Neural Implicit Shape Representations for 3D Reconstruction

Bachelor Thesis
7th February 2022 - 21th July 2022

by Nicolas Samuel Blumer, 19-749-563

Supervisors:
Prof. Dr. Renato Pajarola
Lizeth J. Fuentes Perez

Visualization and MultiMedia Lab
Department of Informatics
University of Zürich

University of Zurich[UZH]

VISUALIZATIONANDMULTIMEDIALAB

Cover Image:
Visualized reconstruction of a chair by IM-NET

Zusammenfassung

Die 3D-Rekonstruktion ist ein Problem, welches die Informatik seit Jahrzehnten beschäftigt. Die Arbeit mit 3D-Daten ist aufgrund der Komplexität der Berechnungen und des Mangels an Daten im Vergleich zu anderen Bereichen bekanntermassen schwierig. Mit dem Aufkommen von Machine Learning und insbesondere Deep Learning wurden neue Ideen entwickelt, um die 3D-Rekonstruktion anzugehen. Eine Richtung, die stark an Bedeutung gewonnen hat, ist die ”implizite Repräsentation”. Die Idee impliziter Methoden besteht darin, ein Signal, wie z. B. 3D-Daten, als implizite Funktion darzustellen, die durch ein neuronales Netz approximiert wird. In dieser Arbeit wird dieser Bereich anhand von drei ausgewählten Methoden namens IM-NET, NDF und SIREN untersucht. Diese Methoden werden theoretisch analysiert und verglichen und es werden Experimente durchgeführt, um einen Einblick in implizite Repräsentationen für die 3D-Rekonstruktion zu erhalten. Darüber hinaus wird ein Werkzeug zur Analyse der Vektordarstellungen von Formen entwickelt, die einige Methoden erzeugen.

Abstract

3D reconstruction is a problem that has occupied the minds of computer scientists for decades, and working with 3D data is notoriously difficult because of the computational complexity and lack of data compared to other areas. With the rise of machine learning and especially deep learning, new ideas have been developed to tackle 3D reconstruction. One direction that has recently gained traction is called ”implicit representations”. The idea of implicit methods is to represent a signal, like 3D data, as an implicit function that is being approximated by a neuronal net. This thesis explores three selected papers about neural implicit representations, IM-NET, NDF, and SIREN. These methods are analyzed and compared theoretically, and experiments are conducted to gain insight into implicit representations for 3D reconstruction. In addition, a tool is created for analyzing the vector representations of shapes some methods generate.

Contents

List of Figures

1 Introduction

Neural implicit representations offer a novel form to represent data. As opposed to more classical ways to describe data where usually a discrete set of values describing something are present, in this representation, the data is being characterized by a continuous function [Par+19]. Think, for example, about an (RGB-)image. It is usually represented as a fixed number of fields on a grid, each of which stores the red, green, and blue values at any given position. A neural implicit representation of the same image could be a continuous learned function where an RGB value of the image at this position is returned after inputting a set of coordinates. Because the function is continuous, the value of an arbitrary point in the domain can be returned, which makes the image have any desired resolution. For example, this application of neural implicit representations has been explored in [CLW21]. What has just been described is but one use case and only our imagination limits the number of signals, that could be represented in such a fashion.

Equally many applications exist that could benefit from these new representations, but research has mainly been focused on representing images [CLW21; Sit+20; Dup+21] and 3D structures [Par+19; Sit+20; Sit+20] implicitly. The goals of this research include image-compression [Dup+21], image-generation [Cha+21], scene understanding [Zha+21a], shape generation [CZ19], shape reconstruction [CZ19; CMP20] and many more. Reconstructing 3D shapes and scenes, which is the focus of this thesis, is a problem with many applications in robotics, computer vision, and computer graphics such as navigation, object recognition, and 3D modeling, which has been studied for a long time [HLB19].

The idea behind implicit representations in the context of 3D shape data is that there is a function, which can be ”asked” how a shape looks at a certain position. Depending on the method, the system will return a distance to the closest point on the surface [Par+19; CMP20] or tell if this point is part of the shape or not [CZ19; Mes+19]. While there might be other functions with the capability to represent 3D spatial data, these are the types of functions researchers have primarily been focused on.

The main advantage of implicit representations over traditional representations has already been mentioned: an implicit representation has an infinite resolution in theory. Since the function is the same, irrespective of what resolution a user might be interested in, it has a constant, predefined memory. This can potentially save space. There are also clear disadvantages of the methods used. The main one is the limitations of the function approximators, the neural networks. As is shown in chapter 5, the functions are not always learned perfectly. For example, the input might not be optimal by being noisy. Another problem is the time complexity of the methods, making the learning very slow compared to generating conventional representations like meshes.

This paper has several goals, which are stated in a more compact fashion below. First, as a side goal, an overview of the topic of ”implicit representations for 3D reconstruction” is needed, which will be in chapter 3. Then, three methods have been selected which are to be theoretically analyzed and compared. The selected methods are [Sit+20] by Sitzman *et al.*, [CZ19] by Chen *et al.* and [CMP20] by Chibane *et al.* Afterwards, experiments are conducted to compare the performance of the selected methods. The next goal is to experiment with one of the selected methods to see if the performance can be improved. The chosen method for this part is ”SIREN” [Sit+20]. Finally, a visualization is to be created to visualize the outputs of the encoders of the methods that have one. The embeddings need to be dimensionality-reduced to be able to visualize them; the visualization should make it possible to compare the embeddings of different methods.

Part of this thesis are the following tasks:

- **RQ1**: What are the similarities and differences between the selected methods?
- **RQ2**: How do the selected methods compare in a practical setting?
- **RQ3**: How did the SIREN extensions change the performance?
- **RQ4**: How do the shape embeddings of the selected methods compare?

1.1 Outline

Based on the aforementioned aspects and the given requirements, the structure of this document is built as follows: chapter 2 contains background knowledge some readers might not be familiar with to make reading the rest of the thesis easier. Chapter 3 is dedicated to the related work concerning the reconstruction of 3D shapes with a focus on implicit representations. In chapter 4 the selected methods will be explained and compared with respect to topics like architectural or loss function-related differences. The experiments can be found in chapter 5. This includes the practical comparison of the methods, as well as the experiments done to try to improve one of the selected methods. Section 5.4 describes how the shape encodings have been visualized. The results of the experiments and the theoretical analysis of the methods are located in chapter 6. Additional thoughts regarding the topics of the thesis as well as limitations and future work is can be found in chapter 7. Finally, chapter 8 concludes this thesis.

2 Background

In this chapter, we provide some background knowledge to facilitate the understanding of the following chapters.

2.1 Distance Fields

A distance field is a space of scalar values representing the distance of a point to a shape or surface. The distance between a point and a shape is defined as the distance of the point to the closest point on the surface. Distance functions can be signed or unsigned, where the sign indicates whether a point is inside or outside the shape [Hua+01]. Some, like [Hua+01], define a negative distance as being outside the shape while a positive distance means the point is inside the shape. We will be using the definition of [Par+19] which assigns negative distances to points inside a shape.

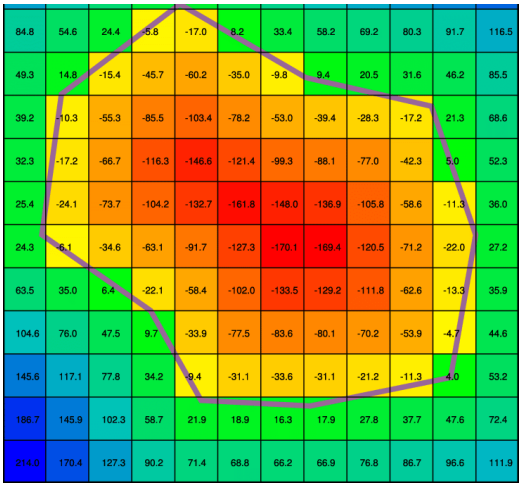


Figure 2.1: Discretized signed distance field of a 2D polygonal obstacle described by Zucker *et al.* [Zuc+10].

We define an unsigned distance function as a function that returns the distance to the closest point of a surface of the input point p. Such a function could be defined as follows:

$$dist_{\Sigma}(p) = \inf_{x \in \Sigma} ||x - p|| \tag{2.1}$$

Here, Σ is a set of points on the surface of the shape [JBS06].

The signed distance function (SDF) is a function representing a distance field. Similar to the unsigned distance function, the input is a point and its distance is the output, but like in distance fields, the output value has a sign. A crucial property, which is used by one of the methods discussed in chapter 4, is that the derivative of the SDF equals one almost anywhere. This does not hold for points that do not have a single closest point [JBS06]. SDFs are a popular way to implicitly describe a shape, which can be seen in chapter 3 and chapter 4.

2.2 Neural Net Architectures

2.2.1 Auto-encoder

An auto-encoder (AE), is a neural network architecture with use cases in many areas such as language [Lio+14], finance [GKX21] or computer vision [Vou+18]. It was proposed by LeCun [LeC87; Zha+18]. AEs consist of two parts: an encoder and a decoder. The encoder transforms the input into an abstract vector representation which the decoder either converts back into the original form or returns as something different but related. It has many use cases like dimensionality reduction and generation [GBC16]. There are many variants in what the encoder and decoder can be. Either part can consist of fully connected or convolutional layers, but does not have to be limited to neural networks [Zha+18]. Often, the encoder compresses the input down to a smaller size and the decoder tries to recreate the original, similar to what can be seen in figure 2.2 [Par+19].

The variational auto-encoder is a specialized version of an auto-encoder which we will from now call VAE. As opposed to a standard AE, a VAE has the ability to generate different samples from a vector representing the encoder created. This means the decoder output is probabilistic. Often constraints upon the encoder are imposed to create a vector that fits a distribution, commonly a Gaussian distribution. VAEs can generate different samples by changing the latent encoding a little [Zha+18].

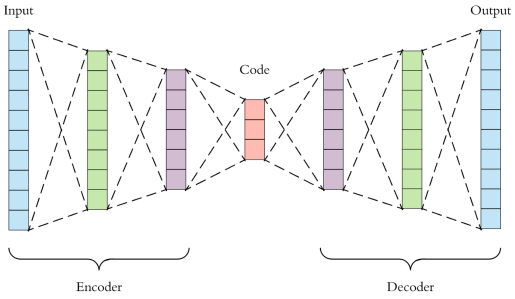


Figure 2.2: Visual example of how an auto-encoder can look like by Dertat. The ”code” is the latent/vector representation the encoder creates [Der17].

2.2.2 Auto-decoder

An auto-decoder is comparable to an AE but without an encoder. Instead, every input sample gets assigned a random initial latent representation. During training, the representations are being improved by back-propagating through the input representation, as seen in figure 2.3. By fitting this representation to a distribution (e.g., Gaussian), it is possible to use the decoder similar to how a VAE is used to sample new shapes [Par+19]. The disadvantage of this approach is that there is no way to reconstruct a shape the model did not see during training. In comparison, an AE has the ability to also use its encoder for unseen data.

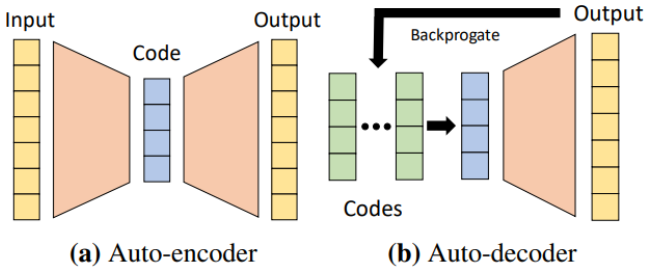


Figure 2.3: Comparing an auto-encoder with an auto-decoder by Park *et al.* [Par+19].

2.3 Metrics

Metrics are needed to compare the quality of 3D reconstructions. Some metrics that are used in other areas that can be applied for 3D reconstructions are the mean squared error and the intersection over union error. [CZ19] chose these metrics to evaluate their models. They also proposed the light field descriptor error, which uses 2D renderings from several angles to compare the quality. There are many more metrics that can be used, but one of the most important ones is the Chamfer Distance because of its widespread use.

2.3.1 Chamfer Distance

Chamfer Distance is a metric for comparing point sets like point clouds. To calculate the chamfer distance, for each point in both sets, the distances to the nearest points in the other set are added together. For our purpose, the two sets are the ground truth and the reconstructed shape in point cloud format. The distance is defined as follows:

$$d_{CD}(S_1, S_2) = \frac{1}{|S_1|} \sum_{x \in S_1} \sum_{y \in S_2} min(||x - y||_2) + \frac{1}{|S_2|} \sum_{x \in S_2} \sum_{y \in S_1} min(||x - y||_2) \tag{2.2}$$

It is one of the most common metrics in the realm of 3D reconstruction and has been used in [CMP20; AL21; CZ19; BKG22; Par+19], which is the reason why it is the metric of choice for this thesis too. Other variations of this metric, such as the Density-aware Chamfer Distance, have been proposed. Other norms, like the L_1 norm, can be used instead of the L_2 in equation 2.2 [Ton+21]. We will use the formula form equation 2.3 which can be found in the code from [BKG22] and seems to deliver the closest values to what some of the core papers of this thesis use.

$$d_{CD}(S_1, S_2) = 0.5 * (\frac{1}{|S_1|} \sum_{x \in S_1} \underset{y \in S_2}{min}(||x - y||_2^2) + \frac{1}{|S_2|} \sum_{x \in S_2} \underset{y \in S_1}{min}(||x - y||_2^2)) \tag{2.3}$$

3 Related Work

This chapter highlights past research in 3D reconstruction and related areas. Section 3.1 is concerned with alternative methods that do not use implicit representations in the context of working with 3D spacial data. In section 3.2 different implicit representations and the methods using them are being discussed. How deep learning methods can deal with having to learn multiple shapes is the topic of section 3.3.

3.1 Non-Implicit Representations

In this section, other representations for 3D spatial data that compete with the implicit representations that are the focus of this thesis will be looked at. Representations used in methods to reconstruct or generate shapes with a learning element are mainly being discussed as they are possible rivals to implicit representations.

One class of methods tries reconstructing shapes by parametrizing (one or many) base shapes. A popular base shape in this area is the sphere, which is well suited to fit genus-0 models [SPR+07]. This is done by starting from some initial parameters and then iteratively improving them [Ale99]. Information about the normals and angles can be preserved with sphere parametrization methods [Gu+04]. The core challenge of these methods is that the spherical triangles do not overlap [Got+03]. Several of these spheres can be merged to create a smooth transformation between them [Ale99]. Also, ellipsoids can be deformed to progressively generate more details of a shape. [Wan+18] uses a neural network consisting of 3D convolutions and pooling layers to achieve that. Several methods need the shape to have certain qualities like being genus-0 which limits the power of the approach.

Another approach is to have a database of shapes, which can be linearly combined to form a new one with the help of eigenvalues. Such an approach can, however, only interpolate between the shapes it has in its database [BV99].

A natural representation for 3D data is using voxels, which is the 3D version of what pixels are in the 2D domain. Girdhar *et al.* use a 3D CNN to get a vectorized representation of a shape and then use deconvolutions to get the input back. In addition, an image of the shape is the input to a separate 2D CNN and the goal of the method is to make the output of the two encoders similar. During deployment, only photos are required to get 3D models of shapes [Gir+16]. There are methods to directly learn to output a voxelized shape from

an image [Yi+17]. Häne *et al.* proposed a ”hierarchical surface prediction” with which progressively higher resolution voxel-grids can be generated [HTM17]. Diffusion models generating shapes in voxel form have been proposed [ZDW21]. Wu *et al.* uses a Generative Adversarial Network to generate voxel-shapes [Wu+16]. Using voxel as an output forces a certain size upon the system. All generated or reconstructed shapes will have this predefined size.

Another popular way of representing 3D data is using point clouds. They are also extensively used for 3D reconstructions. Either to generate the input data needed to train methods using implicit representations [Che22] or to learn from point clouds directly. Achlioptas *et al.* get their vector representation of a shape and decode it back to a fixed number of coordinates. This approach requires the shapes to consist of a predefined amount of points which limits the versatility of the method [Ach+18]. There also exist methods that learn to produce a point cloud from only image input [FSG17]. These methods have similar problems to the ones using voxels because the output sizes of the neural networks are fixed.

More abstract ways to represent the data have also been designed. Li *et al.* developed a neural network that captures the hierarchical structure of most everyday items. A graph is used to represent the hierarchy. Their decoder can then recombine the pieces and return a shape [Li+17].

This section represents a non-exhaustive list of methods for representing 3D spatial data, and each of the shown methods has its use-cases where they are appropriate. All these representations are limited in the things they can represent. The resolution of their output is usually fixed. They have some number of parameters to change a base shape, some number of voxels or points. This fixes the resolution or number of details that can be represented.

3.2 Implicit methods

As alluded to in chapter 1, the goal of implicit representations is to describe something, in our case shapes, in terms of a function. The shapes can then be reconstructed from this function in any desired resolution by sampling it at different positions. The papers that first used implicit representations in the 3D domain with such success that the approach became popular are ”Occupancy Networks: Learning 3D Reconstruction in Function Space” [Mes+19], ”Learning Implicit Fields for Generative Shape Modeling” [CZ19] and ”DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation” [Par+19].

3.2.1 Occupancy

We will name the first kind of implicit representation to be discussed occupancy representation. This class of implicit methods learns a function, which classifies a coordinate to be inside or outside the shape. This idea was popularized by the methods [CZ19] and [Mes+19]. The first one classifies whether the coordinate is outside the shape, while the second method does the opposite, but this should not impact the quality of the methods, as just switching 0 and 1 should not lead to radically different results. Figure 3.1 shows the side view of what an occupancy field of a chair could look like. It's important to note that even though the image has been rasterized in a certain resolution, the implicit function can be sampled at any resolution. This occupancy can be thought of as taking the sign of a signed distance field [CZ19].

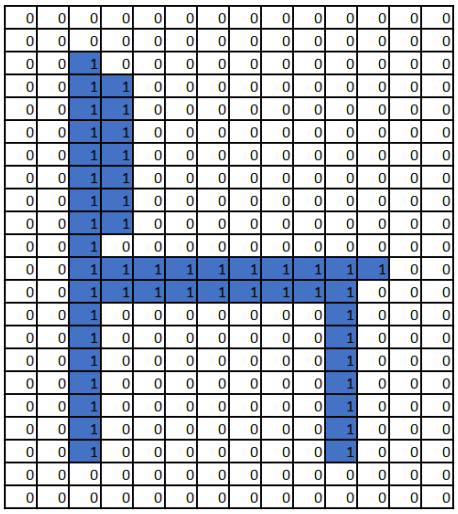


Figure 3.1: Side view of a chair visualized as occupancy field

In terms of loss functions, occupancy methods can use the same classification losses other fields use. Chen *et al.* [CZ19] uses a weighted mean squared loss, [Mes+19] and [Nie+19] use a cross-entropy loss and [Hua+20] use a loss consisting of several losses added together, also factoring in the prediction of color. Any classification loss can potentially be used. If the latent representation of the shapes has to fit a certain distribution, this part must be added to the loss.

Numerous coordinates have to be classified in the domain of the function to recover the shape in a visualizable format from this kind of implicit function. The collected information then needs to be aggregated to get back the shape. This procedure can be done by rasterizing the domain into pieces. How fine-grained the raster has to be depends on how good the quality of the reconstructed shape needs to be. Algorithms like marching cubes can be used to recover a mesh from this voxelized representation. If point clouds are needed, points can be sampled from the mesh [CZ19].

These first methods were limited to small shapes or low resolutions. To capture whole scenes with a neural

network, researchers started to segment the input and learn local implicit occupancies with the new problem of how to recombine them. These methods often need accurate normals to prevent conflicting signs in overlapping regions of the local fields. Sign agnostic methods were designed to combat this [Tan+21; AL20].

There have also been approaches that go a dimension higher. In 4D reconstruction, not only one scan is being reconstructed, but a video over time. Here the input is not only a set of coordinates indicating which point in space needs to be classified but also an additional scalar indicating which point in time needs to be reconstructed [Nie+19].

Also, methods have been developed that use humans as input shapes and are able to not only reconstruct the occupancy of the input but can also reconstruct the person in canonical space. With this canonical representation, the person can not only be reconstructed in the original pose but the human model can be made to have a completely different pose. Normals and color can be constructed with this method as well [Hua+20].

For architectures, all kinds of encoders discussed in section 3.3 are used. The decoder, which is the part that learns the implicit function, usually consists of several fully connected layers. [CZ19; Mes+19; Hua+20] use fully connected layers and use skip connections for the first layers transporting the initial input coordinates and vector encoding of the desired shape.

3.2.2 Distance Functions

The second type of implicit representation uses distance functions. A distance function of a shape gets coordinates as input and returns the distance to the closest point of the shape as output. These methods learn the continuous distance field of a shape. Park *et al.* popularized this representation by predicting the signed distance of the desired point [Par+19] as visualized in figure 3.2.

Distance functions come in different kinds of flavors. The first one is the classical signed distance function used by [Par+19] which has already been described in section 2.1. Truncated signed distance functions see use in research as well [Byl+13; Cho+21]. Another approach is to not use an SDF but an unsigned distance function. Not having a sign has the advantage compared to SDFs in that there is no inside the shape. It is possible to model the interiors of a shape if points can be inside and outside something simultaneously, which would not be possible with SDFs as seen in figure 3.3. Despite that, SDFs are more popular and are easier to render than unsigned distance functions [CMP20; Zha+21b].

Numerous loss functions used for regression problems are potentially applicable to optimize a distance function. Park *et al.* [Par+19] use their clamped L_1 cost function. Sitzmann *et al.*[Sit+20] and Zhao *et al.*[Zha+21b] compare the absolute difference of the predicted and the ground truth unsigned distance, but have a maximal distance after which exact values are not important anymore. More details about this loss can be found in section

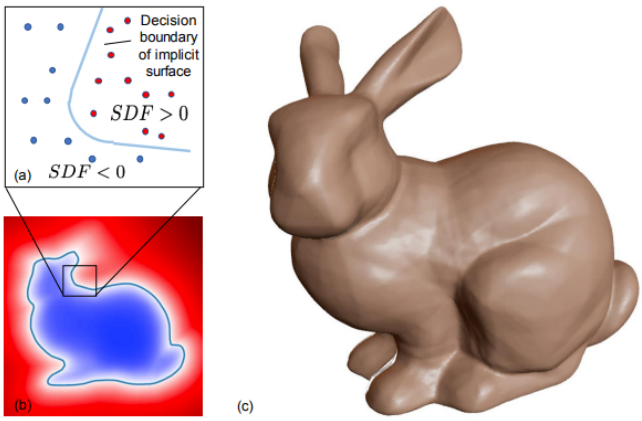


Figure 3.2: Example of a bunny (c) being represented by a distance field (b). The decision boundary of the SDF separates the inside and outside of several coordinates in (a) by Park *et al.* [Par+19].

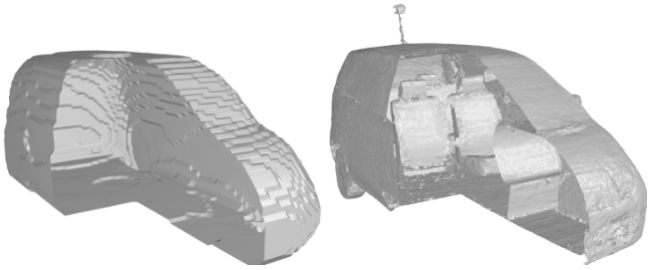


Figure 3.3: Comparing the reconstruction of SAL (left) vs. NDF (right) to highlight unsigned distances (right) can learn interiors by Chibane *et al.* [CMP21].

4.3. SIRENs' [Sit+20] loss function, which has been analyzed in this section is a loss consisting of 3 different parts. This loss handles on-shape points differently from off-surface points and imposes constraints upon the network, making it represent the normals more accurately [Sit+20]. Many losses do not use SDFs values but rather the rendered image or a combination of both. One example method uses the L_1 between the correct and the predicted color, the absolute difference between the predicted SDF and the estimated one, and the difference between the deep image features an encoder returns for the original and the reconstructed shape [ZYQ21].

Several methods exist for restoring the whole shape from a distance function. Marching cubes or raycasting is used to get the shape back from the isosurface of the SDF [Par+19]. Differentiable renderers have been designed to be able to learn the SDF of a shape without having 3D data available [Jia+20].

Not only 3D data is used to learn a distance function. Zhao *et al.* [Zha+21b] uses images and some anchor points in 3D to improve their single view reconstruction of open shapes. Choe *et al.* [Cho+21] reconstructs whole scenes using truncated signed distance functions using images. From the images it gets, the method creates a depth map, which is used by its auto-encoder to learn the TSDF.

Sitzmann *et al.* [Sit+20] developed a method using sine activation functions instead of more traditional activations like the ReLU. The advantage of sine activations is that the derivative of the network can be trained to be similar to the normals of the shape. Similar to the methods in section 3.1 using a sphere as a starting point to fit a shape, [BKG22] developed a way to initialize methods using sine waves to start with a sphere.

Similar to the occupancy methods, most methods using distance functions usually approximate them with fully connected layers [Par+19; Zha+21b; CMP20]. [CMP20] only use CNN layers in their method. Their distance function is learned through 1D CNN layers.

3.3 Shape Space Learning

Whether we are using implicit representations or some other deep learning approach, to be able to learn what more than one shape looks like, the model needs to know which shape the user is currently interested in. This means there is a need for a way to encode a shape space in a way the models can understand. Usually, this machine-understandable way to represent a shape is an additional input for the model trying to reconstruct the shape. For implicit methods, the input is commonly the coordinates of the point of interest plus an extra vector encoding a shape [Par+19; CZ19; CMP20]. There are two popular techniques to get the encoding of a shape in the context of 3D reconstruction. The first and most used method is an auto-encoder or encoder-decoder architecture. Some papers using this methods are [Mes+19; CZ19; CMP20]. The second approach, wihc is implemented by [Par+19; BKG22], is using an auto-decoder.

As there are many input types, many different ways to encode them exist. Similar to the 2D domain, CNNs are very popular for encoding 3D data. 3D CNNs are very computationally expensive, but the quality of their output makes many researchers use them [Kam+17]. The most straightforward approach is using a few layers of a 3D CNN and using the final output as the encoding of the input shape. Chen *et al.* [CZ19] uses this approach to gain their embeddings. There are many deviations from this base version that can be experimented with. One modification is to not only use the output of the last CNN block to obtain the encoded shapes but to use deep features from several layers to get a richer representation. One idea is to only use part of the output of the encoder because the output of a CNN is a space itself, and the model can focus on the interesting region of the shape. An adjustment that [CMP20] has used together with deep features is not using all the features. Only the features that are in the vicinity of the input coordinates are forwarded to the decoder. Adding norms like the batch norm and pooling layers are used to improve the quality of the encodings [CMP20; Wan+18].

Not only 3D CNNs are used to obtain a vector representation of a shape. For some applications, it is beneficial to get an encoding from only an image or several images. Zhang *et al.* [Zha+21a] uses an image of a scene as the input and uses a ResNet to encode the objects found in the image. Entire scenes have been reconstructed

this way by acquiring the SDF of the detected shapes and placing them in a space. Also, 1D convolutions have been used to encode shapes, especially for encoding point clouds [Wan+18].

Away from CNNs, [JA21] learns how to create an encoding out of several images with a multi-layer perceptron. This model then uses this encoding to create images of the shape from different angles. Mo *et al.* [Mo+19] introduced StructureNet, which can represent a shape as a graph. These graphs can then be interpolated and used to create new shapes.

As seen above, there are many ways to encode a shape. The most commonly used method for small shapes when scans are available, which is the focus of this thesis, is a 3D CNN.

When using an auto-decoder, there are not many variants as with the different encoders. The only thing that can be adapted, apart from the size of the vector, is if the encodings need to fit a distribution. If that is the case, the type of distribution needs to be selected as well. Some papers that use this representation are [Par+19; BKG22].

Deciding whether to use an auto-encoder or an auto-decoder depends on the problem that is to be tackled and the data available: if the model needs to be able to encode shapes it has not seen during training, an encoder of some sort is required. On the other hand, if the data only needs to work on the training set, an auto-decoder is sufficient. If the shapes are not complete, for example, if only a part of a shape is present if the form of a point cloud, then 1D CNNs or fully connected layers are some options that should be considered. If the system needs to encode a shape from images alone, then a 2D CNN is a reasonable choice. Also, memory and time constraints must be considered when choosing the models' architecture. Many more parameters need to be learned and fit into memory if an architecture using many CNN, pooling, and norm layers is chosen, compared to an auto-decoder. Finally, if the goal is to get a reconstruction of a single shape, there is no need for an encoder and it can be skipped altogether like [Sit+20].

4 Methods

This chapter aims to analyze and compare the selected methods, which have been named in chapter 1. In this section, the methods will be described broadly, and the following sections focus on a certain part of the methods.

IM-NET The first and oldest selected paper is ”Learning Implicit Fields for Generative Shape Modeling” by Zhiqin Chen and Hao Zhang, which got published in 2019 and is recognized as one of the papers popularizing implicit methods for 3D reconstruction as described in chapter 3. It is a neural network learning to approximate occupancy functions. This model will from now be referenced as IM-NET, which is the name the authors used in the paper. They propose to use their new decoder to learn an implicit function instead of the previously used decoders. The paper uses existing state-of-the-art auto-encoders for representation learning and reconstruction for their experiments. Additionally, they worked on shape generation, where they used a Generative Adversarial Network to generate fake latent vectors to generate shapes. Their goal is to demonstrate the superiority of their implicit decoder in various tasks. In addition, they use progressive training techniques, which means they first train on lower resolutions and add more details later [CZ19].

NDF The second paper, ”Neural Unsigned Distance Fields for Implicit Function Learning, ” was written by Julian Chibane, Aymen Mir, and Gerard Pons-Moll in 2020. The method they developed is called Neural Distance Fields or NDF. Using unsigned distance functions as an alternative to SDFs is the idea behind the paper. As shown in chapter 3, unsigned distance functions have the advantage that they can not only learn the surface of a shape but also the interior structures and make it possible to learn open shapes. Previous renderers are not equipped to deal with unsigned distances, which is why they also invented their methods for rendering the shapes from NDF and for generating point clouds from them. The rendering will not be discussed to a great extent in this thesis as rendering is not its focus and the other methods don’t have their custom renderers to compare to, but because point clouds are used in the experiments, it is still relevant [CMP20].

SIREN SIREN is the name of the last selected method and is from the paper ”Implicit Neural Representations with Periodic Activation Functions” by Vincent Sitzmann, Julien N. P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. They show that sine waves are a great alternative to traditional activation functions like ReLUs. Sine activation functions have more advantageous derivatives in cos, -sin, etc. than ReLUs. With this property, the authors do not only fit the neural network to the functions but also use the derivatives, which makes their method learn more details of the targeted signal. For 3D shapes, SIREN does not only learn the SDF but also uses the property that the spacial gradient should be one almost anywhere as discussed in chapter 2. SIREN is not only used to learn the SDF of a single shape but also to test it on a video,

images, and several equations like the Poisson equation. To make the sine activations work, the weights of each layer need a special initialization. The distribution of the weights should have a uniform distribution from $-\sqrt{6/n}$ to $+\sqrt{6/n}$ where n is the number of inputs [Sit+20].

4.1 Architectures

A core part of comparing methods in deep learning is comparing different architectures. We will analyze the design of the selected methods starting with IM-NETs:

IM-NET The first part of IM-NET is its 3D CNN which encodes a voxelized shape into a 128-dimensional feature vector. This vector is concatenated with the input coordinates and the result is the input for the decoder. According to [CZ19] the decoder consists of six fully connected layers with dimensions 2048, 1024, 512, 256, 126 and 1. They also added skip connections until the layer with dimension 256 and the activation function used are leaky ReLUs [CZ19; CZ21].

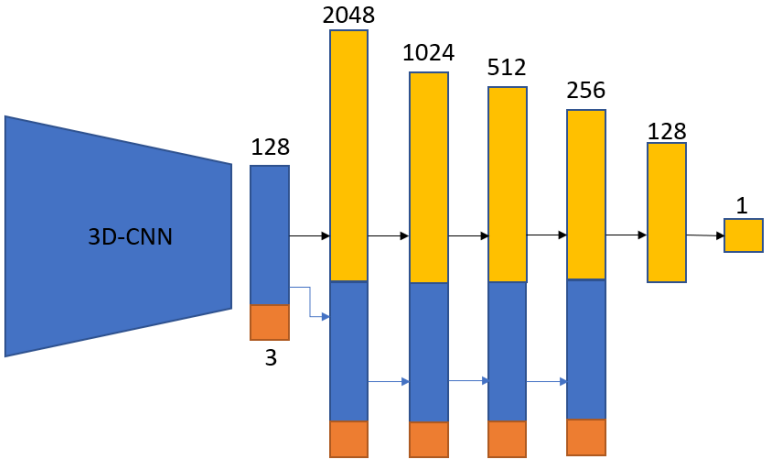


Figure 4.1: Architecture of IM-NET: The blue trapezoid represents the 3D CNN encoder. The output of the last layer (blue) is the embedding of the input shape. The coordinates (orange) are added to the embedding and the result is the input of the decoders' fully connected layers (yellow). The input to the decoder is also given to the subsequent layers in the form of skip-connections.

An improved version was released, removing the skip connections and replacing the layer of size 2048 with two of size 1048. In addition the feature vector now has a length of 256 [CZ21]. The improved version can be seen in figure 4.2.

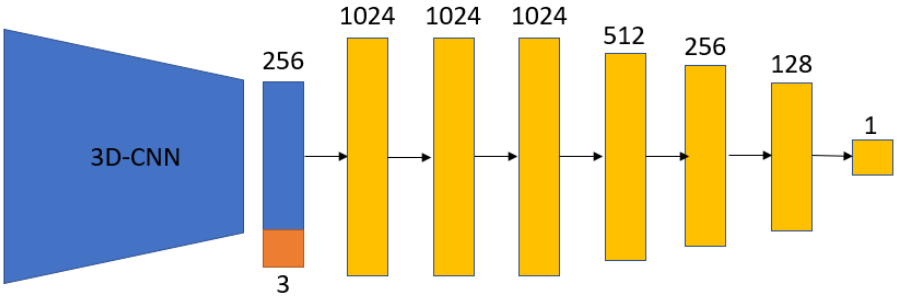


Figure 4.2: Improved architecture of IM-NET: The blue trapezoid represents the 3D CNN encoder. The output of the last layer (blue) is the embedding of the input shape. The coordinates (orange) are added to the embedding and the result is the input of the decoders' fully connected layers (yellow). There are no skip connections and the size and number of layers have changed from the original version.

NDF The architecture of NDF also starts with a 3D CNN, but the decoder does not simply take the output of the last CNN layer as the encoding. Instead, a multi-scale grid of deep features is generated by taking the output of several layers. However, only the relevant part is given to the decoder as highlighted in red in figure 4.3 [CMP20].This "taking the relevant part" can be imagined as the model only using the part that encoded the top left of an input shape when asked to return the distance of a point on the top left of the frame. In the code [CMP20] published, they used a batch norm layer and a max pool layer after every two 3D CNN layers and the selected non-linearity are ReLUs [CMP21].

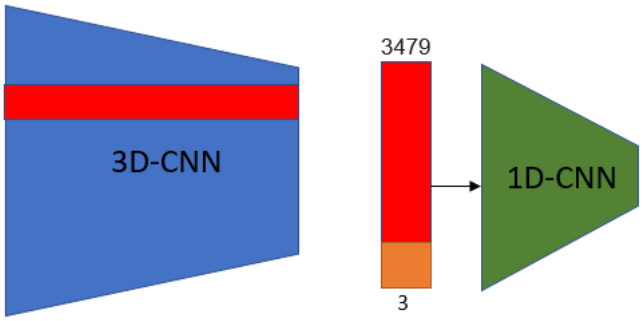


Figure 4.3: Architecture of NDF: The blue trapezoid represents the 3D CNN encoder. The red rectangle represents the part of the CNN, that is used for the decoder. The coordinates (orange) are added to the deep features and the result is given to the decoder (green) which consists of layers of 1D convolutions.

SIREN SIREN only consists of a decoder containing several fully connected layers, with the specialty being the activation function used. The activation function is a sine wave, which has the desired property of having more useful derivatives as opposed to ReLU, whose first derivative is one of two constants [Sit+20]. For learning SDFs, SIREN uses four layers of 256 neurons followed by an output layer of size one [Sit+22].

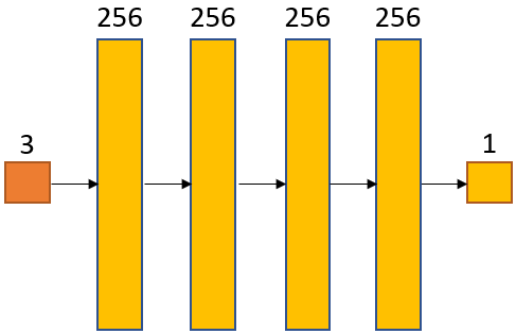


Figure 4.4: Architecture of SIREN: the coordinates (orange) are fed to the decoders' fully connected layers with sine activations (yellow).

4.2 Data Processing

An important part of learning systems is what kind of data they get. The preparation steps that required to get the data the models need are being described in this part.

IM-NET IM-NET obtains the voxelized shape directly from the shape files, which need to be in .binvox format. There are different ways, but this is the one being used in this thesis. These voxels need to have a resolution of 64^3 [Che22].

The point-occupancy pairs are generated by iterating over the voxelized shape and saving a point if it has inside and outside points in its vicinity, which means it is a point on the surface of the shape until the number of points surpasses a selected batch size. This batch size depends on the resolution of the input grid. If the batch size is exceeded before the iteration is completed, the process is started over again. But this time with a stride of 2 instead of 1. During this second iteration, not only points on the border are saved, but all selected coordinates make it into the training set. This is to prevent all the data from being only from one part of the shape. Instead, the shapes' structure is conserved without having to save too many sample points. Afterwards, if after the iteration over the whole shape the batch size has not been surpassed, random unseen points are chosen until the number of points matches the batch size. This plan is executed for resolution 64^3 and 32^3 voxel. To get the data for resolution 16, all 16^3 points are saved [Che22]. During training, you start by giving the model the 16^3 data to learn the rough structure. After 200 epochs, the 32^3 samples are presented to the model and after 200 more, it gets the 64^3 data. This way additional details are added as time progresses. Learning the broad structure first and details later makes training the model faster [CZ19].

To reconstruct the shape from the implicit representation, a grid is created and the resulting coordinates are

fed to the neural network in batches and the output is saved in an array. Similar to the preprocessing, coordinates that both have neighbors which are considered to be inside the shape and outside are filtered. These points are added to a queue to investigate further. For each of the points in the queue, a more fine-grained grid is prepared and used as the new input for IM-NET. In the code, the original grid has a size of 64 in all 3 directions and the fine-grained grid separates each gridpoint in the big grid again into 64^3 areas. If the maximum of the output of the fine-grained grid surpasses a threshold, its neighboring coordinates are added to the queue if they have not been examined before. The result of all this is an array with dimensions 256, 256, and 256 consisting of 1s and 0s indicating if this part is inside the shape or not[CZ21].

NDF To obtain the voxelized shape that is needed for the encoder of NDF, a k-d tree with the desired dimensions is built. 256^3 voxels are required for the input of the 3D CNN. The output space fills an area from -0.5 to +0.5 in all dimensions. 10'000 points are sampled from the input mesh. The occupancy of the voxelized shape is obtained by finding the closest neighbor of each point in the tree. The parts of the space with no point being its nearest neighbor are considered to be empty [CMP21].

A fixed number of points are sampled near the surface of each shape to create the input for the decoder [CMP20]. Chibane *et al.* sample 100'000 points of a mesh and add random noise to it to create off-surface points. The standard deviations (std) they used were 0.08, 0.02, and 0.003. This means, for each of the shapes, 100'000 points have been sampled per standard deviation. Afterwards, the signed distance of each point to the mesh is computed and the absolute value of the result is the ground truth unsigned distance. During training time, 1% of the data points the model sees are from the 0.08 std, 49% from the 0.02 std, and the rest are points with the std of 0.003. This means that points closer to the surface appear more often during training. It is important to note that what data the model gets is a stochastic process meaning the model is not guaranteed to see all data [CMP21].

To get back a point cloud from NDF, random coordinates between -1.5 and 1.5 are generated. For a fixed number of times, the normalized gradient is subtracted from the samples to get closer to the surface. The authors set the number of repetitions to 7. Points whose predicted distance is smaller than a threshold, which is set to 0.009 in the code, are saved. For points with a predicted distance of less than 0.03, random noise is added and the new coordinates are the input of the next iteration of this procedure to get more surface points. This algorithm is repeated until 900'000 surface points are gathered[CMP21].

SIREN SIREN requires a shape in .xyz as its input. The on-shape coordinates and normals used are the points present in the .xyz file. No sampling is being performed. During training, random coordinates are generated and treated as being off-surface points and their normal is set to [-1, -1, -1]. This means the same coordinate could be considered on and off the surface simultaneously if the random coordinates happened to match the coordinates of the on-surface points [Sit+22]. It is unlikely that a random number generator gets the exact coordinates, but if the number of on-surface points is limited, the learned shape can have holes where it should be closed. Similar to IM-NET a grid is created in the desired dimensions, in our case 512^3. Afterwards, the

marching cubes algorithm is used to obtain a mesh and the output is changed to fit the original coordinate system and scale.

4.3 Loss Functions

This section will compare the loss functions of the selected methods. In the following equations, θ will represent the weights if the neural network, $f_\theta()$ is a function parameterized by these weights. In other words, $f_\theta()$ is the neural net. S will represent the 3D shape, p stands for a point or coordinate, X stands for the set of all shapes we are interested in and lastly, $\mathcal{L}$ stands for the loss function. The functions are directly from the papers, but the notation has been changed to make them more uniform, which makes it easier to compare them.

IM-NET Starting with IM-NET, the loss function is characterized as follows:

$$\mathcal{L}(\theta) = \frac{\Sigma_{p \in S} |f_\theta(p) - \mathcal{F}(p)|^2 * w_p}{\Sigma_{p \in S} w_p} \tag{4.1}$$

In this equation w_p refers to the inverse sampling density near the point p. This has the goal to compensate for sampling more points near the surface of the shape. $\mathcal{F}$ stands for the ground truth inside/outside field [CZ19]. The loss function squares the difference between the predicted value and the correct one, which are both values between 0 and 1, and then takes the weighted average over all the points. In the code there does not seem to be a w [CZ19] as writing about in the paper [CZ21].

This loss function has only been defined for one shape in the paper. For learning more than one shape, the loss function could look as follows:

$$\mathcal{L}(\theta) = \Sigma_{S \in X} \frac{\Sigma_{p \in S} |f_\theta^S(p) - \mathcal{F}^S(p)|^2 * w_p^S}{\Sigma_{p \in S} w_p^S} \tag{4.2}$$

In 4.2 you sum up the loss over all shapes in your training's set. Additionally, the neural network, the ground truth, and the weights now depend on the shape.

NDF NDFs' loss function is defined as:

$$\mathcal{L}_\mathcal{B}(\theta) = \Sigma_{S \in B} \Sigma_{p \in P} |min(f_\theta^S(p), \delta) - min(UDF(p, S), \delta)| \tag{4.3}$$

Here, the function as been specified for a batch $\mathcal{B}$. P is a subset of the points in S and UDF(p, S) returns the true unsigned distance of point p in shape S. δ is a hyper-parameter that determines the maximal distance at

which you want your learned unsigned distance function to be exact. For instance, if a δ of 10 is used; if the real distance is 12, but the predicted one is 15, then this is considered correct because both are above the threshold. But if the distance was 8, and the predicted value was 7, then the error would have been 1. This part can be tuned, depending on how far away from the surface of a shape you want the function to be accurate [CMP20]. This highlights that it is not very important to know if a point is far or very far away from the surface as this method is only used to reconstruct the surface. For close points, 4.3 sums up the absolute difference between the predicted, and the correct distance of the points in P.

SIREN SIRENs' loss function for signed distance functions is:

$$\mathcal{L}(\theta) = \int_S |||\nabla_p f_{\theta(p)}| - 1||\,dx + \int_{S_0} |f_\theta(p)| + (1 - \langle \nabla_p f_\theta(p), n(p) \rangle)\,dx + \int_{S \setminus S_0} exp(-\alpha * |f_\theta(p)|)\,dx \quad (4.4)$$

S_0 are on-surface points of the shape in the equation. Equation 4.4 is comprised of three parts. The first part uses the property that signed distance fields have gradients of unit length and punishes the model if it deviates from this fact. The second part focuses on the zero-level set of the SDF and makes the points there have a distance of 0. Furthermore, this part makes the dot product of the derivative of the function and the normals of the points equal to 1, which means the two vectors need to have the same direction. The last section penalizes the neural network to predict a distance close to 0 for off-surface points, where α is a hyper-parameter for determining how much. α has to be above 1 for this to work [Sit+20]. This means that SIRENs SDF does not need to return an accurate estimate for the distance of off-surface points. It suffices that the distance is not near 0. SIREN needing the normals makes the sampling for the method harder, as not all shape file formats save normals.

5 Experiments

The chapter 4 compares the methods on a theoretical level. This approach is not sufficient to evaluate the methods, as it entirely ignores the differences in quality of the resulting reconstructions the different methods output. Here, experiments are conducted to compare the effectiveness of IM-NET, NDF, and SIREN, which corresponds to RQ2. In addition, this chapter will also show the attempts to improve SIREN (RQ3) and the visualization of the shape embeddings (RQ4).

NDF and IM-NET are trained on part of the ShapeNet data set as described in section 5.1. As SIREN can only handle a single shape, instead of learning the whole data set, a single chair is chosen as the target of the reconstruction. The documentation of these experiments can be found in 5.2.

Afterwards, in section 5.3 SIREN has been selected as the method to experiment with to try to improve its results. In the last section, the procedure is outlined, which is used to visualize the embeddings of the shapes the selected methods NDF and IM-NET generated.

The weights and biases library was used to record the experiments. With this tool, the loss progression and the GPU utilization have been documented. To ensure the hardware is fully exploited, hyper-parameters, like the batch size, have been tuned to maximize GPU usage. The results of the experiments can be found in chapter 6.

5.1 Data

For all the experiments the popular data-set ShapeNet has been used to supply the shapes. ShapeNet is a data-set containing millions of models of 3D shapes consisting of everyday objects. Some of these shapes have been hierarchically categorized. A part of this data set has been annotated geometrically, affordances have been added, and many more potentially useful annotations have been included. What's essential for the experiments is that the models have a consistent canonical orientation, which helps the neural networks not return upside-down shapes. ShapeNetCore is a part of this data set with a manually verified categorization and alignment. This subset of ShapeNet consists of popular shape classes like cars, tables, airplanes, chairs, and many more [Cha+15].

The classes of shapes that have been selected for our experiments are 03001627 and 04379243, which are chairs and tables respectively. 7668 chairs and 8436 tables are present in the current version of ShapeNetCore.

There are big varieties in the categories; for example, some chairs look more like sofas, making the data set diverse. The data has been sorted alphabetically by name of the file and the last 30% are the validation set(10%) and the test set(20%) for the experiments. The data is available in several file formats, but the format utilized during the experiments is .obj. There are textures and different materials saved for some shapes, but these are not relevant for the neural network part but can be seen in the rendered 2D versions of the shapes in section 5.4.

Figure 5.1: Examples of chairs (top-left), laptops (top-right), benches (bottom-left) and airplanes (bottom-right) of ShapeNet which have been aligned by Chang *et al.* [Cha+15].

5.2 Comparison

This section is concerned with the experiments conducted to compare the performance of the selected methods. As only NDF specializes in not only learning the exterior structure of shapes but also its interiors, this function is not being evaluated. Each subsection is dedicated to one of the methods.

5.2.1 IM-NET

The experiments conducted with IM-NET have been performed with a computer using Ubuntu 21.10. The GPU used is an NVIDIA GeForce 2080 super. The code runs on Python 3.6 and uses PyTorch 1.4 and Cuda Toolkit 10.1. The preprocessing used is the same as described in section 4.2. As the shapes are not available in the .binvox file format, a script has been created to transform the data from .obj to .binvox. The shapes are made to be in a space from -0.5 to 0.5 in all dimensions. The generated data is in a .hdf5 file. The architecture of the

improved version has been utilized for the experiments.

The model has been progressively trained with the same weights on 16^3, 32^3 and 64^3 voxel data of both classes with 200 epochs on each resolution, and the input of the encoder had a resolution of 64^3. IM-NET was trained using the Adam optimizer with a learning rate of 5e-5 and a beta1 of 0.5. Training on all the resolutions with both classes took about 15.5 hours and 12 hours for just the chair data.

As visible in figure 5.2, the training on 16^3 resolution does most of the work. This run is flat at first but then learns the broad structure of the shapes. The later parts of the training with higher resolutions are learned in a few epochs and only small gains are made after the first few epochs.

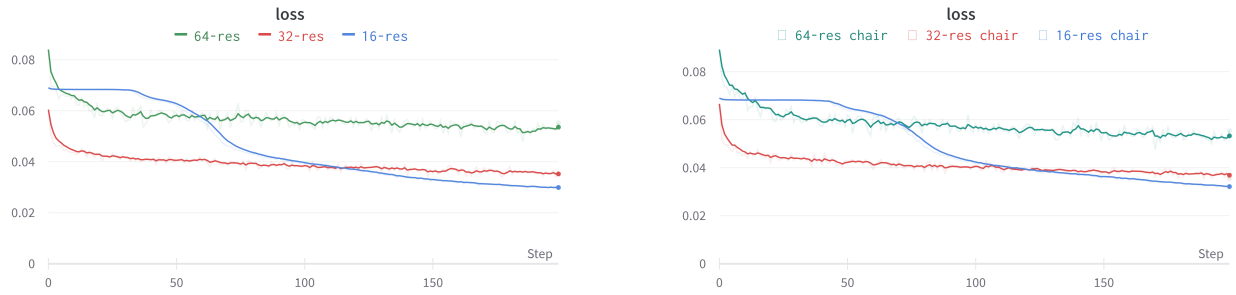


Figure 5.2: Loss progression of the training of IM-NET with both classes (left) and only chairs (right). Blue shows the training on resolution 16, red the training on 32, and green the training on 64 voxel resolution data. Steps correspond to epochs. An exponential moving average with a smoothing of 0.5 was used to smoothen the curve.

5.2.2 NDF

Because the compute requirements of NDF are sizable, the experiments have been done on a GPU cluster of the University of Zurich. Trials have been conducted with two versions of NDF for the single- and multi-class experiments. The first version for both uses only one NVIDIA GeForce 2080 TI GPU, but this did not seem sufficient for the multi-class trial, as shown in figure 5.3; the loss stayed approximately the same for the whole duration of the training. The second version of the multi-class experiment was done using 4 NVIDIA GeForce 2080 TI GPUs. All versions needed to be early-stopped because the selected number of epochs would take months or years to train. The actual number of training epochs for the single class experiments is 30 hours with one GPU and 5 days and 3 hours for 5 GPUs, corresponding to 5 and 78 epochs, respectively. For the multi-class experiments, the single-GPU experiment was trained for 15 epochs (9 days and 18 hours) and 54 epochs (10 days and 7 hours) of training were used for the experiment with 4 GPUs. The loss curves in figure 5.3 suggest that further training would not improve the quality anymore. All experiments have been conducted on a

cluster running Ubuntu 21.10 and Python 3.6 using PyTorch 1.4 and Cuda Toolkit 10.1 was the tech stack used.

The preprocessing used is the same as described in section 4.2. The same standard deviations of 0.08, 0.02, and 0.003 have been used and the data is distributed such that 1% is from the first, 49% from the second, and 50% from the third standard deviation, which are values that worked for [CMP20]. The .obj files of ShapeNet have been used to obtain the samples. The input resolution for the encoder is 256. The batch size corresponds to the number of GPUs used. A learning rate of 1e-5 and the Adam optimizer were used for all but the single-GPU single-class trials. For this experiment, a learning rate of 1e-6 was used. In figure 5.3 the validation loss represents running 15 random shapes from the validation set and calculating the average. It was decided against setting epochs as steps because it takes too much time to gain information about how the training is doing otherwise.

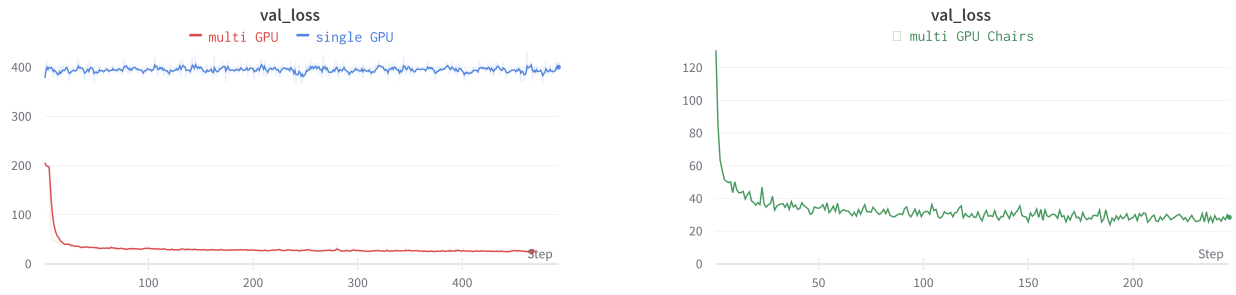


Figure 5.3: Validation loss progression of the training of NDF on both classes (left) and only chairs (right). Blue shows the training with one GPU and batch-site of 1, red the training with 4 GPUs and batch-size of 4. Green represents the single-class training with a batch size of 5. Steps correspond to training time. One step is one hour for the multi-class training and 30 minutes for the single-class. An exponential moving average with a smoothing of 0.5 was used to smoothen the curve.

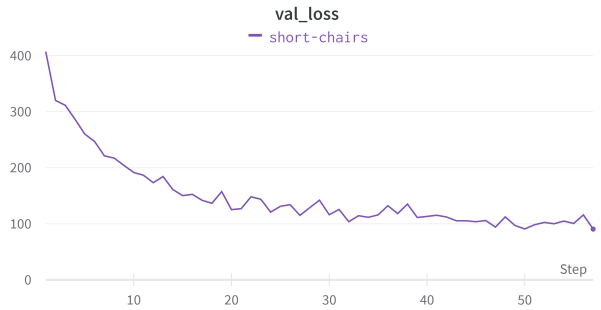


Figure 5.4: Validation loss progression of the training of NDF on chairs with one GPU and a batch size of 1. An exponential moving average with a smoothing of 0.5 was used to smoothen the curve.

5.2.3 SIREN

SIREN has been tested with a computer with Ubuntu 21.10, and the GPU used is an NVIDIA GeForce RTX 3090. The code runs on Python 3.6 and uses Pytorch 1.5 and Cuda Toolkit 10.1.

A script has been created to obtain the shapes in .xyz format, sampling points from a .off file of ShapeNet using PyMeshLab. The selected shape is a chair with the code 1a6f615e8b1b5ae4dbbc9440457e303e and can be seen in figure 5.5.

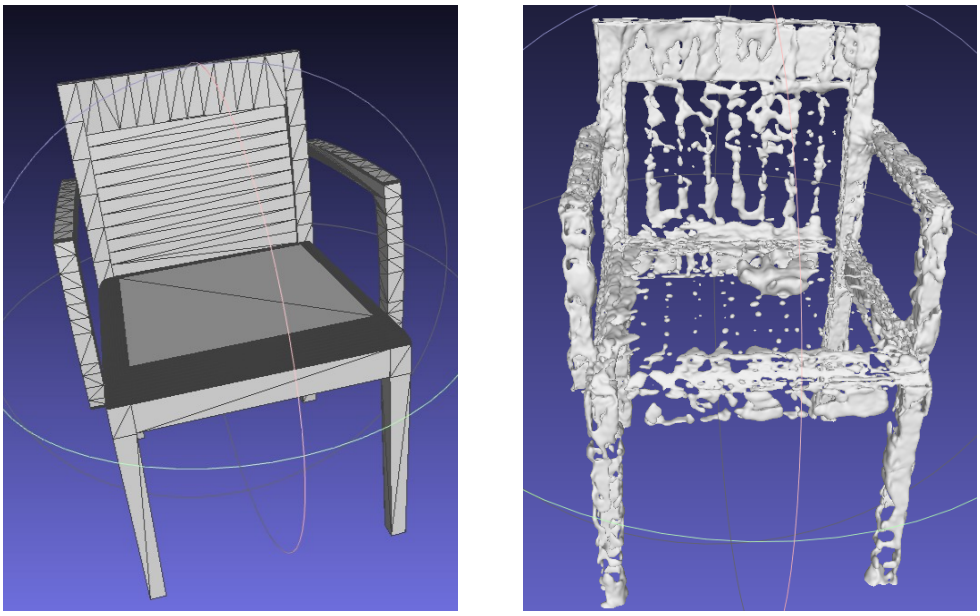


Figure 5.5: Chair 1a6f615e8b1b5ae4dbbc9440457e303e from ShapeNet visualized in MeshLab (left). Reconstructed chair from SIREN with standard MeshLab sampling (right).

The initial result with the standard sampling settings was substandard and produced a reconstruction with many holes, as shown in figure 5.5. This result led to the usage of a very high sampling density of 0.01%.

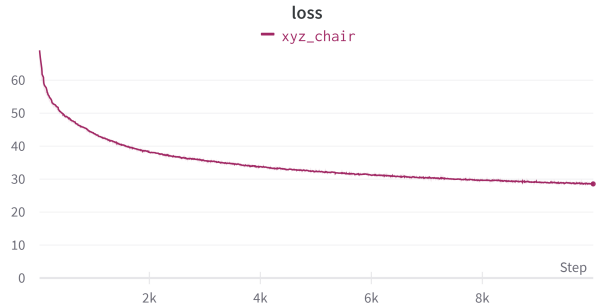


Figure 5.6: Loss progression of the training of SIREN. An exponential moving average with a smoothing of 0.5 was used to smoothen the curve.

5.3 SIREN Extension

SIREN has been chosen as the target for extra experiments testing if some extensions positively impact the performance. It was chosen because it can only encode one shape and the way it uses the data seems sub-optimal. The authors do not measure distances of points as seen in section 4.2. We consider sampling the most significant weakness and the main focus of our trials. The code NDF uses for point sampling the shapes is changed to fit SIREN and used as the basis for the experiments. The reason for this choice is that the sampling they use can easily be adapted by for example using other standard deviations, using points on the shape, etc. In addition, the normals are already there for the taking, which is an important component for making SIREN work. The environment used for the experiments is mostly the same as the one in the SIREN paper and what was used in section 5.2.3, but some extra libraries needed to be added for some tests. 300'000 points are shown per epoch of training.

Because the loss of SIREN deals with off-surface points differently from on-surface points, it has to be decided how to define surface points and how to generate them. If the sampling of NDF is used, which adds some noise to surface points to generate the samples, some points are naturally close to the surface of the shape. SIREN requires a lot more on-surface samples than NDF. Half of the data SIREN uses are surface points. Only having a few on surface samples would not work, as assigning large distances for all coordinates would be an easy solution to lower the loss. One idea is to only add this noise to a fraction of the samples. This way, clean on-surface points are generated. There is also the problem of separating on-surface from off-surface points. Considering all points with noise to be off-surface creates a similar problem as the original SIREN had, which is having points very close to the surface, or even on the surface, considered off-surface points. An obvious solution is setting an ϵ. Any point with a distance less than this number is considered an on-surface point. But then the new problem arises of deciding how big this ϵ is. Here are some trials for testing this idea of sampling similar to NDF and using an ϵ:

Figure 5.7 shows some example runs with different decision boundaries and amounts of samples with no noise

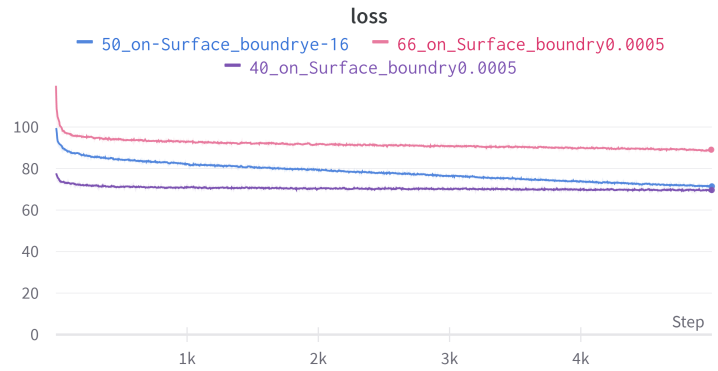


Figure 5.7: Loss progression of the training of SIREN with different decision boundaries and % of points of samples without noise. The red curve consists of 66% points with no noise and a boundary of 0.0005. Violet uses 40% noiseless points and the same boundary. The blue loss curve has a 50/50 split of points with and without noise and the boundary is set to $1e^{-16}$. An exponential moving average with a smoothing of 0.5 was used to smoothen the curve.

added. When changing the number of points that have noise, the loss curve shifts vertically, indicating the exact amount of points with noise is not crucial to how well the model performs. The shift occurs because the error of on-surface points is weighted higher in the code. When using a boundary of 0.0005, SIRENs loss curves are mostly flat, which leads to the belief that the model does not learn successfully. With a lower boundary, the loss has a more negative slope indicating some improvement. Due to computational inaccuracies with the function that returns the distance of a point to a surface, distances greater than $1e^{-16}$ cannot be used. Going forward, only boundaries of at least $1e^{-15}$ will be used. Figure 5.8 shows a point cloud of all the samples that are considered to be on the surface. It clearly reassembles a chair which leads us to believe the data is of sufficient quality to learn the broad structure of a chair.

SIREN does neither calculate the distance of points nor does it try to learn them. All it does is try to make on-surface points have low values and other points have high values. A change from this separation of categories towards trying to learn the exact distance could improve the reconstructions. The new loss is defined as:

$$\mathcal{L}(\theta) = \int_S ||\,|\nabla_p f_{\theta(p)}| - 1\,||dx + \int_{S_0} (1 - \langle \nabla_p f_\theta(p), n(p) \rangle) dx + \int_S |f_\theta(p) - SDF(p)| dx \tag{5.1}$$

The slope of the loss curves in figure 5.9 with the loss from equation 5.1 are steeper than the reference, but overall the losses look very similar. The difference in noise only changes the curves by a constant factor. More noise means the distance is on average farther away.

As can be seen in the original and the updated loss function of SIREN, only the gradients of the on-surface points are used. This is because we have no access to the normals of the off-surface points in the original version. By not adding random noise to the original surface points but sampling along the normal, it is possible to

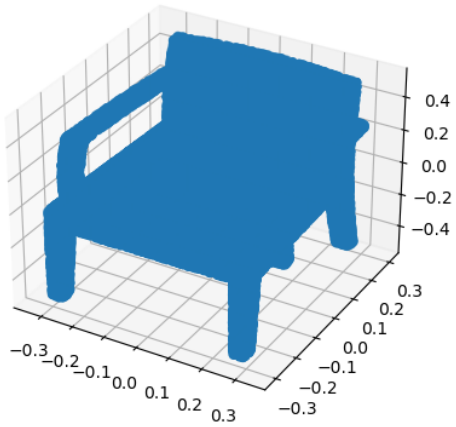


Figure 5.8: 3D scatter-plot of the coordinates of a chair that are considered to be on-surface points.

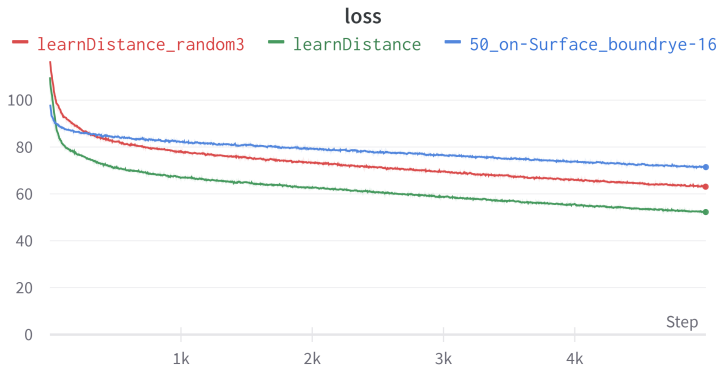


Figure 5.9: loss progression of the training of SIREN when learning exact distances. The blue loss curve is a reference using the original loss. The red and green curves are the result of learning the exact distance between points and the surface. For the red curve, more noise was added. An exponential moving average with a smoothing of 0.5 was used to smoothen the curve.

also have off-surface data to work with. One issue with using all normals is the farther away from the original surface point a sample is, the bigger the chance that the normal might change, distorting the data. Despite that, it is worth exploring if this additional data helps the system learn. If we continue using the loss from equation 5.1, the loss can look as follows:

$$\mathcal{L}(\theta) = \int_S \Big|\big||\nabla_p f_{\theta(p)}| - 1\big|\Big| dx + \int_S (1 - \langle \nabla_p f_\theta(p), n(p) \rangle) dx + \int_S |f_\theta(p) - SDF(p)| dx \tag{5.2}$$

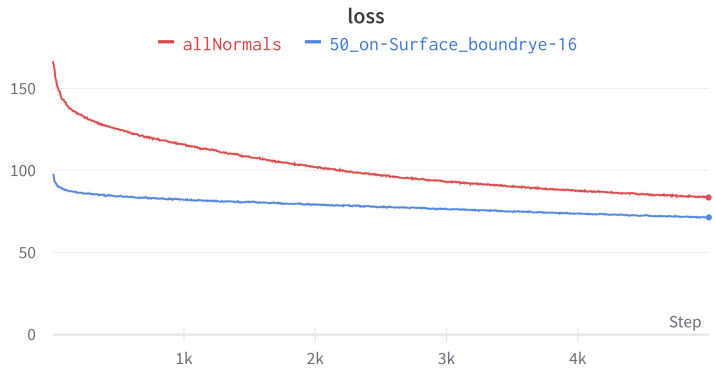


Figure 5.10: Loss progression of the training of SIREN when learning exact distances. The blue loss curve is a reference using the original loss. The red curve is the result of learning the exact distance between points and the surface, sampling along the normals, and making the off-surface normals have the same normal constraint as on-surface points. An exponential moving average with a smoothing of 0.5 was used to smoothen the curve.

All shown experiments in this section were trained over 5000 epochs, which was sufficient to show how the versions performed. Some models have been trained longer, but no visible benefit was detected in the reconstructions. Furthermore, changing the amount of data drawn from each distribution did not affect the results. The encoders of the other two methods have also been tested together with SIREN, but these experiments have been abandoned.

5.4 Visualisation

As seen in chapter 3 and 4, to not only learn one shape but many shapes, a way to encode them is needed. This section deals with creating a tool that helps to get a feeling of what these embeddings might look like and analyze them. Visualizing the embeddings in a lower-dimensional space facilitates their understanding. Thirty chairs and tables from the test set have been randomly selected, whose vector representation is the target of the visualization. The tool needs to have the affordance to visualize multiple embeddings simultaneously to be able to compare them. In addition, users need to be able to see what the original shapes look like and be able to inspect the reconstructed ones.

5.4.1 Preprocessing

The pre-processing step computes the embeddings to accelerate the loading in the visualization tool. For IM-NET, a script has been created to save the encodings in a .npz file. This script is used to get the representations at the end of the 16, 32, and 64 voxel training. A second script has been created to perform dimensionality reductions on the embeddings and prepare the data for visualization. T-distributed stochastic neighbor embedding with 250 iterations and principal component analysis are the dimensionality reduction techniques that have been selected. For both methods, the transformation with the first two components have been saved together with a path to the reconstructed shape. For all resolutions, the first two principal components preserve about 45% of the variance.

To obtain the embeddings for NDF the ”generate” script has been changed to also save the embeddings and a second script selects the relevant shapes and uses the same dimensionality reduction methods to prepare the data. As discussed in section 4.1, NDF samples the CNN outputs and only uses the relevant part. For the visualization, however, the embedding of the whole shape is needed. We decided to take all CNN outputs the model might use. The size of such a flattened embedding is over 60’000 dimensional, which is very big compared to the IM-NET. Due to the number of dimensions being this high, the number of points the dimensionality reductions can handle in terms of memory and speed is limited. Having too many points makes the visualization hard to read, which is why this is not a big problem. Another option would have been to only use the output of the last layer, but because the model uses more than that, this variant would miss most of the decoders’ input.

To get a sense of what the shapes look like, 2D renderings of them have been created with a script. The tool used for the rendering is an extension to the PyTorch library called PyTorch3D. With this tool, the angle, resolution, distance, and elevation level of the renderings can be specified. An image size of 256 has sufficient details to see what shape is currently selected. An angle of 120° with no elevation and a camera distance of 1 was used to get the best view of the shape on average. Figure 5.11 shows an example of a rendered image.

Figure 5.11: A chair rendered with PyTorch3D.

The question of why the embeddings are precomputed and not generated in real-time might come up. Note that, because creating the embeddings takes a long time, it is necessary to use a good GPU. Also, the dimensionality reductions and renderings have been performed beforehand to save time.

5.4.2 Visualisation Tool

Visualizing the prepared data was done with the python library Bokeh. Three subplots have been created to show several embeddings simultaneously. The number three has been selected because there are three stages of training for IM-NET whose embedding needs to be able to be compared easily. Each of these subplots shows a scatter-plot of the embeddings in two-dimensional space. The color of the plot indicates the category the points belong to. Each data point can be selected by clicking on it, and a fourth plot shows the rendered image of the selected shape. This point will also be highlighted on the other plots to see where in space the shape is positioned with the different methods. On the right, there are some drop-down menus for choosing different embeddings and dimensionality reductions for the different subplots. Finally, there is a button with the function to open a MeshLab window visualizing the reconstructed shape. The desired sup-plot can be selected with a drop-down, whose reconstruction should be visualized.

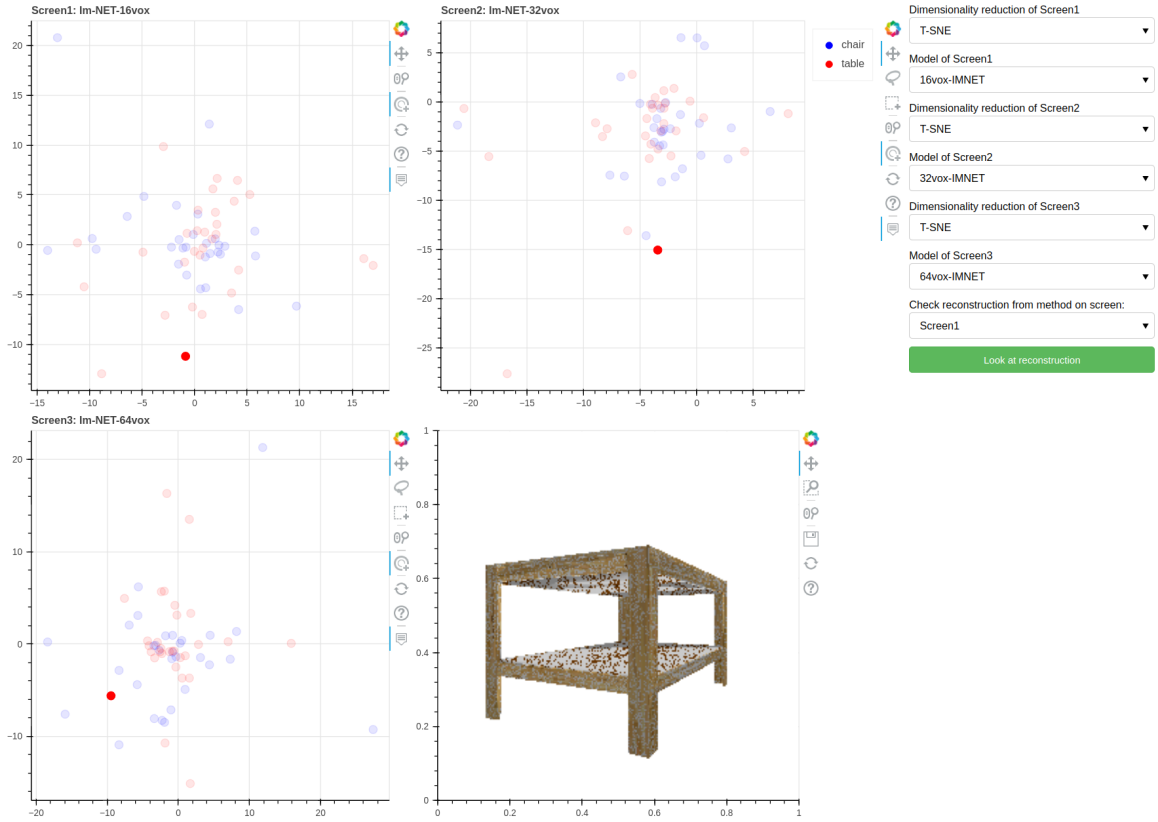


Figure 5.12: Screenshot of the created visualization tool. All but the selected data-point have been made partly transparent. The red points are tables and the blue points are chairs. The selected methods are: IM-NET with only 16 voxel training(top left) and IM-NET with 16 and 32 voxel training (top right), IM-NET after all the training (top left). T-SNE is the selected dimensionality reduction technique for all embeddings.

6 Results

In this chapter, the results of the experiment as well as additional thoughts are presented. Each section will deal with one of the questions described in chapter 1.

6.1 RQ1: What are the similarities and differences between the selected methods?

Using the information collected from chapter 4 the RQ1 will now be answered. Again, the focus will be on comparing attributes of the methods that chapter 4 used.

The three papers have very different goals. IM-NET wants to introduce implicit methods to the 3D reconstruction and generation domain. Due to them being one of the first methods using implicit representations for 3D reconstruction, the paper is more focused on showing it can work, not about squeezing every drop of performance out of the idea. NDF adds the ability to learn open shapes and interiors to implicit methods for 3D reconstruction. It is a later paper than IM-NET, trying to get the best result possible using implicit representations. Dealing with the challenges of using unsigned distances, like challenges in rendering, is also part of the paper. Lastly, SIREN focuses on making sine activations work. With these alternative activations, they show fine details on a single shape, but only a part of the paper is reserved for shapes. They are interested in all kinds of signals, such as videos or images. The different focuses make it hard to compare the methods. IM-NET is one of the first methods using implicit representations and the later methods had more related work in the area to gain insights and design better models. All the methods have been influential in their own way.

Going from the general idea of the papers over to architecture, SIREN is the biggest outlier because this method does not have an encoder at all. This has the disadvantage that it can only learn a single shape. In general, their architecture is very simple. The authors only spent time on making sine activations work, not finding the best architecture around it. NDF only looks at how the area of interest is depicted in the input, while IM-NET gets an encoding of the whole shape. It should be relatively easy to return a large distance for empty areas as empty with access to the outputs of all the CNN layers in the region. Its encoder makes NDF worse in tasks like shape generation because not only a single reasonable feature vector needs to be generated, but the output of all CNN layers needs to be synthesized. Every point needs its own generated embedding and they somehow need to work together. This is relevant because Im-NET is also used to generate shapes. NDF can get better performance by

ignoring generation tasks.

Two of the methods and many more [Par+19; Mes+19] use fully connected layers to represent the decoder. NDF goes a different route as it does not use fully connected layers at all, but only CNN, norm, and pooling layers. The parameter sharing a CNN uses might be necessary as NDF already uses a lot of memory. The activation functions of IM-NET and NDF are very standard, but SIREN uses sine activations which also separates it from all other previously used methods as seen in chapter 3.

The different architectures highlight the differences in goals the authors of the papers had. SIREN gets good at representing one thing, for example, one shape, one video or one image [CMP20], NDF is designed to be very good at reconstructing several shapes [Sit+20] and IM-NET needs to be good at various tasks like 3D reconstruction, generation and interpolation [CZ19].

Regarding data processing, SIREN and IM-NET need the data to be in a special format and the ShapeNet data-set is not available in them. The step of obtaining the correct data type might already mess with the results of the methods. NDF can use several formats but can already use the data provided in ShapeNet, giving them an edge in this regard. Additionally, it uses the most complex method to obtain the data they need. Sampling hundreds of thousands of points per shape takes a lot of time. This means enough time must be planned to prepossess the data when planning to use this method. SIREN depends heavily on the quality of the input file. If the points are not very dense, then the sampling causes these points to be considered off-surface. Both SIREN and NDF produce data that might have errors in them. NDF might create a wrong voxel input because the random sampling misses a part of the shape and SIREN tends to create off-surface points that are very close to the surface or even on the surface. IM-NET should be different from the other two methods when it comes to data usage as it uses occupancy instead of distances, but SIREN does not calculate distances either and considers two classes, off- and on-surface points. SIRENs sampling appears to not use the input data as well as it could. The preprocessing of the methods is very different. No standard in this regard seems to have been established.

SIRENs loss function equation 4.4 despite only being designed for one shape, is the most complex of the three. This complexity may be necessary to make the method work because [BKG22] took away the part with the normals and the performance deteriorated. The IM-NET loss is very different from the others because coordinates are classified, which is different from the regression-like objectives of NDF and SIREN, but similar to what has been said earlier about the data preparation, SIRENs loss has aspects of a classification loss. It has a part for pseudo classifying close points as far away and one for off-surface points being classified as on the surface. The two methods also differ in the level of accuracy the distances of the samples need to have for the system to work. NDF tries to get the exact distance of the points close to the shape. SIREN, on the other hand, wants the distance of off-surface points to be big. The sampling differences discussed in chapter 4.2 also play a factor in this as SIREN does not compute SDFs as NDF does. It is interesting to see that exact distances do not appear to be crucial to learning a distance function.

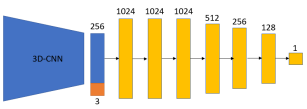
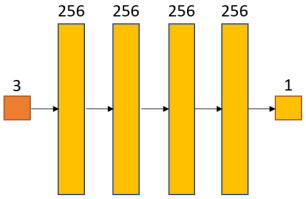
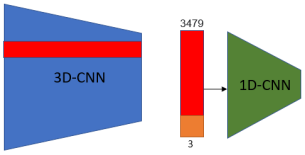
Comparison Overview			
	IM-NET	NDF	SIREN
Type	Occupancy	unsigned distance	SDF
Non-Linearity	Leaky ReLU	ReLU	sine
Encoder	simple CNN	CNN with sampled deep features	-
Decoder	fully connected	1D convolutions	fully connected
Architecture			
Data	sample coordinates from .binvox files	generate points with noise from .obj	utilize points from .xyz and generate random off-surface points
Loss description	weighted MSE	clipped absolute differences	contrastive loss including normals

Table 6.1: Overview of the selected methods.

6.2 RQ2: How do the selected methods compare in a practical setting?

6.2.1 Evaluation

10'000 Points are generated per ground truth shape to calculate the chamfer distance. The shape in the .obj file format is sampled to get the points. The formula used to obtain the distances has been described in 2.3. Figure 6.1 shows how the reconstructions do not cover the same area as the original shape. To combat this, scaling and shift factors needed to be found to make the comparison better. The shift factor is used to add to the coordinates and the coordinates are multiplied by the scaling factor. Additionally, the values outside the range of -0.5 and 0.5 are clipped to prevent random errors from distorting the distances too much. The full formula can be seen in equation 6.1.

$$\boldsymbol{coordinates}_{\boldsymbol{scaled}} = (\boldsymbol{coordinates} + shift) * scale \tag{6.1}$$

This distortion might not make the shapes perfectly align, but it is better than comparing shapes that are far away, as their distance would be guaranteed to be large. The reconstructions and the original are also not completely aligned because the reconstructions are suboptimal. There is no evaluation script in the GitHub directories of the methods, which makes it difficult to compare the methods optimally.

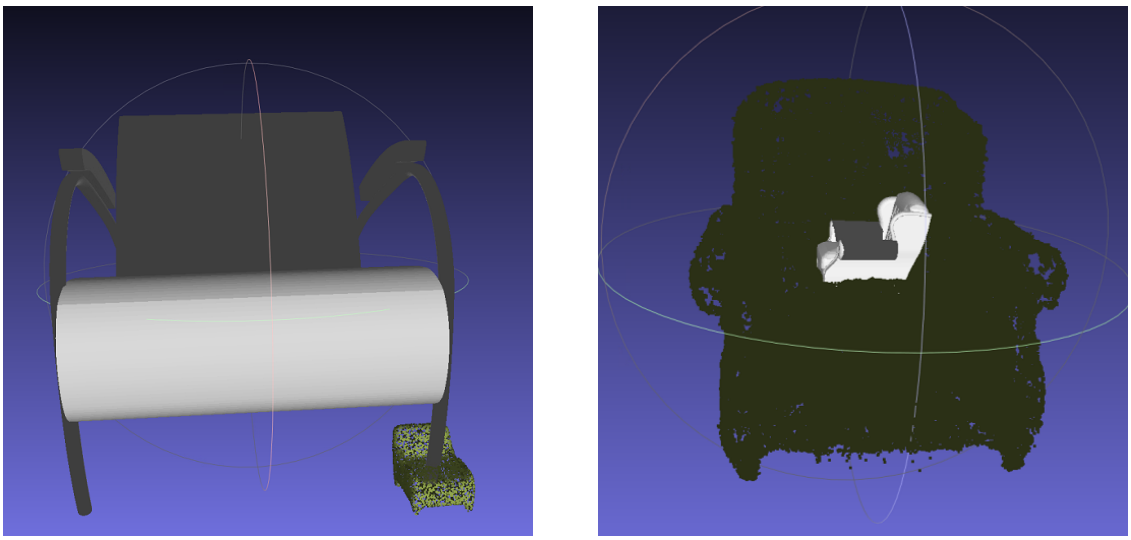


Figure 6.1: A misaligned reconstruction of IM-NET of a chair is depicted on the left. The original is the big shape, and the reconstruction the small chair on the bottom right. The right image shows the misalignment of NDF. The original is the white small shape, and the reconstruction is the big point cloud. Not only the size, but the directions of the chairs are different.

The scale and shift factors have been obtained by manually changing the values of the variables until acceptable scalars have been found which have a satisfying fit for the selected shapes and are compatible with the code base. What makes this problematic is that the reconstructions are not optimal. The scales could amplify the errors caused by the reconstructions. Still, this solution is better than the alternatives found. Both methods naturally create point clouds as outputs, which are used to calculate the distance. For IM-NET a few thousand points are being created. NDF, as can be seen in 4.2, NDF generates almost a million points per sample. This difference affects the chamfer distance because the more points there are, the smaller should the distance between a point and the closest point on the other set be if the reconstructions are good. It was decided to not sample differently because this way we stick closer to the original papers. For IM-NET the shapes are not only on a different scale but also shifted. A shifting factor of 0.375 and a scaling factor of 3 were determined to be the best value for the shapes on average. The corresponding image showing the alignment of an example shape can be seen in figure 6.2 on the left. NDF needed only to be re-scaled, not shifted. $\frac{1}{3}$ was used to create the point cloud in figure 6.2 on the right, which is consistent with the code. In the point-cloud generation procedure, points between -1.5 and 1.5 are generated and the ground truth spans an area of -0.5 and 0.5. The reconstructions from NDF are turned 90° to have the same orientation as the original shape.

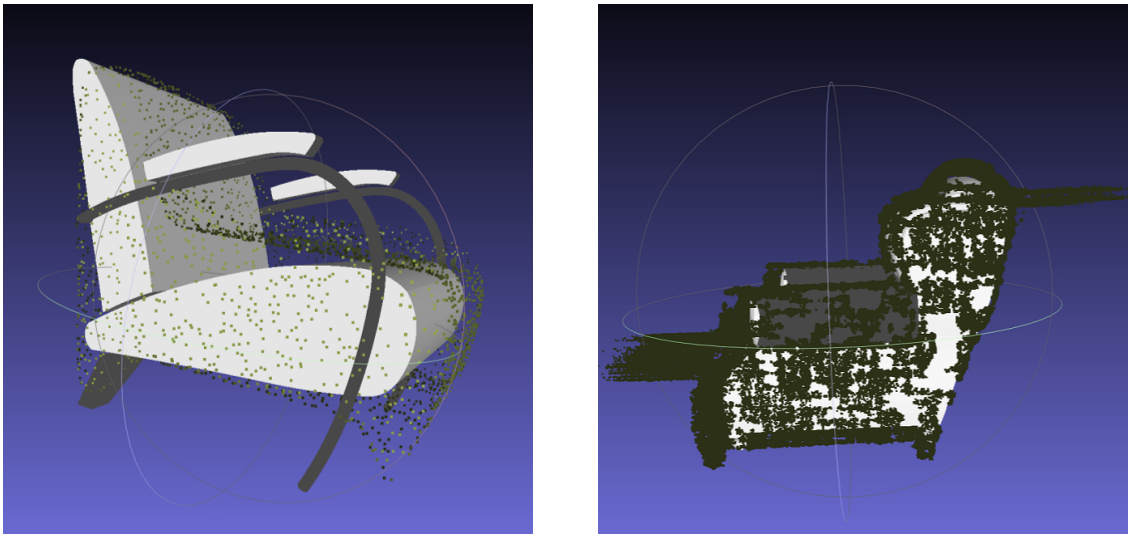


Figure 6.2: Re-scaled reconstructions from IM-NET(left) and NDF(right). The ground truth shapes are the smooth shapes and the reconstructions are the point clouds.

6.2.2 SIREN

SIREN is handled differently than the other two methods as it can only learn one shape. It does not need to generalize, just to memorize the training example.

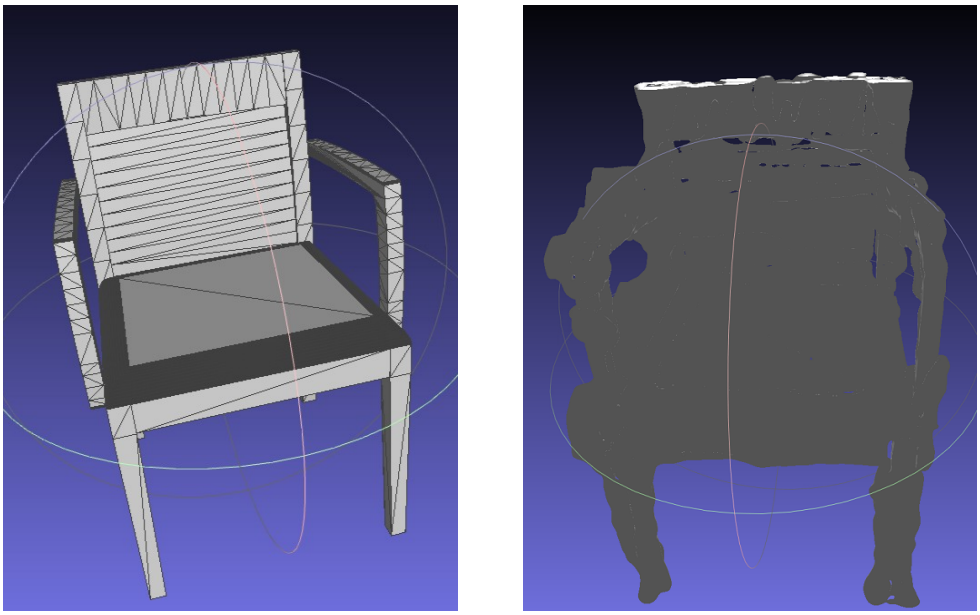


Figure 6.3: The original chair from ShapeNet is visualized on the left. A reconstructed chair from SIREN with high sampling can be seen on the right.

Figure 5.5 shows how SIREN behaves when a ShapeNet model is converted without much thought into an .off file and used as training data. The resulting reconstruction has many holes. This is due to SIRENs sampling treating random coordinates as off-surface samples. If the .off file has areas with few points, as is often the case for flat surfaces, the model is made to learn there are holes in the shape. In figure 6.3 an unusually high sampling rate is used to create the .off file and the resulting reconstruction does not have big holes anymore. Still, SIREN was not able to learn flat surfaces correctly. Instead, the parts that should be flat look like waves. This might be a consequence of using sine waves, but we believe better sampling could lead to better surfaces.

The authors of SIREN included their own example shape to test their method with. Their shape is a Thai statue with lots of details in the .xyz file [Sit+22]. In figure 6.4 can be seen, that the method performs very well on this shape because it naturally has a very dense setup of surface points and does not have large flat areas.

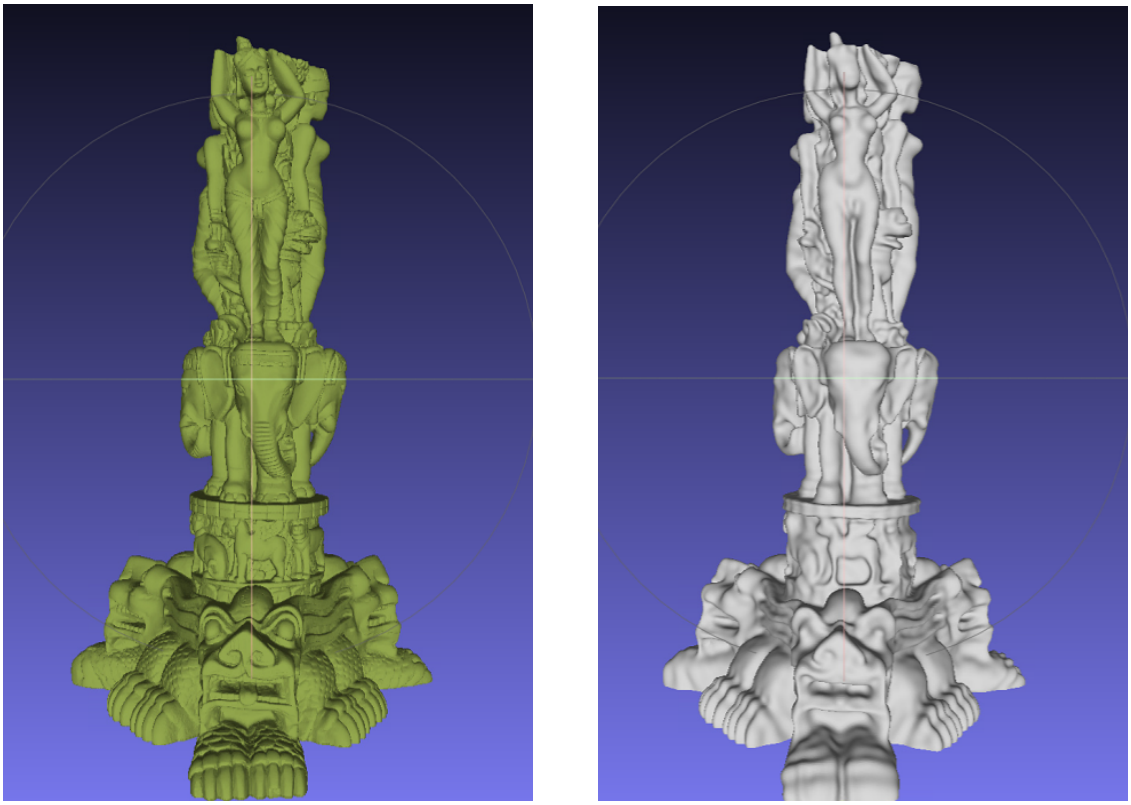


Figure 6.4: The ground truth Thai-statue from [Sit+22] on the left is reconstructed by SIREN on the right.

6.2.3 Chairs

This subsection is concerned with the IM-NET and NDF methods trained on the chairs category on ShapeNet. As opposed to SIREN the evaluation is conducted on a separate set of shapes, which the methods have not seen during training.

IM-NET Figure 6.5 shows qualitative examples of how IM-NET reconstructed the ground truth shapes from ShapeNet. The reconstructions look like chairs, but don't show the same level of details as the originals. Reconstructed chairs are rounder and thin parts, like the legs of chairs, are missing. Holes are closed, as can be seen in the first reconstructed shapes' armchairs. The chamfer distance can be seen in table 6.2.

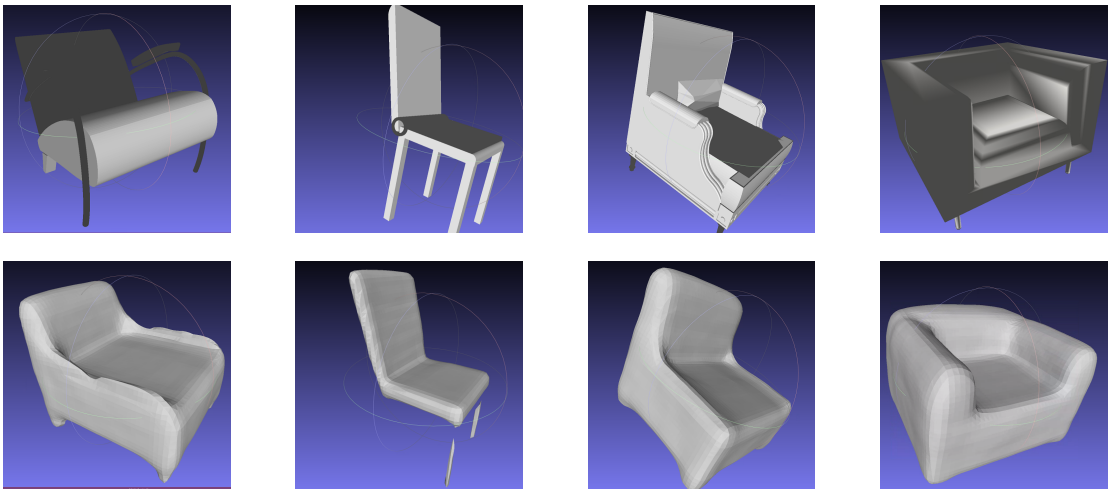


Figure 6.5: Examples of chairs that have been reconstructed by IM-NET. The top images represent the ground truth and the bottom the reconstructions.

NDF The reconstructions after running the whole training procedure are points over the whole codomain. No shape can be detected by looking at the reconstructions, which all look like figure 6.6.

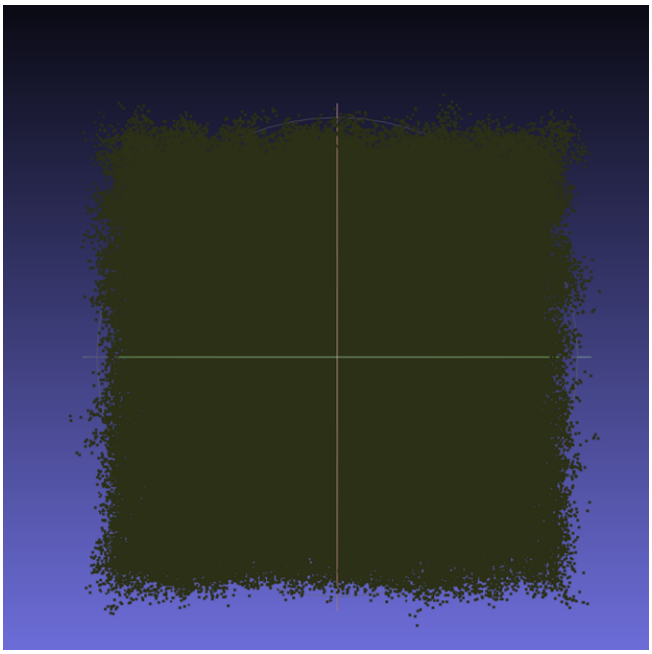


Figure 6.6: Example of a by NDF reconstructed chair after the whole training duration.

Earlier experiments with fewer epochs showed that NDF was able to produce reasonable encodings as can be seen in figure 6.2, which led to the hypothesis that the training took too long. An additional trial was conducted using 30 hours of training with one GPU, which corresponds to 5 epochs. This configuration was chosen

because it worked in the earlier experiment. The reconstructed chairs have a high similarity to the ground truth scans. The method is able to correctly learn flat areas and can learn thin parts like the legs of chairs. A characteristic error of NDF, which can be seen in figure 6.8, is having noisy points forming lines near the shape.

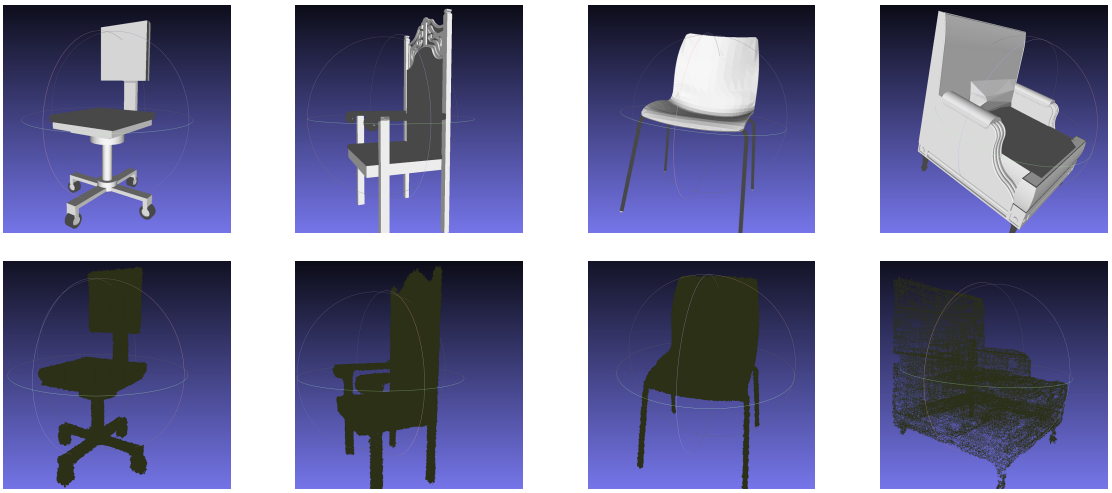


Figure 6.7: Examples of chairs that have been reconstructed by NDF. The top images represent the ground truth and the bottom the reconstructions. For the reconstruction of the rightmost chair, the size of the points needed to be shrunk, because no details would have been visible otherwise.

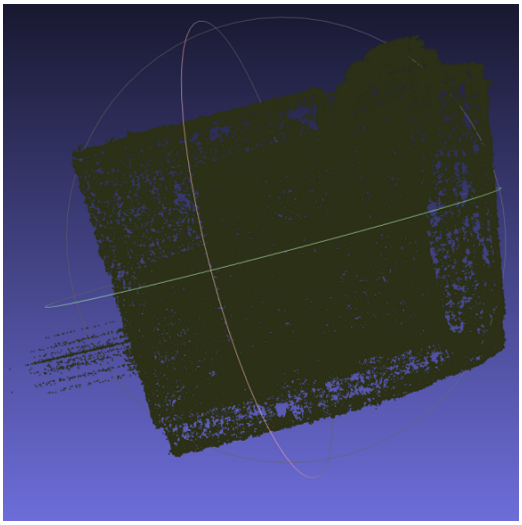


Figure 6.8: Example of a by NDF reconstructed chair with an error on the bottom left.

6.2.4 Chairs and Tables

In the experiments from chapter 5, IM-NET and NDF have also been trained on tables and chairs together. The results of this endeavor are described in this subsection. It is important to note when looking at the results, that the methods have not been designed to learn two classes.

IM-NET As for qualitative results, figure 6.9 shows some reconstructed shapes. The chairs were often still discernible, but most of the tables we have looked at either resembled chairs too, or nothing at all. Figure 6.10 shows one example of a table whose reconstruction looks like a chair which has been discovered through the visualization tool. Similar to the IM-NET, which has only been trained on chairs, the shapes do not have sharp edges and it struggles to learn thin parts like the legs of chairs and tables. The quantitative results from table 6.2 show a slight improvement in learning chairs over the IM-NET that only got chairs as examples, indicating that learning more shapes at once can improve the results of individual shapes. break

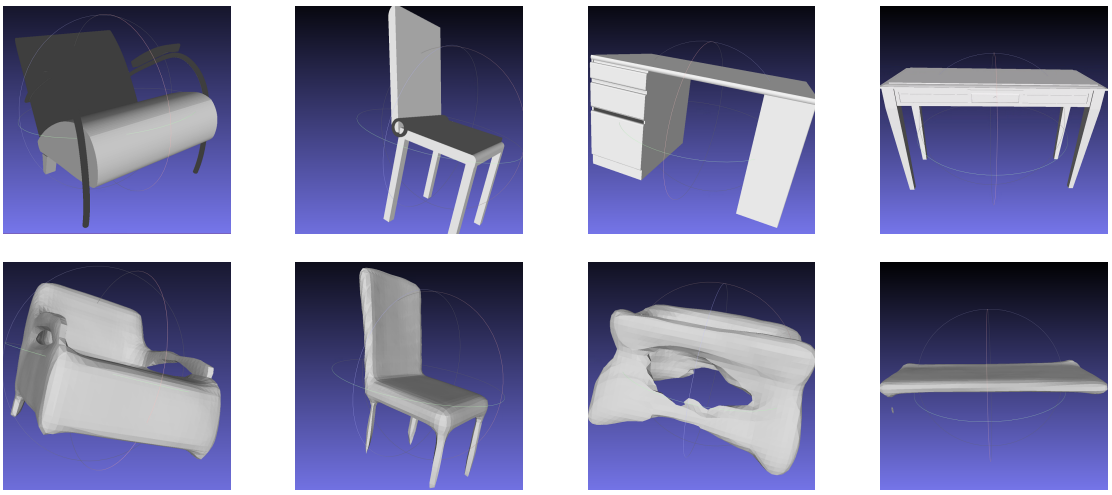


Figure 6.9: Examples of shapes that have been reconstructed by IM-NET. The top images represent the ground truth and the bottom the reconstructions.

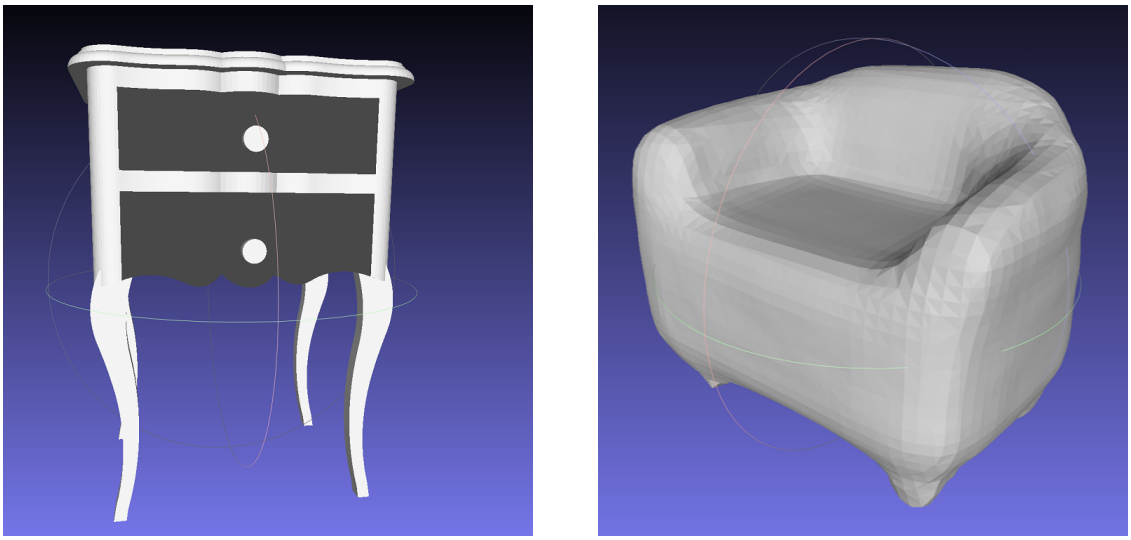


Figure 6.10: On the left, the original table is visualized and the reconstruction on the right looks like a chair.

NDF The multi-class experiments produced reconstructions of similar quality to 6.6. Unfortunately, no better results could be found, reconstructing from earlier checkpoints, but this might be due to not being able to find the optimal time to stop. It takes many hours to reconstruct the shapes from one checkpoint and there was not enough time left to find the optimal one. The chamfer distances of these experiments are not included in the table 6.2 because there is no point in showing the distances of all reconstructions look the same.

Chamfer Distance Overview			
Method	mean	median	standard deviation
IM-NET-Single Chairs	1.388522e-02	2.932810e-03	7.149892e-02
NDF-Single Chairs	3.190426e-03	2.071407e-03	3.541181e-03
IM-NET-Multi Chairs	1.318854e-02	2.644943e-03	6.983923e-02
IM-NET-Multi Tables	4.249876e-02	5.961613e-03	1.297119e-01

Table 6.2: Chamfer distance means, medians, and standard deviations of the different experiments.

6.3 RQ3: How did the SIREN extensions change the performance?

The resulting reconstructions of all the experiments look very similar. One example can be seen in figure 6.11. All reconstructions consist of a set of seemingly random points covering a cube. Not many insights about the different versions can be derived from the reconstructed shapes. The sampling used in NDF and its derivations do not seem to work on SIREN. One possible shortcoming is that the original SIREN gets off-surface coordinates, which are far away. NDFs' samples are generally relatively close to the shape. Also, the architecture and hyper-parameters might be fit for the original SIREN but not for the new sampling methods. Similar to the original SIREN, the reconstructions look wavy.

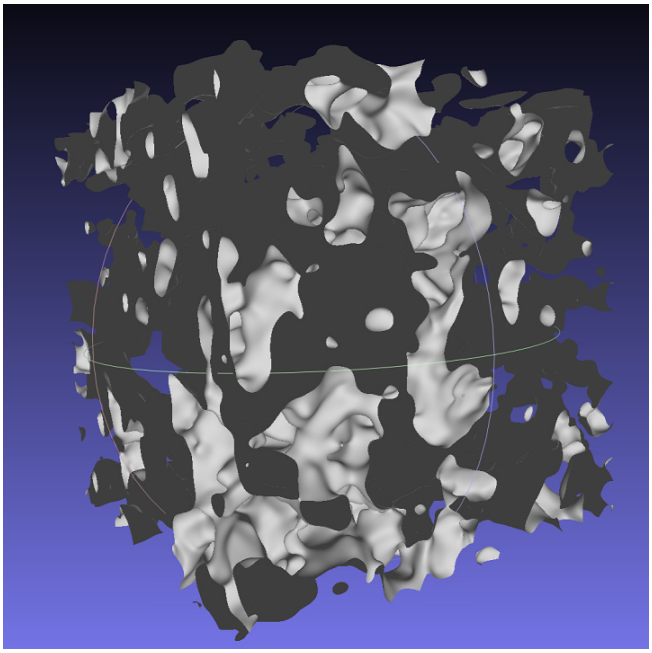


Figure 6.11: Example of a reconstruction attempt by an extension of SIREN

6.4 RQ4: How do the shape embeddings of the selected methods compare?

First, we can get SIREN out of the way by asserting that it does not have an encoder and subsequently no shape-vectors. It can not encode shapes and only learns to represent a single one. As the visualization is meant to be interacted with, static images will not represent all the results. Creating one image per selected point would waste too much space. It is recommended to interact with the visualization for more insights.

IM-NET Before NDF and IM-NET are compared, the differences within IM-NET need to be analyzed. The encodings of IM-NET at the end of each progressive training phase have been visualized in figure 6.12. One plot per dimensionality reduction has been created. T-SNE might not preserve as much of the variance as PCA. Selecting an embedding, which looks like an outlier in one shape, is not an outlier in the other subplots, as seen on the right side of figure 6.12. When using PCA, outliers on one subplot are often at the edge of the other subplots. Considering the embeddings are from the same method but taken from different points during training, they should look very similar. The differences have two possible conclusions, the encoder changes the embeddings drastically, even late into training, or the dimensionality reductions are not consistent enough.

Similar to the example with only chairs, PCA performs better than T-SNE to consistently position outliers farther away from the other points when visualizing chair and table embeddings. This can be seen in figure 6.13. A trend can be seen in the PCA plot, where a cluster is present, mainly containing chairs and most of the points that are far away from the cluster are tables. This supports the intuition that an encoder separates chairs from tables. The tables that are very close to the chair cluster are reconstructed as chairs as seen in section 6.2, which means the visualization is consistent with the qualitative results. Since the reconstructions are very bad when learning two classes of shapes, as seen in 6.2. it can be assumed that the embeddings are also sub-optimal. With a better method for reconstructing multiple classes, the embeddings might be separated more clearly, but this is only a hypothesis.

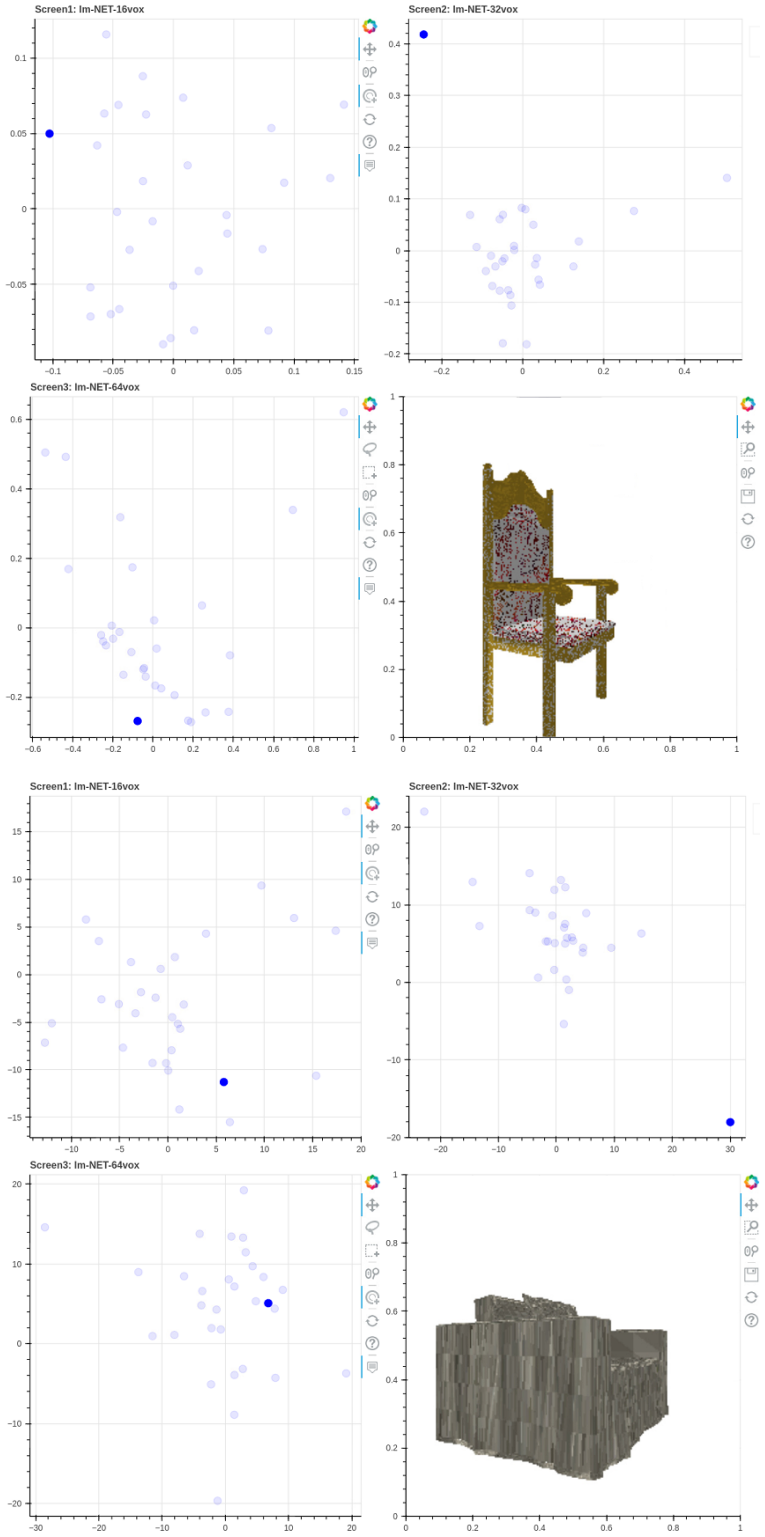


Figure 6.12: Visualization of the chair-embeddings of IM-NET. For the top figure(two rows on the top), T-SNE has been used, and PCA for the bottom figure. For both figures, the top left represents the embeddings of the 16 voxel-, the top right the 32 voxel- and the bottom left the 64 voxel training. The bottom right shows a rendering of the selected shape.

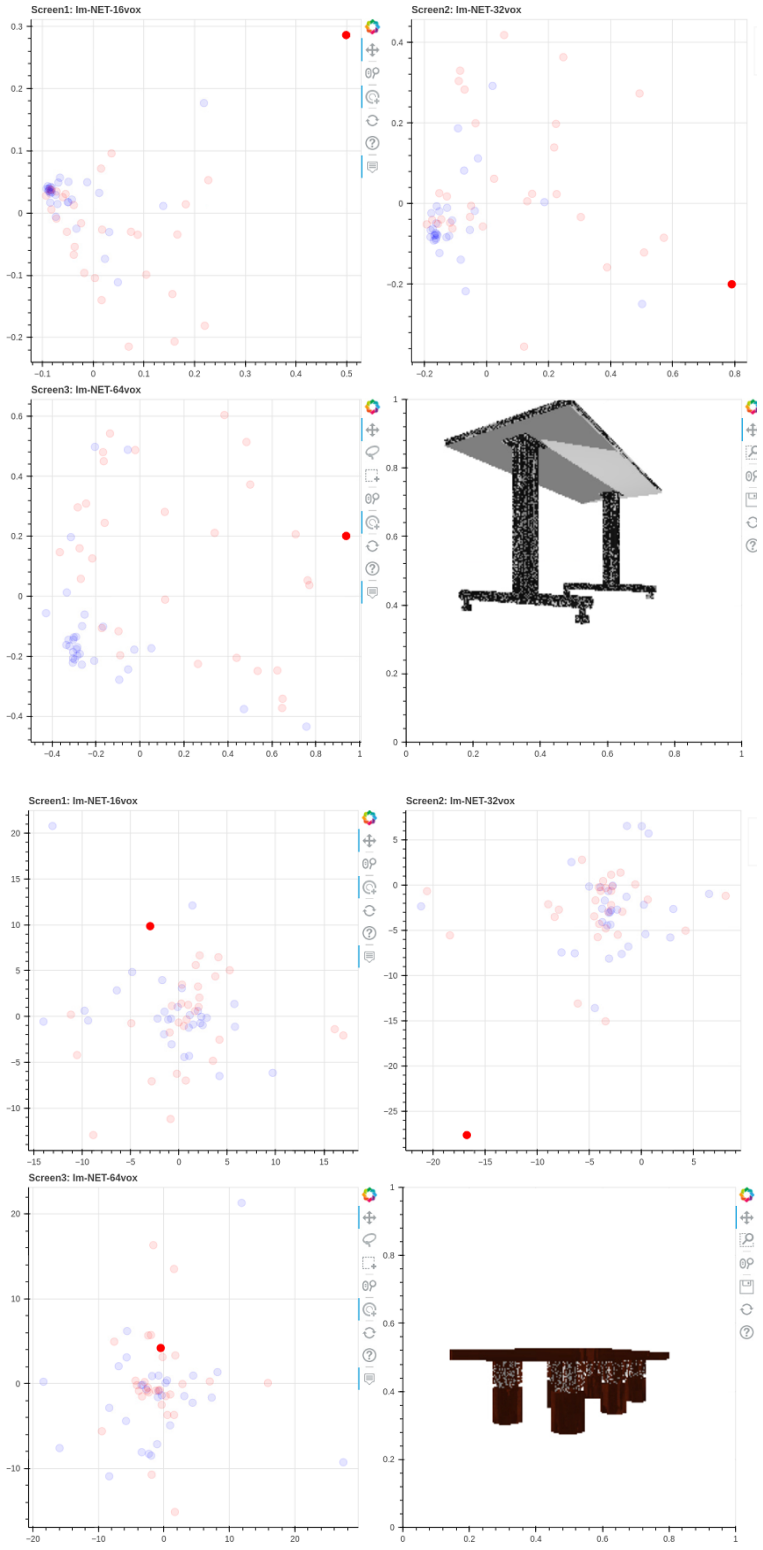


Figure 6.13: Visualization of the chair- and table-embeddings of IM-NET. For the top figure (two rows on the top), T-SNE has been used, and PCA for the bottom one. For both figures, the top left represents the embeddings of the 16 voxel-, the top right the 32 voxel- and the bottom left the 64 voxel training. The bottom right shows a rendering of the selected shape. The red dots represent chairs and the blue ones are tables.

NDF vs IM-NET Now the embeddings of IM-NET are compared to NDFs embeddings. The fully trained IM-NET is used for the comparisons.

The embeddings of the two different methods seem to be very different from each other. A point without close neighbors in one method is often in the middle of a group in the other method. This makes sense because the embeddings themselves are very different. IM-NET has one vector which describes the whole shape, and NDF is designed to use local features. Comparing the multi-class embeddings does not make sense, because the embeddings of NDF are not useful if the reconstructions are not working. Figure 6.15 shows the embeddings of NDF are mostly concentrated in one point, which explains why NDF did not perform.

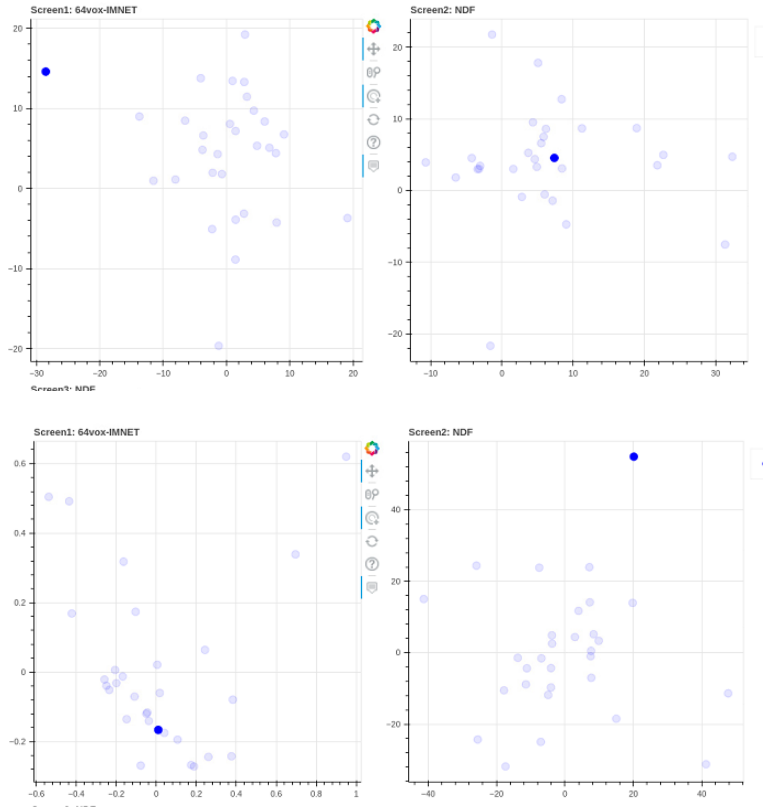


Figure 6.14: Visualization of the chair-embeddings of IM-NET and NDF. For the top row, T-SNE has been used, and PCA for the bottom one. For both figures, the left represents the 64 voxel-training of IM-NET and the right NDF.

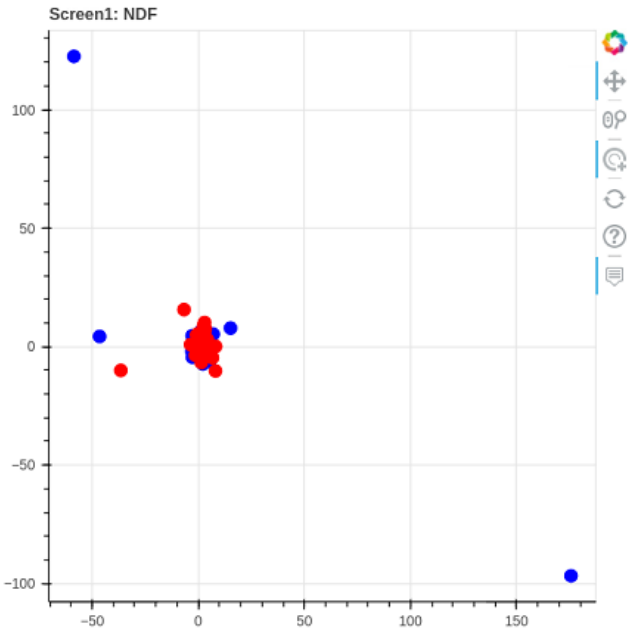


Figure 6.15: Multi-class embeddings produced by NDF.

7 Discussion and Conclusion

3D reconstruction is still an open problem in computer graphics. This thesis provided an overview of the notion of using implicit representations to solve this problem. To gain insight into how methods using implicit representations work three methods have been selected and theoretically compared based on picked areas. A case can be made for most other papers in this area to be picked instead of the ones that have been analyzed, but the number of papers was fixed and the selection represents the biggest variance that could be found. The input was fixed to be ShapeNet data and the output has to be a 3D model. Also, including 2D inputs or outputs would make the methods so far apart from each other that an analysis would lose most of its value. Additional or different areas could have been chosen, which changes insights from the analysis. To the best of our knowledge the general idea, architecture, use of data, and the loss are the only areas with enough significance and sufficient variance between the methods to provide valuable insight. An additional focus area could for example be optimization, but the focus of the papers was not to work on the problem of improving the optimization used. For all three methods gradient descent was used and the Adam optimizer was chosen in all the papers [Sit+20; CLW21; Sit+22].

SIREN is not comparable to the other two methods in terms of evaluating the outputs. The other two methods are evaluated on unseen shapes, while SIREN only has to overfit to one example, which is much easier. But even with one shape, some qualities of SIREN can be seen. Due to its very unusual preprocessing, the quality of the output highly depends on what kind of shape the model is given to learn. The results of the IM-NET experiments are not as good as what is being presented in the paper [CZ19], despite that the same training procedure and code were used to conduct the experiments as in the original. Because different data was used than the GitHub provided, it had to be pre-processed, but the pre-processing was as close to the original as possible. The large performance gap could partially be explained by randomness and different hardware, but something has to have gone wrong somewhere. Nothing specific could be found by inspecting the steps carefully. It could not be found out, why IM-NET performed worse on tables than on chairs. The amount of data was evenly distributed such that no class should be significantly more in the training set. NDF seems to be very fragile as there is a big variance in the quality of the reconstructions. The correct stopping point needs to be found, which maximizes the quality of the output. Unfortunately, for the experiments with chairs and tables, no good point has been found in the limited amount of time. Despite that, we believe the method has the capabilities to deliver high-grade reconstructions if the correct stopping point can be found. For reconstructing a single class, NDF is superior to IM-NET quantitatively as well as qualitatively. This is no surprise as NDF is a newer method than its competitor. The drawbacks of NDF are also not insignificant as it is much harder to train and the pre- and post-processing takes much more time than IM-NET on comparable hardware and the reconstructions use up

more memory.

The reconstructed shapes needed to be re-scaled and the factors are not obtained very scientifically in the case of IM-NET. This is very problematic, but the published codes from the papers did not include ways to measure the quality of the reconstruction. This makes it hard to compare results qualitatively and reproduce the numbers from the papers. A different idea was to scale both the ground truth and the reconstructions to be in a specific range, but this would cause even more problems. In figure 6.9 a reconstruction of a table can be seen where the legs are missing. Scaling the points of the output to be in a specific range would make this flat surface become something resembling a cube, which makes the quantitative less reliable. It was decided against using an algorithmic solution to find the correct scaling/shifting factor, because of the potential time complexity, and changing the values until the chamfer distance is minimized could give a method an unfair advantage. In general, methods in the area of "3D reconstruction with implicit representations" was perceived to be much harder to evaluate, compare, and reproduce than previously encountered areas of machine learning. A welcome change would be if more papers included the code with which they got their quantitative results in their GitHub directories. A positive example in this regard represents [BKG22].

It is very apparent that the struggles to improve SIREN have not been successful and there is no big variance between the different trials. It turned out to be very hard to work with. It can be the case that some parameter was very wrong and SIREN would have worked when for example using different standard deviations when sampling because the points of the original SIREN can be very far away from the surface. The reconstructions do not look remotely similar to a chair which makes us believe there is a bigger problem than this. It could also be the case that some coding error slipped into a part of the code, but visualizing the samples showed that the close points looked like a chair, which makes us believe the sampling should be good enough to learn something resembling a chair. One possible shortcoming of the new sampling is that it does not generate many points that are far away from the shape, whereas the original SIREN gets coordinates from the entire space. NDF seems to produce similar cubes to what the changed SIREN created. This might highlight that the sampling is not very suitable for implicit representations as the results vary a lot or that not the correct stopping point has been found for SIREN. It has been decided against adding additional changes, like adding more layers, or using more layers per layer as the selected hyperparameters have been shown to be enough to learn a single shape.

When it comes to the analysis of the embeddings, the visualization only produces circumstantial evidence of how the different shapes are encoded. Only using double-digit amounts of shapes for the visualization is not statistically relevant enough to draw any conclusions. The visualization is meant to be a tool to gain an intuition of what the embeddings might look like. It was decided to not use more shapes because too many points would make the visualization very hard to read. Even with this number of points, it is already hard to select single points where the embeddings are very close together, so showing more shapes would make it very strenuous to use the tool. That being said, nothing prevents the user from encoding many more points. There might be problems when applying the dimensionality reduction techniques with many shapes as the embeddings are

sometimes very big, for example for NDF, so significant resources are required to reduce the dimensionality. Visualizing embeddings is more useful if the input consists of several classes rather than just one. When analyzing a single class, only insights about which shapes might be similar or very different can be gained. The multi-class analysis helps find candidate shapes that cause problems for the decoder and provides an intuition about how the classes are separated in space, which can have some potential uses as alluded to in section 7.2.

A question that arises while researching SIREN is if it should be considered to be an SDF. It uses the normal constraints from SDFs but does not learn distances as has been pointed out several times in this thesis. For lack of a better term and because the authors intended it to approximate an SDF, it is considered one.

7.1 Limitations

The memory requirements when dealing with 3D data is a significant limiting factor inhibiting the methods from learning larger shapes. This limitation leads to the methods often sticking to smaller shapes. Learning multiple shapes or scenes has been done before, for example in [Cho+21]. In such settings, however, memory still limits the resolution or size of the output. Also, very thin areas, like the legs of chairs, can cause problems, as can be observed in section 6 and has been described in [BKG22].

The quality of the reconstructions highly depends on the quality of the input data. Having low-resolution noisy scans can make the models learn incorrect things, and even if the data is good and abundant enough for a task, the preprocessing and sampling might not hold up to the task.

With the multi-class experiments, the same training procedure was used for IM-NET as for the one-class trials. The reconstructions might be better if the number of epochs was changed to fit the new task. The loss curves in section 5.2.1 looked very similar, indicating that using the same number of epochs does similarly well in both cases, which suggests the procedure is also valid for two classes.

Regarding the visualization with both chairs and tables, chapter 6 shows that the reconstructions when using both classes were of very low quality. Because of these bad reconstructions, it can be assumed that the embeddings are also not very plausible. A better method might separate the two classes in space, which is not happening in the selected methods. Also, the usual drawbacks of dimensionality reductions apply; the variance that is still present in two-dimensional space might not be enough to gain insights into the embeddings.

In general, it can be said that, while these methods have made immense progress, they do seem to be far away from being used in a commercial setting. It takes a lot of time and hardware to learn implicit functions, and reconstructing shapes from the functions is more complex than just using a classical representation.

7.2 Future work

With computer hardware, especially GPUs and TPUs becoming more and more powerful, this opens the door to using bigger, deeper models and allows using more data, which is getting more prevalent. In addition, 3D scanners get more affordable, and 3D printers can potentially supply more data, too. This abundance of data and computational improvements make it almost inevitable that the methods using implicit representations for 3D reconstruction and other areas will improve.

But there is a lot of work to be done in preprocessing and using the data. As described in section 4.2 and 5.3, the sampling the selected papers use leave room for improvements. But as the experiments conducted suggest, it is not easy to create sampling and preprocessing that the models can use. This is where more focus should go, but designing a new architecture seems to be the more prestigious work.

More research needs to be conducted to improve the methods themselves, testing the capabilities of upcoming architectures like transformers and graph neural networks. Moreover, research has mostly been focused on representing the data as a distance or occupancy function. There might be unexplored kinds of functions that give neural networks an easier time approximating them.

The quality of the results limited the analysis of the embeddings. Showing the shape-space of methods that can learn multiple classes with a good reconstruction quality should be explored. Despite the reconstructions for the multi-class experiments not being of satisfying quality, the visualization tool with PCA was able to find the tables which get reconstructed as chairs. This finding might be used to create a tool to search for possible outliers by checking which shape is in the wrong neighborhood. Also, training procedures could focus on better representing these outliers, possibly improving the reconstructions.

8 Acknowledgement

I would like give a big thanks to all people who provided me with technical or moral support. First my appreciation goes to Dr. Prof. Renato Pajarola, Lizeth F. Perez and Haiyan Yang from the University of Zurich, who advised me and supplied the needed infrastructure for the experiments. They made this scary but interesting project possible. Also, thanks go out for my family and friends, which provided mental support as well as provided useful feedback.

Bibliography

[Ach+18] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. “Learning representations and generative models for 3d point clouds”. In: *International conference on machine learning*. PMLR. 2018, pp. 40–49.

[AL20] Matan Atzmon and Yaron Lipman. “Sal: Sign agnostic learning of shapes from raw data”. In: *Proceedings IEEE Computer Vision and Pattern Recognition*. 2020, pp. 2565–2574.

[AL21] Matan Atzmon and Yaron Lipman. “SALD: Sign Agnostic Learning with Derivatives”. In: *International Conference on Learning Representations*. 2021.

[Ale99] Marc Alexa. “Merging polyhedral shapes with scattered features”. In: *Proceedings IEEE Shape Modeling and Applications*. 1999, pp. 202–210.

[BKG22] Yizhak Ben-Shabat, Chamin Hewa Koneputugodage, and Stephen Gould. “DiGS: Divergence guided shape implicit neural representation for unoriented point clouds”. In: *Proceedings IEEE Computer Vision and Pattern Recognition*. 2022, pp. 19323–19332.

[BV99] Volker Blanz and Thomas Vetter. “A morphable model for the synthesis of 3D faces”. In: *Proceedings ACM Computer graphics and interactive techniques*. 1999, pp. 187–194.

[Byl+13] Erik Bylow, Jürgen Sturm, Christian Kerl, Fredrik Kahl, and Daniel Cremers. “Real-time camera tracking and 3D reconstruction using signed distance functions.” In: *Robotics: Science and Systems*. Vol. 2. 2013, p. 2.

[Cha+15] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. “ShapeNet: An Information-Rich 3D Model Repository”. In: (2015).

[Cha+21] Eric Chan, Marco Monteiro, Petr Kellnhofer, Jiajun Wu, and Gordon Wetzstein. “pi-GAN: Periodic Implicit Generative Adversarial Networks for 3D-Aware Image Synthesis”. In: *Proceedings IEEE Computer Vision and Pattern Recognition*. 2021.

[Che22] Zhiqin Chen. *implicit-decoder*. original-date: 2018-12-16T00:43:17Z. June 2022. URL: `https://github.com/czq142857/implicit-decoder` (visited on 06/19/2022).

[Cho+21] Jaesung Choe, Sunghoon Im, Francois Rameau, Minjun Kang, and In So Kweon. “Volumefusion: Deep depth fusion for 3d scene reconstruction”. In: *Proceedings IEEE Computer Vision*. 2021, pp. 16086–16095.

[CLW21] Yinbo Chen, Sifei Liu, and Xiaolong Wang. "Learning continuous image representation with local implicit image function". In: *Proceedings IEEE Computer Vision and Pattern Recognition*. 2021, pp. 8628–8638.

[CMP20] Julian Chibane, Aymen Mir, and Gerard Pons-Moll. "Neural Unsigned Distance Fields for Implicit Function Learning". In: *Advances in Neural Information Processing Systems*. Dec. 2020.

[CMP21] Julian Chibane, Aymen Mir, and Gerard Pons-Moll. *Neural Unsigned Distance Fields*. 2021. URL: https://github.com/jchibane/ndf.

[CZ19] Zhiqin Chen and Hao Zhang. "Learning Implicit Fields for Generative Shape Modeling". In: *Proceedings IEEE Computer Vision and Pattern Recognition* (2019).

[CZ21] Zhiqin Chen and Hao Zhang. *IM-NET-pytorch*. Publication Title: GitHub repository Published: (https://github.com/czq142857/IM-NET-pytorch. 2021. URL: https://github.com/czq142857/IM-NET-pytorch.

[Der17] Arden Dertat. *Applied Deep Learning - Part 3: Autoencoders*. en. Oct. 2017. URL: https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798 (visited on 06/20/2022).

[Dup+21] Emilien Dupont, Adam Golinski, Milad Alizadeh, Yee Whye Teh, and Arnaud Doucet. "COIN: Compression with Implicit Neural representations". In: *International Conference on Learning Representations*. 2021.

[FSG17] Haoqiang Fan, Hao Su, and Leonidas J Guibas. "A point set generation network for 3d object reconstruction from a single image". In: *Proceedings IEEE Computer Vision and Pattern Recognition*. 2017, pp. 605–613.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[Gir+16] Rohit. Girdhar, David F. Fouhey, Mikel Rodriguez, and Abhinav Gupta. "Learning a Predictable and Generative Vector Representation for Objects". In: *ECCV*. 2016.

[GKX21] Shihao Gu, Bryan Kelly, and Dacheng Xiu. "Autoencoder asset pricing models". In: *Journal of Econometrics* 222.1 (2021). Publisher: Elsevier, pp. 429–450.

[Got+03] Craig Gotsman, Xianfeng Gu, Alia Sheffer, and Sphehc Rabbis. "Fundamentals of Spherical Parameterization for 3D Meshes". In: *ACM Transactions on Graphics* 22 (Apr. 2003). DOI: 10.1145/882262.882276.

[Gu+04] Xianfeng Gu, Yalin Wang, T.F. Chan, P.M. Thompson, and Shing-Tung Yau. "Genus zero surface conformal mapping and its application to brain surface mapping". In: *IEEE Transactions on Medical Imaging* 23.8 (Aug. 2004). IEEE Medical Imaging, pp. 949–958. ISSN: 1558-254X.

[HLB19] Xian-Feng Han, Hamid Laga, and Mohammed Bennamoun. "Image-based 3D object reconstruction: State-of-the-art and trends in the deep learning era". In: *IEEE Pattern Analysis and Machine Intelligence* 43.5 (2019), pp. 1578–1604.

[HTM17] Christian Häne, Shubham Tulsiani, and Jitendra Malik. "Hierarchical surface prediction for 3d object reconstruction". In: *Proceedings IEEE Computer Vision*. 2017, pp. 412–420.

[Hua+01] Jian Huang, Yan Li, Roger Crawfis, Shao-Chiung Lu, and Shuh-Yuan Liou. "A complete distance field representation". In: *Proceedings IEEE Visualization*. 2001, pp. 247–561.

[Hua+20] Zeng Huang, Yuanlu Xu, Christoph Lassner, Hao Li, and Tony Tung. "Arch: Animatable reconstruction of clothed humans". In: *Proceedings IEEE Computer Vision and Pattern Recognition*. 2020, pp. 3093–3102.

[JA21] Wonbong Jang and Lourdes Agapito. "Codenerf: Disentangled neural radiance fields for object categories". In: *Proceedings IEEE Computer Vision*. 2021, pp. 12949–12958.

[JBS06] Mark Jones, Andreas Bærentzen, and Milos Sramek. "3D distance fields: A survey of techniques and applications". In: *IEEE Visualization and Computer Graphics* 12 (Aug. 2006), pp. 581–99.

[Jia+20] Yue Jiang, Dantong Ji, Zhizhong Han, and Matthias Zwicker. "Sdfdiff: Differentiable rendering of signed distance fields for 3d shape optimization". In: *Proceedings IEEE Computer Vision and Pattern Recognition*. 2020, pp. 1251–1261.

[Kam+17] Konstantinos Kamnitsas, Christian Ledig, Virginia FJ Newcombe, Joanna P. Simpson, Andrew D. Kane, David K. Menon, Daniel Rueckert, and Ben Glocker. "Efficient multi-scale 3D CNN with fully connected CRF for accurate brain lesion segmentation". In: *Medical image analysis* 36 (2017), pp. 61–78.

[LeC87] Yann LeCun. "PhD thesis: Modeles connexionnistes de l'apprentissage (connectionist learning models)". In: (1987). Publisher: Universite P. et M. Curie (Paris 6).

[Li+17] Jun Li, Kai Xu, Siddhartha Chaudhuri, Ersin Yumer, Hao Zhang, and Leonidas Guibas. "Grass: Generative recursive autoencoders for shape structures". In: *ACM Transactions on Graphics* 36.4 (2017), pp. 1–14.

[Lio+14] Cheng-Yuan Liou, Wei-Chen Cheng, Jiun-Wei Liou, and Daw-Ran Liou. "Autoencoder for words". In: *Neurocomputing* 139 (2014). Publisher: Elsevier, pp. 84–96.

[Mes+19] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. "Occupancy Networks: Learning 3D Reconstruction in Function Space". In: *Proceedings IEEE Computer Vision and Pattern Recognition*. 2019.

[Mo+19] Kaichun Mo, Paul Guerrero, Li Yi, Hao Su, Peter Wonka, Niloy J. Mitra, and Leonidas J. Guibas. "StructureNet: hierarchical graph networks for 3D shape generation". In: *ACM Transactions on Graphics (TOG)* 38.6 (2019). Publisher: ACM New York, NY, USA, pp. 1–19.

[Nie+19] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. "Occupancy flow: 4d reconstruction by learning particle dynamics". In: *Proceedings IEEE Computer Vision*. 2019, pp. 5379–5389.

[Par+19] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. "DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation". In: *Proceedings IEEE Computer Vision and Pattern Recognition*. June 2019.

[Sit+20] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. "Implicit neural representations with periodic activation functions". In: vol. 33. 2020, pp. 7462–7473.

[Sit+22] Vincent Sitzmann, Julien N.P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. *Implicit Neural Representations with Periodic Activation Functions*. original-date: 2020-06-23T04:41:51Z. June 2022. URL: `https://github.com/vsitzmann/siren` (visited on 06/20/2022).

[SPR+07] Alla Sheffer, Emil Praun, Kenneth Rose, et al. "Mesh parameterization methods and their applications". In: *Foundations and Trends® in Computer Graphics and Vision* 2.2 (2007), pp. 105–171.

[Tan+21] Jiapeng Tang, Jiabao Lei, Dan Xu, Feiying Ma, Kui Jia, and Lei Zhang. "Sa-convonet: Sign-agnostic optimization of convolutional occupancy networks". In: *Proceedings IEEE Computer Vision*. 2021, pp. 6504–6513.

[Ton+21] Wu Tong, Pan Liang, Zhang Junzhe, WANG Tai, Liu Ziwei, and Lin Dahua. "Density-aware Chamfer Distance as a Comprehensive Metric for Point Cloud Completion". In: *Advances in Neural Information Processing Systems*. 2021.

[Vou+18] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. "Deep learning for computer vision: A brief review". In: *Computational intelligence and neuroscience* (2018). Publisher: Hindawi.

[Wan+18] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. "Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images". In: *ECCV*. 2018.

[Wu+16] Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. "Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling". In: vol. 29. 2016.

[Yi+17] Li Yi, Lin Shao, Manolis Savva, Haibin Huang, Yang Zhou, Qirui Wang, Benjamin Graham, Martin Engelcke, Roman Klokov, Victor Lempitsky, et al. "Large-scale 3d shape reconstruction and segmentation from shapenet core55". In: *arXiv preprint arXiv:1710.06104* (2017).

[ZDW21] Linqi Zhou, Yilun Du, and Jiajun Wu. "3d shape generation and completion through point-voxel diffusion". In: *Proceedings IEEE Computer Vision*. 2021, pp. 5826–5835.

[Zha+18] Junhai Zhai, Sufang Zhang, Junfen Chen, and Qiang He. "Autoencoder and its various variants". In: *IEEE Systems, Man, and Cybernetics*. 2018, pp. 415–419.

[Zha+21a] Cheng Zhang, Zhaopeng Cui, Yinda Zhang, Bing Zeng, Marc Pollefeys, and Shuaicheng Liu. "Holistic 3d scene understanding from a single image with implicit representation". In: *Proceedings IEEE Computer Vision and Pattern Recognition*. 2021, pp. 8833–8842.

[Zha+21b] Fang Zhao, Wenhao Wang, Shengcai Liao, and Ling Shao. "Learning anchored unsigned distance functions with gradient direction alignment for single-view garment reconstruction". In: *Proceedings IEEE Computer Vision*. 2021, pp. 12674–12683.

[Zuc+10] Matt Zucker, Andrew J. Bagnell, Christopher G. Atkeson, and James Kuffner. "An optimization approach to rough terrain locomotion". In: *IEEE Robotics and Automation*. 2010, pp. 3589–3595.

[ZYQ21] Jingyang Zhang, Yao Yao, and Long Quan. "Learning signed distance field for multi-view surface reconstruction". In: *Proceedings IEEE Computer Vision*. 2021, pp. 6525–6534.