FH Salzburg
MultiMediaTechnology

Sketch to 3d-Model using Deep Learning and Differentiable Rendering

Master thesis in partial fulfilment of the requirements for the degree of

Master of Science

Author: Kerstin Hofer, BSc.

Submitted to the Master degree program MultiMediaTechnology, Salzburg University of Applied Sciences

Advisor: DI Philipp Erler
Second Advisor: DI Dr. Clemens Havas, BSc.

Salzburg, Austria, 03.03.2023

Affidavit

I herewith declare on oath that I wrote the present thesis without the help of third persons and without using any other sources and means listed herein; I further declare that I observed the guidelines for scientific work in the quotation of all unprinted sources, printed literature and phrases and concepts taken either word for word or according to meaning from the Internet and that I referenced all sources accordingly.

This thesis has not been submitted as an exam paper of identical or similar form, either in Austria or abroad and corresponds to the paper graded by the assessors.

Date *Signature*

First Name *Last Name*

Kurzfassung

3D Modelle aus 2D Sketches zu generieren ist seit einigen Jahren ein relevantes Thema, da es die Möglichkeit bietet, den Prozess der Ideenrealisierung einiger Berufsgruppen wie Künstler, Techniker und Designer zu vereinfachen. Mit der wachsenden Bedeutung Neuronaler Netzwerke wurden neue Rekonstruktionsmethoden entwickelt, die sich diese Lernfähigkeit zu Nutze machen. Einige dieser Methoden setzten Differentiable Rendering ein. Hierbei wird durch Deformierung eines Basisobjekts der Unterschied zwischen dessen Differentiable Renderings und Vergleichsrenderings minimiert. Da diese Vergleichsbilder, wie z.B. Normal- oder Depthmaps, dem System inhärent nicht verfügbar sind, müssen diese anhand der gegebenen Informationen, in diesen Fall der Input-Sketches, ermittelt werden.

Das Ziel dieser Arbeit ist es ein End-To-End-Framework vorzustellen, welches vorberechnete auf 2D Bildern 3D Modelle rekonstruiert. Hierzu werden generierte Normal- und Depthmaps als Vergleichsbilder sowie ein Basisobjekt basierend auf der Topologie des Eingabebildes in der Deformierung verwendet. Durch einen Vergleich der generierten Modelle dieser Pipeline mit den Resultaten einer etablierten Methode und einer Ablation Studie wird die Wirkung der einzelnen Verbesserungen und die Grenzen des Frameworks evaluiert. Dadurch wird ermittelt ob die eingebrachten Optimierungen die Qulität der resultierenden Meshes verbessern und allgemeinere Objekte unabhängig von Klasse/Typ rekonstruieren können.

Schlüsselwörter: mehrschichtiges Lernen, Modell Generierung mittels Einzelbildes, Topologie, differenzierbares Rendering, 2.5D Zwischendarstellung

Abstract

Deriving 3D models from sketches has been a relevant research topic for years now, mainly due to its potential to simplify the realization of ideas and drafts for certain professions like artists, engineers and designers. With the raise of neural networks, new methods start to form utilizing the learning capability of those networks. One such method is the application of differentiable rendering, where a base mesh is deformed until the difference between the differentiable renderings and the target image is reasonably small. However, those target images, e.g., normal or depth maps, are not inherently available to the system, since the only input is a sketch. Therefore, intermediate representations predicted from the sketch need to be generated, that provide the required information.

The aim of this thesis is to provide a an end-to-end framework that reconstructs 3d models from 2d sketches using intermediate depth and normal representations and differentiable rendering. In addition to that, the topology of the sketch is determined and a respective base mesh is used for the mesh deformation. By comparing this setup to an established network as well as conducting an ablation study to reveal the effects the improvements have, the system and its limitations are evaluated. This will determine if the proposed optimisations facilitate input generalisability beyond class/object restrictions and improve mesh quality.

Keywords: deep learning, single-view model generation, topology, differentiable rendering, 2.5d intermediate representation

Contents

List of Figures

Listings

List of Tables

Abbreviations

AOV	Arbitrary Output Variables
cGAN	Conditional Generative Adversarial Network
CNN	Convolutional Neural Network
CUDA	NVIDIA CUDA
DR	Differentiable Rendering
GPU	Graphics Processing Unit
GT	Ground Truth
ICP	Iterative Closest Point
IoU	Intersection over Union
JSON	JavaScript Object Notation
OBJ	Wavefront OBJ
PDE	Partial Differential Equation
PLY	Stanford Triangle Format
PNG	Portable Network Graphics
R-CNN	Region Based Convolutional Neural Networks
ReLU	Rectified Linear Unit
RGB	Red Green Blue
RGBA	Red Green Blue Alpha
SDF	Signed Distance Function
UV	UV coordinate
WGAN	Wasserstein Generative Adversarial Network

1 Introduction

Deriving 3d models from 2d images has been a relevant research topic for years now since there is great potential to simplify the model creation processes of certain professions like artists, engineers, and designers. Following the progress made in researching deep neural networks, recently the usage of deep learning in order to generate a 3d model from a 2d image was widely researched.
But the main problem still remains: How can a computer interpret a 2d image as a 3d object? Many methods have been developed for different types of 2d images, with the majority focusing on greyscale or RGB images, like state-of-the-art models such as SoftRas (Liu et al. 2019) or MarrNet (Wu et al. 2017). However, for images that provide less information, like sketches, those methods are not applicable. This is due to the fact that sketches are binary RGB images representing a simplified version of a given scene using only lines. Those sparse line drawings lack any information like colours that indicate which regions belong together or shading, which would be an implication for depth and therefore supply valuable information about the size and proportion of the depicted object. Furthermore, those sketches are often riddled with incomplete and false lines or additional orientation lines which do not actually belong to the object itself, making the interpretation of these inputs even more complex (Xiang et al. 2020, 1). On the other hand, this style is more accessible and less time-consuming to create, leading to a quicker realization of an idea or at least a draft of that. This makes it a valuable medium in a lot of fields and creating models from sketches is applicable in many areas.

Since there is no solution on how to get more data out of a sketch without modifying the input itself by adding more information like colour, which results in extra work for the creator of the sketch, research over the past years has tried different solutions in order to provide additional data which can be used to interpret the 2d drawing as a 3d model:
Traditional research approaches use different techniques employing extra rules, like line labelling rules in order to classify 2d information and to be able to interpret this as 3d information, e.g. depth or normal, as proposed in the work by Malik 1987. Another prominent concept is to use cross-sections of clean curves, which compose the sketch, as the source of the geometric information needed to generate a model, like Shao et al. 2012 did in their research. Those approaches have the disadvantage of being restricted by very specific sketches they can interpret, e.g. the approach by Shao et al. 2012 needs a sketch consisting of clean curves in order to work as expected, and as mentioned before, are in need of a set of additional rules in order to produce the desired output (Xiang et al. 2020, 3).

The model generation became less restrictive with the progress in researching neural networks. By using multiple different perspectives representing the sketched object in order to provide more data to the reconstruction network, like the research by Lun et al. 2017 shows, using additional rules is redundant. Also, the object classes, which can be recovered, are not as restricted, since the information used for the reconstruction can mostly be recovered directly from the input, as seen in Lun et al. 2017, 69. However, those have the disadvantage of requiring multiple sketches as input, which is not always guaranteed to be provided.

To compensate for this disadvantage approaches using only a single input sketch instead of multiple have also been investigated, e.g. by Smirnov, Bessmeltsev, and Solomon 2021 and Delanoy et al. 2018, which, although relying on multiple images within the learning process, can also be used with single sketches as input. There are techniques that do not use the representation of the object from multiple viewpoints in either their input or as an intermediate representation, like methods using differentiable rendering in the learning process, as for example, Kato, Ushiku, and Harada 2018, Xiang et al. 2020 and Liu et al. 2019 did in their research. Differentiable rendering is used in the deformation process of a given mesh in order to compute the relevant information needed to minimise the difference between the characteristics like normals of the deformed object and the desired object. However, like the other approaches presented above, one of the core problems of this technique is how to predict information, which can then be used for the differential rendering process. A solution for that is to use 2.5d representation, like Xiang et al. 2020 did by using a normal map predicted from the given sketch, which works fairly well for their use case. But as Smirnov, Bessmeltsev, and Solomon 2021 states, normal maps are not always accurate and can differ from other representations of the model, e.g. depth maps. Furthermore, relying on information like normals or silhouettes introduces the problem of depth ambiguity. To avoid problems caused by that, strong prior knowledge of the object classes that are reconstructed is required, which means that the datasets used for learning need to be uniform and heavily class-based, and therefore makes the methods themselves vastly restrictive.
Another problem some techniques like the one by Xiang et al. 2020 and Kato, Ushiku, and Harada 2018 have is that they rely on one base shape, which then is deformed by using input, in this case, from the normal map to achieve the anticipated result. In most cases, this is either a sphere or an ellipse, since those are a relatively good basis for simple models with genus 0. But if the topology model is more complex, those models tend to produce flawed results (Xiang et al. 2020, 9). This was improved by using shapes more similar to the desired model, essentially a simplified version of the model, as done e.g. by Smirnov, Bessmeltsev, and Solomon 2021. However, the added input needed for each shape requires additional work by the user or additional simplification operations on the ground-truth model. Other approaches tried to shape the deformed sphere afterwards by removing excessive faces and refining steps like Ben Charrada et al. 2022 or the mesh processing tool by Chen et al. 2020, which refines meshes produced by Marrnet (Wu et al. 2017).

The aim of this thesis is to further improve the generation of a 3d model from a single sketch. The thesis results in an end-to-end learning framework, which recovers a 3d model from a 2d sketch. The input sketches used are RGB raster images, representing the outline and details of an object using lines. Technical sketches are not considered. Furthermore, the sketch is required to have closed lines and no noise, since existing algorithms for line closing and sketch cleanup could be used in a pre-processing step before applying the reconstruction framework. However, implementing such an algorithm within this work would go beyond the scope of this thesis.
This pipeline can be divided into three modules:

- Base mesh determination: The topology is determined using an image segmentation method, isolating the object from the background of the input sketch. The resulting im-

age is used as a target image for the silhouette in the deformation module as well as to determine the number of holes in this module. Using this number, a base mesh is chosen. This is essentially a more generalised approach of Smirnov, Bessmeltsev, and Solomon 2021 since a general mesh is used for each genus type rather than the simplified version of the ground-truth model. Therefore, this model is able to reconstruct meshes without requiring a specific ground-truth model or additional user input in form of a base mesh.

- Normal and depth map reconstruction: Similar to Xiang et al. 2020, the normal resp. depth map reconstruction will be conducted by a trained image-to-image translation network. The network structure is inspired by Su et al. 2018 and Isola et al. 2017. The training process is conducted by using created datasets consisting of rendered sketches and normal resp. depth maps.

- Mesh deformation: For the mesh deformation, a differentiable renderer is used. The mesh obtained in the base mesh determination step is used as the base mesh for the deformation. The losses supervising the reconstruction are the depth and the normal loss, for which the maps of the second step are used as target images, the silhouette loss, for which the filled sketch from the base mesh determination module is used, as well as two regularisers.

The main contributions of this network are the combination of the normal and depth map in a mesh reconstruction network using a differentiable renderer, which was not done in this form before as far as the author is aware of. This is done to improve model accuracy since two losses instead of only the normal loss provide more information to the reconstruction process and remove the problem of depth ambiguities. Furthermore, using a base mesh with the same topology as the desired object is enough to be used on a variety of objects with the same genus. This implies that more objects can be accurately predicted compared to similar methods with more restrictive templates like Smirnov, Bessmeltsev, and Solomon 2021. The resulting framework is expected to provide a less restrictive approach for single-view image reconstruction compared to prior work.
The effects of the improvements are evaluated by comparing the pipeline to a state-of-the-art method by Kato, Ushiku, and Harada 2018 in both a quantitative way using distance metrics and a qualitative way. Furthermore, an ablation study where different versions test the absence of depth loss and the topology-specific base mesh is conducted. The results of these studies will showcase the benefits of the proposed method as well as the limitations and possible improvements that can further advance the system. In addition to that, the insights gained from the evaluations will answer whether the addition of the depth map and the base mesh determination improve the state-of-the-art methods in terms of mesh quality and usage of input meshes beyond class/object restrictions.

2 Related Work

The aim of this thesis is to estimate a 3d model from a drawn sketch, which is based on various prior research in the field of computer vision. This includes work regarding GANs and their

Method	Input					2.5d Intermediate Representation	Differentiable Rendering	considering Topology
	Sketch	Greyscale	RGB	RGBA	other			
Implicit Function based								
Schmidt et al. 2006	x							
Xu et al. 2019				x				
Chen and Zhang 2019		x						
Voxels								
Choy et al. 2016			x					
Yan et al. 2016*			x				x	
Tatarchenko, Dosovitskiy, and Brox 2017*			x			x		
Wu et al. 2017			x			x	x	
Delanoy et al. 2018	x							
Wu et al. 2018**			x		x	x		
Point Cloud								
Fan, Su, and Guibas 2017			x					
J. Wang et al. 2020	x							
Gao et al. 2022	x					x		
Mesh								
Lun et al. 2017	x					x		
Kato, Ushiku, and Harada 2018*				x			x	
N. Wang et al. 2018			x					
Liu et al. 2019* ***			x		x		x	
Smirnov, Bessmeltsev, and Solomon 2021	x						x	x
W. Wang et al. 2019****					x			
Chen et al. 2020****					x			x
Xiang et al. 2020	x					x	x	
Guillard et al. 2021	x					x	x	
Ben Charrada et al. 2022			x					x

Table 1: General overview of the discussed models. The models are grouped by their output type. The used input type and whether they use 2.5d representations and differentiable rendering in their reconstruction process are depicted. Furthermore, the consideration of the topology is charted. However, this is only relevant for reconstruction methods using triangle meshes.
* No clear input specification in the paper, the given input medium is based on images and the general description of the method as well as the implementation if existing.
** depth map as input possible
*** silhouette map as input possible
**** mesh as input required

WGAN variant, image and map prediction techniques, DR techniques as well as single-view image reconstruction and image segmentation.

2.1 Single-View Deep Learning Techniques 2d Image to 3d Model

Although deriving a 3d model from a 2d image has been a research topic for many years now, especially in recent years following the progress in the development of neural networks there has been a growing interest in this research field (Xiang et al. 2020, 1). While many research models like Lun et al. 2017 and Delanoy et al. 2018 use multiple viewpoint representations of the image as input, single-view techniques have become more relevant since different angled images from a model cannot always be provided and the quality of the models recreated by

those techniques is on a par with the results from the multi-view techniques. Additionally, the research approaches vary in their form of input, like using an RGB image, a sketch or additional information like viewpoint information or depth maps. Some approaches also use prediction networks, that start with an input variant offering little information like a sketch and predict additional information in form of normal maps, as done by (Xiang et al. 2020, 1), or normal and depth maps, as seen in Lun et al. 2017. In recent work, the topology of the objects was also considered in the reconstruction part either by removing parts of the deformed mesh to match the genus, e.g. done by Ben Charrada et al. 2022, or by using respective base meshes in the deformation process, as seen in Smirnov, Bessmeltsev, and Solomon 2021. The resulting models can either be represented by implicit functions, voxels, point clouds or triangle meshes (Chen et al. 2020, 1090). A general overview of the discussed models is given in Table 1.

2.1.1 Implicit Function based Techniques

Representing the reconstruction of an image as an implicit function in the form of e.g. implicit surfaces or volumes has been used for years, like in Schmidt et al. 2006. With the utilisation of neural networks to retrieve 3d models from 2d images research has also used this form to describe the resulting models, like Xu et al. 2019 and Chen and Zhang 2019 did. The latter for example employs an implicit field, which is defined by a continuous function over the 2d/3d space that assigns a binary value to each point in the 3d space to determine, whether that point is in the shape or not Chen and Zhang 2019, 5932.
Although methods outputting such structures might have several advantages like not being restricted regarding the model's resolution, they generally are not that well suited for depicting sharp or hard-edged features and details (Xu et al. 2019, 490f), as seen in Chen and Zhang 2019, 5939. While this problem was tried to be eliminated by research like Xu et al. 2019, who added an additional feature extractor by estimating the viewpoint of the image, encoding implicit fields in neural networks is more complex and the implicit function has to be converted into another representation form if it should further be used by humans. This can be done in various ways depending on the method given and the desired output representation. For example, the conversion is done by the methods mentioned above by requiring a point cloud as additional input (Xu et al. 2019, 497) or using a method like marching cubes (e.g. Lewiner et al. 2003) in order to retrieve a mesh (Chen and Zhang 2019, 5935). Using these can lead to quality loss and if the overall representation should be a mesh or voxel, these methods add additional computational costs (Xu et al. 2019, 491) (Chen and Zhang 2019, 5939).

2.1.2 Voxel based Techniques

Many research projects in the field of neural networks and 3d model reconstruction use voxels as their geometric representation. Those methods exploit the progress made in 2d image processing by replacing pixel grids with voxel grids and 2d convolutions with 3d convolutions (Ben Charrada et al. 2022, 337).
Early examples utilising voxels as a model representation include Delanoy et al. 2018, Choy et al. 2016 and Yan et al. 2016. Those methods can be used as single input methods or extended

to use multiple views to create the requested model. Delanoy et al. 2018 uses the additional input sketches in order to refine the model recreated from a single input sketch, Choy et al. 2016 uses one or more images of various viewpoints to reconstruct a 3d model and Yan et al. 2016 employs projection loss in order to backpropagate the feedback of other views than the viewpoint of the given input image. This is accomplished by deriving various 2d silhouettes of the predicted 3d model from various viewpoints using differentiable rendering. The silhouettes are then compared to ground-truth silhouettes resulting in the loss backpropagated to the mesh generator (Yan et al. 2016, 1698). Although these methods yield reasonable results, both regarding quality and performance, they can have problems when it comes to recreating thin structures (Delanoy et al. 2018, 10) and as mentioned above, there might be additional input required in order to achieve the best result possible (Choy et al. 2016, 640).
While the methods mentioned use the given input directly in order to learn the new shapes, there are approaches that use the given image in order to generate more information that is used within the neural networks. For example, Marrnet by Wu et al. 2017 and ShapeHD by Wu et al. 2018 employ an estimated 2.5d representation of the object in order to recover a 3d model from an RGB image. Unlike Wu et al. 2018, Wu et al. 2017 use the estimated 2.5d representations to supervise the model generation. While Wu et al. 2018 adopt depth, normal and silhouette values to recover an initial 3d representation that is further shaped by using the cross-entropy loss between the predicted and the target voxels as well as learned shape priors (Wu et al. 2018, 650), Wu et al. 2017 utilise those representations in order to compute the loss for the backpropagation (Wu et al. 2017, 542). Despite their different approaches, both methods fail in recovering thin and complex structures (Wu et al. 2018, 659) (Wu et al. 2017, 546), which is also partially due to the nature of voxels since their nature encourages the network to focus on the wider parts of the models (Wu et al. 2018, 660). These shortcomings were partially rectified by research such as the TPWCoder by Chen et al. 2020. Unlike the methods mentioned above, their research takes the topology of the object into account, which resulted in a topology-aware encoder-decoder structure. This structure can be used to refine the reconstruction of shapes of other reconstruction methods such as Marrnet Wu et al. 2017 by using a combination of the loss between the ground-truth genus and predicted genus, as well as the topology respectively and the loss between the by the reconstruction method predicted model and the model predicted by the TPWCoder. This results in finer details and in the case of a combination with Marrnet also improved performance (Chen et al. 2020, 1090–1094).
Voxels are still a commonly used output form for recovering 3d models because neural networks can easily process them. However, this representation form has some shortcomings other forms do not. As mentioned, thin structures are not as well recovered, which can require an extra fine-tuning step in order to improve the quality of the model. Furthermore, the resulting models can lack overall resolution, since high-resolution voxels are difficult to process due to them being regularly sampled from 3d space and having a poor memory efficiency (Kato, Ushiku, and Harada 2018, 3907). In order to compensate for these drawbacks, research like Tatarchenko, Dosovitskiy, and Brox 2017 has tried to use other data structures like octrees to reorganise the used voxels to improve computation and memory efficiency (Tatarchenko, Dosovitskiy, and Brox 2017, 2107), and research like Chen et al. 2020 tried to improve the visual flaws of the resulting models by using additional fine-tuning steps. Although those are promising approaches, other methods like point clouds do not inherently have those problems and achieve overall better

performance and quality of recovered models in most cases.

2.1.3 Point Cloud based Techniques

Point clouds have been used as a model representation method due to their simple structure that is easy to learn within a neural network (Fan, Su, and Guibas 2017, 2463) and are more computationally and memory efficient compared to voxelised representations as well as producing an overall better quality model compared to methods using voxels (J. Wang et al. 2020, 7).
Therefore, point clouds are not only used as the final output medium of a model but also as an intermediate representation, which is then converted to a regular mesh. This is done since point clouds are not as efficient in representing the underlying 3d geometry (Fan, Su, and Guibas 2017, 2463) and, like voxels, it is more complex to apply texture or lighting to these structures (Kato, Ushiku, and Harada 2018, 3907). This intermediate representation has been utilised by e.g. Lun et al. 2017 and W. Wang et al. 2019. The research by W. Wang et al. 2019 focuses on deforming an existing mesh into another shape by sampling points on the target and source object, effectively transforming them into point clouds, and extracting feature vectors in order to compute the vertex offsets. The loss is then computed by applying the Chamfer and Earth's Mover distance to sampled points of the predicted model and the target model. In contrast, Lun et al. 2017 take an approach similar to Wu et al. 2017 and Wu et al. 2018. They take one or more line drawings as input and convert them into multi-view depth and normal maps and compute a foreground map, which predicts whether a pixel is a foreground pixel or not. The point cloud is then generated by mapping the foreground pixels to the 3d points, which are then aligned according to the depth map and known camera parameters. The normal of the point is reproduced according to the normal map. Since there are inconsistencies in the maps, in a second step ICP (Rusinkiewicz and Levoy 2001) is run in order to correct those flaws and the method tries to optimise the point values according to various terms, like if they are consistent with depths and normals of corresponding 3d points generated from other viewpoints. The final mesh reconstruction is done using the screened Poisson Surface Reconstruction algorithm (Kazhdan and Hoppe 2013) and the resulting mesh can be further refined by deforming it according to 2d contour points extracted from the input sketch (Lun et al. 2017, 70f). By using point clouds as an intermediate representation, the benefits of the simplicity of a network handling those structures and the final mesh representation, which is discussed later in this section, are combined. However, an extra step of reconstructing the mesh from the point cloud is needed and it is usually not trivial to convert the point set into a ready-to-use mesh, since the individual points do not have information about their neighbours and the connectives (N. Wang et al. 2018, 58).

Alternatively, point clouds have also been used as the final representation method for reconstructions, not just as intermediate representations. J. Wang et al. 2020 standardised single sketches as input for their network. Those sketches are, in their case, created from synthesised sketches, which are distorted and then dilated and refined in a domain translation module in order to mimic hand-drawn sketches. This helps with the training process since the synthesised data, the networks are normally trained on, do not mimic freehand drawings in a correct way.

Therefore, deformations are applied and different styles are translated to the input image in order to mimic various drawing styles and geometric distortions of a hand-drawn sketch. To reconstruct a point cloud from the standardised sketch, the viewpoint is estimated and used to generate a point cloud. This point cloud is further transformed by a rotation matrix so that the point set aligns with the ground-truth point set. The loss is computed by the Chamfer distance between the two point clouds (J. Wang et al. 2020, 2, 4).
Another method that uses point clouds as its final representation is Fan, Su, and Guibas 2017. They use a random vector and an RGB input image in order to output a matrix of coordinates that represent the predicted shape. In their research they also introduced an additional predictor branch that operates parallel to the first branch and that learns to construct a 2d image containing coordinates. This results in a more ordered depiction of the surface and is useful to capture the general structure of the object, while the first branch completes the object with the more detailed components (Fan, Su, and Guibas 2017, 2466–2469).
A recent method, which also results in point clouds was introduced by Gao et al. 2022. Similar to Wu et al. 2017 and Wu et al. 2018, as discussed in Section 2.1.2, they use intermediate 2.5d representations in form of density maps in order to represent a predicted density distribution of a 2d input sketch. To do that, they employ an encoder-decoder-based convolutional neural network, which extracts the features from the input sketch and outputs them into a feature map. An L1 loss between the ground-truth density map and the predicted density map is used for loss computation. In order to improve the accuracy of the resulting point cloud, all upsampled feature maps are concatenated and used for further steps instead of only the final map. This helps with the prediction of the final point cloud, which is done by sampling the feature map and constructing a 2d point cloud using the sampled x,y coordinates. Once this is completed, the z coordinate is predicted by using a GAN, which takes a noise variable and a local feature. The Chamfer distance between the output point cloud and the ground-truth point cloud is used to supervise this network. Using the three predicted coordinates, a 3d point cloud can be constructed (Gao et al. 2022, 468–470).

2.1.4 Mesh based Techniques

Another form, models can be represented as, is polygon meshes. Although using the meshes and mesh connectivity within the neural network is still a problem (W. Wang et al. 2019, 1038), they are a compact data structure, memory efficient (Kato, Ushiku, and Harada 2018, 3908) and they are the representation form that is commonly used in applications like 3d modelling software, since they hold qualities like being easily deformable and capable of depicting shape details (N. Wang et al. 2018, 55).
Early methods that have used meshes as their output representation are Kato, Ushiku, and Harada 2018 and N. Wang et al. 2018. Both methods, as many others working with this output type, used a given mesh and deformed it, similar to the research by W. Wang et al. 2019 mentioned before. N. Wang et al. 2018 use a convolutional network to extract features from the given RGB image by projecting the 3d vertex positions of the model on the 2d input image plane. In a second network, the extracted features, which contain both information about the 3d shape and the 2d image, are then shared between neighbouring vertices, resulting in the addition

of more vertices and the displacement of the already existing vertices. The mesh reconstruction is regularised by 4 losses (N. Wang et al. 2018, 60):

- The sum of the loss between the normals to ensure the consistency of surface normals
- The Chamfer loss for vertex location consistency between the GT and the predicted mesh
- An edge length regularisation to prevent outliers
- Laplacian regularisation to preserve the relative location between the vertices

Contrary to the sum of losses and regularisations that N. Wang et al. 2018 employ in order to regularise the mesh reconstruction, the Neural Mesh Renderer by Kato, Ushiku, and Harada 2018 build upon the work of Yan et al. 2016, which was discussed above and employs the silhouette loss utilising the Intersection over Union metric and a smoothness loss, which acts as a regulariser. The mesh generator, to which this loss is backpropagated, is part of a bigger reconstruction framework. This reconstruction consists of the mesh prediction, which works by deforming a 3d sphere based on a 2d input image, and a neural renderer, which takes the 3d mesh as input and uses differentiable rendering to generate the silhouette of the mesh (Kato, Ushiku, and Harada 2018, 3907). A disadvantage of this approach is that it works only for a simple type of geometry. Reconstructing complex objects like cars or objects with various topologies did not result in accurate models (Kato, Ushiku, and Harada 2018, 3913). N. Wang et al. 2018 illustrate similar problems in their research, they can only recover objects similar to the topology of the initial object (N. Wang et al. 2018, 65).
However, incorporating DR in the learning process of the reconstruction of 3d models is a promising method that was employed to a greater extent in recent years. A closer description of differentiable rendering itself is provided in Section 2.2. Due to its increasing importance in retrieving models from 2d input images some methods, like Yan et al. 2016 and Kato, Ushiku, and Harada 2018, employ DR in their work, as discussed above. In addition to those methods, recent research, e.g. conducted in Liu et al. 2019 and Xiang et al. 2020, employs a differentiable renderer. As illustrated in Yan et al. 2016 and Kato, Ushiku, and Harada 2018, they use DR in order to extract certain features, which are backpropagated to the mesh generator. By using gradients retrieved via their DR technique, which is described in Section 2.2, Liu et al. 2019 are able to convert features from image pixels to shape and colour. The silhouette, colour, and geometry of this mesh are compared to those characteristics in the ground-truth mesh, resulting in the respective losses. The final loss consists of the weighted sum of those three losses (Liu et al. 2019, 7711). Contrary to that, Xiang et al. 2020 consider model generation as a problem of normal map generation, similar to Wu et al. 2018, Wu et al. 2017 and Lun et al. 2017. From a single input sketch, a normal map is generated, using conditional generative adversarial networks, based on the work of Su et al. 2018 and Isola et al. 2017, which is discussed in Section 2.4. The resulting normal map is then used to deform a given template geometry, which in their case is a sphere, into the desired model. From this mesh, a normal map is computed using a differentiable renderer. This map and the original normal map are then used to compute the normal loss of the mesh prediction. This loss is added to the silhouette loss, computed from

the difference between the predicted mesh and the ground-truth silhouette using the Intersection over Union metric, the edge loss, derived from the As-Rigid-As-Possible energy, and the smoothness loss, which ensures the consistency of the surface (Xiang et al. 2020, 5).
A method similar to this was part of the research of Guillard et al. 2021. They compared two methods, where one used the same approach as Xiang et al. 2020, while the other approach involves the determination of 3d points from the sketch directly and optimising them by minimising the Chamfer distance between the resulting contour and the input sketch. This is done by projecting the mesh onto an image and writing the external contours to that image. Then, the pixels not belonging to the external lines of the depicted projection and the mesh face that projects to it, are matched. By trying to minimise the Chamfer between the projected external contours and the real external contours of the input sketch the loss is computed. This method allows for partial sketches to be used, which can be exploited by mesh deformation operations where only parts of a given input mesh should be deformed according to sketch or mesh refinement processes. In this case, the Chamfer-based approach has a clear advantage over the other method. Furthermore, the training of an image translation network is not required, which makes the second approach more deployable (Guillard et al. 2021, 13006).
Although the mentioned methods yield promising results, they still have the same problems that were already mentioned by Kato, Ushiku, and Harada 2018. While the results of SoftRas have been better in terms of the level of detail the Neural Mesh Renderer is able to recover the overall shape more accurately according to the conducted experiments (Liu et al. 2019, 7713). Xiang et al. 2020, 9 reported having issues related to that, e.g. non-connecting edges and acknowledged that using base shapes that have the same topology as the desired output could minimise that problem.
In order to manage the problem of missing or ignored topology information, over the years, there have been a number of scientific papers published that try to identify the topology of 2d images and 3d models. In reconstruction networks there has also been research conducted that takes the model's connectivity into account, e.g. the refined network by Chen et al. 2020. Since reconstruction networks with meshes as learning parameters tend to use basic meshes that are deformed into the desired object (Ben Charrada et al. 2022, 337), some research has made it part of their recovering network to take the topology into account. Smirnov, Bessmeltsev, and Solomon 2021 use templates, which specify the connectivity of the Coon patches they used to describe their reconstructed surfaces. Those templates can be obtained beforehand using segmentation algorithms such as Yi et al. 2016 on the ground-truth model. One or more renderings are put into the reconstruction network, which tries to fit a collection of patches to the ground-truth model the renderings were obtained by optimising against a sum of losses (Smirnov, Bessmeltsev, and Solomon 2021, 3–5). While Smirnov, Bessmeltsev, and Solomon 2021 obtain their templates beforehand, Ben Charrada et al. 2022 use an extra pruning network that applies the topology to the mesh. They extract image features from an RGB input image by utilising an image encoder and use the retrieved information to deform an ellipsoid in three deformation stages. After each deformation stage, a vertex is added to each edge, essentially subdividing the mesh and obtaining a more detailed representation of each stage. Once the deformation stages are completed, the face pruning network intelligently deletes faces that represent large areas that do not belong to the object as depicted by the input image. Depending on how accurate the resulting mesh is, the agent performing the deletions receives a reward. In

the final step, another deformation network, similar to the first one, is used in order to refine the pruned mesh and get rid of edgy boundaries (Ben Charrada et al. 2022, 338f).

2.2 Differentiable Rendering

Rendering is the process to project a 3d scene defined by several parameters like geometry and camera properties onto a 2d screen. However, due to the complexity of these operations, inverting this procedure in order to obtain 2d information and differing levels of supervision for a 3d scene understanding is not straightforward. A solution to do that is to integrate the rendering process into a neural network. The methods that obtain the gradients of the rendering process, which can then be used for example in the neural networks, are defined as Differentiable Rendering. Via backpropagation, the gathered gradient's scene parameters are optimised. Therefore, in recent years DR methods in combination with neural networks have been used to tackle several problems in the field of computer vision, like 3d object reconstruction, as mentioned in Section 2.1, human-pose estimation, hand-pose estimation and face recognition (Kato et al. 2020, 1f).

Early work in the field of differentiable rendering involved computing gradients with respect to simple materials, but other parameters like vertex position or placement were not handled. The influence of those variables has been included in later work, e.g. via approximate differentiable rasterisation of meshes, like Loper and Black 2014 did. The work by Loper and Black 2014 also marked a milestone in DR research, since their method introduced the first general-purpose differentiable renderer named OpenDR (Loubet, Holzschuch, and Jakob 2019, 2).
Nowadays various DR methods exist. Like rendering processes, those techniques differ mainly depending on the type of structure they are operating on:
For point clouds, the rendering process is quite straightforward: The first step is to compute the screen space coordinates of a 3d point. Then, for each 3d point, the influence on the pixel's colour is determined and finally, their influences and z-values are used to accumulate points to compute the pixel's colour. Since the first step is a matrix multiplication, this is easily differentiable. However, because steps two and three are not as straightforward to differentiate, multiple methods have been developed: One way to make the second step differentiable would be to create influences larger than the one pixel per screen space coordinate in the image e.g. using the Gaussian blur like Yifan et al. 2019 did. The third step could be made differentiable using a multitude of rendering methods, for example by computing the weighted sum of the point's colours based on the point's influence, as Yifan et al. 2019, 3 did or by weighing all 3d points according to the distance from the camera and the spatial influence at each pixel (Kato et al. 2020, 5f).
Since voxels represent points in a regular 3d grid, they can be rendered using rays by projecting the hit voxels to the pixel the ray descends from. This can be done by first collecting the voxels along the ray, which is differentiable regardless if it is performed in world or screen space, and then aggregating the voxels along the ray, like Yan et al. 2016 did by associating an occupancy value to each voxel and then choose the one with the highest value on each ray (Kato et al. 2020, 5).

Implicit function representations, similar approaches to the one rendering voxels, are the simplest to use in DR: Random points or regular points with random perturbations along a ray or sampling random points in the 3d space and checking their intersections with rays are the most common ways the rendering of implicit representations can be made differentiable (Kato et al. 2020, 7).
Two main methodologies have been established for making the rendering process of meshes differentiable: Approximated Gradients and Approximated Rendering. The first one builds on the idea of approximating the backward pass of the neural network (Kato et al. 2020, 3), implemented by e.g. Loper and Black 2014, Kato, Ushiku, and Harada 2018 and Xiang et al. 2020. The idea implemented by Loper and Black 2014, 160f is to use image-space filters in order to take the neighbouring pixels into account when computing gradients, similar to a solution approach used by point clouds. Kato, Ushiku, and Harada 2018, 3909f use a similar approach, however, they do not utilise local filters. Instead, they use gradients that incorporate back-propagated gradients of non-direct neighbours of the initial pixels, which results in complex computation. Xiang et al. 2020, 5 build upon the idea of Kato, Ushiku, and Harada 2018. However, they use their technique in order to incorporate the normal loss into their reconstruction framework, unlike most differentiable renderers used for reconstruction tasks, which mostly differentiate the silhouette or RGB rendering of a scene, as described in Section 2.1.
The second methodology, which is widely used to differentiate mesh rendering, approximates the rasterisation instead of the backward pass. This was used by e.g. Liu et al. 2019, who consider every pixel of the RGBA input image to be a triangle of the resulting mesh. Via the usage of a probability map, which defines the probability of the pixel is inside the triangle, and an aggregate function, merges the colour maps of the pixels with the output of the probability map and the relative depths, 3d parameters like camera, texture or material can receive gradients from the image (Liu et al. 2019, 7709).

In order to apply the correct DR method, the problem that should be solved via using DR, the parameters and the DR technique itself need to be well understood. However, it is not always necessary to implement methods from scratch. Despite using open-source DR techniques, DR Libraries provide a wide range of functionalities, which allow for a rather straight-forward implementation of DR. Well-established libraries are Tensorflow Graphics by Google, Kaolin by Nvidia, PyTorch3d by Facebook and Mitsuba 2 (Nimier-David et al. 2019). While the first three rely on rasterisation, Mitsuba 2 utilises raytracing. Furthermore, Mitsuba 2 uses automatic differentiation supported by their custom library Enoki (Jakob 2019), while the other libraries rely on the backpropagation capabilities of PyTorch or Tensorflow (Kato et al. 2020, 13–15).
In mid-2022 Mitsuba 3 by Jakob, Speierer, Roussel, and Vicini 2022 and Jakob, Speierer, Roussel, Nimier-David, et al. 2022 was released, which also relies on a custom automatic-differentiation, Dr.Jit, which is the predecessor of Enoki. It outperforms its predecessor by relying on megakernels, which compile the rendering graph to data-parallel kernels (Jakob, Speierer, Roussel, and Vicini 2022, 9). Furthermore, features like the usage of geometry-based differentiation are provided by the latest revision, unlike in Mitsuba 2, where this is provided in an extra branch on the repository based on the work of Loubet, Holzschuch, and Jakob 2019, which is not up to standard with the current state of the renderer. This differentiation however

is crucial for tasks like 3d mesh reconstructions from 2d input images and in Mitsuba 3 inspired by the work of Loubet, Holzschuch, and Jakob 2019 and Bangaru, Li, and Durand 2020 and therefore different from the other libraries, which rely on techniques based on rasterisation. The main challenge that Jakob, Speierer, Roussel, and Vicini 2022 need to overcome in their work is to make the Monte Carlo sampling differentiable to certain parameters like positions. This is inherently not possible, because these parameters compared to others like colour do not change the composition of the scene, which would require the sampling positions to move along with the deformations. To solve that, Jakob, Speierer, Roussel, and Vicini 2022 use a combination of the technique by Loubet, Holzschuch, and Jakob 2019 and Bangaru, Li, and Durand 2020 and results in re-parameterisation of the rays: During the initial rendering, the re-parameterisation does not have any effect on the path tracing process. When differentiated, auxiliary rays are used to intersect with the shape and based on that information construct a warp field. Since this field contains information about how the silhouette of the object was deformed, in the back-propagation it is now possible to scatter the parameter gradients into intersected shapes (Jakob, Speierer, Roussel, and Vicini 2022, 14).

2.3 Generative Adversarial Networks and Wasserstein Generative Adversarial Network

Generative Adversarial Networks (Goodfellow et al. 2014) have become widely used in the field of computer vision and are used primarily to create new content like images. These outputs are variants of content in a large dataset, which the network learns the features of and uses to compose its own versions.
The structure of these networks consists of 2 parts: a generator that predicts images and a discriminator, which challenges the generator and gives feedback on the predicted images. The respective loss functions and therefore objectives are different in the various GAN variants. A traditional GAN type like a Conditional Generative Adversarial Network classifies a given image either as real or as fake. The objective of the network is to train the generator to output predictions of a given input in order to fool the discriminator into classifying the predicted images to be real ones. Formally, in the training process, the goal is to minimise a divergence, like Kullback-Leibler or Jensen-Shannon, between the real and the predicted data distributions. However, this can lead to vanishing gradients due to the local saturation of the discriminator (Gulrajani et al. 2017, 5768). Furthermore, it is possible for the discriminator to get stuck at a local minimum when the generator produces identical samples, which always fools the discriminator, also known as mode collapse.
In order to avoid these problems, Wasserstein GANs are used, which employ the Earth-Mover distance, which defines the minimum cost of transporting mass in order to transform one distribution into another. Their value function is defined as (Arjovsky, Chintala, and Bottou 2017, 4):

$$\min_{G} \max_{D \in \mathcal{D}} \mathbb{E}_{x \sim \mathbb{P}_r}[D(x)] - \mathbb{E}_{\tilde{x} \sim \mathbb{P}_g}[D(\tilde{x})] \tag{1}$$

where the distance between the distribution of the real images and the predicted images is described. Furthermore, the discriminator, which is known in WGANs as a critic but is referred

to as discriminator for consistency, is trained to optimality, which in practice means that it is trained more often than the generator to avoid the possibility of mode collapse, which is different from the classic GAN structure, where the generator and discriminator are both trained every epoch (Arjovsky, Chintala, and Bottou 2017, 5). However, since the $\mathcal{D}$ in equation 1 is defining a set of 1-Lipschitz functions, which ensures that the norm of their gradients is 1 nearly everywhere in order to provide reasonable gradient information to the generator, constraints need to be enforced on the discriminator, which makes this worse but is justified by the improved training of the generator (Gulrajani et al. 2017, 5768, 5775). Arjovsky, Chintala, and Bottou 2017, 4 solve this by employing weight clipping in order to ensure that the loss is within a compact space. This introduces some problems, e.g. it can lead to vanishing gradients or biases the discriminator towards learning the simpler functions (Gulrajani et al. 2017, 5769). Therefore, the gradient penalty introduced by Gulrajani et al. 2017 is used to enforce the Lipschitz continuity while omitting the mentioned problems. The idea is to penalise the model if the gradient norm between points interpolated between the real and the predicted data is not 1. Since using batch normalisation in the discriminator would introduce correlations in the batches, this would invalidate the gradient penalty because the objective is to compute the penalty for the gradient wrt. every input (Gulrajani et al. 2017, 5769f). Therefore, when implementing a discriminator the usage of layer normalisation is recommended over the usage of batch normalisation (Gulrajani et al. 2017, 5).

2.4 Map Generation

Depth and normal estimation are critical parts of scene understanding tasks in computer vision. Such an understanding is crucial for a variety of tasks, like pose estimation tasks, image segmentation and in 3d reconstruction networks where the maps are used as 2.5d representations of an input image (Eigen and Fergus 2015, 2650), as proposed by Xiang et al. 2020 and Lun et al. 2017.

Modern approaches utilise neural networks in order to learn how to map normal or depth values onto sketch inputs. A common way of doing this is using Convolutional Neural Networks, as Eigen and Fergus 2015, Wang, Fouhey, and Gupta 2015 and Eigen, Puhrsch, and Fergus 2014 did. CNNS consist of multiple convolution layers, which convolve the input and pass the output to the next layer, pooling layers that discard redundant information, and fully connected layers, which are usually the last stage of the CNN and used to assign the final values like probabilities. How each network is designed depends on the task at hand.
The aforementioned techniques employ at least two CNNs, where one captures the whole scene, and the second is responsible to fine-tune the coarse output of the first network, like e.g. in Eigen, Puhrsch, and Fergus 2014, 2369, or predicting local features, which are then combined with the output of the first network like Wang, Fouhey, and Gupta 2015 did. Since CNNs are generally supervised, meaning the loss is computed by comparing the predicted image to the GT, the methods also applied variations of those in order to backpropagate. Therefore, they learn how to compute the various output images, like normal maps, depth maps or image labels.

Another way of reconstructing depth and normal maps is to use an Encoder-Decoder network structure, as done by Lun et al. 2017. This network structure consists of an encoder part, which converts a given sketch into a compact representation of the shape information by employing a series of convolution layers and batch normalisation layers, which use transformations in order to normalise the inputs of the layers to make the prediction process faster and more stable. The decoder part uses the output of the encoder and outputs the depth, normal and foreground probability map for the given input. To accomplish that, they adapt the U-Net structure, which concatenates encoder and decoder layers, utilising upsampling and convolutional layers employed along with batch normalisation layers and dropout layers that essentially remove nodes at random in order to force the network to essentially connect in different ways every iteration leading to a more robust network. Like in the CNN techniques, the network's loss is computed utilising the weighted difference between the predicted and ground-truth maps (Lun et al. 2017, 69). The outputs of this network are then used as 2.5d representations of the input sketch in a reconstruction network, as described in Section 2.1.

Isola et al. 2017 and Su et al. 2018 took a similar approach to Lun et al. 2017, however, they used a cGan, which encapsulates an Encoder-Decoder structure. This means, they employ a Generator-Discriminator structure, in which the generator network learns to predict the images, while the discriminator network learns to differentiate between ground-truth images and predicted ones. In order to do that, the generator part is similar to the one by Lun et al. 2017. The discriminator part is described structurally similar to the CNN structure described above, but missing the final fully connected layer. Each of the networks has its own loss function, with the discriminator comparing the generator's prediction to the GT and the generator utilising the sum of the prediction of the discriminator on the predicted image and an L1 loss of the predicted and real image. That way, the networks try to minimise their loss against each other, as described in Section 2.3.
Su et al. 2018 used the approach by Isola et al. 2017, but instead of a general-purpose image translation network, they focused on predicting normal images from sketches, which however does not change the structure of the network, they just verified their thesis for normal maps and does not indicate that other image types such as greyscale images cannot be predicted using this network. They introduced the Wasserstein loss (Arjovsky, Chintala, and Bottou 2017) as part of their loss function instead of the loss utilised by Isola et al. 2017. Su et al. 2018 also employed user interaction by adding a mask, which can be used by the user in order to define certain normal values at specific points. Generated masks are also incorporated in the training process and therefore included in the overall loss function for the Generator, alongside the Wasserstein loss mentioned above and an L1 difference between the ground-truth and the predicted image (Su et al. 2018, 4–6).
Xiang et al. 2020 employed a similar approach to Su et al. 2018 and Isola et al. 2017 in their reconstruction network in order to generate a 2.5 representation in form of a normal map from an input sketch, as discussed in Sections 2.1 and 2.2. The main difference to the other functions is the generator loss, for which they used the prediction of the discriminator on the generated image, the difference between the predicted and the ground-truth normal map, and the loss at local sampled pixels of sharp geometric features like corners or edges (Xiang et al. 2020, 6).

2.5 Image Segmentation and 2d Topology Awareness

Image segmentation has been an important topic in the field of image processing and analysis since the early days of those research areas. The objective is to partition an image into multiple images based on the similarity of the regions in the pixels, like colour values, textures or intensity (Kaur and Kaur 2014, 810). Based on what and how the images are segmented is given by various image segmentation techniques, which have been developed over the last decades.
Segmented images can be applied to many problems, for example, image compression, object recognition, medical imaging, input images for neural networks, robot vision, etc. However, segmented images can provide information about the topological properties of the depicted object: Once the picture has been decomposed into its subsets, relations between them can be described. This is because the topology of an object is defined by the connectivity of the subsets or lack thereof, segmented images can be used in order to describe the topology of a depicted object (Rosenfeld 1979, 621–623).

There are many different problems, which can be solved using image segmentation. However, the techniques are often very specific to a problem, which is why many different segmentation techniques exist. Instead, over the years many specialised methods were invented, which can be applied to a specific task. Those techniques can be categorised in several ways. One widely used accepted split is to classify them based on the properties of images into discontinuity and similarity based approaches (Kaur and Kaur 2014, 810):

- Discontinuity detection partitions the image based on sudden changes in the intensity levels of the colour values. A common technique that uses discontinuity is edge detection.
- Similarity based segmentation divides the image into regions based on predefined criteria, like thresholding techniques, region-based approaches and clustering techniques.

However, nowadays this split does not suit all techniques equally well, since neural networks do not adhere to either of those classes. When dealing with techniques like that, a much more fitting classification would be (Sultana, Sufian, and Dutta 2020, 3):

- Semantic segmentation, where every pixel is assigned to a dedicated class.
- Instance segmentation, which is very similar to the practice of object detection.

Following, some of the most relevant image segmentation techniques are discussed: thresholding techniques, segmentation techniques employing neural networks, edge-detection-based techniques and region-based approaches. Other important segmentation methods include:

- Clustering techniques: Pixels are assigned to specific clusters based on similar characteristics. This can be done in either a hierarchical way, where a tree structure is utilised to represent all clusters and their subclusters, or in a fuzzy way, where pixels are not necessarily assigned to one class only (Kaur and Kaur 2014, 812).

- Watershed methods: The image can be viewed as a relief where pixel intensities represent a certain height. This relief is filled with water until a certain height is reached. The resulting filled basins represent the connected regions. This technique is very similar to region-based techniques like flood fill, which is discussed in Section 2.5.4 (Vincent and Soille 1991, 584f).
- PDE techniques: e.g. Snakes by Kass, Witkin, and Terzopoulos 1988, where the idea is to define regions by using energy-minimising splines, which are guided by external and internal forces to capture structures of the image (Kass, Witkin, and Terzopoulos 1988, 321).

These methods are not discussed in detail, since they are not relevant to this thesis.

2.5.1 Threshold Techniques

One of the easiest ways to split an image into homogeneous parts is to use a value that poses as a threshold. The segmentation works by comparing the values of the pixels to this threshold and then assigning the pixel to a particular class or region based on whether its value is higher or lower than the threshold. This threshold value can be provided manually, e.g. a user, or easily chosen via the peaks of the image histogram. These thresholds are a constant value for the whole image, also known as a global threshold, which results in binary images. If the threshold varies across the image, it is either a variable threshold, where e.g. every subsection has its own threshold or multiple global thresholds applied to the image (Kaur and Kaur 2014, 811).
Although this method is fairly easy, it comes with some drawbacks. It does not take the spatial characteristics of an image into account and is very sensitive to noise, which can lead to the false assignment of pixels to certain regions or classes (Kaur and Kaur 2014, 813). Therefore, topological information regarding the scene can be wrong, if pixels are assigned to the wrong classes. Furthermore, since the values of the pixels of the image are used to filter whether it belongs to a region, only images where the pixel has a value that is relevant to the overall depiction can be segmented by this method, like RGB images or greyscale images. Simplified representations of objects, like sketches that are only described by their outline, cannot be segmented from e.g. the background of the image.

2.5.2 Neural Network-based Segmentation Techniques

Utilising neural networks for image segmentation has been around for years. Since neural networks essentially learn from inputs and targets to predict respective outputs, the applicability of these techniques covers a wide range of fields. They are suitable for both semantic and instance segmentation tasks and can be used on greyscale images as well as RGB or sketches and even videos.

Modern semantic image segmentation utilising neural networks mainly employ convolutional network, especially Fully Convolutional Networks. This specific type of network is able to

conserve the spatial information of the input image since the fully connected layers used in traditional CNNs, the structure of which is described in Section 2.4, are removed (Sultana, Sufian, and Dutta 2020, 4).
A recent example of semantic image segmentation was introduced by Ku, Yang, and Zhang 2021. They employ a convolutional encoder-decoder structure. While the decoder part of the network consists of convolutional networks in order to reduce the channels, the encoder part uses dilated convolutional layers in an aggregation module to obtain multilevel feature maps. This is done by feeding an RGB image into a CNN to obtain 4 layers of hierarchical feature maps. Those are then processed by the multilevel aggregation modules, which consist of 4 dilated convolution layers and a final pooling layer. In order to preserve the semantic information dilated convolution is used rather than standard convolution layers. This means that holes are inserted in the consecutive elements, which results in a larger receptive field. However, this can lead to the information being no longer as relevant, since a larger area is now involved, or the loss of consecutive information due to the holes, which do not provide any. In order to solve this, Ku, Yang, and Zhang 2021 use different dilation ratios for the four layers in their multilevel aggregation module and the information of all levels is extracted for feature aggregation. In the final step, the resulting feature maps are then fused with the hierarchical feature maps and the high-level feature maps are upsampled and spliced with the lower-level feature maps to make the final predictions (Ku, Yang, and Zhang 2021, 4–6).

Since the majority of instance image segmentation also relies on convolutional neural networks, the basic idea of feeding an input image to a network and a segmented output image is the same as in semantic segmentation. They are often divided into two modules, wherein the first stage of object detection is performed and in the second stage those detected objects are masked. Furthermore, it is also common that object detection models like Faster R-CNN are used and a binary mask is added, which results in an instance image segmentation like Mask R-CNN (Sultana, Sufian, and Dutta 2020, 17).

Such segmentation algorithms result in reliable segments and once trained they produce those outputs in a timely manner, they are tasked with critical operations like segmentation of medical images in order to reprocess them for medical diagnosis or are used within autonomous car driving. However, since there is a lot of data required to train those images and the training processes themselves are more computationally and timely expensive than the other approaches mentioned (Kaur and Kaur 2014, 813).

2.5.3 Edge-Detection-based Techniques

The aim of edge detection algorithms is to locate points in an image where the intensity of the image suddenly changes, also known as the discontinuity approach described above. These sudden changes usually indicate the boundaries of objects, therefore the outlines of the objects are the result of these techniques. Over the years, many methods have been developed in order to detect the edges of these results. Some of the most popular to this day are (Al-Amri, Kalyankar, and Khamitkar 2010, 805f):

- Using the Sobel operator to detect edges works by applying 2 3x3 kernels to the image. The kernels are the same, with one simply rotated by 90°. As shown in Table 2, the middle rows respond to the vertical resp. horizontal edges are filled with 0, while the rest of the kernel is weighted with 1 and 2, both negative and positive.

-1	0	1	1	2	1
-2	0	2	0	0	0
-1	0	1	-1	-2	-1

Table 2: Kernels used by the Sobel edge detection

- The Prewitt Edge detector is basically the same as the Sobel operation, however the weight of the current row resp. the column is not incorporated, therefore all values except for the middle row resp. the column is 1 or -1 resp., as shown in Table 3.

-1	0	1	1	1	1
-1	0	1	0	0	0
-1	0	1	-1	-1	-1

Table 3: Kernels used by the Prewitt Edge detector

- Contrary to the other algorithms mentioned, the Canny edge algorithm does not use kernels. The process involves a Gaussian function in order to smooth the image and the application of a difference gradient in order to compute the edge strength. Once that is done, gradient operation as mentioned in the previous edge detectors can be applied in order to compute the gradient magnitude and direction. The final steps consist of applying non-maximal or critical suppression to the gradient magnitude and threshold to the non-maximal suppression image in order to determine potential edges and suppress weaker or non-connected edges.

Although the edge detectors are able to detect edges on images different images, noise amount, computation time requirements, etc. may make one algorithm superior to another when applied to a certain task. For example, the Sobel operator is more sensitive to noise in images than the Canny Edge, since the smoothing operation reduces the image noise in the beginning, however, it is simpler to implement (Al-Amri, Kalyankar, and Khamitkar 2010, 805f). Another example would be that the Prewitt operators have been found to be more applicable to satellite images than the other techniques mentioned (Al-Amri, Kalyankar, and Khamitkar 2010, 807).
However, all edge detectors fail when there are too many edges since there is a chance of them detecting wrong edges, which in turn leads to the wrong representation of the depicted object. Another aspect that all methods have in common is that their output is basically a binary image depicting the outlines of the image scenery (Kaur and Kaur 2014, 813). This makes this technique optimal to be paired with another method, e.g. boundary depictions of image

sceneries are used in the region filling approach like by He, Hu, and Zeng 2019 can be used as input images and Chudasama et al. 2015 used Canny edge detection in a first step and later apply morphological operations and region filling in order to segment the image. This combination can be used for the detection of the topology of the depicted objects, however, the output of edge detectors itself does not provide meaningful information about the connectivity of objects.

2.5.4 Region-based Techniques

Region-based approaches can be divided into two basic methods: Region splitting and merging and Region growing. Region splitting and merging use two basic techniques, where images are in an iterative process split into smaller sections until only homogeneous regions are left and the sections are combined based on similar characteristics in order to form similar regions (Chudasama et al. 2015, 16).
Region growing algorithms start from one point within a region, also known as a seed, and then grow until a stopping criterion is met, e.g. the bounds of the region, similar to the watershed algorithm discussed above. In image grids, this is done by exploiting the 4- or 8-connectedness of the regions, which describes the relation of pixels with their neighbours (Kaur and Kaur 2014, 811), which according to Rosenfeld 1979, 623 describes the neighbourhood relations between pixels, and therefore if a region is simply connected or has cuts or holes.

Scan-Line. The starting points of the seeds can either be determined automatically, which are also known as scan-line or edge filling algorithms, or seed filling, also known as flood filling algorithms. The first algorithm relies on the detection of spans of scan lines lying inside the objects, with automatic seed detection along those lines and a filling process, which starts to fill the region to the right and the left of the determined starting point. An example of this was presented by He, Hu, and Zeng 2019. Their approach was to classify the pixels images in either background or boundary pixels. They then start with a flood fill from the padded origin of the image, in order to classify the initial exterior and classify the visited pixels as exterior pixels. After that, all other pixels in the image are considered to be a new starting point for the remaining flood fill operations and fill all its connected regions with either an exterior or interior colour. When those steps are completed, the interior class is swapped with pixels labelled as masks, and the exterior class is swapped back to background-labelled pixels. This way, the algorithm can determine where holes are, based on the premise, that adjacent connected regions should be treated differently, which essentially means when a region inside an interior region is automatically classified as a hole (He, Hu, and Zeng 2019, 763–765). While a clear advantage of these algorithms is that the seed positions are determined in an automatic manner, this limits the applicability to a certain degree of complexity which can be correctly segmented. E.g., for very detailed sceneries, where many little detailed elements are part of an object, those details can easily be identified as external or background objects, although they are detailed elements on top of the base element rather than holes.

Seeded Process. The second approach, which is also known as the seeded process, is much more popular. The seeding process is usually done in a manual manner, where user interaction

is involved. However there have been approaches to determine the seed positions automatically, but as mentioned above, determining whether a pixel belongs to an object, a detail belonging to said object or another object or the background can be a quite difficult task, especially when dealing with sketches, since they do not have colour and intensity values, from which membership could be determined.
An early example of this algorithm was introduced by Adams and Bischof 1994. They compare region growing to watershed approaches since they both exploit the similarities and connectivity of the regions and start their filling processes from designated areas within the regions. In their algorithm, they start from a set of determined seeds, which were given by users and then the 8-connectivity of the regions and the similarities between the pixels are used to fill the regions until the similarity criterion is not met anymore, meaning the neighbouring pixel belongs to another region, or all pixels are classified. If a pixel value is not similar enough to any seed pixel, they are subsumed by surrounding regions (Adams and Bischof 1994, 641–643).
While compared to the scan-line algorithms these images usually require human interaction for seeding processes, which in turn means, that more complex images can be segmented. However, automatic seeding processes are more time and memory-consuming than manual seeding. Despite this, the flooding process of both variants of the region growing algorithm already relies on the region connectedness, they can be used for topology determination, similar to watershed algorithms.

While region-based algorithms are intuitive and they are basically immune to noise, the clear disadvantage is that they require more memory and time to be computed than other algorithms (Kaur and Kaur 2014, 813). They can be applied to all kinds of images if the stop and similarity criteria are compatible with the image types. Therefore, these techniques can also be applied to sketches, however here the stop criterion has to be either based on properties of the flooded region, like the size, or the lines of the sketches, which in turn require clean, connected edges. In order to ensure this, either a stable edge detector has to be applied if the initial image is a grey or RGB image, or pre-processing techniques like those mentioned in Xiangyu et al. 2002 be applied in order to connect broken boundaries and remove unnecessary line segments.

3 Method

The implemented system takes in a user-generated sketch and returns a 3d representation of that. Python 3.10.4 is the python version used to implement the project and the system was tested on both Windows 10 and 11 with CUDA 11.4 as well as the Linux distribution Ubuntu 22.04. with CUDA 11.5. The main packages used are pytorch 1.12.1, pytorch lightning 1.7.5 (Falcon and team 2019), Mitsuba 3.0.2 (Jakob, Speierer, Roussel, Nimier-David, et al. 2022) and Dr.Jit 0.2.2 (Jakob, Speierer, Roussel, and Vicini 2022). For Mitsuba 3 as well as for Dr.Jit the CUDA variant is used to profit from the parallel computing capabilities of the GPU in the computations. Other relevant packages used are mentioned in the following as the part of the implementation they are used in is discussed.

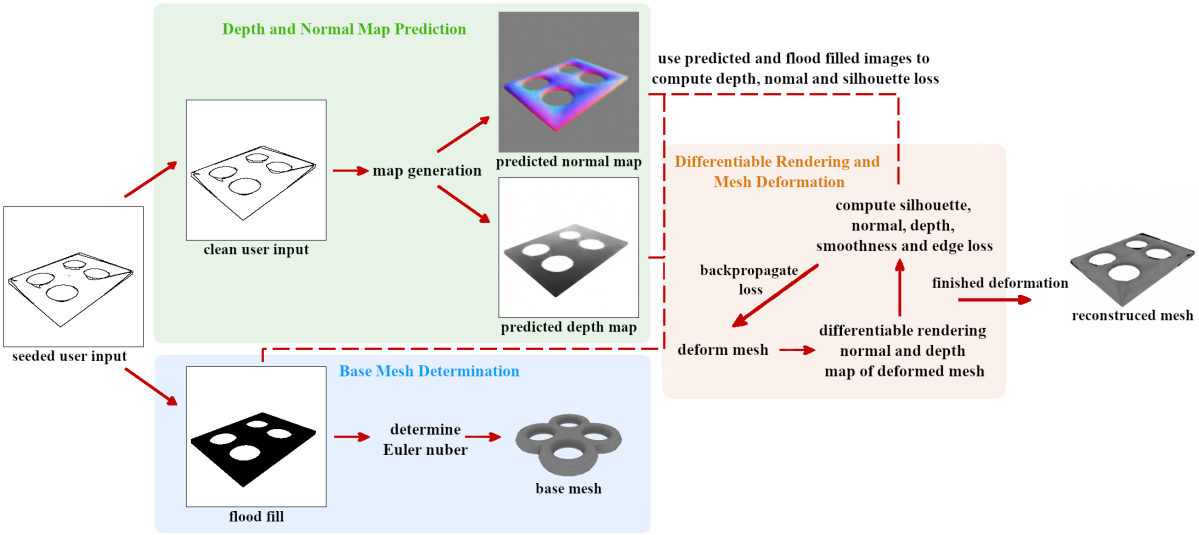


Figure 1: General system overview. From a seeded user sketch, the silhouette image, the normal and depth maps are translated and a base mesh is determined. Using those, a differentiable renderer is used in order to predict a 3d model.

As depicted in Figure 1, the system pipeline consists of three modules:

- Topology determination, where the genus of the sketched object is determined and a base object is chosen. This is done using a flood fill algorithm and the computation of the Euler number. This number representing the genus and number of depicted objects is used to select a matching base mesh from a variety of models.
- Trained neuronal networks that establish a normal and depth map based on the given sketch. The network architecture is based on the work of Su et al. 2018 and Isola et al. 2017 and two networks for the depth resp. normal map predictions are trained.
- A differentiable render deforms the base mesh using losses based on the depth and normal maps, the silhouette representation as well as the edge and smoothness regularisers until a 3d representation of the input sketch is created.

In the following sections, these three modules are discussed as well as the findings and practice used, which were either not mentioned within the research the modules are based on or specific to this framework but are crucial in order to ensure the creation of reasonable outputs.

3.1 Base Mesh Determination

To get a base mesh for the reconstruction, first, a flood fill is performed in order to differentiate the object from the background, which is then processed to determine the genus of the depicted object. The user-provided sketch has to include seeds, from which the fill algorithm starts. Therefore, the user has to consciously mark every part of the object with a colour that is neither

white (RGB(1,1,1)) nor black ((RGB(0,0,0))), as depicted in the first step of the topology part in Figure 1. It is also irrelevant if multiple colours are used for these seeds since the fill colour is predefined and is not the same as the colour used for the seeds. Also, multiple seeds in one segment are not problematic, since the algorithm visits the neighbours of the seeds and if they are already coloured, the algorithm goes right on to the next seed in the list. Although this seeding process puts more workload on the user compared to other segmentation methods discussed in Section 2.5 like neural networks, this can be justified since the user already has to interact with the sketch when creating it and putting seeds into the object is a timely insignificant addition to that process. Furthermore, additional time and resources would be required in order to implement and train an extensive segmentation method, which does not require a user input method, like a neural network, while the contribution to an improvement of framework output is expected not to be significant.

3.1.1 Flood Fill

For the segmentation, a stack-based 8-connected flood fill is used. This requires the lines to be closed as well as to be 4-connected, otherwise, the filling algorithm spills into other segments, and possibly the background. Although this requires the input to be clean, algorithms like Yi et al. 2016 could be used to pre-process such sketches to ensure that the input is free of any noise and that the lines meet the requirements. Despite this disadvantage, the algorithm itself enables the segmentation of complex objects with a lot of details, which is due to the user-placed seeds, while other algorithms of the same simplicity fail to do so.
After the image is loaded, it is transferred to a range [0, 1] and padded. The seeds from the provided input sketch act as starting points for the fill algorithm. Then, it is saved in a lossless format and, if needed for debugging purposes, as a png file. A binary image is created where the edges of the object merge with the object's body because, for the application of the Euler number computation, this solid unit is required.

3.1.2 Genus Computation and Base Mesh Determination

The Euler number is a topological attribute, that describes the relationship between the holes and connected components in an object. It is defined by:

$$E = C - H \tag{2}$$

where C is the number of connected components and H is the number of holes. Since the user sketches are required to depict a single solid object (how this could be applied to sketches depicting multiple objects is described in Section 5.1), C, in this case, is 1. Therefore, in order to determine the number of holes, the Euler number has to be determined to apply the above-rewritten formula:

$$H = C - E \tag{3}$$

A common method to apply this method in applications, for example, described in Pratt 2007, is to match segments of the image to patterns, so-called bit quads. This method is similar to the

edge detection methods using kernels, like the Sobel operator, described in Section 2.5.3. There are 3 sets of detectors, Q1, Q3 and QD, which are depicted in Tables 4, 5 and 6 resp.

1	0	0	1	0	0	0	0
0	0	0	0	1	0	0	1

Table 4: Bitquads Q1.

0	1	1	0	1	1	1	1
1	1	1	1	0	1	1	0

Table 5: Bitquads Q3.

1	0	0	1
0	1	1	0

Table 6: Bitquads QD.

If one of the patterns of a group matches a 2x2 segment of the image, said group value is increased by 1. In this work, this was simply done by adding the values of the current pixel segments and comparing them against a value, that determined which bit quad group this segment fell into as depicted in Listing 1.

```
for x in range(shape_x-1):
    for y in range(shape_y-1):
        current_pixel = image[x][y]
        neighbours = [(x+1, y), (x, y+1), (x+1, y+1)]
        neighbour_sum = 0
        for i in neighbours:
            neighbour_sum += image[i[0], i[1]]
        sum = neighbour_sum + current_pixel

        if sum == 1:
            Q1 += 1
        elif sum == 3:
            Q3 += 1
        elif sum == 2:
            if current_pixel == 1 and image[neighbours[2][0], neighbours
                [2][1]] == 1:
                QD += 1
            if current_pixel == 0 and image[neighbours[2][0], neighbours
                [2][1]] == 0:
                QD += 1
```

Listing 1: Implementation of pattern matching in order to determine Euler calculation

Once the bit quads were compared to every 2x2 segment of the image, the final formula for the determination of the Euler number is:

$$E = \frac{1}{4}[n\{Q1\} - n\{Q3\} - 2n\{QD\}] \tag{4}$$

where $n\{Qx\}$ is the number of matches detected. With the usage of this formula, an 8-connectivity of the object is assumed, which complies with the 8-connectivity-based flood fill used for the segmentation. Furthermore, it is assumed, that the object is white while the background is black, however, this is inverted in this work, therefore Q1 and Q3 were swapped, leading to the used formula:

$$E = \frac{1}{4}[n\{Q3\} - n\{Q1\} - 2n\{QD\}] \tag{5}$$

Once the Euler number is computed via the application of the formula 3, the genus of the object is determined. This number is used to fetch the base mesh from a variety of predefined meshes, as depicted in Figure 1. For this thesis, five base meshes with different genera were created, which are depicted in Figure 2. All the base meshes are low poly with a vertex count of around 650 vertices per mesh, which is around the same vertex count that similar work has used in the past. However, the possibility of adding more or different meshes as well as using all types of different mesh formats is given, since in order to do a simple configuration of the dictionary which matches the genus to the corresponding model file name is stored in a JSON file, which has to be located within the same directory as the given base meshes. Therefore, when adding a new file or changing the filename of an existing file requires adding resp. changing the entry in this file as well as adding resp. changing the model to the directory is the only step needed to enable the usage of the said base mesh. Newly added meshes have to adhere to certain standards, which can be enforced using the script used for the mesh pre-process operation, which was also employed when creating the datasets for the map generation module, as described in Section 3.2.2. However, it is not mandatory to use this function, as long as the model is watertight, and normalised, which in this case means that the diagonal of the bounding box has a length of 1, manifold and the centre of the bounding box is located at the origin of the coordinate system. Deviations from these requirements can lead to problems in the mesh deformation module due to the rather fixed rendering setup, which would otherwise have to be adjusted accordingly. Furthermore, Mitsuba 3 requires either the use of Wavefront OBJ or Stanford Triangle Format models. It is important to use only vertex indices to describe the faces over variations like vertex data and texture coordinates or vertex data and normals since the face indices, which are used in the deformation module as described in Section 3.3, returned by Mitsuba 3 is the last indices of the face descriptors. This means when vertices and UVs describe a face, only the UVs are returned, when vertices, UVs and normals are used, only the normals are returned and when only the vertices are used to describe the faces, those are returned. The standard meshes provided are stored in PLY format, since this is recommended by the developers over using OBJ models[1].

1. https://mitsuba.readthedocs.io/en/latest/src/generated/plugins_shapes.html

Figure 2: Basic mesh templates with genus 0, 1, 2, 3 and 4.

3.2 Normal and Depth Map Generation

The normal and depth map generation rely on an image translation network implemented using PyTorch lightning, inspired by the research conducted in Su et al. 2018 and Isola et al. 2017. The network employs a WGAN architecture, consisting of a generator and discriminator, where the generator creates normal resp. depth images from a given input sketch, and the discriminator tries to maximise the predicted distance between real reference images and predicted images, as described in Section 2.3.
The WGAN is trained on datasets consisting of either normal and sketch images or depth and sketch images. Once the training process is completed, the resulting models can be used by feeding them a sketch and they return either a depth or a normal map. In order to incorporate this in the overall flow of the reconstruction network, the seeded user sketch is first cleaned from the seeds. This is done to remove the noise the seeds introduce, which could be interpreted by the network as part of the object and therefore relevant to the map reconstruction process, which could ultimately lead to less accurate maps.
The training step itself is divided into a train and validation step, as explained in Section 3.2.2. An additional function, test, can be used to get normal resp. depth maps from input sketches using trained networks. This split in train, validation and test is used since this is the most common division used in PyTorch lightning models.
The following sections describe the network architecture of the WGAN used as well as the performed training process in depth.

3.2.1 Network Architecture

The network used for the map generation is a Wasserstein GAN network like Su et al. 2018 rather than a conditional GAN, like Isola et al. 2017 did in their research. For the generator as seen in Figure 4, an encoder-decoder network is used. The encoder part consists of 8 convolutional layers, batch normalisation and ReLU activation functions, while the decoder consists of the same number of deconvolutional layers, layer normalisation and leaky ReLU activation functions. As described by Isola et al. 2017, the layers, except for the innermost encoder and outermost decoder layer are concatenated with their respective counterpart in the encoder resp. decoder part, meaning d1 is connected with e7, d2 is connected with e6 and so on, forming skip layers. That helps to preserve fine details since features are directly passed from the encoder to the decoder part instead of being further up and downsampled. Since the concatenation of those layers gives the network a U-shape-like appearance, as discussed in Section 2.4, this network

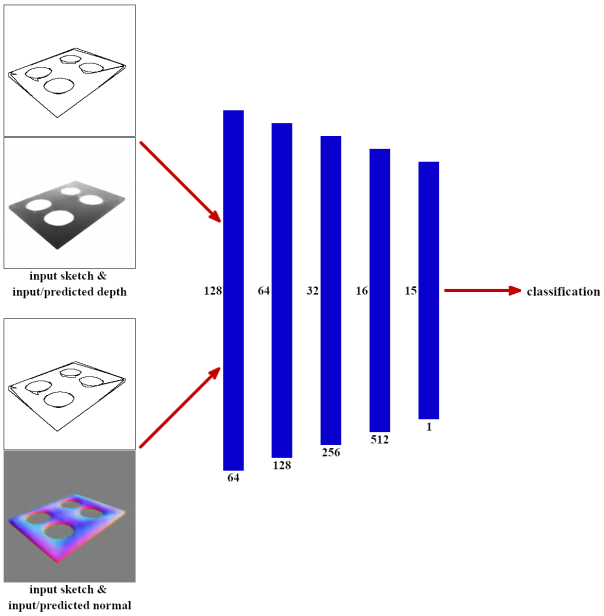


Figure 3: Network Discriminator, which takes the concatenated input and predicted image as well as the concatenated input and target image. Both concatenations are classified and the resulting classifications are then used to calculate the final discriminator loss.

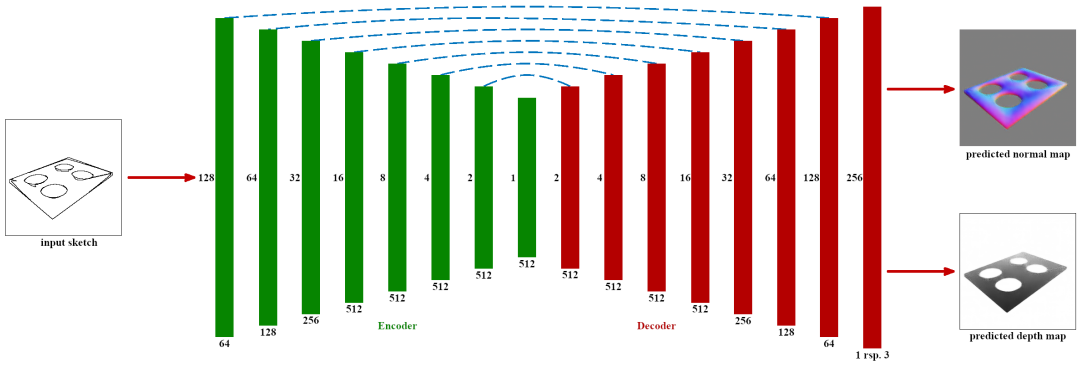


Figure 4: Network Generator, which takes a greyscale resp. RGB raster sketch as input and uses an encoder-decoder network. It predicts either a greyscale image, which depicts the depth map or an RGB image, which depicts the normal map, based on the input sketch.

structure is known as U-Net. This structure is used since previous research in the field of computer vision employed it successfully in tasks such as image-to-image translation, which is the goal of the normal and depth map generator.

The discriminator takes in the input sketch as well as either the predicted or the ground-truth normal resp. depth map, which can be seen in Figure 3. A total of five layers, consisting of 2d convolutional layers, layer normalisation and a leaky ReLU as an activation function are used to process the dataflow. The loss function described in Section 2.3 can be rewritten and simplified

to a loss function, which the discriminator is trying to maximise:

$$loss_{\text{discriminator}} = -(loss_{\text{real}} - loss_{\text{fake}}) \tag{6}$$

where $\text{loss}_{\text{real}}$ and $\text{loss}_{\text{fake}}$ are the mean of the prediction made on the real resp. fake images made by the WGANs discriminator
This loss function is extended by a gradient penalty. The implementation of this can be seen in Listing 2. Here, the first pairs of points on the real and the fake images are sampled uniformly and the results are interpolated. Then, the gradient for the data distribution of the interpolated image predicted by the discriminator is computed. Finally, the mean of the squared distance between the gradient's norm and 1 is relayed back to the discriminator, where it is weighted and added to the loss described above. The final loss function for the discriminator can formally be defined as:

$$\max_{D \in \mathcal{D}} \mathbb{E}_{x \sim \mathbb{P}_r}[D(x)] - \mathbb{E}_{\tilde{x} \sim \mathbb{P}_g}[D(\tilde{x})] + \lambda \mathbb{E}_{\hat{x} \sim \mathbb{P}_{\hat{x}}}[(||\nabla_{\hat{x}} D(\hat{x})||_2 - 1)^2] \tag{7}$$

or simplified:

$$\text{loss}_{\text{discriminator}} = \text{-(loss}_{\text{real}} \text{ - loss}_{\text{fake}}\text{) + weight}_{\text{gradient\_penalty}} \text{ * gradient\_penalty(self, real\_images, fake\_images)} \tag{8}$$

```
def gradient_penalty(self, real_images, fake_images):
    alpha = torch.rand((real_images.size(0), 1, 1, 1)).to(real_images.
        device)
    alpha = alpha.expand_as(real_images)
    interpolation = alpha * real_images + ((1 - alpha) * fake_images).
        requires_grad_(True)
    d_interpolated = self.D(interpolation)
    gradients = torch.autograd.grad(
        outputs=d_interpolated,
        inputs=interpolation,
        grad_outputs=torch.ones_like(d_interpolated),
        create_graph=True,
        retain_graph=True,
    )[0]
    gradients = gradients.view(real_images.size(0), -1)
    grad_norm = gradients.norm(2, 1)
    return torch.mean(torch.square(grad_norm - 1))
```

Listing 2: Gradient Penalty

For the generator, the goal is to minimise the Wasserstein distance between the real and generated images, as seen in formula 1. However, this can be simplified, as shown by (Arjovsky, Chintala, and Bottou 2017, 4):

$$\min_G \mathbb{E}_{x \sim \mathbb{P}_r}[D(x)] - \mathbb{E}_{x \sim \mathbb{P}_g}[D(\tilde{x})] \equiv \min_G -\mathbb{E}_{x \sim \mathbb{P}_g}[D(\tilde{x})] \tag{9}$$

As done in the work by Su et al. 2018, to this loss the weighted L1 loss between the predicted and the target image, which is either the ground-truth normal or depth map, is added to the Wasserstein distance. This leads to the final loss function:

$$loss_{\text{generator}} = -loss_{\text{fake}} + L1(fake, target) * weight_{\text{L1}} \tag{10}$$

3.2.2 Network Training

The models of the used datasets, which are ShapeNet3d (Chang et al. 2015) for the comparison and a combination of the ABC dataset (Koch et al. 2019) and the Thingy10k (Zhou and Jacobson 2016 for the ablation study as mentioned in Section 4.1.1 and 4.2.1 respectively, are filtered and pre-processed. The images are processed using Mitsuba 3 (Jakob, Speierer, Roussel, Nimier-David, et al. 2022). The camera is positioned using a 225° azimuth angle and with a 30° elevation. All depth and normal images are rendered using 256 rays per pixel, which was found to result in a detailed image.
For the rendering of the normal maps a custom implementation returning the shading normals is used and for the depth maps a custom depth implementation is used. The shading normals are chosen over the geometry normals since the interpolation of the normals provides a smoother value distribution over the surface, which makes the loss computation more accurate. For the creation of the maps the custom depth resp. normal integrator, described in greater detail in Section 3.3, are used. Although Mitsuba 3 provides the same functionality in their AOV integrator and the re-parameterised rays are not needed for a non-differentiable rendering, the AOV integrator or the depth integrator of Mitsuba 3 could be used as well. But since the reparam integrators are used in the DR process using those for the rendering of the target images ensures that the maps are as similar as possible regarding background colours and tensor shape. Furthermore, since the depth values are returned by the depth integrator resp. the AOV integrator is not normalised and is in fact the absolute distance to the camera, this can lead to some problems within the training process of the neural network: The object could, in theory, have very similar values as the background, which makes the learning process more difficult. Masking the image would be possible, however, due to the similar values, it is very likely that the object itself is outlined by a darker outline that has slightly greater values than 0 and could also be found in the object itself, but belongs to the background. Therefore the created integrator returns 2 if the ray hits no object in order to create a clear boundary between the object's values and the background values. The value 2 can be replaced by any other value, however, depending on the normalisation of the model certain values can be interpreted as object values or the threshold needs to be chosen carefully, e.g. using 0, which is used in the depth interpreter provided by Mitsuba 3, can introduce difficulties if the rendered model is close to the camera. The resulting image is then masked using the distance between near and far planes as the threshold. If the pixel value is higher than the threshold, the value assigned is 1, otherwise, it is the pixel value divided by the distance between the near and far planes. This process ensures a uniform background, which is not necessarily provided by the initial rendering where the values can vary by small float values, which in turn can introduce errors in the mesh generation. Furthermore, the object values are normalised while preserving the placement wrt. the camera placement and the size ratio of the object's parts.
The maps are then saved as OpenEXR files since conversion to a standard image format like PNG would cause data loss due to the conversion from float to int. Furthermore, when using converted image data in the differentiable renderer the differentiated renderings also would have to be converted, which would have added an extra unnecessary step, which also introduced slight inconsistencies with the missing data.
For the sketches, Mitsuba 3 is used to render a greyscale image, which is then processed by the

Canny Edge detection. Since when using a small output format, like 256x256 pixels as done for the normal and depth map, artefacts like broken lines or merged parallel lines occurred, the images are rendered with an output format of 1024x1024 pixels using 256 resp. 100 sample rays per pixel. This different sample size is caused by the size of the ABC dataset compared to the dataset used for the ablation study. Using more sample rays to render an image, would have resulted in an unreasonably long rendering time for the whole dataset while the quality of the sketches did not change significantly. Therefore, the sample size used to render the Thingy10k/ABC dataset is deemed impractical for the ABC dataset and the number of used sample rays is adjusted. In order to generate a sketch from the rendered models, a Gaussian Blur, as well as the Canny Edge detection, are applied and the resulting lines are dilated before resizing the image to 256x256 pixels. Resulting in colour artefacts being removed by thresholding the image. For all image processing operations, the OpenCV2 library (Bradski 2000) is used.

The generated images are split into training, validation and test datasets, which are used for the operations described in the next paragraph. The concrete partition is described in Sections 4.1.1 and 4.2.1.
In order to utilise the generated datasets, custom data loaders are implemented. The implementation of the data loader used for the Thingy10k/ABC dataset is straightforward: The given input and target folder are iterated and the paths are stored in a list. During the training process, for the given indices those lists are used to load the images and returned with their respective paths to the training, test or validation function. It is also possible to only get the input image and its path, which can be utilised during the test process, as described in the following paragraph.
The data loader of the ShapeNet3d dataset is inspired by the one used by Liu et al. 2019: Since this dataset is divided into classes, the training process needs to include equal amounts of renderings for every class. This number for the training process is given by the user and the size of the validation set is computed based on the ratio of the training set size and the validation set size given by the state-of-the-art method, which this work is compared to. In the data loader, the paths are not stored in a list but in a target and an input dictionary which are composed of lists of paths and their respective classes as keys. During the training process, the classes are chosen at random, which is used to get the list of the respective images of the directory. Then, an image at a random index within that list is returned. For the test method, the same method used for the Thinky10k/ABC dataset is used since this should process the whole dataset, however, the implementation needed to be adjusted to the composition of the dataset, which is different in the two datasets as described in Sections 4.1.1 and 4.2.1.
While this implementation has the disadvantage of not producing the same models for each training, even if the parameters are all equal, it works for this dataset, since within the classes the objects are very similar and of one type. While there are alternatives which ensure that all classes are trained equally, e.g. by using a probability function suited for that cause, this approach worked for the purpose of this thesis.

The training itself is conducted on 4 Nvidia A40 GPUs, using the Distributed Data Parallel

strategy. The input consisted of the sketch images, which served as input images, and either the normal or the depth images, which were used as the target images. For the training input, the integer values of the input sketch were transformed from a range of [0,255] to float values with a range of [-1,1] and the float values of the depth map were transformed from a range of [0,1] to [-1,1]. The normal map is not transformed, since the output of the normal rendering process is already mapped [-1,1]. Furthermore, for the depth map only 1 channel is used, since all 3 channels have the same value and using only one channel in the training process omits the problem of variance in pixel values between channels.
At the end of every training process, a validation step is performed, where the L1 loss between the predicted image and the target image is computed. Those are transformed to a 0-255 value range and logged in order to follow the learning process in a visual way in addition to the logged losses as well as to provide some information about the quality of the resulting images without having to generate images using the trained models. Those checkpoints are also created during the validation step. A total of 6 checkpoints are saved during the training process, five depending on the smallest L1 loss of the validation and one after the last step, which concludes the training process. Once the training is completed, a test step is performed where the test set is fed into the model with the lowest validation loss and depth resp. normal maps are created from that, both in lossless formats as well as in png formats for better debugging. In addition to that, if a target folder for the test set exists, the output and the target png are merged into one image for better comparison, otherwise only the predicted image is created. For predicting depth images, an additional step includes transforming the output back to 0-1.
This test function is used in the framework to employ the trained models to create the depth resp. normal maps from the given user input. A target folder with corresponding target depth resp. normal maps is given, and the output of the prediction and this target are concatenated into one png for better comparison. If no target image is given, the predicted image is output on its own. The test step can be performed on any number of GPUs. Like in the training step, the strategy is Distributed Data Parallel and mixed precision is also used.

3.3 Mesh Reconstruction using Differentiable Rendering

In the final step, the base mesh and the flood-filled sketch returned by the base mesh determination module, as described in Section 3.1, and the predicted by the trained neural networks normal and depth map, as described in Section 2.4, are used in a differentiable renderer to construct the output mesh. This is done by offsetting the initial vertex positions, which are obtained via the scene traverse method of Mitsuba 3, of the input mesh by certain values. Those values are optimised iteratively. For the loss, which is backpropagated with the associated gradient to the optimiser, 5 losses are computed in each iteration and a weighted sum is computed:

$$loss = loss_{\mathrm{si}} * weight_{\mathrm{si}} + loss_{\mathrm{d}} * weight_{\mathrm{d}} + loss_{\mathrm{n}} * weight_{\mathrm{n}} + loss_{\mathrm{sm}} * weight_{\mathrm{sm}} + loss_{\mathrm{e}} * weight_{\mathrm{e}} \tag{11}$$

The depth loss ($\mathrm{loss_d}$), the normal loss ($\mathrm{loss_n}$) and the silhouette loss ($\mathrm{loss_{si}}$) are determined by the differentiable rendering of the deformed base object. For this process, Mitsuba 3 is used with 16 sample rays per pixel, since this number of rays proved to gather enough information

regarding the scene parameters to result in reasonable deformation and produces a rather small gradient, which is beneficial for the time it takes to backpropagate the loss. However, in order to speed up the deformation process, this number can be reduced but can impact the quality of the deformed mesh since less information can be retrieved from the resulting images. Furthermore, it is mandatory that the masking process for the depth map, which is needed in order to separate the object from the background as described in Section 3.2.2, is excluded from the backpropagated gradient associated with the depth map. This is done by utilising the suspend_graph function for the computation of the mask. If this is not excluded, this computation is captured in the graph and backpropagated which leads to the mesh getting distorted in undesired ways leading to faulty normals, which in turn can lead more likely to the normal map renderer returning nan values and ultimately to a failure of the deformation process. Although faulty normals should not occur when this is taken into account, the rendered normals are still checked for invalid values in order to log the failing mesh and the associated meshes. This is included for debugging purposes.
Mitsuba 3 was chosen over the others described in Section 2.2, since the possibility of adding plugins or other code parts and integrating them with the existing framework is more accessible than in the other renderers. Although other libraries, e.g. Pytorch3d, already provide losses like the smoothness loss or the edge loss, which are discussed in the following paragraphs, the benefit of Mitsuba 3 is that it is low level compared to the other libraries, which allows for own extensions to be easier implemented. Furthermore, custom rendering techniques, which are referred to as integrators in Mitsuba 3 and its predecessors, are fairly straightforward to implement. Each integrator solves the light transport equation, which is an essential part of the Monte Carlo method that is used in Mitsuba 3 as described in Section 2.2. However, since the provided integrators for the AOVs, which include the normals and depths, are not inherently differentiable and there is no silhouette renderer, this needed to be implemented for this thesis. In order to implement differentiable integrators, the init as well as the reparam function, which makes the re-parameterisation of the used integrator possible and is needed to make it differentiable as discussed in Section 2.2, are constructed according to already existing reparam integrators. The init function includes the following parameters:

- The maximal depth, to which the re-parameterisation is applied.
- The number of auxiliary rays used to evaluate the parameterisation, which is described in Bangaru, Li, and Durand 2020, 8.
- A probability distribution parameter, which specifies the sampling positions of the auxiliary rays.
- A weight exponent is used for weighing the integral along the auxiliary ray.
- Whether antithetic sampling is used, which is a variance reduction method used in Monte Carlo sampling and also adopted by Bangaru, Li, and Durand 2020, 11 in their work as an option.

The other addition to the classic depth resp. normal integrator used in Mitsuba 3 is the reparam function, which returns a re-parameterised ray. These two functions are taken from other built-

in reparam integrators and combined with the depth resp. shading normal computation method, which is used in the AOV integrator. This part uses Monte Carlo sampling and adds either the shading normal or the depth value of the hit value to a spectrum, which is then returned. If a ray hits nothing, a predefined value, which in this case is 2 enables the masking processes as described in Section 3.2. Once the differentiable depth resp. normal maps are rendered, the L1 loss is used in order to get the difference between the GT and the rendered maps.

For the rendering of the silhouette, another integrator is implemented, which returned 0 if the object is hit and 1 otherwise. Although a differentiable renderer is not mandatory to get the silhouette of the objects since masking the depth or normal map would also be an option, this is implemented because the division of the renderings into independent steps seemed to be a reasonable step in order to make debugging and the evaluation of contributions of the respective losses and therefore the choice of the weights easier. Once the silhouette is rendered, the Intersection over Union metric is used to compute the difference between the flood-filled image acquired from step 3.1.1, which is used as the target image, and rendered image. This metric is popular when computing silhouette loss because it is a straightforward way to compute similarities between two binary images. Therefore, many researchers have included this step in their works, e.g. Xiang et al. 2020 and Liu et al. 2019.
The computation process is depicted in the function Intersection over Union in Listing 3: First, the horizontal and the vertical sum of the two tensors representing the rendered and the input silhouette are computed, which declares the intersection of the 2 images. The union is the horizontal and vertical sum of the added images subtracted by the multiplied images. For the horizontal and vertical sum, the torch implementation of the sum function is employed, since Dr.Jit reduces tensors over dimension 0, which in this case leads to wrong results. This is represented by the torch_add function depicted in Listing 3. Once the intersection and the union are computed, the final computation returns a value describing how similar the intersection and the union values are. To avoid a division by 0, ε is added, which is 1e-6 in this work. If the similarity divided by the elements in a column or row of the input tensors approaches zero, 1 is returned, which is the desired value for the silhouette loss.

```
@dr.wrap_ad(source='drjit', target='torch')
def torch_add(self, x):
    dims = tuple(range(x.ndimension())[1:])
    return torch.sum(x, dim=dims)

def iou(self, predict, target):
    intersect = self.torch_add(predict * target)
    union = self.torch_add(predict + target - predict * target)
    intersect_shape_x = intersect.shape
        x = 1.0 - dr.sum(intersect / (union + 1e-6)) / intersect_shape_x[0]
    return x
```

Listing 3: Implementation of Intersection over Union for the smoothness loss and the torch sum function used to compute the sums used in the IoU computation.

Like in Xiang et al. 2020, for the edge loss ($loss_e$) the As-Rigid-As-Possible energy is used:

$$loss_e = \frac{1}{n_e} \sum (e_{initial} - e_{current})^2 \tag{12}$$

In order to do this, the vertex indices associated with an edge are stored in pairs in a list beforehand. This is done by traversing the faces of the mesh obtained by the scene traverse function of Mitsuba 3 and storing the vertex indices of the edges. Using those indices, the vertex positions are gathered from the deformed vertex position list and the length between two vertices belonging to an edge is computed. These resulting edge lengths are described by the variables $e_{initial}$ and $e_{current}$ in equation 12. The $e_{initial}$ are computed using the initial vertex positions before the deformation loop starts and the $e_{current}$ is computed in every iteration using the deformed vertex positions. As seen in the formula above, the L2 distance of the $e_{initial}$ and the $e_{current}$ are then multiplied by 1 divided by the number of edges in order to get the edge loss. As mentioned in 2.1, this loss is employed to regularise the edge lengths and therefore keep the mesh structure consistent.

For the smoothness loss ($loss_{sm}$), which is inspired by the work of Kato, Ushiku, and Harada 2018 as described later in this paragraph, the needed vertex indices are also pre-processed beforehand in order to minimise the computation time of the loss during the deformation iterations. This is done in two steps: For the first step, the two face indices of the faces with 2 same edge vertices are stored in a tuple in a directory with the edge vertices as keys. In a second step, for which the implementation is depicted in Listing 4, this directory is iterated and for the adjacent face indices the 4 vertices are stored in one list based on the vertex they belong to and whether they are the x, y or z coordinate. This appending process is done in a closure function since this reduces code and this function must not be globally accessible. The x, y and z lists belonging to a vertex are concatenated in a single list once the iteration process is completed. v1 and v2 are the lists containing the vertex indices that are contained by both faces, while v3 and v4 consist of the vertex indices located in only one of the two faces. These containing vertex lists can be used in the deformation loop to gather the current vertex positions by using the gather function provided by Dr.Jit. The smoothness loss computation, which is described in the following, is then executed two times, using the gathered vertex indices from the lists v1 and v2 both times and the vertices from the lists v3 and v4, which are contained by only one of the faces, only once. Using this prepossessing and list-based approach is performing much better when using the CUDA-based approach, due to the parallel computing capability of the GPU. This reduces the computation time significantly compared to a sequential approach, which is crucial due to the usage of a large number of iterations to deform the mesh.

```
1 def pre-process_smoothness_params(self, edge_vert_faces, face_indices):
2     def generate_vertex_list(self, vertex_list_x, vertex_list_y,
          vertex_list_z, vertex_index):
3         vertex_list_x.append(3 * vertex_index)
4         vertex_list_y.append(3 * vertex_index + 1)
5         vertex_list_z.append(3 * vertex_index + 2)
6     v1_x, v1_y, v1_z = [], [], []
7     v2_x, v2_y, v2_z = [], [], []
```

```
    v3_x_f1, v3_y_f1, v3_z_f1 = [], [], []
    v3_x_f2, v3_y_f2, v3_z_f2 = [], [], []

    for key in edge_vert_faces:
        curr_faces = edge_vert_faces[key]
        vert_idx_face1 = [face_indices[0][curr_faces[0]],
                          face_indices[1][curr_faces[0]],
                          face_indices[2][curr_faces[0]]]
        verts_idx_face2 = [face_indices[0][curr_faces[1]],
                          face_indices[1][curr_faces[1]],
                          face_indices[2][curr_faces[1]]]
        joined_verts = list(set(vert_idx_face1).intersection(
            verts_idx_face2))
        generate_vertex_list(v1_x, v1_y, v1_z, joined_verts[0])
        generate_vertex_list(v2_x, v2_y, v2_z, joined_verts[1])
        v3_face1 = (set(vert_idx_face1).difference(joined_verts).pop())
        generate_vertex_list(v3_x_f1, v3_y_f1, v3_z_f1, v3_face1)
        v3_face2 = (set(verts_idx_face2).difference(joined_verts).pop())
        generate_vertex_list(v3_x_f2, v3_y_f2, v3_z_f2, v3_face2)
    v1 = [v1_x, v1_y, v1_z]
    v2 = [v2_x, v2_y, v2_z]
    v3_face1_idx = [v3_x_f1, v3_y_f1, v3_z_f1]
    v3_face2_idx = [v3_x_f2, v3_y_f2, v3_z_f2]
    return v1, v2, v3_face1_idx, v3_face2_idx
```

Listing 4: Implementation of pre-processing operation for smoothness function. For two faces the vertex indices used as edge points for both vertices are stored in v1 and v2. Vertex indices belonging to only one face are stored in v3 resp. v4.

The smoothness loss is itself the same as used by Xiang et al. 2020 and Kato, Ushiku, and Harada 2018:

$$loss_{\mathrm{s}} = \sum_{(f_{\mathrm{i}}, f_{\mathrm{j}} \in F)} (1 + cos < f_{\mathrm{i}}, f_{\mathrm{j}} >)^2 \tag{13}$$

where f_{i} and f_{j} are two adjacent face pairs. As described above, in two steps the current vertex indices of adjacent triangles and the angle between them are computed. A depiction of the edge and vertex of the faces values can be found in Figure 5.
As seen in Listing 5, for each face the adjoined edge a and another edge b are computed, using the given vertex data v1, v2 and v3 or v1, v2 and v4. Then, the vector c, which describes v2 to the edge of a projected point v3, is computed. This is done using scalar projection of one edge b onto the edge a. Returned are then the distance b to c, which describes the dashed line in Figure 5 and its squared magnitude.
Once cb and the length of that are computed, the angle between the two faces can be computed using:

$$cos(\theta) = \frac{cb\_f1 \cdot cb\_f2}{mag_{\mathrm{cb\_f1}} * mag_{\mathrm{cb\_f2}}} \tag{14}$$

which is the equivalent of the second term of the addition in equation 13. Like in the computation of the IoU, *epsilon* of 1e-6 is added if there is a chance of a division by zero. As described in equation 5, the result is added to 1 and squared before computing the final sum

of the resulting list, which serves as the smoothness loss. This implementation is a simplified version of the one Kato, Ushiku, and Harada 2018 did in their work, which is implemented in their project repo[2]. However, their implementation is more complex and adds the epsilon value multiple times in the various steps, but is not explained nor validated in their work. Therefore, this framework uses an intuitive, simpler version with only the necessary eps.

```
def smoothness_helper(self, v1, v2, v3):
    a = v2 - v1
    b = v3 - v1
    sqr_magnitude_a = dr.sum(dr.sqr(a))
    dot_ab = dr.sum(a * b)
    l = dot_ab / (sqr_magnitude_a + 1e-6)
    c = a * l
    cb = b-c
    l1_cb = dr.sqrt(dr.sum(dr.sqr(cb)))
    return cb, l1_cb

cb_1, l1_cb_1 = self.smoothness_helper(v1, v2, v3_face1)
cb_2, l1_cb_2 = self.smoothness_helper(v1, v2, v3_face2)
cos = dr.sum(cb_1 * cb_2) / (l1_cb_1 * l1_cb_2 + 1e-6)
smoothness_loss = dr.sum(dr.sqr(cos+1))
```

Listing 5: Implementation of computation of values used for the smoothness loss.

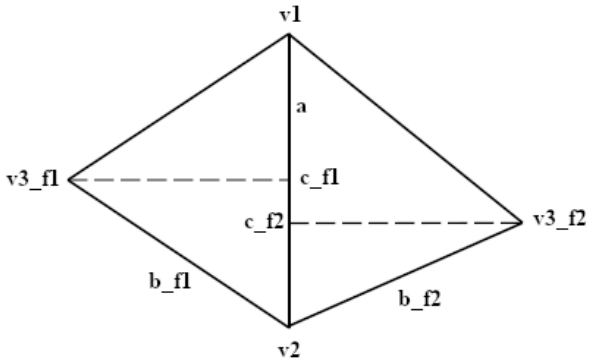


Figure 5: Depiction of important values used for smoothness loss.

4 Evaluation

In order to evaluate the implemented system two studies are conducted: A comparison with a state-of-the-art model and an ablation study. The first intends to rate the output mesh of the proposed pipeline compared to the output of an established state-of-the-art model as well as to identify certain factors that could be improved in future work. The ablation study is conducted in order to prove that the incorporation of the depth map and basing the base mesh used for

2. https://github.com/hiroharu-kato/mesh_reconstruction/blob/master/mesh_reconstruction/loss_functions.py

the deformation process on the genus of the sketch do improve the reconstruction process. The results of the studies are presented in the following. Finally, the limitations of this pipeline and possible ways of overcoming them are discussed.

4.1 Comparison with State-of-the-Art Model

To establish how the proposed model performs in comparison to state-of-the-art models, this thesis' system is compared with the Neural Mesh Renderer by Kato, Ushiku, and Harada 2018. This is done in both a quantitative way, using metrics to measure the similarity between the ground-truth model and the generated models and evaluating the results for both the Neural Mesh Renderer and the proposed pipeline, as well as a qualitative way, where the resulting objects where examined based on how they appear and possible geometric flaws.
In the following section, the experimental setup is explained including the used dataset, parameters, comparison model and evaluation images. Thereafter, the results of the qualitative and quantitative study respectively are presented.

4.1.1 Experimental Setup

Dataset. The dataset used for the comparison with a state-of-the-art model is based on ShapeNet3d v1 (Chang et al. 2015). This is chosen over the dataset used in the ablation study, since attempts to train models on this dataset are unsuccessful, which can be attributed to the variety of the dataset. Therefore, the standard dataset is utilized, which most of the models in this field used to train their methods. This however meant that the impact of the proposed improvement on the predefined base model based on the genus could not be evaluated for most sketches, since the classes offered by ShapeNet consist of models which mostly have genus 0. Furthermore, the ShapeNet data had to be extensively pre-processed, since this proved to be unusable due to various problems like inconsistent normals, holes, and coincident or overlapping faces. Therefore, the pre-processed ShapeNet data provided by Xu et al. 2019 is used, and the remaining data is cleaned using the research by Xu et al. 2019 and Sin, Schroeder, and Barbič 2013. Although this caused artefacts in some meshes like missing or disconnected parts, the resulting renderings proved to be usable. Another option would have been to reconstruct the meshes from the dataset provided by Kato, Ushiku, and Harada 2018 using a technique like marching cubes (e.g. Lewiner et al. 2003).
After the pre-processing step that repairs the broken ShapeNet data, a second pre-processing step is applied before rendering the dataset images. This ensures that the mesh is watertight, and normalised, which in this case means that the diagonal of the bounding box has a length of 1 as mentioned in Section 3.1.2, at the centre of the coordinate system, has consistent normals and has no holes.
Since Kato, Ushiku, and Harada 2018 does not provide their exact split, the dataset has to be split according to the ratio used by Kato, Ushiku, and Harada 2018, which means that the datasets the Neural Mesh Renderer was trained on and the one used in this thesis might be different.
The resulting dataset consisted of a total of 13 classes and 43783 256x256 images, which are

not evenly distributed amongst the classes. The dataset is split into a sketch dataset, and a target dataset, which is split into train, test, and validation folders. Those contain folders for the classes, in which the sketches resp. normal or depth maps for the associated class are stored. The ratio in which the train, validation, and test images are split is 70:10:20, which is based on the split ratio of the dataset used by Kato, Ushiku, and Harada 2018. This results in 30643 training images, 4378 validation images and 8762 test images.

Map Generation Parameters. For the map generation training the randomised data loader for the ShapeNet3d dataset, which is described in Section 3.2.2, is used. From this dataset 782 training images are chosen at random for each training epoch and 112 validation images are used. The model is trained over 3000 epochs, using a batch size of 91. The learning rate is set to 2e-5. For the network optimiser, RMSProp is used, since those proved to be the most effective in the research by Arjovsky, Chintala, and Bottou 2017, 7, however, using the Adam optimiser would also have been an option since this performed the best in the study conducted by Gulrajani et al. 2017, 5775. The discriminator is trained five times more often than the generator since the WGAN structure demands the discriminator to be trained more than the generator, as discussed in Section 2.3. Based on experience and manual optimization, for the weight of L1 loss between the predicted and target image added to the WGAN loss in the generator, a factor of 500 is used. For the gradient penalty weight in the discriminator 10 is used, since this is the value successfully used by Gulrajani et al. 2017, 5770 in their experiments. This setup is the fastest tested while producing reasonable results, taking about a day to complete the training. The networks could have been trained for longer to produce better output, however, the estimated time for that would not warrant the predicted improvement. Therefore, the checkpoints with the lowest validation error of the 6 options produced during the training to predict the maps in the proposed pipeline are selected. The chosen checkpoints are the ones produced at epoch 2843 with a validation loss of 0.025612 for the depth map prediction and produced at epoch 2724 with a validation loss of 0.027100 for the normal map prediction.

Comparison Model. The model chosen for this comparison is the Neural Mesh Renderer by Kato, Ushiku, and Harada 2018. Although this is one of the older networks existing in this field, it is still regarded as state-of-the-art. Furthermore, the authors provided their pre-trained models in the repository[3] associated with their study and the information on how to use those, which is not the case for all potential comparison models. The difference to the proposed version is that they use RGBA images in their work instead of sketches and they produce intermediate multi-view silhouettes from a given input sketch. Furthermore, the dataset they trained their model on, is the voxelised version of ShapeNet3d, which for their approach should not make a difference since their deformation process is based on silhouettes rather than normal and depth maps as well as a small input image size which includes less information about details. Therefore, it is expected that the Neural Mesh Renderer is able to better capture the overall shape of the input image, while the proposed method should output more details in the model, which is also expected to be reflected in the results of the quantitative evaluation. In addition to that, they

3. https://github.com/hiroharu-kato/mesh_reconstruction

trained their model on the classes for each model, resulting in a pre-trained model for each class, while the thesis approach uses one pipeline and the pre-trained depth and normal map generator for each object independent of their class. This likely makes their output for the classes more accurate, however, a more generalised approach to reconstructing models from image input is provided by using one solution for all options. Despite the differences in the approaches in the reconstruction techniques, the output of the study will provide valuable information on how the proposed technique performs against an established method and which parts of the work by Kato, Ushiku, and Harada 2018 might be worth investigating and incorporating in future work.

Pipeline Parameters. The parameters set for the pipeline refer to the mesh deformation module, since the other relevant parameters are used in the map generation module and therefore described in the Dataset and Map Generation Parameters paragraph of this section. The deformation process is performed over 4000 epochs, which takes about 6 hours per mesh to complete. The Adam optimiser is configured with a learning rate of 0.0001, beta1 of 0.9 and beta2 of 0.999. While those values are not the ideal parameters for all models due to the variance and therefore the different requirements to the optimization parameters in the dataset, they are chosen since they did work for all input sketches without producing major errors as well as producing reasonable output meshes. The weights mentioned in Section 3.3 are set to weight_d 0.002, weight_n 0.002, weight_sm 0.02, weight_si 0.9 and weight_e 0.9. Therefore, the setup of the weights is similar to Xiang et al. 2020 with the exception of the addition of the depth weight for the depth loss and the weight for the smoothness parameter. The latter parameter is not 0.01, as used in Xiang et al. 2020, due to some errors occurring in the normal renderings and the resulting problems which became apparent in the ablation study. This is further explained in Section 4.2.2.
Since Kato, Ushiku, and Harada 2018 used images with a size of 64x64, the meshes are reconstructed using this thesis' pipeline with both the setup with a 256x256 input image size, as recommended with this framework, and the 64x64 input image size. This is done to investigate how the framework and the output are affected by this change as well as to compare the output of the state-of-the-art method with the parameters as well as with theirs in case that has a huge impact on the reconstruction.

Evaluation metrics. The Intersection over Union metric and the Chamfer distance are evaluated for the models reconstructed by the proposed variant and Kato, Ushiku, and Harada 2018. These metrics are chosen since they are the most commonly used metrics to compare meshes and using various metrics rather than one offers a more objective result, since different methods may respond differently to certain features in the output meshes.
Since the computation of the Chamfer distance is designed for point clouds rather than for triangle meshes, 10000 points are sampled on the ground-truth and the output object, resulting in the point sets A and B. Those are used in the following equation to compute the Chamfer distance:

$$d_{\text{ch}}(A,B) = \frac{1}{|A|} \sum_{(p_\text{i} \in A)} \underset{(p_\text{j} \in B)}{min} ||p_\text{i} - p_\text{j}||_2^2 + \frac{1}{|B|} \sum_{(p_\text{j} \in B)} \underset{(p_\text{i} \in A)}{min} ||p_\text{j} - p_\text{i}||_2^2 \tag{15}$$

While this metric is a good determinator of whether there are excess or missing parts, this makes

it very sensitive to outliers. Therefore, the IoU metric is also used. While the basic computation of this is already explained in Section 3.3, since the evaluation is done on 3d objects 10000 SDF values per mesh are computed. Those are then used in logical operation in order to determine the Intersection and the Union, similar to the implementation presented in Listing 3.
The results of the comparison are presented in the quantitative study in Section 4.1.3.

Evaluation Images. For the evaluation, the sketches utilised as input for the proposed pipeline are chosen from the test part of the dataset used in the map generation part, for which the generation is illustrated in Section 3.2.2, and the split is described above. A rendered image for each class (aeroplane, bench, dresser, car, chair. display, lamp, speaker, rifle, sofa, table, telephone, vessel) in the ShapeNet3d v1 dataset is used, resulting in 13 evaluation sketches, which are depicted in Figure 7.

Figure 6: Images used as input for the pre-trained models of Kato, Ushiku, and Harada 2018 256x256.

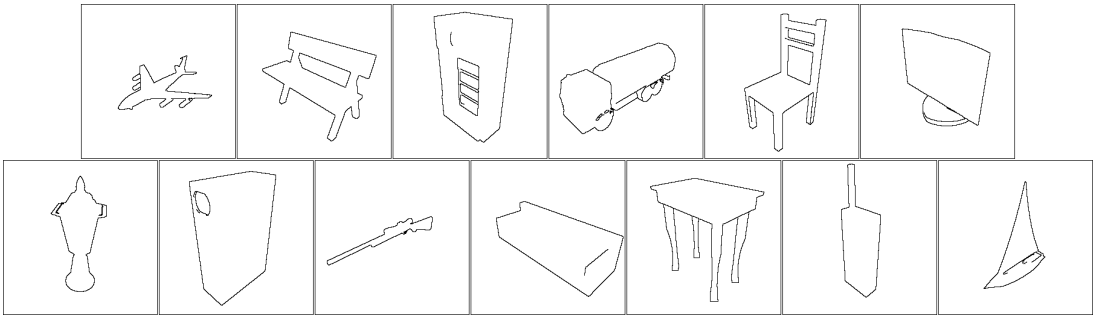


Figure 7: Sketches used as input for this thesis' pipeline.

The same models are used to generate the evaluation images for this thesis' pipeline as well as for the comparison models. Since the dataset split by Kato, Ushiku, and Harada 2018 is not disclosed in their work, there is no guarantee that the split for this thesis resembles the one they used in their research. For the renderings themselves, the direct integrator provided by Mitsuba 3 is employed, which is also utilised to render images to apply the Canny Edge detection to generate the sketches, as described in 3.2.2. However, for these images, a point light emitter is used instead of a constant emitter as done with the temporary rendering for the proposed setup

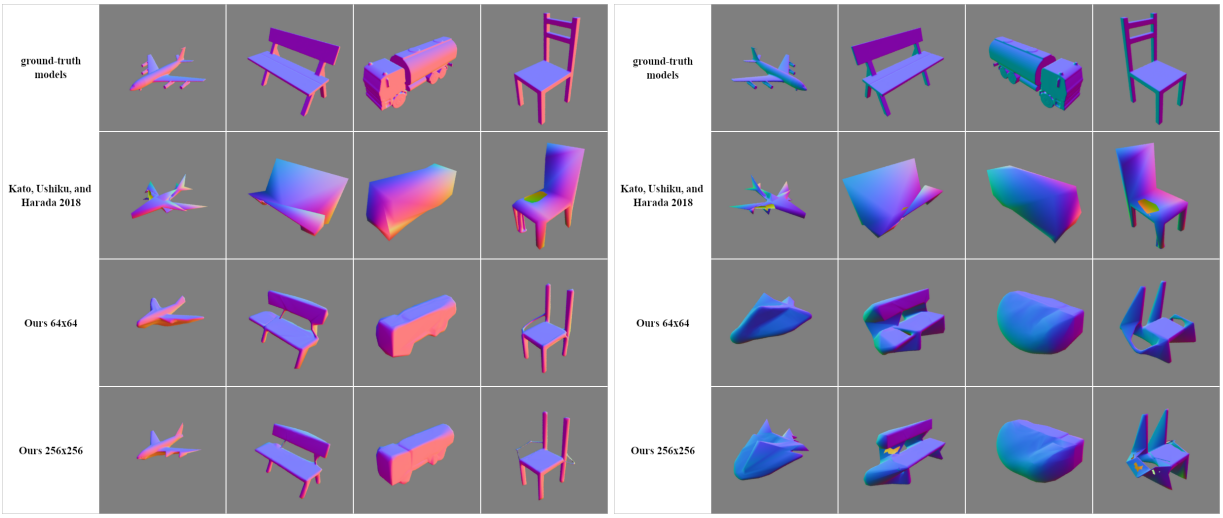
in order to equip the RGB part of the images with a black background instead of a white one, since that causes problems with the pre-trained models by Kato, Ushiku, and Harada 2018. It needs to be acknowledged, that this is not the only way to ensure a black background in those images, however, since these images work for the comparison it is used in this setup. To the RGB image rendered with the direct integrator, an inverted silhouette image rendered with the silhouette integrator generated for the mesh deformation module, as described in Section 3.3, and added to the RGB layers to form the alpha layer. The finished and assembled images are 64x64, unlike the 256x256 input sketches, and are rendered from the same viewpoint as the sketches, which is possible since Kato, Ushiku, and Harada 2018 used multiple angles in their training. This meets the requirements to be used in the model by Kato, Ushiku, and Harada 2018 and the rendered images are depicted in Figure 6.

4.1.2 Qualitative Evaluation

As already determined in the quantitative evaluation there are differences between the outputs by the proposed pipeline and the ones by Kato, Ushiku, and Harada 2018. The values computed using the Intersection over Union and Chamfer distance indicate that the methods overall produce faulty meshes. Furthermore, it can be assumed that the main difference between the proposed pipeline and the state-of-the-art method is that the network by Kato, Ushiku, and Harada 2018 is better at reproducing the general shape of the mesh, while this thesis' framework is better at reconstructing details. In order to confirm these theses and to further identify qualitative differences in the output objects, in the following the reproduced meshes are compared and evaluated.

The first difference that needs to be acknowledged is the difference in the normals of the reconstructed meshes, which can be seen in 8. Here are examples depicted that indicate that the normals rendered from the same viewpoint used to render the input images are less flawed in the meshes produced by the proposed pipeline, which is portrayed in Figure 8a. This is caused by the usage of the normal loss in the loss function, which takes the loss between the predicted normal and the normal of the deformed mesh and is consistent for all deformed meshes. This is supported by the similarity between the predicted and the rendered normals of the meshes used in the ablation study in Figure 13.
If the meshes are rendered from different viewpoints, e.g. like in Figure 8b, the models also display some faulty normals. This is caused by the same problem Kato, Ushiku, and Harada 2018 has, namely that the deformation is not supervised wrt. the normals from that viewpoint, which leads to inverted faces. However, the overall flaws in these meshes are very minimal, and the value distribution of the meshes produced by the proposed framework seems to be closer to the one by Kato, Ushiku, and Harada 2018. Therefore it can be assumed that the proposed pipeline is superior to the Neural Mesh Renderer in terms of correctly reproducing the normals of the meshes and avoiding inverted faces.

Another difference that was assumed at the beginning of this evaluation is that the proposed method is better at capturing and reconstructing the details of the given input meshes. This can be seen in Figure 15, especially with the table and the dresser. While those details are not



(a) (b)

Figure 8: Normal values of reconstructed aeroplane, bench, car, and chair rendered from 4 different azimuth angles (225, 315, 45, 135) using a single view image (225° azimuth, 30° elevation) as input. The first line shows the ground-truth meshes, the second row the meshes generated by Kato, Ushiku, and Harada 2018 and the last rows the results of the proposed pipeline with 64x64 resp. 256x256 images as input. In (a) the same viewpoint the input images are rendered from, 30° elevation and 225° azimuth, are used, in (b) an azimuth angle of 315° for the camera position is used. The depicted meshes are normalised for the rendering process to fit in the field of view.

perfect, there are at least hints of them in the meshes. What is notable is that the input image resolution does not have a considerable impact on the reconstruction of Kato, Ushiku, and Harada 2018. This is demonstrated in e.g. the reconstruction of the aeroplane, where the details in the object, especially in the tail and on the wings, are similar to the ground-truth mesh as well as the reconstruction where a 256x256 sketch is used as input, whereas the reconstruction based on the 64x64 image lacks those. Due to that as well as the dissimilarity between the lamp and the bench shape it can be reasonably assumed that the Neural Mesh Renderer relies heavily on the classes and is able to only reconstruct multi-view images that are familiar to it. If the input shape is not as prevalent in the training set, the results will not resemble the GT. This is similar to the normal and depth map generators in this thesis, for which these problems are discussed in Section 5.2. Despite the proposed pipeline suffering from the problem introduced via the map prediction, it still can be argued that it is not as significant since the normal and depth map are a part of the generation process along with more regularisers and a loss. While the work of Kato, Ushiku, and Harada 2018 also includes the silhouette, smoothness and edge loss, the information from the normal and depth map is missing, making the shape less detailed and reliant on the class of the shape.

This reliance on the class is essential for the Neural Mesh Renderer since there is no other source regarding how parts that are not shown in the image look like. If that information is not

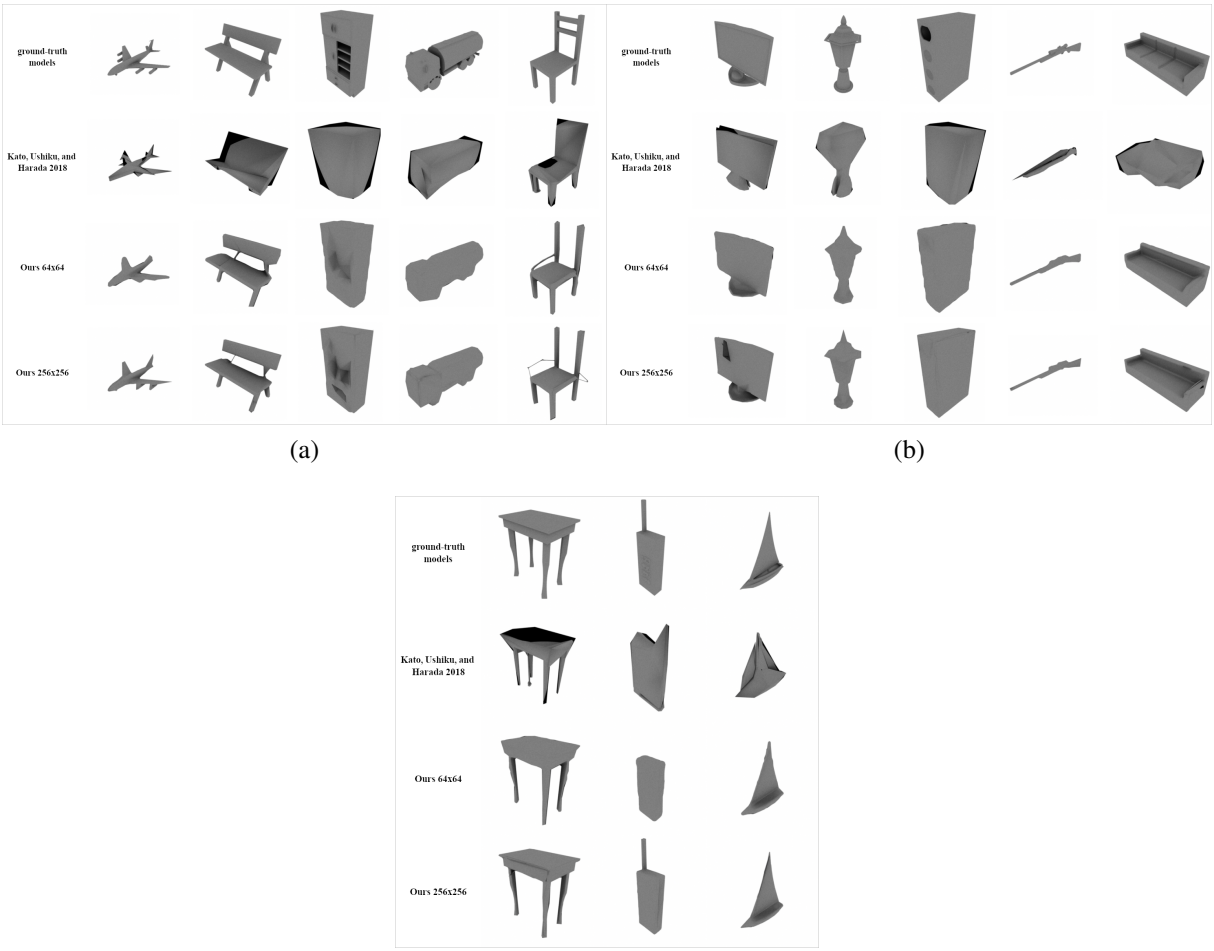
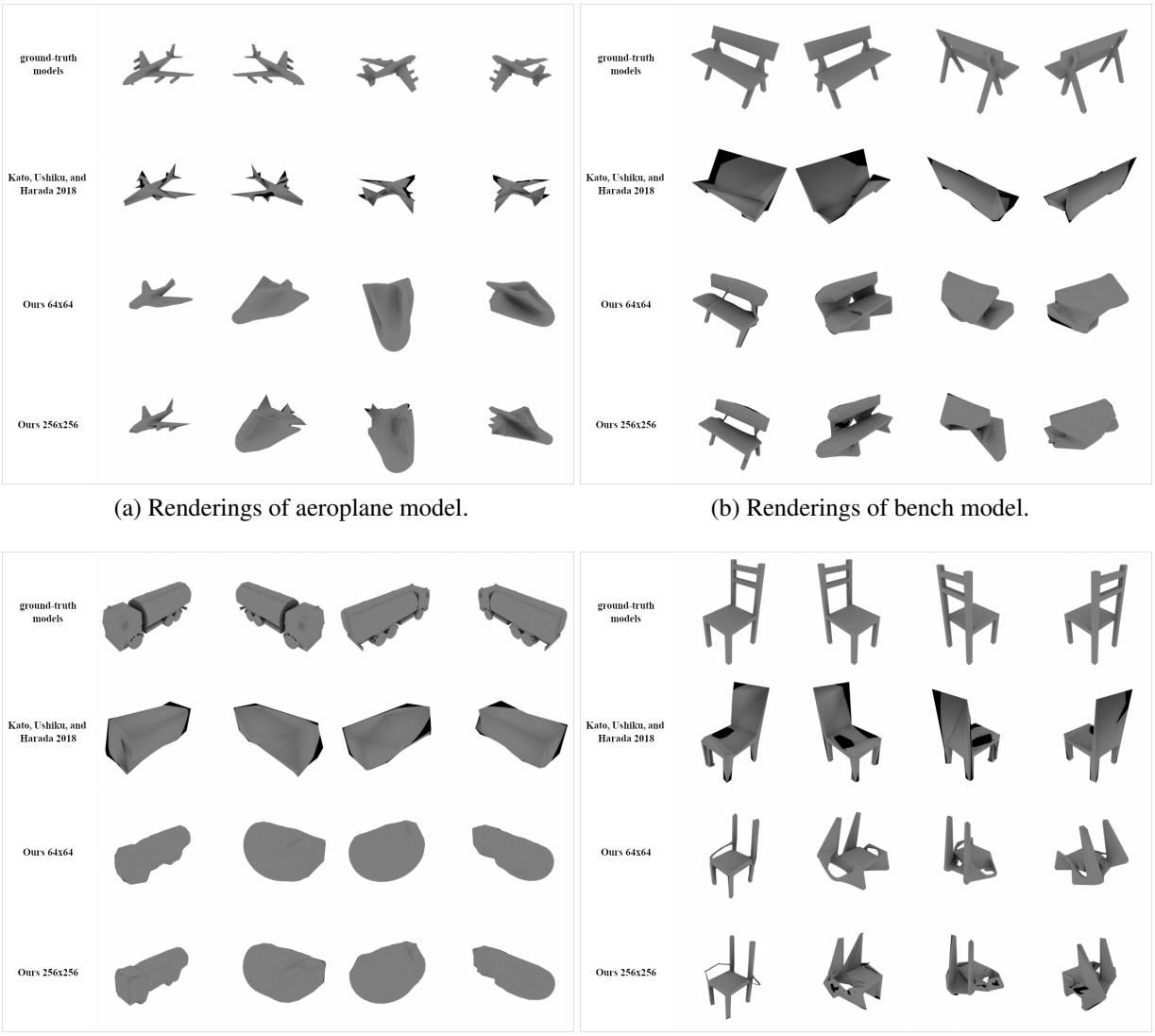


Figure 9: Output meshes of the different comparison variants rendered from the same viewpoint as the input images. The first line shows the ground-truth meshes, the second row the meshes generated by Kato, Ushiku, and Harada 2018 and the last rows the results of this research's pipeline with 64x64 resp. 256x256 images as input. For the rendering process, the meshes by Kato, Ushiku, and Harada 2018 are normalised to fit in the field of view.

available to the system, the prediction of the other views would not be possible and not be able to be reconstructed. This happens in the proposed pipeline and is also a great disadvantage of the system. As seen in Figure 10, the proposed system is not able to reconstruct mesh parts it has no information about, which is discussed more in detail in Section 5.3. This also aligns with the output of the Chamfer distance in the quantitative evaluation of this chapter, since the reconstructed model has more excess mesh parts. Furthermore, this confirms the thesis about the characteristics of the different systems, with the Neural Mesh Renderer being able to recover the general shape better, while the pipeline of this thesis is able to reconstruct details better.



(a) Renderings of aeroplane model.

(b) Renderings of bench model.

(c) Renderings of car model.

(d) Renderings of chair model.

Figure 10: Generated meshes using a single view input image (225° azimuth, 30° elevation) rendered from 4 different azimuth angles (225, 315, 45, 135). The first line shows the ground-truth meshes, the second row the meshes generated by Kato, Ushiku, and Harada 2018 and the last rows the results of the proposed pipeline with 64x64 resp. 256x256 images as input. The depicted meshes are normalised for the rendering process to fit in the field of view.

4.1.3 Quantitative Evaluation

Prior to evaluating the metrics for the qualitative evaluation the generated meshes had to be pre-processed in order to ensure that they have the same position, are normalised and rotated to ensure that they have the same orientation. To guarantee the first two requirements, the normalisation method ensuring that the diagonal of the bounding box of the mesh is 1 is used in conjunction with a transform of the centre of the object to the centre of the coordinate system.

The rotation part is only necessary for the objects produced by the models of Kato, Ushiku, and Harada 2018 since their orientation is different from the produced models using the proposed pipeline as well as the ground-truth models.
The results of this evaluation are presented in Tables 7 and 10. It needs to be recognised, that the greater image size did improve this thesis' system, which is attributed to an overall higher emphasis on the normal, depth, and silhouette loss as well as the details that could be reconstructed due to the higher resolution. However, this will be analysed, discussed, and compared to the output by Kato, Ushiku, and Harada 2018 in greater detail in 4.1.2, because this discrepancy becomes more apparent when analysed visually rather than via metrics.

Comparison Variant	**Output Model**													**Mean**
	Aeroplane	Bench	Dresser	Car	Chair	Display	Lamp	Speaker	Rifle	Sofa	Table	Phone	Vessel	
Kato, Ushiku, and Harada 2018	**0.143**	0.018	0.188	**0.444**	0.123	**0.282**	0.205	0.089	**0.125**	0.117	**0.246**	**0.281**	0.077	**0.180**
proposed pipeline 64x64	0.021	0.060	0.257	0.314	0.244	0.076	0.177	0.115	0.053	0.137	0.093	0.162	**0.127**	0.141
proposed pipeline 256x256	0.043	**0.088**	**0.270**	0.313	**0.308**	0.108	**0.211**	**0.131**	0.087	**0.150**	0.156	0.196	0.125	0.168

Table 7: **Results Intersection over Union**; higher values indicate better reconstruction results. The proposed method using 256x256 sketches as input as well as Kato, Ushiku, and Harada 2018 produced the best outputs an equal amount of times. For the remaining model, the proposed method scored better, with the 64x64 input sketch resulting in a slightly better result. Overall, the scores indicate little similarity between the base meshes and the reproduced meshes.

Intersection over Union. The results of the Intersection over Union metric do not indicate a clear superiority of the method by Kato, Ushiku, and Harada 2018 compared to the results of the Chamfer distance, as discussed in the next paragraph. From the 15 tested models, the proposed method of this thesis yielded better results in 7 cases, however, the mean over the results of all models is better for the Neural Mesh Renderer. While the values of the classes where Kato, Ushiku, and Harada 2018 scored the best are generally higher than the classes in the proposed method that produced the best results, the difference between the values of the proposed method and Kato, Ushiku, and Harada 2018 is for most classes significant. This leads to the conclusion, that there is a clear advantage for most classes to use one method over another. However, since there is no clear trend for which shapes one method outperforms the other, there is not really a conclusion as to when one method should be considered over another. A possibility that needs to be considered is the influence of the likely different training sets used Kato, Ushiku, and Harada 2018 and the proposed method, but that cannot be confirmed for certain. Therefore, the results of the qualitative evaluation need to be taken into account to determine the best application cases for the specific implementations.
However, the results in Table 7 indicate an overall inaccurate mesh reconstruction. This does not match the results in the original work of Kato, Ushiku, and Harada 2018, for which there are several reasons: They used more primitive meshes, more specifically the meshed version of the voxelization of the ShapeNet3d v1 dataset according to the dataset they provided in their project repository. Since they used also only 64x64 images as input, this mesh quality is sufficient for their studies, since details in the meshes as well as in the images are not prevalent and can therefore not be reconstructed. This is supported by comparing the 64x64 version to

the 256x256 version, where the higher resolution for most cases results in better scores for both the IoU and Chamfer distance. Using the reconstructed meshes as GTs will therefore result in worse metric outputs for the comparison than if the version by Kato, Ushiku, and Harada 2018 is used. It must also be acknowledged, that those simpler models could improve the quality of the output meshes of the pipeline, since the values of the normal maps are more unified, which can improve the prediction process, and the shapes are simpler, which can reduce error when it comes to reconstructing the details. This is however only an assumption that could be evaluated in future work. Another factor that could attribute to the overall worse result is that the test models used for the evaluation are not the same as Kato, Ushiku, and Harada 2018 used in theirs. In selecting the evaluation images there is a focus on having a variety of sketches in terms of their similarity to the training data. This can cause images, which are less familiar to the trained models by Kato, Ushiku, and Harada 2018 as well as the map generation module to work better for certain objects.

Comparison Variant		**Output Model**													**Mean**
		Aeroplane	Bench	Dresser	Car	Chair	Display	Lamp	Speaker	Rifle	Sofa	Table	Phone	Vessel	
ground-truth Values	abs	*59*	*93*	*135*	*111*	*89*	*100*	*78*	*166*	*40*	*106*	*102*	*78*	*70*	**94**
Kato, Ushiku, and Harada 2018	abs	**163**	1592	1254	**567**	**377**	**335**	**639**	1454	**309**	1254	**755**	**678**	**666**	**773**
	rel.	**2.763**	17.118	9.289	**5.108**	**4.236**	**3.350**	**8.192**	8.759	**7.725**	11.830	**7.402**	**8.692**	**9.514**	**7.998**
proposed pipeline 64x64	abs	1264	1008	1796	865	1011	1693	1303	**1140**	1046	1299	2152	1088	1011	1283
	rel.	21.424	10.839	13.304	7.793	11.360	16.930	16.705	**6.867**	26.150	12.255	21.098	13.949	14.443	14.855
proposed pipeline 256x256	abs	1657	**991**	**1165**	912	962	1498	1140	1173	1244	**1240**	2037	1670	1076	1290
	rel.	28.085	**10.656**	**8.630**	8.216	10.809	14.980	14.615	7.066	31.100	**11.698**	19.971	21.410	15.371	15.585

Table 8: **Results Chamfer distance**; lower values indicate better reconstruction results. The absolute values of the Chamfer distance of the ground-truth models to the ground-truth models are represented in the first row. The absolute values of the outputs of comparison variants as well as the relative values to the GT are represented in the respective rows.
The method of Kato, Ushiku, and Harada 2018 scored best for nearly all models. For two classes the proposed method produced better models, while for the other two, the results are very similar for all tested variants.

Chamfer Distance. When reviewing the results of Table 8 it is clear that the Neural Mesh Renderer produces better output meshes. While this thesis' pipeline performs better in 4 out of 15 cases, the difference for 2 cases is very minimal to the output of Kato, Ushiku, and Harada 2018 and can therefore be neglected. Analysing the remaining 2 classes where the proposed framework outperformed the comparison method, there is no real reason for the speaker class to perform better, the improvement of the bench class could be attributed to the usage of a matching base class. But this cannot be determined for certain, since for the chair class this effect did not take place, and the holes are not similar in terms of size and placement to the ones in the sketch. Therefore, while this might have caused the improvement, there is not enough data to confirm that this is the case here.
Since the Chamfer distance is susceptible to outliers, it was expected that the method by Kato, Ushiku, and Harada 2018 is superior to the thesis' when looking at the results of the metric. This is due to the Neural Mesh Renderer using intermediate multi-view images to supervise the deformation, while the proposed pipeline uses only single-view supervision. This confirms the thesis that Kato, Ushiku, and Harada 2018 is better at capturing the overall shape since it is

likely that the hidden parts of the mesh still resemble its base shape or are deformed to minimise the regularisers. Therefore, the results need to be interpreted with the results of the qualitative evaluation in 4.1.2.

Overall it can be determined, that the Neural Mesh Renderer is better at capturing the shape of the meshes, which was expected. However, the outputs of the proposed pipeline are not much worse than the ones by Kato, Ushiku, and Harada 2018, especially when looking at the results of the Intersection over Union evaluation. Therefore, it is reasonable to assume that if improvements, as suggested in Section 5 are implemented in the proposed system, the outputs are better or at least on par with established state-of-the-art methods.

4.2 Ablation Study

To evaluate whether the incorporation of the depth map and the genus of the model that should be reconstructed impact the generation of the mesh in a positive way, an ablation study is conducted. For that, 4 variants of the pipeline are evaluated both in a quantitative and qualitative way. For the quantitative evaluation metrics are used to compare the resulting scores of the different variants' outputs for the given input sketches. The qualitative evaluation aims to describe the flaws and differences in the reconstructed meshes in a holistic way. In this section, also a comparison of the reconstructed maps is analysed in order to explain their impact on the imperfections of the meshes and a general comparison of the variants and the influence of their setup on the overall output is evaluated. Furthermore, this evaluation also included non-artificial sketches and their reconstructed meshes in order to prove that the proposed pipeline can also produce reasonable output for hand-drawn images as input.
The following sections describe the experimental setup used for the study as well as discuss the aforementioned evaluations and present their results.

4.2.1 Experimental Setup

Dataset For the generation of the dataset, meshes from the ABC dataset (Koch et al. 2019) and the Thingy10k (Zhou and Jacobson 2016) dataset are adopted. Most models contained in these datasets are flawless or at least easily fixable to be used in the rendering process. In addition to that, they are not based on classes but offer a wide variety of models from different areas. This also introduces a greater variety of topologies compared to ShapeNet3d data, which is used for the comparison study as described in Section 4.1.1.
With the combination of those two sets, it is anticipated to enable the model to reconstruct a greater spectrum of maps of different shapes and to test the capabilities of the constructed model in terms of topology reconstruction. This is contrary to most similar reconstruction methods that used this map generation technique beforehand. They overfitted their models on a set of different classes, like e.g. Lun et al. 2017, 72 which generated three collections, character, aeroplane and chair, for their training or Kato, Ushiku, and Harada 2018, which trained a model for each

of the 13 ShapeNet classes as explained in the previous section.
In preparation for the training, the models are pre-processed in order to make sure, that only 1 model is processed and that the mesh is watertight, normalised, at the centre of the coordinate system, has consistent normals and has no unintended holes. After rendering the sketches, normal maps and depth maps, the resulting dataset consisted of a total of 7491 256x256 images. The dataset is split into a sketch dataset, containing sketch files, and a target dataset, containing either normal or depth maps. Those folders are split into the train, validation and test, with 1611 test, 5480 train and 400 validation images respectively.

Map Generation Parameters. For the training itself, a batch size of 83, over 3000 epochs and mixed precision is used. The other parameters are the same as in the comparison evaluation, which is described in Section 4.1.1. The models used in the pipeline are selected based on the lowest validation loss. These are the checkpoints at epoch 2929 for the depth map prediction with a validation loss of 0.019534 and at epoch 2986 for the normal map prediction with a validation loss of 0.034091.

Comparison Variants. In order to evaluate whether the usage of the depth map in the mesh generation process and a base mesh based on the input sketch impact the quality of the output mesh, 4 variants of this work are tested:

- The variant introduced in this work, in the following referred to as Variant A.
- A variant which does not use the genus evaluated by the topology determination module of the proposed pipeline but a sphere as a base mesh for all input sketches. This variant is referred to as Variant B.
- Variant C does not use depth loss in the deformation process.
- The last variant is an implementation inspired by Xiang et al. 2020, where both the depth loss and the base mesh determination are not used. In the following sections, this is referenced as Variant D.

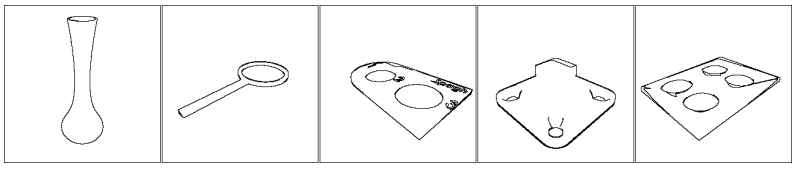
By using 4 different variants instead of only Variant A and Variant D, which is similar to the setup by Xiang et al. 2020 and acts as the base method that is intended to be improved in this work, comparing the output to Variant B and C will determine how the proposed changes affect the model generation.

Pipeline Parameters. The pipeline parameters used in the ablation study are the same as described in Section 4.1.1, however, for the ablation study only 256x256 images are used as input since this is the recommended input size for the pipeline. For variants C and D, the weights are not changed, but the depth loss and therefore the depth weight is not considered here. This cuts down the reconstruction time by around 1 hour due to the faster optimization step, which is a direct result of the smaller gradient associated with the loss that had to be backpropagated. Therefore, the setup for the mesh deformation resembles the setup by Xiang et al. 2020, except

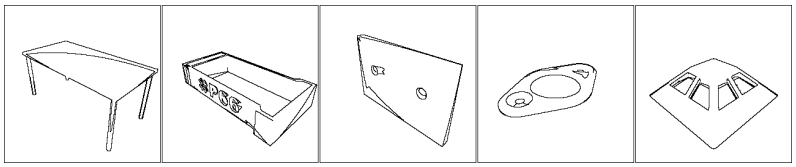
for the smoothness weight, which is chosen to avoid errors introduced by flawed normal map renderings. This is explained in more detail in Section 4.2.2.

Evaluation metrics. Same as in 4.1.1.

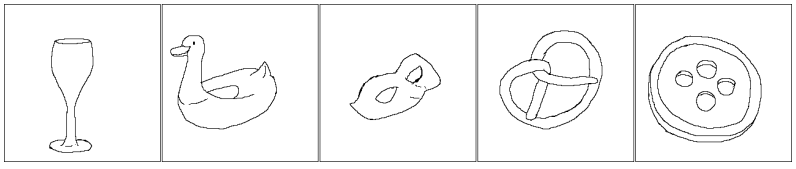
Evaluation Images. The evaluation images are chosen in order to test and compare the variant based on the impact of the quality of the predicted maps, and the complexity of the deformations in the mesh. They are selected from the datasets created for the map generation network, for which the creation process is described in Section 3.2.2, and grouped based on certain qualities the variants are tested and compared on. This results in three categories of images with the topologies of the base meshes, as depicted in Figure 2, being present in every category: Category A with easy-to-reconstruct images, Category B with complex images and Category C with user-generated input. While the images from categories A and B are generated artificially, they are pre-processed in order to make sure that the flood fill and therefore the base mesh determination worked as intended, since this algorithm imposes some limitations on the proposed framework as described in Section 5.1. The images are referred to in the following by their category and the genus of the depicted model over the actual names of the depicted images in order to make the classification straightforward.



(a) Simple images. The images depict a vase (A0), a magnifying glass (A1) and various CAD objects (A2, A3, A4).



(b) Complex images. This set consists of a table (B0) and several CAD objects for genus 1 to genus 4 (B1, B2, B3 and B4).



(c) User-generated images. The drawn sketches included a wine glass (C0) for genus 0, an inflatable in form of a duck (C1) for genus 1, a mask (C2) for genus 2, a pretzel (C3) for genus 3 and a button for genus 4 (C4).

Figure 11: Input images for ablation study.

Category A, which can be seen in Figure 11a, consists of images used in the training set for the map generation module. This should be an advantage in the deformation process, since the normal and depth map from the training set is more likely to be reconstructed correctly, especially in such a diverse training set, compared to images unfamiliar to the network. The shapes of the depicted models are not too different from the base shapes, which means that especially the placement and size of the holes are as similar as possible to those characteristics in the base meshes. Due to the limited size of the training set, those characteristics are sometimes met only partially, however, it is expected that all variants are capable of producing reasonable output meshes.
Category B is depicted in Figure 11b and contains sketches from the test set of the dataset. This introduces the challenge to reconstruct the images unknown to the network and might in itself cause problems if the shapes are not similar to anything the network is already familiar with. This factor can contribute to the reconstruction process in a negative way since there is the added complexity of flawed normal and depth maps. Furthermore, the shapes chosen are less similar to the base meshes. This becomes especially apparent with the sketches chosen for genera 3 and 4 since the arrangement of the holes is different from the base meshes. The size/shape of the holes is also a challenge for the deformation module since for deformations of holes the relation to the body of the model has to be respected in order to prevent errors in the mesh such as extensive inversion of faces.
For the last category, which is portrayed in Figure 11c, 5 human-created sketches are used. The main focus when creating the sketches is that the placement of the holes matched the base meshes in an attempt to guarantee reasonable output meshes. However, the challenge with these sketches is the possibility of flawed depth resp. normal meshes, as with the complex sketches, and that user-generated sketches are different from artificial images, which the network is trained on. The unsteady lines in the sketches, which did not occur in the artificial sketches, are expected to cause some problems with the map generation as well as in the deformation process. However, the primary focus of this test set is to prove that the proposed framework is able to also reconstruct non-artificial images in a way where the relation between the input sketch and the output mesh is apparent and that Version A still does a better job in reconstructing the meshes as opposed to the other variants despite the added difficulty introduced by the potentially flawed reconstructed normal and depth map.

4.2.2 Qualitative Evaluation

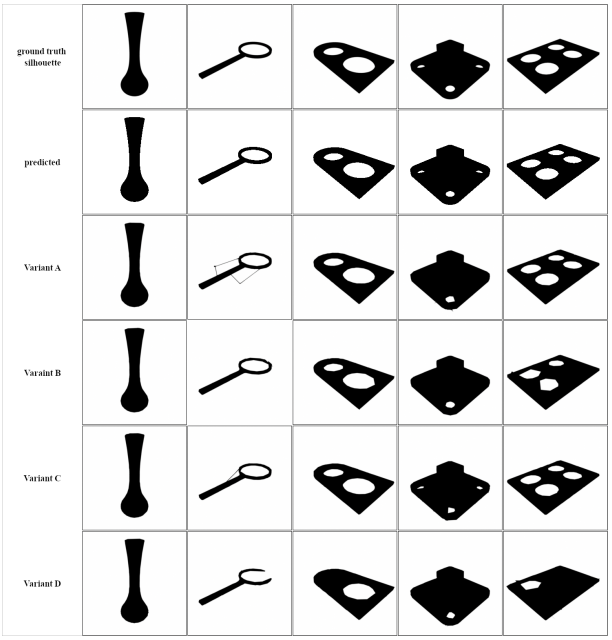
In order to fully understand and interpret if the addition of the depth map and a topology based base mesh improves the reconstruction of 3d models from 2d sketches, both the results of the quantitative evaluation and the qualitative evaluation need to be considered. For the latter, it is important not only to describe and compare the output of the mesh of the different variants, but the resulting silhouette, normal and depth map also needs to be evaluated. Therefore, this section will outline the general impact of the depth and various base meshes on the reconstruction, as well as the comparison of the silhouette maps, the normal maps, and the depth maps, followed by an evaluation of the output mesh in order to determine the difference between the various variants in a qualitative way. The resulting implications on and limitations of the pro-

posed pipeline will be discussed in depth in Section 5.

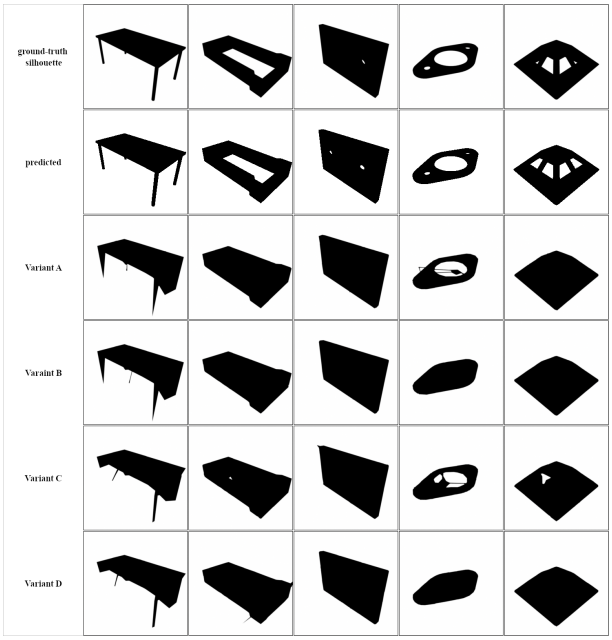
Although the base mesh does not have an impact on whether the deformation process works, it is found that the addition of the depth map stabilises the mesh generation, making it more robust in terms of the selection of the weights and the learning rate. While this may not be important for other systems, the rendering of the normal map in Mitsuba 3 can return invalid numbers, leading to a system failure. This happens due to twisted geometry, which can occur as a result of the randomness of the optimization process. Therefore, if the same input is used in another attempt, this error is likely to not happen again. While this problem cannot be guaranteed to be eradicated with the addition of a depth map and subsequent depth loss since the sample size does not produce a generalisable result on that matter, it is found that it occurs less often or not at all. However, the learning rate and weights still have to be adjusted thoughtfully in order to produce reasonable results.

The results of the silhouette renderings are illustrated in Figure 12. It needs to be acknowledged, that the difference between the predicted, which in this case refers to the flood-filled input sketch, and the ground-truth images, as seen for example in the silhouette of model A4 in Figure 12a, are due to the manual alterations made to the sketches, which are mentioned in the setup part of this chapter. These alterations are minor, however, they could affect the output meshes and the subsequent outcome of the evaluation.
The output of all silhouette renderings emphasizes the results of the quantitative evaluation, with variants D and B being less likely to form holes. However, variants A and C also fail to reproduce the holes adequately in some instances. They are either missing, as seen with model B1 in 12b, reproduce an incorrect number of holes, as seen with model A3 produced by variant A, or deform the hole in ways where the original hole is destroyed and formed into a new structure, as implied by the silhouette of model A1 in Figure 12a. These problems can be partially attributed to flaws in the reconstructed normal and depth maps, which produce losses that are considered more important by the proposed system, as well as the random optimization process, which are discussed in depth in Sections 5.2 and 5.3. In general, it can be determined that while the holes are problematic in some of the output meshes and therefore their silhouette rendering, the outline is relatively accurate for all meshes. Thin lines like the table legs in Figure 11a are not reconstructed accurately, which will also be reflected in the model itself, and occasionally thin, antenna-like structures are added, e.g. in C0 by variant A and C1 by variant B as seen in Figure 12c. These structures, as well as if the small white spots depicted in, e.g. C3 by variants C and D are in fact holes, will be further evaluated in the following with the assessment of the output meshes.

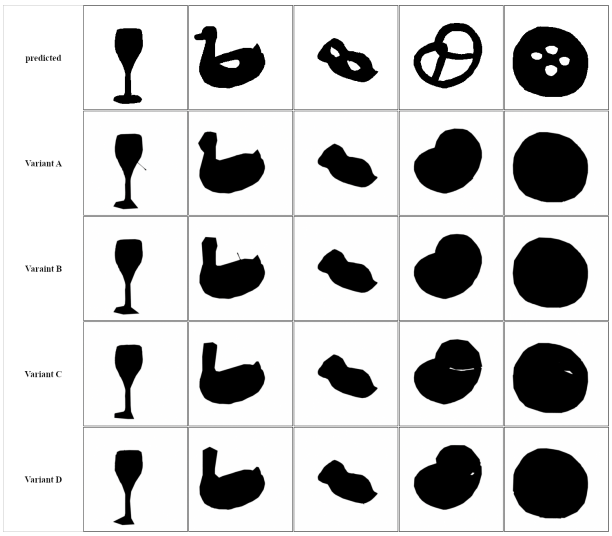
Compared to the silhouette images, the predicted normal maps introduce a greater problem to the mesh reconstruction, since this process is not guaranteed to produce the output depicting similar to the ground-truth normal map and therefore the mesh deformation relies on a distorted version of normal loss. This can even be seen in the normal maps generated from the user-generated images, which is depicted in Figure 13c. Although there is no GT to compare the output to, the noise in e.g. A1 and the filled holes can be regarded as non-desired, since the



(a) Results from simple sketches.



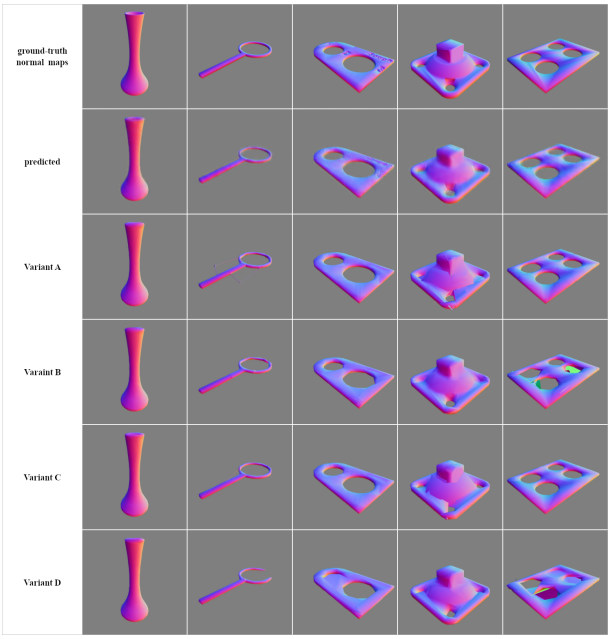
(b) Results from complex sketches.



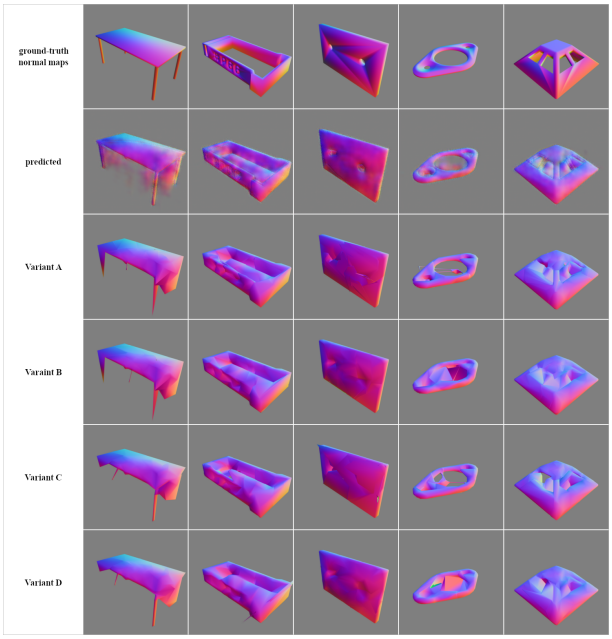
(c) Results from user-generated sketches.

Figure 12: Silhouettes rendered from the same viewpoint as input sketch. The ground-truth row shows the rendered silhouettes of the ground-truth meshes, in the predicted row the flood-filled sketches are depicted and in the remaining rows are the rendered silhouettes of the reconstructed meshes.

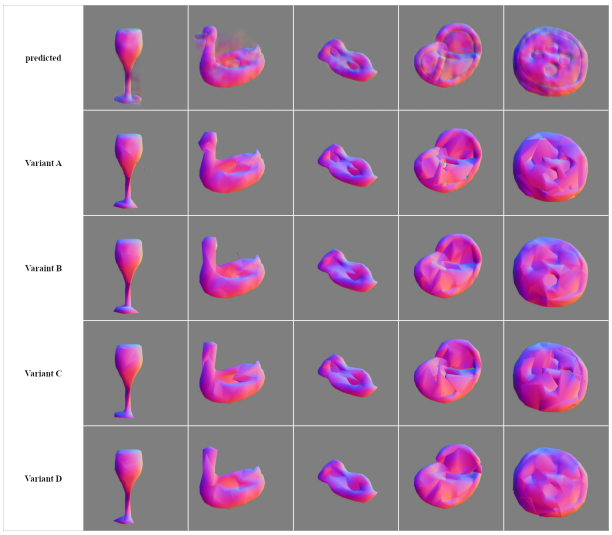
sketches, the silhouettes, as well as common human knowledge about shapes like buttons, do not obtain those features. This is affecting the mesh reconstruction and in turn introducing limitations, which are further discussed in Section 5.2. Furthermore, the values of the predicted images of the complex and user-generated sketches are distributed less evenly than the values of



(a) Results from simple sketches.



(b) Results from complex sketches.



(c) Results from user-generated sketches.

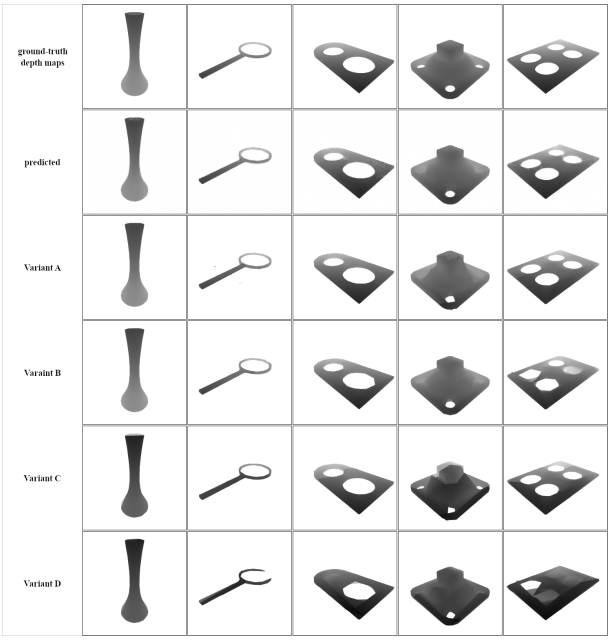
Figure 13: Normal maps rendered from the same viewpoint as input sketch. In the ground-truth row the rendered normal maps of the ground-truth meshes are depicted. The predicted row shows the maps normal map prediction of the map generation network and in the remaining rows the rendered normal maps of the deformed meshes are presented.

the simple images. These problems are likely a result of the unfamiliarity of the network with those sketches, leading to the patchy appearance, closed holes and noise.
As for the differences between the output normals of the models of the different variants they are in general very similar. There are some diversities when it comes to mesh parts other models

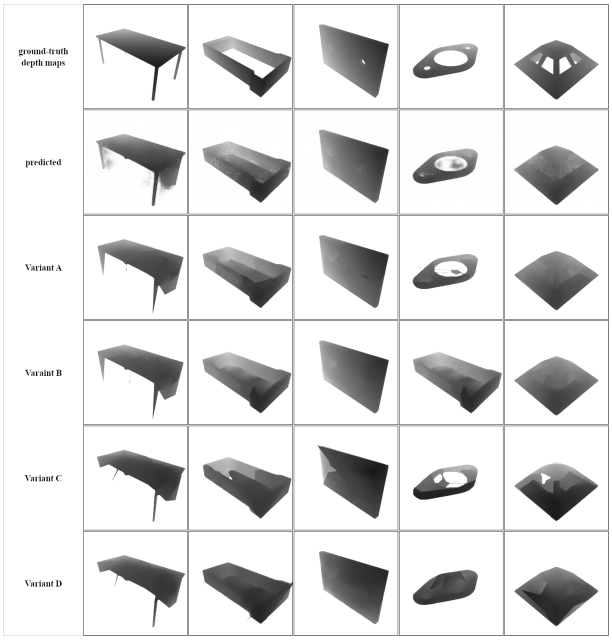
do not have, e.g. B3 in Figure 13a or A4 in Figure 13b, since there is evidently an extended part of the mesh that should not be there. This becomes more evident when comparing the ground-truth and rendering results of the silhouette renderings, e.g. in Figure 12b, and the renderings of the models, e.g. in Figure 15b, themselves. Other than that, there are no visual outliers in the rendered maps of the variants, and they all look similar to the predicted mesh in terms of colour distribution.

The results of the comparison of the depth maps for variants A and B as well as discrepancies between predicted and ground-truth maps, which are shown in Figure 14, are similar to the results of the normal maps in the previous passage. However, the depth maps for variants C and D are also rendered, which do not utilise the depth loss in the reconstruction process. This does not make a huge difference for simple structures like A0, A1, A2 and A4 in Figure 14a with them only being a little bit darker, which indicates that the model is closer to the camera and therefore smaller. Since the model can be re-scaled without any issues and because the proportions are correct this is only a minor issue. The remaining map in this Figure, A3, depict less smooth colour transitions compared to the maps of variant A and B, which can be seen clearly in the bottom part of this mechanical part. This indicates that this part is a little bit bulkier and that the mesh is less smooth compared to the other variants, but overall the difference is not significant.
These map and subsequent model flaws are more apparent in the Figures 14b and 14c. B2, B3, B4, C1 and C4 of variant C and B4 and C1 of variant D depict sharp edges caused by high colour value changes, that indicate either sudden changes in the mesh structure or even overlaps of mesh parts with the overlapping parts not being at the same values as the other mesh and sticking out of the mesh. Therefore, using depth maps and the subsequent depth loss in the mesh deformation not only stabilises the mesh generation, which is discussed in this section beforehand but also seemingly introduces a smoother surface.

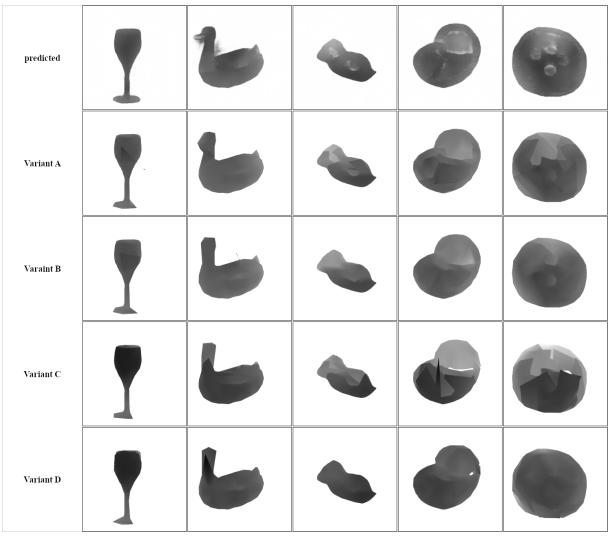
As the results of the previous comparison of the silhouette, normal, and depth map indicate, the variants produced different versions of the models, which can be seen in Figure 15. For the simple models, the variants using a base mesh based on the respective genus of the model produced the best output, followed by the outputs of variant B, with the exception of model A3. This however can be attributed to the discrepancy between the placement and size of the holes in the base and desired meshes, which is noted as a possible problem in the setup of this section as well as in Section 5.2. The results of the evaluation of the renderings of the reconstructed models from the simple sketches also support the results of the quantitative evaluation regarding the improvement the proposed additions introduced, however, there are some inconsistencies due to the deformed hidden parts of the meshes, which are briefly mentioned in the previous part and will be discussed in depth in Section 5.3. Furthermore, as e.g. the open part in A1 of variant D indicates, with more epochs in the deformation process, this problem could have been avoided. Using the same weights and learning rates for the models is likely to become an issue that is already predicted in the pipeline parameters part of Section 4.1.1 and is further discussed in Section 5.3.
Another problem with the genera, which becomes more apparent when looking at the rendered images from the complex and user-generated based meshes in Figures 15b and 15c, is that the



(a) Results from simple sketches.



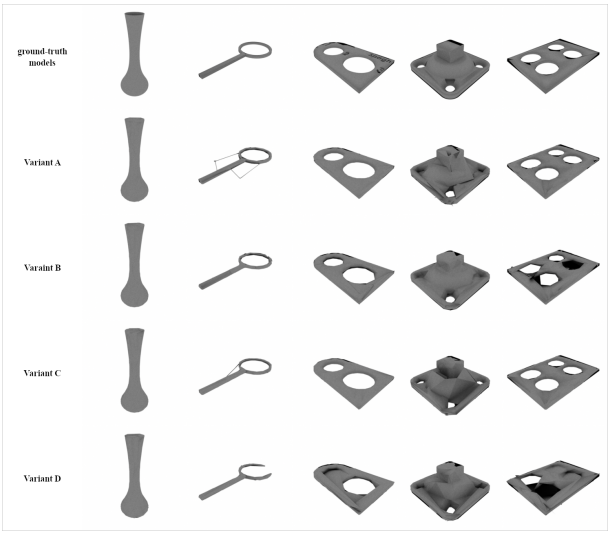
(b) Results from complex sketches.



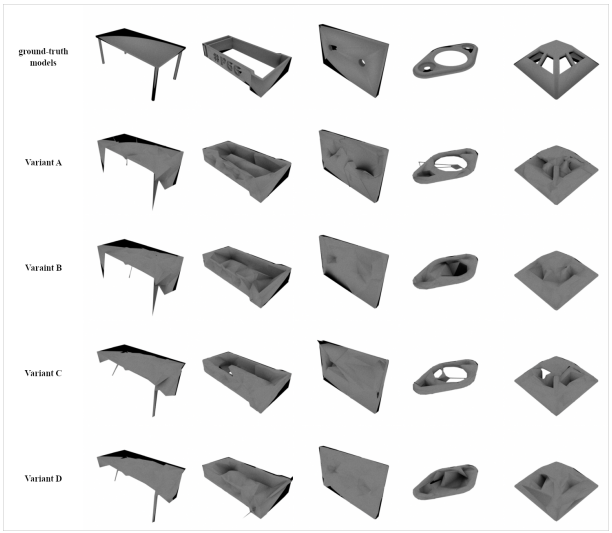
(c) Results from user-generated sketches.

Figure 14: Depth maps rendered from the same viewpoint as input sketch. In the ground-truth row the depth maps of the ground-truth meshes are displayed. The predicted row shows the maps predicted from the trained depth map generation network and in the variant rows the rendered depth maps of the deformed meshes are depicted.

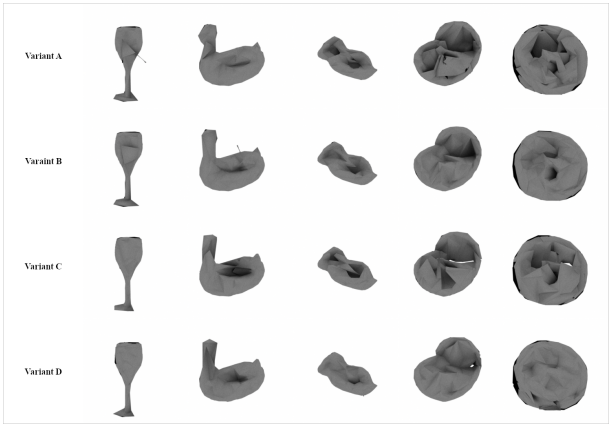
holes are not reconstructed correctly, which is a direct result of the flawed predicted normal and depth maps as well as the randomness of the optimization, which are both discussed in Section 5.3. This undermines possible improvements with the addition of a genus-specific base mesh, resulting in holes to be closed and producing overlaps of mesh parts, which can be seen



(a) Results from simple sketches.



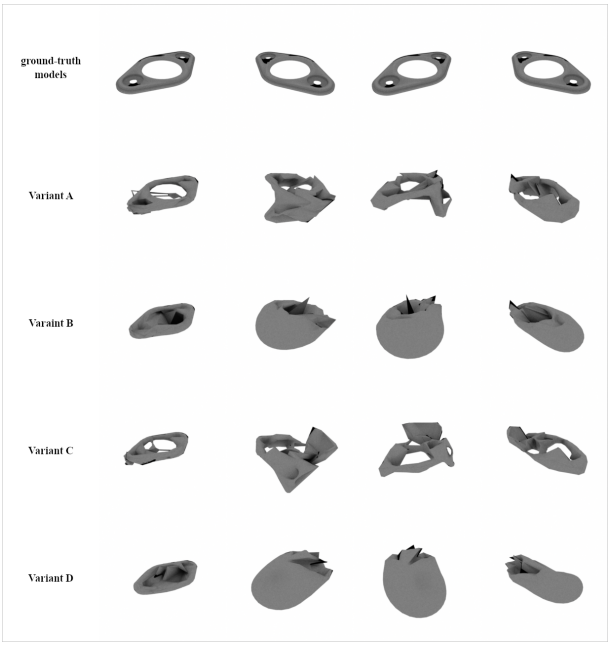
(b) Results from complex sketches.



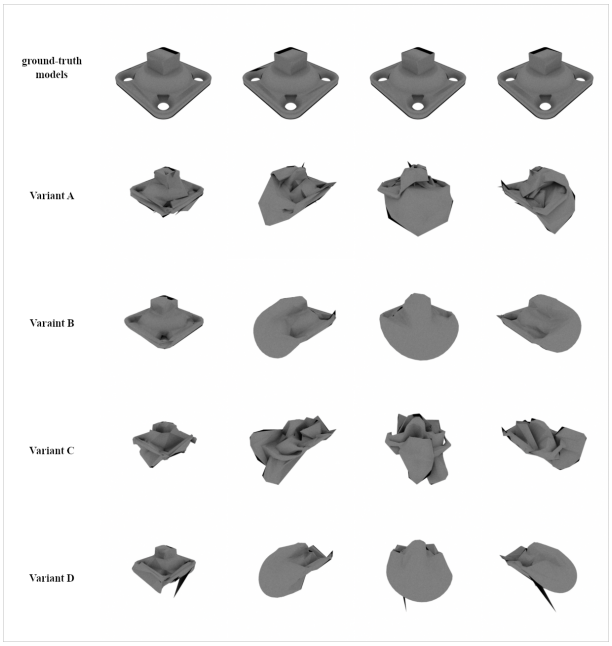
(c) Results from user-generated sketches.

Figure 15: Resulting meshes rendered from the same viewpoint as input sketch. In these figures, the results of the deformation process of the various variants are depicted in their respective row, along with a rendering of the ground-truth mesh if available.

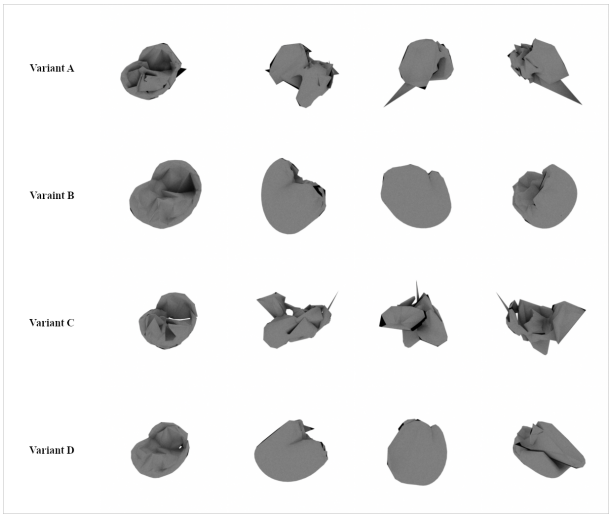
in e.g. C3 and C4 of variants A and C in Figure 15c. However, in these models holes can occur, which is confirmed by reviewing the renderings as well as the meshes themselves, as seen in the models of variants C and D based on complex and user-generated sketches. This could be attributed to the missing supervision of the depth values in the optimization process. Despite that, the renderings of the complex output models support the conclusions of the quantitative evaluation, which established that no variant reconstructed all models equally well.
Since the mesh quality largely exhibits the same general problems for all used variants, these will be discussed in greater detail in Section 5.3. Apart from that, there are a few differences: The models recovered from the simple input sketches depict a few inverted and overlapped faces and random mesh distortions in the parts of the meshes hidden and not visible from the



(a) Results from simple sketch.



(b) Results from complex sketch.



(c) Results from user-generated sketch.

Figure 16: Meshes of all categories with genus 3 rendered from 4 different azimuth angles (225, 315, 45, 135). If available, the first row shows the renderings of the base mesh. In the other rows, the output meshes of the 4 variants are depicted. The depicted meshes are normalised for the rendering process to fit in the field of view.

rendering viewpoint and therefore deformation view, which is a general problem in the proposed pipeline. However, it needs to be noticed, that those distortions are more extreme in shapes that are different from the base mesh or if the base mesh hole positions do not line up with the hole positions of the input sketch, as for example the case with models A1 and A3.

The evaluation of the complex meshes also showcases greater overlaps and self-intersections, which happens with all variants. Furthermore, looking at meshes from other points of view reveal that the holes in the meshes might not be closed if the normal and the depth map and their respective losses direct the reconstruction to close them, but they might get pushed out of the view of the camera, hidden behind other parts of the meshes. The incorrect reconstruction of the holes can also result in the antenna-like structures mentioned beforehand in the silhouette evaluation part and can be seen in variants A and C of Figure 16a. These structures occur primarily if the system does not deform the holes of the base mesh in the intended way but tries to create new holes or structures from other parts of the mesh. This reconstruction problem and the problem of matching the sketches' holes with the base mesh holes are further discussed in Sections 5.1 and 5.3.
Differences between the variants occur more clearly when looking at the outputs using the user-generated sketches as input. Here it becomes apparent that details and proportions are better recovered in variants A and C which used the depth loss in the mesh deformation process, which confirms the conclusion made in the evaluation of the depth maps above.
When it comes to the deformation of the parts that are not visible from the angle the sketch is rendered, the difference depends on the genus rather than the optimization process. As depicted in the examples in Figure 16, in the renderings of the base shape using genus 0 the sphere is still partially visible, while this is not the case for variants A and C because the vertices that make up the holes of the base mesh are moved in the optimization process. However, there is no consistency in the deformations between the variants that share the same improvements, which can be attributed to the randomness of the vertex translations, which is further discussed in Section 5.3. Furthermore, the results do not confirm the theory that the hidden parts are responsible for the difference in the output of the Chamfer metric in Table 10, since they are approximately equally faulty and therefore have about the same influence on the metric.

4.2.3 Quantitative Evaluation

In order to evaluate the metrics for the quantitative evaluation, the generated meshes are pre-processed in order to be normalised and at the centre of the coordinate system. This is done to ensure that the ground-truth mesh and the generated meshes have the same placement and dimensions.
The evaluation results are presented in Tables 9 and 10. These depicted values show that the addition of the depth map as well as using a base mesh similar to the anticipated output shape does have a positive effect on the model reconstruction. However, there is a difference in how this is reflected in the results of the used metrics: While the results for the Chamfer distance clearly mark variant A as the best and variant D as the worst, the output of the IoU present variants A and C as the best and variants D and B as the worst, with variant B having slightly better results than variant D. Therefore it can be reasonably assumed, that the usage of the base mesh based on the genus of the desired mesh improves the quality of the output mesh. The results of the addition of the depth map are not as obvious, however, since the depth map only affects the deformation of the mesh from one view with no impact on the hidden parts of the models, that might explain the results in the IoU, since these parts could impact the output of

the IoU differently than the Chamfer distance.

Comparison Variant	**Output Model**										**Mean**
	simple images					complex images					
	A0	A1	A2	A3	A4	B0	B1	B2	B3	B4	
A	**0.029**	0.091	**0.188**	0.126	0.057	0.020	0.057	**0.174**	0.115	**0.189**	0.105
B	0.022	0.056	0.060	**0.211**	0.027	0.019	0.036	0.155	0.032	0.070	0.069
C	0.023	**0.100**	0.111	0.173	**0.258**	**0.038**	**0.073**	0.071	**0.120**	0.099	**0.107**
D	0.025	0.050	0.028	0.181	0.034	0.035	0.037	0.136	0.009	0.083	0.062

Table 9: **Results Intersection over Union** for the four comparison variants. Variants C and A scored best on average, with C being better than A and the evaluation of B resulting in better values than the evaluation of D. The proposed variant scored best in 4/10 methods and resulted in reasonable results in another 2/10 categories. Overall the metrics indicate poor quality of the reproduced meshes. Higher values indicate better reconstruction results.

Intersection over Union. When evaluating the results in Table 9, it becomes apparent that the results for variant C are slightly better than for variant A. Since for many models the difference is very small the proposed method can still be seen as an improvement to the ground-truth variant D. This variant scored the best in 4 of 10 models and is a little bit worse than variant C when deforming the models A1 and B3. The results of the evaluation of the variant are not impacted based on whether the input is a simple or a complex sketch, however, the best results are scored on model A3 with variant B and A4 with variant C, but these are rather outliers than a general trend for the results.

Comparison Variant		**Output Model**										**Mean**
		simple images					complex images					
		A0	A1	A2	A3	A4	B0	B1	B2	B3	B4	
ground-truth Values	abs.	*101*	*44*	*82*	*110*	*90*	*92*	*102*	*103*	*71*	*112*	*91*
A	abs.	996	**1150**	**338**	**1200**	**479**	1551	1330	**826**	978	1832	**1068**
	rel.	9.861	**26.136**	**4.122**	**10.909**	**5.322**	16.859	13.039	**8.019**	13.775	16.357	**12.440**
B	abs.	**973**	1238	1300	1291	1789	**1497**	1274	1244	1164	**1138**	1291
	rel.	**9.634**	28.136	15.854	11.736	19.878	**16.272**	**12.490**	12.078	16.394	**10.161**	15.264
C	abs.	1638	1154	376	1795	511	1971	1486	1273	**770**	1420	1239
	rel.	16.218	26.227	4.585	16.318	5.678	21.424	14.569	12.359	**10.845**	12.679	14.090
D	abs.	1171	1781	1293	1990	1446	1944	1585	1474	1369	1247	1530
	rel.	11.594	40.477	15.768	18.091	16.067	21.130	15.539	14.311	19.282	11.134	18.339

Table 10: **Results Chamfer distance** for the four comparison variants, lower values indicate better reconstruction results. The absolute values of the Chamfer distance of the ground-truth models to the ground-truth models are represented in the first row. The absolute values of the outputs of the variants of the model as well as the relative values to the GT are represented in the respective rows.
The evaluation of the proposed model A resulted in the best values and the evaluation of variant D in the worst. This indicated that despite the overall inadequate quality of the output meshes, the proposed method improves the base variant D inspired by research such as Xiang et al. 2020.

Chamfer Distance. The insignificance of the category of the model is not reflected in the results of the Chamfer distance, which are presented in Table 10. Here, it is clear that the proposed variant scores best for 4 out of 5 models and second best for the remaining model, with the result of variant C being better. For the complex category variant A is the best for model B2, and second best to the results of variant C for model B3. The results of the other models do not show a clear trend despite variant B scoring the best, however, the distribution of result values among the other models does not reflect the distribution of the scores of the simple images. Therefore, for complex models the variant or included parameters have to be more carefully chosen compared to the simple models. This is discussed in more detail in Section 5.3. Furthermore, it needs to be acknowledged that due to the unsupervised deformation of the hidden parts of the meshes, those could introduce coincidental outliers, that are considered by this metric. While this likely does not have a great influence, the results of the Chamfer distance should still be interpreted with the results of Section 4.2.2.

Overall it can be determined, that there is very little similarity in the output of the IoU and Chamfer distance regarding the ranking of the models, but it is indicated that the implemented adjustments improve the base variant. Furthermore, the results do not indicate a close similarity between the ground-truth meshes, neither for the results of the IoU nor the results of the Chamfer distance. The reason for this stems from the limitations the proposed system has, which are further investigated in Sections 4.2.2 and 5. It needs to be acknowledged, that those results can be improved, which will be further discussed in the Sections 5 and 6, however, those improvements are beyond the scope of this work, which is why the results reflect the output of an implementation including the essential functionalities to reproduce a mesh from a sketch rather than a state-of-the-art implementation.

5 Limitations and Possible Solutions

While the proposed pipeline is able to reconstruct models from sketches, there are limitations to this implementation. These are not limited to a single part of the implementation, but every module of the three that compose the proposed framework, which is outlined in Section 3, could be improved in order to correct errors and flaws in the output meshes. While some of those are already mentioned in the previous chapters, especially in Section 4, in the following the limitations of the pipeline are discussed in detail and suggestions for improvement are made. The limitations are grouped based on the part of the framework they occur within.

5.1 Hole Detection and Topology Determination - Base Mesh Determination Module

The flood fill algorithm and Euler number to generate the silhouette image as well as to determine the genus of the depicted sketch impose several problems, that can cause issues in the

reconstruction process.

The first issue is touched upon in the introduction of this method in Section 3.1.1: Since an 8-connected filling algorithm is used, the sketch lines need to be thoroughly closed. This, as well as the fact that the sketch needs to be seeded in order to provide starting points for the algorithm, requires user interaction, especially when it comes to artificial sketches. For human-generated sketches, the difference is not significant since the process of creating the sketch is already a manual task performed by the user, but for artificial sketches, there has to be an extra pre-processing step introduced in order to seed and clean the sketch. A solution to that could be an automatic pre-processing step that thickens the sketch lines that in addition to an algorithm like the one by Yi et al. 2016 for cleanup to provide a clean input for the pipeline. However, this does not solve the issue that the image needs to be seeded by hand. Since a sketch does not provide any information to distinguish the regions, methods like watersheds, thresholds or region-based methods are not applicable to that issue, because it cannot be determined whether a region belongs to an object or not. Attempts of using automatic methods like scan-line algorithms, e.g. as done by He, Hu, and Zeng 2019, had some success in that, however, this classification alternates between closed regions and holes, as explained in Section 2.5.4. This is not necessarily the case, especially for detailed sketches. Using neural networks could be a solution, however, based on the model several issues could occur: If an image-to-image translation network similar to the normal and depth map generator in this thesis is used, the issue of closed holes could occur, which would impact the genus determination. For the segmentation algorithms, there are not really datasets that provide meshes as well as the needed information for the segmentation. The easiest solution for that could be to add information about the genus and the connectivity to an existing dataset like ShapeNet and use a classification network. While this is the simplest way to limit human effort as much as possible, it would be an extensive addition to the pipeline with a benefit that is very minimal compared to if the seeding process is still done by the user, especially if the cleanup process is already automated by the proposed sketch pre-processing module.

However, the classification network could theoretically improve the problems introduced by the Euler number: This is only applicable to a single object depicted in the sketch in the proposed pipeline. For multiple objects, this method does not work anymore, since assuming the connected components C of the formula 3 are assumed to be 1. Here the classification network could determine how many objects are depicted. This could also be realised by a type of object detection network. However, using multiple meshes introduces added complexity in the map prediction module. Therefore, it is easier to separate the split image into several, with a split depicting only one object. This solution would also be applicable in the deformation module, which cannot deform multiple meshes in one process, but multiple instances of that module could be started parallel using a split as input. For the split process, object detection could be utilised and a combination of a flood-filled image as well as a connected component analysis, which is, e.g., provided by the OpenCV library or just different coloured seeds for the objects could be used. However, it needs to be ensured that the separated images are square, preferably 256x256, and have no blurred lines, etc. that can occur during the resizing process, which in-

troduces other challenges to a possible pre-processing module.

Another problem that is the direct result of the usage of the Euler number to determine the genus is that the resulting number is not necessarily correct. In Figure 17 three examples are depicted, where the proposed method will determine the wrong base mesh for the reconstruction method. In the first image, the tentacles curl at the ends, which will be identified as holes by this algorithm since here the outlines themselves are considered part of the object. This would lead to a genus 6 for that object for which no base mesh is provided, instead of the genus 0, which would be sufficient to reconstruct that image. A solution would be to not include the outline of the object in the final filled object, however, for some objects, this is not possible, like the second sketch depicted in Figure 17. Here the object itself forms a hole from the viewpoint it is looked at and removing the outline pixels does not change that. Therefore, this is classified to have a genus of 1, while a human that recognises these objects or objects similar to that knows, that for both sketches the genus would be 0. Using that concept in the pipeline could also provide a solution for that problem, however, would require an object recognition algorithm or at least some form of line labelling to determine whether a line is an outline and base the genus determination on that, which is a complex addition to the existing pipeline. This could also solve the problem of the third image in 17: Here the problem is not this algorithm, since the sketch only shows 3 holes and therefore the base mesh is correctly determined to be of genus 3. However, as established from evaluating multiple perspectives of this mesh, as depicted in Figure 16a, and a human would be able to guess since such parts are usually symmetrical, this mesh does not fit the actual genus of 4. But this cannot be determined solely from the input, more information, e.g. in form of an additional object recognition or classification network would have to be provided to solve this issue.

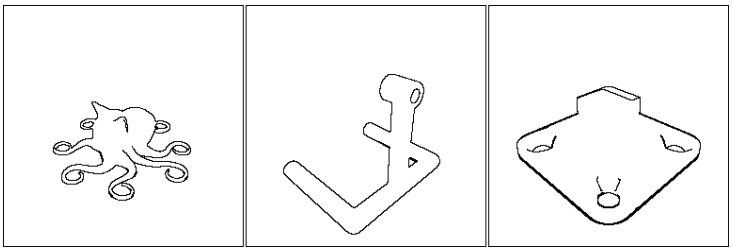


Figure 17: Sketches where using the flood fill algorithm in combination with the determination will result in wrong base meshes.

The last problem observed that can occur due to the topology and base mesh determination is if the genus of an object is correctly determined, however, the base mesh just does not fit the aimed shape. This can be seen e.g. in the reconstruction A1 or B3 in Figures 15a and 15b and is a result of the combination of the deformation module and the base mesh, with the influence of the former part being discussed in depth in 5.3. The part of the issue stemming from the base mesh determination is that generic base meshes are used and that only the genus is considered, but not how the holes are located relative to each other. An attempt to tackle that problem is to

find the shortest connections between the holes in the flood-filled sketch and by comparing the lengths of the connections more information about the locations could be gathered. Using that information combined with the genus of the desired object can result in choosing a more fitting base mesh. While this would result in a requirement for more base meshes, it is a more general approach than using specific base meshes for a certain class, like e.g. Smirnov, Bessmeltsev, and Solomon 2021 did.
However, not only the placement of the holes relative to each other matters but also the global placement of the holes. This can e.g. be seen in A1 in Figure 15a, where the existing hole is not used to reconstruct the hole of the depicted mesh. Trying to fit those global positions to align in the sketch and the base mesh would be difficult with higher genera since this would mean that the base mesh would have been deformed before the actual deformation process, and while it is likely that the pre-existing holes remain if they align the predicted maps and the silhouette image, it is not guaranteed, since the optimization process moves the vertices at random in order to get the optimal output. Therefore this is rather a problem of the differentiable renderer and the deformation part of the pipeline, which is why this is discussed in Section 5.3.

5.2 Map and Hole Reconstruction - Depth and Normal Map Prediction Module

The most obvious problem in the map generation are the closed holes in the output meshes and the uneven normal resp. depth value distribution in some models, e.g. in the complex and user-generated models in Figure 13. As seen in e.g. Figures 13b and 14c, where the parts which should be white are coloured with normal resp. depth values. This influences the normal resp. depth loss in the mesh reconstruction module, because those losses have a huge impact on the overall loss and therefore the mesh is optimised to fit the normal resp. depth map as closely as possible to the predicted mesh. This issue stems from the large variety of the Thingy10k/ABC dataset used in the ablation study because the placement of the holes as well as the value distribution can differ drastically between the models and can therefore be not easily learned. Furthermore, the testing images may also not resemble the training data, which makes it more difficult if the sketches used for the evaluation are taken from the test set. This does not have such an impact for datasets with more uniform data like ShapeNet3d, especially if a separate model is trained for each class/object type. Despite unifying and dividing the dataset, adapting the parameters, especially the number of epochs, can also improve the predictions, since the parameters used are chosen to produce reasonable outputs rather than optimal outputs. This is because for this process there is no empirical way of choosing the correct values, which in turn makes it not generally applicable to every prediction network equally well. However, there is no guarantee that the holes are reconstructed correctly if sketches are used as input to the network on which it is not trained and is therefore not familiar with. This is even worse for inputs where the size and location of the holes are not similar to those known to the trained network. An adaption and unification of the dataset is the best way to improve the prediction of the maps and therefore the overall output mesh. This will also improve the overall prediction of the value distribution, since, as mentioned before, the maps resulting from this prediction can be noisy, which can result in additional mesh parts that are not in the original mesh, e.g. with

B0 in Figure 13b.
Another minor problem, which needs to be acknowledged, is that the network is trained with images depicting the objects from a single view. While this is not necessarily a problem for the Thingy10k dataset since there the models do not have a coherent up vector, for the ShapeNet dataset this limits the ability to reconstruct sketches. For this problem, the solution would be to render the sketch and the target images from several angles, as done by Kato, Ushiku, and Harada 2018. Since there is only a single view needed to examine the proposed pipeline, renderings from multiple viewpoints are not included, because this would likely also enhance the training time as well as the time it takes to render the datasets. However, extending the prediction to multiple viewpoints would not be a problem, since the proposed implementation for the dataset generation would be capable of rendering the meshes from multiple camera locations and angles and nothing would change for the actual training of the network.

5.3 Limitations introduced by Single View and Deformation Process - Differentiable Rendering and Mesh Deformation Module

The third module of the proposed pipeline has several limitations that drastically influence the quality of the output mesh. The first problem stems from the given base meshes and their respective vertex counts. For this thesis, as well as for other reviewed work in this field, base meshes with approximately 600-700 vertices are used in the deformation process, which restricts the methods to only reconstructing low-poly meshes. While using meshes with more vertices would be possible in this system as well as in other work, this is not the norm due to several problems: Using more vertices would expand the gradient, which in turn would increase the time the optimization process takes to backpropagate the loss. Furthermore, the edge and smoothness loss would take longer times to compute and since these have to be evaluated every epoch, the overall time the mesh deformation would take increases. Another problem with high-poly meshes is that the output mesh is likely to have more flaws like inverted faces and fold-overs since there are more vertices that can be deformed in various ways. This would require an adjustment of the optimization process by e.g. introducing a local bias that keeps the vertices in relation to each other or the weights for the edge and smoothness regularisers need to be enhanced, which could undermine the influence of the losses and affect the reconstruction. Therefore, while using only low-poly meshes limits the proposed pipeline, there are valid reasons this is done and unless the above-mentioned problems are solved, high-poly meshes cannot be recovered from images in a reasonable way.

Despite the limitations caused by the parameters and the number of vertices in the base mesh, the main problem is introduced by the optimization process itself. As pointed out in the qualitative evaluations in Sections 4.1.2 and 4.2.2 as well as depicted in Figures 10 and 16 it becomes apparent that the proposed pipeline cannot recover parts of the object that are not depicted in the sketch. Furthermore, the randomness of the optimization process can lead to undesirable deformations of the holes, where the system tries to form new holes instead of using existing

ones. As a result, those parts are deformed in various ways and do not resemble the intended shape in any way. A solution to that problem could be to use an attempt similar to Kato, Ushiku, and Harada 2018, where multiple views are predicted, which is however heavily class-based, as described in Section 4.1.2. Another suggestion is to use biases in the optimization process, which are added to the vertex value, as done e.g. by Xiang et al. 2020. Since they do not disclose how their network computes those biases, which is assumed to be either based on the class or the symmetry of the models. This, in turn, further restricts the network as well as requires another system, which needs to be added to the framework in order to predict the biases. However, restricting the optimization of the possible vertex deformations could also improve issues like closing holes, undesired deformation of holes, fold-overs and inverted normals, which are prevalent in most research in this field. For the latter two problems other solutions of varying complexity and effectiveness are applicable like using another output representation like point clouds or implicit functions, which are described in Section 2.1, using re-meshing algorithms or methods from recent research like Nicolet, Jacobson, and Jakob 2021. However, the problem of the deformed hidden part of the mesh would still remain. A simple improvement would be to cut the deformed mesh in half, mirror the remaining part and connect the two halves. While this would only work for symmetrical objects and would require the sketch to display the image in a way that it can be cut along a specific axis, it would provide a quick resolution for some models. Furthermore, the previous idea of predicting biases also needs some information about class or symmetry, which would introduce similar restrictions but be more time-consuming. Therefore, a combination of the mirroring approach along with some form of re-meshing as done e.g. with the cleanup of the ShapeNet dataset using the work by Xu et al. 2019 and Sin, Schroeder, and Barbič 2013 could improve the output of the proposed pipeline.

6 Conclusion and Future Work

This thesis proposes a method to recover 3d meshes from 2d sketches using DR and deep learning. The research includes the implementation of a pipeline, which consists of 3 modules that serve different purposes: A module to determine the silhouette as well as the topology and, based on that, a base mesh; another module that employs an image-to-image translation neural network based on the work of Isola et al. 2017 and Su et al. 2018 to predict the normal and depth map of the mesh; and a third module for the mesh deformation process, in which the differentiable renderer Mitsuba 3 is used to deform the determined base mesh based on several losses including the silhouette, normal and depth loss for which the predicted maps and silhouette image from the other models are used. By using the depth map in the loss computation of the DR process as well as a base mesh based on the topology of the sketch, this thesis aims to improve the existing work like the one by Xiang et al. 2020.

In order to prove the improvements made in the pipeline and determine whether the addition of the depth map and the base mesh determination improve the state-of-the-art methods in terms of mesh quality and usage of input meshes beyond class/object restrictions, two studies are conducted: A comparison with a state-of-the-art model to determine how the system performs

against an established network, as well as an ablation study in which different versions of the implemented pipeline are tested and compared.
For the first evaluation the Neural Mesh Renderer by Kato, Ushiku, and Harada 2018 is used as a comparison model. The results of the quantitative and the qualitative evaluation confirm the assumption that the Neural Mesh Renderer is good at capturing the overall shape of the object, while the proposed pipeline is better at reconstructing details. However, the results of this evaluation are significantly worse compared to the ones in other research. This can be attributed to the dataset, for which the method of Xu et al. 2019 and Sin, Schroeder, and Barbič 2013 is utilised to make the data of the ShapeNet v1 dataset usable, which resulted in more detailed meshes compared to the re-meshed representation that is based on the voxelised data of ShapeNet, used in the work by Kato, Ushiku, and Harada 2018. Furthermore, for this thesis a split different from the ones by Kato, Ushiku, and Harada 2018 to divide the dataset into train, validation and test parts are generated since Kato, Ushiku, and Harada 2018 did not disclose in their work which split they used. Considering that, better results would likely be achieved if those alterations were not made, especially for the output of the Neural Mesh Renderer. Furthermore, due to the simplicity, a smaller input image could be used in the pipeline since the details are not as prevalent in those ground-truth meshes. While this would be a valid test to conduct in future work, the simplification of the ground-truth models as well as the input image makes the usage of meshes obsolete, since one of the great advantages of that is that details are better reconstructed and those are expunged if the ground-truth meshes are simplified. In that case, models based on point clouds or voxels would be sufficient, which would also not exhibit the problems introduced by the mesh deformation.
For the second evaluation, 4 variants of the thesis pipeline are tested against each other: The proposed version, one with a sphere as base mesh, one with no depth loss, and a fourth with a sphere as base mesh as well as no depth loss. For the evaluation, a for the thesis constructed dataset is used, which is composed of data from the Thingy10k dataset and the ABC dataset in order to get a large variety of models with different genera. The results conclude that the addition of the genus-based base mesh and the depth map resp. depth loss did improve the mesh reconstruction, with the additions having different influences on the deformation process: While the genus-based base mesh allowed for a better overall reconstruction of the shape, the depth map added additional information to the reconstruction process, which also had the side effect of avoiding invalid normal values in the differentiable rendering and therefore stabilizing the deformation process.

While the proposed system improves the base version inspired by Xiang et al. 2020 and performs as expected compared to a state-of-the-art method, there are several limitations, which need to be improved in order to produce usable output meshes. Since many problems stem from missing information, which the sketch does not provide due to the single view as well as the simplicity of this medium, employing some form of object recognition network would be a valuable addition to the network. This would make the topology-determination module obsolete, as well as could provide more information for the deformation process, e.g. in the form of images depicted from different views. While this is a similar approach to multi-view approaches like Kato, Ushiku, and Harada 2018 or Lun et al. 2017, the difference would be that

there is not a trained network per class and can therefore be generalised if a non-uniform dataset is used. However, this addition would require a modification of the dataset as well as the implementation of a new network, which are both time-consuming tasks. An easier way to improve the output meshes is to add small improvements like using only half of the input mesh and mirroring that half to construct the output mesh and use a re-meshing approach on this mesh. This with other small improvements like the cleanup of the input sketch, using more base meshes as well as improving the topology determination also take the relations of the position of the holes into account and optimise the parameters. While this will not yield optimal results and still could suffer from the problems like undesired deformation of holes, these optimizations can be implemented within a reasonable time and are expected to improve the system profoundly.

Overall, the output meshes of the thesis pipeline are not sufficient to be used as representations of sketches without them being extensively edited. As of right now, most reconstruction methods do not have this quality due to various problems like fold-overs and inverted faces, for which extensive adjustments need to be made, as e.g. presented in the work of Nicolet, Jacobson, and Jakob 2021. Despite that, it is proven that the addition of a depth map to the deformation process, as well as the use of a general base mesh based on the genus of an object, did improve the reconstruction process. While this approach, along with other methods like Ben Charrada et al. 2022, is a step to generalise the reconstruction models more, there are various limitations that require extensive improvements, both in the system itself as well as in external data like more diverse datasets. However, incorporating the suggested improvements should reduce the limitations of the pipeline and lead to favourable results.

References

Adams, Rolf, and Leanne Bischof. 1994. "Seeded Region Growing." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 16 (6): 641–647.

Al-Amri, Salem Saleh, NV Kalyankar, and SD Khamitkar. 2010. "Image segmentation by using edge detection." *International journal on computer science and engineering* 2 (3): 804–807.

Arjovsky, Martin, Soumith Chintala, and Léon Bottou. 2017. "Wasserstein Generative Adversarial Networks." In *Proceedings of the 34th International Conference on Machine Learning*, 70:214–223. Sydney, NSW, Australia: PMLR.

Bangaru, Sai, Tzu-Mao Li, and Frédo Durand. 2020. "Unbiased Warped-Area Sampling for Differentiable Rendering." *ACM Trans. Graph.* 39 (6): 245:1–245:18.

Ben Charrada, Tarek, Hedi Tabia, Aladine Chetouani, and Hamid Laga. 2022. "TopoNet: Topology Learning for 3D Reconstruction of Objects of Arbitrary Genus." *Computer Graphics Forum* 41 (6): 336–347.

Bradski, G. 2000. "The OpenCV Library." *Dr. Dobb's Journal of Software Tools.*

Chang, Angel X., Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, et al. 2015. *ShapeNet: An Information-Rich 3D Model Repository.* Technical report arXiv:1512.03012 [cs.GR]. Stanford University — Princeton University — Toyota Technological Institute at Chicago.

Chen, Qimin, Vincent Nguyen, Feng Han, Raimondas Kiveris, and Zhuowen Tu. 2020. "Topology-aware single-image 3d shape reconstruction." In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 1089–1097. Seattle, WA, USA: IEEE.

Chen, Zhiqin, and Hao Zhang. 2019. "Learning Implicit Fields for Generative Shape Modeling." In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 5932–5941. Long Beach, California, USA: IEEE.

Choy, Christopher B, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 2016. "3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction." In *Computer Vision – ECCV 2016*, 628–644. Springer International Publishing.

Chudasama, Diya, Tanvi Patel, Shubham Joshi, and Ghanshyam I. Prajapati. 2015. "Image Segmentation using Morphological Operations." *International Journal of Computer Applications* 117 (18): 16–19.

Delanoy, Johanna, Mathieu Aubry, Phillip Isola, Alexei A. Efros, and Adrien Bousseau. 2018. "3D Sketching using Multi-View Deep Volumetric Prediction." *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 1 (1): 21:1–21:22.

Eigen, David, and Rob Fergus. 2015. "Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-Scale Convolutional Architecture." In *2015 IEEE International Conference on Computer Vision (ICCV)*, 2650–2658. Santiago, Chile: IEEE.

Eigen, David, Christian Puhrsch, and Rob Fergus. 2014. "Depth Map Prediction from a Single Image Using a Multi-Scale Deep Network." In *Advances in Neural Information Processing Systems*, 27:2366–2374. Montreal, Quebec, Canada: Curran Associates, Inc.

Falcon, William, and The PyTorch Lightning team. 2019. "PyTorch Lightning." Accessed February 13, 2023. https://www.pytorchlightning.ai.

Fan, Haoqiang, Hao Su, and Leonidas J. Guibas. 2017. "A Point Set Generation Network for 3D Object Reconstruction from a Single Image." In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2463–2471. Honolulu, Hawaii, USA: IEEE.

Gao, Chenjian, Qian Yu, Lu Sheng, Yi-Zhe Song, and Dong Xu. 2022. "SketchSampler: Sketch-based 3D Reconstruction via View-dependent Depth Sampling." In *Computer Vision – ECCV 2022*, 464–479. Springer Nature Switzerland.

Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. "Generative Adversarial Nets." In *Advances in Neural Information Processing Systems*, 27:2672–2680. Montreal, Quebec, Canada: Curran Associates, Inc.

Guillard, Benoit, Edoardo Remelli, Pierre Yvernay, and Pascal Fua. 2021. "Sketch2Mesh: Reconstructing and Editing 3D Shapes from Sketches." In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 13003–13012. Montreal, Quebec, Canada: IEEE.

Gulrajani, Ishaan, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. 2017. "Improved Training of Wasserstein GANs." In *Advances in Neural Information Processing Systems*, 30:5767–5777. Long Beach, California, USA: Curran Associates, Inc.

He, Yixuan, Tianyi Hu, and Delu Zeng. 2019. "Scan-flood Fill(SCAFF): an Efficient Automatic Precise Region Filling Algorithm for Complicated Regions." In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 761–769. Long Beach, California, USA: IEEE.

Isola, Phillip, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. 2017. "Image-to-Image Translation with Conditional Adversarial Networks." In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 5967–5976. Honolulu, Hawaii, USA: IEEE.

Jakob, Wenzel. 2019. "Enoki: structured vectorization and differentiation on modern processor architectures." Accessed July 27, 2022. https://github.com/mitsuba-renderer/enoki.

Jakob, Wenzel, Sébastien Speierer, Nicolas Roussel, Merlin Nimier-David, Delio Vicini, Tizian Zeltner, Baptiste Nicolet, Miguel Crespo, Vincent Leroy, and Ziyi Zhang. 2022. *Mitsuba 3 renderer.* V. 3.0.2. Https://mitsuba-renderer.org.

Jakob, Wenzel, Sébastien Speierer, Nicolas Roussel, and Delio Vicini. 2022. "Dr.Jit: A Just-In-Time Compiler for Differentiable Rendering." *Transactions on Graphics (Proceedings of SIGGRAPH)* 41 (4): 124:1–124:19.

Kass, Michael, Andrew Witkin, and Demetri Terzopoulos. 1988. "Snakes: Active Contour Models." *International Journal of Computer Vision* 1 (4): 321–331.

Kato, Hiroharu, Deniz Beker, Mihai Morariu, Takahiro Ando, Toru Matsuoka, Wadim Kehl, and Adrien Gaidon. 2020. "Differentiable Rendering: A Survey." *arXiv preprint arXiv:2006.12057*, 1–20.

Kato, Hiroharu, Yoshitaka Ushiku, and Tatsuya Harada. 2018. "Neural 3D Mesh Renderer." In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 3907–3916. Salt Lake City, Utah, USA: IEEE.

Kaur, Dilpreet, and Yadwinder Kaur. 2014. "Various Image Segmentation Techniques: A Review." *International Journal of Computer Science and Mobile Computing* 3 (5): 809–814.

Kazhdan, Michael, and Hugues Hoppe. 2013. "Screened Poisson Surface Reconstruction." *ACM Transactions on Graphics* (New York, New York, USA) 32 (3): 29:1–29:13.

Koch, Sebastian, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. 2019. "ABC: A Big CAD Model Dataset For Geometric Deep Learning." In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 9593–9603. Long Beach, California, USA: IEEE.

Ku, Tao, Qirui Yang, and Hao Zhang. 2021. "Multilevel feature fusion dilated convolutional network for semantic segmentation." *International Journal of Advanced Robotic Systems* (London, England, United Kingdom) 18 (2): 1–11.

Lewiner, Thomas, Hélio Lopes, Antônio Wilson Vieira, and Geovan Tavares. 2003. "Efficient implementation of Marching Cubes' cases with topological guarantees." *Journal of graphics tools* 8 (2): 1–15.

Liu, Shichen, Weikai Chen, Tianye Li, and Hao Li. 2019. "Soft Rasterizer: A Differentiable Renderer for Image-Based 3D Reasoning." In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 7707–7716. Seoul, South Korea: IEEE.

Loper, Matthew M., and Michael J. Black. 2014. "OpenDR: An Approximate Differentiable Renderer." In *Computer Vision – ECCV 2014*, 154–169. Springer International Publishing.

Loubet, Guillaume, Nicolas Holzschuch, and Wenzel Jakob. 2019. "Reparameterizing Discontinuous Integrands for Differentiable Rendering." *ACM Transactions on Graphics* (New York, New York, USA) 38 (6): 228:1–228:14.

Lun, Zhaoliang, Matheus Gadelha, Evangelos Kalogerakis, Subhransu Maji, and Rui Wang. 2017. "3D Shape Reconstruction from Sketches via Multi-view Convolutional Networks." In *2017 International Conference on 3D Vision (3DV)*, 67–77. Qingdao, China: IEEE.

Malik, Jitendra. 1987. "Interpreting Line Drawings of Curved Objects." *International Journal of Computer Vision* 1 (1): 73–103.

Nicolet, Baptiste, Alec Jacobson, and Wenzel Jakob. 2021. "Large Steps in Inverse Rendering of Geometry." *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 40 (6): 248:1–248:13.

Nimier-David, Merlin, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. 2019. "Mitsuba 2: A Retargetable Forward and Inverse Renderer." *Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 38 (6): 203:1–203:17.

Pratt, William K. 2007. "Digital image processing: PIKS Scientific inside." Chap. 18, 4:623–650. John Wiley & Sons, Ltd.

Rosenfeld, Azriel. 1979. "Digital Topology." *The American Mathematical Monthly* 86 (8): 621–630.

Rusinkiewicz, Szymon, and Marc Levoy. 2001. "Efficient Variants of the ICP Algorithm." In *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, 145–152. Quebec City, Quebec, Canada: IEEE.

Schmidt, Ryan, Brian Wyvill, Mario Costa Sousa, and Joaquim A Jorge. 2006. "ShapeShop: Sketch-Based Solid Modeling with BlobTrees." In *ACM SIGGRAPH 2006 Courses*, 14–es. Boston, Massachusetts, USA: ACM.

Shao, Cloud, Adrien Bousseau, Alla Sheffer, and Karan Singh. 2012. "CrossShade: Shading Concept Sketches Using Cross-Section Curves." *ACM Transactions on Graphics* (New York, New York, USA) 31 (4): 45:1–45:11.

Sin, Fun Shing, Daniel Schroeder, and Jernej Barbič. 2013. "Vega: Nonlinear FEM Deformable Object Simulator." In *Computer Graphics Forum*, 32:36–48. 1. Oxford, Uk: UK: Blackwell Publishing Ltd.

Smirnov, Dmitriy, Mikhail Bessmeltsev, and Justin Solomon. 2021. "Learning Manifold Patch-Based Representations of Man-Made Shapes," 1–24.

Su, Wanchao, Dong Du, Xin Yang, Shizhe Zhou, and Hongbo Fu. 2018. "Interactive Sketch-Based Normal Map Generation with Deep Neural Networks." *Proceedings of the ACM on Computer Graphics and Interactive Techniques* (Montreal, Quebec, Canada) 1 (1): 22:1–22:17.

Sultana, Farhana, Abu Sufian, and Paramartha Dutta. 2020. "Evolution of Image Segmentation using Deep Convolutional Neural Network: A Survey." *Knowledge-Based Systems* 201:106062:1–106062:38.

Tatarchenko, Maxim, Alexey Dosovitskiy, and Thomas Brox. 2017. "Octree Generating Networks: Efficient Convolutional Architectures for High-resolution 3D Outputs." In *2017 IEEE International Conference on Computer Vision (ICCV)*, 2107–2115. Venice, Italy: IEEE.

Vincent, Luc, and Pierre Soille. 1991. "Watersheds in Digital Spaces: An Efficient Algorithm Based on Immersion Simulations." *IEEE Transactions on Pattern Analysis & Machine Intelligence* 13 (6): 583–598.

Wang, Jiayun, Jierui Lin, Qian Yu, Runtao Liu, Yubei Chen, and Stella X Yu. 2020. "3D Shape Reconstruction from Free-Hand Sketches." *arXiv preprint arXiv:2006.09694*, 1–13.

Wang, Nanyang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. 2018. "Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images." In *Computer Vision – ECCV 2018*, 55–71. Springer International Publishing.

Wang, Weiyue, Duygu Ceylan, Radomir Mech, and Ulrich Neumann. 2019. "3DN: 3D Deformation Network." In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 1038–1046. Long Beach, California, USA: IEEE.

Wang, Xiaolong, David Fouhey, and Abhinav Gupta. 2015. "Designing Deep Networks for Surface Normal Estimation." In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 539–547. Boston, Massachusetts, USA: IEEE.

Wu, Jiajun, Yifan Wang, Tianfan Xue, Xingyuan Sun, Bill Freeman, and Josh Tenenbaum. 2017. "MarrNet: 3D Shape Reconstruction via 2.5D Sketches." *Advances in Neural Information Processing Systems* (Long Beach, California, USA) 30:540–550.

Wu, Jiajun, Chengkai Zhang, Xiuming Zhang, Zhoutong Zhang, William T Freeman, and Joshua B Tenenbaum. 2018. "Learning Shape Priors for Single-View 3D Completion and Reconstruction." In *Proceedings of the European Conference on Computer Vision (ECCV)*, 646–662. Springer International Publishing.

Xiang, Nan, Ruibin Wang, Tao Jiang, Li Wang, Yanran Li, Xiaosong Yang, and Jianjun Zhang. 2020. "Sketch-based modeling with a differentiable renderer." *Computer Animation and Virtual Worlds* 31 (4-5): e1939:1–e1939:12.

Xiangyu, Jin, Liu Wenyin, Sun Jianyong, and Zhengxing Sun. 2002. "On-line Graphics Recognition." In *10th Pacific Conference on Computer Graphics and Applications, 2002. Proceedings.* 256–264. Beijing, China: IEEE.

Xu, Qiangeng, Weiyue Wang, Duygu Ceylan, Radomir Mech, and Ulrich Neumann. 2019. "DISN: Deep Implicit Surface Network for High-quality Single-view 3D Reconstruction." *Advances in Neural Information Processing Systems* (Vancouver, Canada) 32:490–500.

Yan, Xinchen, Jimei Yang, Ersin Yumer, Yijie Guo, and Honglak Lee. 2016. "Perspective Transformer Nets: Learning Single-View 3D Object Reconstruction without 3D Supervision." *Advances in Neural Information Processing Systems* (Barcelona, Spain) 29:1696–1704.

Yi, Li, Vladimir G Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. 2016. "A Scalable Active Framework for Region Annotation in 3D Shape Collections." *ACM Transactions on Graphics* (New York, New York, USA) 35 (6): 210:1–210:12.

Yifan, Wang, Felice Serena, Shihao Wu, Cengiz Öztireli, and Olga Sorkine-Hornung. 2019. "Differentiable Surface Splatting for Point-based Geometry Processing." *ACM Transactions on Graphics* (New York, New York, USA) 38 (6): 230:1–230:14.

Zhou, Qingnan, and Alec Jacobson. 2016. "Thingi10K: A Dataset of 10,000 3D-Printing Models." *arXiv preprint arXiv:1605.04797.*

Appendices

A git-Repository

According to the respective guidelines.
The repository must be uploaded to the MMT/HCI git server gitlab.mediacube.at
https://gitlab.mediacube.at/fhs41347/masterthesis_hofer_kerstin