our given query $\mathbf{q}_1$ the respective field will have the largest influence on the queries encoding. We therefore reason that queries with a similar respective field (e.g. at a corner of the point cloud) have similar encodings.

3.7 Training

For the training of our network we use the ShapeNet v2 dataset [1] to create point clouds, queries and ground truth SDF values as described by Lombardi *et al.* [4]. We are using 400 shapes each from the following five categories: chair, table, plane, lamp and sofa. It is worth noting that the training samples generated from ShapeNet are synthetic and are therefore without noise and outliers. In order to make our pipeline applicable for real-world point clouds that often contain discontinuities, we apply multiple perturbations to our data samples. We introduce point cloud holes, outliers, varying levels of noise and rotate our training shapes. Additionally only half of our queries are sampled near the surface, whereas the other query points are random points in space. Lastly we scale each point cloud to the unit cube.
For our neural net we use the Adam optimizer [36] with a constant learning rate of $1e^{-3}$ and batch size of 1. Other learning rates were considered however the reconstruction results were significantly worse for other learning rates. Since each epoch has 2000 shapes we only trained the net for 5 epochs. Our best performing model is not generic. However we also consider a variational approach for which we trained for 50 epochs. All other settings are the same. We compare both models in chapter 4.

3.8 Inference

The input point clouds can contain millions of points during inference time. We therefore apply an octree onto the point cloud, where we recursively subdivide the point cloud into eight smaller chunks until every leaf has less points than a chosen threshold. Each chunk will be processed individually. First the points are normalized to the unit cube as the training was done on normalized point clouds. Afterwards query points are generated near the point cloud points. We then apply our neural net to predict a SDF value for each query point. In order to produce the mesh we then run Marching Cubes [34] on the SDF values in an online fashion. As a final step the different meshes of all chunks are merged together to produce the surface representation.

4 Experiments

The pipeline is implemented in pytorch with the help of some open-source libraries [37, 38]. For all experiments we used an NVIDIA RTX 3090 GPU and AMD Ryzen 9 5900X CPU. In Section 4.1 we show that our method performs well on synthetic point clouds. We then demonstrate in Section 4.2 that our neural net can qualitatively reconstruct much larger scenes. The most challenging task for 3D shape reconstruction are real-world point clouds that can contain noise and outliers. We show how our pipeline performs on real-world data in Section 4.3. In Section 4.4 we analyze the inference time of our pipeline. Lastly we explore the effect of different hyperparameters of our network in Section 4.5.

4.1 Performance on ShapeNet

Figure 4.1: Reconstruction results on ShapeNet [1]

We present qualitative results on five ShapeNet [1] categories: chair, table, plane, lamp and sofa. Figure 4.1 shows one sample from each category.

For comparison with other state-of-the-art object reconstruction methods we use Lombardi *et al.* [4], DeepSDF [5] and OccNet [8]. While we use the pretrained model of OccNet, we have trained Lombardi *et al.* and DeepSDF form scratch according to their instructions. Figure 4.2 gives a qualitative comparison. Observe that we reconstruct finer details than DeepSDF and OccNet, presumably due to the usage of an hierarchical feature map instead of a single global latent code. Visually our method seems to be on par with Lombardi *et al.* In the Appendix 5 we provide further qualitative comparisons between our method and Lombardi *et al.*

In order to show quantitative results, which we report in Table 4.1, we reconstruct 1K (200 from each category) test shapes and report a series of different metrics.

Intersection of Union (IoU)
With IoU we try to measure the reconstruction accuracy. Let I be the intersection of the ground truth volume with the prediction volume and let U be the union of the two volumes.

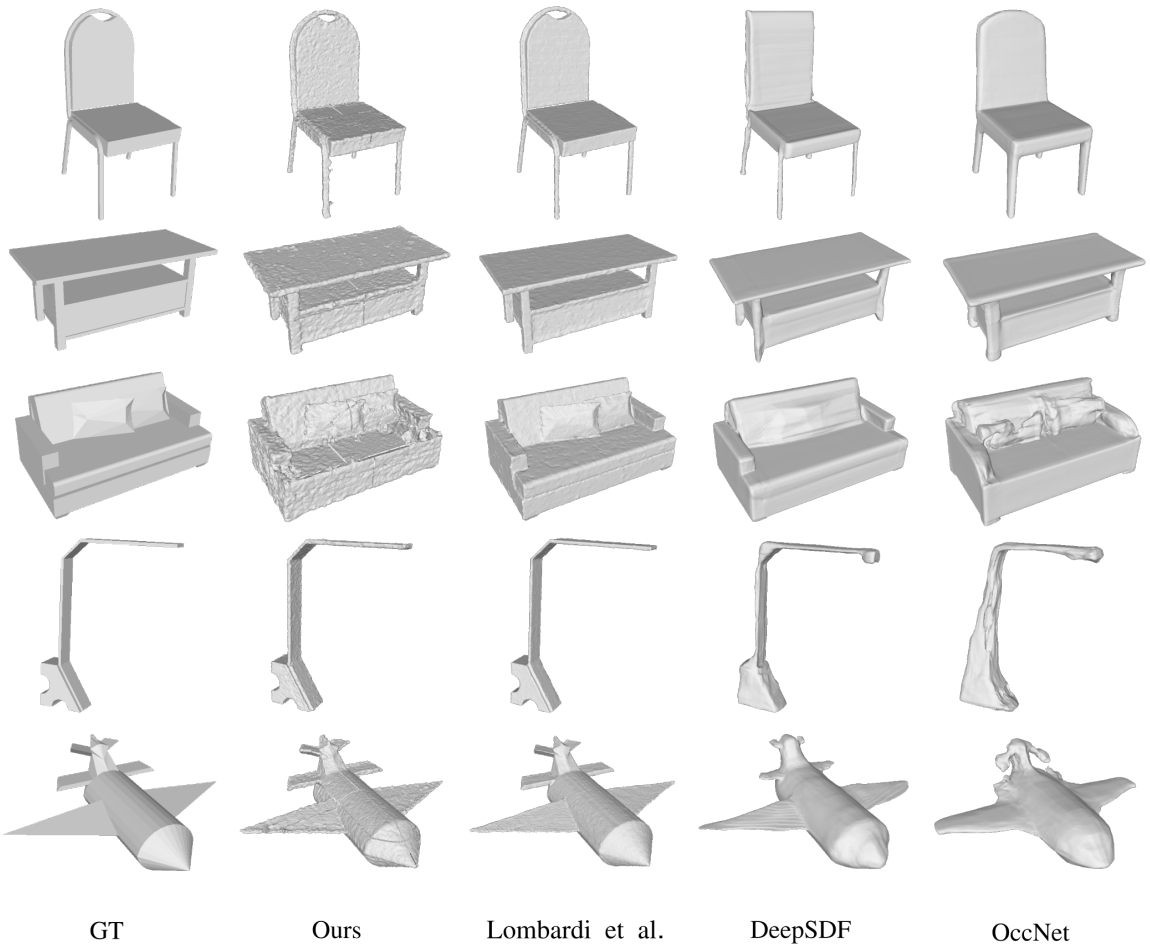


Figure 4.2: Comparison to other state-of-the-art reconstruction methods. Reconstruction results on some ShapeNet [1] samples. Our method reconstructs finer details (e.g. Handle of the chair, small nob at the base of the lamp) than DeepSDF [5] and OccNet [8]. Additionally we don't oversmooth spiky surfaces like the wings of the airplane.

Then the IoU is defined as the fraction:

$$IoU = \frac{I}{U} \tag{4.1}$$

We compute the IoU over occupancies, which we can infer by the sign of our SDF value.

Hausdorff Distance (HD)
The Hausdorff Distance is defined as the distance of the furthest point-to-point distance between the predicted and ground truth point cloud.

Normal Consistency (ND)
With the normal consistency we try to measure how well the predicted and the ground truth meshes align. In our case we have our predicted surface S_p and the ground truth surface S_{gt}.

	IoU ↑	HD↓	NC ↑	CD↓
DeepSDF	**0.82895**	0.08987	**0.89980**	0.00687
OccNet	0.59534	0.09311	0.085924	0.01332
Ours	0.77000	**0.04875**	0.88922	**0.00620**
Lombardi *et al.*	**0.91355**	**0.03370**	**0.93908**	**0.00439**

Table 4.1: Quantitative results on the ShapeNet v2 dataset [1]. We outperform DeepSDF in two and OccNet in all metrics.

The normal consistency is calculated as the average dot product between the normals of S_p and the corresponding nearest neighbour normals of S_{gt}.

Chamfer-L1 Distance (CD)
The Chamfer-L1 Distance calculates the distance between S_{gt} and S_p by taking the average of all nearest neighbour distances.

Table 4.2 shows how the metrics are distributed over the different categories. Based on the IoU score, our method performs best in the sofa category. However in all other metrics the category of planes is first.

	IoU ↑	HD↓	NC ↑	CD↓
Plane	0.789201	**0.037745**	**0.90522**	**0.004184**
Chair	0.78206	0.050914	0.882168	0.00702
Lamp	0.720649	0.042335	0.888334	0.005255
Sofa	**0.806795**	0.060259	0.882066	0.007685
Table	0.75371	0.052289	0.883729	0.006884

Table 4.2: Score Distribution over the five categories

4.2 Generalization to Larger Scenes

In the previous section we have shown that our method achieves high quality reconstruction results on synthetic objects. Since our pipeline is scalable to millions of points, we can process large-scale scenes. In this section we show that our method generalizes to larger synthetic scenes. Figure 4.3 depicts our reconstruction of the *Living Room* scene of the ICL-NUIM dataset [2]. The input point cloud was constructed by fusing every 10^{th} RGB-D frame of each track with the provided ground truth pose. In the end we downsampled the generated point cloud to 2 million points. The final reconstruction has some artifacts at high frequency features but overall accurately represents the living room.

4.3 Generalization to Real-World Data

The most challenging task for 3D shape reconstruction are large-scale real-world scenes in which the input point clouds are often noisy and contain outliers. In this section we investigate if our method can generalize to such scenes. In order to show qualitative results we construct

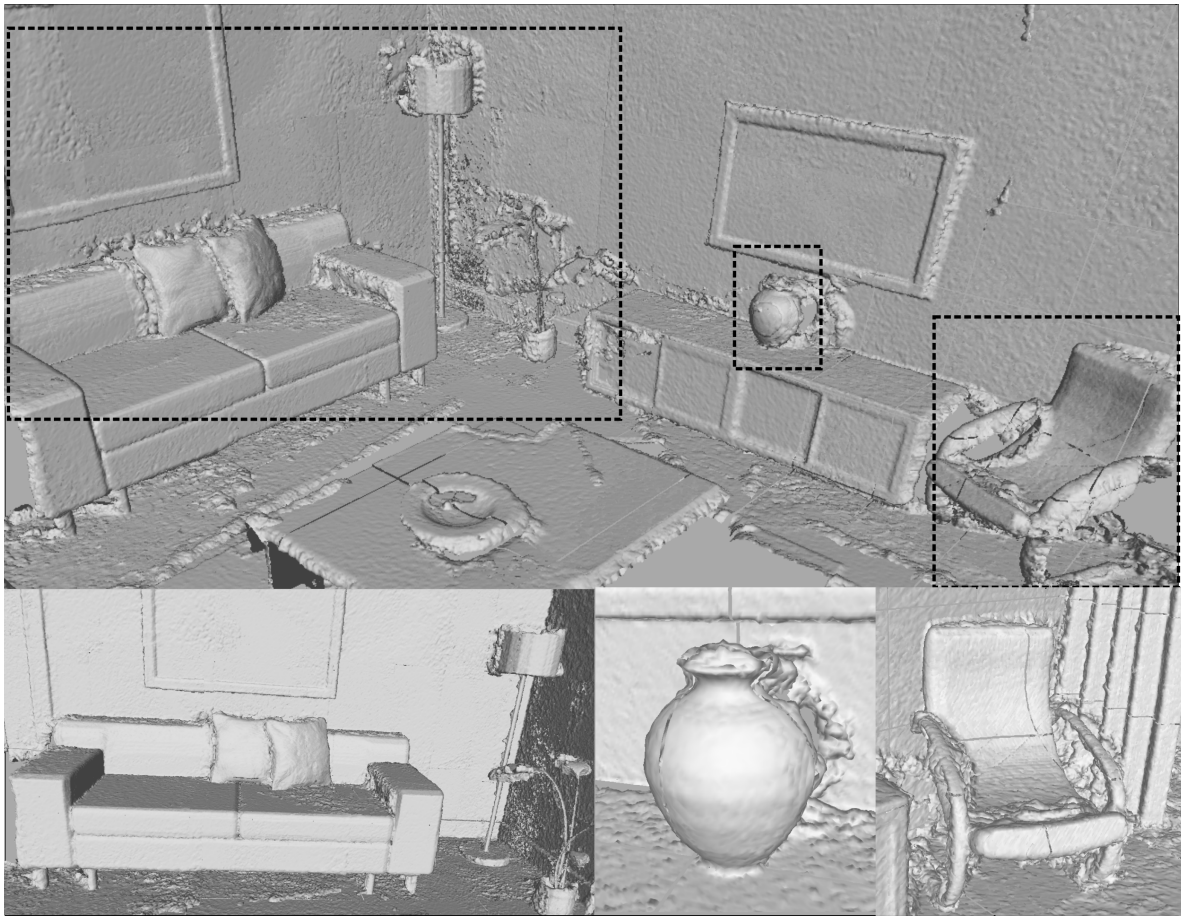


Figure 4.3: Reconstruction of the *Living Room* from ICL-NUIM [2] by our pipeline. Although we observe some noise we can safely assume that our method generalizes to larger scenes.

an input point cloud of the *Burghers of Calais* scene of the Scenes3D [3] dataset by again fusing every 10^{th} frame and downsample the generated point cloud to 2 million points. In Figure 4.4 we compare the qualitative result of our method with Lombardi *et al.* [4]. We observe that our method reconstructs a noisier mesh and contains more artifacts. However both methods preserve high frequency details like the facial expressions of the statues. This is a big advantage over voxel-based methods that have problems with fine features due to voxel discretization. We provide further reconstruction results in the Appendix 5.

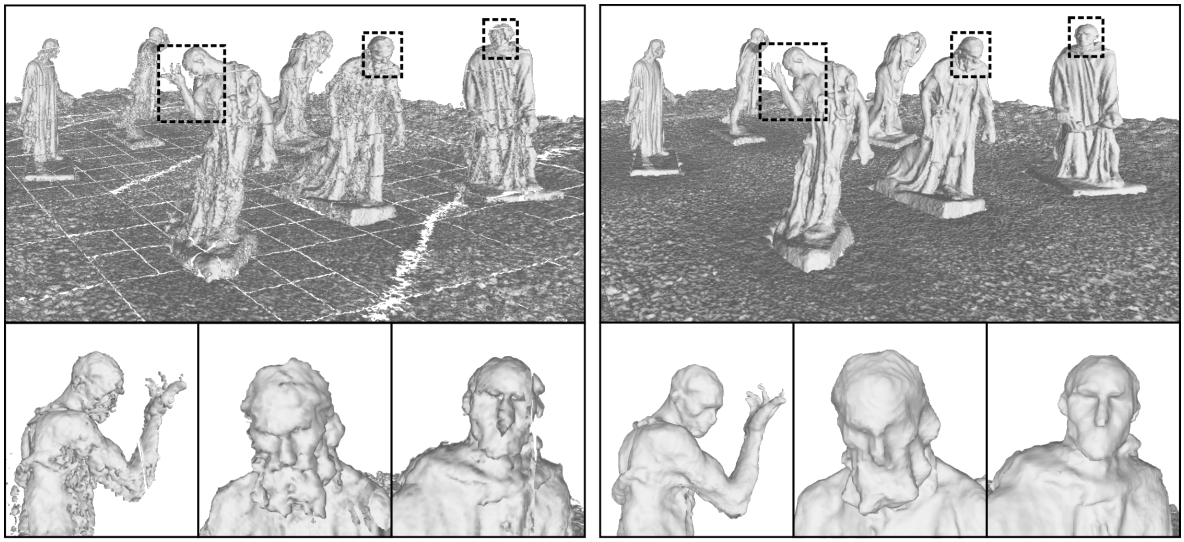
4.4 Time Complexity

During Inference the pipeline executes four major components:

(1) Data preparation (Octree and Query-generation, KNN for ReconRLA)

(2) ReconRLA Network

(3) Implicit Decoder



Ours Lombardi et al.

Figure 4.4: *Burghers of Calais* scene comparison. Although our method has some noise artifacts we achieve a comparable result to Lombardi *et al.* [4]

(4) Marching cubes + Merging the different Meshes

Parts (2),(3) and (4) can all run in a parallel fashion on the GPU. The bottleneck of our pipeline lies in the KNN algorithm. For our best performing model with only one aggregation step in the FA module, we still have to run $4 \cdot (1(\text{encoding layer}) + 2(\text{decoding layer})) = 12$ KNN's. A more detailed analysis can be found in the Appendix 5 where we conclude on the following runtime where M stands for the amount of query points:

$$O(M \log(M)) \tag{4.2}$$

Note that this runtime is based on a sequential KD-based KNN algorithm. Although there exist parallel CUDA-based implementations, we find that our network has a significant speedup if we use the sequential CPU version. In Figure 4.5 we visualize how point and query quantity

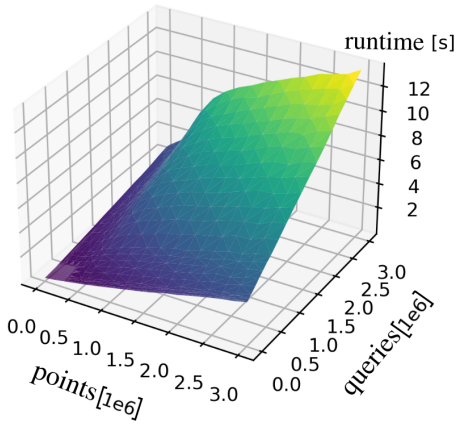


Figure 4.5: KNN runtime visualization

#Points	#Queries	Runtime [s]
20K	20K	0.03s
100K	100K	0.1s
20K	2M	1.6s
20K	1M	0.8s
1M	1M	3.5s
20K	10M	8s
5M	5M	23.7s
10M	10M	50.9s
20M	10M	110s

Table 4.3: KNN algorithm runtime

effect the KNN runtime. Additionally we report in Table 4.3 the exact runtimes of some KNN runs.

Due to chunking our method can process point clouds of any scale. Our pipeline first recursively splits the point cloud in an octree fashion until each leaf has less than a threshold of points. We then generate for each chunk query points. In some cases the amount of queries can get too large, in which case we apply the same idea of chunking to the query points. We report the inference time for different amount of queries and input points in Table 4.4. Note that the threshold for point cloud chunking was set to 30 thousand and 500 thousand

#Points	#Queries	#PointChunks	#QueryChunks	Runtime [s]
20K	4M	1	32	20s
50K	42M	8	240	149s
200K	60M	27	317	213s
2M	75M	229	544	306s
20M	81M	2134	2404	372s

Table 4.4: Inference time of the whole pipeline for different amount of input points and total queries (K=10^3, M=10^6). The Chunking threshold for input points was 30K and for queries 500K.

for the query chunking. Our pipeline can process a point cloud of 20 thousand points and 4 million query points in only 20 seconds. Even for almost 100 million query points our pipeline achieves an inference time of only a few minutes.

4.5 Architecture Parameter Influence

Table 4.5 shows the current setting of our model .In order to better understand our network we conduct an ablation study on the different architecture parameters.

(1) Variational Encoder Decoder As described in Section 3.6 we can transform our network to a variational encoder decoder. With this setting we train our model for 50 epochs with a constant learning rate of $1e^{-3}$.
(2) More neighbours Instead of only 5 neighbours each (layer before and skip connection) we increase them each to 8 neighbours.
(3) Variational with 2x aggregation We study if two aggregation modules in the decoding layers have the same effect in the variational setting as the non-variational one.
(4) 2x decoder aggregations In Section 3.5 we presented a new FA module that performs two aggregation and pooling steps. In this experiment we investigate the influence of this module on the reconstruction performance.

For each network architecture we present qualitative (Figure 4.6) and quantitative (Table 4.6) results. In Table 4.6 we present the IoU, HD, NC and CD metric for each network. We notice that our original method outperforms each ablated network in every metric. The variational approach performs worse than the classical methods. Additionally the FA module with two aggregations seems to worsen the performance, as both the variational and non-variational ablated network performed better with the original FA module.

Training	
optimizer	Adam [36]
learning rate	$1e^{-3}$
# epochs	5
batch size	1
Network	
latent size	64
activations (ReconRLA)	leaky ReLU
activations (Implicit decoder)	ReLU
downsampling factor (Encoding layers in ReconRLA)	4
upsampling factor (Decoding layers in ReconRLA)	4
# neighbours in KNN (Implicit decoder)	3
# aggregations in decoding layer	1
# neighbours in KNN (layer before)	5
# neighbours in KNN (skip layer)	5
# neighbours in KNN (2nd aggregation)*	5
variational	False

Table 4.5: Architecture Parameters. *Fields are only used if we are performing two aggregation steps in the FA module.

	IoU ↑	HD↓	NC ↑	CD↓
(1) Variational	0.631284	0.065756	0.81359	0.008679
(2) More Neighbours	0.753255	0.069495	0.886738	0.00644
(3) Variational + 2x aggregation	0.631402	0.066244	0.813644	0.008689
(4) 2x aggregation	0.738497	0.057598	0.86912	0.006587
Original	**0.77000**	**0.04875**	**0.88922**	**0.00620**

Table 4.6: Score Comparison of our Original model and the four ablated networks (1),(2),(3),(4) on the ShapeNet [1] dataset

With Figure 4.6 we report qualitative results on a sample of each category of the ShapeNet [1] dataset. We observe that network architectures with two aggregation steps, hence (3) and (4), have more noise and artifacts. Especially the airplane reconstruction contains many unwanted mesh faces at edges and corners. We further note that the variational approach with only one aggregation returns qualitative results that are on par with our original method. However we notice that some parts are not reconstructed (e.g. connection at the top of the lamp). Network (3) also shows some artifacts and seems to worsen the reconstruction performance. This is counterintuitive since we expect that allowing the network to have more information about its surrounding neighbourhood would improve the reconstruction in tradeoff with higher computational effort. We believe that by taking more neighbours our network adds new feature vectors that are further and further apart from our query location, our network could aggregate encodings that no longer describe the local neighbourhood accurately but rather a different neighbourhood, which adds noise to our feature encoding. However we still need to investigate this hypothesis further.

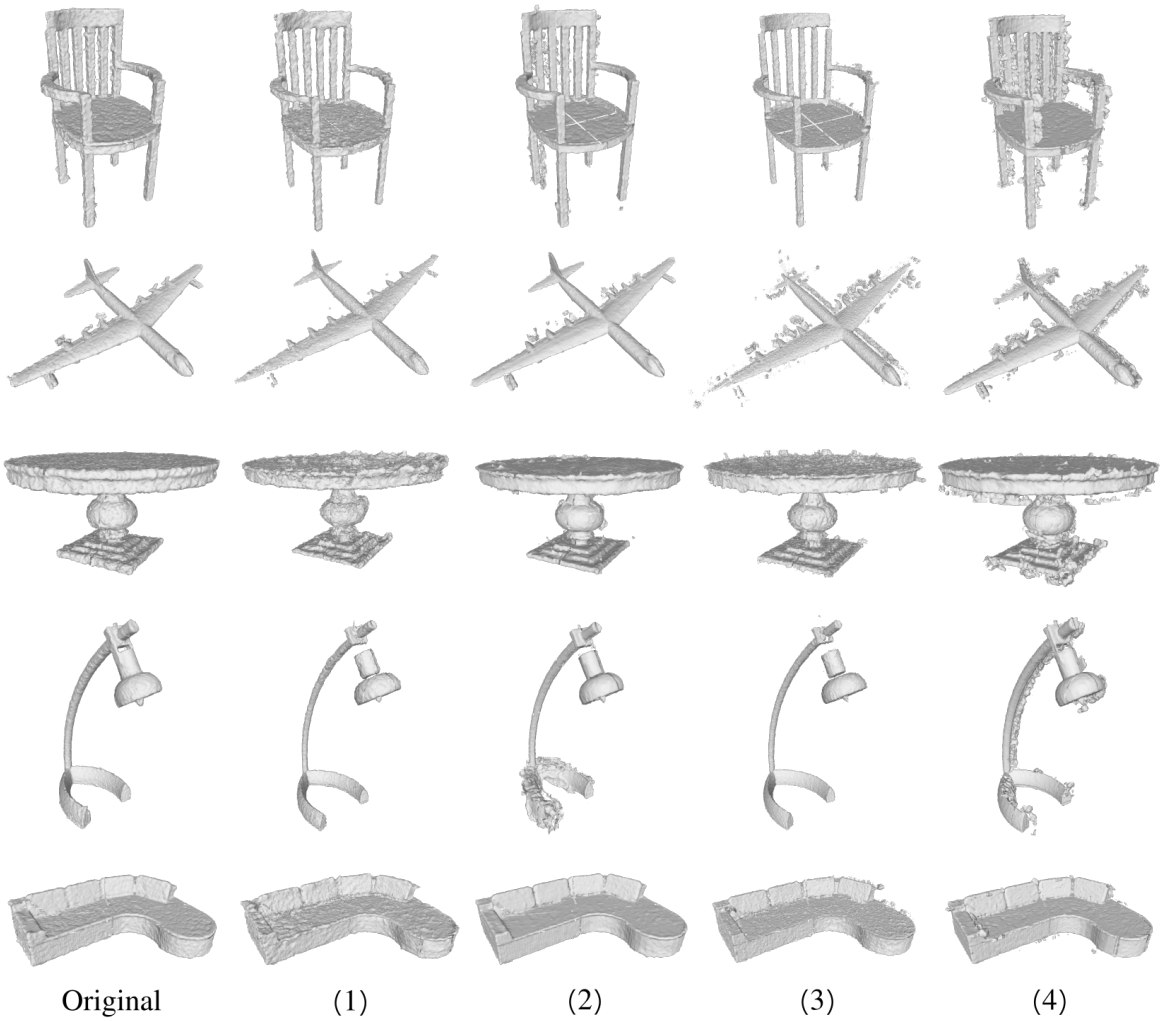


Figure 4.6: Qualitative comparison between our original approach and (1),(2),(3),(4)

5 Conclusion

We proposed the first ever learning-based grid-free scalable pipeline for 3D shape reconstruction from point clouds. Instead of using complicated implementations to speed up grid-based methods, our approach operates directly on the input point cloud. With ReconRLA we extract hierarchical point features which are used to generate a latent code for our query points for which we assign a corresponding SDF value in our decoding step. Since we use our deep neural net as an implicit representation we can potentially achieve infinite resolution reconstructions by querying millions of query points. Additionally our method is very simple and can therefore easily be improved. In Experiments we showed generalization of our method to real-world large-scale scenes and evaluated our performance on synthetic data. Future work will try to replace the K nearest neighbour algorithm by a parallelized radius search, which should enable us to achieve real-time performance on medium sized point clouds. We also plan to improve our feature aggregation module in the decoding layer to increase the prediction accuracy.

Bibliography

[1] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, *et al.*, "Shapenet: An information-rich 3d model repository," *arXiv preprint arXiv:1512.03012*, 2015.

[2] A. Handa, T. Whelan, J. McDonald, and A. J. Davison, "A benchmark for rgb-d visual odometry, 3d reconstruction and slam," in *2014 IEEE international conference on Robotics and automation (ICRA)*. IEEE, 2014, pp. 1524–1531.

[3] Q.-Y. Zhou and V. Koltun, "Dense scene reconstruction with points of interest," *ACM Transactions on Graphics (ToG)*, vol. 32, no. 4, pp. 1–8, 2013.

[4] S. Lombardi, M. R. Oswald, and M. Pollefeys, "Scalable point cloud-based reconstruction with local implicit functions," in *2020 International Conference on 3D Vision (3DV)*. IEEE, 2020, pp. 997–1007.

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

[6] M. Michalkiewicz, J. K. Pontes, D. Jack, M. Baktashmotlagh, and A. Eriksson, "Deep level sets: Implicit surface representations for 3d shape inference," *arXiv preprint arXiv:1901.06802*, 2019.

[7] Z. Chen and H. Zhang, "Learning implicit fields for generative shape modeling," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 5939–5948.

[8] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger, "Occupancy networks: Learning 3d reconstruction in function space," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4460–4470.

[9] R. Chabra, J. E. Lenssen, E. Ilg, T. Schmidt, J. Straub, S. Lovegrove, and R. Newcombe, "Deep local shapes: Learning local sdf priors for detailed 3d reconstruction," in *European Conference on Computer Vision*. Springer, 2020, pp. 608–625.

[10] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 652–660.

[11] Q. Hu, B. Yang, L. Xie, S. Rosa, Y. Guo, Z. Wang, N. Trigoni, and A. Markham, "Randla-net: Efficient semantic segmentation of large-scale point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 108–11 117.

[12] J. Chibane, T. Alldieck, and G. Pons-Moll, "Implicit functions in feature space for 3d shape reconstruction and completion," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 6970–6981.

[13] Z. Mi, Y. Luo, and W. Tao, "Ssrnet: scalable 3d surface reconstruction network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 970–979.

[14] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 1996, pp. 303–312.

[15] M. Rothermel, N. Haala, and D. Fritsch, "A median-based depthmap fusion strategy for the generation of oriented points." *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, vol. 3, no. 3, 2016.

[16] N. Marniok, O. Johannsen, and B. Goldluecke, "An efficient octree design for local variational range image fusion," in *German Conference on Pattern Recognition*. Springer, 2017, pp. 401–412.

[17] K. Kolev, M. Klodt, T. Brox, and D. Cremers, "Continuous global optimization in multiview 3d reconstruction," *International Journal of Computer Vision*, vol. 84, no. 1, pp. 80–96, 2009.

[18] M. Kazhdan and H. Hoppe, "Screened poisson surface reconstruction," *ACM Transactions on Graphics (ToG)*, vol. 32, no. 3, pp. 1–13, 2013.

[19] V. Estellers, M. Scott, K. Tew, and S. Soatto, "Robust poisson surface reconstruction," in *International Conference on Scale Space and Variational Methods in Computer Vision*. Springer, 2015, pp. 525–537.

[20] L. Ladicky, O. Saurer, S. Jeong, F. Maninchedda, and M. Pollefeys, "From point clouds to mesh using regression," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 3893–3902.

[21] Y.-P. Cao, Z.-N. Liu, Z.-F. Kuang, L. Kobbelt, and S.-M. Hu, "Learning to reconstruct high-quality 3d shapes with cascaded fully convolutional networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 616–633.

[22] I. Cherabier, J. L. Schonberger, M. R. Oswald, M. Pollefeys, and A. Geiger, "Learning priors for semantic 3d reconstruction," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 314–330.

[23] A. Dai, D. Ritchie, M. Bokeloh, S. Reed, J. Sturm, and M. Nießner, "Scancomplete: Large-scale scene completion and semantic segmentation for 3d scans," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4578–4587.

[24] M. Tatarchenko, A. Dosovitskiy, and T. Brox, "Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2088–2096.

[25] W. Dong, Q. Wang, X. Wang, and H. Zha, "Psdf fusion: Probabilistic signed distance function for on-the-fly 3d data fusion and scene reconstruction," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 701–717.

[26] V. Sitzmann, J. Martel, A. Bergman, D. Lindell, and G. Wetzstein, "Implicit neural representations with periodic activation functions," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[27] A. Gropp, L. Yariv, N. Haim, M. Atzmon, and Y. Lipman, "Implicit geometric regularization for learning shapes," *arXiv preprint arXiv:2002.10099*, 2020.

[28] C. Jiang, A. Sud, A. Makadia, J. Huang, M. Nießner, T. Funkhouser, *et al.*, "Local implicit grid representations for 3d scenes," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 6001–6010.

[29] J. Li, B. M. Chen, and G. H. Lee, "So-net: Self-organizing network for point cloud analysis," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 9397–9406.

[30] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph cnn for learning on point clouds," *Acm Transactions On Graphics (tog)*, vol. 38, no. 5, pp. 1–12, 2019.

[31] H. Su, V. Jampani, D. Sun, S. Maji, E. Kalogerakis, M.-H. Yang, and J. Kautz, "Splatnet: Sparse lattice networks for point cloud processing," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2530–2539.

[32] S. Xie, S. Liu, Z. Chen, and Z. Tu, "Attentional shapecontextnet for point cloud recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4606–4615.

[33] L. Landrieu and M. Simonovsky, "Large-scale point cloud semantic segmentation with superpoint graphs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4558–4567.

[34] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *ACM siggraph computer graphics*, vol. 21, no. 4, pp. 163–169, 1987.

[35] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[36] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[37] W. Falcon and .al, "Pytorch lightning," *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, vol. 3, 2019.

[38] E. Riba, D. Mishkin, D. Ponsa, E. Rublee, and G. Bradski, "Kornia: an open source differentiable computer vision library for pytorch," 2019.

Appendix

1 KNN Runime Analysis for ReconRLA

For all encoding layers E_i with $i \in \{0, .., k\}$ (left to right, k=#Encoding layers) of ReconRLA we require a KNN between $\mathbf{P}^i$ and $\mathbf{P}^i$ where $\mathbf{P}^i$ denotes the i-times downsampled point cloud. Note that after each KNN our encoding layer downsamples the points. This is why $\mathbf{P}^k$ exists but we will not perform a KNN on it. For k encoding layers we therefore require the following computations :

$$\sum_{i=0}^{k-1} \text{KNN}(\mathbf{P}^i, \mathbf{P}^i) \tag{1}$$

We denote with KNN($\mathbf{A}$,$\mathbf{B}$) the KNN with $\mathbf{A}$ as the queries for which we want to find the nearest neighbours and $\mathbf{B}$ as the base points.

In all decoding layers D_i with $i \in \{0, .., k\}$ (left to right, k=#Decoding layers) of ReconRLA we have two KNN's. We again denote with $\mathbf{Q}^i$ the i-times downsampled query point cloud. For a specific layer D_i we need to find for $|\mathbf{Q}^{(k-1)-i}|$ queries an encoding. Example: For four decoding layers (k=4) and the fourth decoding layer D_3 (since i starts with 0) we need to find for each of the query points an encoding. This corresponds to the 0-times downsampled query point cloud $\mathbf{Q}^0 = \mathbf{Q}^{(4-1)-3)}$. In the feature aggregation module we perform two KNN's. The first is between our query points and the previous layer. While for the first decoding layer D_0 the layer before is the k-times downsampled input point cloud $\mathbf{P}^k$, for all other decoding layers D_i the previous layer is $\mathbf{Q}^{(k-1)-i+1} = \mathbf{Q}^{k-i}$. The second KNN is between $\mathbf{Q}^{(k-1)-i}$ and $\mathbf{P}_{(k-1)-i}$ since the encodings come from the skip connection. We therefore have the following computations for the decoding layers:

$$\text{KNN}(\mathbf{Q}^{k-1}, \mathbf{P}^k) + \text{KNN}(\mathbf{Q}^{k-1}, \mathbf{P}^{k-1}) + \sum_{i=1}^{k-1} \text{KNN}(\mathbf{Q}^{(k-1)-i}, \mathbf{Q}^{k-i}) + \text{KNN}(\mathbf{Q}^{(k-1)-i}, \mathbf{P}^{(k-1)-i}) \tag{2}$$

If we include 2 aggregation modules for each decoding layer D_i, we have an additional KNN between the query points of that layer with itself.

$$\sum_{i=0}^{k-1} \text{KNN}(\mathbf{Q}^{(k-1)-i}, \mathbf{Q}^{(k-1)-i}) \tag{3}$$

Without the additional aggregation module we have a total of $3 \cdot$ #encoding layers KNN's. In practise these KNN's accumulate. However in theory we have the following runtime (without the second aggregation):

$$O(KNN) = O(\sum_{i=0}^{k-1} \text{KNN}(\mathbf{P}^i, \mathbf{P}^i) \tag{4}$$

$$+ \text{KNN}(\mathbf{Q}^{k-1}, \mathbf{P}^k) + \text{KNN}(\mathbf{Q}^{k-1}, \mathbf{P}^{k-1}) \tag{5}$$

$$+ \sum_{i=1}^{k-1} \text{KNN}(\mathbf{Q}^{(k-1)-i}, \mathbf{Q}^{k-i}) + \text{KNN}(\mathbf{Q}^{(k-1)-i}, \mathbf{P}^{(k-1)-i}) \tag{6}$$

Which results in

$$O\left(\max(\text{KNN}(\mathbf{Q}, \mathbf{Q}), \text{KNN}(\mathbf{P}, \mathbf{P}))\right) \tag{7}$$

To this end we are using a K-D tree based sequential KNN implementation. If we have a KNN($\mathbf{A}$,$\mathbf{B}$) then we first need $O(|\mathbf{B}| \log |\mathbf{B}|)$ to construct the K-D tree and then need $O(|\mathbf{A}| \log |\mathbf{B}|)$ to find the nearest neighbour for each query point in $\mathbf{A}$. Since we have more queries than input points we conclude on the following runtime (With $|\mathbf{Q}| = M$):

$$O\left(M \log M\right) \tag{8}$$

2 KNN Visualization

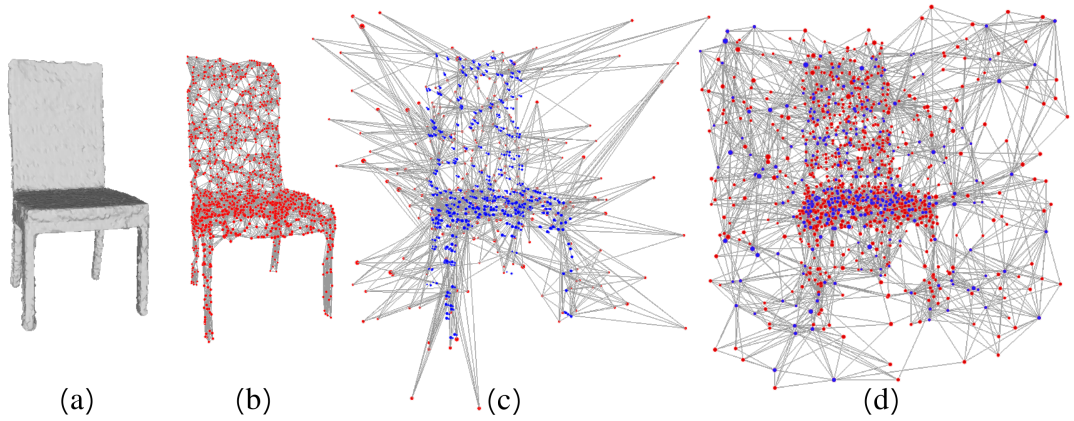


Figure 1: KNN Visualization

With Figure 1 we visualize the KNN computations. The edges symbolize the connection to the nearest neighbours. While (a) depicts our reconstruction result, we show in (b) the KNN result between $\mathbf{P}_3$ and $\mathbf{P}_3$. Note that (b) visualizes a typical KNN computation in the encoding layer. Visualization (c) shows the KNN result that is calculated in the first decoding layer, where we perform a KNN between $\mathbf{P}_k$ and $\mathbf{Q}_0$. The KNN for the SC-AM (Skip connections) will have a similar looking result. Lastly (d) visualizes the KNN result between $\mathbf{Q}_3$ and $\mathbf{Q}_2$ namely two query point clouds. This is a typical look for the KNN in the LB-AM module since the current layer D_i and the previous layer D_{i-1} both contain (expect for the case $i = 0$) query point clouds. In Figure 2 we show how the KNN looks in the SC-AM module of the last decoding layer where we calculate the KNN between all queries and all

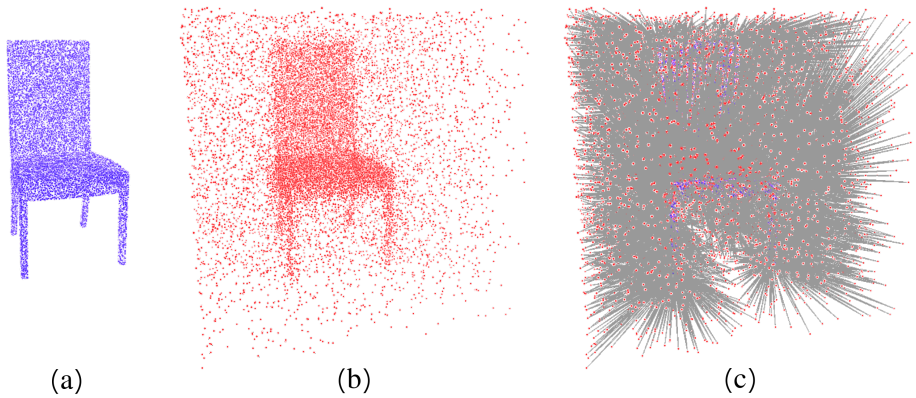


Figure 2: KNN Visualization between all query and input points

input points. In (a) we show the input points and (b) visualized the queries. With (c) we show the KNN result.

3 Additional Qualitative Results

In this section we show some additional qualitative results.
Figure 3 shows the *Lounge* scene from the Scenes3d [3] dataset. The input point cloud is constructed from real-world data, that is noisy and contains outliers. Although we observe some unwanted artifacts (*e.g.* bottom part of the second chair from the left), we overall return an accurate reconstruction. Our method preserves fine detail (*e.g.*book on the table) and doesn't fuse components together (*e.g.* plant pot and wall are separated).

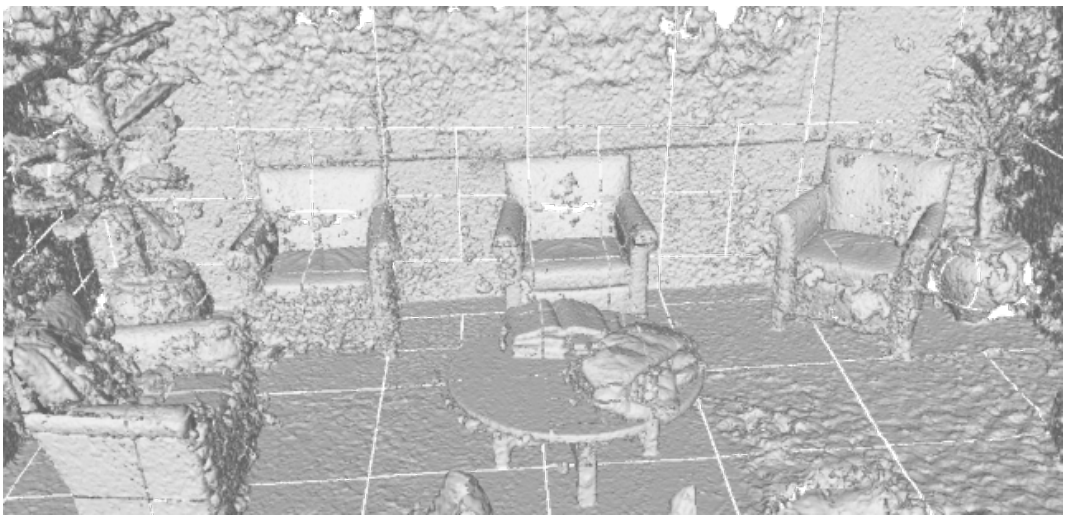


Figure 3: Our lounge scene from the Scenes3d dataset [3]

In Figure 4 we compare our method with Lombardi *et al.* [4]. We observe that the surface

of Lombardi *et al.* is a bit smoother and contains a little less noise. However both approaches return an accurate reconstruction that preserves fine details.

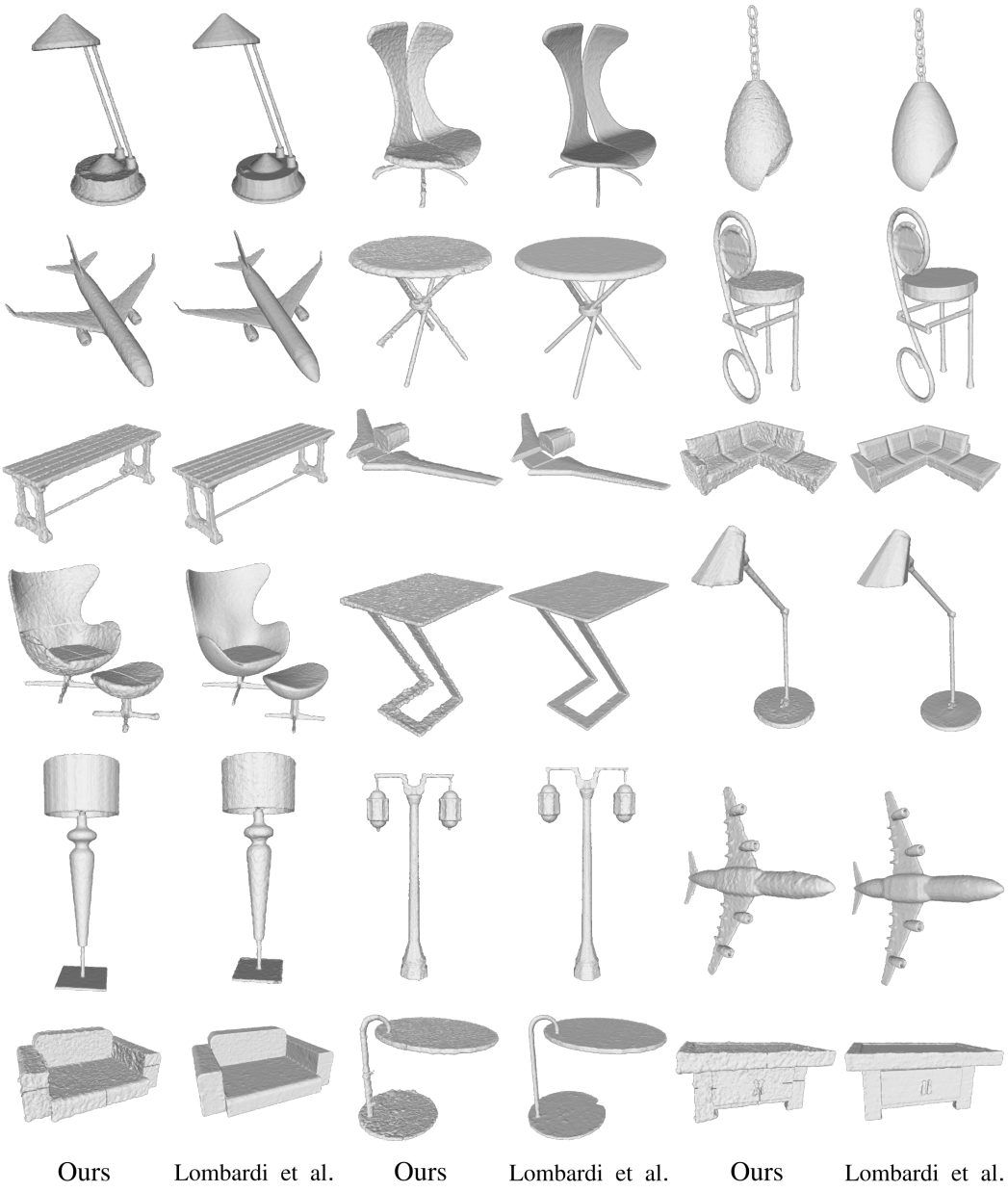


Figure 4: Shapenet sample comparison with Lombardi *et al.* [4]

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Eigenständigkeitserklärung

Die unterzeichnete Eigenständigkeitserklärung ist Bestandteil jeder während des Studiums verfassten Semester-, Bachelor- und Master-Arbeit oder anderen Abschlussarbeit (auch der jeweils elektronischen Version).

Die Dozentinnen und Dozenten können auch für andere bei ihnen verfasste schriftliche Arbeiten eine Eigenständigkeitserklärung verlangen.

Ich bestätige, die vorliegende Arbeit selbständig und in eigenen Worten verfasst zu haben. Davon ausgenommen sind sprachliche und inhaltliche Korrekturvorschläge durch die Betreuer und Betreuerinnen der Arbeit.

Titel der Arbeit (in Druckschrift):

ReconRLA: Point-based large-scale SurfaceReconstruction from Point Clouds

Verfasst von (in Druckschrift):

Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich.

Name(n):	**Vorname(n):**
Muntwyler	Nicolas

Ich bestätige mit meiner Unterschrift:

- Ich habe keine im Merkblatt „Zitier-Knigge“ beschriebene Form des Plagiats begangen.
- Ich habe alle Methoden, Daten und Arbeitsabläufe wahrheitsgetreu dokumentiert.
- Ich habe keine Daten manipuliert.
- Ich habe alle Personen erwähnt, welche die Arbeit wesentlich unterstützt haben.

Ich nehme zur Kenntnis, dass die Arbeit mit elektronischen Hilfsmitteln auf Plagiate überprüft werden kann.

Ort, Datum	**Unterschrift(en)**
30.3.2021	N. Muntwyler

Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich. Durch die Unterschriften bürgen sie gemeinsam für den gesamten Inhalt dieser schriftlichen Arbeit.