



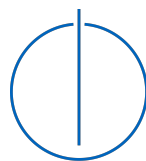
DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

**3D Shape Completion from Sparse Point
Clouds Using Deep Learning**

Christian Diller





DEPARTMENT OF INFORMATICS

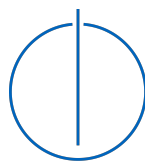
TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

3D Shape Completion from Sparse Point Clouds Using Deep Learning

Dreidimensionale Vervollständigung von Punktwolken mithilfe von Deep Learning

Author:	Christian Diller
Supervisor:	Prof. Dr. Matthias Nießner
Advisor:	Prof. Dr. Matthias Nießner
Submission Date:	15.07.2019



I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15.07.2019

Christian Diller

Acknowledgments

The personal acknowledgments have been removed for this public version.

Danke.
Thank you.
谢谢。

Christian Diller

Munich, Summer of 2019

Abstract

Over the last few years, the field of computer vision has evolved rapidly. New techniques like deep learning have revolutionized the way we approach problems in 2D, e.g. classification and detection on images. However, there are applications like autonomous robotics and 3D scanning that rely on a deeper understanding of the environment than 2D data is capable of providing. This is where three-dimensional understanding comes into play: For these applications to be successful, they have to use hardware that allows them to sense the space they operate in as well as software that can process this data and extract relevant information.

While handheld 3D scanning devices have gained more and more traction in the past few years, especially since the introduction of the Microsoft Kinect, there are applications that rely on accurate and precise measurements of the environment. This is what laser and LiDAR sensors offer: They generate a cloud of points where each point represents a precise measurement of the distance between observer and object surface.

While precision is important, point clouds have two major downsides: They often suffer from sparsity and incomplete geometry. However, in the field of content creation, dense representations are crucial to make virtual objects more realistic. Density is also useful for autonomous robotics: Being able to work with complete objects which were only partially visible to the 3D scanner can yield important information about the properties of certain shapes; this can then be used for more accurate decision making.

Thus, it is desirable to combine the advantage of point clouds with those of dense representations. We tackle exactly this problem of generating dense representations from sparse and partial point clouds. We achieve this with a data-driven approach which learns to complete incoming sets of 3D points in a fully-supervised manner.

To this end, we first prepare a suitable dataset containing partial scans, each one correlated with complete ground-truth representations. We base this on the widely used ModelNet40 dataset which features high-quality CAD scans of 40 different object categories.

Our deep learning architecture directly operates on point clouds; converting them into a different representation is not needed. This sets our method apart from approaches like 3D-EPN [13] that expect 3D data in a regular grid format.

We also study several different extensions and alternative architectures and provide extensive evaluation, also comparing our method directly to 3D-EPN. The results show that our method, while being easier to train, outperforms more involved approaches, giving us the ability to efficiently predict dense shapes from sparse and partial point clouds. Thus, we are able to bridge the gap between precise point cloud measurements and convenient dense representations.

Contents

Acknowledgments	iii
Abstract	v
1 Introduction	1
1.1 Utilizing the Third Dimension	1
1.2 Sparsity and Partialness	3
1.3 From Point Clouds to Dense Representations	4
1.4 Thesis Structure	4
2 Background	7
2.1 Three-Dimensional Data Representations	7
2.1.1 Regular Voxel Grids	7
2.1.2 Point Clouds	8
2.1.3 Polynomial Meshes	8
2.2 Direct Shape Completion	8
2.2.1 Energy Minimization with Shape Primitives	8
2.2.2 Database Priors	9
2.2.3 Symmetry Detection	9
2.3 Data-Driven Deep Learning Methods in 3D	9
2.3.1 3D Datasets	10
2.3.2 Operating on Regular Voxel Grids	10
2.3.3 Applying Convolutions to Point Clouds	11
2.3.4 Directly Operating on Point Clouds	11
2.3.5 Generative Models	12
2.3.6 Autoregressive Methods	12
2.3.7 Variational Autoencoders	12
2.3.8 Generative Adversarial Networks	13
3 Method Overview	15
4 Data Generation	17
4.1 Dataset Preparation	17
4.2 Partial Input Data	18
4.2.1 Virtual Camera Trajectory Sampling	18
4.2.2 Partial Input Point Cloud	19
4.2.3 Signed Distance Field	20

4.3	Complete Ground-Truth Data	21
4.3.1	Unsigned Distance Field	21
4.3.2	Ground-Truth Point Cloud	21
5	Network Architecture	23
5.1	Encoder	23
5.2	Decoder	24
5.3	Design Choices	24
5.3.1	Encoders	24
5.3.2	Classification	25
5.3.3	Point Cloud Generation	25
5.3.4	Hybrid Decoder	26
5.4	Loss Function	26
5.4.1	Smooth l_1 Loss for Distance Field Output	27
5.4.2	Distance Field Truncation and Logarithmic Scaling	28
5.4.3	Chamfer Loss for Point Cloud Generation	29
5.5	Training	29
6	Results	31
6.1	Quantitative Evaluation	31
6.2	Ablation Studies	32
6.2.1	Classification	32
6.2.2	Predicting Complete Point Clouds	32
6.2.3	Hybrid Decoder	33
6.3	Qualitative Samples	34
6.4	Limitations	34
7	Conclusion	39
	List of Figures	41
	List of Tables	43
	Bibliography	45

1 Introduction

The field of computer vision is evolving at a rapid pace. This has not only been driven by scientific advances, but also by improvements in computing and sensing hardware. The computing power we can harness today is significantly greater than that from a few years ago; also, cheaper, more accessible and more capable visual sensors such as high-quality photo and video cameras as well as 3D sensors have substantially influenced our ability to take on more complex problems. All of this led to some major breakthroughs, most recently with the general adoption of deep learning techniques which would not have been feasible a decade ago.

These developments first led to improvements in the domain of 2D image understanding. Techniques like classification, segmentation and detection all greatly benefitted from novel machine learning approaches. This has also sparked interest in new areas like autonomous robotics, augmented reality and content creation from real-world objects. It is obvious that an autonomous device which has to navigate in and interact with the real world needs to gain understanding of its surroundings; this is also true for augmented reality. And while content creation is still mostly done by hand, being able to directly import real geometry into the creation pipeline would make such work much easier.

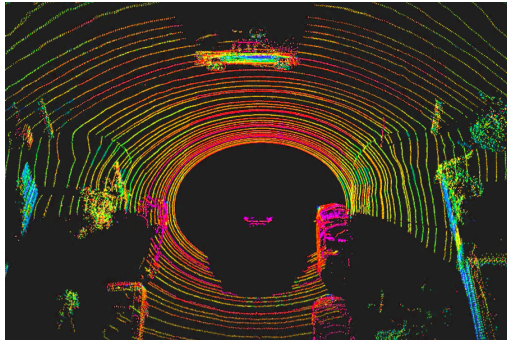
These applications cannot purely operate on 2D data; for actual and complete spatial understanding, vision methods must be able to sense and model the three-dimensional space.

1.1 Utilizing the Third Dimension

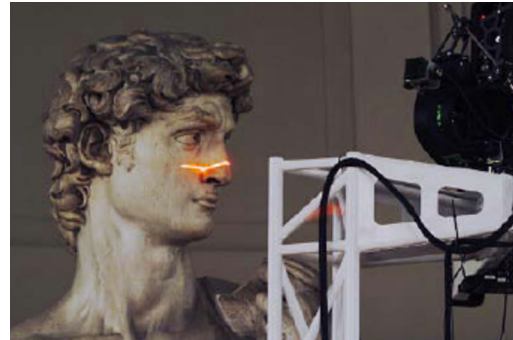
Methods working in 2D data only operate on a projection of the real world, losing important information about the spatial depth of the environment. This depth information, however, is crucial for accurately modeling the real world. Creating a virtual model of the real world has two important benefits: Being able to generate virtual replicas of objects and sensing the structure of the immediate environment.

Virtual objects nowadays are usually still created manually by artists who are designing them from scratch in an entirely virtual environment. Being able to scan an actual object and bridging that gap between bottom-up creation and real objects would not only make that process easier but could also help increase the quality of the resulting objects.

The second benefit is important especially for autonomous robotics: They have to be able to map their surroundings and localize themselves inside that map at any time. Working in 3D enables them to achieve this; moreover, it allows them to reason about



(a) LiDAR visualization with the car in the middle sensing its surroundings (from [6])



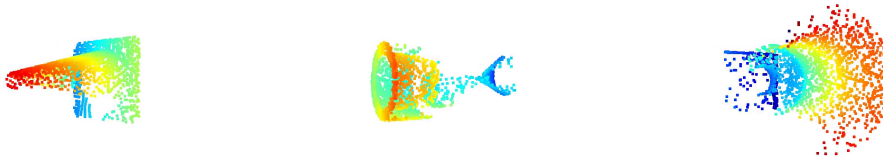
(b) The David statue being laser scanned as part of the Digital Michelangelo project (from [29])

Figure 1.1: How point clouds are generated: (a) LiDAR scanner enabling an autonomous car to precisely sense its environment and (b) the David statue being scanned in detail as part of the Digital Michelangelo project. Note the sparsity and partialness of objects in the LiDAR scan and the laser scanline moving across David's face

the relationships of objects in the vicinity. It is even possible to create arbitrary views of the surroundings, enabling decision making based on a bird's view. Beltran et al. [4] recently presented a method which makes use of this possibility.

Another, increasingly important, aspect is augmented reality. It allows for the placement of virtual objects in a real environment while also being able to interact with it. A complete spatial understanding is crucial here to allow for realistic interactions.

Today, there exists a multitude of 3D sensor hardware that produces representations of the real world for different applications. On one hand, there are relatively cheap and widely available handheld scanner systems. These either work by actively scanning the environment (using techniques like structured light or time-of-flight) or by passive sensing (using multi-view stereo). They are ideal for scanning entire scenes or creating low-resolution representations of real objects. A few sensors that should be named in this context are the Microsoft Kinect, the Intel RealSense, and the Google Tango project which has brought depth-sensing cameras into consumer mobile phones. Because of this general availability, they have enjoyed much attention from researchers over the last couple of years. One of the first works detailing the possibility of using them in a 3D reconstruction context was KinectFusion [42]. The general idea here is to fuse frames with depth measurement together into an internal representation which can then be used to generate the final reconstruction. Subsequent works addressed this work's limitations [74, 10, 8] and explored alternative methods of fusing depth frames [43]. More recently, BundleFusion [12] is a state-of-the-art method for reconstructing 3D objects and scenes that allows for real-time consistent and high-quality reconstructions. A comprehensive survey of available methods was published by Zollhöfer et al. [88].



(a) Only one side captured (b) Part of the foot missing (c) Very sparse, side missing

Figure 1.2: How the world is represented in point clouds: Every object is a set of measurements, each one representing a single point in 3D space. This leads to sparsity (space without measurements between points) and, depending on the location of the scanner, to partialness (object surface not uniformly covered with points)

On the other hand, there are high-end scanner systems that focus on measuring their environment with high precision. Examples for this category are laser scanners and LiDAR systems. Laser scanners can accurately measure distances from a sensor to the scanned object. They do this point-by-point which means that the resulting representation consists of a collection of highly-precise point measurements. One example for this is the work of Levoy et al. [30] where a scanline approach was used to digitally recreate statues in very high resolution. This is perfect for content creation: They enable highly accurate and precise scans of a single object. The other category of scanners are Light Detection and Ranging (LiDAR) sensors. They differ from laser scanners mostly in their use case: Instead of scanning one object, they are made to sense the environment and thus have to cover a much larger total surface area. Also, they usually use lower-frequency light to prevent interference with the environment. They are used for autonomous robotics and cars that need to perform real-time motion planning which requires knowledge of exact distances to obstacles. Figure 1.1 shows a visualization of LiDAR scan data as well as an image of the scanning approach used in the Digital Michelangelo project [30].

Both laser and LiDAR scanners have the format of their output data in common: They produce sparse point clouds where each point represents one measurement on an object's surface. This very common representation has some important advantages: First, it gives an accurate and precise reconstruction of the scanned environment as each point is a high-fidelity measurement. Second, as they are represented as a list of points, it is easy to display, filter, process, and store them.

1.2 Sparsity and Partialness

While point clouds offer great accuracy and precision in their point measurements, they are usually relatively sparse. This means that they do not contain any information about

the space in between measured points. The more surface area has to be covered (e.g. an autonomous car has to map several square meters around itself), the more space lies between any two points.

This deviates from the desired scan representation: We want to represent the entire surface of an object, not just a collection of some seemingly arbitrarily distributed points. This is obviously needed for content creation; for objects to look real, they need to have a closed, smooth surface. But it is also beneficial in the case of autonomous robotics: A closed surface mesh of an obstacle yields more information about its nature which helps a robot (or a self-driving car) to better decide which action it has to take next.

Besides the aspect of sparsity in a point cloud there is another issue to consider: It is quite hard to retrieve a complete scan of an object. While it is a matter of carefulness in a controlled environment such as a turntable laser scanner, it is much harder in the real world. In our example of self-driving cars, for instance, an object might only ever be seen from one side while driving by (as visualized in figure 1.1). This leads to only a partial scan being available in the resulting point cloud which is equivalent to a significant loss of information. Figure 1.2 visualizes this with objects from our dataset: Depending on the location of the 3D scanner, some of the geometry is missing.

1.3 From Point Clouds to Dense Representations

Dense representations like polygonal meshes are the preferred way to model virtual 3D objects. They approximate the geometry of an object in a dense manner, meaning that 3D information is available for the entire object surface.

Polygonal meshes have been used in areas like content creation and video gaming for quite some time now. With advances in recent years, they have allowed virtual objects to look almost perfectly realistic. Needless to say, it is imperative for a method importing real geometry into such areas to output dense meshes.

This can also benefit autonomous robotics applications: The more complete a reconstruction of its surroundings is, the more conclusions about the properties of objects contained in it can be drawn.

This is why there has been active research into this topic over the last several years. While there exist methods that try to directly reconstruct dense geometry by fitting primitives or exploiting inherent structures of 3D objects, the area of data-driven completion has gained traction recently. With the advent of viable machine learning algorithms, many new approaches have been introduced that strive to tackle the problem. Chapter 2 gives an overview over the current research landscape.

1.4 Thesis Structure

In this thesis, we present a method that tackles both sparsity and partialness in one pipeline. We devise a data-driven approach that directly consumes point clouds and

outputs a dense surface representation. For this, we roughly follow the idea of 3D-EPN [13]: We train an autoencoder-like network architecture with pairs of partial and corresponding complete objects in a fully supervised manner. This then gives the network the ability to complete unseen shapes at test time.

The rest of this thesis is structured as follows:

Chapter 2 gives an overview over existing approaches to sparse and partial data completion using different methods ranging from simple shape fitting to complex deep learning techniques. It also explores different 3D data representations and how they can be used in a deep learning context, emphasizing the difficulty of directly operating on point clouds.

Chapter 3 presents the main ideas behind our method and the techniques used to realize them. It summarizes the details of the following two chapters to give a high-level overview.

Chapter 4 explains how the data our method uses is created from an existing dataset. It details what kinds of data representations are needed, their properties, and how they are generated.

Chapter 5 presents all components that comprise the neural network architecture, the choice of loss function, the training process and some alternative design choices which were considered.

Chapter 6 shows and discusses the results of our method, compares the performance of different design choices and how it compares to an established completion method operating on volumetric grids.

Chapter 7 concludes this work and gives an outlook at possible improvements and work for the future.

2 Background

Three-dimensional data can be acquired by several different means, as has already been described in chapter 1. In this work, we focus on the post-processing step of the outputs of such methods, more specifically on the question of how to generate complete 3D meshes from sparse and partial point clouds.

In this chapter, we will give an overview of the field of 3D completion. First, the major types of 3D representations will be discussed which makes it easier to distinguish methods based on their input and output. We then give an overview over influential and recent methods which we categorize into two groups: Direct methods and data-driven methods. While most recent works rely on a data-driven approach, especially since the advent of deep learning, this categorization also takes solutions into account which operate directly on the input data without the need of a preceding learning phase.

2.1 Three-Dimensional Data Representations

There exist several different methods for representing 3D data; each one has specific properties that influence the way they can be used. The most widely used ones are regular voxel grids, polygonal meshes and point clouds.

2.1.1 Regular Voxel Grids

The idea behind voxel grids is the same as for 2D data: The Euclidian space is divided into a regular grid, each element of which is called voxel in 3D. This representation can be used in a multitude of ways, depending on which meaning is given to the values associated with each voxel.

The two most widely utilized voxel value interpretations are occupancy grids and implicit functions. Occupancy grids are a very simple solution: Each voxel indicates whether it is occupied by the 3D structure the grid holds or not, therefore acting as a piecewise constant function. In contrast, implicit functions are piecewise linear and encode the surface of a 3D structure as their zero-level set. Distance fields (see section 4.2 for more detail) are a very effective such function as they encode the distance to the closest surface in each voxel and are therefore able to store data with sub-voxel precision.

While regular grids are relatively simple and well-suited for deep learning methods (see section 2.3), they have some significant drawbacks. First, as they contain regularly-spaced voxels, there is a natural limit as to how big they can become before they are computationally too hard to handle. Furthermore, they are usually relatively coarse as

fine geometric details can only be represented when increasing the overall resolution. This means that shapes can usually not be stored in their original resolution, thus losing geometric detail.

2.1.2 Point Clouds

A much more lightweight alternative to the volumetric representations from above are point clouds. These are collections of points where each point encodes its own coordinates (x, y, z) in 3D space. The main property of 3D point clouds is the lack of an underlying structure; by definition, they are simply unordered collections of points irregularly distributed in 3D space.

This exact property makes them harder to deal with, especially in the context of convolutional neural networks. Furthermore, apart from lacking a smooth underlying structure they also often exhibit topological defects, such as holes.

So even though they are more lightweight and precise, dense surface reconstruction from them is a highly challenging problem that has received a lot of attention over the last few years.

2.1.3 Polynomial Meshes

Polygonal meshes are the representation we want to generate as an output of our completion pipeline. They describe a dense surface by approximating it with a set of polygons (usually triangles), connected to each other.

There exist established methods like Marching Cubes [35] which make it possible to use a regular grid (which is much easier to deal with in a deep learning context) as intermediate representation and to then extract a mesh from it.

2.2 Direct Shape Completion

We categorize methods that are able to complete 3D shapes from sparse and partial point clouds without needing to be trained on some dataset first as direct shape completion methods.

The survey of Berger et al. [5] gives an extensive overview over such methods; in general, we can classify existing direct solutions into the following classes.

2.2.1 Energy Minimization with Shape Primitives

These methods pose shape completion as an energy minimization problem: They use primitives and fit them onto the sparse point cloud, resulting in a mesh representation that approximates the shape underlying the point cloud.

One fundamental approach is Poisson Surface Reconstruction [22, 23]. Here, piecewise linear planes are fitted onto the surface of a point cloud, utilizing the point-to-plane

distance. Least-squares meshes [65] uses the point cloud as control points for a mesh approximation based on least squares optimization.

However, the quality of such methods is highly dependent on the input point cloud density. This not only means that the method produces low-quality results for very sparse point clouds but it is also not able to fill bigger holes in the input scan.

Operating on sub-optimally reconstructed meshes, there exist methods that are able to repair and fill holes [41, 86]; however, they are not able to deal with more serious defects that arise when a substantial part of the geometry is missing from the input point cloud.

2.2.2 Database Priors

An approach that is more capable in these extreme scenarios is to utilize a database with structural priors or 3D CAD models. This prior knowledge can then be used to fill in missing geometry or replace broken shapes altogether by retrieving and aligning appropriate structures from the database.

This method works well for completing shapes even from noisy 3D scans [32, 46]. To stay consistent with the input point cloud, aligned models from the database might also get deformed [56].

Recent advances in geometric feature matching have greatly benefitted this approach; this also makes it possible for them to not only operate on single 3D shapes, but also scale to whole scenes [24, 34, 40, 60].

Even though this is a relatively straight-forward solution to the problem, the major downside here is that generalization to new shapes is problematic as the underlying assumption is that the database always contains identical or at least very similar shapes.

2.2.3 Symmetry Detection

Another possibility that does not rely on a database is to leverage symmetries in 3D models. Many objects found in the real world are designed with symmetry and certain regularities in mind; exploiting this fact by reflecting, rotating and translating existing geometry leads to usable shape completions [69, 39, 47, 63, 66].

However, relying on only this idea leads to a similar problem as with database priors: The space of possible shape completions is limited by the choice of symmetry detection and completion algorithms.

It has also been shown that a combination of the direct methods discussed above and data-driven approaches for structural priors leads to usable results [68]. Purely data-driven approaches are explored in the next section.

2.3 Data-Driven Deep Learning Methods in 3D

As stated above, direct approaches yield good results but suffer from the problem of being constricted to underlying handcrafted criteria.

Data-driven methods promise to generalize better to unseen shapes. With the advent of deep learning methods in recent years many such solutions have been proposed.

2.3.1 3D Datasets

As is true for any data-driven method, the quality of the dataset used for training is one of the most important aspects of successful deep learning methods. For this reason, there have been several recent publications that propose new and purpose-specific datasets ready for use in deep learning.

For our use case of shape completion, the most prominent ones are ModelNet [77] and ShapeNet [7]. Both of these datasets contain several thousand high-quality 3D CAD models across different categories. As discussed in chapter 4, we make use of the ModelNet40 [77] dataset which we use for our partial and ground-truth data generation pipelines.

While deep neural networks operating on 2D data have now actively been the center of attention in the computer vision community, operating on 3D data is a relatively new area of research. This is not only due to the fact that 3D data representations are usually more computationally demanding but also to the fact that there exist several distinct representations of 3D data as discussed in section 2.1.

In the following, we give a quick overview over the methods used for each representation. The extension of 2D convolutions into the three-dimensional space with regular voxel grids is the most straight-forward approach; however, methods that are able to consume point clouds directly are more interesting for our method. We then also give an overview over current generative methods that can be used for shape completion, including variational autoencoders, autoregressive methods, and generative adversarial networks.

2.3.2 Operating on Regular Voxel Grids

Convolutional Neural Networks in 3D are a simple extension of the ones operating on 2D images: They make use of a convolution operator which gives them the ability to leverage spatially-local correlations in the input data.

While all of these approaches operate on regular grids, the specific representation they use as input varies widely. There are methods that work on occupancy grids generated from point clouds [37] or on distance fields [13]. Qi et al. [49] provide a comparison of volumetric CNNs with multi-view ones which operate on a set of 2D views. Another possibility is to use a regular voxel grid as a probability distribution of binary values on which a Convolution Deep Belief Networks is then trained [77].

The major disadvantage of regular grids is, as stated before, their inefficiency in storing 3D data as they represent important geometry such as surfaces the same way as non-important ones like empty space. One approach to tackle this problem is to use so-called octrees: They are able to partition the 3D space in a way that free space

requires less storage while important geometry can be represented in more detail [55, 71].

However, the most compact way of storing 3D information are point clouds. They are, however, not trivial to use with convolutional neural networks as they lack the regularity needed by convolutions.

2.3.3 Applying Convolutions to Point Clouds

One approach to solving this problem is to apply a custom operator to point clouds which then makes it possible again to apply 3D convolution.

This can be achieved by either using convolutional filters on point neighborhoods [28, 78] or by overlaying a regular grid and encoding points into voxels using custom descriptors [27, 87].

Other publications also introduce novel convolutional operators that can be applied to each point of a point cloud [18, 76].

Apart from that, ideas like converting a point cloud into a regular representation internally [54] or generating 2D representations of 3D data and then applying 2D convolutions [58] have also been explored.

While these methods already perform quite well on point clouds, they still assume an underlying regular representation.

2.3.4 Directly Operating on Point Clouds

Another direction are networks directly operating on point clouds. They consume the raw array of points without any assumptions about the underlying geometric space or influences by 3D convolutions.

One widely used approach is PointNet [48]. It generates a latent space vector from a set of unordered points while being invariant to the actual order of the points. While it is easy to use, its major issue is that it does not take into account local structures very well. This has been addressed in the subsequent work PointNet++ [50] which introduces a hierarchical model of PointNets. Other publications building on that original idea add graph-based enhancements to the encoder [81] or use custom kernels that take local geometric structures into account [62]. We discuss PointNet and how we use it in our method in more detail in chapter 5.

Further possibilities include modeling the spatial distribution of point clouds by aggregating features from sparsely distributed radial basis function kernels [9], dynamically generating internal graph representations of point clouds [73], or permuting the input point cloud into a canonical order [16, 33]. Several methods also exist that specifically focus on the task of completing point clouds [83, 82].

Recently, there has also been some work on lifting the need for a fixed number of input points that can be consumed [53] while also outputting a dynamic number of points.

2.3.5 Generative Models

While most approaches focus on more traditional tasks like classification and segmentation of 3D point clouds, the topic we focus on in this thesis is 3D shape completion, i.e. reconstructing a dense surface from sparse and partial point cloud data.

Several deep learning methods have been proposed to solve this problem; Achlioptas et al. [1] provide a study of different generative models. They all evolved around the general idea of an autoencoder which combines an encoder with a decoder and learns to reproduce its input. This means that it learns a latent representation of shapes. We highlight the work of Dai et al. [13] which uses this concept for shape completion: During training time, incomplete models are fed into the network which is trained to recreate the complete shape. Therefore, it intuitively learns how to fill in missing geometry, ultimately completing shapes. We follow this general idea in our work but instead of operating on a regular grid as input, we consume a sparse point cloud and generate a dense representation.

Other recent works focussing on completing 3D shapes from point clouds are able to operate on raw point clouds without any prior assumptions [83], use feature pyramids and grid deformation for reconstruction [36], and use graph-based enhancements in the encoder [81]. A more involved approach is the one of Wang et al. [70] which uses extensive pre-processing, parts segmentation, generation of curve skeletons, and user interaction for shape completion.

With these autoencoder methods already producing good reconstruction results, there has also been work recently on more specialized approaches.

2.3.6 Autoregressive Methods

Having first been applied for the generation of 2D images, autoregressive approaches have recently also been used to generate and complete 3D geometry. Van den Oord et al. proposed PixelRNN [45], a two-dimensional recurrent neural network that predicts individual pixel values dependent on previous pixels. Furthermore, they also introduced PixelCNN [44] which leverages a convolutional structure to be more efficient on images. Further improvements to this idea like parallelization [52] led to this method becoming more viable, even in three-dimensional space.

ScanComplete [14] presented an approach for 3D scan completion based on convolutional autoregression. It is not restricted to individual shapes but can take into account full 3D scenes and achieves impressive reconstruction results.

2.3.7 Variational Autoencoders

The original concept of training a latent vector using an encoder-decoder structure can be extended by adding a constraint which forces the latent vector to roughly follow a certain distribution [26]. For inference, one can then sample an element from this distribution to generate valid output.

For shape completion purposes, such an autoencoder can learn to reconstruct from noisy input data (it is then called denoising autoencoder) or with noise injected into the hidden layer. In [61], the authors learn a volumetric representation from noisy point clouds while [20] combines both noise in the input and the hidden layer. Another approach performs shape inpainting on local patches using a denoising autoencoder [59] which can faithfully repair moderately sized holes.

For real-world data, there often does not exist a ground truth for 3D shapes that have to be completed, making fully-supervised training impossible. Variational autoencoders have been shown to still be a viable option in these circumstances; they can be trained in a weakly supervised manner [67] or entirely unsupervised [85].

As these methods still follow the generic autoencoder approach; they are quite convenient in the sense that generated models can be directly compared to a ground truth. However, due to their formulation they tend to produce relatively blurry output.

2.3.8 Generative Adversarial Networks

Another approach that fixes the problem of blurriness are Generative Adversarial Networks (GANs) which have first been proposed by Goodfellow et al. [17]. Here, a deep neural network architecture is trained in an adversarial process: A generator tries to produce data that resembles the training dataset while a discriminator learns to decide whether an instance is real or generated. Over the last few years, several works have been published that improve upon the original formulation [15, 51], allowed for the generation of multi-modal models [38], and improved training speed and stability [57, 21].

More recently, GANs have also been used for 3D data generation and shape completion. One of the first works was 3D-GAN [75] which generates 3D shapes directly from a probabilistic space. In a subsequent work, Smith et al. [64] presented another approach to 3D GANs that improves the generation of shapes but more notably, also enables the reconstruction of full 3D objects from 2D images and 2.5D occluded range images. The idea of retrieving a full 3D shape from just one 2.5D depth image was also pursued by more recent works [80, 79].

There has also been some research on end-to-end adversarial autoencoders. Zamorski et al. [84] propose a solution that is able to produce completed point clouds from partial inputs. Also operating on point clouds are Point Cloud GAN [31] and the approach of Achlioptas et al. [1] which learn versatile latent representations of point sets.

However, as these methods are usually based on 3D convolutions, their output resolution is limited. Wang et al. tackle this problem by using a GAN for shape inpainting [72].

Also noteworthy is the solution of Kingkan et al. [25] who combine variational autoencoders with GANs while using a PointNet-like encoder operating directly on point clouds and generating surface meshes.

However, stable generation of 3D objects still has problems in training time and stability as well as output resolution. This is especially true if the underlying dataset

consists of many object classes and orientations.

For our method, we combine the ease of an autoencoder architecture with the shape completion idea of Dai et al. [13] and an encoder that can directly operate on sparse point clouds, based on PointNet [48].

3 Method Overview

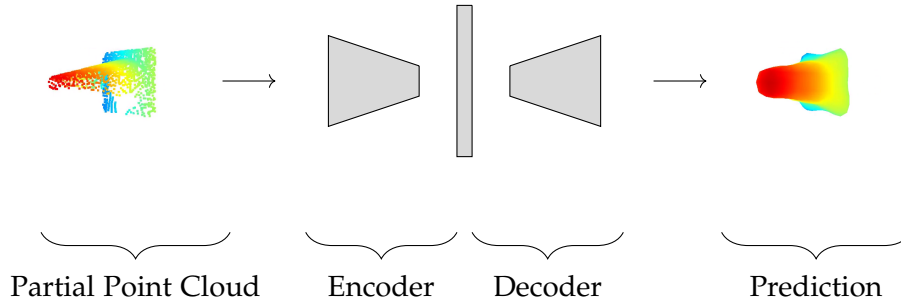


Figure 3.1: We train an autoencoder-like network with a sparse and partial point cloud as input and the corresponding complete distance field as target. The encoder directly consumes and compresses the point cloud into a latent space vector from which the decoder learns to generate a 3D distance field. At test time, the final mesh can then be extracted from the volumetric prediction

Our method aims at generating dense 3D surface representations from sparse input point clouds using a data-driven approach. Furthermore, our approach is able to reconstruct plausible geometry even with major parts missing from the input data. We achieve this by training an autoencoder-like structure that learns shape completions in a fully supervised manner: It is trained on pairs of partial point clouds and complete dense 3D models. The compression to a latent space vector enables it to learn how to reconstruct missing object parts.

We generate input and ground-truth data for training and testing by partially scanning high-quality 3D CAD models from the ModelNet40 dataset [77]. Input data is represented as unstructured sparse point clouds with a fixed number of points. The ground-truth data is a volumetric distance field containing 32^3 voxels. For each partial scan, a random trajectory consisting of virtual cameras which render the object from several different angles is created. The depth maps of all cameras are then fused to produce the final scan; chapter 4 goes into the implementational details.

Our approach is based on the 3D Encoder-Predictor Networks of Dai et al. [13]: An encoder learns to compress the input data into a latent representation. From there, a decoder predicts a completed volumetric representation which is learned to match the ground truth. In our case, however, we directly operate on unstructured point clouds using an encoder following the idea of PointNet [48] instead of ingesting a volumetric distance field.

Figure 3.1 visualizes the high-level architecture of our method. Like an autoencoder, it compresses the input and then decompresses the latent vector to retrieve the target. Unlike standard autoencoders, we do not simply recreate the original shape but instead learn a correlation of partial scans and their associated completed shape. The intuition behind this is that the network will not just learn to recreate entire shapes but eventually gain the ability to predict completions for missing parts of previously unseen objects.

The decoder expands the latent representation into a 3D distance field by using multiple transpose convolutional layers. The final completed mesh is generated by extracting the zero-level set via the Marching Cubes algorithm [35].

We compare the performance of our architecture, which is presented in detail in chapter 5, to the approach of Dai et al. [13] and conduct several design studies such as using a bigger PointNet, adding a classification pseudo-loss, and using a hybrid decoder to output both completed distance fields as well as point clouds.

4 Data Generation

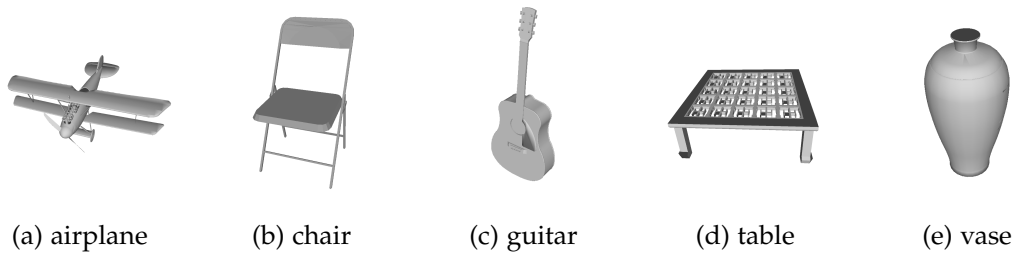


Figure 4.1: We generate our dataset based on the CAD models from ModelNet40. This dataset contains thousands of 3D objects, categorized into 40 classes, samples of 5 of which are shown above. Note the wide variety of objects our method has to be able to take into account: From the relatively simple geometry of a vase to more complex structures like an airplane and also very thin structures like the legs of a chair

Our approach to shape completion requires feeding 3D point clouds of partial scans of objects into a network architecture (which is discussed in detail in the next chapter). We train this network in a fully supervised manner which means that we first need to generate the partial input point clouds as well as the target volumetric representations and ground-truth point clouds. In this chapter, we discuss the process of generating a dataset that contains all these representations.

The commonly used and publicly available 3D shape dataset ModelNet40 [77] serves as the basis of our dataset generation process. It contains CAD models of 40 object categories which we use for both input and ground-truth data generation.

Fundamentally, we apply some transformations and data augmentation strategies first and then generate input data by virtually scanning all objects in the augmented dataset, resulting in partial point clouds and meshes (in the form of signed distance fields). We then generate ground-truth data by uniformly sampling the original CAD meshes in order to retrieve target point clouds and by using a 3D scanline approach [2] to generate target meshes (in the form of unsigned distance fields) used for training and evaluation.

4.1 Dataset Preparation

The ModelNet40 dataset was created as part of the Princeton ModelNet project [77] which aims at providing a comprehensive and clean collection of 3D CAD models

for the most common object categories. The models are collected from online sources and manually verified to be fitting and of high quality. Furthermore, the models are manually aligned to all have the same orientation.

For our method, we used the version of ModelNet with 40 classes (which is simply referred to as ModelNet40) as we want to generalize to as many object categories as possible. Figure 4.1 shows sample CAD models from five of the 40 overall object classes contained in the dataset. We use the original train/test split which results in a total of 9843 training CAD models and 2468 test CAD models across all classes. The ratio of training to test samples is therefore roughly 80:20. We keep this ratio throughout all our experiments.

All the shapes contained in ModelNet40 are stored as surface meshes with vertex locations and face definitions. Prior to any further processing, we normalize all vertex locations to lie within a unit cube which is centered at the origin while preserving the geometry of each model; the models are not warped or partially stretched in any way. The intention behind this normalization step is to prevent potential problems from differences in scaling or translation across models.

Before creating the actual data used as input and ground-truth in our method, we first generate some additional augmentations by randomly rotating each model. This results in 5 additional rotated models per original CAD model (we always keep the original rotation for each model) which gives us an overall initial augmentation factor of 6. In our experiments in chapter 6 we then evaluate the impact of using a varying number of rotational augmentations during training to the overall results.

We then use our normalized and augmented version of ModelNet40, still consisting of high-quality 3D CAD meshes, as the basis for our input and ground-truth data generation pipelines detailed below.

4.2 Partial Input Data

4.2.1 Virtual Camera Trajectory Sampling

The input data to our completion pipeline consists of partial scans of the objects in our prepared ModelNet40 dataset. To generate those, we first sample a virtual camera trajectory which simulates a depth sensor moving around the object. We subsequently then fuse the depth data generated by these virtual cameras into a point cloud as well as an implicit volumetric surface representation. While our method does not make use of this representation, generating it allows us to compare our method’s performance to the one of Dai et al. [13] which operates on such a regular grid representation.

A virtual camera trajectory consists of 7 camera centers which are sampled on a Great Circle of a sphere with radius 2 around each object. First, all centers are generated to lie on the equatorial circle with a fixed angle α between them ($\alpha = 30^\circ$ in our case). We generate 7 camera centers in each trajectory, meaning each trajectory forms a half-circle around the object. Then, we randomly generate 6 different rotation matrices and rigidly

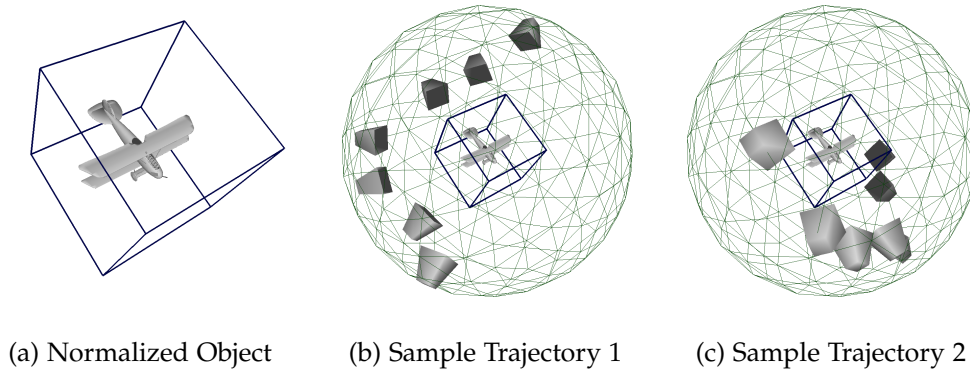


Figure 4.2: We prepare each CAD model in ModelNet40 by first normalizing it to a unit cube (a). We then create partial scans by sampling a virtual camera trajectory on a sphere around each object. Random translational noise is added to each camera center to simulate a more realistic trajectory. Figures (b) and (c) show two different, randomly generated camera trajectories. We then fuse the depth information from each camera to create our final, partial point cloud

rotate all centers within that initial trajectory. This gives us 6 different trajectories for each object; we use all of them in our training, augmenting our dataset by a factor of 6. The intuition behind this is that the network should learn to complete different parts of each object, making it less dependent on the trajectories used with a specific object. For our experiments, we only use three (the first three) camera centers of each trajectory as we found that that leads to the best tradeoff between partialness and performance.

The extrinsic parameters of each virtual camera are set in a way that each camera directly looks at the origin, i.e. the center of each object. These are the extrinsics later used when capturing depth maps. The intrinsic parameters are fixed to be equal to the Kinect intrinsics for each virtual camera.

In order for our virtual trajectories to resemble a real-world handheld scanning device more closely, some jitter is added which translates each virtual camera center randomly in x -, y - and z - direction with at most 10% of the sphere radius.

A graphical overview of this can be seen in figure 4.2. The camera centers, visualized by grey cones, lie on a sphere around the normalized object. This also shows the impact of adding jitter as the centers are not uniformly distributed on the sphere.

With the camera trajectories available, we generate our input data. Depth maps are collected by rendering the object with camera poses defined in the sampled trajectory. Those depth maps are then passed on to the fusion step.

4.2.2 Partial Input Point Cloud

First, a point cloud is generated by back-projecting the points from all depth maps into a global coordinate system using Kinect intrinsics. The resulting point cloud is then

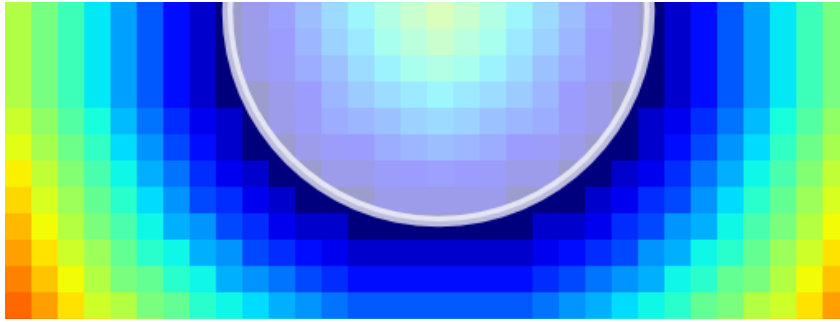


Figure 4.3: A 2D slice through a distance field of a sphere (in white). One pixel represents one voxel in the 32^3 volume, the color encodes its value: Close to the surface, it is small (blue) while it turns to green and finally red for voxels far away from the surface. This is the representation our decoder learns to generate. From there, we can then extract the final mesh using Marching Cubes

uniformly down-sampled such that it consists of exactly 2048 points. We make sure that clouds with less than 2048 points get re-generated with another trajectory to prevent degenerate point clouds in our dataset.

4.2.3 Signed Distance Field

We also generate a Signed Distance Field (SDF) with 32^3 voxels from the rendered depth maps. This representation encodes the distance to the closest surface in each voxel; figure 4.3 visualizes a slice through the SDF generated from a spherical mesh.

A signed distance field consists of voxels whose values encode the distance to the surface. Further away pixels have a higher value (red in figure 4.3), closer ones a lower one. They are also signed: Voxels outside of the object have a negative sign, the ones inside a positive one. The actual surface can therefore be found at the zero-crossing between voxels. Extraction methods like Marching Cubes [35] make use of this fact when generating actual surface meshes from these distance fields.

For the fusion of our depth maps into such an SDF, we integrate all depth maps into a shared volumetric grid using the fusion approach described by Curless and Levoy [11]. Our bounding box equals the unit cube we earlier normalized all models to. In doing so, we retrieve a Signed Distance Field with 32^3 voxel cells which will serve as input data to the method of Dai et al. [13] which we compare our method to.

Note how this approach to storing volumetric data provides information about the free space around and inside objects, in contrast to the raw point clouds we are using for our method. In theory, this should provide a network more information to learn from compared to only using sparse point clouds without information about the space in between them.

4.3 Complete Ground-Truth Data

As for input data, we also generate two different kinds of 3D representation for the ground-truth data. First, we uniformly sample points on the surface of each model to get a complete point cloud. Second, we generate an Unsigned Distance Field (UDF) as a target for mesh generation.

4.3.1 Unsigned Distance Field

We follow the argument of Dai et al. [13] for using unsigned distance fields as the target instead of signed ones: Determining which area is inside and which is outside of a CAD model is non-trivial as some of them might not be closed.

The distance values are generated using a 3D scanline approach [2]. The result is equal to the SDFs generated before: Each voxel represents the distance to its closest surface.

The generated volume has the same size as as the input SDFs, i.e. 32^3 voxels. At inference time, we extract surface meshes from this representation using Marching Cubes [35].

4.3.2 Ground-Truth Point Cloud

To retrieve the ground-truth point cloud, we uniformly sample points from each model surface. The original CAD models contain triangle faces of varying size: For example, the even surface of a table takes less but bigger triangles to model than the rounded surface of a vase. This means that we need to distribute more points on larger faces than on smaller ones in order to achieve a uniform distribution of points.

We want to distribute $N = 4096$ points of the surface of each model as we expect most models to only be visible from one side; therefore, a doubling of points should help the network fill in missing structures without having to also redistribute them. For each point, we choose a face on which to place it. Each face has a probability attached to it: It is the face's area divided by the total surface area of the mesh. This means that faces with more area get more points than those with less, achieving our goal of a uniform distribution.

With the face to place the point on chosen, we make use of barycentric coordinates to find the exact coordinates of the point on that surface. The formulation of barycentric coordinates gives $\mathbf{p} = u * \mathbf{a} + v * \mathbf{b} + (1 - (u + v)) * \mathbf{c}$ for a point \mathbf{p} in a triangular face spanned by points \mathbf{a} , \mathbf{b} , and \mathbf{c} . We choose u and v randomly from $[0, 1]$ such that $u + v \leq 1$ (if that is not the case, we can simply use $u = 1 - u$ and $v = 1 - v$).

As stated above, we add augmentations to the dataset by using 6 trajectories per object each of which is present in 6 different rotations. In the results (chapter 6), we evaluate how different numbers of rotational augmentations affect the overall performance. More specifically, we tested with 1 rotation (no augmentations), 3 rotations and 6 rotations.

rotations	1	2	3	4	5	6
train	59,058	118,116	177,174	236,232	295,290	354,348
test	14,808	29,616	44,424	59,232	74,040	88,848

Table 4.1: Exact number of train and test samples our generated dataset contains. We generate 6 sets with different numbers of rotational augmentations. For our experiments, we compare the performance when trained on 1, 3, and 6 rotations. The ratio between train and test samples is always roughly 80:20 as we keep the original split given by ModelNet40

The number of training and test samples after the data generation pipeline is detailed in table 4.1.

5 Network Architecture

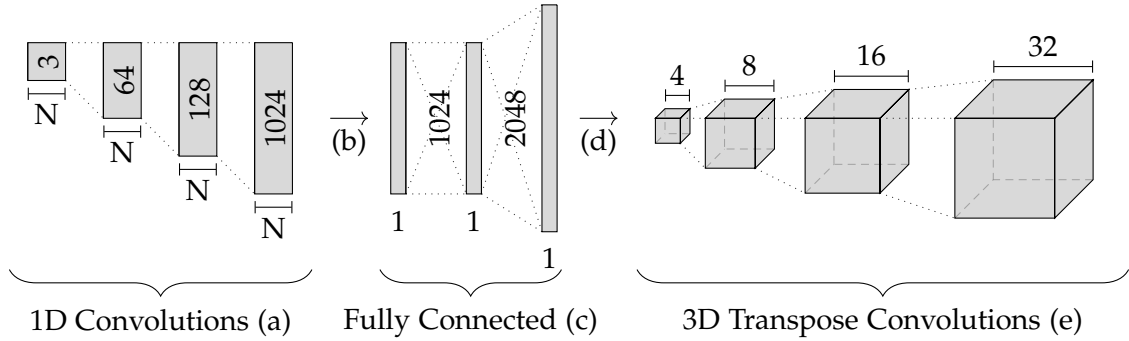


Figure 5.1: Our network architecture consumes a cloud of N 3D points and predicts an unsigned distance field. 1D convolutions (a) and max pooling (b) are used as an encoder (representing a simplified version of PointNet [48]). The decoder consists of two fully-connected layers (c) followed by a reshape operation (d) and several 3D transpose convolutions (e)

We propose a network architecture with an autoencoder-like structure consisting of an encoder operating on 3D point clouds and a decoder that outputs volumetric unsigned distance fields. Figure 5.1 depicts the overall structure. The inputs to the network are 3D point clouds of N points with 3 coordinates each. In our case, we chose $N = 2048$. In order to encode those points clouds, we use a set of 1D convolutions which resembles a simplified version of the PointNet [48] encoder and use max pooling to generate our latent space vector of size 1024. After two fully-connected layers, the vector is then reshaped and upscaled to a 32^3 unsigned distance field cube using 3D transpose convolutions. For training, we chose to use an l_1 smooth loss function which balances the benefits and downsides of both l_1 and l_2 losses.

Below, we go into more detail on the encoder and decoder parts of the network as well as some alternative design choices. Furthermore, the details of the loss function and training procedure are described.

5.1 Encoder

As shown by Qi et al. [48], the problem with taking point clouds as inputs lies in their unstructured nature. As opposed to regular grids used in methods that take volumetric

representations as inputs, the order of points in point clouds does not affect the geometry they are representing. This means that when reading a point cloud, no specific canonical order can be expected. Therefore, a neural network that consumes points has to be invariant to permutations amongst points. Solutions for this can be categorized into three classes: Either sort every point cloud canonically before feeding them into the network, take them in sequentially by using a RNN-like structure, or make use of a symmetric function that is invariant to the order of input data.

This last approach is the one found to be the most viable for point clouds [48]. We incorporate it into our network as a max pooling layer ((b) in figure 5.1) after the first three layers. These three layers perform 1D convolutions (which represents the MLP layer with shared weights from PointNet [48]).

Together, the convolutional and max pooling layers perform a compression of the input point cloud data to the latent space vector. The following two fully connected layers, operating on that latent space vector, then allow the network to learn how to complete the input data.

5.2 Decoder

We first reshape the output vector of the second fully-connected layer to a 4^3 cube with 32 filter channels. Then, the following three transpose convolutional layers grow the output cube resolution to 8, 16, and 32 while reducing the filter channel sizes to 16, 8, and 1. For these convolutional layers, we use a stride of 2 and a kernel size of 4.

The output from the last layer is a 32^3 unsigned distance field. The final mesh can be extracted at inference time by applying the Marching Cubes algorithm [35] to the predicted distance field.

5.3 Design Choices

Together with our main method presented above, we also experimented with some additional design choices which we briefly detail in this section. We explored if exchanging the encoder makes a difference and if adding a small classification network improves performance by providing additional information to the generator. We also evaluated a different, yet related objective: generating completed point clouds by just exchanging the decoder. This led us to our final design study, namely using a decoder that combines two generators, one for distance fields and one for point clouds. Intuitively, training both at the same time might boost performance as features learned by one can be useful to the other.

5.3.1 Encoders

As stated in section 5.1, we use a simplified version of the PointNet [48] encoder in our network. Our simplification consists of the removal of the transformation helper

networks in the original paper. They are placed before and after the first layer and both approximate a transformation matrix, the first one in point space and the second one in feature space. These matrices are applied to the incoming features which aligns them to a canonical space. The authors reported a gain in classification and segmentation performance when using these.

The original idea was taken one step further by PointNet++ [50]. Here, the authors proposed a more complex network structure which recursively applies the original PointNet. In doing so, local structures can now be taken into account which conceptually extends its use case to more fine-grained structures as well as large scenes.

We report the results of using the encoders of both of the methods above in chapter 6.

5.3.2 Classification

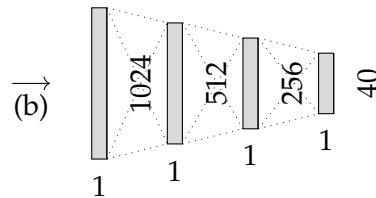


Figure 5.2: The architecture we use for the classification branch is the same as in PointNet [48]: Several fully connected layers learn to output a vector of size 40 which is the number of object classes in the dataset. It shares the latent space vector generated by the max pooling layer (b) with the reconstruction branches

Another direction we examined was not to swap out the encoder, but to modify the decoder part of our network. First of all, the initial idea was to aid the network by introducing classification as a pseudo loss. The intuition behind this is that if the network has to learn not only how to complete a certain shape but can also choose parts specific to a certain class and therefore, boost the overall completion performance.

We use the classification network used in PointNet [48] for their classification task. A visualization can be seen in figure 5.2. It branches off from the encoder (arrow (b) denotes the same max pooling operation as in figure 5.1, our main method). It then takes the the latent space vector of size 1024 and forwards it through three fully-connected layers, each one (except the last) followed by ReLU and batch normalization operations. After the second one, dropout with $p = 0.3$ is applied. The output is a vector of size 40, representing the 40 model classes in our dataset.

The classification loss is then measured and optimized for by a cross-entropy loss over the class labels.

5.3.3 Point Cloud Generation

Another path we explored was that of direct point cloud generation. This does not directly aid our main method of generating dense surface representations; however,

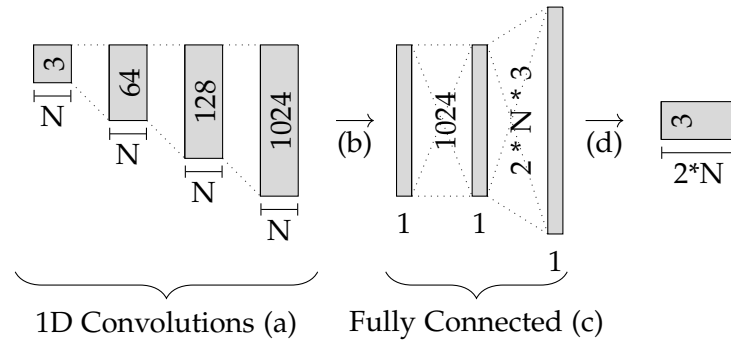


Figure 5.3: We modified our main network to still take a cloud of N 3D points but to then generate a completed point cloud instead of a distance field. The encoder is the same as in figure 5.1 ((a) and (b)). As a decoder, two fully connected layers are used (c). The output of the second one has size $2 * N * 3$, representing twice the amount of 3D points. A reshape operation (d) is then used to retrieve the actual point cloud

it is an easy modification of the original architecture and gives us another method to compare the hybrid decoder presented below against.

Figure 5.3 shows the general structure. Compared to the distance field generating network in figure 5.1, only the decoder changes: We use the same simplified point cloud encoder for point consumption. For the decoder, we replace the 3D transpose convolutions with another fully connected layer that generates $2 * N * 3$ values, i.e. twice the amount of 3D points. We use a factor of two as we want to keep the density of the input models which are oftentimes only visible from one side. In the end, a simple reshape into a $2 * N * 3$ array gives the coordinates of $2N$ points.

5.3.4 Hybrid Decoder

Following the introduction of a point cloud decoder, now combine both decoders into one which means performing both distance field and point cloud generation at the same time. The intuition behind this is that the generation of both forms of geometry might aid each other and therefore increase the overall prediction performance.

This design utilizes the established encoder used before for both distance field and point cloud prediction. After reaching the latent vector, the decoder splits into two branches: One for the distance field decoder and the other one for the point cloud decoder.

5.4 Loss Function

We considered several different loss functions for our network. Losses are needed for comparing the distance field predictions to their targets, comparing output point clouds

to the ground-truth, and comparing class labels to the actual object classes.

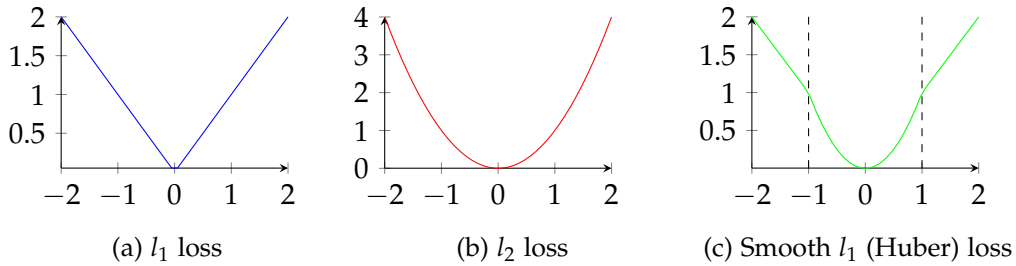


Figure 5.4: Differences between l_1 , l_2 , and l_1 smooth (Huber) loss. The difference between compared points (i.e. $x - y$ in equations 5.1, 5.2, and 5.3) is plotted along the x-axis and the respective loss function output along the y-axis. We use l_1 smooth loss in our experiments as it combines the stability of l_2 with the robustness of l_1

5.4.1 Smooth l_1 Loss for Distance Field Output

As we are comparing a predicted and a target distance field consisting of real numbers, intuitive choices for a loss function are either l_1 or l_2 losses. However, both choices have downsides that need to be addressed. While l_2 loss (equation 5.2) is considered to be more stable than l_1 loss (equation 5.1) for small values, it is also less robust to outliers due to its quadratic formulation. This means that if the dataset contains any 3D models which deviate from the others of a certain class too much, it disproportionately affects the loss function.

$$d_{l_1}(x, y) = \frac{1}{n} \sum_i |x_i - y_i| \quad (5.1)$$

$$d_{l_2}(x, y) = \frac{1}{n} \sum_i (x_i - y_i)^2 \quad (5.2)$$

Figure 5.4 visualizes both loss functions and compares them to the l_1 smooth loss used in our method.

Therefore, to keep the stability of the l_2 as well as the robustness of the l_1 loss functions, we use the so-called smooth l_1 (or Huber) loss [19]. Equation 5.3 shows its definition: It behaves like an l_2 loss if the difference of x and y values is less than 1 and like an l_1 loss otherwise.

$$d_{\text{huber}}(x, y) = \frac{1}{n} \sum_i z_i \text{ with } z_i = \begin{cases} \frac{1}{2}(x_i - y_i)^2 & \text{if } |x_i - y_i| < 1 \\ |x_i - y_i| - 0.5 & \text{otherwise} \end{cases} \quad (5.3)$$

This choice of loss function is a small but important detail in our architecture as it makes our approach more robust against outliers while preserving stability for small differences.

5.4.2 Distance Field Truncation and Logarithmic Scaling

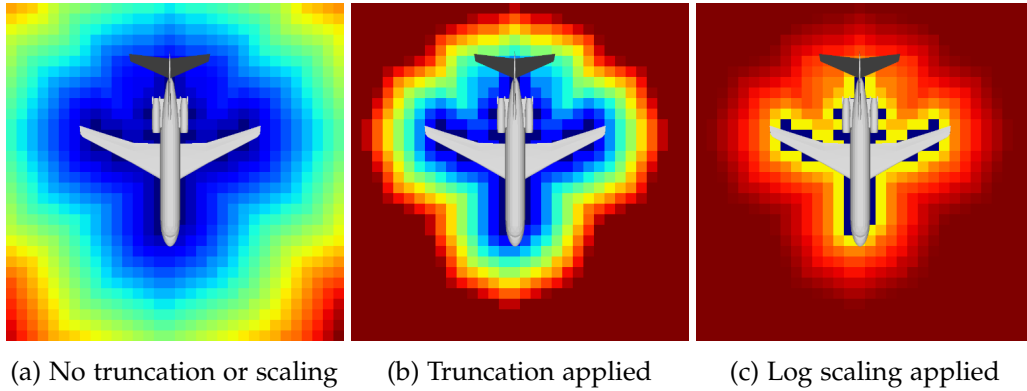


Figure 5.5: Comparison between the original distance field (a), the same distance field after truncation (b) and finally, after applying log scaling to the truncated volume (c). Depicted is a slice of size 32^2 of a distance field volume with the original CAD model overlaid. The color represents the distance values: It ranges from blue (low distance to surface) to red (high distance). Note that after applying truncation, the voxels furthest from the model all have the same color while after log scaling, the difference between neighboring distance values increases the closer to the surface it is. In our experiments, we apply both methods

One major caveat that has to be taken into account when comparing two volumes using smooth l_1 loss is that all voxels are weighted the same. This means that the difference between voxels close to the object’s surface receive the same attention as the ones furthest away. However, due to the formulation of distance fields, voxels far from the surface hold high values which means that distances between them also tend to be higher. Consequently, these voxels actually influence the loss to a much higher degree than the ones close to the surface.

This is in contrast to the expected behavior: Even little changes in the values of surface-near voxels have a high impact on the final reconstruction. Therefore, we want to condition our loss function mainly on the surface region while giving less attention to voxels only encoding free space.

There are two important techniques that we can facilitate to this end: Truncation of the distance values and logarithmic scaling. A visualization is given in figure 5.5.

Truncation limits the maximum value of the distance field and clamps all higher values to that value. This eliminates the problem of voxels far from the surface having too big of an impact on the overall loss when they actually do not add any additional useful information. The space of values to optimize is therefore limited to a certain region of interest around the actual object surface (as seen in figure 5.5b).

However, the impact of voxels closest to the surface can be further increased. This is achieved by scaling distance values below the truncation threshold with a function which

increases significantly as the input value decreases; the logarithm is such a function. Figure 5.5c shows this: Values closest to the surface now have significant impact on the loss while the impact decreases quickly when moving further away until reaching the truncation threshold.

Results show a significant increase in performance when applying both of these techniques; comparisons are presented in chapter 6.

5.4.3 Chamfer Loss for Point Cloud Generation

As point clouds are unstructured by design and we do not want to impose any kind of structure on them, the loss functions explained above cannot be used here. Instead, we use the Chamfer distance [3]. It is relatively easy to compute and, run on a GPU, sufficiently fast for use in a deep neural network. It works by finding the nearest neighbor for each point between two point clouds. Then, the distances of each pair are squared and summed up. This is done for both directions and both sums are then also summed. Equation 5.4 shows its formal definition.

$$d_{\text{chamfer}}(C_1, C_2) = \sum_{x \in C_1} \min_{y \in C_2} \|x - y\|_2^2 + \sum_{x \in C_2} \min_{y \in C_1} \|x - y\|_2^2 \quad (5.4)$$

This loss function allows us to perform the design study for point cloud generation in section 5.3.

5.5 Training

Except for the last one, we use batch normalization and ReLU operations after each layer. Each model is trained for 100 epochs with an initial learning rate of 0.001. We use ADAM as the optimizer and decay the learning rate by half every 20 epochs, following [13]. This was found to give a significant performance boost throughout the training. The batch size is fixed to 32 input point clouds per batch, except for the training of PointNet++ where it had to be reduced to 16 due to memory constraints.

We follow the original ModelNet40 train/test split with a ratio of roughly 80:20 training to test samples.

6 Results

All training and evaluation was performed on the ModelNet40 dataset [77] as discussed in chapter 4. We train on three different versions of the dataset: We always use 6 different trajectories for every object as an augmentation strategy and then individually train with 1, 3, and 6 rotations per object. We evaluate all models on the dataset with one rotation, giving us an overview over the usefulness of rotational augmentations during training.

We call the encoder of our method, as presented in chapter 5, simply "Point Cloud" and compare it to several alternative methods: First, 3D-EPN [13] which operates on volumetric grids. Second, the original PointNet [48] and third, the subsequently published PointNet++ [50].

Furthermore, we evaluate some of the design choices discussed in section 5.3: Adding a classification pseudo loss, using truncation and logarithmic scaling of the output volume and using a hybrid decoder to jointly predict distance fields and point clouds.

In the end, we also present some qualitative samples and discuss some of the limitations of our approach.

6.1 Quantitative Evaluation

In table 6.1, we evaluate the mean l_1 distance over all 40 object classes in the dataset. It can clearly be seen that the addition of truncation and logarithmic scaling of the output volume has a significant impact on the overall performance, independent of the number of rotations used for training.

The addition of more rotational augmentations, however, has less of an impact on the performance: The PointNet encoder with its transformation networks performs slightly better than our method in the case of 3 rotations, but is outperformed by the Point Cloud encoder in 1 and 6 rotations, while having a considerably higher number of parameters and a higher training time. PointNet++ does not give any significant improvements at all; considering that training with this method takes several times as long as our Point Cloud encoder, we find it to be infeasible for our task of shape completion.

The 3D-EPN approach operating on regular voxel grids also does not perform consistently better than our approach. While it is slightly better for 3 rotations, it falls behind Point Cloud's performance for 1 and 6 rotations. This is an interesting finding as it means that the additional information 3D-EPN can gain from its encoding of empty space do not give it a significant advantage compared to using point clouds as input.

Encoder	Rotations	Classification	Truncation & Log	l_1 Distance
Point Cloud	1			0.016105
Point Cloud	1		✓	0.000881
Point Cloud	1	✓	✓	0.001272
PointNet	1		✓	0.001145
PointNet++	1		✓	0.001126
3D-EPN	1		✓	0.000967
Point Cloud	3			0.016066
Point Cloud	3		✓	0.000996
PointNet	3		✓	0.000854
3D-EPN	3		✓	0.000907
Point Cloud	6			0.016103
Point Cloud	6		✓	0.001123
PointNet	6		✓	0.001212
3D-EPN	6		✓	0.001150

Table 6.1: We compare our encoder ("Point Cloud") to the volumetric approach 3D-EPN [13] as well as the original PointNet [48] and PointNet++ [50] encoders. We also compare different numbers of rotational augmentations as well as architectural choices classification and truncation and logarithmic scaling. Reported is the mean l_1 distance over all 40 object classes

6.2 Ablation Studies

We conducted several ablation studies, evaluating the design choices presented in section 5.3.

6.2.1 Classification

The addition of a classification pseudo-loss seems to have a negative impact on the reconstruction performance. This contrasts our expectation that the network would use information about the specific class of an incoming object to more efficiently generate a latent space representation used for completion. We suspect this result means that the network learns geometry reconstructions which are applicable to multiple classes. The constraint to a certain class per object might then limit this ability.

6.2.2 Predicting Complete Point Clouds

In addition to dense surface meshes, we also evaluate the reconstruction performance when generating point clouds. We report the mean accuracy and completeness over all 40 classes. Accuracy is defined to be the percentage of points in a prediction with a distance to their nearest neighbor in the ground truth above a certain predefined threshold (following the same basic concept as the Chamfer Distance, detailed in section

5.4). The completeness is defined as the reverse: The percentage of points in the ground-truth point cloud with a distance less than a certain threshold to their associated point in the prediction.

Table 6.2 shows the results. Classification again negatively impacts completion performance. The other major observation is that more rotations during training lead to more accurate point clouds. On the flip side, their completeness suffers noticeably in this case. Following these results, it seems like only using one rotation actually gives the best tradeoff between good accuracy and completeness.

Encoder	Rotations	Classification	Accuracy	Completeness
Point Cloud	1		82.2%	79.2%
Point Cloud	1	✓	75.8%	65.1%
Point Cloud	3		87.2%	69.3%
Point Cloud	6		87.8%	67.1%

Table 6.2: We use our Point Cloud encoder and predict point clouds, comparing the impact of a varying number of rotational augmentations and the addition of a classification pseudo-loss

6.2.3 Hybrid Decoder

Following distance field and point cloud evaluations, table 6.3 compares the results of both to those of the hybrid decoder which combines both.

The hybrid architecture increases the performance of point cloud generation. However, it cannot outperform the architecture which only uses a single distance field decoder.

Decoder	Rotations	l_1 Distance	Accuracy	Completeness
Point Cloud	1	-	82.2%	79.2%
Point Cloud	3	-	87.2%	69.3%
Point Cloud	6	-	87.8%	67.1%
Distance Field	1	0.000881	-	-
Distance Field	3	0.000996	-	-
Distance Field	6	0.001123	-	-
Hybrid	1	0.000970	82.7%	79.3%
Hybrid	3	0.000988	88.4%	69.7%
Hybrid	6	0.001266	87.3%	66.4%

Table 6.3: Comparing the results with only one Point Cloud decoder and Distance Field decoder with the combined hybrid decoder. The encoder is fixed to Point Cloud

6.3 Qualitative Samples

We use the Marching Cubes implementation of scikit-learn to extract surface meshes from the predicted distance field volumes. Figures 6.1 and 6.2 show some predicted meshes across different object classes of ModelNet40. Meshes are colored by distance from the camera to give a better sense of depth. Figure 6.2 (right column) also shows some results of our point cloud decoder.

In most circumstances, our method is able to generate a complete surface mesh even with major geometry parts missing from the input point cloud.

6.4 Limitations

Some visible failure cases of our completion pipeline are visualized in figure 6.3.

The most obvious limitation of the presented method is the low resolution of the output mesh. This is due to the size of the predicted volumetric distance field which in turn is restricted by limits in computing power, a problem shared by most 3D-CNN architectures.

There is also an issue with small structures: Thin geometry such as legs of tables and chairs might be cut off, very fine details are sometimes fused together, and thin surfaces might still not get completely reconstructed in the final prediction.

Furthermore, the approach we use for ingesting point clouds only works with a predefined number of points in each cloud. This means that unseen point clouds need to be preprocessed before they can be fed into our network.

As we employ a fully supervised approach, we also restrict ourselves to scenarios in which there exist both the partial scans as well as complete representations of each object. However, for training on real-world data, these representations might not be available.


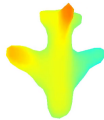
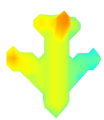

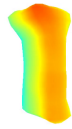

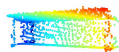
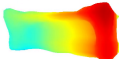
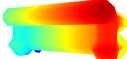
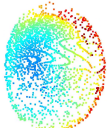
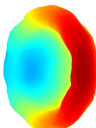
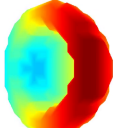

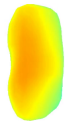
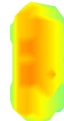
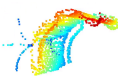
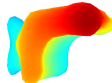
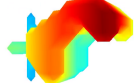






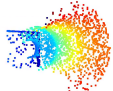






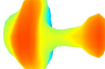
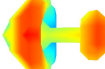
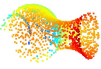
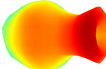
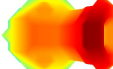
Input	Result	Target	Input	Result	Target
airplane			bed		
					
bench			bowl		
					
car			chair		
					
cone			cup		
					
flower pot			guitar		
					
lamp			vase		
					

Figure 6.1: Qualitative Results: Shown are samples from different object categories. For each object, input point cloud, completed mesh, and target mesh are visualized

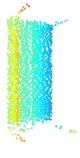
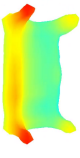




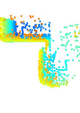
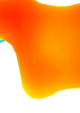

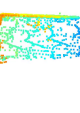
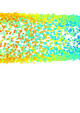
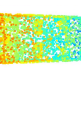
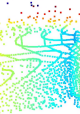


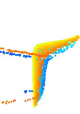
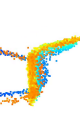
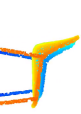
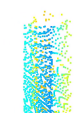
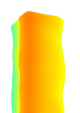
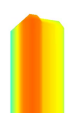

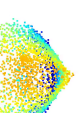
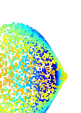

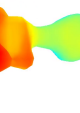
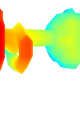
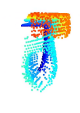
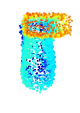
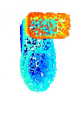
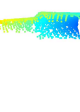
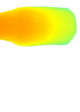
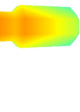
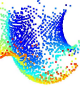

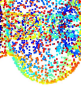
Input	Result	Target	Input	Result	Target
table			airplane		
					
toilet			car		
					
night stand			chair		
					
glass box			cone		
					
stool			toilet		
					
bottle			vase		
					

Figure 6.2: Qualitative Results continued: Left column shows more results for mesh completion while the right column shows some results of point cloud generation


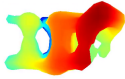

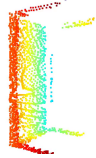
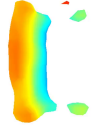
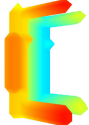

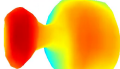
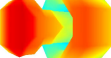
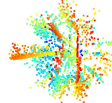
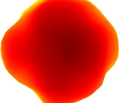
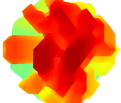
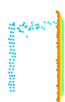
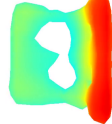
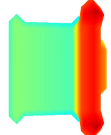
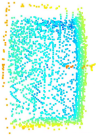
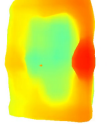
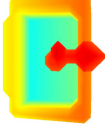
Input	Result	Target	Input	Result	Target
chair			desk		
					
lamp			plant		
					
laptop			sink		
					

Figure 6.3: Some examples for limitations of our method:

Missing geometry for fine structures like the legs of a chair or a table
(objects chair and desk in the top row),

Fusion of fine structures with little space in between
(objects lamp and plant in the middle row),

Missing geometry for thin parts if the input did not contain enough information there
(objects laptop and sink in the bottom row)

7 Conclusion

We have presented a method that is capable of predicting completed shapes from sparse and partial input point clouds. Our autoencoder-like architecture directly operates on an unstructured set of 3D points and learns shape completion in a fully supervised manner.

Our results show that this data-driven concept not only works on one or a few object categories but is able to predict completions across all 40 classes of ModelNet40.

While there are still some limitations, we can conclude that data-driven post-processing of point clouds is viable and produces good results.

Further work might explore some of those limitations: Outputting meshes with higher resolution and lifting restrictions such as the need for a fixed number of input points and a fully supervised training.

List of Figures

1.1	Visualization of LiDAR and Laser Scanning Techniques	2
1.2	Sparsity and Partialness in Point Clouds	3
3.1	High-Level Method Overview	15
4.1	ModelNet40 CAD Model Samples	17
4.2	Virtual Scanning of ModelNet40 Objects	19
4.3	Visualization of a Distance Field	20
5.1	Distance Field Generating Network Architecture	23
5.2	Classification Network Architecture	25
5.3	Point Cloud Generating Network Architecture	26
5.4	Comparison between l_1 , l_2 , and l_1 smooth (Huber) loss	27
5.5	Comparing Unscaled Distance Field, Truncation, and Logarithmic Scaling	28
6.1	Qualitative Results for Mesh Output	35
6.2	Qualitative Results for Mesh and Point Cloud Output	36
6.3	Limitations	37

List of Tables

4.1	Statistics of the Generated Dataset	22
6.1	Evaluation Results for Distance Field Output	32
6.2	Evaluation Results for Point Cloud Output	33
6.3	Evaluation Results for Hybrid Output	33

Bibliography

- [1] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. J. Guibas. "Learning Representations and Generative Models for 3D Point Clouds." In: *ICLR* (2018).
- [2] J. Amanatides and A. Woo. "A Fast Voxel Traversal Algorithm for Ray Tracing." In: (1987).
- [3] H. G. Barrow, J. M. Tenenbaum, R. C. Bolles, and H. C. Wolf. "Parametric Correspondence and Chamfer Matching - Two New Techniques for Image Matching." In: *IJCAI* (1977).
- [4] J. Beltrán, C. Guindel, F. M. Moreno, D. Cruzado, F. García, and A. de la Escalera. "BirdNet - A 3D Object Detection Framework from LiDAR Information." In: *ITSC* (2018), pp. 3517–3523.
- [5] M. Berger, A. Tagliasacchi, L. M. Seversky, P. Alliez, G. Guennebaud, J. A. Levine, A. Sharf, and C. T. Silva. "A Survey of Surface Reconstruction from Point Clouds." In: *Comput. Graph. Forum* (2017).
- [6] O. Cameron. *An Introduction to LIDAR: The Key Self-Driving Car Sensor*. 2017. URL: <https://news.voyage.auto/an-introduction-to-lidar-the-key-self-driving-car-sensor-a7e405590cff> (visited on 07/06/2019).
- [7] A. X. Chang, T. A. Funkhouser, L. J. Guibas, P. Hanrahan, Q.-X. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. "ShapeNet - An Information-Rich 3D Model Repository." In: *CoRR* (2015).
- [8] J. Chen, D. Bautembach, and S. Izadi. "Scalable real-time volumetric surface reconstruction." In: *ACM Transactions on Graphics* (2013).
- [9] W. Chen, X. Han, G. Li, C. Chen, J. Xing, Y. Zhao, and H. Li. "Deep RBFNet - Point Cloud Feature Learning using Radial Basis Functions." In: *CoRR* (2018).
- [10] S. Choi, Q.-Y. Zhou, and V. Koltun. "Robust reconstruction of indoor scenes." In: *CVPR* (2015), pp. 5556–5565.
- [11] B. Curless and M. Levoy. "A Volumetric Method for Building Complex Models from Range Images." In: *SIGGRAPH* (1996), pp. 303–312.
- [12] A. Dai, M. Nießner, M. Zollhöfer, S. Izadi, and C. Theobalt. "BundleFusion - Real-Time Globally Consistent 3D Reconstruction Using On-the-Fly Surface Reintegration." In: *ACM Transactions on Graphics* (2017).
- [13] A. Dai, C. R. Qi, and M. Nießner. "Shape Completion Using 3D-Encoder-Predictor CNNs and Shape Synthesis." In: *CVPR* (2017), pp. 6545–6554.

- [14] A. Dai, D. Ritchie, M. Bokeloh, S. Reed, J. Sturm, and M. Nießner. “ScanComplete - Large-Scale Scene Completion and Semantic Segmentation for 3D Scans.” In: *CVPR* (2018), pp. 4578–4587.
- [15] E. L. Denton, S. Chintala, A. Szlam, and R. Fergus. “Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks.” In: *NIPS* (2015).
- [16] M. Gadelha, R. Wang, and S. Maji. “Multiresolution Tree Networks for 3D Point Cloud Processing.” In: *ECCV 11211.6* (2018), pp. 105–122.
- [17] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. “Generative Adversarial Nets.” In: *NIPS* (2014).
- [18] B.-S. Hua, M.-K. Tran, and S.-K. Yeung. “Pointwise Convolutional Neural Networks.” In: *CVPR* (2018), pp. 984–993.
- [19] P. J. Huber. “Robust Estimation of a Location Parameter.” In: *Ann. Math. Statist.* 35.1 (Mar. 1964), pp. 73–101. doi: 10.1214/aoms/1177703732.
- [20] D. J. Im, S. Ahn, R. Memisevic, and Y. Bengio. “Denoising Criterion for Variational Auto-Encoding Framework.” In: *AAAI* (2017).
- [21] T. Karras, T. Aila, S. Laine, and J. Lehtinen. “Progressive Growing of GANs for Improved Quality, Stability, and Variation.” In: *ICLR* (2018).
- [22] M. M. Kazhdan, M. Bolitho, and H. Hoppe. “Poisson surface reconstruction.” In: *Symposium on Geometry Processing* (2006).
- [23] M. M. Kazhdan and H. Hoppe. “Screened poisson surface reconstruction.” In: *ACM Transactions on Graphics* (2013).
- [24] Y. M. Kim, N. J. Mitra, D.-M. Yan, and L. J. Guibas. “Acquiring 3D indoor environments with variability and repetition.” In: *ACM Transactions on Graphics* (2012).
- [25] C. Kingkan and K. Hashimoto. “Generating Mesh-based Shapes From Learned Latent Spaces of Point Clouds with VAE-GAN.” In: *ICPR* (2018), pp. 308–313.
- [26] D. P. Kingma and M. Welling. “Auto-Encoding Variational Bayes.” In: *ICLR* (2014).
- [27] T. Le and Y. Duan. “PointGrid - A Deep Network for 3D Shape Understanding.” In: *CVPR* (2018), pp. 9204–9214.
- [28] H. Lei, N. Akhtar, and A. Mian. “Spherical Convolutional Neural Network for 3D Point Clouds.” In: *CoRR* (2018).
- [29] M. Levoy and K. Pulli. *The Digital Michelangelo Project: 3D Scanning of Large Statues (Presentation Slides)*. 2000. URL: <http://graphics.stanford.edu/talks/DigMich.html> (visited on 07/06/2019).
- [30] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. E. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk. “The digital Michelangelo project - 3D scanning of large statues.” In: *SIGGRAPH* (2000), pp. 131–144.
- [31] C.-L. Li, M. Zaheer, Y. Zhang, B. Póczos, and R. Salakhutdinov. “Point Cloud GAN.” In: *CoRR* (2018).

-
- [32] D. Li, T. Shao, H. Wu, and K. Zhou. "Shape Completion from a Single RGBD Image." In: *IEEE Trans. Vis. Comput. Graph.* (2017).
- [33] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen. "PointCNN - Convolution On X-Transformed Points." In: *NeurIPS* (2018).
- [34] Y. Li, A. Dai, L. J. Guibas, and M. Nießner. "Database-Assisted Object Retrieval for Real-Time 3D Reconstruction." In: *Comput. Graph. Forum* 34.2 (2015), pp. 435–446.
- [35] W. E. Lorensen and H. E. Cline. "Marching cubes - A high resolution 3D surface construction algorithm." In: *SIGGRAPH* (1987), pp. 163–169.
- [36] P. Mandikal and V. B. Radhakrishnan. "Dense 3D Point Cloud Reconstruction Using a Deep Pyramid Network." In: *WACV* (2019), pp. 1052–1060.
- [37] D. Maturana and S. Scherer. "VoxNet - A 3D Convolutional Neural Network for real-time object recognition." In: *IROS* (2015), pp. 922–928.
- [38] M. Mirza and S. Osindero. "Conditional Generative Adversarial Nets." In: *CoRR* (2014).
- [39] N. J. Mitra, L. J. Guibas, and M. Pauly. "Partial and approximate symmetry detection for 3D geometry." In: *ACM Transactions on Graphics* 25.3 (2006), p. 560.
- [40] L. Nan, K. Xie, and A. Sharf. "A search-classify approach for cluttered indoor scene understanding." In: *ACM Transactions on Graphics* (2012).
- [41] A. Nealen, T. Igarashi, O. Sorkine, and M. Alexa. "Laplacian mesh optimization." In: *GRAPHITE* (2006), p. 381.
- [42] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. W. Fitzgibbon. "KinectFusion - Real-time dense surface mapping and tracking." In: *ISMAR* (2011), pp. 127–136.
- [43] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger. "Real-time 3D reconstruction at scale using voxel hashing." In: *ACM Transactions on Graphics* (2013).
- [44] A. van den Oord, N. Kalchbrenner, L. Espeholt, K. Kavukcuoglu, O. Vinyals, and A. Graves. "Conditional Image Generation with PixelCNN Decoders." In: *NIPS* (2016).
- [45] A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu. "Pixel Recurrent Neural Networks." In: *ICML* (2016).
- [46] M. Pauly, N. J. Mitra, J. Giesen, M. H. Gross, and L. J. Guibas. "Example-Based 3D Scan Completion." In: *Symposium on Geometry Processing* (2005).
- [47] M. Pauly, N. J. Mitra, J. Wallner, H. Pottmann, and L. J. Guibas. "Discovering structural regularity in 3D geometry." In: *ACM Transactions on Graphics* 27.3 (2008), p. 1.
- [48] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. "PointNet - Deep Learning on Point Sets for 3D Classification and Segmentation." In: *CVPR* (2017), pp. 77–85.

- [49] C. R. Qi, H. Su, M. Nießner, A. Dai, M. Yan, and L. J. Guibas. "Volumetric and Multi-view CNNs for Object Classification on 3D Data." In: *CVPR* (2016), pp. 5648–5656.
- [50] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. "PointNet++ - Deep Hierarchical Feature Learning on Point Sets in a Metric Space." In: *NIPS* (2017).
- [51] A. Radford, L. Metz, and S. Chintala. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks." In: *ICLR* (2016).
- [52] S. E. Reed, A. van den Oord, N. Kalchbrenner, S. G. Colmenarejo, Z. Wang, Y. Chen, D. Belov, and N. de Freitas. "Parallel Multiscale Autoregressive Density Estimation." In: *ICML* (2017).
- [53] E. Remelli, P. Baqué, and P. Fua. "NeuralSampler - Euclidean Point Cloud Auto-Encoder and Sampler." In: *CoRR* (2019).
- [54] D. Rethage, J. Wald, J. Sturm, N. Navab, and F. Tombari. "Fully-Convolutional Point Networks for Large-Scale Point Clouds." In: *ECCV 11208.6* (2018), pp. 625–640.
- [55] G. Riegler, A. O. Ulusoy, and A. Geiger. "OctNet - Learning Deep 3D Representations at High Resolutions." In: *CVPR* (2017), pp. 6620–6629.
- [56] J. Rock, T. Gupta, J. Thorsen, J. Gwak, D. Shin, and D. Hoiem. "Completing 3D object shape from one depth image." In: *CVPR* (2015), pp. 2484–2493.
- [57] T. Salimans, I. J. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. "Improved Techniques for Training GANs." In: *NIPS* (2016).
- [58] K. Sarkar, E. Mathews, and D. Stricker. "Structured 2D Representation of 3D Data for Shape Processing." In: *CoRR* (2019).
- [59] K. Sarkar, K. Varanasi, and D. Stricker. "3D Shape Processing by Convolutional Denoising Autoencoders on Local Patches." In: *WACV* (2018), pp. 1925–1934.
- [60] T. Shao, W. Xu, K. Zhou, J. Wang, D. Li, and B. Guo. "An interactive approach to semantic modeling of indoor scenes with an RGBD camera." In: *ACM Transactions on Graphics* (2012).
- [61] A. Sharma, O. Grau, and M. Fritz. "VConv-DAE - Deep Volumetric Shape Learning Without Object Labels." In: *ECCV Workshops* (2016).
- [62] Y. Shen, C. Feng, Y. Yang, and D. Tian. "Mining Point Cloud Local Structures by Kernel Correlation and Graph Pooling." In: *CVPR* (2018), pp. 4548–4557.
- [63] I. Sipiran, R. Gregor, and T. Schreck. "Approximate Symmetry Detection in Partial 3D Meshes." In: *Comput. Graph. Forum* 33.7 (2014), pp. 131–140.
- [64] E. J. Smith and D. Meger. "Improved Adversarial Systems for 3D Object Generation and Reconstruction." In: *CoRL* (2017).
- [65] O. Sorkine and D. Cohen-Or. "Least-Squares Meshes." In: *SMI* (2004), pp. 191–199.

-
- [66] P. Speciale, M. R. Oswald, A. Cohen, and M. Pollefeys. "A Symmetry Prior for Convex Variational 3D Reconstruction." In: *ECCV* 9912.1 (2016), pp. 313–328.
- [67] D. Stutz and A. Geiger. "Learning 3D Shape Completion From Laser Scan Data With Weak Supervision." In: *CVPR* (2018), pp. 1955–1964.
- [68] M. Sung, V. G. Kim, R. Angst, and L. Guibas. "Data-driven structural priors for shape completion." In: *ACM Transactions on Graphics (TOG)* 34.6 (Nov. 2015), pp. 175–11.
- [69] S. Thrun and B. Wegbreit. "Shape from Symmetry." In: *ICCV* (2005), 1824–1831 Vol. 2.
- [70] M. Wang, M. Ju, Y. Fan, S. Guo, M. Liao, H. Yang, D. He, and T. Komura. "3D Incomplete Point Cloud Surfaces Reconstruction With Segmentation and Feature-Enhancement." In: *IEEE Access* (2019).
- [71] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong. "O-CNN - octree-based convolutional neural networks for 3D shape analysis." In: *ACM Transactions on Graphics* (2017).
- [72] W. Wang, Q. Huang, S. You, C. Yang, and U. Neumann. "Shape Inpainting Using 3D Generative Adversarial Network and Recurrent Convolutional Networks." In: *ICCV* (2017), pp. 2317–2325.
- [73] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. "Dynamic Graph CNN for Learning on Point Clouds." In: *CoRR* (2018).
- [74] T. Whelan, R. F. Salas-Moreno, B. Glocker, A. J. Davison, and S. Leutenegger. "ElasticFusion - Real-time dense SLAM and light source estimation." In: *I. J. Robotics Res.* 35.14 (2016), pp. 1697–1716.
- [75] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum. "Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling." In: *NIPS* (2016).
- [76] W. Wu, Z. Qi, and F. Li. "PointConv - Deep Convolutional Networks on 3D Point Clouds." In: *CoRR* (2018).
- [77] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. "3D ShapeNets - A deep representation for volumetric shapes." In: *CVPR* (2015), pp. 1912–1920.
- [78] Y. Xu, T. Fan, M. Xu, L. Zeng, and Y. Qiao. "SpiderCNN - Deep Learning on Point Sets with Parameterized Convolutional Filters." In: *ECCV* 11212.4 (2018), pp. 90–105.
- [79] B. Yang, S. Rosa, A. Markham, N. Trigoni, and H. Wen. "3D Object Dense Reconstruction from a Single Depth View." In: *CoRR* (2018).
- [80] B. Yang, H. Wen, S. Wang, R. Clark, A. Markham, and N. Trigoni. "3D Object Reconstruction from a Single Depth View with Adversarial Learning." In: *ICCV Workshops* (2017), pp. 679–688.

- [81] Y. Yang, C. Feng, Y. Shen, and D. Tian. "FoldingNet - Point Cloud Auto-Encoder via Deep Grid Deformation." In: *CVPR* (2018), pp. 206–215.
- [82] L. Yu, X. Li, C.-W. Fu, D. Cohen-Or, and P.-A. Heng. "PU-Net - Point Cloud Upsampling Network." In: *CVPR* (2018), pp. 2790–2799.
- [83] W. Yuan, T. Khot, D. Held, C. Mertz, and M. Hebert. "PCN - Point Completion Network." In: *3DV* (2018).
- [84] M. Zamorski, M. Zieba, R. Nowak, W. Stokowiec, and T. Trzcinski. "Adversarial Autoencoders for Generating 3D Point Clouds." In: *CoRR* (2018).
- [85] W. Zhang, Z. Yang, H. Jiang, S. Nigam, S. Yamakawa, T. Furuhashi, K. Shimada, and L. B. Kara. "3D Shape Synthesis for Conceptual Design and Optimization Using Variational Autoencoders." In: *CoRR* (2019).
- [86] W. Zhao, S. Gao, and H. Lin. "A Robust Hole-Filling Algorithm for Triangular Mesh." In: *CAD/Graphics* (2007), pp. 22–22.
- [87] Y. Zhou and O. Tuzel. "VoxelNet - End-to-End Learning for Point Cloud Based 3D Object Detection." In: *CVPR* (2018), pp. 4490–4499.
- [88] M. Zollhöfer, P. Stofko, A. Görlitz, C. Theobalt, M. Nießner, R. Klein, and A. Kolb. "State of the Art on 3D Reconstruction with RGB-D Cameras." In: *Comput. Graph. Forum* 37.2 (2018), pp. 625–652.