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 14, 2019

Abstract

points2mesh, the happening

Auszug

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

1. robotics, interact with real world
2. understand env
3. what is it, they are working with
4. autonomous driving
5. thus object analysis
6. digitize objects somehow
7. round about of different kinds already present
8. lots of data readily available
9. present novel network configurations deep learning based
10. end to end solution for shape inference from PC to mesh
11. some modified loss functions for good results
12. determine how well it is doing this functions
13. compare to work doing similar things
14. non learning based/ learning based
15. maybe small user study??

1.1. Problem Statement

mathmatic formulation

1.2. chapter recapitulation

2. Background

1. Convolutional neural network
2. Semi-Supervised Classification with Graph Convolutional networks

-nearest neighbor search

-whats a graph

-whats a mesh

2.1. Machine Learning review

TODO: different name for section - Convolutional neural networks - graph Convolution -

2.2. stuff

2.2.1. Weighted reservoir sampling

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. Methods

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, section 4.2 specifies the dataset used to train the networks, as well as how it was designed and constructed.

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 4.1.2 and 4.1.3 , Subsequentially, a closer look at utilized objective functions in subsections 4.1.4 will be taken. Finally, every considered configuration of the neural network is outlined in subsection 4.1.5.

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 $\mathbf{n}_{x_i}$.

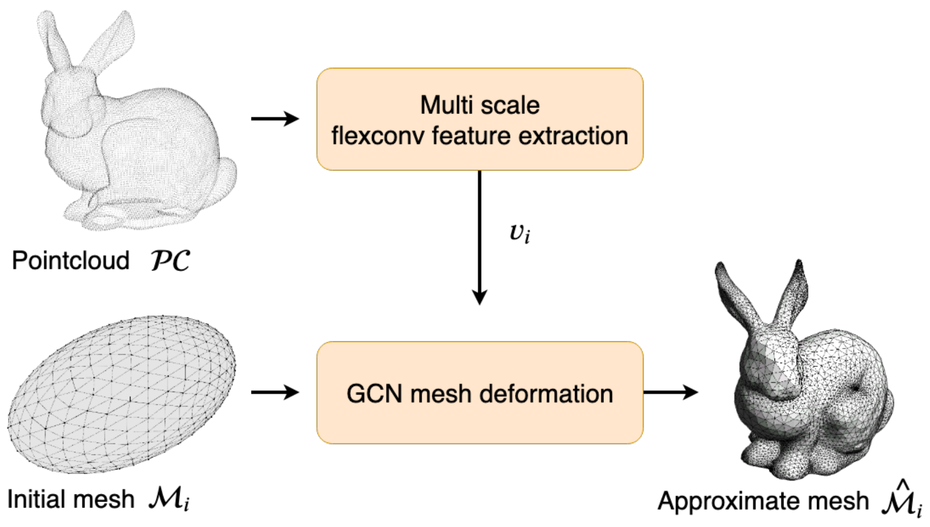
Multi scale
flexconv feature extraction
v_i
Pointcloud $\mathcal{PC}$
GCN mesh deformation
Initial mesh $\mathcal{M}_i$
Approximate mesh $\hat{\mathcal{M}}_i$

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.

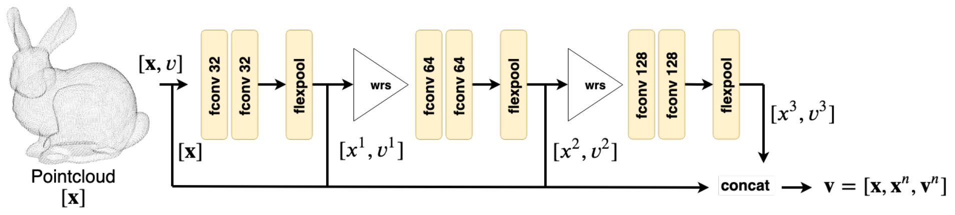

Figure 4.2.: $\mathcal{N}_{flex}$ with point cloud $\mathcal{PC}$ as input, computing features v^j in three blocks of flexconv layers and a trailing flexpool layer After each convolutional block, weighted random sampling reduces the number of samples of $\mathcal{PC}$ by a factor of four. Feature vectors v^j are concatenated with their respective positions x^j in a vector **v**.

4.1.2. 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 4.2, $\mathcal{N}_{flex}$ processes $\mathcal{PC}$ to compute a collection of three feature vectors v_{x_n} on a multiscale measure, working as follows. The network $\mathcal{N}_{flex}$ 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.

4.1.3. 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

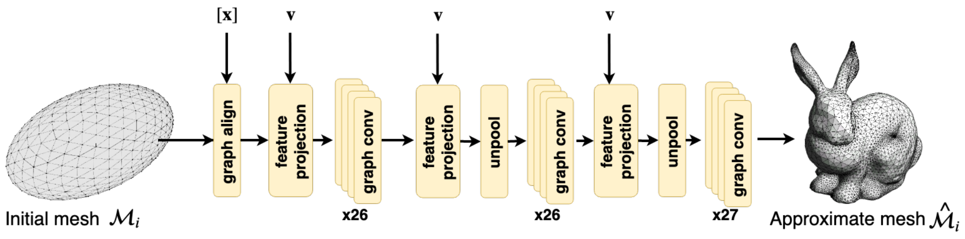

Figure 4.3.: $\mathcal{N}_{gcn}$ with initial mesh $\mathcal{M}_i$ as input. Feature vector $\mathbf{v}$ is projected onto the mesh $\mathcal{M}_i$ and deformed according to it. Then an unpooling layer increases the number of vertices in $\mathcal{M}_i$. Projecting $\mathbf{v}$ onto the graph and deforming it afterward is repeated two more times.

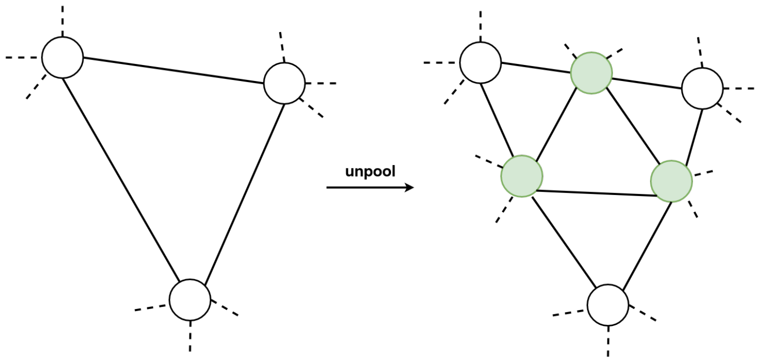

Figure 4.4.: The *unpooling* layer of $\mathcal{N}_{gcn}$ increases the number of vertices by introducing a new node between each neighboring node (white) in $\mathcal{M}_i$. A new node (green) is added precisely in the middle of each old edge. Finally, the new nodes are connected, while also updating the neighborhood of the old nodes.

set of edges, connecting V. Furthermore, similarly to 4.1.2, 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 4.3. 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 $\hat{\mathcal{M}}$, 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

Figure 4.5.: 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.

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 4.4. 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 4.2), it is essential to compensate for the positional bias of the $\mathcal{PC}$ in relation to $\mathcal{M}_i$ as seen in figure 4.5. $\mathcal{N}_{gcn}$ 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 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

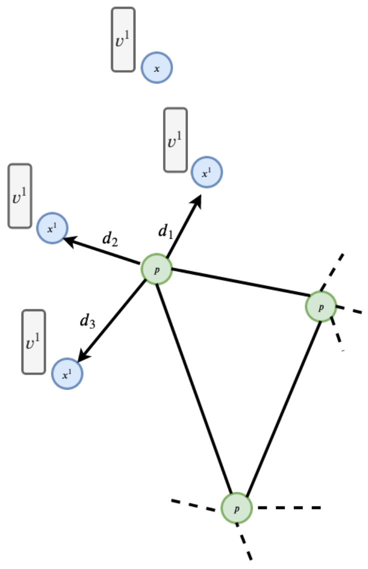

Figure 4.6.: *graph projection* layer, assigning feature vectors $\mathbf{v}^j$ to nodes in $\mathcal{M}_i$ based on the distance to their nearest neighbor in $\mathcal{PC}$.

$\mathcal{PC}$ 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 4.6, 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 $[\mathbf{v}^s_n, \cdots, \mathbf{v}^s_{n+k}]$ of points $[\mathbf{x}^s_n, \cdots, \mathbf{x}^s_{n+k}]$ 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:

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

Where v_i is the respective feature vector stored at point $\mathbf{x}^s_i$. 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 4.2).

4.1.4. 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 $\mathcal{PC}$.

4.1.5. Network configurations C

Finding suitable hyperparameters and an appropriate network structure is no easy task. Tuning hyperparameters in small increments, removing or including

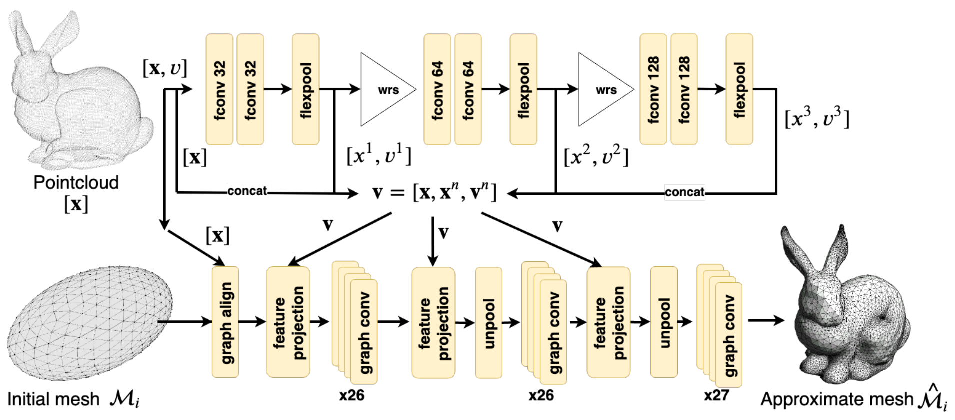

Figure 4.7.: Baseconfiguration C_1 of $\mathcal{N}_{recon}$.

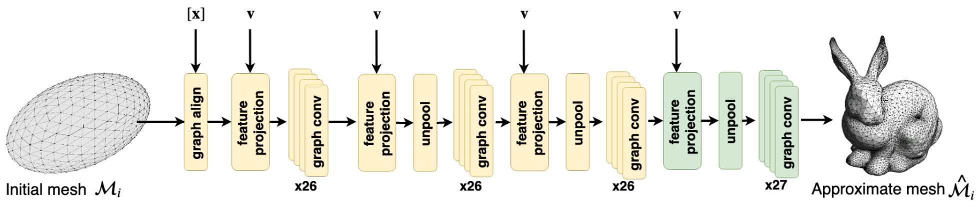

Figure 4.8.: Configuration C_2 of $\mathcal{N}_{recon}$, showing its difference in $\mathcal{N}_{gcn}$ to configuration C_1. C_2 introduces an extra *projection*, *unpooling* layer and a *graph convolution* block. Thus, allowing for a higher detailed approximate mesh $\hat{\mathcal{M}}$

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 4.1.3. 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 4.7, 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 4.8 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 $\mathcal{PC}$.

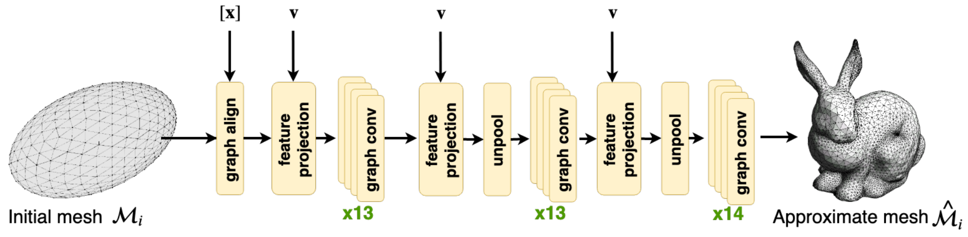

Figure 4.9.: Showing only $\mathcal{N}_{gcn}$ of $\mathcal{N}_{recon}$ with $\mathcal{N}_{flex}$ being the same as in configuration C_{1-3}. In contrast to C_1, Configuration C_4 reduces the number of used layers for deforming the mesh in $\mathcal{N}_{gcn}$.

C_3: Biased neighbor features

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, configuration C_4 reduces the amount of convolutional layers in $\mathcal{N}_{gcn}$ as seen in figure 4.9. 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.

4.2. 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 transformation of the classical *ModelNet40* dataset to a point cloud based one which can be utilized by $\mathcal{N}_{recon}$.

This section describes their way of transforming the dataset, on which categories $\mathcal{N}_{recon}$ is trained on, and an additional data augmentation step to enhance the dataset.

4.2.1. Dataset generation

4.2.2. Utilized object categories

TODO: Bilder fuer object categories or interesting features Even though *ModelNet40*, and thus 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 four subsets of $\mathcal{D}_{m40}$ are defined as follows:

- *Big*:[airplane,bed,bottle,bowl,chair,guitar,sofa,toilet]
- *Small*:[airplane,chair,guitar,toilet]
- $single_1$:[airplane]
- $single_2$:[toilet]

Big 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. *Small* serves as a verification for *Big*, by checking the difference between reconstructed objects of categories contained in both. Comparing these objects while also considering objects reconstructed with *Big* may lead to information about how well $\mathcal{N}_{recon}$ is able to generalize. Additionally $single_1$ and $single_2$ serve as a verification dataset, showing how well $\mathcal{N}_{recon}$ is able to learn specific categories of objects with interesting and hard to reconstruct features.

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

5. Results

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 5.1, 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 experimental setup are presented in section 5.2, where each experiment is accompanied by renderings of some of the reconstructed meshes.

5.1. 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.

5.1.1. Evaluation aspects

There are several aspects to evaluate *points2mesh*. Not only are network configurations C_i a crucial consideration, but so is training time, 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 4.1.5, 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.
- **Visual quality**: While being subjective, taking a closer look how good the reconstruction seems to represent the ground truth mesh is still relevant.
- **Computation time**: Determining how fast such a reconstruction can be calculated may change which method to use depending on the application.
- **Training time**: Measured in iterations, the convergence time offers valuable insight for networkconfigurations C_i and how it learns from the input data.
- **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*.

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

5.1.2. 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 and accuracy of the predicted mesh in comparison to the ground truth mesh. Based on uniformly sampled points on both meshes, completeness and accuracy can be defined. While the completeness describes the distance of points of ground truth to the predicted mesh, accuracy describes the distance of points from predicted mesh to ground truth. Stutz et al. [Stu17, SG18] introduce a parallel $C++$ implementation of *meshdistance* which calculates accuracy and completeness, given a ground truth mesh and a prediction. Distances are calculated by a point-to-triangle method by Christopher Batty[1] (See chapter 2.1)

Chamfer distance d_C

Similarly to the *meshdistance* method, to calculate the *chamfer distance*, both meshes have to be uniformly sampled. In contrast, only the distance to each nearest neighbor from ground truth to reconstruction, as well as from reconstruction to ground truth, is considered for the metric. While the *chamfer distance* is a more simplified version of the meshdistance, it is still able to provide valuable information about the quality of reconstructions.

Hausdorff distance d_H

The *hausdorff distance* describes a more sophisticated, asymmetric metric for measuring distances between two similar meshes. Anew, the ground truth mesh is sampled with the same amount of samples as the ground truth mesh possesses. For each sample, its closest point on the predicted mesh is calculated, resulting in minimum, maximum values, as well as mean values for the *hausdorff distance* metric.

$$d_H(\mathcal{M}_{gt}, \mathcal{M}_i) = \underset{y \in \mathcal{M}_i}{sup} \underset{y \in \mathcal{M}_{gt}}{inf} \; d(x, y) \tag{5.1}$$

With $d(x, y)$ describing a function, calculating the distance from x to y.

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

[1]https://github.com/christopherbatty/SDFGen last visited: July 14, 2019

indefinite amount of different reconstructions are possible. Therefore, inspecting the meshes is necessary.

5.1.3. 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] (DeepMC), which transforms voxelized data into meshes based on deep learning approaches and the classical marching cubes algorithm.

- **TODO: include pixel2mesh if possible**

5.2. Experiment results

TODO: Bei jeder subsubsection auch visual quality mit zeigen The evaluation of *points2mesh* and its configuration C_i is separated into a numerical and visual evaluation. In the following subsection 5.2.1, the previously explained metrics are listed and compiled against each other. Likewise, *points2mesh*'s reconstructions are listed to comparable methods with the same metrics.

5.2.1. Numerical evaluation

The following subsections present the evaluation of the previously defined metrics. 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 three methods against which *points2mesh* is compared against, should be able to handle point cloud data from ModelNet.

Distance metric results

The first comparison consists of evaluating all distance metrics $[d_m, d_C, d_H]$ for every configuration C_i and *deep marching cubes*, *ball-pivoting algorithm* and *instant field meshes*.

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 1024 samples, the following denotes reconstruction from 7500 points.

Table 5.1.: Distance metric evaluations with 1024 samples per point cloud

Object category	metric	C_1	C_2	C_3	C_4	DMC	BPA	IFM
airplane	d_m	b	c	c	c	c	c	
	d_C	b	c	c	c	c	c	
	d_H	b	c	c	c	c	c	
bed	d_m	b	c	c	c	c	c	
	d_C	b	c	c	c	c	c	
	d_H	b	c	c	c	c	c	
bottle	d_m	b	c	c	c	c	c	
	d_C	b	c	c	c	c	c	
	d_H	b	c	c	c	c	c	
bowl	d_m	b	c	c	c	c	c	
	d_C	b	c	c	c	c	c	
	d_H	b	c	c	c	c	c	
car	d_m	b	c	c	c	c	c	
	d_C	b	c	c	c	c	c	
	d_H	b	c	c	c	c	c	
chair	d_m	b	c	c	c	c	c	
	d_C	b	c	c	c	c	c	
	d_H	b	c	c	c	c	c	
sofa	d_m	b	c	c	c	c	c	
	d_C	b	c	c	c	c	c	
	d_H	b	c	c	c	c	c	
guitar	d_m	b	c	c	c	c	c	
	d_C	b	c	c	c	c	c	
	d_H	b	c	c	c	c	c	
toilet	d_m	b	c	c	c	c	c	
	d_C	b	c	c	c	c	c	
	d_H	b	c	c	c	c	c	
mean metrics	$\bar{d}_m$	b	c	c	c	c	c	
	$\bar{d}_C$	b	c	c	c	c	c	
	$\bar{d}_H$	b	c	c	c	c	c	

Table 5.2.: Distance metric evaluations with 7500 samples per point cloud

Object category	metric	C_1	C_2	C_3	C_4	DMC	BPA	IFM
	d_m	b	c	c	c	c	c	
airplane	d_C	b	c	c	c	c	c	
	d_H	b	c	c	c	c	c	
	d_m	b	c	c	c	c	c	
bed	d_C	b	c	c	c	c	c	
	d_H	b	c	c	c	c	c	
	d_m	b	c	c	c	c	c	
bottle	d_C	b	c	c	c	c	c	
	d_H	b	c	c	c	c	c	
	d_m	b	c	c	c	c	c	
bowl	d_C	b	c	c	c	c	c	
	d_H	b	c	c	c	c	c	
	d_m	b	c	c	c	c	c	
car	d_C	b	c	c	c	c	c	
	d_H	b	c	c	c	c	c	
	d_m	b	c	c	c	c	c	
chair	d_C	b	c	c	c	c	c	
	d_H	b	c	c	c	c	c	
	d_m	b	c	c	c	c	c	
sofa	d_C	b	c	c	c	c	c	
	d_H	b	c	c	c	c	c	
	d_m	b	c	c	c	c	c	
guitar	d_C	b	c	c	c	c	c	
	d_H	b	c	c	c	c	c	
	d_m	b	c	c	c	c	c	
toilet	d_C	b	c	c	c	c	c	
	d_H	b	c	c	c	c	c	
	$\bar{d}_m$	b	c	c	c	c	c	
mean metrics	$\bar{d}_C$	b	c	c	c	c	c	
	$\bar{d}_H$	b	c	c	c	c	c	

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 4.2.2). 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 5.3.: Distance metric evaluations with 1024 samples per point cloud. Testing with point clouds from outside of trained object categories

Object category	metric	C_1^{big}	C_1^{small}	$C_1^{airplane}$	C_1^{toilet}
bathtub	d_m	b	c	c	
	d_C	b	c	c	
	d_H	b	c	c	
person	d_m	b	c	c	
	d_C	b	c	c	
	d_H	b	c	c	
mean metrics	$\bar{d_m}$	b	c	c	
	$\bar{d_C}$	b	c	c	
	$\bar{d_H}$	b	c	c	

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 5.4.: Distance metric evaluations with 1024 samples per point cloud. Initial mesh (torus mesh) with a *genus* of one.

Object category	metric	C_1
chair	d_m	b
	d_C	b
	d_H	b

Noise robustness

Training and evaluation time

TODO: Den teil hier, falls zeit :) For real time applications like autonomous cars, which rely on ...

6. Discussion

ergebnisse objektiv durchgehen Ergebnisse subjektiver betrachten zu anderen paper vergleichen torus schlecht gibt es einen vorteil gegenueber den andern techniken, die im vergleich aufgetaucht sind?

genus problem

was anders machen wie erweitern

konzept an sich kritisieren

7. Outlook

1. loecher einfuegen durch neue layer
2. mehr detail beliebig
3.

A. Appendix

formeln aus methoden

mehr bilder von daten

sonst zeug?

mehr ergbnis bilder von recon

detailed structure from C1 to C4

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.

[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.

[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.

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

[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.

[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.