TU WIEN Informatics

Smart Surface Reconstruction

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der Technischen Wissenschaften

eingereicht von

Dipl.-Ing. Philipp Erler, B.Eng.
Matrikelnummer 01426424

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer

Diese Dissertation haben begutachtet:

Angela Dai	Paolo Cignoni

Wien, 30. Dezember 2025

Philipp Erler

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

TU WIEN Informatics

Smart Surface Reconstruction

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Dipl.-Ing. Philipp Erler, B.Eng.
Registration Number 01426424

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer

The dissertation has been reviewed by:

Angela Dai	Paolo Cignoni

Vienna, December 30, 2025

Philipp Erler

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Dipl.-Ing. Philipp Erler, B.Eng.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, haben ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT- Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 30. Dezember 2025

Philipp Erler

Acknowledgements

Finishing this doctorate would not have been possible without a number of people who supported me for all those years.

First, I thank my supervisor, Michael Wimmer, for the consistent guidance on what to do and the freedom on how to achieve it. His ability to squeeze every last penny out of projects is truly remarkable.

Many thanks to Stefan Ohrhallinger, who kicked off my PhD. Although his idea of advancing-front mesh simplification did not work out in the end, I learned a lot about geometry processing from him. The BBQs on his boat are always a highlight.

Special thanks go to Niloy Mitra and Paul Guerrero, who hosted me as a visiting researcher for almost 6 months in London. They introduced me to deep learning for geometry, which dominated most of my PhD and will likely stay at the core of my research.

I thank the whole Rendering and Modeling Group, all the surrounding groups and institutions, most notably the Vis-Group, VRVis, CVL, and AR/VR group. Specifically, I thank Adam Celarek, Markus Schütz, and Pedro Hermosilla for all the discussions and ideas. Further, I thank Annalena Ulschmid and Johannes Unterguggenberger for taking such a great part of the teaching load with so much enthusiasm.

Thanks to my students and especially Christian Clemenz, Maximilian Riegler, Kerstin Hofer, and Florian Steinschorn, who trusted me to (co-)supervise their theses. They went way beyond the required effort. I learn a lot with every supervision and teaching in general.

Lizeth Fuentes-Perez came to Vienna for half a year, during which she supported me so much. When nothing seemed to work, we could dig through the problem for hours and eventually solve it. Eternal thanks to you for keeping me running and leaving the valley of despair!

Thanks to my parents, Karin and Uwe, as well as my brother Tim, for laying the foundation for the family's first doctorate. Thank you for your continuous support in so many ways.

Finally, I thank my dear Lisa Fellinger for making my life better in every regard. The adventure of the PhD is now coming to an end, but the adventure of parenthood is just beginning.

Kurzfassung

Punktwolken sind zwar leicht zu erzeugen, jedoch oft verrauscht, unvollständig und ohne Konnektivität. Sie können geschlossene Oberflächen nicht effizient darstellen, und eine Weiterverarbeitung für geometrische Anwendungen wie 3D-Druck ist schwierig. Daher möchten wir sie in Oberflächenrepräsentationen, insbesondere in Dreiecksnetze, umwandeln. Diese Dissertation befasst sich mit der Herausforderung, aus fehlerhaften Punktwolken genaue und robuste 3D-Objekte zu rekonstruieren.

Traditionelle Methoden stoßen bei den Defiziten von Punktwolken an ihre Grenzen, was die Erforschung datengetriebener Ansätze motiviert. Deep Learning auf Punktwolken ist jedoch schwierig, da diese ungeordnet und unstrukturiert sind. Zudem ist ein Subsampling notwendig, weil neuronale Netze eine fixe Input-Größe haben. Dieses Subsample erzeugt jedoch Rauschen durch die teilweise zufällige Auswahl der Punkte. Unsere Arbeit untersucht Lösungen für diese Herausforderungen.

Wir präsentieren drei Beiträge zum Feld der Oberflächenrekonstruktion: Points2Surf, das lokale und globale Priors kombiniert, um die Oberflächenrekonstruktion über verschiedene Objektklassen hinweg zu generalisieren; PPSurf, das Point-Convolutions und Attention nutzt, um die Rekonstruktionsqualität weiter zu verbessern; und LidarScout, das eine echtzeitfähige, out-of-core Visualisierung riesiger Luft-LIDAR-Scans durch effiziente und lokale Schätzung von Höhenkarten ermöglicht.

Unsere Ergebnisse zeigen, dass das Gleichgewicht zwischen lokalen Details und globalem Kontext entscheidend für hochwertige, generalisierbare Rekonstruktionen ist. Wir zeigen, dass Modelldesign, Datenaugmentation und effiziente Repräsentationen entscheidend sind, um mit Rauschen und fehlenden Daten umzugehen. Die Erkenntnisse bieten praxisnahe Lösungen für die Rekonstruktion von einzelnen Objekten bis hin zu ganzen Landschaften.

Abstract

Point clouds, while easy to acquire, are often noisy, incomplete, and lack connectivity. They cannot represent closed surfaces efficiently, and processing for geometrical applications like 3D printing is difficult. Therefore, we want to convert them to surface representations, most importantly, triangle meshes. This dissertation addresses the challenge of reconstructing accurate and robust 3D objects from imperfect point clouds.

Traditional methods struggle with point cloud defects, motivating the exploration of data-driven approaches. However, deep learning on point clouds is difficult due to their unordered and unstructured nature. Further, subsampling is necessary to fit them into the fixed-size input of a neural network, which introduces randomness. In this thesis, we explore solutions for these challenges.

We present three main contributions: Points2Surf, which combines local and global priors to generalize surface reconstruction across diverse object classes; PPSurf, which leverages point convolutions and attention to further improve reconstruction quality; and LidarScout, which enables real-time, out-of-core rendering of massive aerial LIDAR scans by focusing on efficient, local heightmap estimation.

Our results demonstrate that balancing local detail with global context is key to achieving high-quality, generalizable reconstructions. We show that model design, data augmentation, and efficient representations are crucial for handling noise and missing data. The findings offer practical solutions for reconstructions from single objects to entire landscapes.

Contents

CHAPTER 1

Introduction

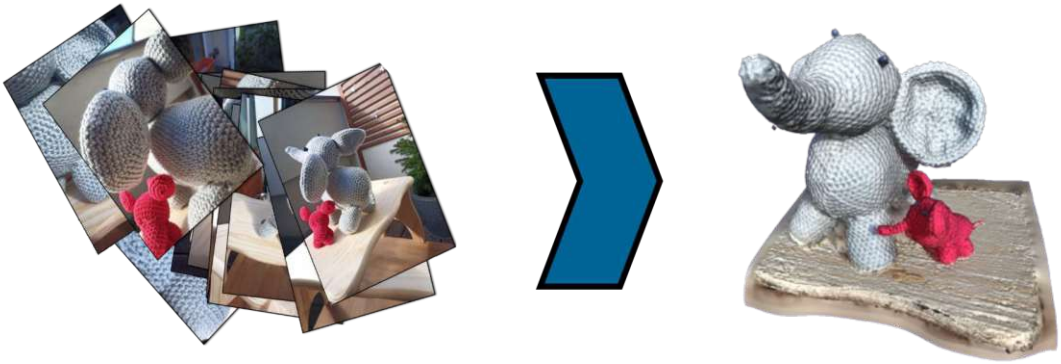


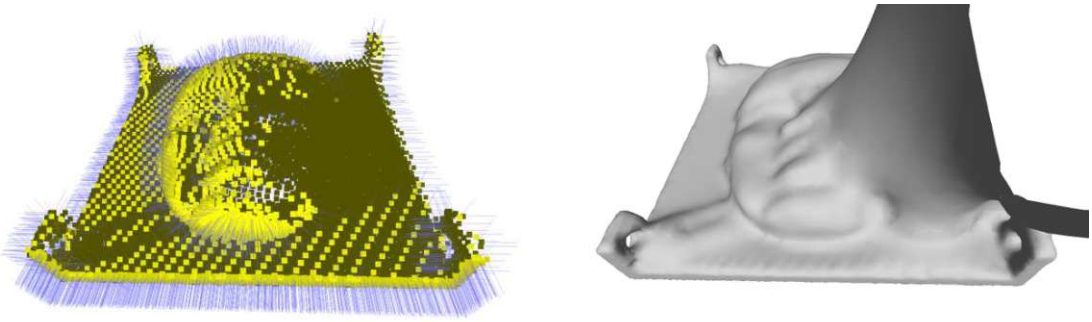
Figure 1.1: Surface reconstruction aims to recover the original surface from scanner data, here images.

1.1 Motivation

Imagine you want to create a digital 3D model of a real-world object like a statue, a car, a house, a street, or even an entire landscape. This is a common need in fields such as movies, games, virtual reality, cultural heritage preservation, autonomous driving, and geology. The first step is often to scan the object using cameras or laser scanners, which results in a collection of samples in space called a *point cloud*. Each sample represents a spot on the surface of the object, but they are not connected.

Thanks to the widespread availability of smartphones, point clouds are now easy to acquire. However, such point sets are not efficient or not usable for many applications. For example, quickly rendering a closed surface or preparing a 3D print requires a more

structured representation. This is where *surface reconstruction* comes in: the process of converting a raw point cloud into a continuous surface. Such a surface is typically represented as a *mesh*, a digital wireframe made of triangles. A *Signed Distance Field (SDF)* is a mathematical function describing the distance to the surface at every point in space, often acting as a clean intermediate representation that is later converted to a mesh. Such a mesh can be further processed with modeling tools and other programs to be used in all the 3D applications that fill our lives.



(a) A point cloud with missing regions (b) A bad reconstruction

Figure 1.2: Point clouds often have defects like scan shadows. A good reconstruction method needs to be robust against this.

When we started this work, classic, non-data-driven methods for surface reconstruction were the state of the art, most importantly Screened Poisson Surface Reconstruction (SPSR) [KH13]. Such algorithms are based on fixed rules and formulas, and they only receive inputs for the object they are trying to reconstruct. Therefore, such methods do not know similar objects, scan patterns, or other possibly relevant context information. As a result, they are not very robust against scan defects like noise, occlusions, and varying point density. Figure 1.2 shows the result of an incomplete point cloud given to SPSR. Further, SPSR and similar methods either require point normals or will produce noisy surfaces.

Deep learning has the potential to solve these issues, since it can learn the context from examples and apply it to the problem at hand. As an example, such a data-driven method can learn that chairs usually have four legs, which it will reconstruct even when it has only samples from three legs. More abstractly, a model will learn how samples from a noisy scanner correlate to a clean surface, which allows for noise-robust reconstruction.

When this thesis started, deep learning for images had already shown impressive results, while deep learning for point clouds was still in its infancy. Point clouds, with their unordered point sets of varying size, pose an elemental challenge to deep learning, which was tackled first by PointNet [QSMG17] but is still a concern. Similarly, early methods like PointNet and AtlasNet [GFK+18] depended strongly on predefined classes, such as chairs and cars, producing bad results for unseen object types. Also, they encoded

the entire shape in a single feature vector, which puts a hard limit on the amount of information that can be stored. This is especially problematic for complex and detailed objects.

In this thesis, we explore the potential of data-driven surface reconstruction with a focus on the differences between local and global priors. Particularly, we compare how they affect efficiency, quality, and generalization of the methods.

1.2 Problem Statement

In this thesis, we focus on data-driven surface reconstruction from point clouds. Here, we briefly introduce the main concepts necessary to understand our contributions and our research goals.

Point clouds: We define point clouds as a set of unordered points in 3D space, usually acquired by a 3D scanner or photogrammetry software. The points can be accompanied by additional information like normals, colors, and reflectance. Typical scan defects include noise, occlusions, and varying point density. For most parts, we do not care about the source of point clouds. We are interested in methods that can replace the current state of the art, and therefore, we aim to make our algorithms robust enough to cater to them all. More precisely, we use a simulated time-of-flight scanner for training and various other sources, like photogrammetry, for testing.

Surfaces: Several definitions of surfaces exist depending on the context. Here, we define them as the set of points on the boundary of an object. We focus on solid objects in three-dimensional Cartesian space. To be solid means that the objects must have a consistent inside and outside. Therefore, objects must be closed and have a clean topology without self-intersections or singularities. While many methods see the objects in their entirety, some others, especially heightmap-based methods, see only a small fraction of an object and split the domain into inside and outside along the up-axis. Surfaces are often represented as implicit surfaces (i.e., signed distance fields) or explicit surfaces (i.e., triangle meshes), both with advantages and disadvantages.

Surface reconstruction: Surface reconstruction is the process of estimating the original surface from which a point cloud was sampled. This is an ill-defined problem, as there are infinitely many surfaces that can produce the same point cloud. However, some surfaces are more likely than others. We aim to produce a clean and accurate surface, even in the presence of missing and distorted data. Further, we need to infer global properties like inside/outside without having clear indications in the input data, i.e., no point normals. Classic, non-data-driven methods fit the estimated surface to the given points, while data-driven ones predict the surface from such points using knowledge from similar objects.

Efficiency and Quality: What makes a good surface-reconstruction method depends to some degree on the task at hand. However, we have some metrics that are generally relevant. The most important aspect of the efficiency of a technique is to use as little

memory as possible and be as fast as possible. Often, everything below a certain threshold is *good enough* and does not receive any more effort for further improvement. We measure the quality mostly by comparing the deviation of the reconstruction from the original surface. This can be done at the surface level with, e.g., Chamfer distance and normal deviation, or over the volume with, e.g., Intersection-over-Union and F1-Score. Since these metrics work with quantization or sampling, they introduce some error and randomness. Efficiency and quality need a careful trade-off to make a method useful for the target application.

Deep learning: One methodology that can be used to develop surface-reconstruction algorithms is deep learning. It is a subfield of machine learning that uses artificial neural networks to learn complex patterns in data. It has been successfully applied to a wide range of tasks, especially in 2D. Deep learning on point clouds for reconstruction was in its infancy at the start of this thesis. At its core, parameters in the layers, together with activation functions, produce signals for their inputs. A loss function measures the error between the final outputs and the desired outcome. Back-propagation assigns a share of this error corresponding to their influence. An optimizer changes the parameters in every iteration to minimize the loss. If the training process and model are well designed, the network's weights converge to an optimum for the given task. We can choose and develop many loss functions for the learning process, emphasizing certain properties of the outputs like smoothness.

Supervised learning: Supervised learning is a machine learning paradigm where we show the model many input-output pairs and let it minimize the deviation from its prediction to the ground-truth output. This is opposed to, e.g., reinforcement learning, a type of unsupervised learning, where we reward the model for changing its environment positively. The main challenge of supervised learning is having a good dataset. The training data should be as close as possible to the real data, which is often prohibitively expensive. Therefore, we work around this issue with realistic but synthetic training data.

Generalization: A key aspect of machine learning is generalization, the ability of a model to perform well on unseen data. In our context, this is most relevant when switching the system from training to testing, since we usually train on synthetic data but apply it to the real world later. A similar problem arises when models are trained on a specific type of object but used for other classes. Another common cause of bad generalization is the point cloud source, e.g., laser scanner vs photogrammetry, each with its different properties and scan artifacts.

Priors: Whether a model can generalize well depends in large part on the priors it encodes, which are essentially assumptions about the solutions of a problem. As an example, a model for signed distance fields could learn any kind of object, but we train it exclusively on solids. Therefore, the model will produce solids, too. This means that the output surfaces will be closed and clean, which are properties that depend on global information. Here, this means that the model will require points on the entire object to determine if a part of the domain is on the inside. When the model encounters an

example in the real world that is missing a part, like a bust without its pedestal, the model will try to close the bottom, and the results will be less accurate. In contrast, other aspects, like preserving fine features, only require local information. Here, the most important prior is likely that the noise is roughly normally distributed without bias, so the original surface is in the to-be-estimated middle. These local and global priors have different advantages and disadvantages, which we explore in this thesis.

From the description of these concepts, we can extract three aspects that are essential for a good surface-reconstruction method:

1. An inconsistent inside-outside classification largely increases the reconstruction error in a global scope. Common artifacts are in the form of topological noise, i.e., small disconnected pieces floating around and unwanted cavities inside the object. Training on solid objects acts as a strong global prior, improving the accuracy.

2. Mistaking noise for signal is a major source of error in a local scope. Typical consequences are over-smoothed edges, bumpy surfaces, and missing fine structures. Training on many regions of various scales and with many scan defects is a strong local prior, which helps reconstruct small features of the input object.

3. Since resources are limited and the reconstruction must be finished in a reasonable time, a good method has to be efficient. Further, the quality and costs require a careful trade-off. While low-level optimizations are always possible, designing and tuning an algorithm towards efficiency is generally more promising. For instance, we can simplify the inside-outside classification when the bottom is always inside. As another example, while we need access to the raw data in the local scope, we can limit the data used and get by with simple representations.

These aspects shed light on possibilities for improvement: higher encoding quality for the input data on a global *and* local scale, and simplifications wherever possible. All in all, the overarching research question of this thesis is: **How can deep learning improve surface reconstruction?**
Or more precisely: **How can we achieve accurate surface reconstruction that works with all kinds of possibly imperfect point clouds and generalizes well to unseen data?**

1.3 Contributions

During this thesis, we contributed three methods for surface reconstruction from point clouds to the field. The first two target the high-quality reconstruction of a few solid objects. The third one aims to reconstruct huge landscapes quickly from sparse and local sub-samples.

1.3.1 Points2surf: Learning Implicit Surfaces from Point Clouds

Summary: A key step in any scanning-based asset creation workflow is to convert unordered point clouds to a surface. Classical methods (e.g., Poisson reconstruction) start to degrade in the presence of noisy and partial scans. Hence, deep learning based methods have recently been proposed to produce complete surfaces, even from partial scans. However, such data-driven methods struggle to generalize to new shapes with large geometric and topological variations. We present Points2Surf, a novel patch-based learning framework that produces accurate surfaces directly from raw scans without normals. Learning a prior over a combination of detailed local patches and coarse global information improves generalization performance and reconstruction accuracy. Our extensive comparison on both synthetic and real data demonstrates a clear advantage of our method over state-of-the-art alternatives on previously unseen classes (on average, Points2Surf brings down reconstruction error by 30% over SPR and by 270%+ over deep learning based SotA methods) at the cost of longer computation times and a slight increase in small-scale topological noise in some cases.

Individual Contributions: The first author, Philipp Erler, devised and implemented the method. The network architecture is heavily based on Paul Guerrero's PCP-Net [GKOM18] who strongly supported every development step. The main idea to use an SDF came from Niloy Mitra. Stefan Ohrhallinger contributed various ideas. Michael Wimmer was involved in high-level discussions and contributed feedback. All authors were involved in writing the paper.

Publication: Philipp Erler, Paul Guerrero, Stefan Ohrhallinger, Niloy J. Mitra, and Michael Wimmer. Points2surf: Learning Implicit Surfaces from Point Clouds. In *European Conference on Computer Vision*, pp. 108–124. Springer, 2020.

DOI: 10.1007/978-3-030-58558-7_7

URLs: Project Page, Source Code

Status 2025-12-30: ECCV 2020 Spotlight (top 25%), 285 citations, 504 GitHub Stars

1.3.2 PPSurf: Combining Patches and Point Convolutions for Detailed Surface Reconstruction

Summary: 3D surface reconstruction from point clouds is a key step in areas such as content creation, archaeology, digital cultural heritage and engineering. Current approaches either try to optimize a non-data-driven surface representation to fit the points, or learn a data-driven prior over the distribution of commonly occurring surfaces and how they correlate with potentially noisy point clouds. Data-driven methods enable robust handling of noise and typically either focus on a global or a local prior, which trade-off between robustness to noise on the global end and surface detail preservation on the local end. We propose PPSurf as a method that combines a global prior based on point convolutions and a local prior based on processing local point cloud patches. We show that this approach is robust to noise while recovering surface details more accurately than the current state-of-the-art.

Individual Contributions: The first author, Philipp Erler, devised and implemented the method in close cooperation with Lizeth Fuentes-Perez. Pedro Hermosilla and Paul Guerrero participated in the discussions and helped write the paper. Renato Pajarola and Michael Wimmer contributed in the high-level discussions and provided feedback.

Publication: Philipp Erler, Lizeth Fuentes-Perez, Pedro Hermosilla, Paul Guerrero, Renato Pajarola, and Michael Wimmer. PPSurf: Combining Patches and Point Convolutions for Detailed Surface Reconstruction. In *Computer Graphics Forum* (Vol. 43, No. 1, p. e15000, 2024).

DOI: 10.1111/cgf.15000

URLs: Project Page, Source Code

Status 2025-12-30: 6 citations, 60 GitHub Stars

1.3.3 LidarScout: Direct Out-of-Core Rendering of Massive Point Clouds

Summary: Large-scale terrain scans are the basis for many important tasks, such as topographic mapping, forestry, agriculture, and infrastructure planning. The resulting point cloud datasets are so massive in size that even basic tasks like viewing take hours to days of pre-processing in order to create level-of-detail structures that allow inspecting the dataset in their entirety in real time. In this paper, we propose a method that is capable of instantly visualizing massive country-scale scans with hundreds of billions of points. Upon opening the dataset, we first load a sparse subsample of points and initialize an overview of the entire point cloud, immediately followed by a surface reconstruction process to generate higher-quality, hole-free heightmaps. As users start navigating towards a region of interest, we continue to prioritize the heightmap construction process to the user's viewpoint. Once a user zooms in closely, we load the full-resolution point cloud data for that region and update the corresponding heightmap textures with the full-resolution data. As users navigate elsewhere, full-resolution point data that is no longer needed is unloaded, but the updated heightmap textures are retained as a form of medium level of detail. Overall, our method constitutes a form of direct out-of-core rendering for massive point cloud datasets (terabytes, compressed) that requires no preprocessing and no additional disk space.

Individual Contributions: The first author, Philipp Erler, devised and implemented the heightmap prediction. Lukas Herzberger wrote a prototype for loading the chunk points quickly and efficiently. Markus Schütz adapted his CUDA-based software rasterizer. Philipp and Markus wrote the paper and did the evaluation. Michael Wimmer contributed in the high-level discussions and provided feedback.

Publication: Philipp Erler, Lukas Herzberger, Michael Wimmer, and Markus Schütz. LidarScout: Direct Out-of-Core Rendering of Massive Point Clouds. In Aaron Knoll and Christoph Peters, editors, *High-Performance Graphics - Symposium Papers*. The Eurographics Association, 2025.

DOI: 10.2312/hpg.20251170

URLs: Project Page, Source Code

Status 2025-12-30: 20 GitHub Stars

CHAPTER 2

Background and Related Work

This thesis deals with various representations of 3D data and the conversions between those representations. The core task is the accurate and robust reconstruction of surfaces from sparse and noisy point clouds. A common tool for these operations is deep learning.

2.1 3D Representations

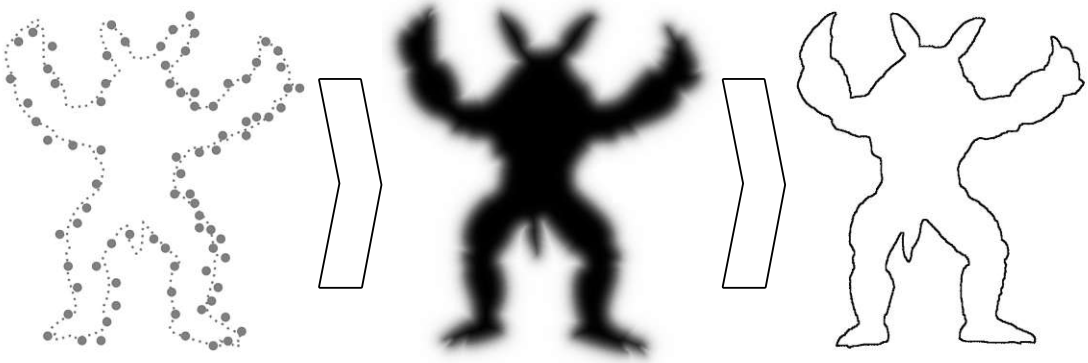


Figure 2.1: 3D representations for our reconstruction pipeline: point cloud, implicit surface, and explicit surface.

This thesis touches several kinds of 3D representations. The typical, SDF-based, data-driven surface-reconstruction pipeline is as follows. First, we feed subsamples of the input point clouds into a Neural Network (NN), which encodes them in feature vectors. Another part of the NN decodes these features and regresses a signed distance. Many of these predictions form an implicit surface, defined as a Signed-Distance Field (SDF) in a grid. Iso-surface extraction, like Marching Cubes, converts the SDF to a triangle mesh.

A key aspect of this thesis is to differentiate between local and global information. Local information captures fine details and properties that can be derived from a small region of the object, such as surface normals and curvature. Global information contains properties that may require seeing the entire object as a whole. Such global properties include topological characteristics like the number of connected components and holes, as well as the bounded volume. Sometimes, these scopes overlap. As an example, the inside/outside can be derived from the nearest surface normals, but having consistent normals on the whole object requires global information. This thinking in different scopes of information helps us estimate what an NN needs for its predictions and where bias may occur. Therefore, these aspects propagate into data-driven surface reconstruction with local and global priors.

2.1.1 Point Clouds

Point clouds are a simple yet widely used representation of 3D surfaces. They are easy to acquire nowadays. Various scanners exist, from consumer-grade handheld devices to expensive ones mounted on satellites. Another common source is taking photos and using photogrammetry. While all methods provide a set of points in 3D space, some offer additional information like surface normals and colors. However, they all can come with some defects, such as noise, varying sampling density, and missing regions. Since connectivity is usually missing, point clouds are hard to process and inefficient to render. Therefore, conversion to other representations is often required, some of which are the reconstruction methods in this thesis.

2.1.2 Implicit Representations

Implicit Surfaces store complex objects in a differentiable and continuous fashion. In such a representation, we assign every point in space a distance to the encoded object. All points having a distance of zero lie on the surface. Since we generally do not have an analytical representation of the object, we need to sample the space, generating a distance field. Usually, we truncate it and encode the inside/outside in the sign, meaning $+1$ is far inside and -1 is far outside. While the sign typically represents occupancy in solid objects, it could also be used to encode other spatial properties, such as visibility from the scanner. As an example, the sign can distinguish between known empty space and unobserved regions. This is particularly useful for scenes and objects that are not watertight or have multiple layers of thin structures.

For more efficiency, we can store the SDF sparsely in an octree or kD-tree. While SDFs are clean and smooth by definition, they are also inefficient for rasterization, which we need for fast rendering. Therefore, we often convert them to triangle meshes using iso-surface extraction methods, like Marching Cubes [LC87].

2.1.3 Explicit Representations

The dominant type of explicit representations is the triangle mesh, which is the cornerstone for fast rendering, since hardware acceleration is mostly targeting rasterization. For this, we store the vertices as a set of points and their connectivity as a tuple of vertex indices (triangle faces). While this representation is very efficient for encoding surfaces, it may contain defects that would be impossible in reality, such as self-intersections and non-manifold elements. Another important usage of explicit representations is geometry processing and modeling, which, however, may require conversion to specialized variants like halfedge, winged edge, or quad-dominant structures.

A different kind of explicit geometry representation is a heightmap. Here, we store height values in a raster image. This 2.5D data structure is very efficient for mostly flat data like large-scale landscapes. However, it cannot represent multiple surfaces at the same coordinate, which commonly happens with caves and buildings, and it is inaccurate at steep slopes.

2.2 Surface Reconstruction

Surface reconstruction aims to recover the original surface from given input samples. These samples are usually a point cloud with all the possible defects described in Section 2.1.1. The outputs are typically either explicit or implicit representations. Special cases like reconstruction to parametric surfaces for CAD objects exist, too. In general, surface reconstruction is an ill-posed problem since multiple surfaces may correspond equally well to the one point cloud.

Surface-reconstruction methods can be categorized in multiple ways. Interpolation techniques assume that the input points are accurate and need to be preserved. They only try to recover the connectivity on the original surface. Approximation techniques, on the other hand, assume that the points are erroneous and the original surface is only near them. Such methods fit a surface to the given samples. For this thesis, the most relevant surface-reconstruction techniques belong to the approximation type.

The relation between points and surface is hard to formulate mathematically. The most important formulation is based on Poisson equations, as in Poisson Surface Reconstruction [KBH06]. Optimization-based methods work similarly as they try to minimize an error defined by a mathematical relationship between points and the surface. While these techniques work well for many point clouds, they tend to fail in difficult cases.

A mathematical formulation may be difficult, but we can easily generate training data by inverting the problem, making surface reconstruction a good candidate for machine learning. We have huge repositories of 3D objects available, such as ABC [KMJ+19], Thingi10k [ZJ16], and Objaverse [DLW+23], we just need to sample point clouds from them. To ensure that these training examples are representative of real-world tasks, we require a good variety of object types and a realistic sampling process. Here, we separate the methods into data-driven and non-data-driven ones, focusing on the former.

2.2.1 Non-Data-Driven Surface Reconstruction

Here, we describe the most relevant classic and overfitting-based methods.

Berger et al. [BTS+17] present an in-depth survey of non-data-driven surface reconstruction until 2017. One of the first 3D reconstruction methods, BallPivot [BMR+99], runs from point to point in a point cloud, connecting them with edges. Scale space meshing [DMSL11] applies iterative mean curvature motion to smooth the points for meshing. It preserves multi-resolution features well. Ohrhallinger et al. propose a combinatorial method [OMW13] which compares favorably with previous methods such as Wrap [Ede03], TightCocone [DG03], and Shrink [Cha03], especially for sparse sampling and thin structures. However, these methods are not designed to process noisy point clouds. Another line of work deforms initial meshes [SLS+06, LLZM10] or parametric patches [WSS+19] to fit a noisy point cloud. These approaches, however, cannot change the topology and connectivity of the original meshes or patches, usually resulting in a different connectivity or topology than the ground truth. The most widely used approaches to reconstruct surfaces with arbitrary topology from noisy point clouds fit implicit functions to the point cloud and generate a surface as a level set of the function. Early work by Hoppe et al. introduced this approach [HDD+92], and since then, several methods have focused on different representations of the implicit functions, like Fourier coefficients [Kaz05], wavelets [MPS08], radial-basis functions [OBS05], or multi-scale approaches [OBS03, NOS09]. Alliez et al. [ACSTD07] use a PCA of 3D Voronoi cells to estimate gradients and fit an implicit function by solving an eigenvalue problem. This approach tends to over-smooth geometric detail. Poisson reconstruction [KBH06, KH13] is the current gold standard for non-data-driven surface reconstruction from point clouds, despite requiring normals. BallMerge [POEM24] uses Voronoi balls to recognize the inside and outside of large scans.

Recent optimization-based works suggest overfitting an NN on a single object by minimizing a loss function. They optimize the parameters of a neural network to predict the signed distance to the surface [AL20, SMB+20, AL21] directly from a single point cloud. In particular, Atzmon and Lipman [AL20] introduced this concept for unoriented point clouds. They optimized the parameters of the neural network with a sign-agnostic loss and a geometric initialization of its parameters. Gropp et al. [GYH+20] and Atzmon and Lipman [AL21] followed up on this work and included a gradient regularization in the loss. Later, Ma et al. [BZYSM21] introduced Neural-Pull, an optimization objective that uses the gradient of the optimized SDF directly to move the query points to the closest point in the input point cloud. In follow-up work, this approach was extended by incorporating a network to classify a point being on the surface or not [CHL23], and an additional loss that aligns the gradient direction between different level sets of the SDF [MZLH23]. In order to improve the quality of the final SDF, Yifan et al. [YWOSH20] and Zhou et al. [ZML+22] proposed to iteratively increase the input point cloud with points sampled from the optimized SDF in the previous iteration. A different approach was proposed by Peng et al. [PJL+21], based on a differentiable Poisson Surface Reconstruction operation that could be used for optimization-based or learned reconstructions. Different from

previous methods, the set of points in the surface is optimized through the differentiable reconstruction instead of a neural network representing the SDF. Lin et al. proposed a parametric Gauss formula for reconstruction [LXSW22], which has quadratic complexity in memory leading to prohibitive costs for larger point clouds. VIPSS by Huang et al. [HCJ19] formulates reconstruction as a constrained quadratic optimization problem. iPSR by Hou et al. [HWW+22] uses an iterative approach to Poisson reconstruction that improves the surface more and more, while removing the need to be given point normals. IsoPoisson by Xiao et al. [XSL+23] incorporates an isovalue constraint to the Poisson equation, which helps with consistent normal orientation and consequently improved reconstruction.

Non-data-driven methods are sensitive to noise, which is usually present in real 3D scans. In order to address this limitation to some extent, Wang et al. [WWW+23] propose Neural-IMLS, a non-data-driven method that regularizes the smoothness of surface normals using an MLP with limited capacity. While this produces smooth surfaces, it also loses some geometric detail due to this non-data-driven regularization. Noise to Noise Mapping by Baorui et al. [MLH23] focuses on the reconstruction of noisy point clouds in an unsupervised overfitting scheme. Additionally, these methods often require significant reconstruction times due to the optimization being performed for each shape individually, which can be a limiting factor for large scans.

None of the above methods make use of a prior that distills information about typical surface shapes from a large dataset. Hence, while they are very general, they fail to handle partial and/or noisy input. Data-driven surface reconstruction can solve this issue.

2.2.2 Data-Driven Surface Reconstruction

Here, we outline a short history of data-driven surface reconstruction before we go into the details of learning with global and local priors.

Several milestones in the history of deep learning contributed to making data-driven 3D surface reconstruction possible. The introduction of backpropagation [RHW86] enabled end-to-end optimization of neural networks. The success of ImageNet [KSH17] and its implementation AlexNet demonstrated the power of supervised learning on large-scale 2D datasets. Later, attention mechanisms [VSP+17] were invented, which largely increased the efficiency of deep learning, especially in high-dimensional problems. While this was focused on human languages, tasks on 3D data also benefit.

For learning on point clouds, the introduction of PointNet [QSMG17] was a crucial turning point, as it enabled learning, independent from the input ordering. Spatial Transformer Networks [JSZ+15] and subsequent works like PCPNet [GKOM18] pushed the limits of local geometric learning on raw point clouds.

A key concept in learning-based approaches is the notion of a *prior*, which encodes assumptions about the solution space, such as smoothness and topology. In deep learning,

these priors are implicitly captured by the architecture, training data, and optimization process. Global priors allow for learning shape semantics, but require lots of memory and usually do not generalize well. Local priors are the opposite, which is why many recent methods (ours included) try to leverage the advantages of both with a hybrid approach.

Global Priors

When this thesis started, the first data-driven surface reconstruction methods had just emerged. They were all learning global priors.

Early methods use voxel-based representations of the surfaces, with spatial data structures like octrees offsetting the cost of evaluating the NN for a full volumetric grid [TDB17, WSLT18].

AtlasNet [GFK+18] approximates the original surface with patches, primitives like subdivided quads. Scan2Mesh [DN19] reconstructs a coarse mesh, including vertices and triangles, from a scan with impressive robustness to missing parts. Both methods have the same drawbacks: the result is typically very coarse and not watertight or manifold, and does not apply to arbitrary new shapes.

A recent line of research has approached the problem of shape reconstruction in a data-driven manner by using a large dataset to learn a prior over the distribution of commonly occurring surfaces and how they correlate with the input point cloud. These approaches are typically fast and robust to noisy inputs compared to non-data-driven approaches. However, in such methods, the resulting reconstruction highly depends on the quality of such priors.

Several works have proposed to use a global prior to capture the distribution over full 3D object surfaces [CZ19, MON+19, PFS+19]. These methods define such a prior as a single latent vector representing the shape, which is then used as a condition in a fully connected network to decode the SDF of a given query point. Usually, the decoder is trained on large datasets with a point-cloud encoder [MON+19, CZ19]. However, Park et al. [PFS+19] proposed to train the decoder directly on such datasets and then optimize the latent vector to match the noisy point cloud during inference. Recently, Zhang et al. [ZTNW23] proposed to use richer global priors. They introduced an encoder-decoder network that encodes the input point cloud using attention modules into a set of latent vectors representing the shape, which are then used to predict the SDF for a set of query points using cross-attention modules.

Local Priors

Recently, several methods have been proposed to learn a prior of typical surface shapes from a large dataset. Early work was done by Sauerer et al. [LSJ+17], where a decision tree is trained to predict the absolute distance part of an SDF, but ground truth normals are still required to obtain the sign (inside/outside) of the SDF. Siddiqui et al. [STM+21] encoded the input point clouds in a set of latent scene patches. These latent vectors

are used to query a database of latent vectors from patches obtained from the training set. The obtained patches are then blended together using an attention mechanism. Ma et al. [BYSZ22] incorporated local priors by including a network pre-trained on a large number of surface patches which classifies a point as being on the surface or not. This network is used to guide an optimization process that learns the shape's SDF using another neural network. Jiang et al. [JSM+20] pre-trained an SDF encoder-decoder on a large dataset of object parts. Then, during the optimization process, only the latent codes of the different parts of the object are optimized. Chen et al. [CTFZ22] propose a dual contouring method learned on a small local prior.

Global and Local Priors

Since global and local priors provide complementary information about the shape, a common approach is to use a prior in the medium range using a hierarchical encoder-decoder network. These approaches reduce the input point cloud to a simplified representation, e.g., voxelization or subsampled point cloud, which is then enriched by the global information provided by the bottleneck of the encoder-decoder architecture.

Chibane et al. [CAPM20] and Peng et al. [PNM+20] proposed a 3DCNN encoder-decoder network to encode the sparse or noisy point cloud to later predict the SDF for an arbitrary point around the surface. Chibane et al. [CMPM20] extended this work to predict an unsigned distance field, which allowed them to represent complex open surfaces. Tang et al. [TLX+21] extended the work of Peng et al. [PNM+20] to include test-time optimization to improve out-of-distribution point clouds. Ummenhofer and Koltun [UK21] proposed a CNN that works directly on an Octree, from which the model was able to predict the SDF. Wang et al. [WLT22] also represented the input point cloud with an octree, from which they constructed a graph. This graph was further processed by a GCN encoder-decoder to generate an embedding for each octree node, from where the final SDF is predicted. Dai et al. [DDN20] instead used a 3D sparse encoder-decoder network to complete partial 3D scans and predict a complete SDF. Lionar et al. [LESP21] also developed an encoder-decoder network but used instead the projection of the input point cloud to a set of arbitrary 2D planes, from which the final SDF was predicted. Boulch and Marlet [BM22] recently proposed to use an encoder-decoder network that directly worked with points, avoiding discretization artifacts from voxel-based representations. Although all these methods work relatively well when compared with methods that use global or local priors alone, they often struggle to accurately capture fine local details of the shapes.

CHAPTER 3

Points2Surf: Learning Implicit Surfaces from Point Clouds

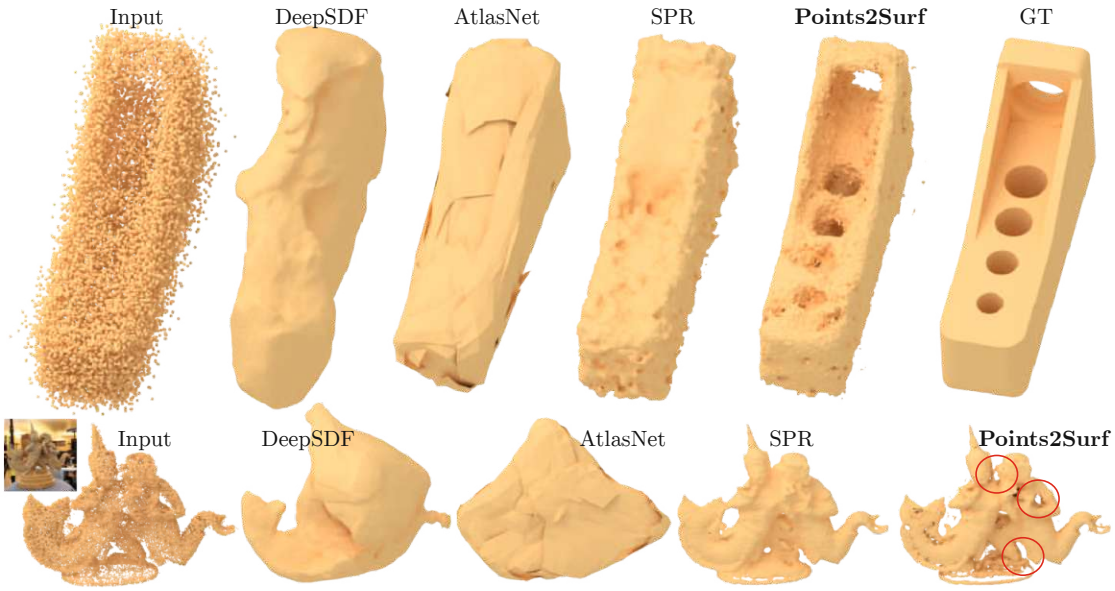


Figure 3.1: Points2Surf is a method to reconstruct an accurate implicit surface from a noisy point cloud. Combining local with global priors allows for detailed reconstruction while preserving the topology. This also makes Points2Surf the first learning-based to generalize well to unseen shapes.

This chapter is based on our publication 'Points2Surf: Learning Implicit Surfaces from Point Clouds' published at the European Conference on Computer Vision 2020 [EGO+20].

3.1 Motivation

As described previously (see Section 2.2.2), when we started this work, deep learning was just being adapted from images to 3D data like point clouds. PointNet [QSMG17] enabled deep learning on unordered point sets for classification and segmentation. Since deep learning showed remarkable results in this and other areas, we wanted to be among the first to try it for surface reconstruction.

Converting unstructured point clouds to surfaces is a key step of most scanning-based asset creation workflows, including games and AR/VR applications. While scanning technologies have become more easily accessible (e.g., depth cameras on smartphones, portable scanners), algorithms for producing a surface mesh remain limited. A good surfacing algorithm should be able to handle raw point clouds with noisy and varying sampling density, work with different surface topologies, and generalize across a large range of scanned shapes.

Screened Poisson Surface Reconstruction (SPR) [KH13] is the most commonly used method to convert an unstructured point cloud, along with its per-point normals, to a surface mesh. While the method is general, in the absence of any data-priors, SPR typically over-smoothes surfaces, can incorrectly close off holes and tunnels in noisy or non-uniformly sampled scans (see Figure 3.1), and further degenerates when per-point normal estimates are erroneous.

Hence, several data-driven alternatives [LSJ+17, DN19, PFS+19, GFK+18] have recently been proposed. These methods specialize in particular object categories (e.g., cars, planes, chairs), and typically regress a global latent vector from any input scan. This means that they try to compress the entire geometry of an object into a limited number of floats, typically a vector of 256-512 dimensions. Afterwards, the networks can decode a final shape (e.g., a collection of primitives [GFK+18] or a mesh [PFS+19]) from the estimated global latent vector. While such data-specific approaches handle noisy and partial scans, the methods do *not* generalize to new surfaces with varying shape and topology (see Figure 3.1).

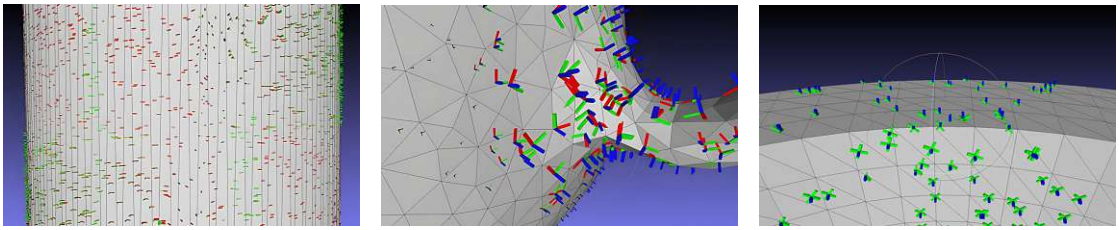
In contrast to such global approaches, the PointNet-based PCPNet [GKOM18] works on local inputs. It estimates the differentiable properties, like curvature and normals, of a query point from points within a certain radius. While global approaches are usually fast and robust against messy inputs, they rarely generalize well to unseen object classes. With these inspirations in mind, we set out to develop a data-driven surface-reconstruction method that combines local and global priors. However, finding a good representation for learning surfaces turned out to be difficult.

3.2 Preliminary Approaches

Before going into the details of the Points2Surf, we want to roughly outline several approaches we tried and why they failed to deliver useful results. These attempts

eventually led to the working idea of learning a signed distance from both global and local point cloud subsamples at the same time.

3.2.1 Principal Curvature Directions



(a) Principal curvature directions, as red and green bars.

(b) GT principal curvature directions on a teapot, scaled by confidence.

(c) Regression of principal curvature directions on a teapot.

Figure 3.2: Principal curvature directions GT data and prediction.

The common basis for the following approaches was PCPNet [GKOM18], a patch-based network for estimating normals and curvature on point clouds. We expected it to be able to learn principal curvature directions as well. Assuming accurate regressions, we could further fit a 4-RoSy field and then apply quad-dominant meshing. First, generating GT training data is not trivial. Figure 3.2a shows how CGAL's Monge via jet fitting is sensitive to the triangulation, producing unusable results for this cylinder with slender triangles. Isotropic remeshing with edge preservation solves this issue. The curvature directions are ambiguous in many cases, i.e., spherical and flat regions. Weighting the curvature directions by the difference between the magnitudes of the principal curvature directions should help the network focus on reliable training examples. An example is shown in Figure 3.2b. The model did not learn anything, even when we allowed negations and swapping k_1 with k_2. Figure 3.2c depicts a flawed prediction. There were several issues with this approach. Another factor for the failure might be the limited encoding fidelity of the underlying PointNet [QSMG17]. Later, Hernandez et al. [HBM23] learned an image-based representation of the mean curvature for generative hole filling on meshes. Harrison et al. [HBS25] recently showed that principal curvature directions as additional input help learning segmentation on meshes. While learning with principal curvature directions is done now, predicting them is still an open issue and might be worth investigating in the future.

3.2.2 Connectivity

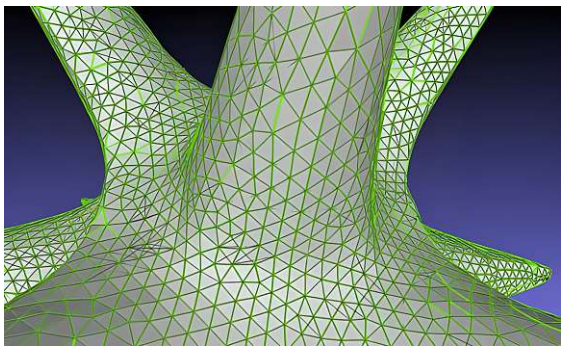


Figure 3.3: Connectivity GT (green) and predicted (black) of a given mesh (grey). Some edges are missing, and some others are superfluous.

After the failed attempt to learn principal curvature directions, we tried to learn connectivity. The idea was to learn which points should be connected to each other, like k-NN with adaptive k. Given a local subsample of the point cloud, the network outputs the connectivity probabilities in an adjacency matrix. Post-processing should resolve self-intersections, non-manifold structures, and ambiguous triangulations. We set our focus on reconstructing thin sheets, for which k-NN usually fails. While the network was able to learn connectivity to some degree (see Figure 3.3), it still failed in the same cases as k-NN. Triangulation ambiguity is likely one difficulty.

Later, Daroya et al. [DAC20] solved the triangulation ambiguity iteratively with two GRU-based RNNs and intricate handling of vertex ids. Similarly, Hanocka et al. [HMGCO20] use iterative shrink-wrapping of the convex hull for watertight reconstruction.

3.2.3 Geodesic Distances

Sticking with the connectivity idea, we thought that geodesic distances efficiently encode which points are close to each other on a manifold and should therefore be connected. Again, in a patch-based approach, the network should regress geodesic distances between all points in a local subsample. After clustering by distances, we essentially have a sparse distance matrix for each connected component. We can feed these distance matrices into Multidimensional Scaling (MDS) [Kru64] as an initial guess for mapping the 3D subsample to 2D. There, we can triangulate the points and transfer the same connectivity to 3D, which results in a triangle mesh. While the approach works in principle, it still produces a reconstruction despite all our optimizations and heuristics (see Figure 3.4).

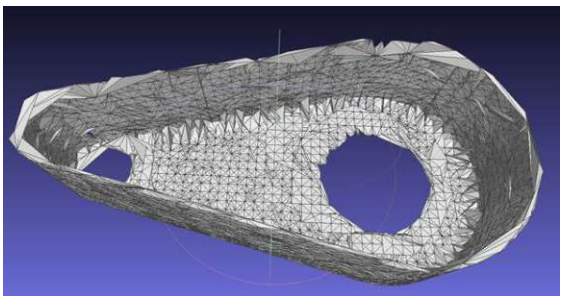


Figure 3.4: Reconstruction of a thin-walled object. The reconstruction suffers from various defects.

While several methods like Huberman et al. [HBK23] and Zhang et al. [ZHA+23] manage

to learn accurate geodesic distances, none can do clean surface reconstruction. Another related open issue is the consistent triangulation of overlapping patches in parallel.

3.2.4 Signed Distance Fields

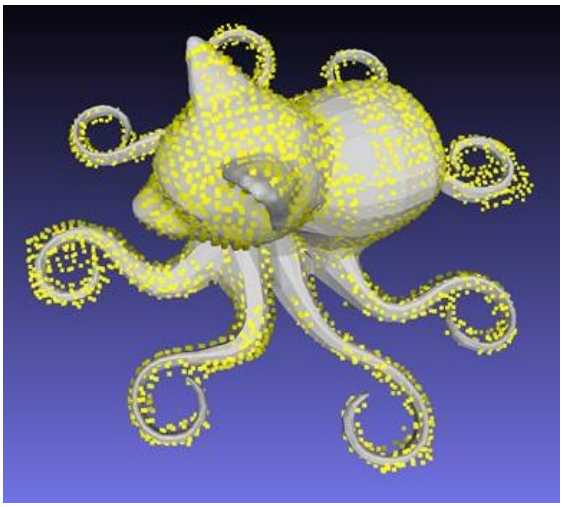


Figure 3.5: GT object in grey, samples on the implicit surface of Berger et al. [BTS+17] in yellow. The reconstruction benchmark produces inaccurate signed distances or takes too long.

The then recently published DeepSDF [PFS+19] inspired us to change the approach again. DeepSDF encodes the entire input point cloud in a single feature vector, which it decodes to an SDF. We knew that this limits the overall shape complexity and makes it dependent on object classes. By combining PCPNet's patch-based approach with their SDF learning, we hoped to preserve fine details and achieve good generalization.

The most difficult step in the journey towards Points2Surf was to generate accurate training data. After some confusion, we found that the reconstruction benchmark framework by Berger et al. [BTS+17] produces inaccurate signed distances or takes too long (see Figure 3.5). Similarly, PySDF [sxy] is very fast, but its signed distances' magnitude is also inaccurate. Lastly, Trimesh [Daw] is rather slow and requires lots of memory, but it is accurate. Combining PySDF's signs with Trimesh's distances proved successful, resulting in the still popular dataset of Points2Surf.

3.3 Overview

As a solution for the lack of generalisation in data-driven techniques (see Section 3.1), we present Points2Surf, a method that learns to produce implicit surfaces directly from raw point clouds. During test time, our method can reliably handle raw scans to reproduce fine-scale data features even from noisy and non-uniformly sampled point sets, works for objects with varying topological attributes, and generalizes to new objects (see Figure 3.1).

Our key insight is to decompose the problem into learning a global and a local function. For the global function, we learn the sign (i.e., inside or outside) of an implicit signed distance function, while, for the local function, we use a patch-based approach to learn absolute distance fields with respect to local point cloud patches. The global task is coarse (i.e., to learn the inside/outside of objects) and hence can be generalized across

significant shape variations. The local task exploits the geometric observation that a large variety of shapes can be expressed in terms of a much smaller collection of atomic shape patches [BGKS20], which generalizes across arbitrary shapes. We demonstrate that such a factorization leads to a simple, robust, and generalizable approach to learn an implicit signed distance field, from which a final surface is extracted using Marching Cubes [LC87].

We test our algorithms on a range of synthetic and real examples, compare on unseen classes against both classical (reduction in reconstruction error by 30% over SPR) and learning based strong baselines (reduction in reconstruction error by 470% over DeepSDF [PFS+19] and 270% over AtlasNet [GFK+18]), and provide ablation studies. We consistently demonstrate both qualitative and quantitative improvement over all the methods that can be applied directly to raw scans.

3.4 Method

Our goal is to reconstruct a watertight surface S from a 3D point cloud $P = \{p_1, ..., p_N\}$ that was sampled from the surface S through a noisy sampling process, like a 3D scanner. We represent a surface as the zero-set of a Signed Distance Function (SDF) f_S:

$$S = L_0(f_S) = \{x \in \mathbb{R}^3 \mid f_S(x) = 0\}. \tag{3.1}$$

Recent work [PFS+19, CZ19, MON+19] has shown that such an implicit representation of the surface is particularly suitable for neural networks, which can be trained as functionals that take as input a latent representation of the point cloud and output an approximation of the SDF:

$$f_S(x) \approx \tilde{f}_P(x) = s_\theta(x|z), \text{ with } z = e_\phi(P), \tag{3.2}$$

where z is a latent description of surface S that can be encoded from the input point cloud with an encoder e, and s is implemented by a neural network that is conditioned on the latent vector z. The networks s and e are parameterized by θ and ϕ, respectively. This representation of the surface is continuous, usually produces watertight meshes, and can naturally encode arbitrary topology. Different from non-data-driven methods like SPR [KH13], the trained network obtains a strong prior from the dataset it was trained on, that allows robust reconstruction of surfaces even from unreliable input, such as noisy and sparsely sampled point clouds. However, encoding the entire surface with a single latent vector imposes limitations on the accuracy and generality of the reconstruction, due to the limited capacity of the latent representation.

In this work, we factorize the SDF into the absolute distance f^d and the sign of the distance f^s, and take a closer look at the information needed to approximate each factor. To estimate the absolute distance $\tilde{f}^d(x)$ at a query point x, we only need a *local* neighborhood of the query point:

$$\tilde{f}^d_P(x) = s^d_\theta(x|z^d_x), \text{ with } z^d_x = e^d_\phi(\mathbf{p}^d_x), \tag{3.3}$$

where $\mathbf{p}_x^d \subset P$ is a local neighborhood of the point cloud centered around x. Estimating the distance from an encoding of a local neighborhood gives us more accuracy than estimating it from an encoding of the entire shape, since the local encoding z_x^d can more accurately represent the local neighborhood around x than the global encoding z. Note that in a point cloud without noise and sufficiently dense sampling, the single closest point $p^* \subset P$ to the query x would be enough to obtain a good approximation of the absolute distance. But since we work with noisy and sparsely sampled point clouds, using a larger local neighborhood increases robustness.

In order to estimate the sign $\tilde{f}^s(x)$ at the query point x, we need *global* information about the entire shape, since the interior/exterior of a watertight surface cannot be estimated reliably from a local patch. Instead, we take a global subsample of the point cloud P as input:

$$\tilde{f}_P^s(x) = \operatorname{sgn}(\tilde{g}_P^s(x)) = \operatorname{sgn}(s_\theta^s(x|z_x^s)), \text{ with } z_x^s = e_\psi^s(\mathbf{p}_x^s), \tag{3.4}$$

where $\mathbf{p}_x^s \subset P$ is a global subsample of the point cloud, ψ are the parameters of the encoder, and $\tilde{g}_P^s(x)$ are logits of the probability that x has a positive sign. Working with logits avoids discontinuities near the surface, where the sign changes. Since it is more important to have accurate information closer to the query point, we sample $\mathbf{p}_x^s$ with a density gradient that is highest near the query point and falls off with distance from the query point.

We found that sharing information between the two latent descriptions z_x^s and z_x^d benefits both the absolute distance and the sign of the SDF, giving us the formulation we use in Points2Surf:

$$(\tilde{f}_P^d(x), \tilde{g}_P^s(x)) = s_\theta(x|z_x^d, z_x^s), \text{ with } z_x^d = e_\phi^d(\mathbf{p}_x^d) \text{ and } z_x^s = e_\psi^s(\mathbf{p}_x^s). \tag{3.5}$$

We describe the architecture of our encoders and decoder, the training setup, and our patch sampling strategy in more detail in Section 3.4.1.

To reconstruct the surface S, we apply Marching Cubes [LC87] to a sample grid of the estimated SDF $\tilde{f}^d(x) * \tilde{f}^s(x)$. In Section 3.4.2, we describe a strategy to improve performance by only evaluating a subset of the grid samples.

3.4.1 Architecture and Training

Figure 3.6 shows an overview of our architecture. Our approach estimates the absolute distance $\tilde{f}_P^d(x)$ and the sign logits $\tilde{g}_P^s(x)$ at a query point based on two inputs: the query point x and the point cloud P.

Pointset sampling. The local patch $\mathbf{p}_x^d$ and the global subsample $\mathbf{p}_x^s$ are both chosen from the point cloud P based on the query point x. The set $\mathbf{p}_x^d$ is made of the n_d nearest neighbors of the query point (we choose $n_d = 300$ but also experiment with other values). Unlike a fixed radius, the nearest neighbors are suitable for query points with arbitrary

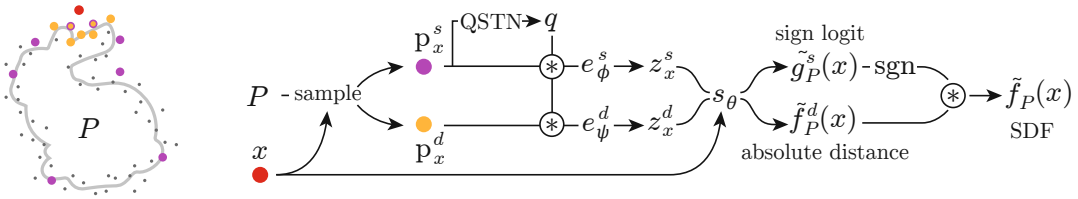


Figure 3.6: Points2Surf Architecture. Given a query point x (red) and a point cloud P (gray), we sample a local patch (yellow) and a coarse global subsample (purple) of the point cloud. These are encoded into two feature vectors that are fed to a decoder, which outputs a logit of the sign probability and the absolute distance of the SDF at the query point x.

distance from the point cloud. The global subsample $\mathbf{p}_x^s$ is sampled from P with a density gradient that decreases with distance from x:

$$\rho(p_i) = \frac{v(p_i)}{\sum_{p_j \in P} v(p_j)}, \text{ with } v(p_i) = \left[1 - 1.5 \frac{\|p_i - x\|_2}{\max_{p_j \in P} \|p_j - x\|_2} \right]_{0.05}^{1}, \tag{3.6}$$

where ρ is the sample probability for a point $p_i \in P$, v is the gradient that decreases with distance from x, and the square brackets denote clamping. The minimum value for the clamping ensures that some far points are taken and the subsample can represent a closed object. We sample n_s points from P according to this probability (we choose $n_s = 1000$ in our experiments).

Pointset normalization. Both $\mathbf{p}_x^d$ and $\mathbf{p}_x^s$ are normalized by centering them at the query point, and scaling them to unit radius. After running the network, the estimated distance is scaled back to the original size before comparing to the ground truth. Due to the centering, the query point is always at the origin of the normalized coordinate frame and does not need to be passed explicitly to the network. To normalize the orientation of both point subsets, we use a data-driven approach: a Quaternion Spatial Transformer Network (QSTN) [GKOM18] takes as input the global subset $\mathbf{p}_x^s$ and estimates a rotation represented as quaternion q that is used to rotate both point subsets. We take the global subset as input, since the global information can help the network with finding a more consistent rotation. The QSTN is trained end-to-end with the full architecture, without direct supervision for the rotation.

Encoder and decoder architecture. The local encoder e_ϕ^d, and the global encoder e_ψ^s are both implemented as PointNets [QSMG17], sharing the same architecture, but not the parameters. Following the PointNet architecture, a feature representation for each point is computed using 5 MLP layers, with a spatial transformer in feature space after the third layer. Each layer except the last one uses batch normalization and ReLU. The point feature representations are then aggregated into point set feature representations $z_x^d = e_\phi^d(\mathbf{p}_x^d)$ and $z_x^s = e_\psi^s(\mathbf{p}_x^s)$ using a channel-wise maximum. The

decoder s_θ is implemented as 4-layer MLP that takes as input the concatenated feature vectors z_x^d and z_x^s and outputs both the absolute distance $\tilde{f}^d(x)$ and sign logits $\tilde{g}^s(x)$.

Losses and training. We train our networks end-to-end to regress the absolute distance of the query point x from the watertight ground-truth surface S and classify the sign as positive (outside S) or negative (inside S). We assume that ground-truth surfaces are available during training for supervision. We use an L_2-based regression for the absolute distance:

$$\mathcal{L}^d(x, P, S) = \| \tanh(|\tilde{f}_P^d(x)|) - \tanh(|d(x, S)|) \|_2^2, \tag{3.7}$$

where $d(x, S)$ is the distance of x to the ground-truth surface S. The tanh function gives more weight to smaller absolute distances, which are more important for an accurate surface reconstruction. For the sign classification, we use the binary cross entropy H as loss:

$$\mathcal{L}^s(x, P, S) = H\Big(\sigma(\tilde{g}_P^s(x)),\ [f_S(x) > 0]\Big), \tag{3.8}$$

where σ is the logistic function to convert the sign logits to probabilities, and $[f_S(x) > 0]$ is 1 if x is outside the surface S and 0 otherwise. In our optimization, we minimize these two losses for all shapes and query points in the training set:

$$\sum_{(P,S)\in\mathcal{S}} \sum_{x\in\mathcal{X}_S} \mathcal{L}^d(x, P, S) + \mathcal{L}^s(x, P, S), \tag{3.9}$$

where $\mathcal{S}$ is the set of surfaces S and corresponding point clouds P in the training set and $\mathcal{X}_S$ the set of query points for shape S. Estimating the sign as a classification task instead of regressing the signed distance allows the network to express confidence through the magnitude of the sign logits, improving performance.

3.4.2 Surface Reconstruction

At inference time, we want to reconstruct a surface $\tilde{S}$ from an estimated SDF $\tilde{f}(x) = \tilde{f}^d(x) * \tilde{f}^s(x)$. A straight-forward approach is to apply Marching Cubes [LC87] to a volumetric grid of SDF samples. Obtaining a high-resolution result, however, would require evaluating a prohibitive number of samples for each shape. We observe that in order to reconstruct a surface, a Truncated Signed Distance Field (TSDF) is sufficient, where the SDF is truncated to the interval $[-\epsilon, \epsilon]$ (we set ϵ to three times the grid spacing in all our experiments). We only need to evaluate the SDF for samples that are inside this interval, while samples outside the interval merely need the correct sign. We leave samples with a distance larger than ϵ to the nearest point in P blank, and in a second step, we propagate the signed distance values from non-blank to blank samples, to obtain the correct sign in the truncated regions of the TSDF. We iteratively apply a box filter of size 3^3 at the blank samples until convergence. In each step, we update initially unknown samples only if the filter response is greater than a user-defined confidence threshold (we use 13 in our experiments). After each step, the samples are set to -1 if the filter response was negative or to +1 for a positive response.

3.5 Results

We compare our method to SPR as the gold standard for non-data-driven surface reconstruction and to two state-of-the-art data-driven surface reconstruction methods. We provide both qualitative and quantitative comparisons on several datasets in Section 3.5.2, perform an ablation study in Section 3.5.3, and provide detailed timings in Section 3.5.4

3.5.1 Datasets

Figure 3.7: Dataset examples. Examples of the ABC dataset and its three variants are shown on the left, examples of the famous dataset and its five variants on the right.

We train and evaluate on the ABC dataset [KMJ+19] which contains a large number and variety of high-quality CAD meshes. We pick 4950 clean watertight meshes for training and 100 meshes for the validation and test sets. Note that each mesh produces a large number of diverse patches as training samples.

Operating on local patches also allows us to generalize better, which we demonstrate on two additional test-only datasets: a dataset of 22 diverse meshes which are well-known in geometry processing, such as the Utah teapot and the Stanford Bunny, which we call the Famous dataset, and 3 Real scans of complex objects used in several denoising papers [WKZ+16, RLBG+19].

Examples from each dataset are shown in Figure 3.7. The ABC dataset contains predominantly CAD models of mechanical parts, while the Famous dataset contains more organic shapes, such as characters and animals. Since we train on the ABC dataset, the Famous dataset serves to test the generalizability of our method versus baselines.

Pointcloud sampling. As a pre-processing step, we center all meshes at the origin and scale them uniformly to fit within the unit cube. To obtain point clouds P from the meshes S in the datasets, we simulate scanning them with a time-of-flight sensor from random viewpoints using BlenSor [GKUP11]. BlenSor realistically simulates various types of scanner noise and artifacts such as backfolding, ray reflections, and per-ray noise. We scan each mesh in the Famous dataset with 10 scans and each mesh in the ABC dataset with a random number of scans, between 5 and 30. For each scan, we place the scanner at a random location on a sphere centered at the origin, with the radius chosen randomly in $U[3L, 5L]$, where L is the largest side of the mesh bounding box. The scanner is oriented to point at a location with small random offset from the origin, between $U[-0.1L, 0.1L]$ along each axis, and rotated randomly around the view direction. Each scan has a resolution of 176×144, resulting in roughly 25k points, minus some

points missing due to simulated scanning artifacts. The point clouds of multiple scans of a mesh are merged to obtain the final point cloud.

Dataset variants. We create multiple versions of both the ABC and FAMOUS datasets, with varying amount of per-ray noise. This Gaussian noise added to the depth values simulates inaccuracies in the depth measurements. For both datasets, we create a noise-free version of the point clouds, called *ABC no-noise* and *FAMOUS no-noise*. Also, we make versions with strong noise (standard deviation is $0.05L$) called *ABC max-noise* and *FAMOUS max-noise*. Since we need varying noise strength for the training, we create a version of ABC where the standard deviation is randomly chosen in $U[0, 0.05L]$ (*ABC var-noise*), and a version with a constant noise strength of $0.01L$ for the test set (*FAMOUS med-noise*). Additionally we create sparser (5 scans) and denser (30 scans) point clouds in comparison to the 10 scans of the other variants of FAMOUS. Both variants have a medium noise strength of $0.01L$. Additionally, we show a comparison with scanned objects from the THINGI10K dataset using the same variants as FAMOUS. We take 100 meshes that are tagged with 'scan' or 'sculpture'. These objects are mostly animals, humans and faces, many of them realistic, some artistic. The training set uses the *ABC var-noise* version, all other variants are used for evaluation only.

Query points. The training set also contains a set $\mathcal{X}_S$ of query points for each (point cloud, mesh) pair $(P, S) \in \mathcal{S}$. Query points closer to the surface are more important for the surface reconstruction and more difficult to estimate. Hence, we randomly sample a set of 1000 points on the surface and offset them in the normal direction by a uniform random amount in $U[-0.02L, 0.02L]$. An additional 1000 query points are sampled randomly in the unit cube that contains the surface, for a total of 2000 query points per mesh. During training, we randomly sample a subset of 1000 query points per mesh in each epoch.

3.5.2 Comparison to Baselines

We compare our method to recent data-driven surface reconstruction methods, AtlasNet [GFK+18] and DeepSDF [PFS+19], and to SPR [KH13], which is still the gold standard in non-data-driven surface reconstruction from point clouds. Both AtlasNet and DeepSDF represent a full surface as a single latent vector that is decoded into either a set of parametric surface patches in AtlasNet, or an SDF in DeepSDF. In contrast, SPR solves for an SDF that has a given sparse set of point normals as gradients, and takes on values of 0 at the point locations. We use the default values and training protocols given by the authors for all baselines (more details in the Supplementary) and re-train the two data-driven methods on our training set. We provide SPR with point normals as input, which we estimate from the input point cloud using the recent PCPNet [GKOM18].

For DeepSDF, we followed the method in the original paper with minor adaptations. For the training set, we take our query points and the corresponding Signed Distances (SD) as GT SDF samples. For the test sets, we take the point clouds from our dataset.

For each point, we generate 2 SDF samples, one in positive and one in negative normal direction, with random offset. The normals are taken from the ground truth face closest to the sample. The corresponding SDs are the +- normal offset. We add 20% random samples from the unit cube with GT SD.

Error metric. To measure the reconstruction error of each method, we sample both the reconstructed surface and the ground-truth surface with 10k points and compute the Chamfer distance [BTBW77, FSG17] between the two point sets:

$$d_{\text{ch}}(A,B) := \frac{1}{|A|}\sum_{p_i \in A} \min_{p_j \in B} \|p_i - p_j\|_2^2 \; + \frac{1}{|B|}\sum_{p_j \in B} \min_{p_i \in A} \|p_j - p_i\|_2^2, \tag{3.10}$$

where A and B are point sets sampled on the two surfaces. The Chamfer distance penalizes both false negatives (missing parts) and false positives (excess parts).

Quantitative and qualitative comparison. A quantitative comparison of the reconstruction quality is shown in Tables 3.1 and 3.2. Figures 3.8, 3.9 and 3.10 compare a few reconstructions qualitatively. All methods were trained on the training set of the *ABC var-noise* dataset, which contains predominantly mechanical parts, while the more organic shapes in the Famous dataset test how well each method can generalize to novel types of shapes.

Table 3.1: Comparison of reconstruction errors. We show the Chamfer distance between reconstructed and ground-truth surfaces averaged over all shapes in a dataset. Both the absolute value of the error multiplied by 100 (abs.), and the error relative to Point2Surf (rel.) are shown to facilitate the comparison. Our method consistently performs better than the baselines, due to its strong and generalizable prior.

	DeepSDF		AtlasNet		SPR		Points2Surf	
	abs.	rel.	abs.	rel.	abs.	rel.	abs.	rel.
ABC no-noise	8.41	4.68	4.69	2.61	2.49	1.39	**1.80**	**1.00**
ABC var-noise	12.51	5.86	4.04	1.89	3.29	1.54	**2.14**	**1.00**
ABC max-noise	11.34	4.11	4.47	1.62	3.89	1.41	**2.76**	**1.00**
Famous no-noise	10.08	7.14	4.69	3.33	1.67	1.18	**1.41**	**1.00**
Famous med-noise	9.89	6.57	4.54	3.01	1.80	1.20	**1.51**	**1.00**
Famous max-noise	13.17	5.23	4.14	1.64	3.41	1.35	**2.52**	**1.00**
Famous sparse	10.41	5.41	4.91	2.55	2.17	1.12	**1.93**	**1.00**
Famous dense	9.49	7.15	4.35	3.28	1.60	1.21	**1.33**	**1.00**
average	10.66	5.77	4.48	2.49	2.54	1.30	**1.92**	**1.00**

Table 3.2: Additional quantitative comparison of reconstruction errors on the Thingi10k dataset.

	DeepSDF		AtlasNet		SPR		Points2Surf	
	abs.	rel.	abs.	rel.	abs.	rel.	abs.	rel.
Thingi10k no-noise	9.16	6.48	5.29	3.74	1.78	1.26	**1.41**	**1.00**
Thingi10k med-noise	8.83	5.99	5.19	3.52	1.81	1.23	**1.47**	**1.00**
Thingi10k max-noise	12.28	4.68	4.90	1.87	3.23	1.23	**2.62**	**1.00**
Thingi10k sparse	9.56	4.54	5.64	2.68	2.35	1.12	**2.11**	**1.00**
Thingi10k dense	8.35	6.19	5.02	3.72	1.57	1.16	**1.35**	**1.00**
average	9.64	5.58	5.21	3.11	2.15	1.20	**1.79**	**1.00**

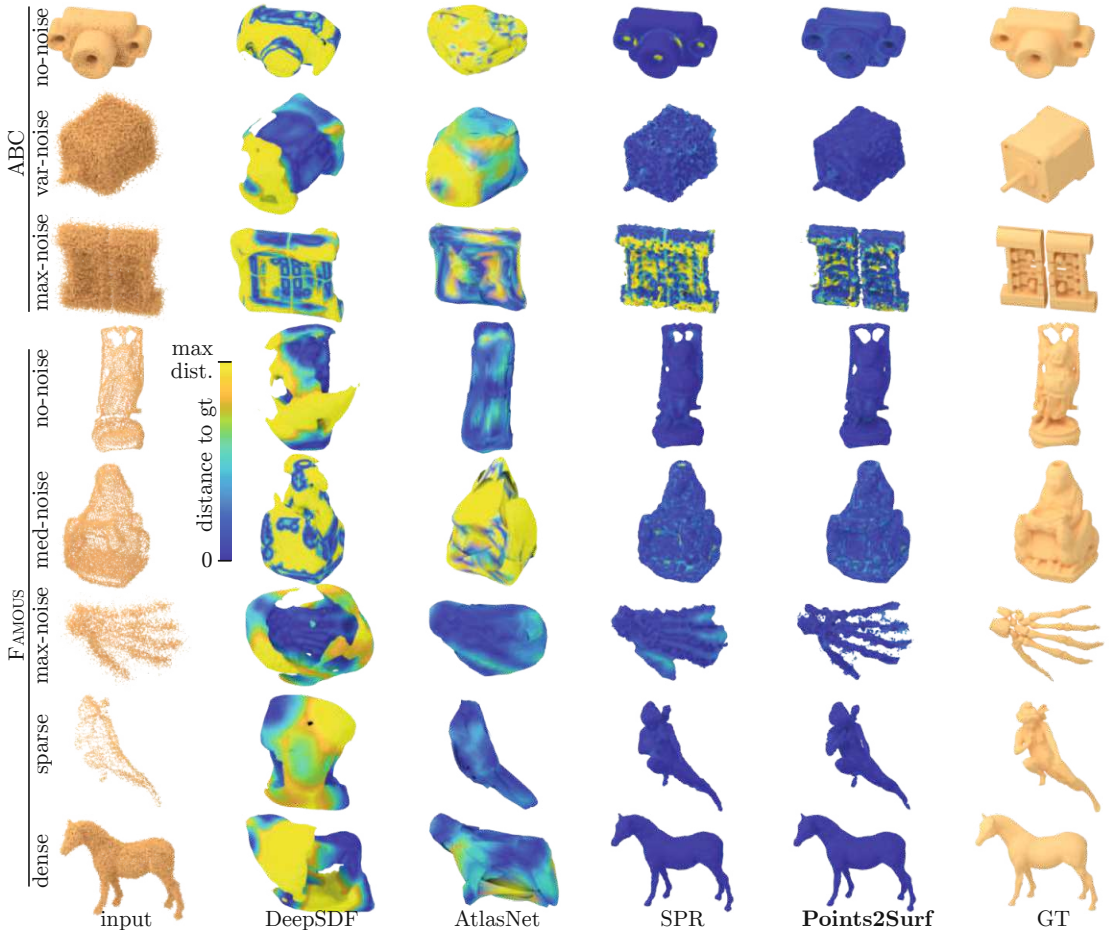


Figure 3.8: Qualitative comparison of surface reconstructions. We evaluate one example from each dataset variant with each method. Colors show the distance of the reconstructed surface to the ground-truth surface.

Both DeepSDF and AtlasNet use a global shape prior, which is well suited for a dataset with high geometric consistency among the shapes, like cars in ShapeNet, but struggles with the significantly larger geometric and topological diversity in our datasets, reflected in a higher reconstruction error than SPR or Points2Surf. This is also clearly visible in Figure 3.8, where the surfaces reconstructed by DeepSDF and AtlasNet appear over-smoothed and inaccurate.

In SPR, the full shape space does not need to be encoded into a prior with limited capacity, resulting in a better accuracy. But this lack of a strong prior also prevents SPR from robustly reconstructing typical surface features, such as holes or planar patches (see Figures 3.1 and 3.8).

Points2Surf learns a prior of local surface details, instead of a prior for global surface shapes. This local prior helps recover surface details like holes and planar patches more

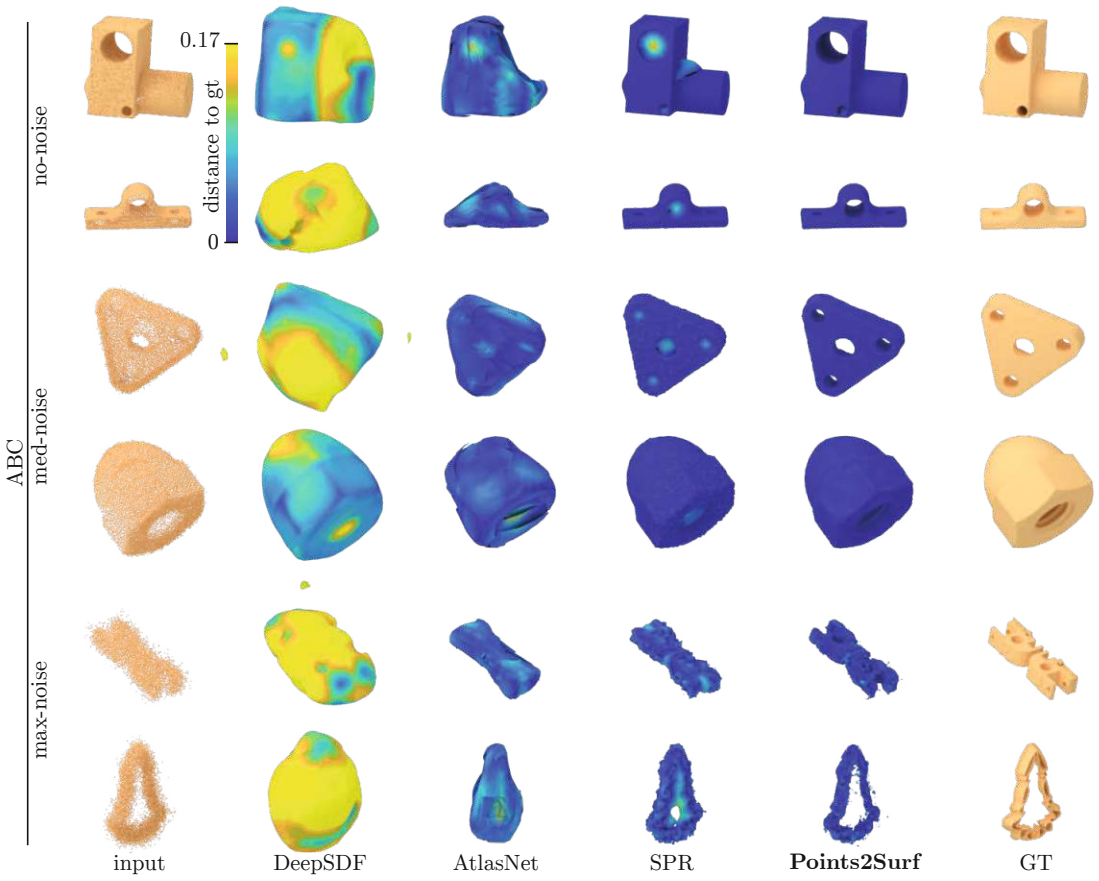


Figure 3.9: Additional qualitative comparison of surface reconstructions on the ABC dataset.

robustly, improving our accuracy over SPR. Since there is less variety and more consistency in local surface details compared to global surface shapes, our method generalizes better and achieves a higher accuracy than the data-driven methods that use a prior over the global surface shape.

Generalization. A comparison of our generalization performance against AtlasNet and DeepSDF shows an advantage for our method. In Table 3.1, we can see that the error for DeepSDF and AtlasNet increases more when going from the ABC dataset to the FAMOUS dataset than the error for our method. This suggests that our method generalizes better from the CAD models in the ABC dataset set to the more organic shapes in the FAMOUS dataset.

Topological Quality. Figure 3.11 shows examples of geometric detail that benefits from our prior. The first example shows that small features such as holes can be recovered from a very weak geometric signal in a noisy point cloud. Concavities, such as the space

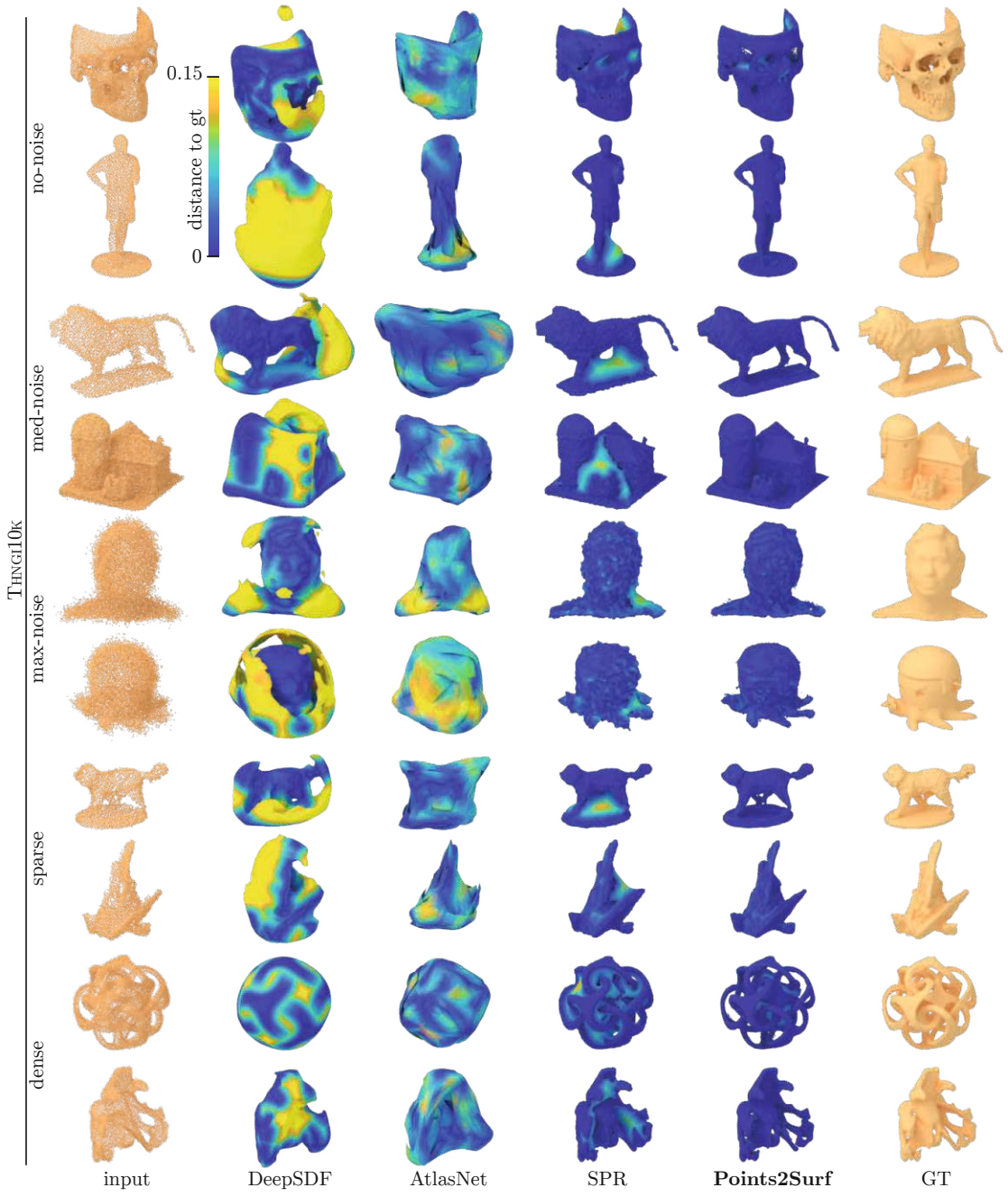


Figure 3.10: Additional qualitative comparison of surface reconstructions on the ThingI10K dataset.

between the legs of the Armadillo, and fine shape details like the Armadillo's hand are also recovered more accurately in the presence of strong noise. In the heart example, the concavity makes it hard to estimate the correct normal direction based on only a local neighborhood, which causes SPR to produce artifacts. In contrast, the global information we use in our patches helps us estimate the correct sign, even if the local neighborhood is misleading.

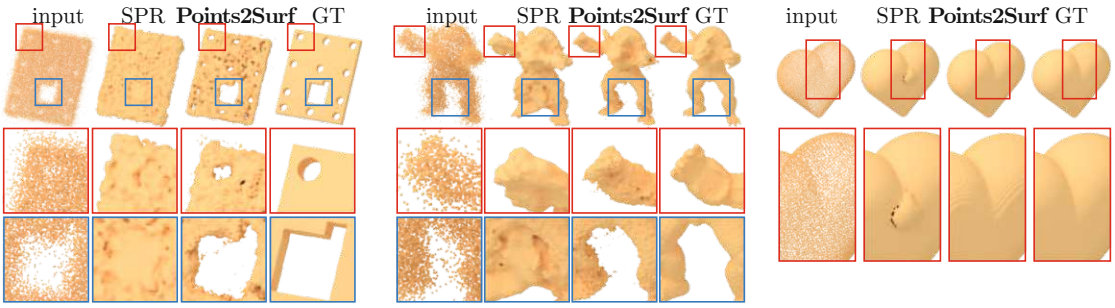


Figure 3.11: Comparison of reconstruction details. Our learned prior improves the reconstruction robustness for geometric detail compared to SPR.

Effect of Noise. Examples of reconstructions from point clouds with increasing amounts of noise are shown in Figure 3.12. Our learned prior for local patches and our coarse global surface information makes it easier to find small holes and large concavities. In the medium noise setting, we can recover the small holes and the large concavity of the surface. With maximum noise, it is very difficult to detect the small holes, but we can still recover the concavity.

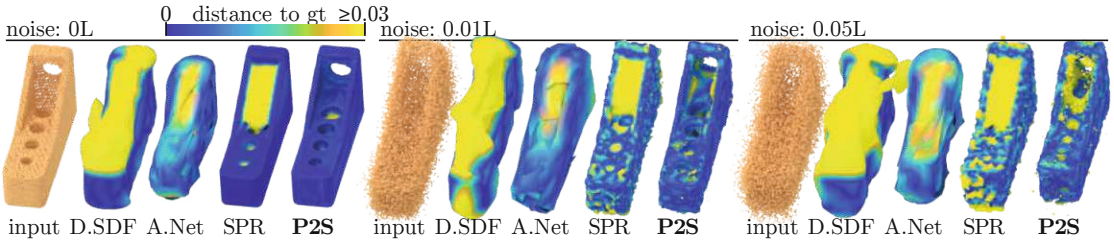


Figure 3.12: Effect of noise on our reconstruction. DeepSDF (D.SDF), AtlasNet (A.Net), SPR and Point2Surf (P2S) are applied to increasingly noisy point clouds. Our patch-based data-driven approach is more accurate than DeepSDF and AtlasNet, and can more robustly recover small holes and concavities than SPR.

Real-world data. The real-world point clouds in Figure 3.1 bottom and Figure 3.13 bottom both originate from a multi-view dataset [WKZ+16] and were obtained with a plane-sweep algorithm [Col96] from multiple photographs of an object. We additionally remove outliers using the recent PointCleanNet [RLBG+19]. Figure 3.13 top was obtained by the authors through an SfM approach. DeepSDF and AtlasNet do not generalize well

to unseen shape categories. SPR performs significantly better but its smoothness prior tends to over-smooth shapes and close holes. Points2Surf better preserves holes and details, at the cost of a slight increase in topological noise. Our technique also generalizes to unseen point-cloud acquisition methods.

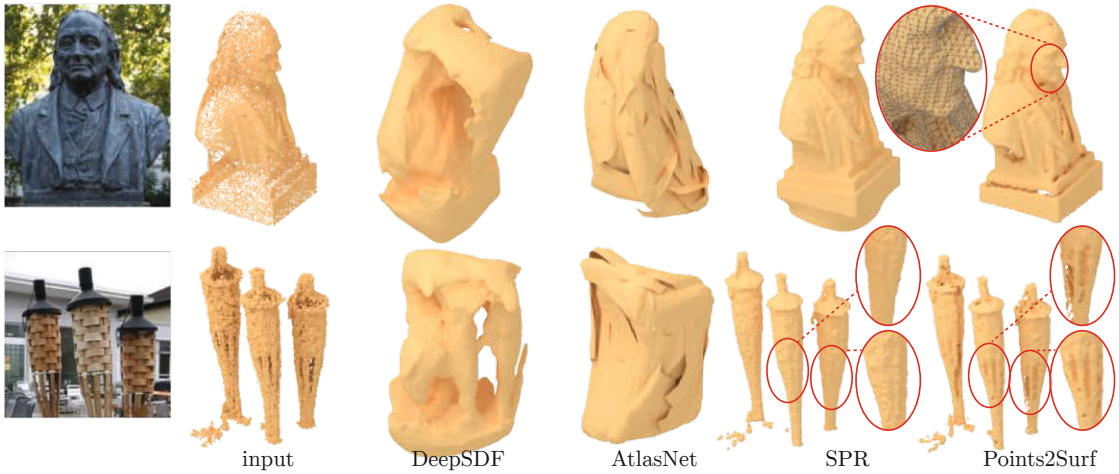


Figure 3.13: Reconstruction of real-world point clouds. Snapshots of the real-world objects are shown on the left. DeepSDF and AtlasNet do not generalize well, resulting in inaccurate reconstructions, while the smoothness prior of SPR results in loss of detail near concavities and holes. Our data-driven local prior better preserves these details.

3.5.3 Ablation Study

We evaluate several design choices in (e_{vanilla}) using an ablation study, as shown in Table 3.3. We evaluate the number of nearest neighbors $k = 300$ that form the local patch by decreasing and increasing k by a factor of 4 (k_{small} and k_{large}), effectively halving and doubling the size of the local patch. A large k performs significantly worse because we lose local detail with a larger patch size. A small k still works reasonably well, but gives a lower performance, especially with strong noise. We also test a fixed radius for the local patch, with three different sizes ($r_{\text{small}} := 0.05L$, $r_{\text{med}} := 0.1L$ and $r_{\text{large}} := 0.2L$). A fixed patch size is less suitable than nearest neighbors when computing the distance at query points that are far away from the surface, giving a lower performance than the standard nearest neighbor setting.

The next variant is using a single shared encoder (e_{shared}) for both the global subsample $\mathbf{p}_x^s$ and the local patch $\mathbf{p}_x^d$, by concatenating the two before encoding them. The performance of e_{shared} is competitive, but shows that using two separate encoders increases performance.

Omitting the QSTN ($e_{\text{no\_QSTN}}$) speeds-up the training by roughly 10% and yields slightly better results. The reason is probably that our outputs are rotation-invariant in contrast to the normals of PCPNet.

Using a uniform global subsample in e_{uniform} increases the quality over the distance-dependent sub-sampling in e_{vanilla}. This uniform subsample preserves more information about the far side of the object, which benefits the inside/outside classification.

Due to resource constraints, we trained all models in Table 3.3 for 50 epochs only. For applications where speed, memory and simplicity is important, we recommend using a shared encoder without the QSTN and with uniform sub-sampling.

Table 3.3: Ablation Study. We compare POINTS2SURF (e_{vanilla}) to several variants and show the Chamfer distance relative to POINTS2SURF. Please see the text for details.

	r_{small}	r_{med}	r_{large}	k_{small}	k_{large}	e_{shared}	$e_{\text{no\_QSTN}}$	e_{uniform}	e_{vanilla}
ABC var-noise	1.12	1.07	1.05	1.08	1.87	1.02	0.94	0.91	**1.00**
FAMOUS no-noise	1.09	1.08	1.17	1.05	8.10	1.06	0.97	0.96	**1.00**
FAMOUS med-noise	1.06	1.05	1.10	1.04	7.89	1.05	0.97	0.97	**1.00**
FAMOUS max-noise	1.07	1.19	1.13	1.09	1.79	1.01	1.05	0.89	**1.00**
average	1.08	1.11	1.11	1.07	4.14	1.03	0.99	0.92	**1.00**

3.5.4 Timings

We compare the wall-clock times of POINTS2SURF vanilla to the baselines. Because AtlasNet is very fast, we show the mean of 3 runs. We take the times of reconstructing the 100 shapes of the ABC med-noise test set using 16 worker processes. With this, we can show a fair comparison that takes parallelization, loading times and different bottlenecks into account. We ran the timings on a consumer-grade PC with a Ryzen 7 3600X, 64 GB DDR4 RAM and a GTX 1070. The mean times per shape in seconds are: POINTS2SURF 712.1, DeepSDF 199.5, SPR 157.5, AtlasNet 0.1. Since DeepSDF and SPR need normals, we included 156.5 seconds for the normal estimation using PCPNet.

The heuristic and sign propagation allows us to reduce the number of inferred query points to 1.67% of a full grid in ABC var-noise. Assuming linear scaling, the inference time is reduced from almost 12 hours to 11.5 minutes. This speed-up comes at the cost of 19.2 seconds per shape. The heuristic is less efficient with noise. For the ABC no-noise, the number of query points is reduced to 0.77% and for ABC max-noise to 2.33%.

We trained a smaller version of e_{uniform} with only about 15% parameters. The mini version reduces the reconstruction time by roughly 68% compared to e_{vanilla} (from 11.5 to 3.7 minutes per mesh), at the cost of an increase in the reconstruction error of roughly 55% on ABC var-noise (Chamfer distance from 150.6 for e_{vanilla} to 234.3 for the small version of e_{uniform}).

3.6 Conclusion

We have presented POINTS2SURF as a method for surface reconstruction from raw point clouds. Our method reliably captures both geometric and topological details, and generalizes to unseen shapes more robustly than current methods.

This work gives us a puzzle piece to answer the research question, 'How can deep learning improve surface reconstruction?' Estimating signed distances from local and global information improves surface reconstruction. While the local subsample conveys fine details, the global one is important for a consistent occupancy.

A major limitation of Points2Surf is the noisy reconstruction, leading to many surface bumps and sometimes floating pieces. One relevant reason is the randomness of neighboring patches, which is introduced by the subsampling for the global branch. In the next chapter, we explore various noise sources and try to eliminate them. Key elements for this are a more robust global encoding and a better system for selecting query points and propagating their results.

CHAPTER 4

PPSurf: Combining Patches and Point Convolutions for Detailed Surface Reconstruction

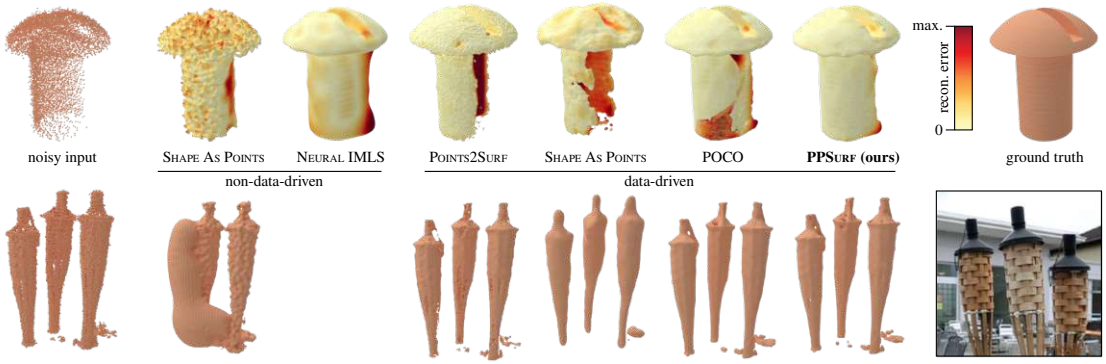


Figure 4.1: PPSurf is a method for reconstructing surfaces from noisy point clouds. Unlike previous methods, our approach combines two strong data-driven priors, one prior over local surface details, and a second prior over the coarse shape of larger surface regions. This makes PPSurf robust to noise, while reconstructing surface detail better than current methods.

This chapter is based on our publication 'PPSurf: Combining Patches and Point Convolutions for Detailed Surface Reconstruction' published in Computer Graphics Forum 2024 [EFPH+24].

4.1 Motivation

As discussed in Chapter 3.6, Points2Surf is still too sensitive to noise, resulting in wrong cavities and small disconnected artifacts. As before, we focus on high-quality reconstruction of single to a few solid objects. Equal to Points2Surf, the input point clouds may come from any kind of scanner or algorithmic source, possibly with noise, varying sampling density, and missing regions. Only 3D coordinates are required, no normals or other attributes. However, we assume that the point clouds are properly registered and mostly free from outliers. With these inputs, we aim to reconstruct meshes clean enough for downstream applications in content creation, archaeology, digital cultural heritage, and engineering.

Due to the large practical interest, surface reconstruction has moved forward significantly since the development of Points2Surf. Both data-driven and non-data-driven methods have advanced, where *optimization-based techniques* have joined the latter as a new branch. This research direction attempts to optimize surface representations by overfitting a model to a single input, e.g., in Shape-as-Points[PJL+21] and NeuralPull [BZYSM21]. They resolve the ambiguity of the ill-posed problem of reconstruction, but are still susceptible to deteriorating conditions of the input points, like the other non-data-driven methods.

On the data-driven side, general developments in deep learning have arrived in surface reconstruction and increased the efficiency and quality of models in various ways, most importantly, attention mechanisms [VSP+17]. Further, mixing global and local priors has become more common. The priors in data-driven methods can range from *global*, where the prior captures a distribution over full 3D object surfaces, to *local*, where the prior captures the distribution over local surface patches. Global priors are the least susceptible to noise and missing points, but have limited capability to capture fine local details. Local priors, on the other hand, can capture such fine details accurately but are more susceptible to strong noise and missing points. Existing methods mostly focus their prior on a small range in this global-local spectrum. For example, DeepSDF [PFS+19] uses a global prior, Points2Surf [EGO+20] mostly focuses on a local prior, while POCO's point convolutions [BM22] learn a prior in the global to medium range that is reasonably robust to deteriorating conditions, but the method still struggles to accurately capture local detail.

In summary, our Points2Surf has opened the door to high-quality surface reconstruction independent of object classes. Various advances show great potential for further improvement, especially for the key limitations of Points2Surf: the noisy topology and bumpy surfaces. As described before (Section 3.6), we suspect the random sampling to cause noise and the PointNet to achieve only a weak global encoding. Further, the query heuristic and sign propagation sometimes create large artifacts. While working on Points2Surf, we developed several ideas for improvements, most of which failed, but a few were successful.

4.2 Preliminary Approaches

While preparing the publication for POINTS2SURF, it became clear that the heuristic for the query point selection with sign-propagation can create artifacts like the blow-up in Figure 4.2. Further, the reconstruction introduces lots of noise, as shown in Figure 4.3. So we tried improving upon POINTS2SURF regarding subsampling and global encoding, which turned out surprisingly difficult. Here are the most interesting approaches we tried before finding the one that worked and resulted in PPSURF, which combines point convolution and attention with our proven local-global network architecture.

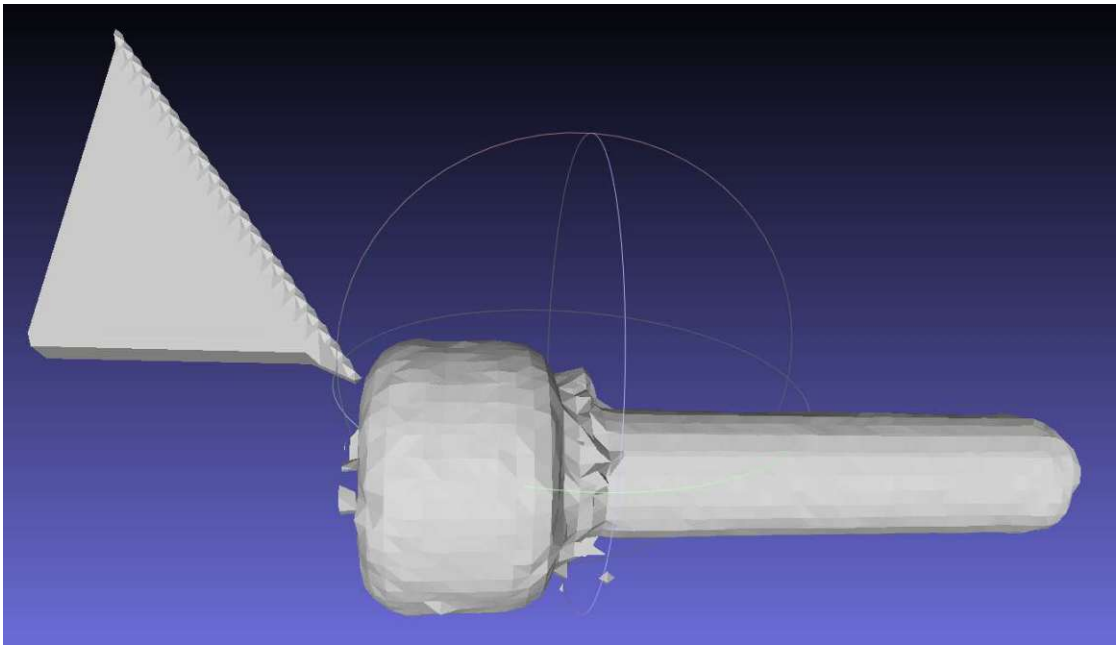


Figure 4.2: Reconstruction with blow-up. The sign-propagation of POINTS2SURF can sometimes create large artifacts.

4.2.1 Hierarchical Reconstruction

Our first attempt at overcoming these issues was to estimate edge intersections instead of signed distances, essentially moving the learned part further into Marching Cubes. Figure 4.4a shows a 2D example of such edge intersections. This would require no global information if we were able to make the occupancy prediction consistent with the neighboring cells. While the approach worked in principle, the reconstructions were still very noisy and no improvement over POINTS2SURF. Liao et al. [LDG18] proposed a similar approach with their differentiable approximation of Marching Cubes, but worked on the entire point cloud at once, missing out on the advantages of patch-based reconstruction. Later, Chen et al. [CZ21] solve these shortcomings with their own face tessellations but work on an SDF instead of a point cloud. A patch-based end-to-end

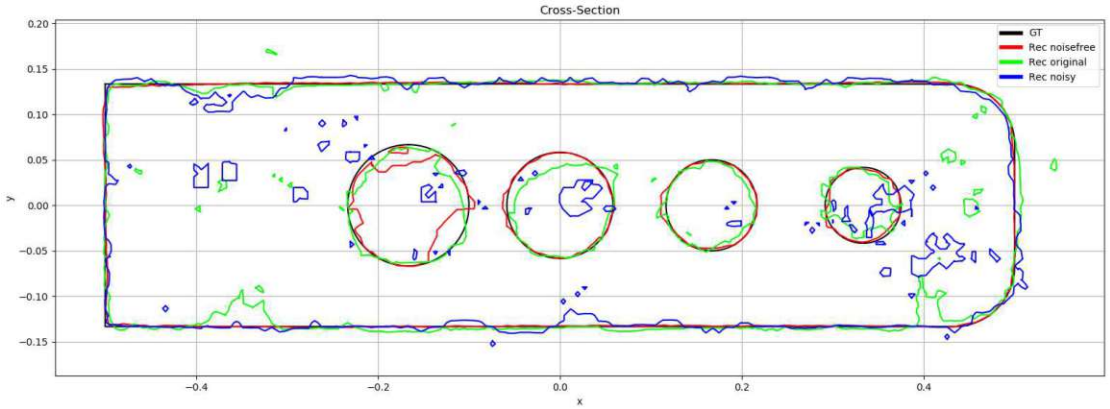
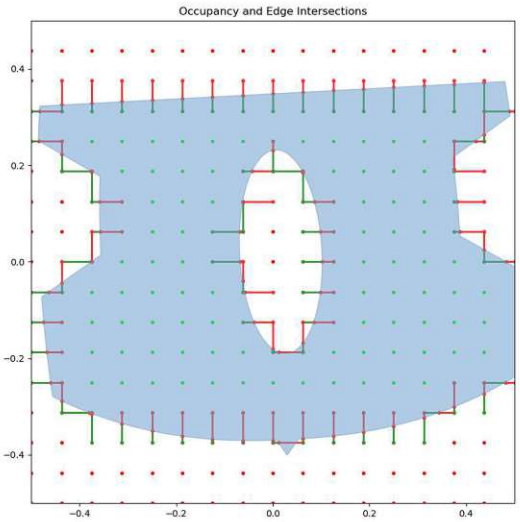
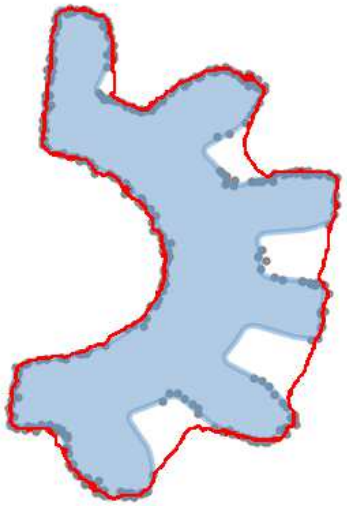


Figure 4.3: Cross-section of a reconstructed SDF. The reconstruction of Points2Surf introduces noise on top of the input noise, especially in cavities.



(a) GT Edge intersections.



(b) GT shape and point cloud in blue, hierarchical reconstruction in red.

Figure 4.4: Approaches tried for PPSurf: edge intersections and hierarchical reconstruction

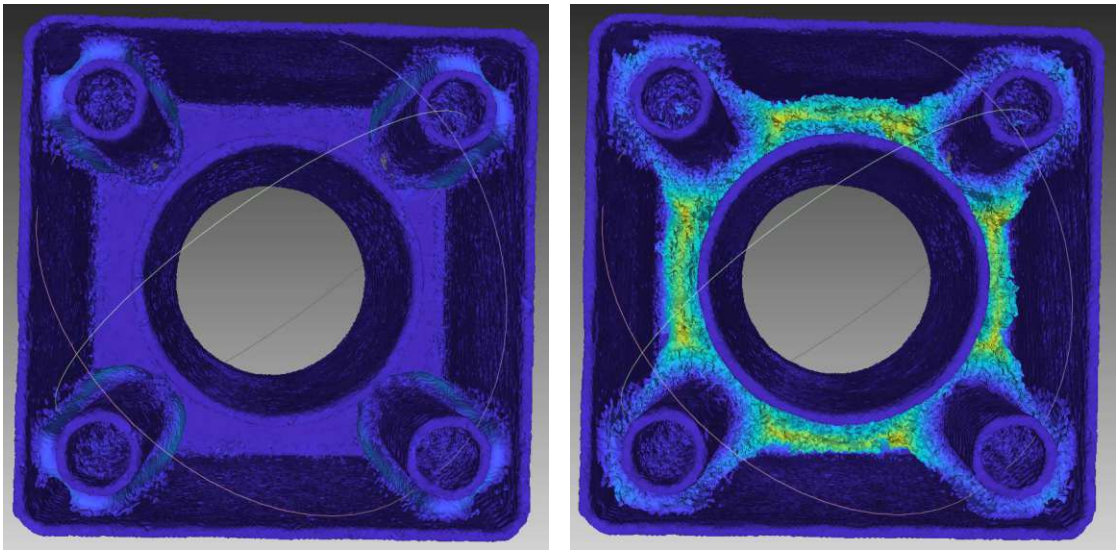
reconstruction method with differentiable Marching Cubes is still an open problem.

Next, we tried a hierarchical reconstruction by subdividing cells several times and re-evaluating the child cells if they are close to the estimated surface. This approach is closely related to the Multiresolution IsoSurface Extraction of Mescheder et al. [MON+19], which, however, only uses a global point cloud encoding. This subdivision removed most of the topological noise but introduced major oversmoothing. The model completely failed to learn cavities and holes, as shown in Figure 4.4b. Further, we experimented

with farthest point sampling for the global subsample and the local subsample with a fixed radius. This improved the quality but created a significant CPU load. Overall, the results were clearly inferior to Points2Surf.

Since re-evaluating helped only with evaluation speed but not accuracy, we tried next to forward coarser features to finer levels. Today, similar approaches exist for LOD and geometry representation, most notably Takikawa et al. [TLY+21]. By overfitting on an SDF, they store feature vectors in an octree and use trilinear interpolation for extraction. A small MLP then predicts the signed distance from the interpolated features. Our implementation of this approach suffered mostly from topological noise inside the objects.

The hierarchical subdivision of the grid did not improve the reconstruction. Therefore, we opted for forwarding information from sampling the intermediate reconstruction instead. While our attempt failed again, Peng et al. succeeded in their Shape As Points in 2021 [PJL+21].



(a) Points2Surf evaluation. (b) Evaluation of the entire volume.

Figure 4.5: The heuristic for query point selection avoids a significant portion of reconstruction noise.

It remains unclear to some degree why all these attempts failed while other groups succeeded with similar approaches. One common part is PointNet as the backbone architecture for all the networks, which we kept from Points2Surf. In hindsight, this backbone seems relatively fast but inaccurate. Although the heuristic for choosing SDF cells for the inference and the sign propagation of Points2Surf was rushed, it effectively suppresses the errors of the network. This can be seen in Figure 4.5, where the heuristic avoids most wrongly classified voxels. In contrast, all the approaches we tried afterward essentially evaluate the network in the entire domain. Forwarding information also forwards errors, which are never corrected. In summary, the simple heuristic for the SDF

evaluation of Points2Surf was the reason for its relatively good quality, and all more advanced methods could not eliminate the poor representation quality of the PointNet backbone, especially in the global branch.

4.2.2 Stronger Encoding

As the previous experiments indicate, we needed a stronger point cloud encoding than PointNet can achieve. Luckily, Boulch and Marlet had published POCO [BM22] recently. It is based on their FKAConv [BPM20], a high-quality point convolution network. POCO adds a second stage to the architecture, which efficiently queries the strong global encoding of the FKAConv for reconstructing an SDF. One training sample of POCO consists of many query points and one random subsample of the input point cloud. This is much more efficient than the one query point per subsample of Points2Surf. Further, the test-time augmentation averages many random subsamples for smooth reconstructions.

POCO essentially solved all the problems we found before, but it reopened one that Points2Surf had fixed: local information for high-quality details. Therefore, we tried adding a local branch, first by duplicating the FKAConv. This requires running the heavy encoding for every query point, which turned out to be much too slow, as a single reconstruction would have taken hours to days. Even a light version of the FKAConv with fewer layers was out of the question. Eventually, we fell back to PointNet for a light-weight local point cloud encoding. Using POCO's attention mechanism as a symmetric operation for aggregating the point features increased the quality further, without too much extra computational effort. This resulted in PPSurf, which we describe in the remainder of this chapter.

4.3 Overview

After exploring many approaches to improve upon Points2Surf regarding the query point selection, sign propagation, and the encoding strength, we finally found a working configuration that produces good results. We propose PPSurf as a method that covers a wider range in the global-local spectrum of priors, by combining the local prior of a patch-based method like Points2Surf with a stronger global prior of a point convolution-based method like POCO. For this purpose, we design an architecture that has two branches: the first branch is based on POCO [BM22] and provides a global prior by applying several layers of point convolutions to a sparse set of support points. To reconstruct geometric details more accurately, we merge features from this first branch with features from a second branch, which processes a local patch of points with PointNet [QSMG17]. We additionally discovered that modifying the architecture of PointNet by replacing the sum aggregation with an attention-based aggregation improves performance. This results in a method that is robust to noise and missing points, while preserving details more accurately than previous methods.

In our experiments, we compare PPSurf to several previous state-of-the-art methods,

both data-driven and non-data-driven, on synthetic as well as real-world data, and demonstrate improved performance on both in-distribution and out-of-distribution surface reconstruction tasks.

4.4 Method

The goal of our method is to take as input an unoriented point cloud $P = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n\}$ that was sampled from an unknown watertight surface $\mathcal{S}^{\text{gt}}$ with a noisy sampling process, and output a surface $\mathcal{S}$ that approximates $\mathcal{S}^{\text{gt}}$ as closely as possible. Similar to several previous approaches, we define the surface $\mathcal{S}$ using an implicit representation, since this guarantees watertightness and naturally handles arbitrary surface topology in a smooth and differentiable way. More specifically, $\mathcal{S}$ is defined as the 0.5-level set of an occupancy field $o(\mathbf{x})$: $\mathcal{S} \coloneqq \{\mathbf{x} \mid o(\mathbf{x}) = 0.5\}$.

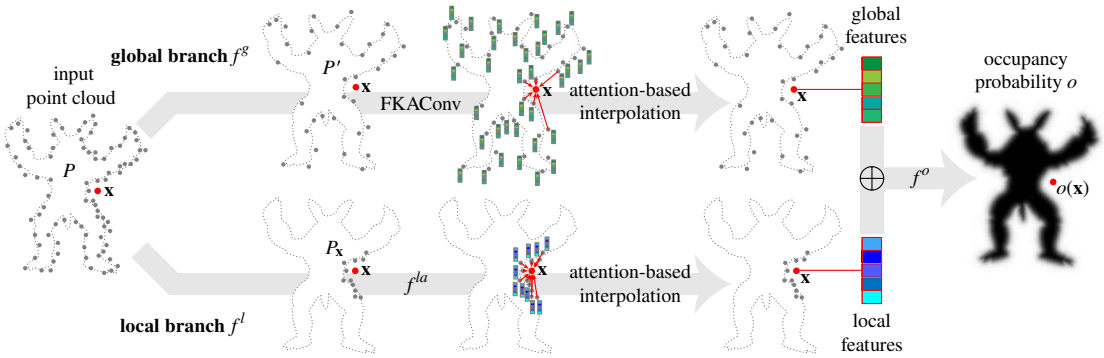


Figure 4.6: PPSurf computes the occupancy probability at a query point $\mathbf{x}$ given a noisy point cloud P. A *global* branch processes a sparse subset $P' \subseteq P$ using point convolutions, followed by an attention-based interpolation to get features at $\mathbf{x}$ that capture the coarse shape of the point cloud. A *local* branch processes a local patch $P_\mathbf{x} \subset P$ using a PointNet [QSMG17] with attention-based aggregation to get features at $\mathbf{x}$ that capture the detailed shape of the point cloud near $\mathbf{x}$. Global and local features are aggregated to compute the occupancy probability at $\mathbf{x}$.

We train a network $f_\theta(\mathbf{x}, P)$ with parameters θ to model the field o given a point cloud P:

$$o(\mathbf{x}) \coloneqq f_\theta(\mathbf{x}, P) \tag{4.1}$$

The network f uses two branches: i) a *global* branch $f^g(\mathbf{x}, P')$ that performs point convolutions [BPM20] on a sparse random subset of points $P' \subseteq P$ and effectively learns a global prior over the coarse shape of $\mathcal{S}$ given the input points P, and ii) a *local* branch $f^l(\mathbf{x}, P_\mathbf{x})$ that processes a small local patch $P_\mathbf{x} \subset P$ around $\mathbf{x}$ and effectively learns a local prior over the detailed shape of local surface patches. Each branch outputs a feature vector for a given query point $\mathbf{x}$ that is combined into a single feature vector before being processed by a small MLP f^o that outputs the occupancy probability $o(\mathbf{x})$:

$$f_\theta(\mathbf{x}, P) \coloneqq f^o(f^g(\mathbf{x}, P') \oplus f^l(\mathbf{x}, P_\mathbf{x})), \tag{4.2}$$

where $\oplus$ is the operation used to combine the two feature vectors, a sum in our experiments. Here, we omit the parameters of the networks f^o, f^g, and f^l to avoid a cluttered notation. Figure 4.6 illustrates our architecture.

In the following, we describe the architecture of PPSurf, including the global and local branches in Section 4.4.1, followed by a description of the training and inference setups in Sections 4.4.2 and 4.4.3, respectively.

4.4.1 Architecture

Global Branch The global branch $f^g(\mathbf{x}, P')$ takes as input a random subset $P' \subseteq P$ and a 3D query point $\mathbf{x}$ and outputs a *global* feature vector for the point $\mathbf{x}$, which encodes information about the coarse shape of the point cloud. We implement the global branch using POCO [BM22], which consists of two main components: i) a point convolution module that computes a feature vector $\mathbf{z}'_i$ for each sparse point $\mathbf{p}'_i \in P'$, followed by ii) an interpolation module that interpolates the feature vectors $\mathbf{z}'_i$ to get the global feature vector at point $\mathbf{x}$.

The point convolution module uses FKAConv [BPM20] to process the sparse point cloud P' into a feature vector for each point:

$$Z' = \text{FKAConv}(P'), \tag{4.3}$$

where $Z' = \{\mathbf{z}'_1, \mathbf{z}'_2, \dots, \mathbf{z}'_{|P'|}\}$ is the set of feature vectors at each sparse point. Due to limitations both in performance and network capacity, convolutions can only be performed on the sparse subset P' instead of the full point cloud P, with $|P'| = 10\text{k}$ in our experiments. This module consists of 10 layers of convolutions. Each layer uses a convolution kernel that operates over the 16 nearest neighbors of each point.

Given a query point $\mathbf{x}$, the interpolation module interpolates the feature vectors $\mathbf{z}'_i$ at the nearest neighbors $\mathcal{N}'_\mathbf{x}$ of the query point to get the global feature vector using an attention-based weighting:

$$f^g(\mathbf{x}, P') \coloneqq f^{gb}\Big(\sum_{j \in \mathcal{N}'_\mathbf{x}} w_{\mathbf{x},j} \; f^{ga}((\mathbf{x} - \mathbf{p}'_j) \| \mathbf{z}'_j) \Big) \tag{4.4}$$

$$\text{with } w_{\mathbf{x},j} \coloneqq \frac{1}{k} \sum_{k=1}^{64} \text{softmax}_j \; f_k^{gw}((\mathbf{x} - \mathbf{p}'_j) \| \mathbf{z}'_j), \tag{4.5}$$

where $\|$ denotes concatenation, f^{ga}, f^{gb} are two MLPs that transform the feature vectors before and after the weighted sum, and f_k^{gw} are learned weighting functions, each implemented as a single linear layer. Analogous to the attention heads in multi-head attention, multiple different weighting functions are used as a form of ensemble learning, 64 in our experiments. Note that when evaluating multiple query points $\mathbf{x}$ for a point cloud, the point convolution module only needs to be evaluated once, while the interpolation module needs to be evaluated once per query point.

Local Branch The local branch $f^l(\mathbf{x}, P_\mathbf{x})$ processes a local patch $P_\mathbf{x}$ around the query point $\mathbf{x}$ and outputs a *local* feature vector for the point $\mathbf{x}$, which encodes information about the detailed shape of the point cloud near $\mathbf{x}$. We base the local branch on the popular PointNet [QSMG17] architecture, which has been successfully applied in various methods that process local point cloud patches [GKOM18, RLBG+19]. We modify the architecture with an attention-based aggregation, instead of the original max- or sum-based aggregation, which we found to improve performance.

We define the local patch $P_\mathbf{x}$ as the 50 nearest neighbors of the query point $\mathbf{x}$. We normalize the patch by centering it at the origin and scaling it to fit into a unit sphere, obtaining the normalized patch $\bar{P}_\mathbf{x}$. Subsequently, we apply PointNet with attention-based aggregation similar to Eqs. 4.4 and 4.5, but without using multiple attention heads:

$$f^l(\mathbf{x}, P_\mathbf{x}) \coloneqq f^{lb}\Big(\sum_{\bar{\mathbf{p}}_j \in \bar{P}_\mathbf{x}} v_j \; f^{la}(\bar{\mathbf{p}}_j) \Big) \tag{4.6}$$

$$\text{with } v_j \coloneqq \text{softmax}_j \; f^{lv}(f^{la}(\bar{\mathbf{p}}_j)), \tag{4.7}$$

where f^{lv} is a learned weighting function implemented as linear layer, and f^{la}, f^{lb} are two MLPs that transform the feature vectors before and after the weighted aggregation.

4.4.2 Training Setup

We train our network with a binary cross-entropy loss $\text{BCE}(o(\mathbf{x}), o^{\text{gt}}(\mathbf{x}))$ supervised by the ground-truth occupancy $o^{\text{gt}}(\mathbf{x})$ on query points defined by the Points2Surf ABC var-noise training set [EGO+20]. We train with AdamW (lr=0.001, betas=(0.9, 0.999), eps=1e-5, weight_decay=1e-2, amsgrad=False) for 150 epochs with scheduler steps at 75 and 125 epochs. On our training machine, we can fully utilize all 4 NVIDIA A40 GPUs with distributed data-parallel training using a total batch size of 50 and 48 workers. The other hyperparameters are mostly based on POCO, namely $10k$ manifold points, a network decoder k of 64 and 2 output classes. One change is the increased latent size of 128, which was 32 in POCO. The additional hyperparameters for the local branch are a PointNet latent size of 256 and a patch size of 50. The training takes about 5 hours.

4.4.3 Inference Setup

We use the inference setup from POCO [BM22], which differs from the training setup in two main aspects: First, we perform test-time augmentation in our global branch to obtain more reliable results. Second, we sample query points in a grid and use a variant of marching cubes to reconstruct a mesh. We describe both in more detail below.

Test-time augmentation. The sparse subsample $P' \subseteq P$ used for the global branch may miss important geometric detail. To improve robustness, we compute the per-point

feature vectors $\mathbf{z}'_i$ for multiple different random subsamples $P'_1, P'_2, \ldots$, until each point in P is included in at least 10 subsamples. The ≥ 10 different feature vectors for each point in P are then averaged before performing the interpolation step.

Mesh reconstruction. We place query points in a 257^3 grid and use a variant of marching cubes [LC87] proposed in POCO to obtain a mesh from the occupancy field $o(\mathbf{x})$. That marching cubes variant uses a region-growing strategy starting from the input points to avoid the costly evaluation at all grid points, and super-samples marching-cube edges that intersect a surface to get a more accurate estimate of the intersection point.

4.5 Results

We evaluate PPSurf by comparing our surface reconstruction performance to several state-of-the-art methods, both data-driven and non-data-driven. We show both quantitative and qualitative comparisons in Section 4.5.1. Additionally, we provide an ablation to empirically validate our main design choices in Section 4.5.2.

Metrics Similar to Section 3.5.2, we use four well-known metrics to evaluate the error of our reconstructed surfaces: the Chamfer distance, the F1-score, the normal error, and the Intersection Over Union (IoU). We evaluate each metric at $100k$ random surface samples for the Chamfer distance and normal error, or volume samples for the IoU. This results in roughly $\pm 0.5\%$ variance between different runs.

The *Chamfer distance* [BTBW77, FSG17] measures the distance between two point sets. We use it to measure the distance between reconstructed and GT surface samples. It is defined as:

$$\frac{1}{|A|} \sum_{\mathbf{p}_i \in A} \min_{\mathbf{p}_j \in B} \|\mathbf{p}_i - \mathbf{p}_j\|_2^2 \; + \frac{1}{|B|} \sum_{\mathbf{p}_j \in B} \min_{\mathbf{p}_i \in A} \|\mathbf{p}_j - \mathbf{p}_i\|_2^2, \tag{4.8}$$

where A and B are point sets of size $100k$ sampled on the surface of the GT object and the reconstructed object.

The *F1 Score* [TH15] measures the overlap between the ground truth surface and the region enclosed by the reconstructed surface, similar to the IoU. It weights precision and recall equally.

The *normal error* measures the difference between the normals of the reconstructed surface and the ground truth normals. We sample $100k$ points uniformly on the ground truth mesh A and the reconstructed mesh B, storing the normals of their originating faces. Then, we find the closest neighbor of each point $b \in B$ in A. We report the average angle between the normals of these point pairs: $\frac{1}{n_s} \sum_{i=1}^{n_s} (\arccos(\mathbf{n}_i^A \cdot \mathbf{n}_i^B))$, where $\mathbf{n}_i^A$ and $\mathbf{n}_i^B$ are ground truth and reconstructed normals, respectively.

Datasets We evaluate our method on the set of dataset variants introduced in P2S [EGO+20]:

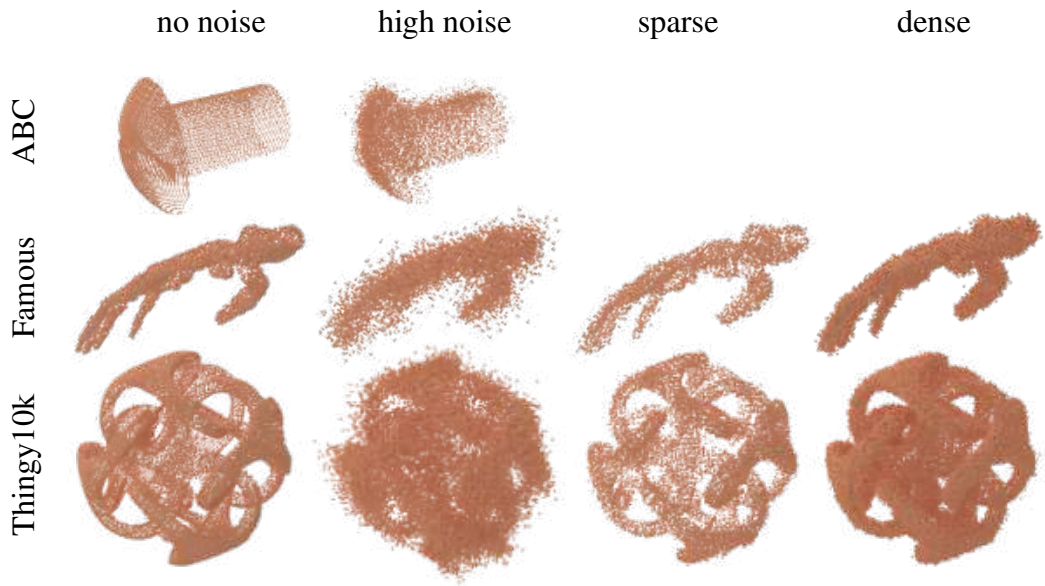


Figure 4.7: Point cloud examples of the datasets used in our evaluation.

- The ABC variant of P2S [EGO+20] is a subset of the ABC dataset by Koch et al. [KMJ+19] and contains 4950 points clouds from high-quality CAD meshes in the training set and 100 point clouds in the test set.
- The Famous [EGO+20] dataset consists of 22 diverse well-known meshes, including the Stanford Bunny, the Utah Teapot, and the Armadillo. We use this dataset for testing only.
- A subset of 100 shapes from the Thingi10k [ZJ16] dataset are used as additional test set. The Thingi10k dataset contains a variety of CAD shapes, but also more organic shapes like statues.
- The Real [EGO+20] dataset consists of 3 real-world point clouds.

All synthetic point clouds were created with the simulated scanner BlenSor [GKUP11] with a scanner resolution of 176×144, using a random number of scans between 5 and 30. Each dataset comes in up to five variants:

- *no noise*: A version without noise
- *med. noise*: A version with noise using a standard deviation of $0.01L$, where L is the largest side of the object's bounding box.
- *high noise*: A version with noise using a standard deviation of $0.05L$.

- *var. noise*: A version with variable noise, where the amount of noise used for a given shape is sampled uniformly in $[0, 0.05L]$ and the number of scans in $[5, 30]$.
- *sparse*: A version with medium noise where all shapes only uses 5 scans, resulting in point clouds between $2k$ and $22k$ points.
- *dense*: A version with medium noise where all shapes use 30 scans, resulting in point clouds between $5k$ and $112k$ points.

For a fair comparison, we train all data-driven methods on the ABC var. noise dataset and evaluate them with each test set. Some point cloud examples of these datasets are illustrated in Figure 4.7.

Figure 4.8 shows comparisons for one example of each dataset variant. While non-data-driven methods give competitive results on low-noise results, PPSurf has a clear advantage with sparse and noisy point clouds.

4.5.1 Comparisons

We compare PPSurf to several recent data-driven and non-data-driven reconstruction methods. PGR [LXSW22], Neural-IMLS (IMLS) [WWW+23] and Shape as Points (SAP-O) are non-data-driven methods that do not train on a large dataset and instead directly fit a surface to the input point cloud. Shape as Points also has a data-driven variant (SAP) that uses a trained network. Additionally, we use Points2Surf (P2S) [EGO+20] and POCO [BM22] as data-driven methods. We took the best available variants and settings for each method: For PGR, we use the default parameters *wmin=0.0015, alpha=1.05* for no noise, med noise and var. noise. We use the following adapted parameters for the other datasets: *wmin=0.03, alpha=2.0* for high noise, *wmin=0.03, alpha=1.5* for dense and sparse. We use *thingi-noisy* for SAP-O, *vanilla* for P2S , and *10k-FKAConv-InterpAttentionKHeadsNet* for POCO. We used the provided *noise-large* configuration for SAP. For IMLS, we used the results provided by the authors (high noise datasets were not provided by the authors). Note that IMLS was developed concurrently with our work.

Qualitative Comparison We show examples on real-world point clouds in Figure 4.9, where PPSurf produces clearer edges and finer details.

Quantitative Comparison Tables 4.1, 4.2, 4.3, and 4.4 show the performance of PPSurf on all dataset variants. We report the average over all shapes in the test set. Similar to the qualitative results, POCO, PPSurf and the non-data-driven methods share the first place in most low-noise dataset variants, but PPSurf 50NN takes the lead in almost all other dataset variants. This confirms that adding the local branch does indeed improve the local reconstruction.

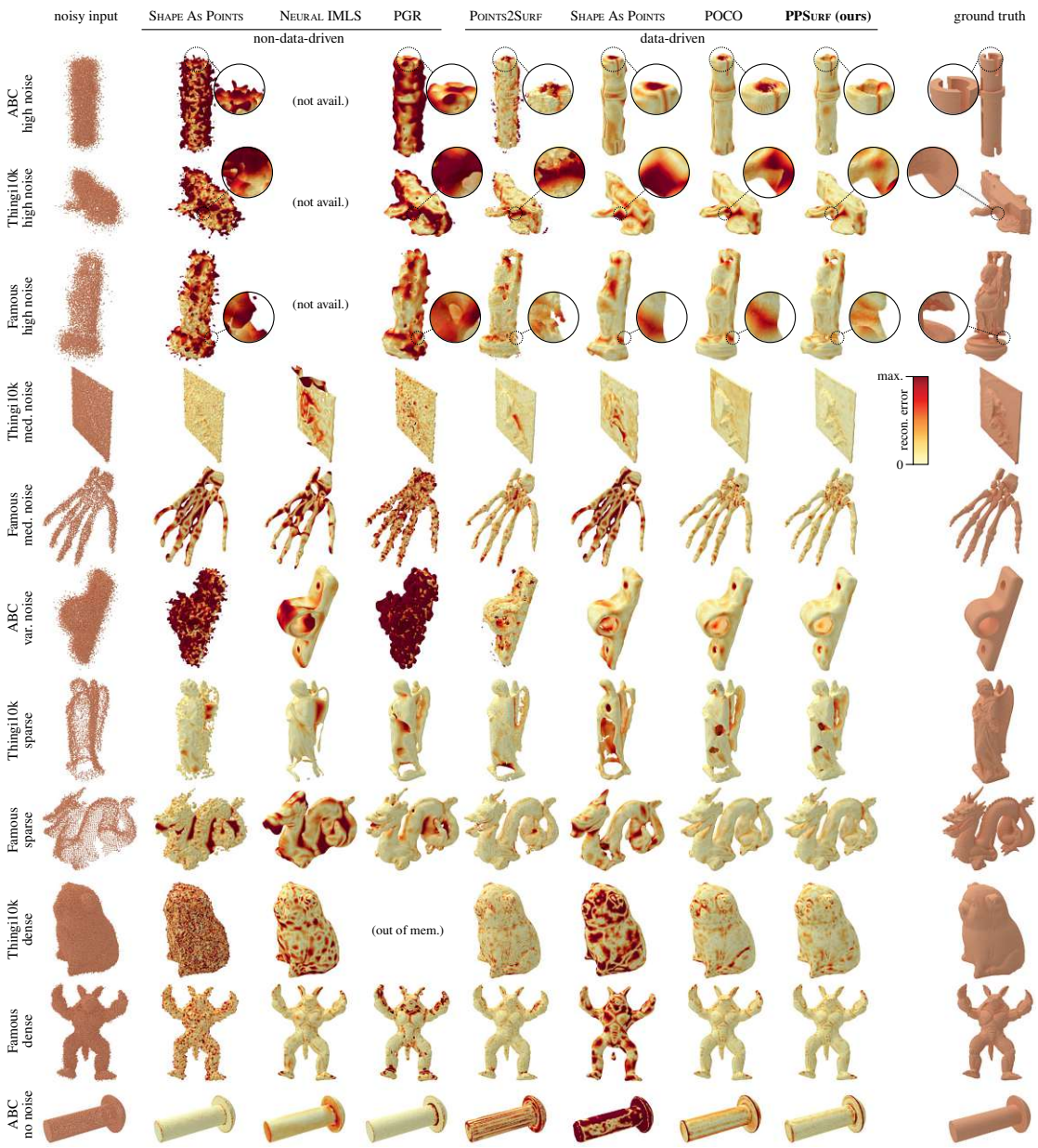


Figure 4.8: Qualitative comparison to all baselines. We evaluate one example from each dataset variant (except for the no-noise variants, where we only show one example due to space constraints). Colors show the distance of the reconstructed surface to the ground-truth surface. Due to our combined local and global branches, PPSurf reconstructs details more accurately than the baselines, especially in the presence of strong input noise. Note that results for Neural IMLS are not provided by the authors for the high-noise dataset variants.

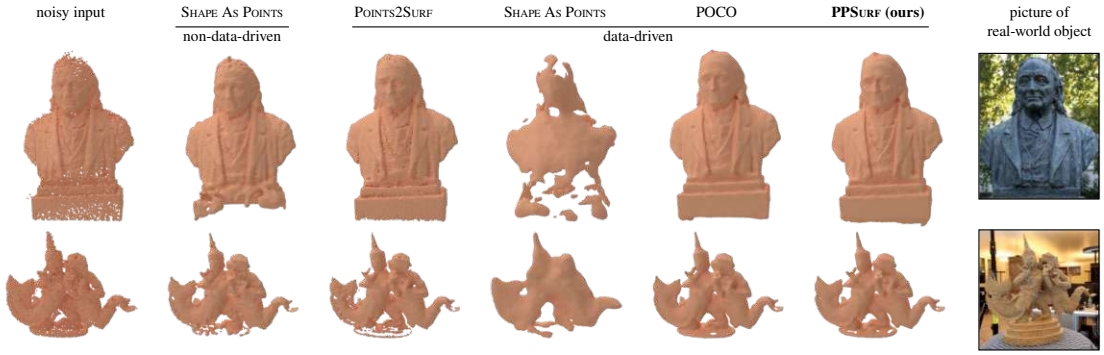


Figure 4.9: Real-world reconstructions. We compare to all baselines on the two point clouds that were obtained from real-world objects.

Table 4.1: Comparison of **Chamfer Distance** (x100). PPSurf consistently performs similar or better than the baselines. Best results per row are bold, second-best underlined.

Dataset	IMLS	PGR	SAP-O	SAP	P2S	POCO	PPSurf
ABC var. noise	1.08	1.60	1.18	1.18	0.84	0.70	**0.66**
ABC no noise	**0.48**	0.53	0.63	1.08	0.61	0.50	**0.48**
Famous no noise	**0.35**	0.36	**0.35**	0.99	0.46	0.39	0.37
Thingi10k no noise	0.40	**0.30**	0.43	0.89	0.39	0.33	0.33
Mean no noise	0.41	0.40	0.47	0.99	0.49	0.41	**0.39**
Famous med. noise	0.54	0.95	0.58	1.06	0.52	0.49	**0.48**
Thingi10k med. noise	0.58	0.93	0.56	0.93	0.44	0.39	**0.38**
Mean med. noise	0.56	0.94	0.57	0.99	0.48	0.44	**0.43**
ABC high noise	–	1.90	1.96	1.51	1.24	1.00	**0.97**
Famous high noise	–	1.86	1.80	1.62	1.14	1.11	**1.01**
Thingi10k high noise	–	1.94	1.89	1.45	1.08	0.92	**0.83**
Mean high noise	–	1.90	1.88	1.53	1.16	1.01	**0.94**
Famous sparse	0.90	0.88	0.71	1.24	0.77	0.67	**0.64**
Thingi10k sparse	0.82	0.89	0.86	1.35	0.78	**0.63**	**0.63**
Mean sparse	0.86	0.88	0.79	1.29	0.77	0.65	**0.63**
Famous dense	0.45	0.70	0.53	0.96	0.41	0.42	**0.40**
Thingi10k dense	0.49	0.67	0.54	0.88	0.36	0.35	**0.33**
Mean dense	0.47	0.69	0.53	0.92	0.39	0.39	**0.37**
Mean overall	0.61	1.04	0.93	1.16	0.70	0.61	**0.58**

Table 4.2: Comparison of **F1 Scores**. PPSurf achieves best or second-best performance across most datasets. Best results per row are bold, second-best underlined.

Dataset	IMLS	PGR	SAP-O	SAP	P2S	POCO	PPSurf
ABC var. noise	0.78	0.50	0.67	0.79	0.83	0.89	**0.90**
ABC no noise	0.92	0.92	0.88	0.80	0.88	**0.94**	**0.94**
Famous no noise	0.95	0.95	0.95	0.84	0.93	0.95	**0.96**
Thingi10k no noise	0.93	0.95	0.92	0.85	0.93	0.95	**0.96**
Mean no noise	0.93	0.94	0.92	0.83	0.91	**0.95**	**0.95**
Famous med. noise	0.91	0.60	0.90	0.83	0.92	**0.94**	0.93
Thingi10k med. noise	0.90	0.57	0.89	0.85	0.92	**0.94**	**0.94**
Mean med. noise	0.91	0.59	0.89	0.84	0.92	**0.94**	**0.94**
ABC high noise	–	0.42	0.49	0.75	0.78	0.84	**0.85**
Famous high noise	–	0.50	0.59	0.78	0.84	0.84	**0.85**
Thingi10k high noise	–	0.51	0.60	0.80	0.84	0.87	**0.88**
Mean high noise	–	0.32	0.56	0.78	0.82	0.85	**0.86**
Famous sparse	0.86	0.88	0.88	0.74	0.89	**0.92**	**0.92**
Thingi10k sparse	0.85	0.86	0.84	0.73	0.87	**0.90**	**0.90**
Mean sparse	0.86	0.87	0.86	0.73	0.88	**0.91**	**0.91**
Famous dense	0.93	0.43	0.90	0.86	0.94	**0.95**	**0.95**
Thingi10k dense	0.91	0.47	0.89	0.87	0.94	0.95	**0.96**
Mean dense	0.92	0.45	0.89	0.86	0.94	0.95	**0.96**
Mean overall	0.89	0.66	0.80	0.81	0.89	0.91	**0.92**

Table 4.3: Comparison of **Normal Errors**. PPSurf shows strong performance, especially in high-noise scenarios. Best results per row are bold, second-best underlined.

Dataset	IMLS	PGR	SAP-O	SAP	P2S	POCO	PPSurf
ABC var. noise	0.55	1.29	1.11	0.52	0.65	0.32	**0.30**
ABC no noise	0.20	0.26	0.30	0.51	0.31	**0.19**	**0.19**
Famous no noise	**0.44**	0.48	**0.44**	0.75	0.57	0.46	0.46
Thingi10k no noise	0.25	0.19	0.23	0.49	0.32	**0.18**	0.19
Mean no noise	0.30	0.31	0.33	0.58	0.40	**0.28**	**0.28**
Famous med. noise	0.57	1.35	0.91	0.78	0.63	0.53	**0.54**
Thingi10k med. noise	0.37	1.32	0.78	0.50	0.38	**0.24**	0.25
Mean med. noise	0.47	1.33	0.85	0.64	0.50	**0.38**	0.39
ABC high noise	–	1.35	1.42	0.65	0.99	0.43	**0.41**
Famous high noise	–	1.35	1.39	0.91	1.04	0.76	**0.72**
Thingi10k high noise	–	1.31	1.36	0.64	0.90	0.47	**0.43**
Mean high noise	–	1.34	1.39	0.73	0.98	0.55	**0.52**
Famous sparse	0.68	0.75	0.86	0.89	0.71	0.60	**0.61**
Thingi10k sparse	0.48	0.53	0.76	0.73	0.51	**0.37**	0.39
Mean sparse	0.58	0.64	0.81	0.81	0.61	**0.48**	0.50
Famous dense	0.52	1.33	1.00	0.74	0.59	0.49	**0.48**
Thingi10k dense	0.30	1.23	0.84	0.47	0.33	**0.21**	**0.21**
Mean dense	0.41	1.28	0.92	0.60	0.46	0.35	**0.34**
Mean overall	0.43	0.98	0.88	0.66	0.61	**0.40**	**0.40**

Table 4.4: Comparison of **IoU**. We show the IoU between reconstructed and ground-truth surfaces averaged over all shapes in a dataset. PPSurf performs similar or better than the baselines. Best results per row are marked in bold and the second-best results are underlined.

Dataset	IMLS	PGR	SAP-O	SAP	P2S	POCO	PPSurf
ABC var. noise	0.69	0.37	0.56	0.69	0.75	0.82	**0.84**
ABC no noise	0.88	0.88	0.83	0.69	0.83	**0.90**	**0.90**
Famous no noise	**0.92**	0.91	**0.92**	0.75	0.88	**0.92**	**0.92**
Thingi10k no noise	0.89	0.93	0.89	0.77	0.90	0.93	**0.94**
Mean no noise	0.89	0.91	0.88	0.74	0.87	**0.92**	**0.92**
Famous med. noise	0.86	0.44	0.83	0.74	0.86	**0.89**	**0.89**
Thingi10k med. noise	0.84	0.42	0.83	0.77	0.89	**0.91**	**0.91**
Mean med. noise	0.85	0.43	0.83	0.75	0.87	**0.90**	**0.90**
ABC high noise	–	0.28	0.35	0.63	0.67	0.75	**0.76**
Famous high noise	–	0.34	0.44	0.65	0.74	0.74	**0.76**
Thingi10k high noise	–	0.35	0.45	0.70	0.76	0.79	**0.81**
Mean high noise	–	0.32	0.41	0.66	0.72	0.76	**0.78**
Famous sparse	0.78	0.79	0.80	0.64	0.81	0.85	**0.86**
Thingi10k sparse	0.78	0.79	0.77	0.62	0.81	0.84	**0.85**
Mean sparse	0.78	0.79	0.79	0.63	0.81	**0.85**	**0.85**
Famous dense	0.88	0.28	0.83	0.77	0.90	0.91	**0.92**
Thingi10k dense	0.87	0.32	0.83	0.79	0.91	**0.93**	**0.93**
Mean dense	0.87	0.3	0.83	0.78	0.90	0.92	**0.93**
Mean overall	0.84	0.55	0.72	0.71	0.82	0.86	**0.87**

Computation Time and Memory Consumption Training PPSurf on the ABC var-noise training set was done in 5 hours on 4 NVIDIA A40 GPUs and 48 AMD EPYC-Milan cores. We reconstruct all shapes in our test sets on a single A40 and 48 CPU cores. See the timings and memory consumption in Table 4.5. While non-data-driven methods tend to be faster than data-driven ones, SAP is a lightning-fast exception. PPSurf with small patch sizes has a negligible impact on resources compared to POCO. Neural IMLS does not report timings. As it is concurrent work, we could not do our own measurements. While it is fast, PGR's memory usage varies a lot with point cloud size, between a few GB to going out-of-memory with >46GB on 21 shapes.

Table 4.5: Comparison of reconstruction times and memory usage. We show the mean reconstruction time per shape and the maximum GPU-memory consumption for each method on the ABC var noise dataset. 200NN uses reconstruction batch size $25k$ instead of $50k$. PGR went out of memory on 21 shapes.

	Time per Shape	Max GPU Memory
PGR	1.9 min	>46GB
SAP-O	1.1 min	3.8GB
SAP	0.8sec	3.1GB
P2S	13.5min	14.3GB
POCO	1.6min	9.0GB
PPSurf 10NN	1.6min	9.1GB
PPSurf 25NN	1.7min	9.1GB
PPSurf 50NN	1.9min	9.3GB
PPSurf 100NN	2.6min	13.7GB
PPSurf 200NN	3.5min	13.2GB

Discussion For dense and noise-free point clouds, non-data-driven methods such as PGR, SAP-O and especially IMLS are a good option. However, their performance is limited in the presence of typical point-cloud artifacts, due to missing data-driven priors. Data-driven methods such as SAP, P2S, POCO and PPSurf can better deal with such artifacts. SAP is the fastest method but lacks accuracy, possibly due to its very small network. A bigger version could perhaps produce competitive results but would require non-trivial changes to the method.

P2S employs a relatively simple PointNet for global shape encoding, which results in a weak global prior that can not reach the quality of a more efficient encoder such as FKAConv. Furthermore, it reconstructs noisy surfaces, which is reflected in the relatively high normal error, even with noise-free inputs.

Apart from some noise-free datasets, only POCO is close to PPSurf's quality. PPSurf achieves similar results on low-noise point clouds, but significantly better reconstructions for noisy point clouds. When predicting the occupancy at the query points, POCO has no direct access to the full point cloud, only to a coarse latent representation. This

inability to accurately represent local information is likely the reason why POCO tends to produce blobby structures and over-smooth the reconstructed surfaces. We avoid this by providing a latent code that captures local detail more accurately by adding a local branch that directly encodes dense local patches of the point cloud.

4.5.2 Ablation

We investigate several design choices in an ablation study on the ABC var-noise test set. Most importantly, Table 4.10 shows that having both global and local branches gains a major advantage. Referring to Table 4.8, the optimal local patch size lies in the range of $25NN$ to $100NN$. Further, attention is a better symmetric operation than max, and concatenating features is similar to summing them. This can be seen in Table 4.6. Please see Tables 4.9 and 4.7 for an evaluation of the most relevant variants on all datasets. We compare the following variants of our method:

- *Full* is the full method as described in Section 4.4.
- For *Sym Max*, we replace the attention-based interpolation used in the local branch with the max, effectively making this branch a PointNet [QSMG17]. The results show an advantage for attention.
- In *Merge Cat*, we concatenate the features of both branches instead of summing them, which leads to twice the input size for the final MLP. Results show that this is slightly worse than *Full*.
- The *QPoints* variant is the same as *Merge Cat*, but additionally, we concatenate query point coordinates to the input of the learned weighting function f^{lv}. However, this results in a slightly worse performance than *Full* and even *Merge Cat*.
- For the *xNN* variants, we take the x nearest neighbors for local subsample. *Full* is equal to 50NN.
- For *Only Local*, we set the global features to zeros, disabling this branch. Based on the results of this experiment, we conclude that this model can not reliably encode any surface since it lacks global knowledge of the surface to reconstruct.
- *Only Global* is similar to POCO as it omits the local branch. The results show that a global prior can help to obtain reliable reconstructions but with lower performance due to the missing fine details.

Table 4.6: Miscellanous Ablation Study. Using the ABC var-noise test set, we compare PPSurf *Full* (which uses Merge Sum and Sym Att) to more variants. The best results per column are marked in bold.

Model	Chamfer (x100) ⇓	F1 Score ⇑	Normal Error ⇓
PPSurf Sym Max	1.11	**0.90**	0.40
PPSurf QPoints	0.67	0.89	0.31
PPSurf Merge Cat	**0.66**	**0.90**	**0.30**
PPSurf Full	**0.66**	**0.90**	**0.30**

Table 4.7: All Datasets Merge Cat vs Merge Sum Ablation Study. The best results per row are marked in bold. *Sum* is equal to PPSurf *Full*.

Dataset	Chamfer (x100) ⇓		IoU ⇑		F1 ⇑		Normal Error ⇓	
PPSurf	Cat	Sum	Cat	Sum	Cat	Sum	Cat	Sum
ABC var. noise	0.661	**0.659**	0.834	**0.835**	0.898	**0.899**	0.305	**0.301**
ABC no noise	**0.495**	0.497	0.902	**0.903**	**0.940**	**0.940**	0.203	**0.199**
Famous no noise	0.411	**0.419**	**0.907**	0.906	**0.947**	0.945	**0.487**	0.491
Thingi10k no noise	0.326	**0.319**	0.933	**0.936**	0.956	**0.958**	0.198	**0.190**
Mean no noise	**0.411**	0.412	0.914	**0.915**	**0.948**	**0.948**	0.296	**0.294**
Famous med. noise	0.497	**0.485**	0.879	**0.883**	0.930	**0.933**	0.555	**0.542**
Thingi10k med. noise	**0.382**	0.384	0.910	**0.912**	0.943	**0.944**	0.249	**0.244**
Mean med. noise	0.440	**0.434**	0.894	**0.897**	0.937	**0.939**	0.402	**0.393**
ABC high noise	**0.956**	0.972	**0.760**	0.759	**0.847**	**0.847**	**0.403**	0.406
Famous high noise	1.026	**0.992**	0.732	**0.749**	0.835	**0.845**	0.756	**0.701**
Thingi10k high noise	**0.820**	0.824	0.809	**0.812**	0.880	**0.883**	0.433	**0.423**
Mean high noise	0.934	**0.929**	0.767	**0.773**	0.854	**0.858**	0.531	**0.510**
Famous sparse	0.691	**0.629**	0.848	**0.853**	0.911	**0.915**	0.619	**0.600**
Thingi10k sparse	0.671	**0.661**	0.840	**0.849**	0.897	**0.904**	0.401	**0.383**
Mean sparse	0.681	**0.645**	0.844	**0.851**	0.904	**0.909**	0.510	**0.491**
Famous dense	0.422	**0.404**	0.907	**0.908**	**0.949**	0.947	0.492	**0.488**
Thingi10k dense	**0.322**	0.326	0.932	**0.933**	**0.959**	**0.959**	**0.209**	0.212
Mean dense	0.372	**0.365**	0.920	**0.921**	**0.954**	0.953	**0.350**	**0.350**
Mean overall	0.591	**0.582**	0.861	**0.864**	0.915	**0.917**	0.408	**0.398**

Table 4.8: Patch Size Ablation Study. Using the ABC var-noise test set, we compare PPSurf *Full* (which is 50NN) to variants with different patch sizes. The best results per column are marked in bold.

Model	Chamfer (x100) ⇓	F1 Score ⇑	Normal Error ⇓
PPSurf 10NN	1.10	**0.90**	0.40
PPSurf 25NN	**0.66**	**0.90**	0.31
PPSurf Full	**0.66**	**0.90**	**0.30**
PPSurf 100NN	**0.66**	**0.90**	**0.30**
PPSurf 200NN	0.67	0.89	0.31

Table 4.9: All Datasets Patch Size Ablation Study. We compare the most relevant patch sizes on all datasets. The best results per row are marked in bold. PPSurf *50NN* is equal to PPSurf *Full*.

Dataset	Chamfer (x100) ⇓			IoU ⇑			F1 ⇑			Normal Error ⇓		
PPSurf	25NN	50NN	100NN	25NN	50NN	100NN	25NN	50NN	100NN	25NN	50NN	100NN
ABC var. noise	0.664	0.660	**0.659**	0.835	**0.838**	0.835	0.898	**0.901**	0.899	0.306	**0.297**	0.301
ABC no noise	0.486	**0.482**	0.497	0.902	0.902	**0.903**	**0.941**	**0.941**	0.940	**0.190**	0.194	0.199
Famous no noise	0.396	**0.373**	0.419	0.920	**0.923**	0.906	0.955	**0.958**	0.945	0.460	**0.455**	0.491
Thingi10k no noise	**0.314**	0.327	0.319	**0.936**	**0.936**	**0.936**	**0.958**	**0.958**	**0.958**	**0.190**	**0.190**	**0.190**
Mean no noise	0.399	**0.394**	0.412	0.919	**0.920**	0.915	0.951	**0.952**	0.948	0.280	**0.279**	0.294
Famous med. noise	0.496	**0.484**	0.485	0.883	**0.887**	0.883	0.934	**0.935**	0.933	**0.539**	0.543	0.542
Thingi10k med. noise	0.384	**0.383**	0.384	0.910	0.910	**0.912**	0.943	0.943	**0.944**	0.245	0.245	**0.244**
Mean med. noise	0.440	**0.434**	**0.434**	0.896	**0.898**	0.897	0.938	**0.939**	**0.939**	**0.392**	0.394	0.393
ABC high noise	**0.953**	0.968	0.972	**0.763**	0.759	0.759	**0.850**	0.846	0.847	**0.399**	0.409	0.406
Famous high noise	1.002	1.010	**0.992**	**0.759**	0.757	0.749	**0.853**	0.852	0.845	0.725	0.723	**0.701**
Thingi10k high noise	**0.818**	0.826	0.824	**0.813**	0.810	0.812	**0.883**	0.881	**0.883**	**0.422**	0.427	0.423
Mean high noise	**0.925**	0.935	0.929	**0.779**	0.775	0.773	**0.862**	0.860	0.858	0.515	0.520	**0.510**
Famous sparse	0.673	0.636	**0.629**	0.852	**0.857**	0.853	0.914	**0.918**	0.915	0.616	0.606	**0.600**
Thingi10k sparse	0.647	**0.629**	0.661	**0.849**	0.846	**0.849**	0.903	0.901	**0.904**	0.387	0.388	**0.383**
Mean sparse	0.660	**0.633**	0.645	0.850	**0.852**	0.851	**0.909**	**0.909**	**0.909**	0.502	0.497	**0.491**
Famous dense	0.411	**0.403**	0.404	0.910	**0.918**	0.908	0.949	**0.954**	0.947	0.482	**0.476**	0.488
Thingi10k dense	0.332	0.327	**0.326**	0.930	0.932	**0.933**	0.957	0.958	**0.959**	0.210	**0.209**	0.212
Mean dense	0.371	**0.365**	**0.365**	0.920	**0.925**	0.921	0.953	**0.956**	0.953	0.346	**0.342**	0.350
Mean overall	0.583	**0.578**	0.582	0.866	**0.867**	0.864	0.918	**0.919**	0.917	0.398	**0.397**	0.398

Table 4.10: Branch Ablation Study. Using the ABC var-noise test set, we compare PPSurf *Full* to variants with disabled branches. The only-local variant failed to produce some meshes, which are ignored in the metrics. The best results per column are marked in bold.

Model	Chamfer (x100) ⇓	F1 Score ⇑	Normal Error ⇓
Only Local	2.69	0.36	1.56
Only Global	0.70	0.89	0.33
PPSurf Full	**0.66**	**0.90**	**0.30**

4.5.3 Limitations

Reconstruction times are still non-interactive, due to the need to evaluate the occupancy at a large number of samples. Possibilities for speed-ups include more efficient sampling strategies to use fewer query points.

As our learned priors were trained on noisy data to make PPSurf more robust to noise, they also bias the reconstructed surface to some extent towards the distributions learned by the priors. This results in some loss of accuracy when applied to noise-free point clouds compared to some of the non-data-driven methods (see Figure 4.10). Learning a prior that is specialized to noise-free point clouds, or including more noise-free point clouds in our training set would alleviate this issue.

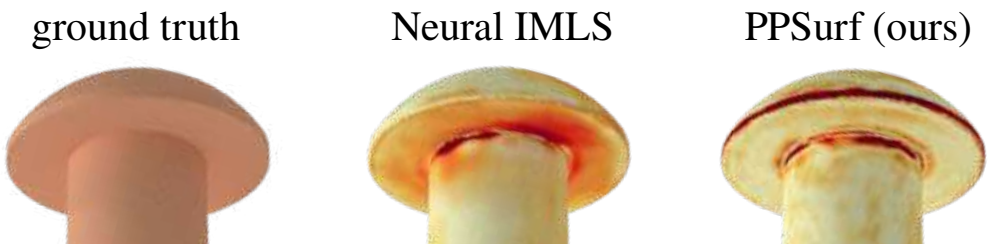


Figure 4.10: Limitations. Our method has difficulties to recover the edges of clean point clouds due to training with noisy point clouds.

While PPSurf is better than the baselines in filling scan shadows, it is not a generative method and cannot generate new geometric detail in large missing regions. This limits the size of missing regions that can be filled with plausible geometry. Combining PPSurf with a generative model would be an interesting direction for future work. See Figure 4.11 for an example of inaccurately filled scan shadows.

4.6 Conclusion

In this paper, we have introduced PPSurf as a method for surface reconstruction from raw, unoriented point clouds. In contrast to previous methods, PPSurf incorporates

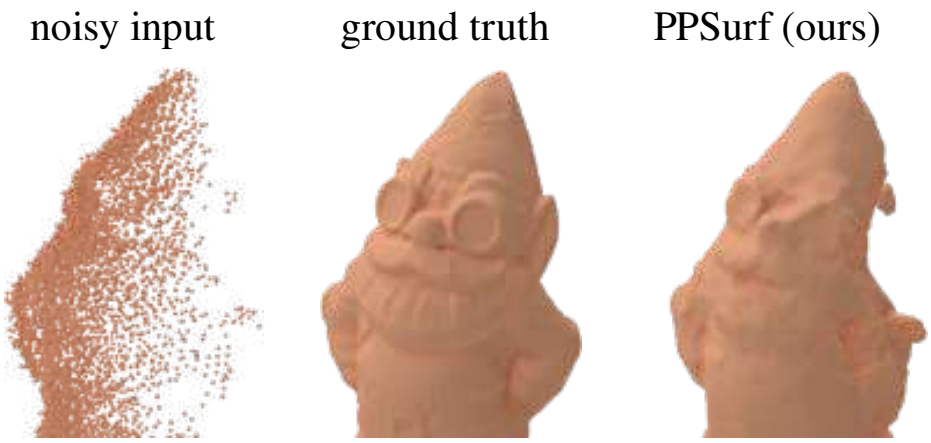


Figure 4.11: Limitations. Our method struggles with reconstructions of large missing areas in the input point cloud since we did not incorporate any generative model capabilities.

strong local and global priors learned from data. Whilst our global prior is based on a point convolutional neural network that processes the point cloud as a whole, fine details are preserved through the local prior based on dense local point cloud patches. We have shown in extensive studies that PPSurf is able to achieve better surface reconstructions than previous data-driven and non-data-driven methods, being more robust to noise in the input point cloud and preserving fine details at the same time.

This work gives us another puzzle piece to answer the research question, 'How can deep learning improve surface reconstruction?' The model must be accurate enough so that forwarding information through hierarchical structures does not just propagate errors but actually improves the surfaces. Test-time augmentation, as in POCO [BM22], eliminates most of the noise caused by the random subsampling, which is necessary for fixed-size model inputs. Modern learning techniques like point convolutions and attention improve the reconstruction further.

In the future, we would like to investigate how modern techniques borrowed from generative models could improve the obtained reconstruction from sparse point clouds where large parts of the shape are missing.

CHAPTER 5

LidarScout: Direct Out-of-Core Rendering of Massive Point Clouds

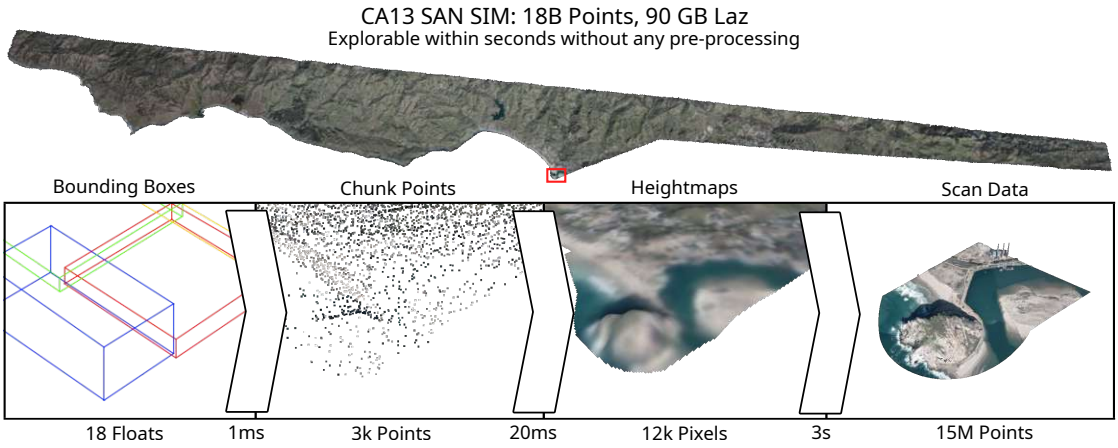


Figure 5.1: We present LidarScout, a method to explore huge, compressed point clouds within seconds. The example shown here contains the Morro Bay area focusing on Morro Rock, rendered with three heightmaps of 640m x 640m.

This chapter is based on our publication 'LidarScout: Direct Out-of-Core Rendering of Massive Point Clouds' published at High-Performance Graphics 2025 [EHWS25].

5.1 Motivation

After PPSurf, the research field had matured and many promising approaches had been explored. We worked on the difficult problem of generative inpainting for incomplete point clouds, which we had envisioned (see Section 4.6) already in parallel to PPSurf. However, it is not yet ready to be published. Luckily, we found an interesting application for surface reconstruction, instead. Since our group is involved in various kinds of point-cloud-related projects, we also have access to large point clouds, i.e., aerial LIDAR scans. Working with such amounts of data is slow and tedious, so we developed the idea of a viewer for large, compressed point clouds that requires no pre-processing and no additional disk space.

Many fields require huge terrain scans, including archeology, infrastructure, bathymetry, agriculture, forestry, flood and landslide prediction, geology, climate research, and many more. Improvements in laser scanners and frequent scanning operations (e.g., 3DEP [Sur18] and AHN [Ned23]) result in country-wide datasets comprising hundreds of billions to trillions of points. These are typically stored in a compressed format (LAZ) and can amount to terabytes. Visualizing these datasets requires out-of-core Level-Of-Detail (LOD) structures that allow loading only those tiny subsets necessary for a given viewpoint. However, generating these structures takes hours to days of preprocessing. For example, AHN2, a point cloud of the entire Netherlands, comprises 640 billion points, and constructing an LOD structure took Martinez-Rubi [MRVv+15] 15 days of processing.

With these huge amounts of data, tasks like viewing become non-trivial. Such supposedly simple tasks include having a quick overview, searching for obvious problems like outliers and noise, finding the relevant files for a specific region, and transferring the data. Compression can reduce transfer and storage problems, but it makes other tasks even slower. In this chapter, we develop LidarScout, a method that makes exploring huge aerial LIDAR scans fast and easy again.

Relating to the research question, 'How can deep learning improve surface reconstruction?', this viewer for large point clouds, which has potential for great practical use, requires shifting the focus from improving the quality of surface reconstruction to improving efficiency, and most of all, inference speed. After we strengthened the global prior for more quality in PPSurf, we now focus on local priors for the high efficiency necessary for LidarScout. Key aspects of the speed-up are efficient inputs and dimensionality reduction of the outputs. Knowing that the volume below the points is always *inside* allows us to take the most important and costly global prior out of the model. Therefore, we can work entirely patch-based with fast local priors.

Since the strong emphasis on efficiency was unfamiliar, finding good inputs and architectures for this task was difficult, as we describe in the next section.

5.2 Preliminary Approaches

While the journey to the publication was relatively straightforward in the case of LidarScout, we still had to find a good trade-off between quality and speed. Before diving into the method's details, we describe a few failed experiments in regard to the network inputs and architecture. The preliminary approaches finally led to LidarScout with its accurate and efficient CNN for textured heightmap estimation from sparse point sets.

5.2.1 Inputs for High-Speed Reconstruction

As LidarScout was intended to be an interactive tool, the reconstruction part had only a minimal computation budget. Further, it had to be a local-only operation that could focus on the visible region. It was clear from the beginning that only heightmaps could provide the required visual quality in the given time. Triangulation with linear interpolation had been used for surface reconstruction in similar cases. However, we deemed its hard discontinuities too bad and only suitable for a baseline. While PointNet was fast enough for PPSurf, quick tests showed that it was too slow for LidarScout. Also, its reconstructed heightmaps were unusable, clearly worse than linear interpolation. From this point on, we knew that the solution would require learning heightmaps with a small, image-based network.

However, CNNs are made for raster images and tend to fail with missing or sparse data. Rasterizing a local subset of the point cloud produced unusable results. Gated convolutions as in NPBG [ASK+20] improve the quality but not over linear interpolation. Quick experiments with NN and linear interpolation proved their potential for CNN inputs.

5.2.2 NN Architecture for High-Speed Reconstruction

After trying PointNet for this task, we tested a slightly modified DCT-Net [ZZX+22], adapted from depthmaps to heightmaps. We expected its spectral representation to be effective for dense raster inputs, and indeed, early experiments showed a better quality than linear interpolation. On our quest for minimal inference times, we wanted to try a small CNN based on U-Net [RFB15]. We expected a faster system at the cost of some quality, but it actually turned out even better than DCT-Net. After some minor issues with dataset bias, triangulation queries, and LibTorch setup, this model was ready for use in LidarScout.

5.3 Overview

Having ruled out inefficient inputs and architectures in the previous section, we arrive at LidarScout, a method that reduces the time it takes to visualize massive datasets from days down to seconds. After a user drops the dataset into the application, we quickly

read all tiles' bounding boxes as the only global operation. Afterward, we efficiently pick a sparse subsample of the compressed point cloud, generate rough heightmaps, refine them with a small neural network, and render them with a CUDA-based software rasterizer, all prioritizing the user's current viewpoint. When zooming in further, we stream the high-resolution scan data and update the hightmaps. LidarScout compares favourably with previous work regarding the reconstruction quality and the time to start the exploration.

Our main contributions are:

1. An interactive point cloud viewer for massive terrain scans that requires no preprocessing and no additional disk space.
2. Efficient extraction of a sparse subsample from compressed point clouds (LAZ).
3. A method to predict high-quality heightmaps from this sparse subsample.

5.4 Related Work

The related work for LidarScout diverges strongly from the other reconstruction methods due to its use case. While some techniques remain relevant, we focus on fast point-based rendering approaches, particularly of massive datasets, and surface reconstruction, especially those related to constructing heightmaps from sparse point samples. We also briefly explore neural rendering methods, as several point-based neural methods reconstruct high-quality images from sparse point samples, a problem similar to constructing heightmaps from sparse subsamples.

Surface Reconstruction Surface reconstruction aims to recover the underlying object that a point cloud was sampled from. Most surface reconstruction methods work in full 3D to generate a mesh or distance field. Only a few works target 2.5D and directly output heightmaps.

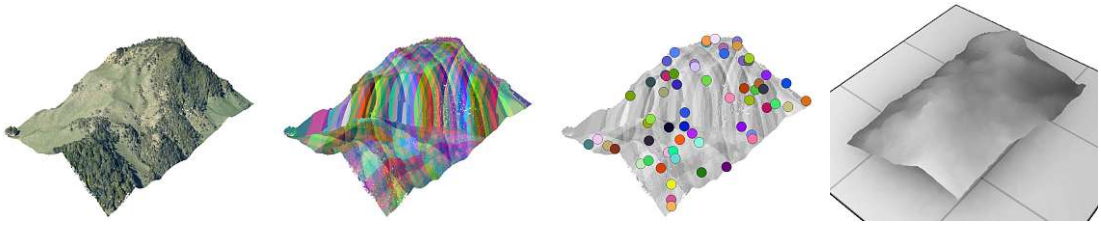
Reconstructing heightmaps from point clouds has seen little work in recent years. However, the field of depthmap estimation is very active, and many results can be adapted to heightmaps. Moving Least Squares [LS81] interpolates a smooth surface from scattered data points. The now classic approach for heightmap reconstruction is to generate a Delaunay triangulation and interpolate it linearly. This is done, e.g., in Las2Dem of the popular LAStools [rG15] suite, which was used for SwissS3D [Swi20], for example. Las2Dem also deals with multiple laser returns falling into the same texel. However, triangulation approaches suffer from the fundamental problem of interpolating within slender triangles. Closely related, most 2.5D reconstruction methods that work on depthmaps are in the context of single-view 3D reconstruction. Recent survey papers [RSL+24, MRC+22] describe the advances of this field from depth-cue-based methods via machine learning and hand-crafted features to deep learning. For other reconstruction methods, especially those targeting individual objects, see Section 2.2.

Overall, surface reconstruction from point clouds has seen significant advances in recent years, especially on the data-driven side. However, heightmap reconstruction for aerial LIDAR scans has been neglected.

Aerial LIDAR Storage The LAS and LAZ formats are specifically targeted towards aerial laser scanning and thus the two most commonly used formats for massive, country-wide aerial point cloud scans. LAZ is a compressed form of LAS that specifically takes advantage of common patterns in point clouds for efficient and lossless compression [Ise13]. For example, LAZ predicts the next position of a point based on the differences of the previous five points and then entropy encodes the difference between prediction and true position, which takes advantage of the fact that laser scanners observe points in a line-wise and thus predictable fashion.

Point-Based Rendering Levoy and Whitted [LW85] proposed points as a meta-primitive that all other surface primitives can be converted to, or to be directly generated from procedural functions. Since then, point cloud rendering has become a widely popular field due to the vast amount of point samples generated by laser scanners, and the resulting need for higher performance and better quality [KB04]. Surfels [PZVBG00] and EWA-Splatting [ZPVBG02] introduced high-quality rendering approaches for point-based primitives. Botsch et al. [BHZK05] propose an efficient GPU-based implementation for high-quality splatting. Günther et al. [GKLR13] and Schütz et al. [SKW22] improve the rendering performance via compute-based solutions that are faster than the triangle-oriented hardware pipeline. Schütz et al. [SMOW20] introduce a progressive rendering approach that renders random subsets each frame and converges towards the true result over the coarse of several frames. 3D Gaussian Splatting [KKLD23] proposes 3-dimensional Gaussians as a geometric primitive for 3D reconstruction and novel-view-synthesis. While some of the referenced works deal with point-based geometry that comprises position as well as orientation and size/scale, our method is targeted toward point clouds from aerial LIDAR scans, whose geometry is solely described by positions.

Point-Based Levels of Detail Rendering massive datasets requires LOD structures that reduce memory usage and increase performance. QSplat [RL00] proposes a bounding-sphere hierarchy as a means to render large splat models. Sequential Point Trees [DVS03] introduces a similar hierarchy but sequentializes it into an array that allows efficient rendering on the GPU by invoking a draw call for a subset of the array, replacing fine-grained hierachy traversal by a draw call to a batch of data. Instant Points [WS06] suggests a nested octree that allows for view-dependent LOD. Wand et al. [WBB+08] and Modifiable Nested Octree (MNO) [SW11] propose modifiable structures that enable efficient selection and deletion on large point datasets. Potree [SOW20] and Lidarserv [BDSF22] improve the LOD construction performance of MNOs. Lidarserv and SimLOD [SHW24] both propose incremental LOD construction algorithms that build and display the LOD structure while additional points are loaded. The former focuses on live-capture of point



(a) Tile (b) Chunks (c) Chunkpoints (d) Patches

Figure 5.2: Massive LIDAR data is stored in rectangular tiles. Tiles store data in compressed chunks of 50k points. Points in a tile are typically stored by timestamp, in this case indicating circular scanning patterns. Chunk points refer to the uncompressed first point of each chunk. Patches denote 640x640 meter (10m/pixel) heightmaps, created from the rapidly loaded chunk points.

clouds, while the latter focuses on GPU-accelerated LOD construction that can build the structure as fast as points can be streamed from disk.

The largest LOD construction study for point clouds that we are aware of was made by Martinez-Rubi et al. – 640 billion points converted to an MNO in 15 days [MRVv+15].

Neural Point Cloud Rendering Neural rendering uses a neural network to synthesize images for given parameters, rather than generating and rasterizing geometry. Tewari et al. [TFT+20] give an overview in their STAR. Neural Point Cloud Rendering was just a side-note in 2020, with Neural Point-Based Graphics [ASK+20] being the only mention. First, they compute feature vectors for the given points. Then, they rasterize them as high-dimensional points in multiple resolutions. Finally, they feed these raw images into a U-Net [RFB15], which outputs a rendering. Since then, many methods have been proposed [KPLD21, NKH+21, WWG+21, RFS22, RALB22, YGL+23, HFF+23, FRFS24, HFK+24], improving various aspects of the approach. These methods share one drawback: they are made for relatively dense point clouds, typically produced by photogrammetry, and many require camera poses, which are not available in our application.

5.5 Method

LidarScout consists of five stages: Quickly loading a sparse subsample of the entire point cloud; generating heightmaps with textures; refining them for the user's viewpoint; loading full-res data in close-up viewpoints and updating heightmaps with full-res data to retain a medium level of detail; and rendering them with a CUDA software rasterizer.

5.5.1 Data and Data Structures

Massive aerial LIDAR datasets are typically distributed using the compressed LAZ format (e.g., OpenTopography [Pac13], AHN5 [Ned23], 3DEP[Sur18], etc.). For this paper's

evaluation, we selected three point clouds from the USA, one from New Zealand, and one from Switzerland, as shown in Table 5.1. The sizes range from 1.6 billion points (ID15_BUNDS) to 262 billion points (Gisborne+Addendum). The full Switzerland dataset would exceed Gisborne (reportedly at least 16 TB), but due to lack of storage we selected a subset of 18.7 GB. In the context of aerial LIDAR scans, a point's geometry is solely defined by its position, unlike other point-based primitives such as Surfels or Gaussian Splats. In many cases, they also lack colors, which is why we include the dataset of Switzerland as a representative example.

In this chapter, we will regularly refer to tiles, chunks, chunk points, and patches, as illustrated in Figure 5.2. **Tiles** correspond to individual LAZ files. Massive LIDAR datasets are almost always organized in such rectangular tiles, typically covering several hundreds to a thousand meters, and storing several millions of points. **Chunks** are an important concept in the LAZ compression algorithm. LAZ compresses points in chunks of 50 000 points. The first point in each chunk is uncompressed, and subsequent points must be decoded sequentially. Multiple chunks may be decompressed in parallel. **Chunk Points** refer to the first uncompressed point in each chunk. These are important in our approach since they are the only ones we can quickly access without the need to set up an expensive arithmetic decoder. **Patches** are quadratic 640 meter x 640 meter regions for which we reconstruct a 64 x 64 pixel heightmap from the surrounding sparse chunk points.

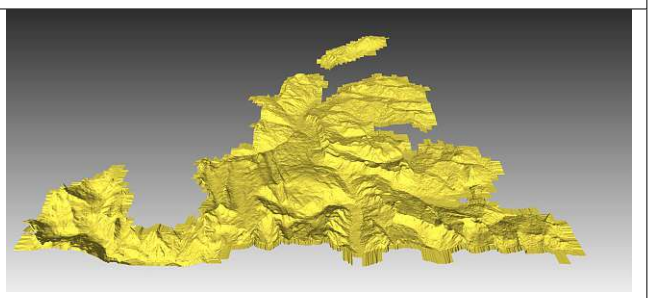
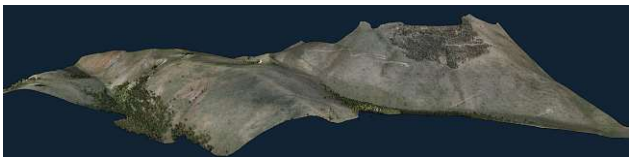
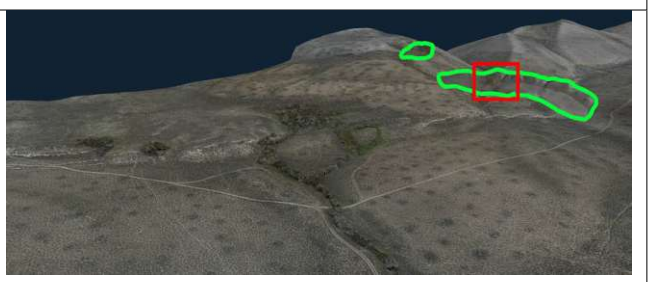
Dataset	Image
CA13 OpenTopography 17.7 B Points 80 GB LAZ 20 $Points/m^2$ LIDAR (linear) Screenshot: 56M points	
SwissS3d Swisstopo 18.7 B Points 18 TB LAS 17 $Points/m^2$ No colors LIDAR (linear)	
Bund_Bora OpenTopography 6.5 B Points 40 GB LAZ 574.22 $Points/m^2$ Photogrammetry	
ID15_Bunds OpenTopography 1.6 B Points 14.8 GB LAZ 387.24 $Points/m^2$ Photogrammetry	
Gisborne+Addendum OpenTopography 95 + 167 B points 850 + 1 550 GB LAZ 30.25 $Points/m^2$ LIDAR (circular) Screenshot: 248M points	

Table 5.1: Datasets used for LidarScout. The table shows close-up screenshots, the entire map with green outline, and the close-up's area marked as red box.

5.5.2 Rapidly Creating an Overview for Billions of Points

In the first stage, we initialize an overview of the entire dataset with a sparse subsample, which poses two challenges: File I/O is optimized for loading sequential data instead of random subsamples. Also, massive datasets are typically compressed sequentially, which further limits our ability to access random sparse subsets.

On modern SSDs, the first challenge is addressed by investigating their 4 kB random access performance. Similar to accessing RAM [Dre07] or GPU memory [Har13], SSDs are also optimized for coalesced access to a range of bytes rather than a few individual bytes. In case of SSDs, access is typically grouped into sectors of 4 kB, i.e., fetching a single point from disk is about as fast as accessing all points in a sector. However, only the first is uncompressed and thus easily accessible. Modern SSDs are capable of reading about 1 million random 4 kB sectors per second, so we are theoretically able to load a subsample of 1 M points of an arbitrarily large dataset – an arguably sufficiently large subset for an overview viewpoint on a 2-megapixel monitor. For example, the model shown in Figure 5.1 depicts the heightmap model that was constructed from that dataset's $\frac{18B}{50k} = 360k$.

The second challenge – the industry standard LAZ compression format for massive aerial LIDAR data – puts a limit on the way we can load the initial sparse sample. LAZ uses arithmetic coding to sequentially compress points one after the other, and in turn we also need to decompress them sequentially [Ise13]. Fortunately, points are compressed in chunks of typically 50 k points, and the first point in each chunk remains uncompressed, thus we are able to quickly load a sparse subsample made of every 50 000th point. Since LAZ is variable-rate compressed, byte locations of each uncompressed chunk point must be obtained by first reading the file's chunk table.

After all chunk points are loaded, we have a sparse subsample of the entire dataset that is sufficiently dense in an overview perspective. When zooming in, holes between points will appear. Since massive aerial LIDAR datasets constitute 2.5D data until one zooms in closely, we propose to fill these holes by constructing high-quality heightmaps from the sparse set of chunk points. We divide the entire map into patches, covering 640x640 meters each, and for each patch, we construct a 64x64 pixel textured heightmap.

5.5.3 Interpolated Heightmaps

The goal of this part is to convert a region of chunk points to rough, textured heightmaps, which we will later refine with a neural network. For any patch of 64x64 pixels (10m/pixel) we first construct two 96x96 pixel heightmaps using nearest-neighbor (NN) and linear interpolation of the triangulated chunk points. The additional padding is applied to avoid seams between adjacent learned heightmaps.

We receive $\mathcal{P}_{ms}$, the relevant chunk points for the current patch, from a grid-based data structure. We transform these points from model space to patch space as follows: $\mathcal{P} = (\mathbf{p} - \mathbf{c})/r, \mathbf{p} \in \mathcal{P}_{ms}$, where $\mathbf{c} \in \mathbb{R}^2$ is the 2D patch center and $r \in \mathbb{R}$ is the padded

patch radius. The triangulation and interpolation for the heightmaps and textures work as described in Algorithm 5.1, omitting minor implementation details and optimizations for clarity. At the core, we fill a face map indicating which triangle of the triangulation covers which pixel. This is mostly done by performing Flood-Fill from the triangle centroids. Due to rasterization, some pixels of very slender triangles may be disconnected. We catch those in a second Flood-Fill step, started at every pixel inside the bounding box of the triangle. The Flood-Fill works in an 8-connected neighborhood. It fills pixels if they are inside a triangle by checking their barycentric coordinates. After collecting the triangle IDs, we interpolate linearly, again with barycentric coordinates. This is only possible inside the convex hull of the triangulation. Therefore, we fall back to NN interpolation on the outside using a KD-Tree of $\mathcal{P}$ for the necessary speed. The NN interpolation uses the same KD-Tree for all pixels.

Algorithm 5.1: Patch-Space Chunk Points to Heightmaps

Input : Chunk points $P = \{p_i\}$ (with $p_i \in [-1, 1]^2$), height values $h = \{h_i\}$, (optional) color $rgb = \{c_i\}$, grid resolution res
Output : Heightmaps $\texttt{hm}_{\text{nn}}$, $\texttt{hm}_{\text{lin}}$, mask `face_map`

`// Initialization`
$N \leftarrow res^2$
$K \leftarrow \text{KDTree}(P)$
Add corner padding points (with NaN values) to P, h and rgb
$\mathcal{T} \leftarrow \text{DelaunayTriangulate}(P)$
$G \leftarrow$ generate regular grid of $res \times res$ points over $[-1, 1]^2$
Initialize mask array $\texttt{face\_map}[1..N] \leftarrow -2$
Allocate $\texttt{hm}_{\text{nn}}[1..N]$ and $\texttt{hm}_{\text{lin}}[1..N]$
`// FloodFill from Triangle Centroids`
foreach *triangle* T_i *in* $\mathcal{T}$ **do**
  $c_i \leftarrow$ centroid of T_i
  $g_{\text{id}} \leftarrow$ index of G-grid point closest to c_i
  `FloodFill` $(g_{id}, i, face\_map, \mathcal{T}, G, res)$
end
`// FloodFill from Disconnected Rests`
foreach *triangle* T_i *in* $\mathcal{T}$ **do**
  $B_i \leftarrow$ bounding box of T_i intersected with $[-1, 1]^2$
  foreach *grid point* g_j *in* B_i **do**
    if $face\_map[j] = -2$ ***and*** $g_j \in T_i$ **then**
      `FloodFill` $(j, i, face\_map, \mathcal{T}, G, res)$
    end
  end
end
`// Remove Face IDs of Padding Triangles`
for $j \leftarrow 1$ **to** N ***where*** $face\_map[j] \geq 0$ **do**
  $i \leftarrow \texttt{face\_map}[j]$
  if *any vertex of triangle* $\mathcal{T}[i]$ *is a padding vertex* **then**
    $\texttt{face\_map}[j] \leftarrow -1$
  end
end
`// Linear Interpolation in Convex Hull`
foreach j *with* $face\_map[j] \geq 0$ **do**
  $i \leftarrow \texttt{face\_map}[j]$
  $T \leftarrow \mathcal{T}[i]$
  bary $\leftarrow$ barycentric coordinates of $G[j]$ in T
  $\texttt{hm}_{\text{lin}}[j] \leftarrow$ interpolate h at triangle vertices of T via bary
    `// Optionally: interpolate color` rgb `via barycentric coordinates`
end
`// Linear Interpolation outside Convex Hull`
foreach j *with* $face\_map[j] = -1$ **do**
  $\texttt{hm}_{\text{lin}}[j] \leftarrow$ nearest neighbor interpolation on h via K at $G[j]$
    `// Optionally: interpolate color` rgb `via nearest neighbor`
end
`// NN Interpolation`
for $j \leftarrow 1$ **to** N **do**
  $\texttt{hm}_{\text{nn}}[j] \leftarrow$ nearest neighbor interpolation on h via K at $G[j]$
end
`// Finish Up`
Remove padding points from P, h, rgb if needed
return $hm_{nn}, hm_{lin}, face\_map$

5.5.4 Learned Heightmaps

The linearly interpolated patches could be used directly for rendering. However, discontinuities and interpolation across slender triangles reduce the visual quality. Therefore, we employ a small neural network that produces accurate and visually pleasing interpolations.

The input of our CNN consists of two 96x96 heightmaps and two 96x96x3 RGB textures, both linearly and NN interpolated. It outputs a 64x64 heightmap and a 64x64x3 RGB texture. The 16 additional texels in each direction give the network context information, so it can produce smooth patch seams. Redundancy in the outputs could smooth the seams further but is not necessary in practice. We batch inference calls to avoid kernel overhead and context switches, increasing efficiency.

Since we need to keep every step of our method local, the model must not depend on the global context. Therefore, we normalize the given heights to patch space with $z = (z_{ms} - c_z)/r$, where z_{ms} is a height in model space (meters above sea level) and c_z is the height of the patch center. This normalization also helps avoid numerical problems in the network. Since the patch centers are created from a grid on the XY-plane, we take c_z from the linearly interpolated heightmap's center. This means that the network cannot know how far above sea level a patch is, which makes it more general but also makes color estimation more difficult.

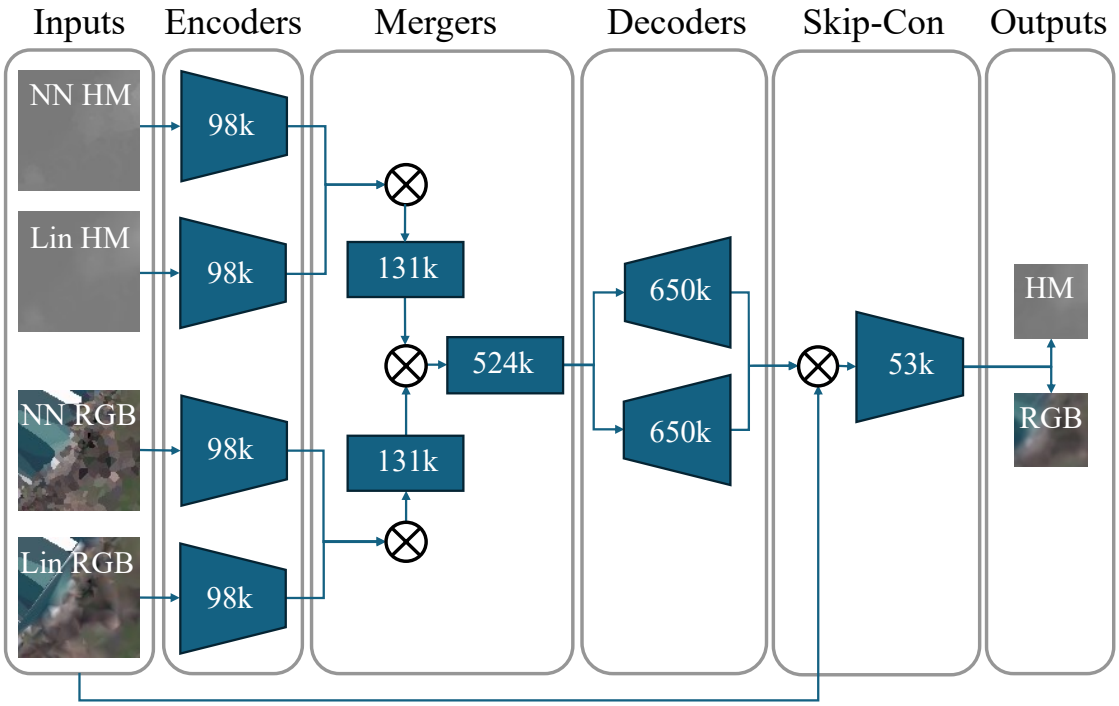


Figure 5.3: LidarScout architecture. The network predicts clean heightmaps and textures from rough ones using a combination of encoders, fully-connected layers, and decoders.

Figure 5.4: Example patch with inputs, processing steps, and network prediction.

Our network architecture (see Section 5.3) is inspired by the U-Net [RFB15], which is also used in Neural Point-Based Graphics (NPBG) [ASK+20]. Unlike NPBG, we encode each input independently with several convolution layers, which increases the model size. However, since we work with dense inputs, we do not need gated convolutions, making our network smaller. Next, we merge the produced feature vector with separate, fully connected layers in two steps. Then, we decode the feature vector with two separate CNNs to heightmap and texture. In contrast to NPBG, we only have one skip connection that concatenates the original inputs and the decoder outputs along their channel dimension. Another CNN reduces this tensor again to the final number of channels. Finally, we take the center 64x64 region for further usage in rendering. In total, the network has 2.5M learnable parameters, 500k more than NPBG. Figure 5.4 shows one example patch with its chunk points, triangulation, interpolations, prediction, and ground truth. The GT cannot be reached from the available sparse chunk points. Compared to linear interpolation, our network prediction is smoother, usually more accurate, and provides better transitions for our implicit level of detail. Note the red outlier at the middle bottom, which interpolation and simple smoothing preserve, but our CNN manages to ignore.

We train LidarScout only on CA13 and SwissS3d, while we evaluate it on all the datasets described in Section 5.5.1 to show its generalizability. For each dataset, we generate our ground-truth data for training and evaluation from these scans. We choose the query points randomly from the entire point cloud. 7000 of them are for training, 3000 for evaluation. For CA13 and SwissS3d, we split the query points by the 70-percentile of their x coordinates into train and test sets. For each query point, we sample a heightmap and a texture by taking the mean of all points that fall onto a texel, which also removes most of the original scanning noise. To simulate the chunk points of LAZ, we randomly select a 50,000th of all points.

We train our network with MSE loss, supervised by the ground-truth patches. We set loss elements corresponding to NaN elements in the GT (gaps in the original scans) to zero. The heightmap and texture losses are clipped to $(0.0 \dots 1.0)$ and averaged. We tried loss functions that put more weight on tile seams or gradients, but they did not make a noticeable difference. L1 loss and SSIM produce very similar, smoothed results, and LPIPS loss creates stripe artifacts. The bad results with LPIPS are likely due to a low number of top-down landscape images in their datasets and our images having a very low resolution. Our optimizer is AdamW (lr = 0.0001, betas = (0.9, 0.999), eps =

1e-5, weight decay = 1e-2, amsgrad = False). With a step scheduler (gamma = 0.1, steps at 25 and 50 epochs), we train for 75 epochs. The training on CA13 and SwissS3D takes about 25 minutes. Training is done in Python using PyTorch, and the inference in C++/CUDA using LibTorch.

5.5.5 Full-Resolution Tiles

Upon navigating close to the surface, we additionally load and display full-resolution point data from tiles with a sufficiently large screen-space bounding box. In our test datasets, tiles typically hold about 1 to 50 million points. Close-range viewpoints such as in Figure 5.5 may require loading about 50-300 million points, but nowadays, compute-based brute-force software rasterizers are capable of rendering up to two billion points in real time [SKW22]. Tiles that are no longer in focus are unloaded to free memory for other tiles. However, we update the heightmap textures to preserve some information for medium zoom levels.

5.5.6 Rendering

We need to render chunk points for any patch whose heightmap is not yet ready, followed by rendering textured heightmaps, and eventually by rendering the full-resolution point cloud data upon zooming in close to a tile. Rendering of points and heightmaps was implemented in a CUDA-based software rasterizer.

For points, we use the approach by Schütz et al. [SKW22]: We launch one thread-block comprising 256 threads for each chunk of 50k points. The threads iterate through all points, projecting them to screen and encoding their depth and color value into a 64 bit integer. We then use a 64 bit atomicMin to evaluate the point with the smallest depth value for each pixel. Afterward, a screen-space resolve pass extracts the color value from the least significant bits of each pixel's 64 bit depth and color value, and stores the result in an OpenGL texture for display.

Heightmaps are rendered with a custom CUDA-based triangle-rasterizer: We invoke a cooperative kernel with 64 threads per block. Each block processes 32 triangles at a time, projects them to screen, and computes the screen-space bounding box. For any triangle that covers less than 1024 pixels, a single thread of the group iterates over the pixels, evaluates the barycentric coordinates. If they indicate that the pixel is inside the triangle, it draws the fragment using the same 64 bit atomic-min logic as the point rasterizer. If a triangle is larger than 1024 pixels, it is added to a queue. After the block finishes rendering the smaller triangles in one thread per triangle, it continues to render the large triangles utilizing all 64 threads for each triangle, one after the other. The thread blocks continue to loop until all triangles of all heightmaps are rendered.

5.6 Results

We evaluate two use cases for our method and compare with several baselines: exploring large point clouds quickly and estimating the surface accurately from local subsamples.

5.6.1 Exploring Large Point Clouds

Table 5.2: Time to construct an LOD structure in Potree vs. time to completely finish each stage in LidarScout. After loading all tiles' metadata, loading the chunk point and the stages of heightmap construction operate progressively, so users can already navigate the dataset before they are finished.

Dataset	Potree	LidarScout		
		Tiles	Chunk Points	heightmaps
CA13	28m44s	0.02s	1.9s	23s
Gisborne	19h56m	0.10s	12.8s	159s
Gisborne+Add	-	0.19s	53.0s	701s

In order to display massive datasets that do not fit in GPU memory, state-of-the-art solutions require creating LOD structures in a preprocessing step. Potential solutions include Entwine [Ent21], Potree [SOW20], MassivePotreeConverter [MRVv+15], and Lidarserv [BDSF22]. We will study the performance of LidarScout in comparison to Potree, the fastest of the related approaches. For rendering, we use this test system: RTX 4090; AMD Ryzen 9 7950X 16-Core; Crucial T700 4TB PCIe Gen5. We show screenshots of such explorations in Figures 5.5 and 5.6.

Case Study: CA13 (17.7 billion points).

As shown in Table 5.2, Potree takes 28m44s until LODs are constructed and the dataset can be explored. LidarScout takes 0.02s to load the metadata of 2336 tiles and another 1.9s until all chunk points are loaded. Heightmap construction takes 23s to finish, but construction prioritizes the current viewpoint so users do not need to wait to see meaningful results.

Case Study: Gisborne (95 billion points).

LidarScout takes 12.8s until a subsample of 1.9 million chunk points has been loaded in order to display an overview of the entire dataset. Heightmaps are then constructed based on the user's current viewpoint. In comparison, users would traditionally have to wait 19h56m to construct an LOD structure before being able to explore the dataset. The LOD structure that was constructed by Potree required 1.7TB of additional disk space.

Case Study: Gisborne+Addendum (262 billion points)

We were not able to evaluate Potree's performance due to lack of additional disk space for the constructed LOD data. Extrapolating from Gisborne without Addendum, Potree would presumably require 2 days and 6 hours to finish LOD construction.

Although LidarScout takes 53s to load the chunk points of the entire overview, users can already start exploring the dataset as soon as the tile metadata is loaded. Chunkpoints are loaded progressively so users may navigate to already prepared regions, and a list of files/tiles allows users to zoom towards specific tiles, which are then loaded in full resolution even before all chunk points are loaded or heightmaps are generated.

Figure 5.5: Screenshots of CA13 (17.7B points, 90GB), Gisborne (262B points, 2.4TB) and CA21_Bunds (8.4B points, 96GB) made with LidarScout.

5.6.2 Surface Reconstruction

Few methods for surface reconstruction are applicable to our use case. Recent and popular global reconstruction methods, such as BallMerge [POEM24], Ball Pivot [BMR+99], Screened Poisson Surface Reconstruction [KH13], and PPSurf [EFPH+24] perform poorly with our chunk points, as shown in Figure 5.7.

Figure 5.8 shows a qualitative comparison of the most relevant local reconstruction methods. Linear interpolation has hard discontinuities and noisy colors (see also Fig. 5.4). Cubic interpolation suffers from overshooting, causing very bright and dark spots, and sometimes peaks of several hundred meters. High-Quality Splatting [BHZK05] either creates blobby structures, smoothed stairs, or gaps due to the fixed-size kernel.

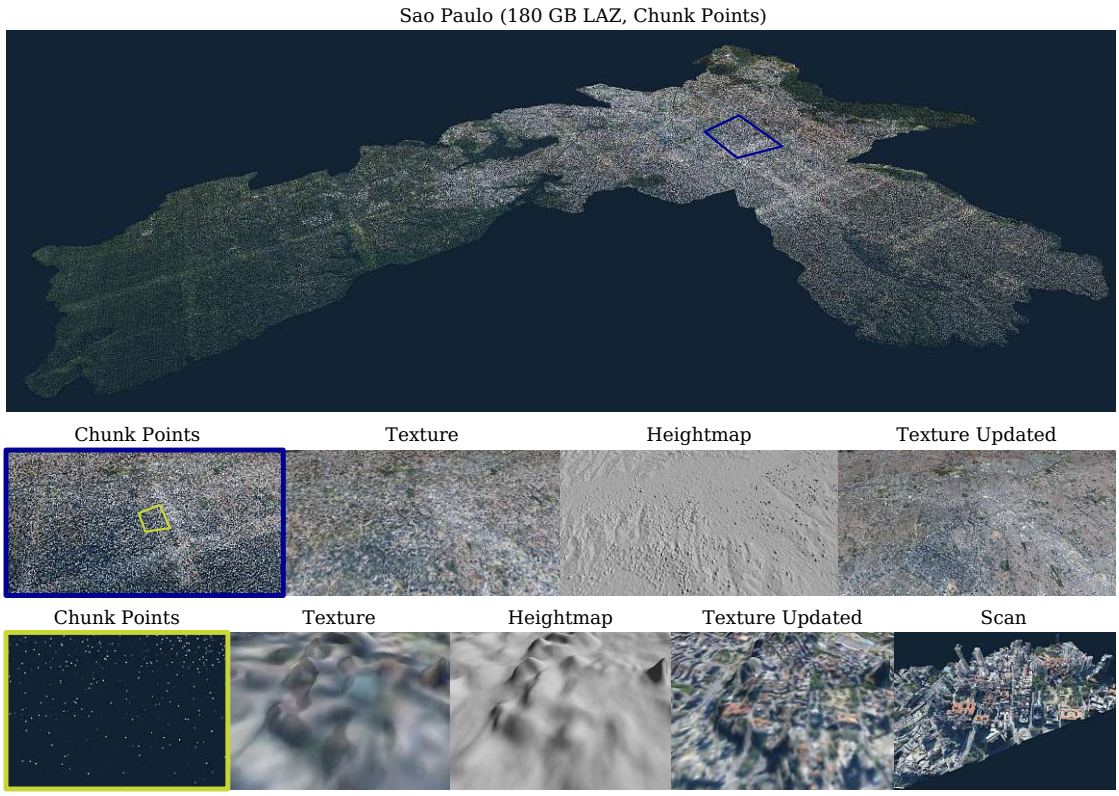


Figure 5.6: Sao Paulo. Screenshots made of the urban center using LidarScout. Source: Brazil LIDAR Survey 2017 `https://portal.opentopography.org/lidarDataset?opentopoID=OTLAS.062020.31983.1`

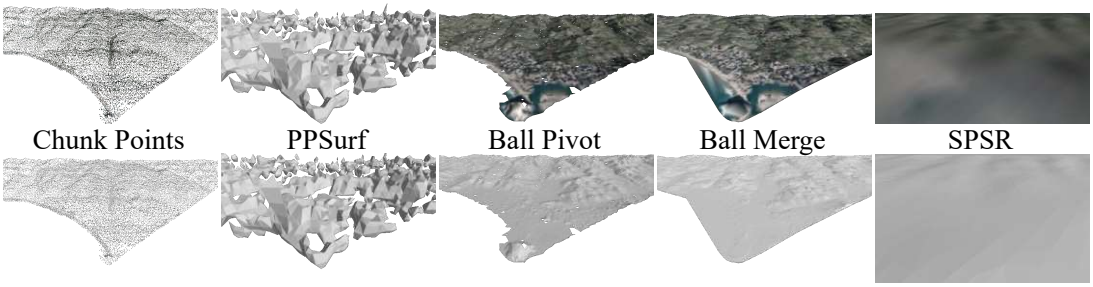


Figure 5.7: Qualitative comparison of several global reconstruction methods.

NPBG [ASK+20] with our inputs is close to LidarScout in quality but distorts colors sometimes.

Quantitative Comparison Tables 5.3 and 5.4 show the performance of LidarScout on all datasets. We report the average over all patches in the test sets. We use Root Mean

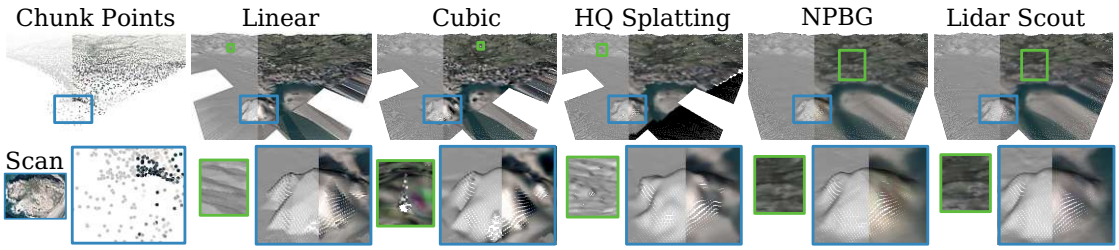


Figure 5.8: Qualitative comparison of the Morro Rock region in CA13.

Squared Error (RMSE) in meters to compare heightmap quality and Peak Signal-to-Noise Ratio (PSNR) in dB for textures.

Table 5.3: Root Mean Square Error (lower is better) of predicted heightmaps. Best results are in bold, second best underlined.

Dataset	Cubic	HQSplat	Linear	LidarScout
CA13	4.36±0.72	5.47±0.69	4.17±0.62	**3.81±0.55**
SwissS3d	11.60±2.14	12.58±2.16	9.26±1.44	**8.56±1.39**
Bund_Bora	1.40±0.82	1.97±0.31	**1.00±0.60**	1.16±0.17
ID15_Bunds	**0.65±0.19**	1.73±0.26	0.89±0.23	1.22±0.21
Gisborne_A	11.38±4.71	7.43±0.93	6.97±1.12	**6.55±0.75**
Gisborne_B	11.32±1.77	6.83±0.41	7.16±0.35	**6.26±0.34**
Gisborne_C	7.41±1.95	6.40±0.48	5.51±0.46	**5.09±0.33**
Mean	6.87±1.76	6.06±0.75	4.99±0.69	**4.66±0.53**

Table 5.4: Peak Signal-To-Noise Ratio (higher is better) of predicted textures. Best results are in bold, second best underlined.

Dataset	Cubic	HQSplat	Linear	LidarScout
CA13	66.08±1.26	69.83±1.14	69.38±1.14	**70.76±1.09**
Bund_Bora	69.57±1.11	**78.42±1.19**	73.29±0.97	77.45±0.86
ID15_Bunds	67.60±3.44	**78.79±0.80**	75.02±0.53	77.89±0.87
Gisborne_A	66.98±1.15	71.30±1.03	70.80±0.95	**72.24±0.77**
Gisborne_B	64.54±0.52	70.86±0.63	68.67±0.54	**71.46±0.48**
Gisborne_C	65.84±0.67	70.47±0.68	69.67±0.57	**71.22±0.52**
Mean	66.77±1.36	73.28±0.91	71.14±0.79	**73.50±0.77**

Our most important baseline is also used as input to our network: linearly interpolated heightmaps and textures. We perform Delaunay triangulation and linear interpolation with barycentric coordinates on the local chunk points, as described in Section 5.5.3. For such a simple baseline, it is surprisingly good. It is also an approximation for Rapidlasso's

Las2Dem[rG15], which takes care of multiple returns per texels in addition. However, this refinement does not make a noticeable difference with our sparse chunk points. We also compare with a cubic Clough-Tocher interpolation implemented in SciPy [SA11], which is accurate in many cases but occasionally overshoots. Lastly, we compare to High-Quality Splatting [BHZK05] by treating each chunk point as a large, fixed-size splat, and using a Gaussian blending function to obtain a smooth transition of heights and colors between overlapping splats. LidarScout performs significantly better than the baselines except for the much denser Bund_Bora and ID15_Bunds datasets.

Computation Time and Memory Consumption Reconstruction was evaluated on an NVIDIA RTX 3090 and an AMD Ryzen 7 3700X 8-Core. The reconstruction of one patch in our C++/CUDA framework takes around 20 ms. Single-threaded triangulation and sampling take 16 ms, inference and buffer copies 4 ms. Batched inference requires copying data to make the input heightmaps and textures contiguous in memory, which means a small overhead. In any case, the timings show that batching always pays off. See the detailed timings of the C++/CUDA inference in Table 5.5.

Table 5.5: Comparison of inference times with different batch sizes. The numbers shown are the median over 1000 runs of inference calls surrounded by device synchronization. In short, batching always pays off.

Batch Size	Time [ms]	Per Tile [ms]
None	4.2	4.2
1	4.2	4.2
2	4.3	2.2
3	4.6	1.5
4	4.7	1.2
5	4.9	1.0
10	6.4	0.6

Ablation Table 5.6 shows an ablation that empirically validates our design choices, comparing different network architectures and inputs. Other architectures like **NPBG** [ASK+20] and **DCTNet** [ZZX+22] perform worse than ours. **Raster**izing points and filling unknown pixels with zeros does not work well as input for our image-based network. This means that dense inputs are necessary for viable quality. The model draws information from both nearest-neighbor (**NN**) and **lin**ear interpolation inputs, especially for the heights. **Lin**ear-only generalizes better across point densities, while **NN**-only is better with similar densities. Combining them is a step towards the best of both. Omitting RGB inputs (**HM only**) has a negligible impact on heightmap quality. Adding **Extra Data** data (ID15_Bunds and Gisborne in addition to CA13 and SwissS3D) improves the quality significantly. Note that only Bund_Bora is completely unseen, making a fair comparison difficult for this variant. Please see Tables 5.7 and 5.8 for detailed statistics.

Table 5.6: Ablation study main results. Best results are in bold, second best underlined. Note that the **Extra Data** variant was partially trained on the test data, so its metrics are positively biased.

Variant	Heights RMSE [m] ↓	Colors PSNR [DB] ↑
NPBG	4.87±0.64	73.30±0.77
DCTNet	4.82±0.56	73.20±0.77
Raster	14.91±3.30	60.98±1.07
HM Only	4.64±0.55	NA
NN only	4.84±0.55	73.50±0.77
Lin only	4.71±0.55	73.53±0.76
Extra Data	**4.56±0.52**	**74.16±0.78**
LidarScout	4.66±0.53	73.50±0.77

Table 5.7: Ablation study full results. We show the RMSE of the heightmaps produced by LidarScout with various changes. Note that Linear interpolation is still better than Extra Data for reconstructing heights (not colors) of the photogrammetry datasets. This indicates a significant bias towards typical point densities of LIDAR scans, which LidarScout cannot fully generalize across. Adapting the patch size solves this issue.

Dataset	NPBG	DCTNet	HM Only	Rast	NN only	Lin only	Extra Data	LidarScout
CA13	3.92±0.53	**3.80±0.55**	3.84±0.55	6.04±0.85	4.10±0.55	3.84±0.56	3.82±0.54	3.81±0.55
SwissS3d	9.25±2.01	8.87±1.52	8.70±1.43	14.07±2.96	9.34±1.55	8.58±1.38	**8.55±1.39**	8.56±1.39
Bund_Bora	1.39±0.19	1.55±0.21	**1.13±0.19**	39.32±11.50	1.28±0.20	1.27±0.18	**1.13±0.17**	1.16±0.17
ID15_Bunds	1.37±0.18	1.54±0.31	1.19±0.27	21.88±5.62	1.21±0.17	1.29±0.28	**1.05±0.18**	1.22±0.21
Gisborne_A	6.70±0.87	6.48±0.66	6.44±0.72	8.39±1.16	6.55±0.68	6.60±0.76	**6.40±0.72**	6.55±0.75
Gisborne_B	6.31±0.35	6.32±0.34	6.19±0.35	7.65±0.34	6.14±0.35	6.26±0.34	**6.00±0.34**	6.26±0.34
Gisborne_C	5.14±0.33	5.18±0.33	5.03±0.34	7.05±0.67	5.25±0.33	5.12±0.34	**4.98±0.32**	5.09±0.33
Mean	4.87±0.64	4.82±0.56	4.64±0.55	14.91±3.30	4.84±0.55	4.71±0.55	**4.56±0.52**	4.66±0.53

Table 5.8: Ablation study full results. We show the PSNR of the textures produced by LidarScout with various changes.

Dataset	NPBG	DCTNet	HM Only	Rast	NN only	Lin only	Extra Data	LidarScout
CA13	70.61±1.05	70.57±1.05	NA	69.61±0.99	70.68±1.08	70.56±1.06	70.74±1.10	**70.76±1.09**
SwissS3d	NA	NA	NA	NA	NA	NA	NA	NA
Bund_Bora	77.03±0.83	76.57±0.88	NA	40.87±1.57	77.31±0.89	77.67±0.88	**78.20±0.91**	77.45±0.86
ID15_Bunds	77.48±0.82	77.07±0.89	NA	45.08±1.74	77.72±0.81	78.13±0.82	**79.26±0.76**	77.89±0.87
Gisborne_A	72.07±0.91	72.20±0.78	NA	70.14±1.15	72.31±0.78	72.24±0.77	**72.57±0.77**	72.24±0.77
Gisborne_B	71.58±0.49	71.51±0.47	NA	70.29±0.53	71.69±0.50	71.45±0.49	**72.42±0.55**	71.46±0.48
Gisborne_C	71.04±0.50	71.27±0.53	NA	69.87±0.45	71.27±0.53	71.17±0.53	**71.77±0.57**	71.22±0.52
Mean	73.30±0.77	73.20±0.77	NA	60.98±1.07	73.50±0.77	73.53±0.76	**74.16±0.78**	73.50±0.77

We can modify LidarScout to receive only height inputs but force it to output heights and colors. However, this colorization model is not very accurate, as shown in Figure 5.9. A generative model running on larger regions could produce good results in the future.

(a) Colorizer (b) Extra Data

Figure 5.9: Colorization comparison on Gisborne.

5.6.3 Discussion and Limitations

Lidarserv [BDSF22] and SimLOD [SHW24] are prior work that follow similar goals: Exploring large datasets without the need to preprocess and wait. Lidarserv specifically aims to enable displaying arbitrarily large point clouds directly during capture, and is able to construct out-of-core LOD structures at rates of up to around 1.8 million points per second. SimLOD aims to visualize large point cloud files quickly and is capable of loading industry standard LAS files at rates of up to 300 million points per second, or compressed LAZ files at up to 30 million points per second. Both display points immediately as they are streamed, without the need to wait until processing is finished. A major difference in our approach is that we aim to display massive datasets in their entirety in a matter of seconds and prioritize more detailed reconstructions towards the user's viewpoint, while the prior works operate on local regions in undefined order without priorization. SimLOD is further limited to datasets that fit in memory, i.e., about 800 million points per 24GB of memory. Our approach, on the other hand, rapidly displays arbitrarily large datasets but lacks level-of-detail structures that would further improve rendering performance, especially for previously visited regions. In the future, we would like to integrate and expand SimLOD's incremental LOD construction in order to build an out-of-core system that is capable of rendering arbitrarily large point clouds with instant overviews of the entire dataset, and priorization towards the current viewpoint.

Although we trained LidarScout only on CA13 and SwissS3d, it generalizes well to other regions of the world. The model has mostly seen colors typical for deserts, small cities, and beaches from the Morro Bay region in California, USA. Nonetheless, it can still reconstruct the colors of the lush vegetation of Gisborne, New Zealand, as shown in Table 5.4 and Figure 5.5. Furthermore, it generalizes well across scan patterns that may affect the sparse subsample. It was trained on the line-wise scanning patterns of

the CA13 aerial LIDAR, but it has no issues with the circular scanning patterns of Gisborne.

Linear and cubic interpolation are competitive on the much denser photogrammetry point clouds of ID15_Bunds and Bund_Bora for predicting heights. However, only HQ-Splatting can compete with our method when estimating colors. This indicates that our method does not generalize too well from point densities of around 20 $points/m^2$ in CA13 to almost 600 $points/m^2$ in Bund_Bora and fails to use the available information. Adapting the patch size solves this issue, as shown in Figures 5.10 and 5.11.

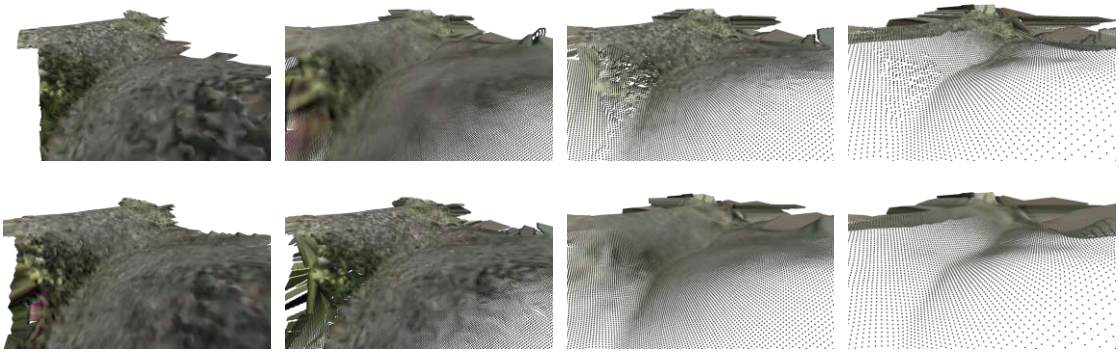


Figure 5.10: Reconstruction of Bund_Bora with different resolutions. Upper: linear interpolation. Lower: LidarScout. Left to right: 1, 2.5, 5, 10 (default) meters per texel.

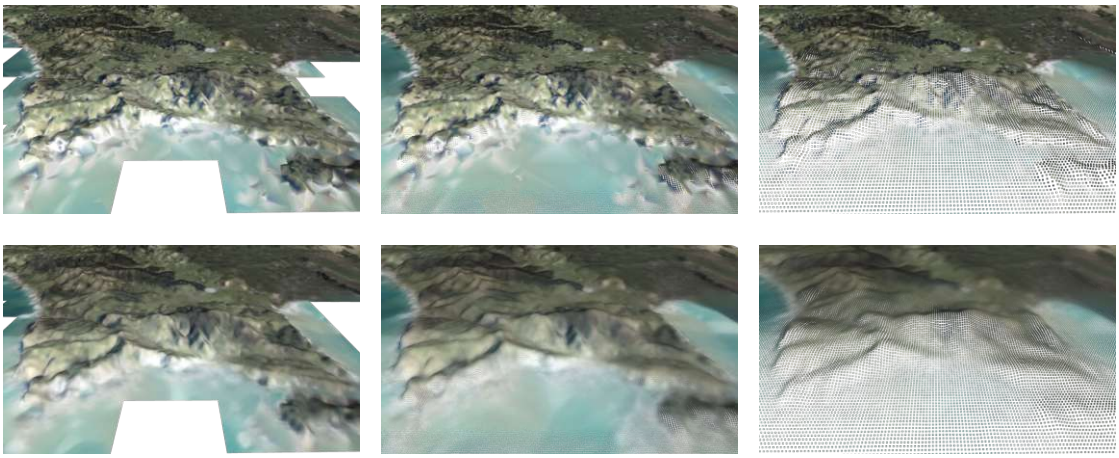


Figure 5.11: Reconstruction of Gisborne_A with different resolutions. Upper: linear interpolation. Lower: LidarScout. Left to right: 10 (default), 20, 40 meters per texel.

5.7 Conclusion

LidarScout is the first point cloud viewer that allows exploring terabytes of compressed LIDAR scans within seconds without any pre-processing. We present fast 2.5D surface reconstruction from sparse, local subsamples with minimal overhead. Our neural network outperforms the current industry standard of triangulation and linear interpolation. In the future, the local chunk points or heightmaps could be streamed from a server for only the required regions, which would drastically reduce data storage on the user side and data transfer costs for everyone.

This work gives the final puzzle piece of this thesis for the research question, 'How can deep learning improve surface reconstruction?'. A more complex model is not necessarily better than a simple one, as we have seen with DCTNet [ZZX+22] and LidarScout's CNN. While global priors might be necessary for many tasks, extracting them from the model can enable great speed-ups, allowing for the reconstruction of huge surfaces. For the sake of speed, efficient inputs and dimensionality reduction are essential.

Our approach for LidarScout is potentially applicable to any data that allows at least partial random access with some learnable patterns, such as huge photographs, volumetric data from CT or MRT scans, weather forecasts, and astronomical simulations. The neural network can be adapted for annotation, segmentation, and classification tasks. With the latter, for example, it could take the number of returns and other extra point properties to detect vegetation. Extending LidarScout to 3D for large urban and indoor scans should be possible by combining our persistent heightmaps with screen-space approaches like ADOP [RFS22] and TRIPS [FRFS24], or by switching from heightmaps to a cubemap-based structure.

5.8 Acknowledgements

We thank the following data set providers: *Bunds at el.* and *Open Topography* for the Bund_Bora [BDG+19] and ID15_Bunds [BDG+20] data sets; *PG&E* and *Open Topography* for CA13 [Pac13]; The *Ministry of Business, Innovation and Employment* and *Toitū Te Whenua Land Information New Zealand* and *Open Topography* for Gisborne [MoBE24]; The *São Paulo City Hall (PMSP)* and *Open Topography* for São Paulo [(PM17]; The *Bundesamt für Landestopografie swisstopo* for swissSURFACE3D [Swi20].

CHAPTER 6

Conclusion

In this thesis, we presented three methods for surface reconstruction from point clouds. Two were focused on high-quality reconstruction of solid objects, independent of object classes. The third one quickly estimates heightmaps for aerial LIDAR scans. Exploring the advantages and disadvantages of global and local priors, as well as finding a good trade-off for the specific tasks, spans across all three parts.

Points2Surf explored the importance of combining global and local priors in the form of two different subsets of the same point cloud. The model we trained this way generalizes well from CAD objects to all kinds of solids.

PPSurf follows a similar approach as Points2Surf, but using point convolutions and attention to fix its shortcomings. Both speed and quality are improved, and most importantly, the topological noise is removed.

LidarScout avoids the costly global priors and reduces the problem space to 2.5D, which allows for very short inference times. This is embedded in a framework for near-instant exploration of huge point clouds.

The central research question of this thesis was: **How can deep learning improve surface reconstruction?** Throughout this work, we have uncovered several key insights towards an answer. The two essential aspects of a good surface-reconstruction method are quality and efficiency. As described above, Points2Surf and PPSurf improve mostly the quality, while LidarScout focuses on efficiency. In other words, if quality is paramount and inputs are messy, a strong global encoding combined with local information is essential. On the other hand, restricting the model to local inputs increases the inference speed.

How can we achieve accurate surface reconstruction that works with all kinds of possibly imperfect point clouds and generalizes well to unseen data? This was the more detailed research question of this thesis.

First, we demonstrated that estimating signed distances from both local and global information leads to more accurate and consistent surface reconstructions. Local subsamples capture fine details, while global context ensures overall coherence. Second, we found that the model's accuracy is crucial. Hierarchical structures and information forwarding can only improve surfaces if the underlying predictions are reliable. Techniques like test-time augmentation (as in POCO [BM22]) are effective at reducing noise from random subsampling, which is necessary for global information. Modern learning methods, such as point convolutions and attention mechanisms, further enhance reconstruction quality.

However, increased model complexity does not always yield better results, as seen with the DCTNet experiment in Section 5.2.2. Simple models with efficient inputs (local-only, grid-based), and outputs of minimal dimensionality, can achieve significant speed-ups with high quality, as seen in LidarScout. While global priors are beneficial for some tasks, extracting them from the model can enable the quick reconstruction of much larger surfaces.

6.1 Lessons Learned

The following general tips are based on our hard-earned lessons from working with the opaque system called deep learning. We hope to help avoid time-wasting mistakes.

1. **Iterate quickly.** Points2Surf had training times of up to 5 days, PPSurf 5 hours, and LidarScout 30 minutes. Only the LidarScout was fast enough to have the bottleneck not on the training side, but in the implementation part. The main reason why the PPSurf paper took that long was the training times. If a change results in decreased quality, the loss curves usually improve less quickly, and you can stop the training. However, be careful with the smoothing of Tensorboard, which distorts recent values with its backward-looking average. If possible, switching from 3D to 2D also helps a lot. Be wary of the curse of dimensionality when switching back to higher dimensions. You will need more samples in the parameter space and possibly a different distribution. Another quick way to check for fundamental problems is to let the model overfit on a single training sample. If this fails, the full dataset will fail as well.

2. **Try everything.** Even the best people in the field can usually not predict how the model reacts to changes. Tiny mistakes can break the whole system in unexpected ways. Therefore, you should iterate quickly with small changes. Do not make assumptions, just try it.

3. **Verify your data.** Visualize the data that the model actually sees. This should include all processing steps that can introduce errors, like the data loader and data augmentation. A simple error there can easily prevent the model from converging. Test-driven development might be overkill, but unit tests for critical parts are highly recommended.

4. **Normalize your data.** A broken normalization can easily prevent the model from learning. As an example, normalizing the CAD objects - some tiny, others huge - to a common range is absolutely necessary for POINTS2SURF and PPSURF. The range $[-1, 1]$ is optimal for floating-point accuracy. For LIDARSCOUT, we need to be very careful with normalization. We tried normalizing all heights independently to a common range. This made originally flat regions look like noisy mountains, which hurt the overall quality. Similarly, preserving a common point (e.g, sea level) in the range might preserve important correlations but may also reduce generalization.

5. **Experiment with model structures.** Sticking too long with the PointNet was another reason why PPSURF took so long. After switching to a POCO, we were able to train the model in a few hours and most reconstruction noise was gone. Similarly, we started with a fairly complicated model for LIDARSCOUT and only later tried a simple CNN that was not just faster but also better.

6. **Models learn only correlations.** Intuition about the data is not reliable. Even trained humans are notoriously bad at estimating probabilities, so test your assumptions, always. Still, the human visual system is very good at seeing patterns. As a rule of thumb, if you cannot see patterns in (probably up to 4D) data, the model is unlikely to see any either. Still, do not pre-process the training samples to eye-pleasing features. The model usually finds its own features, which are harder to understand but more efficient. As an example, the first LIDARSCOUT model used frequency-based inputs, but a simple image-based input proved better.

7. **No premature optimization.** Do not spend time on speeding up code that might be thrown out again, unless it creates a bottleneck. Low-level optimizations for the network, like mixed-precision, different activation functions, and similar loss functions may improve the quality by a few percent, but they will not make the difference between not working and working. Such optimizations are also hard to publish since they are often considered "just engineering".

6.2 Outlook

While surface reconstruction with deep learning was in its infancy when this thesis started, we now have a solution for most tasks. Be it single solids, huge landscapes, high quality, top speed, complex topology, sharp edges, strong noise, without normals, with camera positions, a specific type of object - there is a solution for all these. While there is still lots of untapped potential in general surface reconstruction, future work will likely explore many currently unsatisfied niches.

The recently introduced Neural Radiance Fields [MST+21] and 3D Gaussian Splatting [KKLD23] are novel kinds of scene representation, optimized from multi-view inputs. While some variants for surface reconstruction already exist, the surfaces are still noisy, and processing times are beyond 30 minutes. In any case, there are still many ideas to be explored.

Aside from these general approaches, there are more specific options to improve reconstruction methods like Points2Surf and PPSurf. Some of the failed attempts are worth revisiting, especially the resampling on intermediate reconstructions and hierarchical feature-forwarding (see Section 4.2.1).

One problem still open after PPSurf is the bias to noisy point clouds. PPSurf oversmooths sharp edges even when they are densely sampled without noise. This model cannot be an expert for all amounts of noise. Increasing the dataset size might help, but it will not eliminate the issue altogether. An iterative process similar to NeuralPull [BZYSM21] should help, especially when accompanied by a denoising pre-processing step. We could combine this with resampling on the intermediate reconstruction to make it more robust against varying sampling density. It could also improve the inpainting of missing areas without relying on a heavy generative model. Adding a hierarchical structure on top would allow for reconstructing very thin features that would otherwise be lost to aliasing.

Another persistent limitation of Points2Surf and PPSurf is the focus on solid objects. This means that open surfaces like landscapes are only very badly reconstructed, usually as thin-sheet objects. Similarly, monuments, buildings, and other objects standing on the ground are missing samples on their bottom, which confuses the model. Adding such examples to the training data will likely introduce another bias, reducing the quality of solid objects. Adding predicted or sensed normals to the inputs and switching to Unsigned Distances Fields (UDFs) could help with this issue, but would likely add new problems like the less robust surface extraction from UDFs.

LidarScout opened the door to the efficient exploration of large landscape scans. The same approach, with adaptations, should work for full 3D scans like cities and caves, too. Working with volumetric data would enable a quick preview of high-precision CT scans and large-scale astronomical simulations.

Currently, the performance bottleneck is in the triangulation, which will be even more severe in volumetric data. While a more efficient implementation might improve the issue, a better solution is likely to change the inputs a bit. A linear interpolation in the k nearest neighbors with barycentric coordinates could work efficiently and would only require the same kD-tree we already use for the nearest-neighbor interpolation.

The limiting factor in regards to quality is currently the bias to a specific point density in the patch. LidarScout oversmooths on dense inputs, which wastes available information. A hierarchical reconstruction of heightmaps in a quadtree would surely help, especially when adapting the patch size to the point density and using different models for the hierarchy levels.

Adapting LidarScout to full 3D scans will be a non-trivial task since heightmaps will not work anymore. Subdividing the domain into cubes as in an octree and then predicting heightmaps for the cube faces should be fast enough, but it will introduce new difficulties like achieving consistent edges between the faces. Another idea, which would also work with volumetric data, is to combine LidarScout with screen-space hole-filling approaches like ADOP [RFS22] and TRIPS [FRFS24].

Exploring smart surface reconstruction for this thesis was just the beginning. We are looking forward to the journey ahead.

Overview of Generative AI Tools Used

These AI tools were used for drafts, re-formulation, and spell checking: Chat-GPT, GitHub Copilot, and Grammarly. Img.Upscaler was used for upscaling old images. The German version of the abstract was translated using GitHub Copilot with GPT-4.1 and then manually improved.

List of Figures

List of Tables

List of Algorithms

Bibliography

[ACSTD07] Pierre Alliez, David Cohen-Steiner, Yiying Tong, and Mathieu Desbrun. Voronoi-based variational reconstruction of unoriented point sets. In *Symposium on Geometry processing*, volume 7, pages 39–48, 2007.

[AL20] Matan Atzmon and Yaron Lipman. Sal: Sign agnostic learning of shapes from raw data. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[AL21] Matan Atzmon and Yaron Lipman. SALD: sign agnostic learning with derivatives. In *International Conference on Learning Representations, ICLR*, 2021.

[ASK+20] Kara-Ali Aliev, Artem Sevastopolsky, Maria Kolos, Dmitry Ulyanov, and Victor Lempitsky. Neural point-based graphics. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXII 16*, pages 696–712. Springer, 2020.

[BDG+19] M.P. Bunds, C.B. DuRoss, R.D. Gold, N.G. Reitman, N.A. Toke, R.W. Briggs, S.F. Personius, K. Johnson, L. Lajoie, B. Ungerman, E. Matheson, J. Andreini, and K. Larsen. Lost river fault zone near borah peak, idaho, 2019. Distributed by OpenTopography, Accessed 2025-01-17.

[BDG+20] M.P. Bunds, C.B. DuRoss, R.D. Gold, N.G. Reitman, N.A. Toke, R.W. Briggs, B. Ungerman, , and E. Matheson. Lost river fault at doublespring pass rd, idaho 2015, 2020. Distributed by OpenTopography, Accessed: 2025-01-17.

[BDSF22] Pascal Bormann, Tobias Dorra, Bastian Stahl, and Dieter W. Fellner. Real-time Indexing of Point Cloud Data During LiDAR Capture. In Peter Vangorp and Martin J. Turner, editors, *Computer Graphics and Visual Computing (CGVC)*. The Eurographics Association, 2022.

[BGKS20] Abhishek Badki, Orazio Gallo, Jan Kautz, and P. Sen. Meshlet priors for 3d mesh reconstruction. *ArXiv*, 2020.

[BHZK05] Mario Botsch, Alexander Hornung, Matthias Zwicker, and Leif Kobbelt. High-quality surface splatting on today's gpus. In *Proceedings Eurographics/IEEE VGTC Symposium Point-Based Graphics, 2005.*, pages 17–141. IEEE, 2005.

[BM22] Alexandre Boulch and Renaud Marlet. Poco: Point convolution for surface reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6302–6314, June 2022.

[BMR+99] Fausto Bernardini, Joshua Mittleman, Holly Rushmeier, Cláudio Silva, and Gabriel Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE transactions on visualization and computer graphics*, 5(4):349–359, 1999.

[BPM20] Alexandre Boulch, Gilles Puy, and Renaud Marlet. FKAConv: Feature-Kernel Alignment for Point Cloud Convolution. In *15th Asian Conference on Computer Vision (ACCV 2020)*, 2020.

[BTBW77] 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 *Proceedings of the 5th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'77, pages 659–663, San Francisco, CA, USA, 1977. Morgan Kaufmann Publishers Inc.

[BTS+17] Matthew Berger, Andrea Tagliasacchi, Lee M Seversky, Pierre Alliez, Gael Guennebaud, Joshua A Levine, Andrei Sharf, and Claudio T Silva. A survey of surface reconstruction from point clouds. In *Computer Graphics Forum*, volume 36, pages 301–329. Wiley Online Library, 2017.

[BYSZ22] Ma Baorui, Liu Yu-Shen, and Han Zhizhong. Reconstructing surfaces for sparse point clouds with on-surface priors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[BZYSM21] Ma Baorui, Han Zhizhong, Liu Yu-Shen, and Zwicker Matthias. Neural-pull: Learning signed distance functions from point clouds by learning to pull space onto surfaces. In *International Conference on Machine Learning (ICML)*, 2021.

[CAPM20] Julian Chibane, Thiemo Alldieck, and Gerard Pons-Moll. Implicit functions in feature space for 3d shape reconstruction and completion. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[Cha03] Raphaëlle Chaine. A geometric convection approach of 3-d reconstruction. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 218–229. Eurographics Association, 2003.

[CHL23] Chao Chen, Zhizhong Han, and Yu-Shen Liu. Unsupervised inference of signed distance functions from single sparse point clouds without learning priors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[CMPM20] Julian Chibane, Aymen Mir, and Gerard Pons-Moll. Neural unsigned distance fields for implicit function learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, December 2020.

[Col96] Robert T Collins. A space-sweep approach to true multi-image matching. In *Proceedings CVPR IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 358–363. IEEE, 1996.

[CTFZ22] Zhiqin Chen, Andrea Tagliasacchi, Thomas Funkhouser, and Hao Zhang. Neural dual contouring. *ACM Transactions on Graphics (Special Issue of SIGGRAPH)*, 41(4), 2022.

[CZ19] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *Proc. of CVPR*, 2019.

[CZ21] Zhiqin Chen and Hao Zhang. Neural marching cubes. *ACM Transactions on Graphics (TOG)*, 40(6):1–15, 2021.

[DAC20] Rangel Daroya, Rowel Atienza, and Rhandley Cajote. Rein: Flexible mesh generation from point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 352–353, 2020.

[Daw] Dawson-Haggerty et al. trimesh.

[DDN20] Angela Dai, Christian Diller, and Matthias Nießner. Sg-nn: Sparse generative neural networks for self-supervised scene completion of rgb-d scans. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2020.

[DG03] Tamal K Dey and Samrat Goswami. Tight cocone: a water-tight surface reconstructor. In *Proceedings of the eighth ACM symposium on Solid modeling and applications*, pages 127–134. ACM, 2003.

[DLW+23] Matt Deitke, Ruoshi Liu, Matthew Wallingford, Huong Ngo, Oscar Michel, Aditya Kusupati, Alan Fan, Christian Laforte, Vikram Voleti, Samir Yitzhak Gadre, et al. Objaverse-xl: A universe of 10m+ 3d objects. *Advances in Neural Information Processing Systems*, 36:35799–35813, 2023.

[DMSL11] Julie Digne, Jean-Michel Morel, Charyar-Mehdi Souzani, and Claire Lartigue. Scale space meshing of raw data point sets. In *Computer Graphics Forum*, volume 30, pages 1630–1642. Wiley Online Library, 2011.

[DN19] Angela Dai and Matthias Nießner. Scan2mesh: From unstructured range scans to 3d meshes. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2019.

[Dre07] Ulrich Drepper. What every programmer should know about memory. *Red Hat, Inc*, 11(2007):2007, 2007.

[DVS03] Carsten Dachsbacher, Christian Vogelgsang, and Marc Stamminger. Sequential point trees. *ACM Trans. Graph.*, 22(3):657–662, 2003.

[Ede03] Herbert Edelsbrunner. Surface reconstruction by wrapping finite sets in space. In *Discrete and computational geometry*, pages 379–404. Springer, 2003.

[EFPH+24] Philipp Erler, Lizeth Fuentes-Perez, Pedro Hermosilla, Paul Guerrero, Renato Pajarola, and Michael Wimmer. Ppsurf: Combining patches and point convolutions for detailed surface reconstruction. In *Computer Graphics Forum*, volume 43, page e15000. Wiley Online Library, 2024.

[EGO+20] Philipp Erler, Paul Guerrero, Stefan Ohrhallinger, Niloy J. Mitra, and Michael Wimmer. Points2Surf: Learning implicit surfaces from point clouds. In *European Conference on Computer Vision (ECCV)*, 2020.

[EHWS25] Philipp Erler, Lukas Herzberger, Michael Wimmer, and Markus Schütz. LidarScout: Direct Out-of-Core Rendering of Massive Point Clouds. In Aaron Knoll and Christoph Peters, editors, *High-Performance Graphics - Symposium Papers*. The Eurographics Association, 2025.

[Ent21] Entwine, 2021. `https://entwine.io/`, Accessed 2021.04.13.

[FRFS24] Linus Franke, Darius Rückert, Laura Fink, and Marc Stamminger. Trips: Trilinear point splatting for real-time radiance field rendering. In *Computer Graphics Forum*, page e15012. Wiley Online Library, 2024.

[FSG17] Haoqiang Fan, Hao Su, and Leonidas J Guibas. A point set generation network for 3d object reconstruction from a single image. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 605–613, 2017.

[GFK+18] Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan Russell, and Mathieu Aubry. AtlasNet: A Papier-Mâché Approach to Learning 3D Surface Generation. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[GKLR13] Christian Günther, Thomas Kanzok, Lars Linsen, and Paul Rosenthal. A gpgpu-based pipeline for accelerated rendering of point clouds. *J. WSCG*, 21:153–161, 2013.

[GKOM18] Paul Guerrero, Yanir Kleiman, Maks Ovsjanikov, and Niloy J. Mitra. PCPNet: Learning local shape properties from raw point clouds. *Computer Graphics Forum*, 37(2):75–85, 2018.

[GKUP11] Michael Gschwandtner, Roland Kwitt, Andreas Uhl, and Wolfgang Pree. Blensor: Blender sensor simulation toolbox. In *International Symposium on Visual Computing*, pages 199–208. Springer, 2011.

[GYH+20] Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. Implicit geometric regularization for learning shapes. In *International Conference on Machine Learning (ICML)*. 2020.

[Har13] Mark Harris. How to access global memory efficiently in cuda c/c++ kernels. NVIDIA Technical Blog, 2013. Accessed 2025.01.17.

[HBK23] Saar Huberman, Amit Bracha, and Ron Kimmel. Deep accurate solver for the geodesic problem. In *International Conference on Scale Space and Variational Methods in Computer Vision*, pages 288–300. Springer, 2023.

[HBM23] Marina Hernández-Bautista and Francisco Javier Melero. Deep learning of curvature features for shape completion. *Computers & Graphics*, 115:204–215, 2023.

[HBS25] Josquin Harrison, James Benn, and Maxime Sermesant. Improving neural network surface processing with principal curvatures. *Advances in Neural Information Processing Systems*, 37:122384–122405, 2025.

[HCJ19] Zhiyang Huang, Nathan Carr, and Tao Ju. Variational implicit point set surfaces. *ACM Trans. Graph.*, 38(4), jul 2019.

[HDD+92] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. *Surface reconstruction from unorganized points*, volume 26. ACM, 1992.

[HFF+23] Mathias Harrer, Linus Franke, Laura Fink, Marc Stamminger, and Tim Weyrich. Inovis: Instant novel-view synthesis. In *SIGGRAPH Asia 2023 Conference Papers*, pages 1–12, 2023.

[HFK+24] Florian Hahlbohm, Linus Franke, Moritz Kappel, Susana Castillo, Marc Stamminger, and Marcus Magnor. Inpc: Implicit neural point clouds for radiance field rendering. *arXiv preprint arXiv:2403.16862*, 2024.

[HMGCO20] Rana Hanocka, Gal Metzer, Raja Giryes, and Daniel Cohen-Or. Point2mesh: A self-prior for deformable meshes. *ACM Trans. Graph.*, 39(4), 2020.

[HWW+22] Fei Hou, Chiyu Wang, Wencheng Wang, Hong Qin, Chen Qian, and Ying He. Iterative poisson surface reconstruction (ipsr) for unoriented points. *ACM Trans. Graph.*, 41(4), jul 2022.

[Ise13] Martin Isenburg. Laszip: lossless compression of lidar data. *Photogrammetric engineering and remote sensing*, 79(2):209–217, 2013.

[JSM+20] Chiyu Jiang, Avneesh Sud, Ameesh Makadia, Jingwei Huang, Matthias Nießner, and Thomas Funkhouser. Local implicit grid representations for 3d scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

[JSZ+15] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. *Advances in neural information processing systems*, 28, 2015.

[Kaz05] Michael Kazhdan. Reconstruction of solid models from oriented point sets. In *Proceedings of the third Eurographics symposium on Geometry processing*, page 73. Eurographics Association, 2005.

[KB04] Leif Kobbelt and Mario Botsch. A survey of point-based techniques in computer graphics. *Computers & Graphics*, 28(6):801–814, 2004.

[KBH06] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proc. of the Eurographics symposium on Geometry processing*, 2006.

[KH13] Michael Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG)*, 32(3):29, 2013.

[KKLD23] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), July 2023.

[KMJ+19] Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. Abc: A big cad model dataset for geometric deep learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[KPLD21] Georgios Kopanas, Julien Philip, Thomas Leimkühler, and George Drettakis. Point-based neural rendering with per-view optimization. In *Computer Graphics Forum*, volume 40, pages 29–43. Wiley Online Library, 2021.

[Kru64] Joseph B Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.

[KSH17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[LC87] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *ACM siggraph computer graphics*, volume 21, pages 163–169. ACM, 1987.

[LDG18] Yiyi Liao, Simon Donne, and Andreas Geiger. Deep marching cubes: Learning explicit surface representations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2916–2925, 2018.

[LESP21] Stefan Lionar, Daniil Emtsev, Dusan Svilarkovic, and Songyou Peng. Dynamic plane convolutional occupancy networks. In *Winter Conference on Applications of Computer Vision (WACV)*, 2021.

[LLZM10] Guo Li, Ligang Liu, Hanlin Zheng, and Niloy J Mitra. Analysis, reconstruction and manipulation using arterial snakes. In *ACM SIGGRAPH Asia 2010 Papers*, SIGGRAPH ASIA '10, New York, NY, USA, 2010. Association for Computing Machinery.

[LS81] Peter Lancaster and Kes Salkauskas. Surfaces generated by moving least squares methods. *Mathematics of computation*, 37(155):141–158, 1981.

[LSJ+17] Lubor Ladicky, Olivier Saurer, SoHyeon Jeong, Fabio Maninchedda, and Marc Pollefeys. From point clouds to mesh using regression. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3893–3902, 2017.

[LW85] Marc Levoy and Turner Whitted. The use of points as a display primitive, 1985.

[LXSW22] Siyou Lin, Dong Xiao, Zuoqiang Shi, and Bin Wang. Surface reconstruction from point clouds without normals by parametrizing the gauss formula. *ACM Trans. Graph.*, 42(2), oct 2022.

[MLH23] Baorui Ma, Yu-Shen Liu, and Zhizhong Han. Learning signed distance functions from noisy 3d point clouds via noise to noise mapping. In *International Conference on Machine Learning (ICML)*, 2023.

[MoBE24] Innovation Ministry of Business and Toitū Te Whenua Land Information New Zealand (LINZ) Employment. Gisborne, new zealand 2023, 2024. Released under Creative Commons CC BY 4.0 by NIWA, Collected by Landpro, distributed by OpenTopography and LINZ, Accessed: 2025-01-17.

[MON+19] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proc. of CVPR*, 2019.

[MPS08] Josiah Manson, Guergana Petrova, and Scott Schaefer. Streaming surface reconstruction using wavelets. In *Computer Graphics Forum*, volume 27, pages 1411–1420. Wiley Online Library, 2008.

[MRC+22] Armin Masoumian, Hatem A Rashwan, Julián Cristiano, M Salman Asif, and Domenec Puig. Monocular depth estimation using deep learning: A review. *Sensors*, 22(14):5353, 2022.

[MRVv+15] O Martinez-Rubi, S. Verhoeven, M van Meersbergen, M Schutz, PJM van Oosterom, R Goncalves, and TPM Tijssen. Taming the beast: Free and open-source massive point cloud web visualization. In A MacDonald, editor, *Capturing reality: The 3rd, laser scanning and LIDAR technologies forum*, pages 23–25. s.n., 2015. geen ISBN; Capturing Reality 2015, Salzburg, Austria ; Conference date: 23-11-2015 Through 25-11-2015.

[MST+21] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[MZLH23] Baorui Ma, Junsheng Zhou, Yu-Shen Liu, and Zhizhong Han. Towards better gradient consistency for neural signed distance functions via level set alignment. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[Ned23] Actueel Hoogtebestand Nederland. Eerste deel van ahn 5 is beschikbaar! `https://www.ahn.nl/eerste-deel-van-ahn-5-is-beschikbaar`, 2023. [Accessed 17-01-2025].

[NKH+21] Phong Nguyen, Animesh Karnewar, Lam Huynh, Esa Rahtu, Jiri Matas, and Janne Heikkila. Rgbd-net: Predicting color and depth images for novel views synthesis. In *2021 International Conference on 3D Vision (3DV)*, pages 1095–1105. IEEE, 2021.

[NOS09] Yukie Nagai, Yutaka Ohtake, and Hiromasa Suzuki. Smoothing of partition of unity implicit surfaces for noise robust surface reconstruction. In *Computer Graphics Forum*, volume 28, pages 1339–1348. Wiley Online Library, 2009.

[OBS03] Yutaka Ohtake, Alexander Belyaev, and Hans-Peter Seidel. A multi-scale approach to 3d scattered data interpolation with compactly supported basis functions. In *2003 Shape Modeling International.*, pages 153–161. IEEE, 2003.

[OBS05] Yutaka Ohtake, Alexander Belyaev, and Hans-Peter Seidel. 3d scattered data interpolation and approximation with multilevel compactly supported rbfs. *Graphical Models*, 67(3):150–165, 2005.

[OMW13] Stefan Ohrhallinger, Sudhir Mudur, and Michael Wimmer. Minimizing edge length to connect sparsely sampled unstructured point sets. *Computers & Graphics*, 37(6):645–658, 2013.

[Pac13] Pacific Gas & Electric Company. Pg&e diablo canyon power plant (dcpp): San simeon and cambria faults, ca, airborne lidar survey, 2013. Distributed by OpenTopography.

[PFS+19] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. *arXiv preprint arXiv:1901.05103*, 2019.

[PJL+21] Songyou Peng, Chiyu "Max" Jiang, Yiyi Liao, Michael Niemeyer, Marc Pollefeys, and Andreas Geiger. Shape as points: A differentiable poisson solver. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[(PM17] São Paulo City Hall (PMSP). Sao paulo, brazil lidar survey 2017, 2017. Distributed by OpenTopography and LINZ, Accessed: 2025-06-07.

[PNM+20] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In *European Conference on Computer Vision (ECCV)*, 2020.

[POEM24] Amal Dev Parakkat, Stefan Ohrhallinger, Elmar Eisemann, and Pooran Memari. Ballmerge: High-quality fast surface reconstruction via voronoi balls. In *Computer Graphics Forum*, page e15019. Wiley Online Library, 2024.

[PZVBG00] Hanspeter Pfister, Matthias Zwicker, Jeroen Van Baar, and Markus Gross. Surfels: Surface elements as rendering primitives. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 335–342, 2000.

[QSMG17] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[RALB22] Ruslan Rakhimov, Andrei-Timotei Ardelean, Victor Lempitsky, and Evgeny Burnaev. Npbg++: Accelerating neural point-based graphics. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15969–15979, 2022.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international*

conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18, pages 234–241. Springer, 2015.

[RFS22] Darius Rückert, Linus Franke, and Marc Stamminger. Adop: Approximate differentiable one-pixel point rendering. *ACM Transactions on Graphics (ToG)*, 41(4):1–14, 2022.

[rG15] rapidlasso GmbH. Generating Spike-Free Digital Surface Models from LiDAR - rapidlasso GmbH — rapidlasso.de. `https://rapidlasso.de/generating-spike-free-digital-surface-models-from-lidar/`, 2015. [Accessed 09-01-2025].

[RHW86] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[RL00] Szymon Rusinkiewicz and Marc Levoy. Qsplat: A multiresolution point rendering system for large meshes. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, page 343–352, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[RLBG+19] Marie-Julie Rakotosaona, Vittorio La Barbera, Paul Guerrero, Niloy J Mitra, and Maks Ovsjanikov. Pointcleannet: Learning to denoise and remove outliers from dense point clouds. *Computer Graphics Forum*, 2019.

[RSL+24] Uchitha Rajapaksha, Ferdous Sohel, Hamid Laga, Dean Diepeveen, and Mohammed Bennamoun. Deep learning-based depth estimation methods from monocular image and videos: A comprehensive survey. *ACM computing surveys*, 56(12):1–51, 2024.

[SA11] SciPy Authors. CloughTocher2DInterpolator x2014; SciPy v1.15.1 Manual — docs.scipy.org. `https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.CloughTocher2DInterpolator.html`, 2011. [Accessed 17-01-2025].

[SHW24] Markus Schütz, Lukas Herzberger, and Michael Wimmer. Simlod: Simultaneous lod generation and rendering for point clouds. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 7(1):1–20, 2024.

[SKW22] Markus Schütz, Bernhard Kerbl, and Michael Wimmer. Software rasterization of 2 billion points in real time. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 5(3):1–17, July 2022.

[SLS+06] Andrei Sharf, Thomas Lewiner, Ariel Shamir, Leif Kobbelt, and Daniel Cohen-Or. Competing fronts for coarse–to–fine surface reconstruction. In *Computer Graphics Forum*, volume 25, pages 389–398. Wiley Online Library, 2006.

[SMB+20] Vincent Sitzmann, Julien N.P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[SMOW20] Markus Schütz, Gottfried Mandlburger, Johannes Otepka, and Michael Wimmer. Progressive real-time rendering of one billion points without hierarchical acceleration structures. *Computer Graphics Forum*, 39(2):51–64, 2020.

[SOW20] Markus Schütz, Stefan Ohrhallinger, and Michael Wimmer. Fast out-of-core octree generation for massive point clouds. *Computer Graphics Forum*, 39(7):1–13, November 2020.

[STM+21] Yawar Siddiqui, Justus Thies, Fangchang Ma, Qi Shan, Matthias Nießner, and Angela Dai. Retrievalfuse: Neural 3d scene reconstruction with a database. In *International Conference on Computer Vision (ICCV)*, 2021.

[Sur18] USGS (U.S. Geological Survey). Changes in usgs lidar data distribution announced. `https://www.usgs.gov/news/technical-announcement/3d-elevation-program-distributing-lidar-data-laz-format`, 2018. [Accessed 17-01-2025].

[SW11] Claus Scheiblauer and Michael Wimmer. Out-of-core selection and editing of huge point clouds. *Computers & Graphics*, 35(2):342–351, 2011.

[Swi20] Swisstopo. swisssurface3d raster: Das hoch aufgelöste oberflächenmodell der schweiz. `https://backend.swisstopo.admin.ch/fileservice/sdweb-docs-prod-swisstopoch-files/files/2023/11/14/24d12399-72b8-4000-8544-235023c4369f.pdf`, 2020. [Accessed 09-01-2025].

[sxy] sxyu. GitHub - sxyu/sdf: Parallelized triangle mesh –> continuous signed distance field on CPU — github.com. `https://github.com/sxyu/sdf`. [Accessed 08-05-2025].

[TDB17] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2088–2096, 2017.

[TFT+20] Ayush Tewari, Ohad Fried, Justus Thies, Vincent Sitzmann, Stephen Lombardi, Kalyan Sunkavalli, Ricardo Martin-Brualla, Tomas Simon, Jason Saragih, Matthias Nießner, et al. State of the art on neural rendering. In *Computer Graphics Forum*, volume 39, pages 701–727. Wiley Online Library, 2020.

[TH15] Abdel Aziz Taha and Allan Hanbury. Metrics for evaluating 3d medical image segmentation: analysis, selection, and tool. *BMC medical imaging*, 15(1):1–28, 2015.

[TLX+21] Jiapeng Tang, Jiabao Lei, Dan Xu, Feiying Ma, Kui Jia, and Lei Zhang. Sa-convonet: Sign-agnostic optimization of convolutional occupancy networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021.

[TLY+21] Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Neural geometric level of detail: Real-time rendering with implicit 3d shapes. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11358–11367, 2021.

[UK21] Benjamin Ummenhofer and Vladlen Koltun. Adaptive surface reconstruction with multiscale convolutional kernels. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[WBB+08] Michael Wand, Alexander Berner, Martin Bokeloh, Philipp Jenke, Arno Fleck, Mark Hoffmann, Benjamin Maier, Dirk Staneker, Andreas Schilling, and Hans-Peter Seidel. Processing and interactive editing of huge point clouds from 3d scanners. *Computers & Graphics*, 32(2):204 – 220, 2008.

[WKZ+16] K. Wolff, C. Kim, H. Zimmer, C. Schroers, M. Botsch, O. Sorkine-Hornung, and A. Sorkine-Hornung. Point cloud noise and outlier removal for image-based 3d reconstruction. In *2016 Fourth International Conference on 3D Vision (3DV)*, pages 118–127, Oct 2016.

[WLT22] Peng-Shuai Wang, Yang Liu, and Xin Tong. Dual octree graph networks for learning adaptive volumetric shape representations. *ACM Transactions on Graphics*, 2022.

[WS06] Michael Wimmer and Claus Scheiblauer. Instant Points: Fast Rendering of Unprocessed Point Clouds. In Mario Botsch, Baoquan Chen, Mark Pauly, and Matthias Zwicker, editors, *Symposium on Point-Based Graphics*. The Eurographics Association, 2006.

[WSLT18] Peng-Shuai Wang, Chun-Yu Sun, Yang Liu, and Xin Tong. Adaptive o-cnn: A patch-based deep representation of 3d shapes. *ACM Transactions on Graphics (TOG)*, 37(6):1–11, 2018.

[WSS+19] Francis Williams, Teseo Schneider, Claudio T Silva, Denis Zorin, Joan Bruna, and Daniele Panozzo. Deep geometric prior for surface reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10130–10139, 2019.

[WWG+21] Qianqian Wang, Zhicheng Wang, Kyle Genova, Pratul P Srinivasan, Howard Zhou, Jonathan T Barron, Ricardo Martin-Brualla, Noah Snavely, and Thomas Funkhouser. Ibrnet: Learning multi-view image-based rendering. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4690–4699, 2021.

[WWW+23] Zixiong Wang, Pengfei Wang, Pengshuai Wang, Qiujie Dong, Junjie Gao, Shuangmin Chen, Shiqing Xin, Changhe Tu, and Wenping Wang. Neural-imls: Self-supervised implicit moving least-squares network for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–16, 2023.

[XSL+23] Dong Xiao, Zuoqiang Shi, Siyu Li, Bailin Deng, and Bin Wang. Point normal orientation and surface reconstruction by incorporating isovalue constraints to poisson equation. *Computer Aided Geometric Design*, 103:102195, 2023.

[YGL+23] Meng You, Mantang Guo, Xianqiang Lyu, Hui Liu, and Junhui Hou. Learning a locally unified 3d point cloud for view synthesis. *IEEE Transactions on Image Processing*, 2023.

[YWOSH20] Wang Yifan, Shihao Wu, Cengiz Oztireli, and Olga Sorkine-Hornung. Iso-points: Optimizing neural implicit surfaces with hybrid representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[ZHA+23] Qijian Zhang, Junhui Hou, Yohanes Yudhi Adikusuma, Wenping Wang, and Ying He. Neurogf: A neural representation for fast geodesic distance and path queries. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[ZJ16] Qingnan Zhou and Alec Jacobson. Thingi10k: A dataset of 10,000 3d-printing models. *arXiv preprint arXiv:1605.04797*, 2016.

[ZML+22] Junsheng Zhou, Baorui Ma, Yu-Shen Liu, Yi Fang, and Zhizhong Han. Learning consistency-aware unsigned distance functions progressively from raw point clouds. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[ZPVBG02] Matthias Zwicker, Hanspeter Pfister, Jeroen Van Baar, and Markus Gross. Ewa splatting. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):223–238, 2002.

[ZTNW23] Biao Zhang, Jiapeng Tang, Matthias Nießner, and Peter Wonka. 3dshape2vecset: A 3d shape representation for neural fields and generative diffusion models. *ACM Trans. Graph.*, 42(4), jul 2023.

[ZZX+22] Zixiang Zhao, Jiangshe Zhang, Shuang Xu, Zudi Lin, and Hanspeter Pfister. Discrete cosine transform network for guided depth map super-resolution. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5697–5707, 2022.