EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät

Computergrafik

Masterarbeit

points2mesh: Learning surface reconstruction from point cloud data

Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Wilhelm-Schickard-Institut für Informatik
Computergrafik
Denis Jan Heid, denis.heid@student.uni-tuebingen.de, 2019

Bearbeitungszeitraum: Januar 2019-Juli 2019

Betreuer/Gutachter: Prof. Dr. Hendrik Lensch, Universität Tübingen
Zweitgutachter: Prof. Dr. Andreas Schilling, Universität Tübingen

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind. Diese Masterarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Denis Heid (Matrikelnummer 3827662), July 11, 2019

Abstract

points2mesh, the happening

Zusammenfassung

braucht man doch auch in deutsch?

Acknowledgments

Thänks fabi, bist der betreuer boy. Lensch ist nice von dir dass ich hier das machen kann. Schilling fuer zweitkorrektur. Thänks rike fuer moralische unterstuetzung

Contents

1. Introduction

For humans to interact with the world depends on many variables. An important aspect is to recognize, analyze, and estimate the position, as well as the shape of objects positioned in the world around. For many years, to this day, artificial agents have been developed and deployed in the real world[Bro86, KMS19] to imitate how humans interact with it. Similarly, a robot has to understand its environment by analyzing and interpreting its surroundings. Understanding exactly this requires first and foremost prior comprehension of the shape of nearby objects. Similar to general robotic applications, the by now more feasible solution for autonomous driving needs analogous information of its obstacles to prevent collisions and thus allow for safe guidance through traffic. Typically, objects are recorded in a way, which results in unstructured, three-dimensional point cloud data, for example, with the help of range scanners or structured light.

Though obtaining information about samples of an object already provides a way to process a three-dimensional object; still, the preferred way to operate is in the form of surface information like tri-or quad-meshes. Transforming such point cloud data into meshes is a prevalent problem in the context of computer vision. Thus, numerous methods have been developed, aiming to solve that problem[BMR+99, JTPSH15]. However, finding use in applications like autonomous driving, reconstructing watertight meshes is a crucial step. Calculating robust collisions, analyzing an object's surface or poviding a 3D printable object requires such a complete mesh, without holes, or undefined spots on its surface.

With recent advancements in deep learning and the abundance of readily available data, much work has been poured into neural networks which can process the real world robustly. Consequently, solving the same problem of processing unstructured points in three-dimensional space and obtaining polygonal meshes has garnered attention. Though, many methods rely on voxelized data, since neural networks can naturally process structured data better than unstructured spatial data.

In this work, a novel convolutional neural network configuration is proposed which computes watertight tri-meshes from unstructured point cloud data. By deforming an initial base-mesh based on learned tension feature vectors from the data, the network can reliably generate approximations even with low-resolution or noisy data. Additionally, a supervised and an unsupervised version of the training process is proposed and evaluated. By providing multi-class training, both versions of the network learn the general reconstruction of objects, even allowing for out of class

reconstruction. Since this network is the first of its kind, it is essential to assess its capabilities in comparison to the traditional methods. Therefore, part of this work is to identify its strengths and weaknesses and determine where it may find use and how to build upon it.

The main contributions of this work consist of the following:

- First neural network learning from point cloud data and transforming it directly to a mesh.
- Competitive results with multi category learning.
- On par or close to computation time of traditional reconstruction methods.

Chapter revision

First, a short revision of the basic components, techniques and methods are described in chapter 2 to lay the groundwork for this thesis. In chapter 3, numerous traditional, as well as machine learning based methods, are reviewed, which work on the transformation of three-dimensional data to polygonal meshes. After that, the graph based neural network *points2mesh* is explained in detail, how it handles and learns from point cloud data but also how it deforms a base-mesh to approximate the underlying object represented by the data. Then, in chapter 5 based on different configurations for the neural network, a *supervised* as well as *unsupervised* version, an assortment of resulting meshes are presented and scontrasted against each other. Followed by section 6. a numerical evaluation, and discussion are conducted to assess the capabilities of the network and its varied proposed configurations. By defining a comparable metric for evaluation, related methods achieving similar results can be analyzed and thus measured against each other. To conclude, in chapter 7, some results, advantages and disadvantages of *points2mesh* are weighted up, and finally compared to traditional methods as well as a direct competitor which is utilizing deep learning techniques too. The thesis is resolved by taking a look at what has been achieved and possible ways of expanding on the network.

2. Background

Understanding the fundamental technical background is essential to follow this work at every step. Thus, each primary component is described and set in context with the *points2mesh* network.

Unstructured three-dimensional data

Digitized objects can be represented by a collection of three-dimensional vector samples, positioned on their surface. These unstructured data points are disjoint from each other, coexisting without a direct semantic link, defined as $\mathcal{PC}_x : \mathbf{x}_i \in \mathbb{R}^3$ Every point $\mathbf{x}_i$ in $\mathcal{PC}$ may have a related *normal* vector, describing its normal orientation on the sampled surface, defined as $\mathcal{PC}_n : \mathbf{n}_i \in \mathbb{R}^3$

A novel deep learning technique allows to learn features based on such data, similar to how classical two dimension convolutional networks learn on two dimensional data like images. Groh et al. [GRL17] describe a convolutional layer (flex convolution) which allows to convolve on unstructured point cloud data. By determining k nearest neighors for each point x_i in $\mathcal{PC}$ a scaled convolution on that neighborhood can be computed and feature vectors v_i calculated.

points2mesh requires $\mathcal{PC}$ in multiple resolutions, thus has to be subsampled to several sizes. However, point clouds have parts which have a low density even before removing any samples. Thus, a sampling technique has to be chosen, which samples based on that density. Parts with higher density should be sampled less often while parts with lower density should be sampled more often. *Weighted reservoir sampling* solves this problem. Vieira [Vie14] described a simple way of implementing that sampling technique employing the *gumbel-max trick*, which also found use in *points2mesh*.

Learning on graphs

Digitized objects often are represented as a collection of vertices connected by edges. Each vertex has a threee dimensional coordinate and knowledge about connections to specific neighbors. Though, a mesh can also directly represented as in a graph structure $\mathcal{G} = (V, E)$, where each vertex corresponds to a node in the graph, and each edge, also as an edge in the graph. Additionally the nodes in the graph hold the

positional information of the vertices in each node. Kipf et al. [KW16] describe a graph convolutional neural network (*gcn*) operating on such graph structures. A *gcn* learns a function of features on $\mathcal{G}$, producing an output for each node. In the context of *points2mesh*, the features are described by three dimensional coordinates of their position in space. The output corresponds to the 'deformed' positions of the graph.

3. Related Work

Often in computer graphics, it is necessary to process three-dimensional real-world or digital objects for reconstruction, remeshing or analytical applications. There are many ways to acquire data of the surface of such geometry and more so many techniques to transform and augment that data to different representations. This often non-trivial task is crucial to further process the object in question in later stages of their respective pipeline. Over the years many representations of acquisition data formats and transformation methods, as well as target data formats, have accumulated.
In this chapter, various of these techniques and data formats are examined, of which some of them are used in this work as a vehicle for a novel data transformation routine.
Initially, classical approaches are examined in section 3.1 which do not rely on artificial intelligence or machine learning methods. With the recent advances in machine learning, and more so deep learning, many new approaches have been developed and thus considered in this work. Hence, machine learning based methods are reassessed in section 3.2 which rely on distinctive statistical features in their initial data format or the dataset itself, thus allowing for the transformation.

3.1. Traditional approaches

In this chapter, traditional approaches are examined which do not rely on machine learning to transform data of objects from one format to another.

In general, a three-dimensional object may be specified by its representation of the surface. This representation varies from unstructured point cloud data in three-dimensional space to a graph-based representation like triangle-/ or quad-meshes and even poly meshes with more than four neighbors per node. Information on the object's surface may also be described by three-dimensional discrete scalar field (Voxels). Often the information on an object's surface is only partially defined, where some information may be missing. This can be due to acquisition methods, where parts of the object are partially concealed or recorded from insufficient many sides, or that information is not available.

Starting from these representations of an object's surface, there are many methods of transformation to reach another state of representation. Transforming the data not

only makes further processing easier but rather during their process, more data on the surface is computed based on the given input data.

This task is non-trivial since, from one representation of an object's surface, many reconstructions in the form of another representation are possible. The amount of data of the object's surface can never be entirely perfect, as the resolution is chosen arbitrarily, thus leading to many reconstruction methods. In this section, some of the more important works on this subject of data transformation are examined, with an overview of the different input and output data formats applicable.

3.1.1. Reconstruction methods

Reconstructing surface information of objects has been worked on for many years, dating back to the earliest approaches in 1987 [LC87], with the marching cubes algorithm. Most of these earlier concepts rely on local neighborhood reconstruction mechanism. Either by starting with unstructured points in three-dimensional space [BMR+99, ACK01], discrete scalar fields [LC87] or structured light [GRL17]. Later work incorporated global optimization techniques to achieve better results [KBH06]. Though, more recent work relying on local optimization operators again, with faster computing times and being able to handle a much higher capacity of input data [JTPSH15]. Even though using normal orientations of PC data is typical, advanced reconstruction techniques with limited information of the original object are still present [BL18].

3.2. Machine Learning based approaches

With more advanced acquisition methods, higher computing power and high-speed connectivity over the internet, gathering, interchanging and expanding big datasets are getting progressively more prevalent and important. Thus, allowing for statistical and data-driven algorithms to analyze big datasets, computing characteristics and similarities within them.

In this section, specfically machine learning based approaches are inspected which learn features from big datasets and utilize gained knowledge to infer information on an object's surface given varied input data.

First, subsection 3.2.1 compiles an overview of the work on datasets, consisting of polygonal object data, range data, and object annotations to enable learning in three dimensional structures at all. Then, subsection 3.2.2 current and older approaches to processing and learning from three-dimensional data, independently on their input or output data format. Furthermore, subsection 3.2.3 establishes a frame for this work in contrast to the current, as well as older techniques for data transformation from diverse object datasets to a final reconstruction of surface information.

3.2.1. 3D datasets

In recent years, a tremendous amount of work has been made in the field of learning in three-dimensional space. Still, different objectives on learning goals require different datasets to work on. Some of them are more general set of objects, defined by tri- or quad meshes. Many of these datasets have been developed and published in the last 10 years. Exemplary by Shilane et al. [SMKF04], offering 1800 meshes, with a rendered image of the object and object label. Similarly, proposed in a paper by Wu et al.[ZSK+15], ModelNet is a collection of clean CAD models, with complete polygonal mesh, textures, labels out of currently 662 model categories and 127,915 models[1]. Wu et al. also provide a much smaller and well-arranged subset with only 40 or 10 model categories where each object is axis aligned. Furthermore, Chang et al. [CFG+15] provide an even bigger and richer annotated dataset of more than three million polygonal mesh models and more than four thousand categories. While this dataset annotates objects in minuscule detail and semanticism, it also offers a condensed version of the dataset, ShapenetCore, for a better digest of all objects with 51300 models and 55 categories. Meanwhile also offering a more densely annotated subset, ShapeNetSem, but also PartNet[2], a hierarchical part annotation subset of ShapeNetCore. Xiang et al. [XKC+16] also describe a large scale dataset consisting of 100 categories, 90000 images, 200000 objects in the images of which are 44000 modeled, working towards facilitating object recognition and pose estimation of multiple objects in a scene. Lim et al. [LPT13] propose different dataset of polygonal objects with images of each object for the purpose of better pose estimations. Moreover, Zhou et al. [ZJ16] introduce an comprehensive dataset of 10000 3D-printing Models, providing a basis for structural and semantic analysis. For different three-dimensional learning tasks, complete scenery is needed consisting of a collection of depth information, semantic labels, meshes, orientations, top-down 2D views or images, as proposed by [NSF12, SS15, HTT+17, DCS+17a]

3.2.2. Learning in 3D

Learning in three-dimensional space, especially in the context of neural network and deep learning, has recently started to gain traction and produce results. Still, a tremendous amount of work has been achieved. As there are numerous different applications, goals, and intentions for learning in three-dimensional space, many datasets in various data formats to learn on have been established. Most of which originate from range scans, surface samples, or polygonal mesh structures.

Since a collection of unregulated points in space like Point clouds are hard to structure, many opt to lessen the complexity by structuring them in a voxel-based grid, thus reducing the resolution of the problem size. With this volumetric approach,

[1] As of 1.6.2019
[2] As of March, 2019

various objectives have been worked on, and many results produced. Maturana et al. [MS15], as well as Wu et al.[WSK+15] and Sedaghat et al. [SZAB17], show the possibility of learning object recognition based on voxel data. Others use joint or multiple data formats, not only relying on one form of volumetric data. Consequently, learning object representations in lower dimensional vectors, and thus allowing for object discrimination as well [HZ16, QSN+16, BLRW16].

Furthermore, expanding the resolution of objects by considering point cloud data, learning features for object recognition is still possible, as shown by [QSMG16, QYSG17, KL17, LBSC18, LD18]. Additionally, semantic segmentation of such objects and scenes are possible. While not only bound to voxelized data [DN18, DRB+17], point cloud data [QSMG16, QYSG17, DCS+17b, GRL17, LLZ+17, TCA+17], but also explicit surface definitions like polygonal meshes [FFY+18, KHS10, SvKK+11, KAMC17].

3.2.3. Learning 3D reconstruction

Learning 3D reconstruction can be quite a challenging task. Numerous and complex approaches have been tested to infer surface structure from diverse input training data. Many of which rely on single-image or multiple-image to polygonal mesh transformations. While some try to reconstruct similar, already learned objects of specific categories from RGB images, others try to generalize this step for all objects.

Naturally, object-specific categories are more restrictive in inference as well as generalization. Whereas works by Pontes et al. and Kong et al. rely on first finding a specific object, close to the input data. Then they deform this general class object to a target object, expectedly close to the input data [KTEM18, PKS+17]. Nevertheless, work by Wang et al. [WZL+18] demonstrate a general case of reconstruction from RGB Input images. Combining multi-view depth images of objects as a basis for reconstruction is considered as well [TEM18]. While not restricted to reconstructing from images, voxelized three-dimensional data also is utilized [SFH18] with its drawbacks of native lower resolution. No work in the field is present, as far as transforming point cloud data directly to a polygonal mesh structure.

4. Graph-based surface reconstruction network

In this work, a novel approach for an end-to-end application for explicit surface reconstruction from unstructured point cloud data is proposed, based on a graph convolutional network (GCN) by Wang et al. [WZL+18], transforming images to polygonal meshes. It is heavily modified to allow point cloud data as input, though still able to infer polygonal meshes from previously unseen datapoints. This chapter outlines the changes made to the network by Wang et al. , new extensions for learning in three-dimensional space on point cloud data, and its variations to yield the best possible results. The proposed network (*points2mesh*) learns feature vectors from point cloud data with their normal orientation, which are utilized to deform an initial polygonal mesh object, like an ellipsoid, resulting in an approximation of a mesh, based on the input point cloud.

Section 4.1 specifies the network's structure as well as proper objective functions in detail which operate the two-part convolutional network.

Subsequentially, subsection 4.1.7 specifies the dataset used to train the networks.

4.1. Neural network structure

Choosing suitable configurations, proper hyperparameters tunings, or even an appropriate dataset for a neural network is not a clear-cut decision. Thus, several configurations and hyperparameter tunings have been developed. Subsection 4.1.1 first describes the general idea of the neural network, followed by a detailed break down of its structure and configurations in section 4.1.3 and 4.1.4 , Subsequentially, a closer look at utilized objective functions in subsections 4.1.5 will be taken. Finally, every considered configuration of the neural network is outlined in subsection 4.1.6.

4.1.1. General system overview

The general goal of the neural network $\mathcal{N}_{recon}$ is to provide an end-to-end solution for explicit surface reconstruction given a collection of unstructured point cloud data ($\mathcal{PC}$) in three-dimensional space with their normal orientation $\mathbf{n}_{x_i}$.

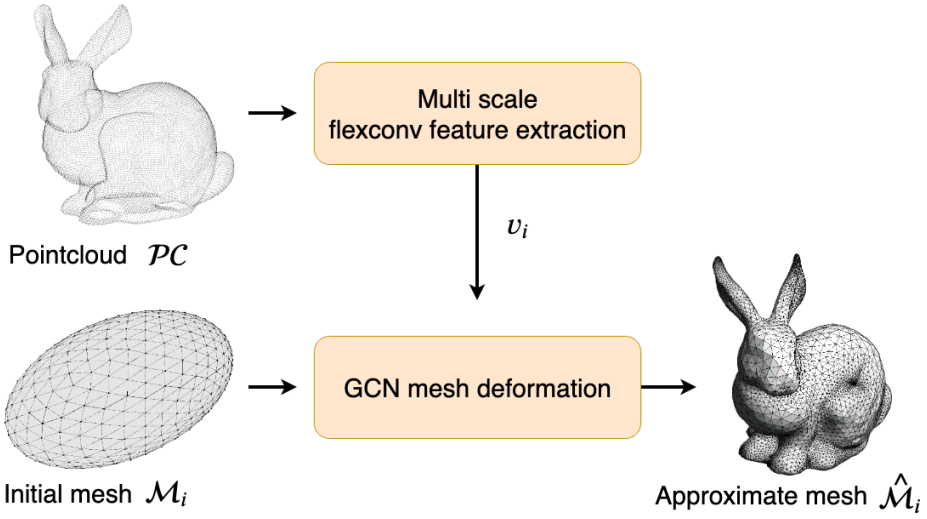


Figure 4.1.: General workflow of *points2mesh* network. $\mathcal{N}_{recon}$ takes a pointcloud $\mathcal{PC}$ and an initial mesh $\mathcal{M}_i$ as input, deforms $\mathcal{M}_i$ based on important features v_i in $\mathcal{PC}$ to compute an approximate mesh $\hat{\mathcal{M}}$

With:

$$\forall \mathbf{x}_i \in \mathcal{PC} : \mathbf{x}_i = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \in [-1,1]^3 \tag{4.1}$$

Overall, the network structure is separated into two distinct parts, each with its specific purpose to fulfill. As seen in Figure 4.1, the neural network $\mathcal{N}_{recon}$ gets two different inputs, whereas each input is processed in a different part of the network. First, the input data $\mathcal{PC}$ is fed into the upper half of the network, called *flexconv feature extraction* $\mathcal{N}_{flex}$ (See Figure 4.1). In a multiscale convolutional operation, a collection of feature vectors v_{x_n} is extracted, which is fed into the lower part of the network.

The second part, as seen in the lower half of figure 4.1, illustrates a *graph convolutional network* $\mathcal{N}_{gcn}$. Given an initial ellipsoid polygonal mesh $\mathcal{M}_e$, $\mathcal{N}_{gcn}$ deforms it, estimating $\hat{\mathcal{M}}$. Where $\hat{\mathcal{M}}$ is the best matching estimate for the ground truth mesh $\mathcal{M}_{gt}$ given $\mathcal{PC}$. The deformation process is assisted by supplying feature vectors v_{x_n} from $\mathcal{N}_{flex}$, also adding more vertices to the initial ellipsoid $\mathcal{M}_e$ in three steps during the deformation process.

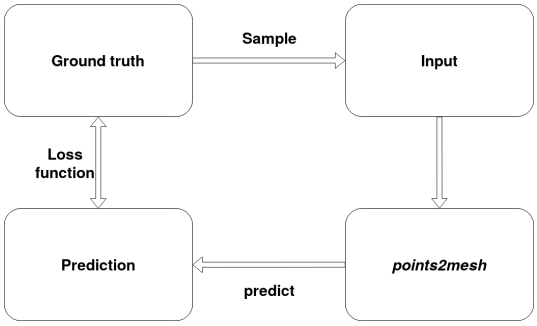


Figure 4.2.: Traditional way to train neural network. Ground truth data and input are separated. Input may be infered from ground truth data.

4.1.2. Network training process

Traditionally, in a supervised training configuration, ground truth data is separated from the input, the input fed to the neural network, and a prediction is calculated based on that data. This is illustrated in figure 4.2. In the case of $\mathcal{N}_{recon}$, the ground truth consists of sampled point cloud data from the original mesh, containing 10000 samples per object. From which a subset is sampled, and used as input for $\mathcal{N}_{recon}$. The predicted mesh is evaluated with a loss function against ground truth data, leading to the network to learn generalizations for reconstruction based on a lower amount of samples.

In contrast, an alternative approach $\mathcal{N}_{opt}$ for training is considered as well, leaning closer to unsupervised training, though not in a traditional sense. More like an optimization process, this second approach disregards the need for ground truth data and operates solely on input data, as seen in figure 4.3. $\mathcal{N}_{opt}$ takes an already sampled point cloud as input, predicts a mesh based on that data; however, evaluates the loss function of the prediction against the same input as already sampled. Though, its focus is less on generalization, more on learning optimization within the same subset of data and thus also considered in this work. $\mathcal{N}_{opt}$ may thus even be trained on natural point cloud data, where no real ground truth is available.

4.1.3. Flexconv feature extraction

In traditional convolutional networks in context of computer graphics, the data at hand is often given as two dimensional images. In the case of three-dimensional processing, structured 3D grids are then a popular tool for natural processing of such. However, since converting data from $\mathcal{PC}$ to a voxelized representation diminished its resolution, it is now kept as its unstructured base form of point cloud data.

As seen in the upper half of figure 4.4, $\mathcal{N}_{flex}$ processes $\mathcal{PC}$ to compute a collection of three feature vectors v_{x_n} on a multiscale measure, working as follows. The network

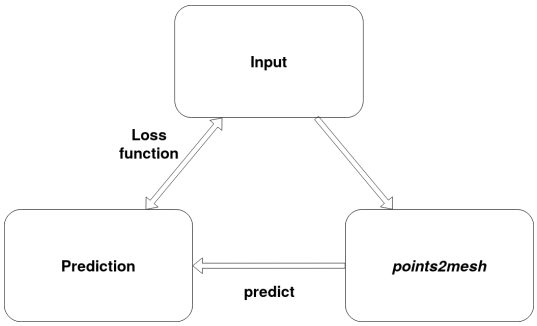


Figure 4.3.: Alternative approach to train $\mathcal{N}_{opt}$. Input and ground truth data are the same in this example.

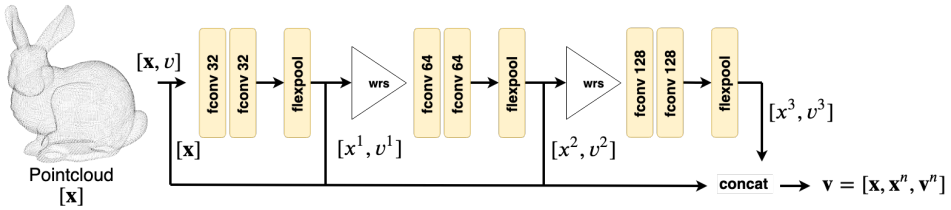


Figure 4.4.: $\mathcal{N}_{flex}$ with point cloud $\mathcal{PC}$ as input, computing features v^j in three blocks of flexconv layers and a trailing flexpool layer After each convolutional block, weighted random sampling reduces the number of samples of $\mathcal{PC}$ by a factor of four. Feature vectors v^j are concatenated with their respective positions x^j in a vector $\mathbf{v}$.

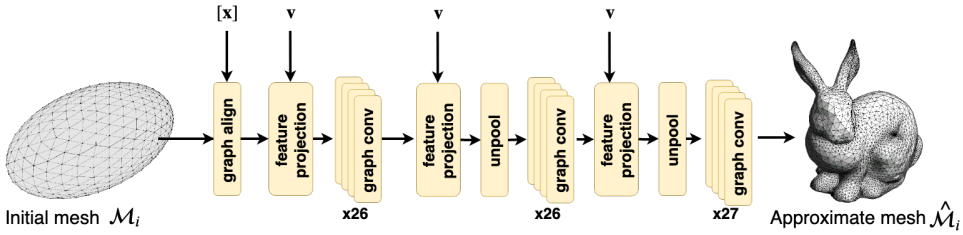


Figure 4.5.: $\mathcal{N}_{gcn}$ with initial mesh $\mathcal{M}_i$ as input. Feature vector $\mathbf{v}$ is projected onto the mesh $\mathcal{M}_i$ and deformed according to it. Then an unpooling layer increases the number of vertices in $\mathcal{M}_i$. Projecting $\mathbf{v}$ onto the graph and deforming it afterward is repeated two more times.

$\mathcal{N}_{flex}$ takes a vector of three-dimensional points $\mathbf{x}_i \in \mathcal{PC}$ of size [3,1024] or [3,8000] (depending on the configuration), assigning each point its own 3D coordinates as his current feature v_i. Then, it is separated into three consecutive parts, where each section $\mathcal{S}$ processes the point cloud vector $\mathbf{x}_i$ and features $\mathbf{v}_i$ with two consecutive flexconv operators and a flexpool operator while expanding the dimension by a factor of 32 in each section. After convolving, the point cloud is sampled down by a factor of four each time by weighted reservoir sampling (wrs) . Following every convolution, the coordinates $\mathbf{x}_i$ and their features $\mathbf{v}_i$ are collected. This compiles the multiscale feature vector v_i^s, which consits of key points in $\mathcal{PC}$ needed for the deformation steps in the second part of the network.

4.1.4. GCN mesh deformation

As for the second part of the network $\mathcal{N}_{recon}$, it consists of a graph convolutional network $\mathcal{N}_{gcn}$, taking a basic polygonal shape $\mathcal{M}_i$ as input, for example an ellipsoid form. The input mesh can directly be transformed into a graph structure, which the *gcn* can process since each vertex in the mesh corresponds to a node in the graph.Its structure, i.e., the number of vertices and edges, has to be known beforehand. Moreover, the same is true for the correspondence of edges in the mesh and edges in the graph. Thus, $\mathcal{M}_i = \mathcal{G}(V,E)$, with V being the vertices in the graph, and E the set of edges, connecting V. Furthermore, similarly to 4.1.3, the three-dimensional coordinates of each vertex in the initial mesh are utilized as feature vectors f_i for $\mathcal{G}(V,E,f_i)$. Since $\mathcal{M}_i$ corresponds to $\mathcal{G}$, in the following sections, the input graph $\mathcal{G}$ of $\mathcal{N}_{gcn}$ is still referenced as input mesh $\mathcal{M}_i$.

$\mathcal{N}_{gcn}$ is separated into three parts, as seen in figure 4.5. The primary process of one such segment consists of convolving on the graph for a set amount of iterations. Then, information from feature vectors $\mathbf{v}_i$ from $\mathcal{N}_{flex}$ to local feature vectors of the graph $f_i = [f_i, \mathbf{v}_i]$ are appended by projecting them from the point cloud data onto the graph. Subsequently, the graph's vertices and edges are increased with a specialized *unpooling operation*. $\mathcal{N}_{gcn}$ repeats this process two more times to compute

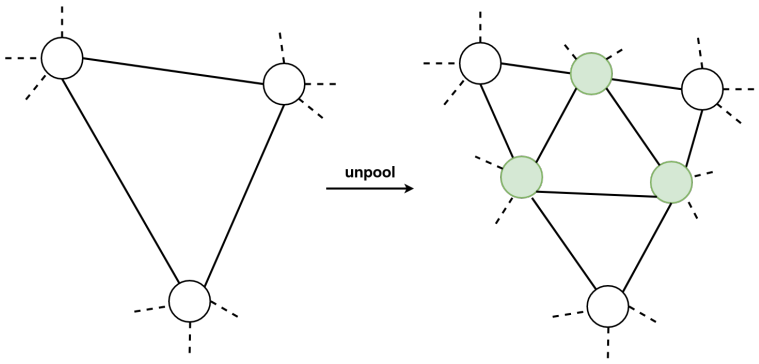


Figure 4.6.: The *unpooling* layer of $\mathcal{N}_{gcn}$ increases the number of vertices by introducing a new node between each neighboring node (white) in $\mathcal{M}_i$. A new node (green) is added precisely in the middle of each old edge. Finally, the new nodes are connected, while also updating the neighborhood of the old nodes.

$\hat{\mathcal{M}}$, approximating the underlying surface information for the input $\mathcal{PC}$.

The graph convolutional network is limited to a rigid structure of graphs on which it operates. Thus, it is necessary to define it before starting to train the network. Theoretically, it is possible to train the complete reconstruction with $\mathcal{N}_{recon}$ on an already highly detailed graph with many edges and vertices. However, such a graph is harder to deform in a correct way to approximate $\hat{\mathcal{M}}$, since it has many more trainable variables and thus may take much longer to converge, if at all. Hence, it would be easier to first deform an initial mesh with a low amount of detail, then increase its detail level over and over, deform anew, until it converges. Wang et al. [WZL+18] describe an *unpooling layer*, which enables $\mathcal{M}_i$ to gain more detail, every time it is applied to it ($\mathcal{M}_0 \cdots \mathcal{M}_2$). This edge-based unpooling layer adds a new edge in the midpoint of each neighboring edge for the graph $\mathcal{G}$ as seen in figure 4.6. Subsequently, it connects the new ones, for that the amount of neighbors for each vertex stays the same. Though, the structure of the resulting graphs has to be known as well. Otherwise stable training of $\mathcal{N}_{gcn}$ would not be possible. $\mathcal{N}_{gcn}$ utilizes the layer twice, each time after deforming the current state of the graph and projecting v_i from the $\mathcal{PC}$ onto $\mathcal{M}_i$.

Since the data in $\mathcal{PC}$, may not always be uniformly sampled (See section 4.1.7), it is essential to compensate for the positional bias of the $\mathcal{PC}$ in relation to $\mathcal{M}_i$ as seen in figure 4.7. $\mathcal{N}_{gcn}$ begins with a **graph alignment layer** , moving the arithmetic midpoint of the graph $\mathcal{M}_i$ on top of the arithmetic midpoint of $\mathbf{x}_i \in \mathcal{PC}$:

$$\forall \mathbf{x}_{gcn} \in \mathcal{M}_i : \mathbf{x}'_{gcn} = \mathbf{x}_{gcn} + \left(\frac{1}{|\mathcal{PC}|} \sum_{x_i \in \mathcal{PC}} x_i - \frac{1}{|\mathcal{M}_i|} \sum_{x_{gcn} \in \mathcal{M}_i} x_{gcn}\right) \tag{4.2}$$

After repositioning the graph, it is rescaled to the value domain of $[-1,1]^3$, if it exceeds it. Besides, this facilitates learning a more generalized form for deforming

Figure 4.7.: Initial ellipsoid in relation to input pointcloud $\mathcal{PC}$ before aligning on the left. After *align graph* layer on the right. Thus, eliminating positional bias and letting the mesh shrink during deformation, rather than letting it grow.

the mesh, for that $\mathcal{N}_{gcn}$ can not rely on absolut but rather relative positioning of $\mathcal{M}_i$ during convolution.

Feature projection

Considering only *convolutional operations*, *graph alignment layer* and *unpooling layer* of $\mathcal{N}_{gcn}$, a connection from $\mathcal{N}_{flex}$ has yet to be made. Ideally, a correlation between $\mathcal{PC}$ and $\mathcal{M}_i$ is created. The goal of $\mathcal{N}_{gcn}$ is to deform $\mathcal{M}_i$ based on learned feature vectors $\mathbf{v}_i$ from $\mathcal{N}_{flex}$. Mathematically by default, no direct correlation between these two exist. Thus, the *feature projection layer* is proposed. It assigns for each vertex in $\mathcal{M}_i$, features from $\mathcal{N}_{flex}$ based on the distance from $\mathcal{M}_i$ to their k-nearest neighbors in $\mathcal{PC}$.

As seen in blue in figure 4.8, every point in $\mathcal{PC}$ resides in the same coordinate system as vertices $p_j \in \mathcal{M}(V,E)$ (Illustrated as green points with their connecting vertices in black). For each vertex p_j, the *feature projection layer* computes the k-nearest neighbors in $\mathcal{PC}$ (Here $k = 6$). Higher values for k compensate for unstable performance of nearest neighbor search, while lower values speed up excecution and thus training time. Features $[\mathbf{v}_n^s, \cdots, \mathbf{v}_{n+k}^s]$ of points $[\mathbf{x}_n^s, \cdots, \mathbf{x}_{n+k}^s]$ are scaled by the distances $d_i = |\mathbf{x}_i - p_j|$. The final feature vector f_i assigned to p_i is calculated like the following:

$$\forall p_j \in \mathcal{M}_i : f_i^s = \frac{1}{k} \sum_{\forall x_i^s \in knn(p_j), s \in \mathcal{S}} \frac{v_i^s}{1 + |x_i^s - p_j|^2} \tag{4.3}$$

Where v_i is the respective feature vector stored at point $\mathbf{x}_i^s$. This is repeated for each scaling s in $\mathcal{S}$ of $\mathcal{N}_{flex}$ where a feature vector is propagated out of the subnetwork (See figure 4.4).

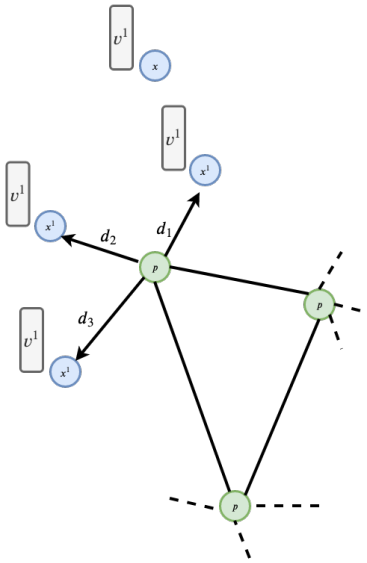


Figure 4.8.: *graph projection* layer, assigning feature vectors $\mathbf{v}^j$ to nodes in $\mathcal{M}_i$ based on the distance to their nearest neighbor in $\mathcal{PC}$.

4.1.5. Loss functions

s The losses are defined based on the network of Wang et al. [WZL+18], and adapted accordingly to work with point cloud based input data. Given $\mathbf{x}_i$ and normal orientation $\mathbf{n}_i$, an objective function $\mathcal{F}$ is defined as follows to help generate good-looking results.

A symmetrical *chamfer distance loss* l_{ch} tries to ensure that vertices of $\mathcal{M}_i$ are located close to other points $\mathbf{x}_i$ in ground truth data $\mathcal{PC}$. It sums up for each vertex $p_j \in \mathcal{M}_i$ the distance to its nearest neighbor in $\mathcal{PC}$ and for each point $\mathbf{x}_i \in \mathcal{PC}$ the distance to its nearest neighbor in $\mathcal{M}_i$. Though, it is scaled by sizes of $|\mathbf{x}_i|$ relative to $|p_j|$, depending on how often $\mathcal{M}_i$ already has been unpooled (See Appendix A for their formula).

Furthermore, Wang et al. propose an edge length loss l_{edge} in $\mathcal{F}$, reducing the overall mean of lengths of the edges. And scaling it up by a predetermined constant value of 300, as seen in formula 4.4. Only considering l_{ch} and l_{edge} ignores the orientation of neighboring faces. Wang et al.'s l_{cos} term guides $\mathcal{F}$ to orientate neighboring edges towards the same direction since most real meshes have smooth surfaces. This approximating term is illustrated in formula 4.5. Additionally, since $l_{ch}, l_{edge}, l_{cos}$ may still lead to optimizations which gravitate towards local minima, they introduce a laplacian regularizer l_{lap} which avoids self intersecting meshes and moving single vertices too freely during the deformation process (See Appendix A for their

formula).

$$l_{edge} = \frac{300}{|E|} \sum_{e \in E} \|e\|^2 \tag{4.4}$$

$$l_{cos} = \frac{1}{2|E|} \sum_{e \in E} normal(e) \cdot e \tag{4.5}$$

In most extreme cases, it is possible for different, not neighboring vertices to land on top on each other, or close to each other with distances $< \epsilon$. Since this case is not covered by avoiding selfintersections with the laplacian regularizer, a *collapse loss* l_{col} is included in $\mathcal{F}$, defined as:

$$l_{col} = \frac{1}{|\mathcal{M}_i|} \sum_{nn(p_j) - p_j > \epsilon, \forall p_j \in \mathcal{M}_i} 1 \tag{4.6}$$

$\mathcal{N}_{recon}$ optimizes against the objective function $\mathcal{F} = l_{ch} + l_{col} + l_{cos} + l_{edge}$ to obtain an approximate mesh $\hat{\mathcal{M}}$ which characterizes $\mathcal{PC}$.

4.1.6. Network configurations C

Finding suitable hyperparameters and an appropriate network structure is no easy task. Tuning hyperparameters in small increments, removing or including more layers in either $\mathcal{N}_{recon}$ or $\mathcal{N}_{flex}$ may change results drastically. Therefore, some configurations have advantages over other configuration, while still having disadvantages in other aspects. Thus, several configurations have been carved out and are specified in detail later evaluation. This section illustrates how the configurations deviate from the general network structure presented in section 4.1.1 through 4.1.4. Though, every configuration uses an ellipsoid triangle/quad mesh $\mathcal{M}_i$ with 156 vertices for the deformation process of $\mathcal{N}_{gcn}$. Consecutive *unpooling* operations increase the count to 618, and finally to 2466 vertices.

C_1**: Baseconfiguration** Configuration C_1, as described in this chapter and seen in figure 4.9, illustrates the base structure of $\mathcal{N}_{recon}$. It is trained on an input $\mathcal{PC}$ with 1024 as well as 800 samples and increases the detail of $\mathcal{M}_i$ twice.

C_2**: High detailed baseconfiguration** Configuration C_2 corresponds to C_1 almost identically, with the exception of increasing the count of *unpooling layer* from two to three, thus increasing the number of vertices to 10626. As seen in figure 4.10 marked as green layer components, $\mathcal{N}_{gcn}$ is expanded by appending another *projection* and *unpooling* layer, as well as a *convolution* block at the end. Similarly C_2 is trained with 1024 as well as 800 samples in $\mathcal{PC}$.

C_3**: Biased neighbor features**

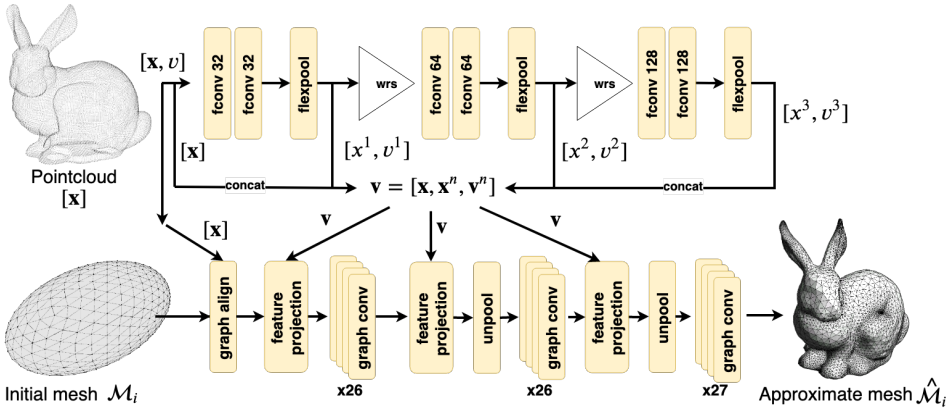


Figure 4.9.: Baseconfiguration C_1 of $\mathcal{N}_{recon}$ showing flex conv feature extraction $\mathcal{N}_{flex}$ in the upper half, and graph deformation network $\mathcal{N}_{gcn}$ in the lower part. The complete network $\mathcal{N}_{recon}$ takes a point cloud and an initial mesh as input, approximating the underlying surface information of the point cloud with $\hat{\mathcal{M}}_i$ after deforming the mesh, based on features learned in the point cloud

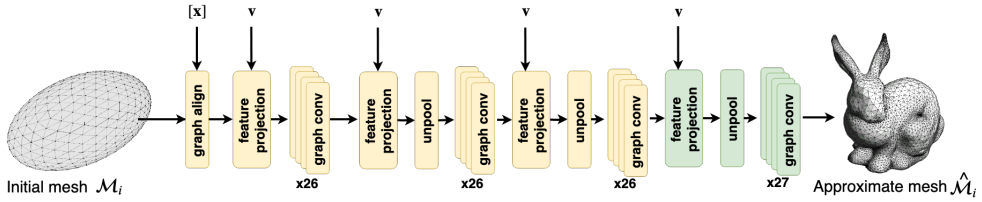


Figure 4.10.: Configuration C_2 of $\mathcal{N}_{recon}$, showing its difference in $\mathcal{N}_{gcn}$ to configuration C_1. C_2 introduces an extra *projection*, *unpooling* layer and a *graph convolution* block. Thus, allowing for a higher detailed approximate mesh $\hat{\mathcal{M}}$

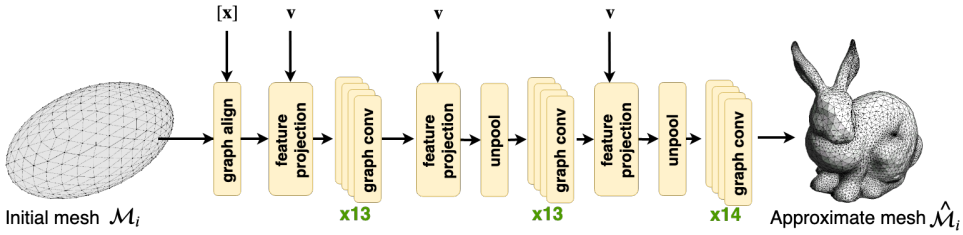


Figure 4.11.: Showing only $\mathcal{N}_{gcn}$ of $\mathcal{N}_{recon}$ with $\mathcal{N}_{flex}$ being the same as in configuration C_{1-3}. In contrast to C_1, Configuration C_4 reduces the number of used layers for deforming the mesh in $\mathcal{N}_{gcn}$.

Configuration C_3, similarly configured like C_1, but with a difference in the implementation of the *projection layer*. Instead of only projecting feature vectors v_i of $\mathcal{PC}$ onto the mesh, it also adds the mean vector $\mathbf{x}_i$ to the final feature vector f_i^s for $\mathcal{M}_i$.

$$\forall p_j \in \mathcal{M}_i : y_i^s = \frac{1}{k} \sum_{\forall x_i^s \in knn(p_j), s \in \mathcal{S}} \frac{x_i^s}{1 + |x_i^s - p_j|^2} \tag{4.7}$$

Thus, $\mathcal{N}_{gcn}$ utilizes the following feature vector f_i after projecting in each detail step of the mesh $\mathcal{M}_i$:

$$f_i = [f_i, y_i^1, v_i^1, y_i^2, v_i^2, y_i^3, v_i^3,]$$

Additionally, as a variation of C_3, the first layer of $\mathcal{N}_{gcn}$, the *graph alignment layer* is omitted and thus defining configuration C_3'. Removing the layer introduces a positional bias of the initial shape $\mathcal{M}_i$ into the network, but leads to interesting reconstructions worthy of evaluating in more detail.

C_4: Compact GCN with simple projection Finally, based on configuration C_1, configuration C_4 reduces the amount of convolutional layers in $\mathcal{N}_{gcn}$ as seen in figure 4.11. Furthermore, the distance based falloff $\frac{1}{1+d}$ in the *projection layer* is excluded, while also computing only one nearest neighbor, $k = 1$ in that particular layer.

4.1.7. Dataset

Finding and designing a suitable dataset $\mathcal{D}$ to train the network on is just as crucial as designing its structure. Such a suitable dataset is *ModelNet*, offering 40 different object categories and thousands of fully meshed objects. By default, it does not offer a point cloud representation, as required by $\mathcal{N}_{recon}$. Qi et al. [QYSG17] propose a resampled version of the classical *ModelNet40* dataset which can be utilized by $\mathcal{N}_{recon}$. The official split of 9843 shapes for training and 2468 for testing is used as well.

Even though *ModelNet40*, and therefore the transformed point cloud dataset by Qi et al. offers, as the name suggests, 40 different object categories, only a subset of them is chosen for training purposes of $\mathcal{N}_{recon}$. The subset of $\mathcal{D}_{m40}$ is defined as

$\mathcal{D}_{big}$ = [*airplane*, *bed*, *bottle*, *bowl*, *chair*, *guitar*, *toilet*] It is chosen based on the goal of learning a generalized deformation operation by showing preferably vastly different object categories to the network, all with interesting object features, which $\mathcal{N}_{recon}$ should learn. Though, all categories are of anorganic matter.

Point cloud data used for inference is not always sampled perfectly on the surface of an object. Random noise on these coordinates is common occurance. For that reason, to aid $\mathcal{N}_{recon}$ learning a more generalized reconstruction, each of the dataset categories are augmented with random noise added to each vertex in the point cloud by a factor q_{noise}.

Interesting features for reconstruction include:

- long appendages (leg of a chair)
- thin appendages (airplane wings)
- hard edges (toilet flushing cistern)
- uniform curvatures (bowl)
- objects of *genus* > 0 (some chairs/airplanes,guitars)

5. Results

By the nature of the defined configurations C_i, the neural network *points2mesh* has several reconstructions for each input point cloud $\mathcal{PC}$. Since it is not easy to estimate the quality of reconstructed meshes only based on numerical values, it is vital to assess their visual quality, tough highly subjective. Therefore, in this chapter, a wide range of reconstructed objects of various categories C_i are presented and displayed next to their input point cloud $\mathcal{PC}$ and their ground truth mesh. Some interesting features of reconstructions are pointed out, and presented with reconstructions of comparable reconstruction methods, as well as the alternative approach to train the network as outline in 4.1.2.

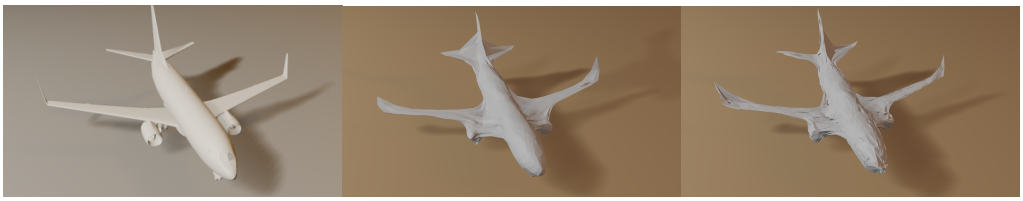
5.1. Compared methods

Restoring polygonal meshes directly from point cloud data is a new domain of geometry reconstruction, which yet has to be explored deeply. Apart from *points2mesh*, no other methods relying on neural networks and deep learning techniques have yet been proposed for this kind of mode of data transformation. For this reason, *points2mesh* has to compete against techniques from traditional approaches like *ball-pivoting algorithm* [BMR+99](BPA) and *instant field meshes* [JTPSH15] (IFM). Furthermore, by adding an intermediate step of voxelization of $\mathcal{PC}$, this work also can be compared against *deep marching cubes* [LDG18] (DMC), which transforms voxelized data into meshes based on deep learning approaches and the classical marching cubes algorithm.

5.2. Result visualization

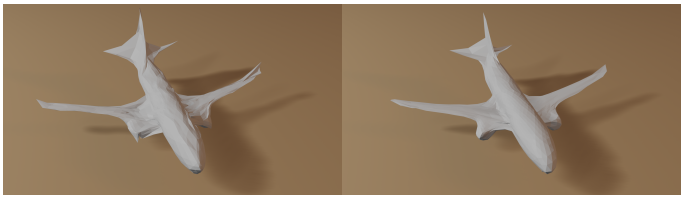
5.1 shows the reconstruction from a point cloud sample size of 1024 from the *category airplane*. 5.1a shows the ground truth mesh, and 5.1d its sampled point cloud. Figures 5.1b, 5.1c, 5.1e, 5.1f show the reconstruction from $\mathcal{N}_{recon}$ with configurations C_1 through C_4.

The last row depicts reconstructions with *BPA*, *IFM*, and *DMC*. While the reconstruction with configuration C_1 represents the most interesting features of the airplane, especially the small winglets, it still has some errors. One of the errors is at the rear of

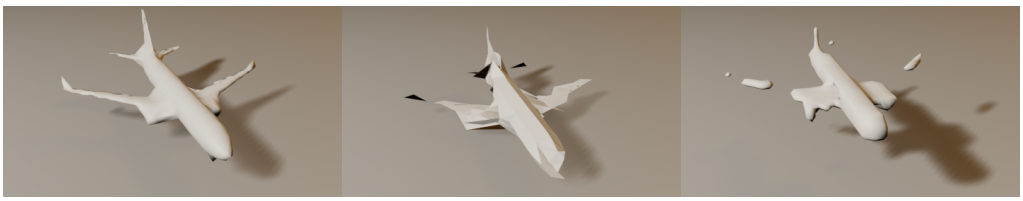


(a) Ground truth mesh. **(b)** Configuration C_1. **(c)** Configuration C_2.

(d) Sampled point cloud from ground truth mesh.

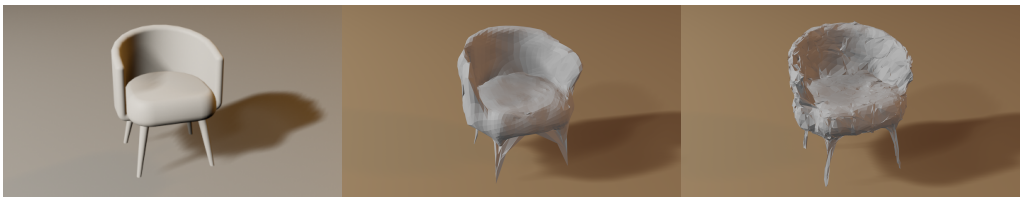


(e) Configuration C_3. **(f)** Configuration C_4.



(g) Reconstrucion with *BPA*. **(h)** Reconstruction with *IFM*. **(i)** Reconstruction with *DMC*.

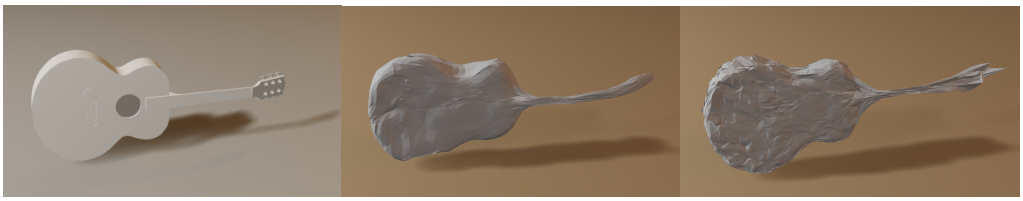
Figure 5.1.: Reconstruction of an airplane from point cloud with 1024 samples.



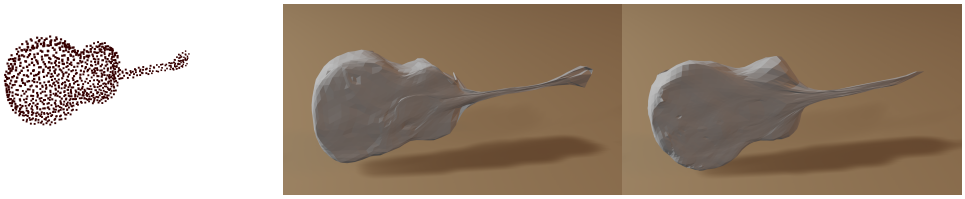
(a) Ground truth mesh. **(b)** Configuration C_1. **(c)** Configuration C_2.

(d) Configuration C_3. **(e)** Configuration C_4.

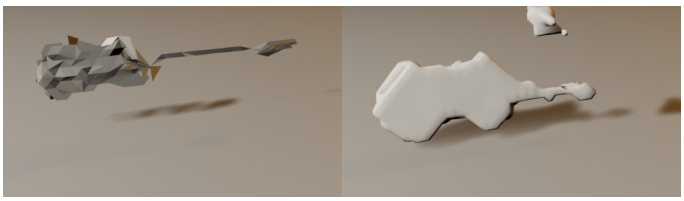
Figure 5.2.: Reconstruction of a chair from point cloud with 1024 samples.



(a) Ground truth mesh. **(b)** Configuration C_1. **(c)** Configuration C_2.

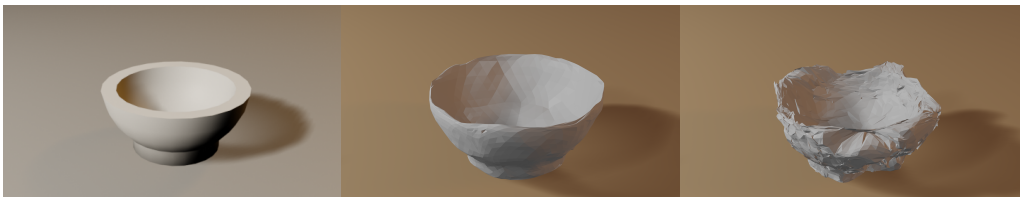


(d) Sampled point cloud from ground truth mesh. **(e)** Configuration C_3. **(f)** Configuration C_4.

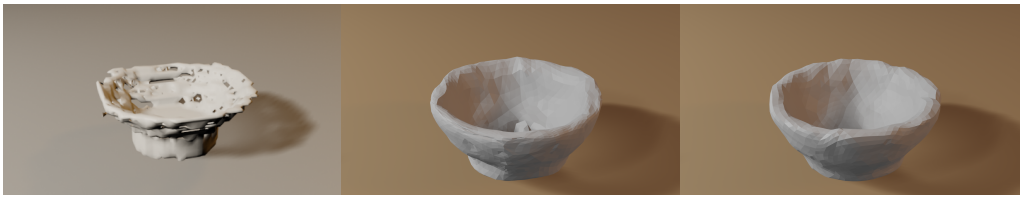


(g) *IFM* guitar reconstruction. **(h)** *DMC* guitar reconstruction.

Figure 5.3.: Reconstruction of a guitar from point cloud with 1024 samples.



(a) Ground truth mesh. **(b)** Configuration C_1. **(c)** Configuration C_2.



(d) Reconstruction with *DMC* **(e)** Configuration C_3. **(f)** Configuration C_4.

Figure 5.4.: Reconstruction of a bowl from point cloud with 1024 samples.

the plane with an edge dangling in the air, spanning over it. The same error happens with configuration C_3 and C_4 too. Configuration C_2 seems to represent the airplane's features the best, though the least smooth of all reconstructions. As seen in figures 5.1f and 5.2c, configuration C_2 is generally depicting the ground truth mesh in a good way, though not smooth. Considering other reconstruction methods, different problems, but also positives can be seen. *BPA* reproduces smooth results as seen in figure 5.1g, while *IFM* does not perform as well on low samples (256 and 1024) as seen in figure 5.1h. Generally *DMC* does not perform good either as seen in figures 5.1i and 5.3h.

Similarly, figure 5.2 through **??** show reconstructions with the same configurations C_i from $\mathcal{N}_{recon}$, but with an object from the categories *chair*, *guitar*, *bowl*. These categories were included in the training set of $\mathcal{N}_{recon}$, though every reconstructed object sample originate from a test set, and thus have not be seen during the training stage. Similar behavior of the four network configurations, as described above, can be seen again, including the spanning of edges over empty space, like in figure 5.2d, between the legs of the chair.

Finally, figure **??** shows the reconstruction of chairs, based on $\mathcal{N}_{opt}$. Only the reconstruction with the configuration C_1 is shown since it produces the most interesting results[1]. Configurations C_2 through C_4 produce similar results. Though no ground truth data was available during training time, reconstruction of the airplane is still possible. It is visible, that only 256 samples are not enough to create a good reconstruction. Starting with 1024 samples, the airplane reconstruction is closer to the same reconstruction as with $\mathcal{N}_{recon}$ as seen in **??**. Similarly, as seen in figure **??**, the reconstructed guitar can be seen, though not as clean as the guitar from 5.3. Reconstrucions of different chairs as seen in figure **??** with $\mathcal{N}_{opt}$ also show the problem with dangling edges in empty space to be a more common occurence.

Reconstructing watertight meshes allows to easily 3D print them. Figure 5.5 shows the reconstruction with $\mathcal{N}_{recon}$ with configuration C_1. The prediction then is fed into a 3D toolpath generator (slicer)[Ar11] and then printed with a *PRUSA I3 MK3S* at $0.015mm$ layer hight, taking approximately 8 hours printing time.

[1] See Appendix A for some reconstruction with other configurations

(a) 3D printed guitar reconstructed with $\mathcal{N}_{recon}$ **(b)** 3D printed car reconstructed with $\mathcal{N}_{recon}$ **(c)** 3D printed toilet reconstructed with $\mathcal{N}_{recon}$

Figure 5.5.: 3D printed reconstruction of guitar, car and toilet of $\mathcal{N}_{recon}$ and C_1 with 1024 samples.

6. Evaluation

An indefinite amount of different meshes can represent the same object. Consequently, even more meshes exist, which approximate that same object. Therefore, quantifying the quality of reconstructed meshes is a critical aspect for this application, likewise for similar ones. Only by being able to determine applications with higher qualitative results facilitate improvement in the field of mesh reconstruction. *points2mesh* and its described configurations C_i aim to improve on previous work, reconstructing meshes from point cloud data. Its advantages and drawbacks, in contrast to similar efforts, or even reconstruction methods working on other data types, rather than point cloud data, have to be clearly defined and evaluated. Even though the goal of this work is to reconstruct visually pleasing reconstructions for mesh approximations, considering raw numeric evaluations is nevertheless a crucial component.

In section 6.1, the experimental setup is described, which tries to assess the quality of reconstructed meshes and compare the defined metrics to other reconstruction methods. Subsequently, the results of the experiment are presented in section 6.2, where each one is accompanied by tabulated results.

6.1. Experimental setup

Finding a proper way to quantify the approximation of a mesh in comparison to its reconstruction is no straightforward task. Many different methods exist, each with its advantages and drawbacks. Depending on the specific application like the size of the mesh, the number of vertices and edges, or features to compare, optimal methods of comparisons may differ, if any even exist. For the proposed network *points2mesh*, several metrics are chosen, which may lead to conclusions about the quality of the reconstructed mesh. Not only are they applicable to *points2mesh*, but to other reconstruction methods as well. Thus, these metrics can be compared to each other, and an argument may be formed about their reconstruction capacity. This section specifies aspects to evaluate of *points2mesh*'s configurations C_i. Furthermore, utilized metrics for the comparison are detailed in the subsequent subsection. Finally, competing methods for reconstruction are specified, to define a frame of reference against which *points2mesh* should be able to hold up or even exceed. Finally, a collection of inference examples of the second approach for network training as detailed in 4.1.2 are listed as well.

6.1.1. Evaluation aspects

There are several aspects to evaluate *points2mesh*. Not only are network configurations C_i a crucial consideration, but so is computation time, as well as generalization capabilities and starting meshes $\mathcal{M}_i$ (Ellipsoid, Torus). Finally, attaining visually pleasing meshes is one such goal too, and thus has to be evaluated as well.

Therefore, the following aspects are chosen to evaluate.

- **Configurations** C_i: Each configuration, as described in 4.1.6, has its purpose for specific reconstruction capabilities. Evaluating every one of them may lead to conlcusions about objective rankings, and thus to a choice of best network configuration for reconstructions with the highest quality.

- **Input sample size**: Based on the input sample size of the point cloud, harder, or easier tasks for deforming the ellipsoid are assumed. Evaluating the quality of the reconstructions may lead to conclusions about the amount of necessary samples for proper inference. The input sample sizes are $\in [256, 1024, 7500]$.

- **Visual quality**: While being subjective, taking a closer look how good the reconstruction seems to represent the ground truth mesh is still relevant.

- **Generalization capabilities**: Evaluating how well configrations C_i are capable of generalize the problem of reconstruction is a crucial aspect for any neural network.

- **Changing the basemesh** $\mathcal{M}_i$: *points2mesh* is only able to reconstruct meshes of the same *genus* as the input mesh $\mathcal{M}_i$. Thus, testing configurations C_i with different $\mathcal{M}_i$ of higher *genus* like a toroidal shape may lead to different results and reconstructions with higher *genus*.

- **Noise robustness**: Acquiring unstructured points in three-dimensional space from real-world object often entails noisy data. The perfect acquisition is already nearly impossible due to imperfection in the hardware. Thus, training $\mathcal{N}_{recon}$ with noisy data is an essential part of the evaluation.

- **Computation time**: When reconstruction methods are used for time critical application, their computation time is important to evaluate.

While evaluating configurations, the visual quality and computation time can be compared to other methods, generalization and different used basemeshes are isolated during evaluation due to their nature and thus are only compared to each other.

6.1.2. Evaluation metrics

Choosing proper evaluation metrics is ambiguous, but several are often chosen in recent work of mesh reconstruction with favorable information content. While some of them may help to lead to objective conclusions about quality, others are subjective in nature. These metrics are described in the following.

Meshdistance d_m

In 3D vision, a commonly used metric to compare the quality of meshes is done with completeness of the predicted mesh in comparison to the ground truth mesh. Based on uniformly sampled points on both meshes, completeness describes the distance of points of ground truth to the predicted mesh. Stutz et al. [Stu17, SG18] introduce a parallel $C++$ implementation of *meshdistance* which calculates completeness, given a ground truth mesh and a prediction. Distances are calculated by a point-to-triangle method by Christopher Batty[1] (See chapter **??**)

Visual evidence

Though subective, taking a closer look at, how well a reconstructed mesh seems to represent the ground truth mesh is important too. Relying only on numeric values may lead to reconstructions with low values, but not visually pleasing results. This is due to the complicated nature of the problem itself. For a given point cloud, an indefinite amount of different reconstructions are possible. Therefore, inspecting the meshes is necessary.

6.2. Experiment results

The numerical evaluation of *points2mesh* and its configuration C_i is denoted in a tabulated form. Since *points2mesh* is trained on the ModelNet dataset, data samples from this dataset are selected for inference. About 20 percent of the provided point clouds are left out of training and thus used as the test-set. Therefore, the network has never seen it before. In that way, its quality can be measured against the other methods and is comparable in the first place. The other two methods against which *points2mesh* is compared against, should be able to handle point cloud data from ModelNet. In the following sections, the previously explained metrics are listed and compiled against each other. Likewise, *points2mesh*'s reconstructions are listed to comparable methods with the same metrics.

[1] https://github.com/christopherbatty/SDFGen last visited: July 11, 2019

6.2.1. Distance metric results

The first comparison consists of evaluating the distance metric d_m for every configuration C_i and *ball-pivoting algorithm*(*BPA*) and *instant field meshes*(*IFM*). One hundred point clouds of each object category are taken at random from the ModelNet dataset. For each method, an approximation mesh M_p is calculated, and all distance measure metrics to its ground truth mesh computed. The mean value of these metrics is noted in the tables. While the first table denotes reconstruction from 256 samples, the following denotes reconstruction from 1024 and subsequently from 7500 points. Furthermore, 6.1 also describes the distance metric d_m from the hundred object samples to a not yet deformed sphere[2]. This baseline for d_m gives a general understanding of d_m. *Deep marching cubes* unfortunately transforms and normalizes their reconstructed mesh in such a way, that it cannot be numerically evaluated and thus not compared to *points2mesh* and *IFM* or *BPA*. However, a visual comparison is possible and still appealing to conduct, since it is the only other deep learning based reconstruction method working (indirectly[3]) on point cloud data, transforming it into a poly-mesh.

First, as seen in figure 6.1 through A.3, reconstructed meshes already differ depending on the employed configuration C_i. For each cell, 100 objects of the *ModelNet* test dataset are used for inference and evaluate with d_m against the ground truth. Denoted is the mean value of that distance value. The columns indicate the used configuration or method for the reconstruction. The last column describes the mean value of d_m of a basic spheroidal shape evaluated against all test samples. Comparing this value offers a baseline value, which is an approximate upper bound for the quality of the reconstructed meshes. A perfectly reconstructed mesh has a d_m value of 0.

The initial goal is to figure out the *best* configuration out of the four proposed ones. Reaching that state requires preceding evaluation of d_m. Each of them has its merits, but finding the configuration, generally capable of the best reconstructions is essential. Taking a look at the mean values of d_m in table 6.2 for each configuration C_i shows a small advantage of C_1 and C_4 over the other two configurations. Though small, smaller values are still visible throughout. Exemplary, in figure 5.1, the resemblance of the reconstructions are pretty close, though with noticeable differences. Taking a closer look at the airplane wings in the first row, C_1, C_2 and C_4 reconstruct the small winglets correctly at the end of each wing blade. With configuration C_3, they are not noticeable anymore. The advantage can also be seen in the reconstruction of chairs with 1024 samples. Configuration C_1, C_2 and C_4 do not have the problem with edges in empty space like configuration C_3, as seen in figure 5.2. The general low values of C_2, with a higher amount of vertices for the reconstructed object can

[2]The sphere object is defined by the implicit surface furmula $x^2 + \frac{y^2}{2} + z^2 = 0$ and created in meshlab with a voxel size of 0.055

[3]*DMC* preprocesses the point cloud by transforming it into a voxelized representation, thus not a direct tranformation from point cloud data to meshes.

also be confirmed visually, if compared to 5.1c, 5.2c and 5.3c. Though representing the features the best, the general smoothness factor of the object is way lower than the other configurations. Configurations C_1, C_3 and C_4 seem visually more pleasing caused by the smoothness of the surface, though C_2 is able to represent the objects in a better way, if the smoothness is disregarded.

Furthermore, comparing the results of d_m between 256, 1024 and 7500 samples of each configuration, a more significant gap between 256 to 1024 is present, than between 1024 and 7500 samples. While the mean value of d_m gets lower using 1024 samples rather than 256, this is generally not true anymore for changing from 1024 samples to 7500. Since solving the problem of reconstruction is objectively harder the more points are present. However, solving the same problem with only 256 samples may also lead to higher values for d_m, since at some amount of samples, there are not enough present anymore to represent the ground truth mesh at an appropriate level. Notably, in the case of configuration C_1, its mean value for d_m for all object categories is 0.020 for 256 samples, reducing to 0.017 at 1024 samples, but increasing again to 0.024 for 7500 samples. Additionally, configuration C_3 has worse results at 7500 samples than at 1024 samples as well.

Confirming these numerical results with the visual evidence in figure **??**, while a big jump in visual quality is evident, changing the sample size from 256 to 1024, the change to 7500 samples not so much. As seen in **??**, the airplane even loses a feature (fron wheel) at the last step and thus could explain for higher or equal mean values of d_m.

Tables **??**, **??** and **??** show the same evaluation of d_m for $\mathcal{N}_{opt}$, though only the mean value is indicated for each configuration. As seen in the reconstructions in figures **??**, **??** and **??**, the results are generally worse. The reconstructions have less smooth surfaces, less distinct features shown, and higher values for d_m. A complete evaluation for d_m can be found in the appendix A

Table 6.1.: Distance metric evaluations with 256 samples per point cloud for $\mathcal{N}_{recon}$. Calculated with *meshdistance* metric. Only considering configurations C_1 and C_2. Lower values are better.

Object	C_1	C_2	C_3	C_4	BPA	*IFM*	None
airplane	0.013	0.009	0.011	0.011	0.013	0.153	0.251
bed	0.019	0.014	0.018	0.017	0.016	0.123	0.163
bottle	0.011	0.008	0.011	0.010	0.007	0.134	0.201
bowl	0.024	0.018	0.018	0.022	0.022	0.133	0.148
car	0.020	0.015	0.019	0.020	0.020	0.199	0.155
chair	0.019	0.013	0.018	0.016	0.027	0.188	0.195
guitar	0.015	0.009	0.015	0.013	0.005	0.325	0.338
toilet	0.043	0.020	0.052	0.027	0.016	0.135	0.138
mean	0.020	0.013	0.021	0.016	0.016	0.200	0.1986

Table 6.2.: Distance metric evaluations with 1024 samples per point cloud for $\mathcal{N}_{recon}$. Calculated with *meshdistance* metric. Lower values are better.

Object	C_1	C_2	C_3	C_4	BPA	*IFM*	None
airplane	0.011	0.006	0.012	0.011	0.089	0.10	0.251
bed	0.016	0.010	0.016	0.014	0.008	0.099	0.163
bottle	0.009	0.007	0.011	0.008	0.003	0.10	0.201
bowl	0.022	0.013	0.020	0.019	0.006	0.097	0.148
car	0.019	0.012	0.018	0.018	0.009	0.163	0.155
chair	0.016	0.009	0.017	0.015	0.008	0.172	0.195
guitar	0.013	0.006	0.022	0.012	0.002	0.234	0.338
toilet	0.037	0.013	0.048	0.024	0.006	0.109	0.138
mean	0.017	0.010	0.021	0.015	0.016	0.134	0.1986

Table 6.3.: Distance metric evaluations with 7500 samples per point cloud. Calculated with *meshdistance* metric.Only considering configurations C_1 and C_2. Lower values are better.

Object	C_1	C_2	C_3	C_4	BPA	*IFM*
airplane	0.015	0.007	0.031	0.011	0.090	0.090
bed	0.021	0.011	0.10	0.015	0.005	0.089
bottle	0.014	0.007	0.115	0.007	0.001	0.096
bowl	0.026	0.016	0.127	0.017	0.003	0.091
car	0.024	0.012	0.111	0.018	0.003	0.153
chair	0.022	0.011	0.064	0.016	0.001	0.126
guitar	0.022	0.007	0.025	0.012	0.001	0.212
toilet	0.047	0.015	0.178	0.031	0.003	0.101
mean	0.024	0.011	0.083	0.016	0.013	0.119

Table 6.4.: Distance metric evaluations with 256 samples per point cloud for $\mathcal{N}_{opt}$. Calculated with *meshdistance* metric. Only considering configurations C_1 and C_2. Lower values are better.

Object	C_1	C_2	BPA	*IFM*	None
mean	0.035	0.061	0.016	0.200	0.1986

Table 6.5.: Distance metric evaluations with 1024 samples per point cloud for $\mathcal{N}_{opt}$. Calculated with *meshdistance* metric. Lower values are better.

Object	C_1	C_2	C_3	C_4	BPA	*IFM*	None
mean	0.018	0.024	0.021	0.018	0.016	0.134	0.1986

Table 6.6.: Distance metric evaluations with 7500 samples per point cloud. Calculated with *meshdistance* metric.Only considering configurations C_1 and C_2. Lower values are better.

Object	C_1	C_2	C_3	C_4	BPA	*IFM*
mean	0.018	0.015	0.025	0.027	0.013	0.119

In contrast, the traditional methods, as in *BPA* and *IFM*, get better results; the more samples are provided, both numerically as well as visually. *IFM* seems to generally struggle to reconstruct visually pleasing results when the number of samples is below 7500. Not only that but its numerical values of d_m are sometimes higher than a basic spheroid shape in contrast to its baseline object (See table 6.1 and 6.2, mean value for *car*). Generally, the metrics have higher values for *IFM*, while *BPA* and N_{recon} are much closer to each other.

Not only are they closer to each other, but rather *BPA* produces lower mean values than any configuration of N_{recon} with the exception of C_2. Though implicating better results for any case, visually for 256 samples and 1024 samples, C_1 looks in some cases more organic, or closer to the ground truth than *BPA* (Compare figures **??** with **??**, and figures 5.1c with 5.1g). However, most reconstructions with either *BPA* or *IFM* have holes in the meshes, thus are not watertight.

Finally, inspecting reconstructions from *DMC* demonstrates generally bad results. The low amount of samples used, less than 7500, seems not to be enough for the deep learning based approach to reconstruct a precise object. At figure **??**, *DMC* somewhat reconstruct the general outline of the object, though producing big holes in the side of the object, while also hollowing out the interior of the car. Additionally, reconstructing a guitar, see figure 5.3h, a second, independent *blob* of incoherent triangles is produced, unrelated to the ground truth. Lastly, as seen in figure 5.1i, reconstructing flat surfaces, like the wings of the airplane is not possible for *DMC*, resulting in big chunks of missing surface information. However, reconstructed objects with *DMC* are watertight.

6.2.2. Generalization capabilities

Making use of reconstruction techniques often requires the method to work in a general setting. The previous experiments already showed reconstruction from *points2mesh* of test samples, which the network has not yet seen. With the same idea of feeding unknown data to the network, the next table of evaluated metrics was created. By extending the generalization concept, point cloud samples of objects outside of the trained object categories are fed into the network. For that reason, assuming to work on configuration C_1, it has been trained on four different object category collections (See section **??**). Every trained instance of C_1, now is fed one hundred point cloud samples of *bathtub* and *person*, evaluating every distance metric. Even though, C_1 has never seen any samples of these object categories.

Table 6.7.: Distance metric evaluations with 1024 samples per point cloud. Testing with point clouds from outside of trained object categories

Object	C_1	C_2	C_3	C_4	None
bathtub	0.021	0.015	0.067	0.024	0.178
person	0.154	0.137	0.181	0.146	0.352

As seen in table 6.7, all configurations except for C_3 result in low values for d_m if compared to the same metrics of no reconstruction at all. Figures **??** and **??** confirm this contention. C_2 keeps the same problem as described above, the general roughness of the surface. C_1 and C_4 produce rather similar results, while C_3 seems to struggle the most to construct a proper mesh.

6.2.3. Varying basemesh $\mathcal{M}_i$

The previously described configurations all are trained on a spheroid as initial mesh $\mathcal{M}_i$. By nature of the neural network and the sphere, objects with *genus* higher than zero cannot be reconstructed such that the predicted mesh has the same *genus*. Since it stays constant, changing the initial mesh to a toroidal mesh may lead to possible reconstructions with *genus* one. For the following evaluation, C_1 is trained on such an initial toroidal mesh.

Table 6.8.: Excerpt of evaluation table. Distance metric evaluations with 1024 samples per point cloud. Initial mesh (torus mesh) with a *genus* of one.

Object	C_1^{torus}	C_1^{sphere}
airplane	0.019	0.011
toilet	0.032	0.029

Though resulting in only marginally higher values of d_m (See table 6.8), starting with a toroidal base mesh does not solve the problem of reconstructing objects with a *genus* higher than 0. The final prediction of $\mathcal{N}_{recon}$, assuming the new base mesh, does not show any signs of a hole anywhere in the mesh, no matter the input point cloud.

6.2.4. Noise robustness

Noise often can be found in point cloud data in real-world scenarios, since recorded data is subject to imperfections in the recording equipment. Thus, reconstructing objects, even with noisy data, is an interesting subject for $\mathcal{N}_{recon}$. Table 6.9 shows similar to the earlier experiments, reconstruction of different objects categories with point cloud data, with added jitter noise (Noise level of 0.01). While d_m metrics are only slightly higher than without noise, the visual evidence shows a noticeable difference. Figure **??** depicts reconstructed airplanes with jitter noise in the input point cloud data. The general reconstruction still achieves the approximate contour of the object, however having more problems with edges dangling in empty space.

Table 6.9.: Excerpt of evaluation table. Distance metric evaluations with 1024 samples per point cloud. Comparison of configuration C_1 trained with and without noise (*noise level* of 0.01). Inference with noisy data of the same *noise level*.

Object	C_1^{noisy}	C_1^{clean}
airplane	0.090	0.012
bed	0.017	0.011
bottle	0.017	0.016
bowl	0.029	0.025
car	0.023	0.021
chair	0.018	0.018
guitar	0.015	0.014
toilet	0.041	0.016
mean	0.031	0.018

6.2.5. Evaluation time

For real-time applications like guiding autonomous cars through traffic without collision, reconstruction should happen virtually instantaneously for it to be advantageous. The computation time may be critical, and thus has been evaluated, as seen in table 6.10. *IFM* split into three configurations. IFM^{500} takes 256 samples as input and produces a mesh with 500 vertices (which is approximately the most vertices for the best result at 256 samples). Similar, IFM^{2400} and IFM^{10100} are defined for 1024 and 7500 samples. $\hat{t}$ is a mean value of the measurement for 100 reconstructed objects. Generally, *IFM* has the fastest computation times[4] at the same amount of samples when compared to C_i and *BPA*.

Table 6.10.: Inference time for configurations C and computation time for other techniques

method	$\hat{t}$ in seconds
$C_{1,3}$	0.061
C_2	0.15
C_4	0.043
IFM^{500}	0.001
IFM^{2400}	0.005
IFM^{10100}	0.167
BPA	0.062

[4]Computed with GeForce GTX 1080 Ti graphicscard and Intel(R) Xeon(R) CPU E5-2660 v3 @ 2.60GHz

7. Discussion

In the previous chapters, several configurations were presented, as well as a *supervised* and *unsupervised* version of *points2mesh*. Section 7.1 discusses their advantages and disadvantages, and analyses which one of them is the most reliable and works in the most general way. Additionally, it is compared to state of the art traditional reconstruction methods and a direct deep learning based competitor.

Furthermore, section 7.2 summarizes the scientific gap, which has been bridged by this work, as well as some general critique of the approach.

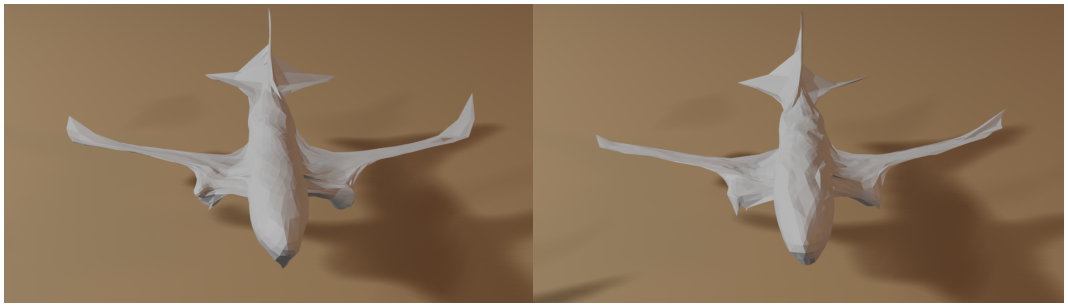
Finally, section 7.3 gives a short outlook about what could follow, and how to improve on this work for potential future work.

7.1. Variety of *points2mesh*

variations

The proposed supervised as well as *un*supervised neural network $\mathcal{N}_{recon}$, and $\mathcal{N}_{opt}$ respectively, reconstruct a tri-mesh from point cloud data. The underlying object to reconstruct has to be singular. Both versions of the neural network are trained on multiple classes at the same time, producing watertight meshes. First, configurations C_i are considered. As already established in 6, the main difference, visually and numerically are when configurations C_1, C_3 and C_4 are compared to C_2. When only looking at evaluations of d_m, configuration C_2 has far lower values than any other configuration, impartial to training with $\mathcal{N}_{recon}$ or $\mathcal{N}_{opt}$. Tables 6.1 through A.3 confirm this proposition. By extending the number of used samples for the reconstruction in the base mesh $\mathcal{M}_i$, with an extra *unpooling* step to ~10k vertices, instead of ~2.4k vertices. , the prediction may then display more detail. However, considering visual predictions of C_2 in contrast to configurations with only ~2.4k vertices, it is evident, that C_2 is generally less smooth. Taking a look at figure 5.1c, not only is the body of the airplane relatively rough, though details are much better visible. In contrast to the other three configurations, C_2 seems to be less subjectively visual pleasing, while still representing the reconstructed object accurately. Configurations C_1, C_3 and C_4 are generally smoother than C_4 as seen in figures 5.1b and 5.1e, also holding true for either $\mathcal{N}_{recon}$ and $\mathcal{N}_{opt}$. Evidently, this is due to changes in $\mathcal{N}_{gcn}$ of the neural network (See figure 4.10). The smoothness may be recovered with more extended training

or with more *hyper parameter tuning* for the normal cosine loss l_{cos}. Applications like accurate collision detection, or preparing 3D printable objects, greatly benefit from smoother predictions as seen with configuration C_1. However, improving it further for configuration C_2 has not been examined notably in the context of this work.



(a) Reconstruction with C_1, 1024 samples. Winglets visible at the end of the wings.

(b) Reconstruction with C_3, 1024 samples. Windlets not as visible at the end of the wings.

Figure 7.1.: Reconstruction from of airplane. Winglets are visible with C_1 but not as prominent with C_3.

Furthermore, configuration C_1 and C_3 are pretty similar in itself. Their main difference beinfg C_1 does not incur the nearest neighbor coordinates during the *feature projection* layer, thus not utilizing it as a feature in $\mathcal{N}_{gcn}$. Nonetheless, figures 7.1a and 7.1b show that they still learn about the same deformations for the same given input. However, C_1 can represent some features of objects better than C_3, like the winglets of the airplane in figure 7.1a or empty space between the legs of the chair in figure 7.2, while also maintaining lower values of d_m. C_1 reconstructing objects better than C_3, not only shows that the former learns a variation of nearest neighbor coordinates during the *flex conv feature extraction* but extends these features further to achieve better predictions. For that reason, C_1 seems to be the superior choice of configuration in comparison to C_3.

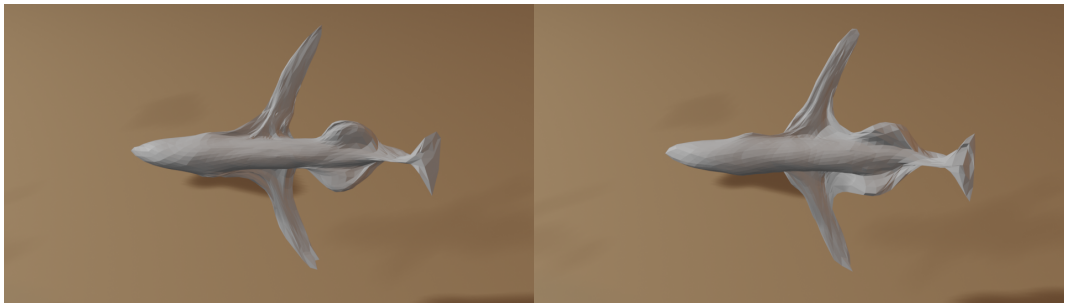
Configuration C_4 describes a more simple configuration for $\mathcal{N}_{recon}$ and $\mathcal{N}_{opt}$. While still maintaining rather low values for d_m, in addition to rather pleasing visual results, its predictions do not reach the same level as C_1 visually, or numerically as C_2 (See tables 6.1, 6.2 and A.3). Considering figure 7.3, C_4 seems to produce a little smoother results than C_1, though not as big of a gap as C_1 to C_2. However, similarly to configuration C_3, suffering from forfeiting various details of the ground truth mesh, as seen earlier in figure 7.1a. C_4 is less complex, faster to train (Around 10%) and provides the fastest inference times (See table 6.10). However, these advantages do not compensate for losing out on some features, which C_1 and C_2 are able to reconstruct.

By nature of the composition of the neural network, the *genus* of the prediction is bound to be of the same *genus* as the input mesh $\mathcal{M}_i$. Thus, if the input mesh is of a spheroidal shape (*genus* 0), the prediction always is of *genus* 0 as well. Albeit

(a) Reconstruction with C_1, 1024 samples. Empty space is present between chair legs.

(b) Reconstruction with C_3, 1024 samples. Dangling edges between legs of chair present.

Figure 7.2.: Reconstruction from of chair. Almost no empty space between legs with C_1 but not more prominent with C_3.



(a) Reconstruction with C_1, 256 samples.

(b) Reconstruction with C_4, 256 samples.

Figure 7.3.: Reconstruction from of airplane. C_1 has a higher detailed reconstruction than C_4, though with the same amount of vertices.

(a) Reconstruction with C_1, 256 samples.

(b) Ground truth mesh.

Figure 7.4.: Comparison of reconstructed chair and ground truth on the right. The prediction on the left cannot recreate the hole between the seating area and the back.

considering an input point cloud with an underlying ground truth mesh of *genus* ≥ 1, the prediction does not change its *genus*. Disparities in the *genus* lead to the cladding of holes in the mesh, as seen in figure 7.4. The ground truth chair has a hole below its backrest just above the chair seat. During the deformation in $\mathcal{N}_{gcn}$, the sphere contracts tighter around the shape of the point cloud. Even if the loss function incentives the network not to put vertices in the empty space, their connecting edges still remain stretched over it. No matter how the initial sphere is deformed, the holes cannot be adequately represented. Thus, as described in 6.2.3, the initial shape was changed to a toroidal model of *genus* 0. The metric evaluation of d_m did not increase a lot as seen in 6.8. However, $\mathcal{N}_{gcn}$ does not distinguish between different input meshes $\mathcal{M}_i$. Hence no incentive exists to either keep the hole or even move the hole towards a point in the point cloud where one could exist in the ground truth model. After only a few deformation steps, the hole collapses in on itself, consequently the difference between deforming a sphere or a torus (Or any other object of a higher *genus* for that fact) is negligible. Reconstructing objects in that way requires a better way to keep the hole in the mesh, as well as find a way to place it at the right spot (s) overlayed on the point cloud. Let alone learning if the underlying mesh even has a hole, or not.

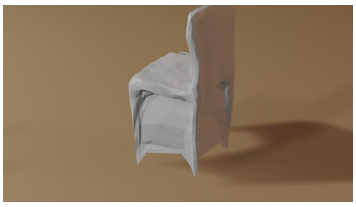
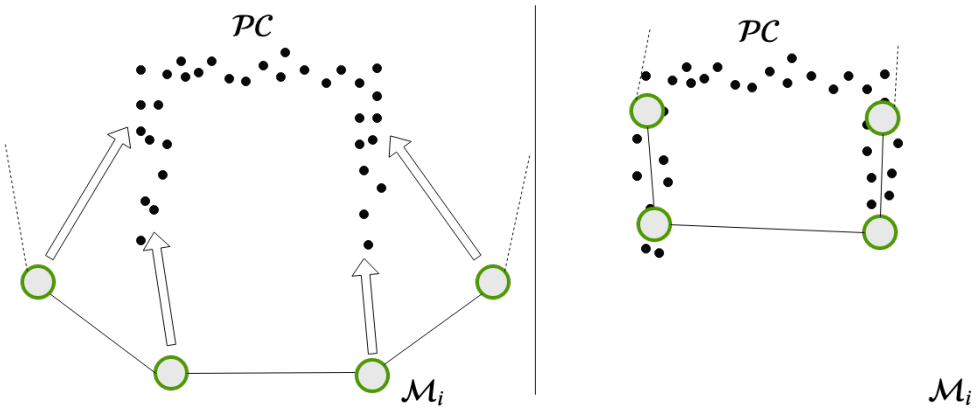


Figure 7.5.: Reconstruction of chair with C_1 at 7500 samples. A lot of edges spanned acroos the legs.

Generally, the biggest hurdle is the problem of stretching the deforming sphere over empty space in the ground truth mesh. Most prominently, it is visible in reconstructions from point cloud depicting chairs. As seen in figure 7.5, in between the legs of the chair, many edges are stretched between vertices placed directly on the leg of the chair.

Contrary to that reconstruction, figure 7.3a depicts a better reconstruction of a chair, without stretched edges over empty space. During deformation, the initial mesh contracts inwards, fitting its surface against the input point cloud data. During this process, two neighboring vertices may be moved towards two distinct semantic parts of the point cloud (Illustrated in figure 7.6a), which leads to the stretching problem. Successfully solving this problem leads to a significant improvement visually as well as numerically in the prediction. Two ways of tackling this problem are practiced in *points2mesh*. The first one is implicit. Since $\mathcal{N}_{gcn}$ split into a three-step deformation process, dangling edges gain an additional vertex during the *unpooling* layer. The newly added vertex then may move towards the connecting surface between the appendages, where the previous dangling edge existed (Illustrated in figure 7.6b).



(a) Two dimensional projection of initial mesh, in green, before convolutional layers, showing where each vertex may move to.

(b) Two dimensional projection of initial mesh, in green, after convolutional layer, moving towards point in point cloud(in black).

Figure 7.6.: Illustration of initial mesh deforming to approximate the underlying mesh of the point cloud $\mathcal{PC}$ of the legs of a chair. The edge of the two vertices in the middle is stretching over the two legs of the chair.

This has to be supported by proper feature vectors learned by $\mathcal{N}_{flex}$. Alternatively, if the vertices are already placed, neighboring vertices with long edges have to be moved in such way, that they get closer again, potentially leading to minimize that problem, which is facilitated by the edge length loss l_{edge}. Configurations C_1 and C_2 generally are able to solve this more consistently than C_3 and C_4. If $\mathcal{N}_{recon}$ is trained without $\mathcal{N}_{flex}$ by only propagating nearest neighbor coordinates in the *feature projection layer*, the stretching problem is much more prevalent, leading to the conclusion that $\mathcal{N}_{flex}$ is crucial in learning feature vectors which give the incentive to move vertices in such a way to solve the problem. Additionally, $\mathcal{N}_{recon}$ is also able to produce less of these edges than $\mathcal{N}_{opt}$, since the former has a lot more information to learn and infer information from than the latter.

On another note, two different ways of training have been proposed, one *surpervised* $\mathcal{N}_{recon}$, and the other a form of *unsupervised* $\mathcal{N}_{opt}$ training. As seen in chapter 5, while both approaches yield visually pleasing reconstructions, the *supervised* generally produces better results. Naturally, the *supervised* training has up to 40 times more samples during the training process and thus has more information with which it can be trained. However, $\mathcal{N}_{opt}$ is still a compelling case to consider, since often in real life situations, no ground truth data is available (Range scanners on autonomous cars). Assessing how good such a network reconstructs meshes based on somewhat limited training data, is therefore appealing. As shown in chapter 5, $\mathcal{N}_{opt}$ is still quite capable, though not as good as $\mathcal{N}_{recon}$. Additionally, in such conditions, noise is inevitable. Nonetheless, $\mathcal{N}_{recon}$ shows that for a noise level of 0.01 (Based on ground truth data in $[-1,1]$) reconstructions are still visually pleasing, though the metric

results of d_m suffer a little. Similarly, the generalization capabilities of $\mathcal{N}_{recon}$ have been assessed in figures **??** and **??**. Despite never seeing data of the category bathtub or person, the reconstruction does not stick out from the trained categories. Thus, somewhat robust training and inference have been achieved.

Finally, real point cloud and short time argument **TODO: reale point cloud**

Comparing alternatives

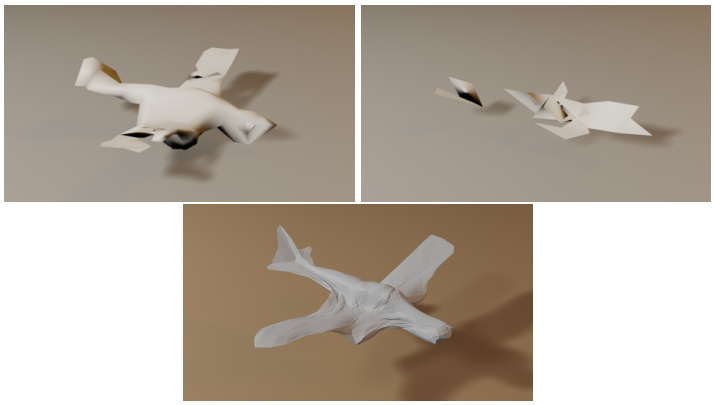


Figure 7.7.: Reconstructions of airplane with 256 samples, with *BPA*(left), *IFM*(righ) and *points2mesh*(bottom)

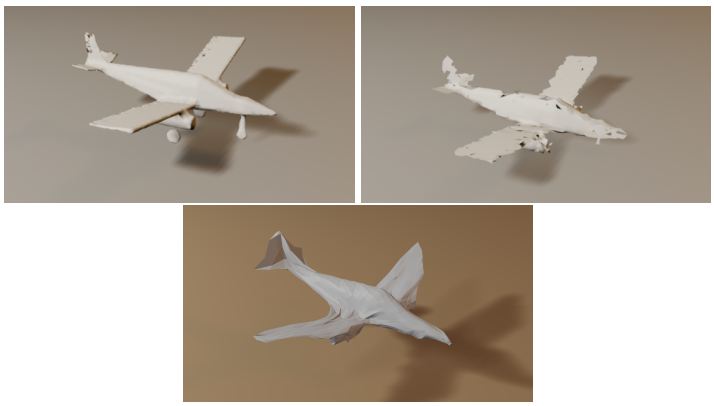


Figure 7.8.: Reconstructions of airplane with 7500 samples, with *BPA*(left), *IFM*(right) and *points2mesh*(bottom)

Reconstructing meshes from point cloud data has been researched for many years. Thus, many techniques have been developed over the years. In the context of this work, assuming traditional, non-machine learning approaches, *BPA* and *IFM* are chosen as a comparison. First, *BPA* generally yields lower values than configuration C_1 for $\mathcal{N}_{recon}$. In contrast, *IFM* has way higher numerical values for d_m in contrast

to $\mathcal{N}_{recon}$ as well as *BPA* (As seen in tables 6.1 through A.3). However, considering the visual evidence, the supposed better values for *BPA* cannot be confirmed to be generally better. Regarding the *BPA* reconstruction and the *IFM* reconstruction in figure 7.7, a clear difference can be seen in contrast to the reconstruction by *points2mesh*. At a low amount of samples (256 samples), neither *BPA* nor *IFM* produces watertight meshes. Even worse, *IFM* struggles to reconstruct the general frame of the object. Increasing the number of samples supports both *BPA* as well as *IFM*, yet is not enough for these approaches to provide watertight meshes. Even considering a reconstruction based on 7500 samples, the reconstructions of either approach is still not manifold as seen in figure 7.8. Filling these holes would be a completely new, non-trivial problem in itself. Though, visually, these reconstructions approximate the ground truth better than any configuration of *points2mesh*. However, only by the innate capability of humans to fill out the holes and thus, inferring how the mesh would look, *if* it did not have them.

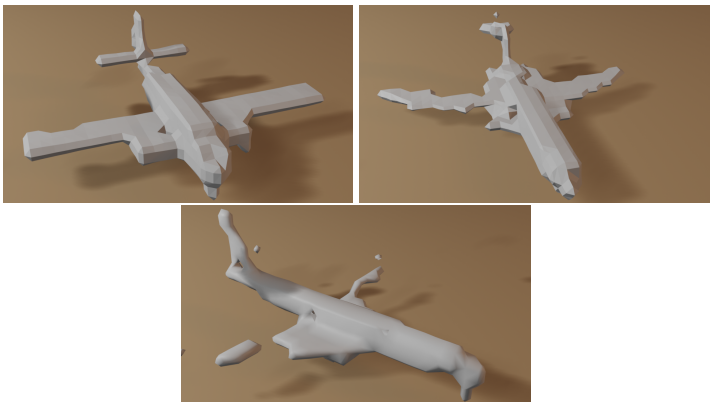


Figure 7.9.: Reconstruction from point clouds of airplanes with *DMC*. Sample size 1024

A crucial part of this work is the attempt to design a neural network for learning surface reconstruction from point cloud data. Thus, comparisons to deep learning based approaches, though not plenty, are even more critical than traditional ones. Since other deep learning based approaches transforming point cloud data directly to meshes have yet to be published, another close technique is chosen as a comparison, *deep marching cubes*. *DMC* however, operates on voxelized data as its input, but also produces meshes watertight meshes. These deep learning approaches for mesh reconstructions are still new and therefore are not as technically mature as current non-deep learning based methods. Considering the reconstructions in figure 7.9, expected features from the traditional *marching cubes* algorithm can also be seen here. Blocky reconstructions, accompanied by predetermined slopes at corners, lend a distinct look to reconstructions by either technique. In contrast, reconstructions from *points2mesh* resemble a more organic look and have to possibility of representing more details. The nature of its voxelized data restricts the visual quality of predictions by *DMC*. Though, the number of vertices is not bound to a set amount for each object,

but rather the resolution of the voxel grid. While both methods produce watertight meshes, *DMC* sometimes produces holes in the mesh where a solid surface is present in the ground truth mesh, as seen in figure 7.10. Additionally, *DMC* at times adds unexplained *blobs* of geometry in the prediction. A post-processing step could easily remove them. However, they are detrimental for applications which rely on the best possible representation for a given point cloud. Still, *DMC* has some clear

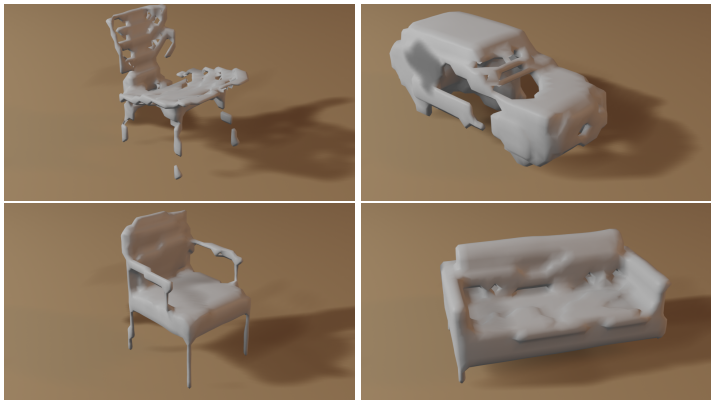


Figure 7.10.: Reconstruction from point clouds with *DMC*. Sample size 1024. Predictions in the top row have holes due to low sample size. Predictions in bottom row look sound due to no almost no curves in ground truth, and only edges along the cardinal axes.

advantages over *points2mesh*, as seen in figure 7.10. *DMC* reconstructs objects with edges only parallel to the cardinal axes better than any configuration of *points2mesh*. However, introducing edges, distributed in any way through space, its limitation are evident as seen in figures 7.9. For example, the wings of the plane on the right and on the bottom are neither x-parallel nor y-parallel. Thus, *DMC* introduces *stair*-like representation of slopes in the mesh. emphDMC does not have any way to represent curves. *points2mesh* works despite these preconditions, yet giving it an organic look. On top of that, *DMC* does not work correctly if the samples are sparse. Both, the car and chair in the top row of figure 7.10 were inferred based on 1024 samples. When the number of samples is too low, *DMC* fails to produce any geometry for parts of the mesh. In contrast, *points2mesh* only needs very few samples to produce geometry at a given part of the point cloud.

7.2. Conclusion

points2mesh describes the first[1] deep learning based neural network, transforming point cloud data to tri-meshes. Excelling in reconstructing with low-resolution data,

[1]No other network was found which directly transforms points cloud data if singular objects to meshes as of July 11, 2019

even generating watertight meshes with only 256 data points while also being capable of generalizing over objects it has never seen during training. Configuration C_1 of the supervised network $\mathcal{N}_{recon}$ is the best choice for the best reconstructions in various situations.

However, its inherent structure is restrictive. By virtue of how $\mathcal{N}_{gcn}$ works with *graph convolution*, the amount of vertices used for the reconstruction is predetermined before the training has even started. Furthermore, only objects with *genus* 0 can theoretically be reconsturcted. Additionally, there are no symmetrical restricions, and thus most predictions seem rather organic than anorganic.

Nevertheless, the inference time is quite fast, being able to keep up with its competitors while also capable of reconstructing the characteristics of organic as well as anorganic objects. On low resolution,*points2mesh* is even able to outperform current simple traditional techniques as well as deep learning based ones. Most importantly, as the first venture of deep learning based mesh reconstruction from point cloud data, it holds up rather good in comparison to its direct competitor *DMC* or even the traditional approaches *BPA* and *IFM*.

7.3. Outlook

For singular object reconstruction with low-resolution input, *points2mesh* is a solid choice to produce watertight meshes. However, due to its restrictions, there are several ways to improve the network structure to augment its results even further.

First, for better generalizations, the data augmentation step could be enhanced by random rotations of the training data. Currently, most objects are aligned by looking in a specific direction with some exceptions of unaligned objects in the dataset. By applying a random rotation to the training data, a lot more training samples could be generated, which also trains *points2mesh* to be rotation invariant. The Training time would increase, though it seems a promising attempt to improve the network.

One of the most critical aspects to improve on is the problem of stretching geometry over empty space. A possible solution could be a smarter *unpooling* step in $\mathcal{N}_{gcn}$. A variable insertion of new vertices at edges where a local loss function is much higher than at its neighbors. This could lend the network the needed vertices to properly represent the part of some complicated part of geometry. However, this poses a problem for *graph convolutional layers* as they need to know the number of vertices and edges before the training even starts. Nonetheless, a possible approach is to set an upper limit of vertices for any reconstruction and keep them in reserve, where they can not be trained. When needed during an *unpooling* step, they would be added to the mesh. In that way, training time could increase immensely, though more detailed reconstructions are possible, and problems with stretching geometry over empty space would disappear. Similarly, introducing holes into the mesh with

a specialized layer could open up many new ways for the network to reconstruct better meshes, and meshes with a *genus* higher than 0.

Graph convolution is sinherently restrictive for operations on graphs representing meshes, which grow in size. Changing it on the fly poses an intricate problem to the neural network. In contrast, *flex convolution* seem a lot more flexible, as the name suggests with ever changing configurations of three dimensional coordinates. A possible attempt to simplify the problem could be to replace $\mathcal{N}_{gcn}$ with another *flex convolutional* based network.

Finally, a new tensorflow graphics library recently was released[2]. Offering specialized layers for non-rigid surface deformation, object rotations and general learning on graph structures enable new possibilities in implementing a similar idea as in this work, however with potentially fewer restrictions.

[2] *https://github.com/tensorflow/graphics* Last visited July 11, 2019

A. Appendix

Reconstructions

Figure A.1.: Reconstruction of all trained categories from point cloud with 1024 samples and N_{recon}. From left to right: ground truth, C_1, C_2, C_3, C_4.

Complete evaluations

Table A.1.: Distance metric evaluations with 256 samples per point cloud for N_{opt}. Caluclated with *meshdistance* metric. Only considering configurations C_1 and C_2. Lower values are better.

Object	C_1	C_2	BPA	*IFM*	None
airplane	0.013	0.12	0.013	0.153	0.251
bed	0.028	0.049	0.016	0.123	0.163
bottle	0.016	0.014	0.007	0.134	0.201
bowl	0.044	0.089	0.022	0.133	0.148
car	0.035	0.058	0.020	0.199	0.155
chair	0.038	0.045	0.027	0.188	0.195
guitar	0.027	0.039	0.005	0.325	0.338
toilet	0.082	0.075	0.016	0.135	0.138
mean	0.035	0.061	0.016	0.200	0.1986

Table A.2.: Distance metric evaluations with 1024 samples per point cloud for N_{opt}. Caluclated with *meshdistance* metric. Lower values are better.

Object	C_1	C_2	C_3	C_4	BPA	*IFM*	None
airplane	0.011	0.091	0.014	0.011	0.089	0.10	0.251
bed	0.016	0.012	0.018	0.016	0.008	0.099	0.163
bottle	0.010	0.007	0.010	0.008	0.003	0.10	0.201
bowl	0.025	0.03	0.029	0.022	0.006	0.097	0.148
car	0.021	0.017	0.024	0.021	0.009	0.163	0.155
chair	0.018	0.012	0.020	0.017	0.008	0.172	0.195
guitar	0.014	0.01	0.022	0.017	0.002	0.234	0.338
toilet	0.029	0.016	0.028	0.025	0.006	0.109	0.138
mean	0.018	0.024	0.021	0.018	0.016	0.134	0.1986

Table A.3.: Distance metric evaluations with 7500 samples per point cloud. Caluclated with *meshdistance* metric.Only considering configurations C_1 and C_2. Lower values are better.

Object	C_1	C_4	BPA	*IFM*
airplane	0.015	0.092	0.090	0.090
bed	0.016	0.015	0.005	0.089
bottle	0.011	0.009	0.001	0.096
bowl	0.021	0.021	0.003	0.091
car	0.019	0.019	0.003	0.153
chair	0.018	0.016	0.001	0.126
guitar	0.017	0.013	0.001	0.212
toilet	0.033	0.028	0.003	0.101
mean	0.018	0.027	0.013	0.119

Bibliography

[ACK01] Nina Amenta, Sunghee Choi, and Ravi Krishna Kolluri. The power crust. In *Proceedings of the Sixth ACM Symposium on Solid Modeling and Applications*, SMA '01, pages 249–266, New York, NY, USA, 2001. ACM.

[Ar11] Joseph Lenox Alessandro ranellucci. Slic3r. `https://github.com/slic3r/Slic3r`, 2011.

[BL18] Dennis R. Bukenberger and Hendrik P. A. Lensch. Hierarchical Quad Meshing of 3D Scanned Surfaces. *Computer Graphics Forum*, 37(5):131–141, 2018. `https://diglib.eg.org/bitstream/handle/10.1111/cgf13497/v37i5pp131-141.pdf`.

[BLRW16] André Brock, Theodore Lim, James M. Ritchie, and Nick Weston. Generative and discriminative voxel modeling with convolutional neural networks. *CoRR*, abs/1608.04236, 2016.

[BMR+99] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):349–359, Oct 1999.

[Bro86] R. Brooks. A robust layered control system for a mobile robot. *IEEE Journal on Robotics and Automation*, 2(1):14–23, March 1986.

[CFG+15] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[DCS+17a] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017.

[DCS+17b] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas A. Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. *CoRR*, abs/1702.04405, 2017.

[DN18] Angela Dai and Matthias Nießner. 3dmv: Joint 3d-multi-view prediction for 3d semantic scene segmentation. *CoRR*, abs/1803.10409, 2018.

[DRB+17] Angela Dai, Daniel Ritchie, Martin Bokeloh, Scott Reed, Jürgen Sturm, and Matthias Nießner. Scancomplete: Large-scale scene completion and semantic segmentation for 3d scans. *CoRR*, abs/1712.10215, 2017.

[FFY+18] Yutong Feng, Yifan Feng, Haoxuan You, Xibin Zhao, and Yue Gao. Meshnet: Mesh neural network for 3d shape representation. *AAAI 2019*, 2018.

[GRL17] Fabian Groh, Benjamin Resch, and Hendrik P. A. Lensch. *Multi-view Continuous Structured Light Scanning*, pages 377–388. Springer International Publishing, Cham, 2017.

[HTT+17] Binh-Son Hua, Quang-Trung Truong, Minh-Khoi Tran, Quang-Hieu Pham, Asako Kanezaki, Tang Lee, HungYueh Chiang, Winston Hsu, Bo Li, Yijuan Lu, Henry Johan, Shoki Tashiro, Masaki Aono, Minh-Triet Tran, Viet-Khoi Pham, Hai-Dang Nguyen, Vinh-Tiep Nguyen, Quang-Thang Tran, Thuyen V. Phan, Bao Truong, Minh N. Do, Anh-Duc Duong, Lap-Fai Yu, Duc Thanh Nguyen, and Sai-Kit Yeung. Shrec'17: Rgb-d to cad retrieval with objectnn dataset, 2017.

[HZ16] Vishakh Hegde and Reza Zadeh. Fusionnet: 3d object classification using multiple data representations. 07 2016.

[JTPSH15] Wenzel Jakob, Marco Tarini, Daniele Panozzo, and Olga Sorkine-Hornung. Instant field-aligned meshes. *ACM Transactions on Graphics (Proceedings of SIGGRAPH ASIA)*, 34(6), November 2015.

[KAMC17] Evangelos Kalogerakis, Melinos Averkiou, Subhransu Maji, and Siddhartha Chaudhuri. 3D shape segmentation with projective convolutional networks. In *Proc. IEEE Computer Vision and Pattern Recognition (CVPR)*, 2017.

[KBH06] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, SGP '06, pages 61–70, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[KHS10] Evangelos Kalogerakis, Aaron Hertzmann, and Karan Singh. Learning 3D Mesh Segmentation and Labeling. *ACM Transactions on Graphics*, 29(3), 2010.

[KL17] Roman Klokov and Victor S. Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. *CoRR*, abs/1704.01222, 2017.

[KMS19] Nayan M. Kakoty, Mridusmita Mazumdar, and Durlav Sonowal. Mobile

robot navigation in unknown dynamic environment inspired by human pedestrian behavior. In Chhabi Rani Panigrahi, Arun K. Pujari, Sudip Misra, Bibudhendu Pati, and Kuan-Ching Li, editors, *Progress in Advanced Computing and Intelligent Engineering*, pages 441–451, Singapore, 2019. Springer Singapore.

[KTEM18] Angjoo Kanazawa, Shubham Tulsiani, Alexei A. Efros, and Jitendra Malik. Learning category-specific mesh reconstruction from image collections. In *ECCV*, 2018.

[KW16] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

[LBSC18] Yangyan Li, Rui Bu, Mingchao Sun, and Baoquan Chen. Pointcnn. *CoRR*, abs/1801.07791, 2018.

[LC87] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, August 1987.

[LD18] Truc Le and Ye Duan. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[LDG18] Yiyi Liao, Simon Donné, and Andreas Geiger. Deep marching cubes: Learning explicit surface representations. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[LLZ+17] Fangyu Liu, Shuaipeng Li, Liqiang Zhang, Chenghu Zhou, Rongtian Ye, Wang Yuebin, and Jiwen Lu. 3dcnn-dqn-rnn: A deep reinforcement learning framework for semantic parsing of large-scale 3d point clouds. 07 2017.

[LPT13] Joseph J. Lim, Hamed Pirsiavash, and Antonio Torralba. Parsing IKEA Objects: Fine Pose Estimation. *ICCV*, 2013.

[MS15] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *Ieee/rsj International Conference on Intelligent Robots and Systems*, pages 922–928, 2015.

[NSF12] Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

[PKS+17] Jhony K. Pontes, Chen Kong, Sridha Sridharan, Simon Lucey, Anders P. Eriksson, and Clinton Fookes. Image2mesh: A learning framework for single image 3d reconstruction. *CoRR*, abs/1711.10669, 2017.

[QSMG16] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593, 2016.

[QSN+16] Charles Ruizhongtai Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J. Guibas. Volumetric and multi-view cnns for object classification on 3d data. *CoRR*, abs/1604.03265, 2016.

[QYSG17] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*, 2017.

[SFH18] Daeyun Shin, Charless C. Fowlkes, and Derek Hoiem. Pixels, voxels, and views: A study of shape representations for single view 3d object shape prediction. *CoRR*, abs/1804.06032, 2018.

[SG18] David Stutz and Andreas Geiger. Learning 3d shape completion from laser scan data with weak supervision. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, 2018.

[SMKF04] Philip Shilane, Patrick Min, Michael Kazhdan, and Thomas Funkhouser. The Princeton shape benchmark. In *Shape Modeling International*, June 2004.

[SS15] J.Xiao S. Song, S. Lichtenberg. SUN RGB-D: A RGB-D Scene Understanding Benchmark Suite. *CVPR2015*, 2015.

[Stu17] David Stutz. Learning shape completion from bounding boxes with cad shape priors. http://davidstutz.de/, September 2017.

[SvKK+11] Oana Sidi, Oliver van Kaick, Yanir Kleiman, Hao Zhang, and Daniel Cohen-Or. Unsupervised co-segmentation of a set of shapes via descriptor-space spectral clustering. In *Proceedings of the 2011 SIGGRAPH Asia Conference*, SA '11, pages 126:1–126:10, New York, NY, USA, 2011. ACM.

[SZAB17] N. Sedaghat, M. Zolfaghari, E. Amiri, and T. Brox. Orientation-boosted voxel nets for 3d object recognition. In *British Machine Vision Conference (BMVC)*, 2017.

[TCA+17] Lyne P. Tchapmi, Christopher B. Choy, Iro Armeni, JunYoung Gwak, and Silvio Savarese. Segcloud: Semantic segmentation of 3d point clouds. *CoRR*, abs/1710.07563, 2017.

[TEM18] Shubham Tulsiani, Alexei A. Efros, and Jitendra Malik. Multi-view consistency as supervisory signal for learning shape and pose prediction. In *Computer Vision and Pattern Regognition (CVPR)*, 2018.

[Vie14] Tim Vieira. Gumbel-max trick and weighted reservoir sampling, 2014.

[WSK+15] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. pages 1912–1920, 06 2015.

[WZL+18] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images. In *ECCV*, 2018.

[XKC+16] Yu Xiang, Wonhui Kim, Wei Chen, Jingwei Ji, Christopher Choy, Hao Su, Roozbeh Mottaghi, Leonidas Guibas, and Silvio Savarese. Objectnet3d: A large scale database for 3d object recognition. In *European Conference Computer Vision (ECCV)*, 2016.

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

[ZSK+15] Zhirong Wu, S. Song, A. Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1912–1920, June 2015.