Utrecht University

By: Adrián Briceño Aguilar
6189865

Reconstruction of 3D Shapes from Cross-Sections using Indicator Networks

Supervised By:
dr. A. Vaxman
Department of Information
and Computing Sciences

1 Introduction

Reconstructing a 3D object from its cross-sections is useful for many applications. For example, reconstructing the surface of an organ taken from an MRI, radiologist have to manually delineate the boundary of the organ on selected 2D slices of the volume. Using multiple of these boundaries one could reconstruct a 3D shape of the organ, however there is multiple ways that can be done.

There is a considerable amount of previous work on surface reconstruction from slices, where the state-of-the-art Algorithms can create watertight surfaces and can address the topological correctness of their output. Unfortunately, no such work has satisfying results concerning the geometric features of the reconstructed surface.

More intricate geometric features could be extracted with the help of machine learning, making the algorithm less reliant on a specific geometric model.

In this work, we propose to tackle the reconstruction of 3D objects from cross-sections with the help of deep learning, focusing on a approach driven by data. Typically, such techniques are used to learn high-level features of 2D images, where their applications range from classification, segmentation to point cloud reconstruction. They can be used in several 3D non-euclidean spaces, such as meshes, point clouds, but one there are no existing definitions for them in the case of cross-sections.

1.1 Contributions

We introduce an algorithm that reconstructs a 3D object using only cross-sections of the original object. By introducing deep binary classification. Creating a continuous function that can predict whether a point is inside of the 3D object in question, or not. Then we use this newfound function to extract the polygonal mesh. Our main contributions are:

- Introducing deep binary classification to the problem of reconstruction from slices.
- Creating a binary classification neural network which creates a continuous function that can predict whether a point is inside and outside of the reconstructed object.

2 Literature review

In this section, we review previous work relating both sides, 3D object reconstruction and deep learning.

2.1 Reconstruction from slices

Reconstructing surfaces of a 3D object from its cross-section is referred as reconstruction from slices. In general, the input is a set of planar cross-sections of an unknown 3D object, which we intend to reproduce. These methods vary by approach and implementation.

An obvious solution to the problem of reconstructing a set cross-section as a 3D object is to interpolate between each slice to recreate a crude representation. Early methods focused on creating polyhedral envelopes around parallel cross-sections. As an example, Welzl and Wolfers (1994) connect two simple polygons P and Q in parallel planes, by a polyhedral surface scaling them with a centre. With this, the resulting surface is the portion of the cone defined by P and apex c between the two planes.

More recent methods handle shape reconstruction from the unorganized (non-parallel) cross-section. In [Boissonnat and Memari (2007)] they compute the arrangement of the cutting planes. Then, in each cell of the arrangement, they reconstruct an approximation of the object from its intersection with the boundary of the cell. In a post-processing stage, they glue the results from different cells together.

Other methods can handle multi-labelled cross-sections, [Barequet and Vaxman (2009)] made this possible by using straight skeletons of every cell of the arrangement of cross-sections. Then for each of the skeletons, they projected the images from every base face onto the other faces of that skeletal cell. Then, they computed the overlay of the projected images on both sides and reconstructed portions of the mesh from these triangulated overlays. Finally, they stitch them together in a consistent orientation and apply a mesh fairing. [Sharma and Agarwal (2017)] method uses a split-and-merge approach, utilizing Hermite mean-value interpolation for triangular meshes. They based this method on the construction of a globally consistent signed distance function over the cutting planes.

These methods can create decent and numerical robust results. However, they rely on explicit geometric interpolation, making post-process smoothing an obligation. Explicit methods are not very useful for our goals since they are geometric interpolation making them not learnable through deep learning algorithms. Moreover,

explicit approaches depend intimately on the quality of the contours and are challenging to control and generalize.

Implicit Methods. Another approach to reconstruction is to create an implicit function for each cross-section and then smoothly interpolate between the functions. They can be seen as a height field $f(x)$ over a 3D domain, where the surface is defined by the set of all points x such that $f(x) = 0$. This method was introduced by [Turk and O'brien (2005)], relying on the scattered data interpolation to solve the shape transformation problem. Another similar method was created by [Marker et al. (2006)] that used Multi-level Partition of Unity (MPU) implicit functions. They fitted them to the input contours, defined as binary images to smooth curves with controllable error bounds. [Braude et al. (2007)] created a smooth closed surface from a 2D contour extracted from a 3D scan. It uses a similar method and implicit function as the [Marker et al. (2006)]. More recently [Bermano et al. (2011)] constructed a indicator function, whose zero set is the surface, interpolating the function into the cell using mean-value transfinite interpolation, which generates a continuum of values.

More recently [Huang et al. (2018)] tackled the problem of reconstruction from incoherent cross-sections, and the fact that is independently generated by one another. They proposed a method that takes the input as an arbitrary number of non-parallel each portion into two or more labels by a curve network, represented as implicit functions. Then, they formulate the deformation task a constrained optimization on the function values with a quadratic energy objective.

Using implicit functions to interpolate between them does not yield a satisfactory reconstruction of the object since they rely on some energy-minimizing blending. Which more often than not over-smooths geometric features and creates "minimal surface" artefacts. To alleviate this problem, we offer a data-driven approach that automatically figures out, a continuous parameterization of the implicit function, using deep binary classification networks.

Topology control in surface reconstruction. Controlling the topology of the output is a way to make sure the quality of the its result, an example of this [Yin et al. (2014)], their method factored 3D editing into two "orthogonal" interactions acting on skeletal and profile curves of the underlying shape, controlling its topology and geometric features, respectively. Another possibility is template fitting; Zeng et al. (2008) did this by reducing to a maximum-flow problem, in a way that allows consideration of topology constraints. However, these methods are not fully automatic and require non-trivial user interaction.

Post-processing methods generally do not know the data from the created surface; they could make repair decisions that deviate from the inputs. One example of this is made by [Ju et al. (2007)], who removed topological errors from an existing surface. Given a source and a target shape given by the user, the algorithm modifies the source so that the resulting model is consistent with the target topologically.

Other methods attempted to incorporate topology constraints in an automatic reconstruction algorithm. [Sharf et al. (2006)] proposes a method designed for point clouds, and which allows control over the genus of the result. These algorithms search possible candidates within each cell of spatial subdivision and measure how suitable it is of each candidate as energy.

More recently [Huang et al. (2017)] created an explicit topology-controlled algorithm for reconstruction of multi-labelled material. Their algorithm combines the use of curve network and labelling regions by types of materials, where for each of labels the user has the option to constrain the number of connected components and genus. Like this, the curve network is interpolated in a way that satisfies the topological requirements.

These methods main concern is to fix topological errors while reconstructing the object. Unfortunately, the geometric fidelity is left unchecked. And most of them require non-trivial user interaction.

2.2 Geometric deep learning

Deep learning refers to the understanding of complicated concepts by building them from simpler ones in a hierarchical way. Because of the ever-growing computational power of modern GPU based computers, it is possible more than ever to train neural networks with many layers and degrees of freedom. The technological advances have led to breakthroughs in many tasks, such as speech recognition, image analysis and computer vision. Nowadays learned models are seamlessly integrated into widely-used technology, such as Siri (Apple's AI assistant) and Google translate. In recent years, learning has become relevant to geometric applications as well. However, there are distinct challenges when trying to define data-driven models on non-Euclidean domains.

The method created by [Henaff et al. (2015)] studied the ability to create network trained by non-Euclidean data. However, it does not apply to more general geometric data, such as implicit functions from cross-sections.

Another approach for deep geometric learning that works with triangular meshes is described by [Monti et al. (2017)]. They proposed a unified model allowing to

design CNNs architectures on non-Euclidean domains such as graphs and manifolds. They formulate convolution-like operators as template matching with local patches. Their method generalizes previous techniques applicable in different geometric deep leaning tasks, but lacks a more unconventional application, like learning geometric features for slices. However, this still is relevant to our work since they do create an architecture, that can be generalized for the cross-section.

More recently [Atzmon et al. (2018)] generalized the convolution operator on point cloud space using an extension and restriction operators. Such operators map the point clouds to an ambient Euclidean space. With these reductions, they were able to use it to create primary classification and segmentation network for point clouds. The main limitation of this framework compared to image CNN is the extra computational burden that the extension and restriction operator create. The applications of these methods concentrate on developing previous techniques in new spaces. Nevertheless, there is not a methodology that tries to contextualize these techniques to the new space, and it does not trivially extend to the cross-section type of data. However, this work creates a novel method to study point clouds with CNNs and could be helpful to create convolutional layers for slices.

Shape modelling with sketches is another interesting problem, since it has some similarities with reconstruction from slices, as they are both accept contours as input. Sketch-based modelling strives to bring the ease and immediacy of drawing to the 3D world. However, sketch-based modelling has relevant applications not only in an artistic way. Like [Han et al. (2017a)] who created a 3D face and caricature modelling system; It allows the user to draw freehand imprecise yet expressive 2D lines representing the contours of facial features. For this, they use a novel CNN based deep regression network that is designed to infer 3D face models from 2D sketches. Their network fuses both CNN and shape-based features of the input sketch and has two independent branches of fully connected layers generating independent subsets of coefficients for the face representation. But sketch modelling can be used to recreate a human face in a 2D image, this task is called image translation, recently [Yi et al. (2017)] used a Conditional Generative Adversarial Networks (GANs) and dual-GAN to be trained from two sets of unlabeled images, the primal GAN learns to translate images from domain U to those in domain V, while the dual GAN learns to invert the task. This closed-loop made by the networks allows images from either domain to be translated and then reconstructed. Sketch modelling has 3D reconstruction applications as well, [Delanoy et al. (2017)] created a network able to learn how to reconstruct 3D shapes from one or more drawings, the core of the method is a CNN that predicts occupancy of a voxel grid from a line drawing. Unfortunately, this type of drawing is highly detailed, and many need more than one to able to recreate the desired object with enough detail and it still lacks geometric fidelity.

Sketch modelling seems to be helpful for our work since they do infer information that is missing, and learn a contour-based database to train the algorithm to fill missing information.

Another exciting application of deep learning is when a trained model is used to find missing data and then infers the missing content, one example of this was made by [Yeh et al. (2017)] who created a method for semantic image inpainting, which generates the missing content by inferring it with the help of the available data. This work is relevant because it deals with inferring the missing data, an essential part of reconstruction from a set of slices. [Karras et al. (2018)], created a generator that enables them to train a network on human faces to create images of human faces that are not real persons. This two researchers used a special type of deep learning algorithm, called GANs, they consist of two sub-networks who has two different goals and for generating or infer missing information. Another similar application of these types of networks is [Wang et al. (2017)], they created a 3D inpainting method who can infer missing data with semantic plausibility and contextual details, by combining Encoder-Decoder Generative Adversarial Network(3D-ED-GANs) who is in charge of filling information and a Longterm Recurrent Convolutional Network (LRCN) by handling the 3D models as 2D slices transforms a coarse 3D shape into a more complete and higher resolution volume. A similar application was made by [Han et al. (2017b)], they created a data-driven method that can recover missing information of a 3D shape, by using 2 different networks a global structure inference network that incorporates a long short-term memorized context fusion module (LSTM-CF) that infers the global structure of the 3D object and a local geometry refinement network. Both of these applications have a different implementation; however, both create compelling 3D in paintings of point clouds. There is a considerable amount of work using GANs focusing on point clouds because three-dimensional geometric data offers a domain excellent for studying representation learning in generative modelling. [Achlioptas et al. (2018)] created multiple networks for 3D recognition tasks that were able to outperform most of the existing methods. On the other hand, [Li et al. (2018)] proposed a modification of the GAN algorithm for learning to generate point clouds, by combining ideas from hierarchical bayesian modelling and implicit generative models. These methods are highly relevant for our work since they create methods that can infer information based on contextual details and they are using a data-driven approach we think is the key to create better 3D reconstruction from slices.

Super-resolution is another application that is interesting to this work because of the restrictive amount of data available to create a highly detailed image.[Sajjadi et al. (2018)] made an end-to-end trainable frame recurrent video super-resolution framework that uses the previously inferred HR estimate to super-resolved the next

frame. Such methods are relevant to our problem since we seek a way to infer the missing information between each the cross-sections and then reconstruct the object.

Another original approach for the reconstruction of point clouds is made by [Williams et al. (2018)]. They over-fit a neural network representing a local chart parameterization to part of a point cloud, by using a distance function as a measure of approximation. By jointly fitting many of these networks to overlapping parts of the point cloud, they compute a manifold atlas and finally by sampling this atlas they can produce a dense reconstruction of the surface approximating the input cloud. Since this work is about reconstruction and deep learning with a novel approach to train models with a data-driven approach, it was some potential to reconstruct slices with great details. This approach is fascinating since it uses the optimization capabilities of neural networks and therefore can create high fidelity reconstructions. Its main idea of over-fitting the network seems excellent for our goal of creating high detailed reconstruction out of cross-sections.

Occupancy networks created by [Mescheder et al. (2019)], is another novel way to deal with the problem of three dimensions not having a non-canonical representation, which is both computationally and memory efficient. Their approach was to implicitly represent the 3D surface as the continuous decision boundary of a deep neural network classifier, by approximating occupancy, not only at fixed discrete 3D locations but for every possible 3D point. They did so by assigning to every location $p \in \mathbb{R}^3$ a probability between $[0, 1]$ attributed to occupancy. This approach seems fitting to cross-section reconstruction since in some method they construct an implicit function, who is in charge of indicating if a point is inside or outside of the 3D object. Our approach takes advantage of this and creates a neural network that assigns every location $p \in \mathbb{R}^3$ to a binary class 0 or 1 that encodes whether it is inside or outside of the object.

Machine learning methods are great at understanding latent information and creating new information based on it; this is precisely what we are setting out to do creating, a method that can infer what is between the slices with high geometric fidelity. Generative models have great potential to recreate the 3D object with more geometric details than the other methods. However, the occupancy networks introduced an easy way to find a continuous implicit function and by overfitting the network as [Williams et al. (2018)] did, we think it could create a highly accurate implicit function to reconstruct the object.

3 Background

3.1 Problem Description

Let be $\{C_0, ..., C_{N-1}\}$ be a set of N cross sections were C_i belongs to a plane P_i, we want to construct the boundary surface S of an object O, such that $S(p_x, p_y, p_z) = C_i$, where $p \in P_i$.

$$S(p_x, p_y, p_z) = C_i \tag{1}$$

Cross Sections. Cross-sections (or slices) are the intersection of an object in three-dimensional space with a flat surface P_i that is both infinitely large and with zero thickness, that obeys the equation:

$$ap_x + bp_y + cp_z = d \tag{2}$$

These intersections contain one or more closed non-intersecting curves, which their function has a constant value; these curves are also known as contour lines. They delimit the region around them by implicitly separating it on outside and inside of the object. They can also contain any number of holes inside of them.

3.2 Implicit Functions

The methods that use implicit functions for reconstruction are diverse in their implementation and approach. We are defining an indicator function I, a scalar function on $p \in P_i$ that assigns a binary value to indicate the type of region they bound, in our case the inside region is denoted by Ω. The function f is:

$$I(p) = \begin{cases} 1, \, p \in \Omega \\ 0.5, \, p \in \delta\Omega \\ 0, \, p \notin \Omega \end{cases} \tag{3}$$

For each plane P_i we have full knowledge of $\Omega \cap P_i$, we would like to have this function not only defined in P_i but overall $\mathbb{R}^3$, we find this interpolation by creating a neural network, that takes $\Omega \cap P_i$ as our training data, and we call the resulting function the *reconstructed indicator function* Φ.

3.3 Machine Learning

Marching Learning algorithms are used to effectively perform a specific task without relying on explicit information and relying more on inference. Their core objective is to generalize from experience; this means that the machine algorithms needs to perform accurately on new and unseen examples/tasks after having experienced a learning data set. In a way, machine learning is related to optimization since many learning problems are formulated as a minimization of a loss function on a training set of examples. The function expresses the difference between the prediction of the trained model and the real answer of the problem (for example a predicted label and the original label).

Deep Learning And Neural Networks They are algorithms that can discover multiple levels of representation or a hierarchy of features, with higher-level defined by lower features. Neural networks use a collection of nodes called "neurons", they calculate a "weighted sum" of its input, add a bias and then decides whether it should pass the output to the next neuron or not, i.e. if this neuron is activated or not. A neuron is defined as:

$$\phi = \sum_i (\theta_i p_i) + bias \tag{4}$$

Where θ_i is the collection of weights in a single neuron, and ϕ is the output of the neuron. By merely stacking a collection of neurons, we obtain a layer who can create more complicated features that can result in more accurate predictions.

Activation Function is the one deciding whether or not the neuron is activated or not. There are different types of activation functions that are useful in different tasks.

For example Rectifier functions or ReLU, are defined on a single neuron:

$$f(\sum_i (\theta_i p_i) + bias) = (\sum_i (\theta_i p_i) + bias)^+ = max(0, (\sum_i (\theta_i p_i) + bias)) \tag{5}$$

this function ensures that our output doesn't go below zero, this will deactivate the neurons that are below this threshold. However deactivating the neurons completely is not the the best option in all cases. In some cases we would like to keep them active but close to zero, so we do not lose completely future connections with other neurons. This is possible using LeakyReLU, this is defined as:

$$f(\sum_i(\theta_i p_i)+bias) = max(0,(\sum_i(\theta_i p_i)+bias))+0.01*min(0,\sum_i(\theta_i p_i)+bias) \quad (6)$$

Instead of going to zero in the negative values, we make it a small fraction in the negative part.

The Loss function estimates how closely the distribution of predictions made by a neural network matches the distribution of target variables in the training data, under the maximum likelihood framework. It means that the loss function is responsible for penalizing the neural network whether or not it predicts an incorrect value.

For example, when we are seeking to find the error between two probability distributions, we can measure it by using cross-entropy. It seeks set weights that minimize the difference between the model's predicted probability distribution given the dataset and the distribution of probabilities in the training dataset. This function is defined as:

$$L = \frac{\sum_i^n l_i}{n}; l_j = -\theta_j[y_j log(x_j) + (1-y_j)log(1-x_j)] \quad (7)$$

where x is the set of the distributions of probabilities in the training dataset, and y is the predicted probability distribution.

Back propagation Normally there needs to be a way to guarantee us that our solution is indeed the best, back propagation does just that by finding a local minimum of a function, by taking steps proportional to the negative gradient at the current point, also known as *gradient descent*.

Learning Rate is a hyper-parameter that controls how much we adjust the weights of the network concerning the loss gradient. The lower the value, the slower we travel along the downward slope.

Overfitting is the production of an analysis that corresponds too closely to a particular set of observations, and it may, fail to predict future outcomes, from new observations, reliably. In the case of neural networks, this might occur for multiple reasons, for example, if the number of the weights (i.e. the model's parameters) is

much higher than the number of training samples, this easily avoidable by creating not too big of a network.

One most common reasons for the mode to overfit is when the network start to become too complex, or in simpler terms, when starts creating too high values in the weight matrix. We can avoid it by adding an extra element to the loss function, which punishes our model from becoming too complex.

Regularization methods, punishes the neural networks for creating too high values in the weight matrix, they do so by adding an extra term in the loss function that is multiplied by a factor λ, also know as regulation rate, if we increase this value we increase the regulation effect.

For example ADAM [Kingma and Ba (2014)], is an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments; this a Least Square Errors method. The main contribution of this method is that they calculate the optimal value to use for the weight decay, it actually depends on the number of iteration during training, this means that the factor λ in this method is not a hyper-parameter. This method is appropriate for problems with very noisy or sparse gradients.

3.4 Deep Classification Networks

Binary classification is the task of classifying the components of a given set in two classes, predicting in which they belong.

This task can be solved easily by learning from existing supervised (pre-classified) data. Creating a neural network whose parametrization creates a function that maps a set of classifiers $c \in \mathbb{R}^n$ to a probability of either belonging to a class $p \in [0, 1]$ represented as:

$$\Phi : \mathbb{R}^n \to [0, 1] \tag{8}$$

since there are only two possible classes to belong to, the probability of a specific element of the set to belong to the other class is easily calculated by $1 - p$.

Cut-off or threshold , in the context of binary classification, is the probability that the prediction is right. It represents the trade-off between false positives and

false negatives. If this probability is higher than 0.5 this means that the classification is not random. If we choose 0.6, for example, the predicted outcome with a probability above it will be classified in the first class.

3.5 Deep Geometric Prior Surface Reconstruction

Our main interest in this method is the use of a neural network as an optimization tool. Therefore they overfitted the network to be able to reconstruct the 3D object with high fidelity.

[Williams et al. (2018)] choose Φ to be a MPL with half rectified ativation function:

$$\Phi(v;\theta) = \theta_{k-1} ReLU(\theta_{k-1} ReLU ... ReLU(\theta_1 v)), \tag{9}$$

where $\theta_i, i = 1...K$ are per-layer weights matrices, and v is the input of the network. They over-parametrize the network such that the total number of trainable parameters $T = dim(\theta_1) + ... + dim(\theta_k)$. This creates the desired outcome, the function created by this network is unable to predict future observations reliably, it only can create the desired parameterization of an specific input.

3.6 Occupancy Networks

Occupancy Networks introduces a new type of representation of 3D geometry, as the continuous decision boundary of a deep neural network classifier.

They create an Occupancy function at every possible 3D point in $p \in \mathbb{R}^3$:

$$o : \mathbb{R}^3 \rightarrow \{0, 1\} \tag{10}$$

Approximating it with a neural network that assigns to every location $p \in \mathbb{R}^3$ to an occupancy probability between 0 and 1. This problem is equivalent to binary classification.

They do so by changing their input of the network since they are trying to achieve a 3D reconstruction of an object based on observations of that object (e.g. images, point clouds, an so forth). They used a clever the following function equivalence: a function that takes an observation $x \in X$ as input and has a function from $p \in \mathbb{R}^3$ to $\mathbb{R}$ as a function can equivalently describe output that takes a pair $(p, x) \in \mathbb{R}^3 \times X$ as input and outputs a real number. This can be parameterized by a neural network represented by:

$$f_\theta : \mathbb{R}^3 \times X \to [0, 1] \tag{11}$$

The main difference our methods between theirs, that they create a learning method where its input is entirely implicit, like images. Also, they reconstruct by learning from the previous reconstruction. Our only concern is to use the neural network as a highly accurate optimization tool.

4 Methodology

4.1 Overview

The input of our algorithm is a set of planar cross-sections,along with the orientation parameters of the planes. Our goal is to create an oracle function that can predict a probability of how likely a point is inside or outside of a 3D object. To make this possible, we sample uniformly each of planes P_i and determine whether each sampled point is inside or outside of the object. However, we would like to have this function not only in a fixed discrete 3D locations but at every possible 3D point $p \in \mathbb{R}^3$ and calling it the reconstructed indicator function:

$$I : \mathbb{R}^3 \to \{0, 1\} \tag{12}$$

We can approximate this function by using a neural network that classifies every point $p \in \mathbb{R}^3$ as inside or outside of the object that we want to reconstruct. Therefor our desired function is equivalent to a neural network for binary classification classification like:

$$\Phi : \mathbb{R}^3 \to [0, 1] \tag{13}$$

that takes a set of points $p_n \in \mathbb{R}^3$ and outputs the likelyhood of belonging inside of the 3D object.

Our contribution is to design a network that over-fits to a specific input and tries to create a faithful reconstruction. After this network is successfully over-fitted, we sample 3D space uniformly, to predict the indicator value for each sampled point to use later as input to Marching Cubes and extract the polygonal mesh.

4.2 2D Sampling.

For each plane P_i we generate a set of samples $\{p_k, \iota_k\}_i$ with location p_k and indicator value ι_k, according to resolution hyper-parameter n. In addition we create a set of 2D grids acting as a bounding box where all points on those grids are classified as outside of the object; this is to make sure the network implicitly learns that points outside of this bounding box can be considered outside of the object, and to avoid creating a tunnel-like effect, where the reconstructed mesh continuous infinitely in a certain direction where is not data to said it otherwise. Then we aggregate all sets $\{...\}_i$ adding into a unified set $\{\mathcal{P}, \mathcal{I}\}$ of all classified data points.

4.3 Neural Network Architecture.

To learn the exact indicator function of an specific 3D object we chose the network Φ to be a Classification network with the half-rectified activation function:

$$\Phi(v;\theta) = \theta_K ReLU(\theta_{K-1} ReLU...ReLU(\theta_1 v)), \tag{14}$$

Where $\theta_i, i = 1...K$ are per-layer weights matrices and the parameters of the network. If we choose, the total amount of parameters by $T = dim(\theta_1 + ... + \theta_K)$ be greater than 3 times the training samples t we make $T >> 3t$. The main goal to over-parametrize is to over-fit the network; this is possible if we add a small learning rate as well, a high number of epochs and a high number of training samples.

As the task of this network is to binary classify points, we choose to use a Cross-Entropy loss function, since is often used as the preferred criterion and can easily be generalized to a multiple labels classifier. We choose to use ADAM as our regularization method since this calculates the optimal regularization rate, and this simplifies our method by letting us not worry about this particular hyper-parameter.

4.4 Reconstruction of the 3D Object.

We reconstruct the object by uniformly sampling 3D space, creating a 3D grid according to a resolution hyper-parameter m around the region of interest. We use our over-fitted network to predict the indicator value of each of those points. Then we use the Marching Cubes Algorithm as a way to extract the polygon mesh, by using the virtual isovalue 0.5.

5 Results

5.1 Our experiments

We run our experiments on a $Intel(R)\ Core(TM)\ i7-7700HQ\ CPU\ @2.80GHz$, $8Gb$ of memory, and a $GeForce\ GTX-1050\ Ti\ Mobile\ GPU$.

We use PyTorch as our main neural network library. ADAM [Kingma and Ba (2014)] is our regularization method, implemented in PyTorch with default parameters. Using a learning rate of 0.01. For the network Φ, we use a classification network with fully connected layer, with the sizes: $(3, 128, 256, 512, 512, 1)$ making sure that our total amount of parameters, since for each layer the total amount of parameters is $sizeofinput * sizeofoutput$ and the total amount T is the sum of them. And using LeakyReLU as the activation is in all the layers.

Since we are only using one GPU and our network contains many parameters, we had to divide our training data into batches, to avoid filling our GPU. We create random batches, since the input data is sorted and similar data points will lie next to each other, and we would like to avoid the slow convergence of our network.

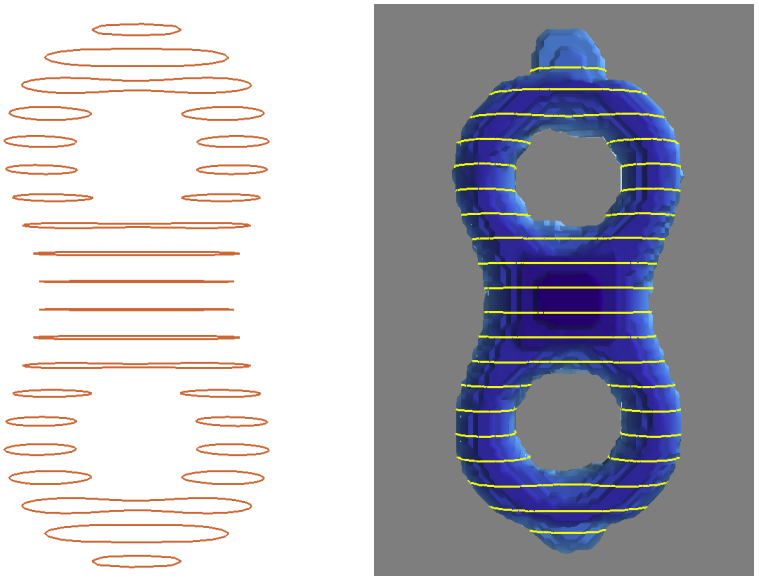
Our available data set comprises of the intersection of 3D objects with planes, allocated in .csl files containing the vertices of the object that intersects with each plane and organized by the contour they comprise. They vary, in the number of planes and objects types.

We show some examples of running our algorithm in several settings. We render the reconstructed mesh with a specific colour, and we show some lines on top of it that are some of the intersections with the planes that we have as input with a different color and we show the original cross-sections to show the interpolation created by our network.

5.2 Branching effect

First, we provide the reconstruction from cross-sections of a mesh, with parallel cross-sections as input. To show how our algorithm deals with not only parallel cross-section but multiple contours in each plane.

As we can see in Figure 1b, our network can interpolate between slices and understand the overall shape of the given input. We made this possible by giving it a high resolution on the 2D sampling, passing through the network the whole data set



(a) Input cross-sections. (b) Our Reconstruction

Figure 1: A reconstruction of a number eight mesh with parallel cross sections thought the y axis, with 216×216 2D samples and a $350 \times 350 \times 350$ 3D samples.

several times (i.e. the number of epochs = 1024) to be sure that it over-fits to our input.

If we take a closer look at the doughnut holes in reconstructed in Figure 1b mesh, we can see how our network deals with multiple contours in one plane. It creates a branching effect that implicitly reconstructs the eight shape.

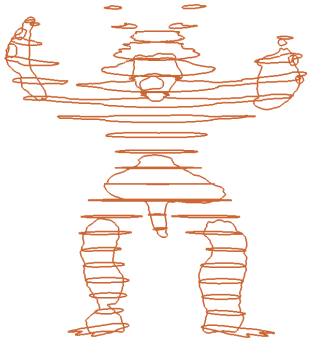
In Figure 1b, we can see that the reconstructed surface is uneven; this is a direct consequence of how we sample 3D space and the way we extract the polygonal mesh (Marching Cubes). However, this interpolation is satisfying since it creates an excellent approximation of the shape and its no to intricate details.

We can see as well at the top a bump type artefact; this might be created by a combination of the way we do the 2D sampling, made by a tunnel effect to the bounding box. And, we might be oversampling 3D space.

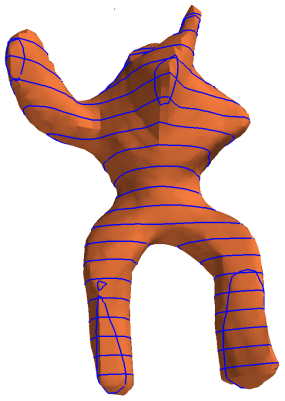
5.3 2D Sampling Effect

We next demonstrate the effects on high-resolution samples of the planes, where we create $n \times n$ grids per plane. We expected that by having low resolution, the network would not be able to recreate small details from the mesh.

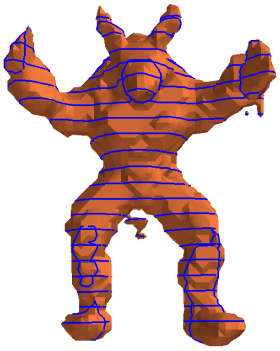
We want to reconstruct an Armadillo object using mostly slices shown in Figure 2a, with the same 3D sampling, $300 \times 300 \times 300$, and the same amounts of epochs, 512 in this case. We ran three different tests, one with 54×54 resolution on each plane as we can see on Figure 2b, another with 108×108 that we see on Figure 2c, and finally one with 216×216 as shown in Figure 2d.



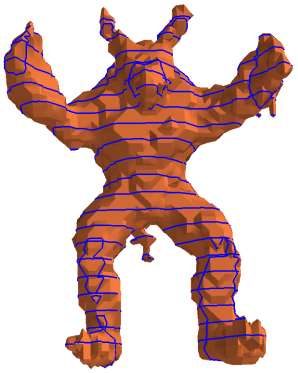
(a) Input cross-sections.



(b) A 2D sampling of 54×54 per plane



(c) A 2D sampling of 108×108 per plane



(d) A 2D sampling of 216×216 per plane

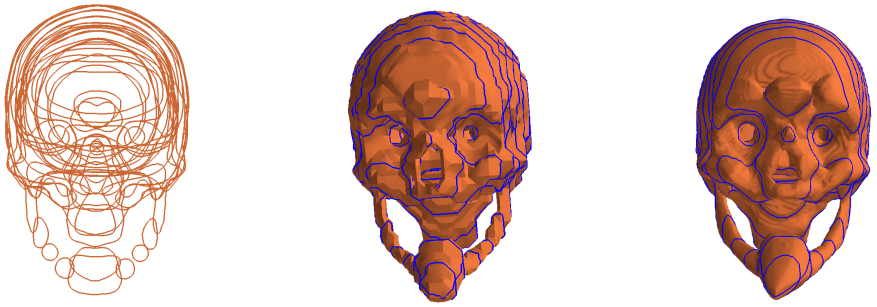
Figure 2: For reconstruction of an Armadillo using the same set of cross-section as input but different resolution of the 2D samples.

Focusing first on the low resolution reconstruction we can see on Figure 2b that the mesh is lacking critical features of the object, e.g. the hands, feet, ears and face are lacking any detail, and even it is missing an arm. This lack of interpolation is due to the fact is missing small features. We can see a smooth effect on the surface of the reconstructed mesh represented in Figure 2b. This effect appears because all the small boundary features are over the sampled resolution.

In Figure 2c we can see that the network is trying to understand this small details better and creating fingers, and creating better-looking ears, and in Figure 2d we can see greater detail on the hand's tails and feet. However, it potentially could become a better reconstruction by having even higher resolution, or creating a more sophisticated method of 2D sampling each plane.

5.4 3D Sampling Effect

Here we discuss how densely we sample in 3D space affects our reconstruction. We reconstructed a skull with cross-section mostly parallel to each other. On Figure 3a and Figure 3b, we can see the results. Generally speaking, both reconstructions are fairly impressive since the jaw details are nicely detached from the upper part of the skull, we can see fairly good were the teeth should go, other points of interest are the nose hole, the eye sockets.



(a) Input cross-sections (b) 3D Resolution of 85 (c) 3D Resolution of 300

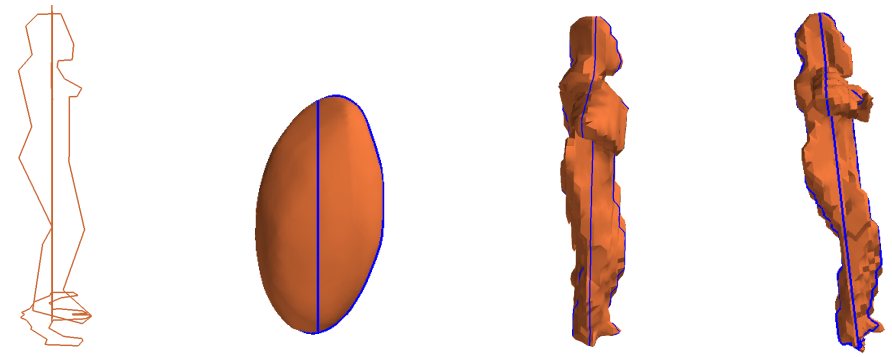
Figure 3: Reconstruction of a skull with different resolution of 3D samples

Comparing the reconstructed objects in Figure 3, Figure 3c has a smoother surface, this makes sense, since we are sampling 3D space more densely than Figure 3b reconstruction, and the marching cubes can find more vertices on the surface of the polygon mesh and create a more smooth looking reconstruction.

5.5 Convergence

Here we explore the effects of how many times we need to pass the entire dataset, forward and backward, through the neural network for it to converge to the optimal shape, i.e. how many epochs it needs to be able to converge to a robust reconstruction.

We used our standard of 216×216 2D sampling and $300 \times 300 \times 300$ 3D sampling for all of our next results. First, we explored by only passing once, then passing it 256 times and finally, we pass it 512 times, showing a set of 4 cross-sections of a human mesh, that are non-parallel to each other. Looking at the first reconstruction in Figure 4a, this lacks any shape of the mesh but is still interesting that our algorithm shapes as an oval. We can see the results in Figure 4a, and Figure 4b, we can see how the shape of the object created by our algorithm improves since we can see the general shape of the human forming.



(a) Input (b) 1 epoch passed (c) 256 epoch passed (d) 512 epochs passed

Figure 4: Reconstruction of a human mesh with low number of slices.

The network seems to need more than one passing of the training data to understand the shape and the details of the object. This makes sense since we want the network to over-fit as much as possible to the object, and passing several times the training data updates the weights of the function to be more optimized to the input data.

In the other hand comparing Figure 4b and Figure 4c, we can see small differences in certain features of the object, for example, the legs and arms seem more refined in Figure 4c, but the faces seem to have lost its details. This can be a difficult parameter to tweak, but overall, our results we found that 512 epochs are an excellent way to create good looking reconstructions.

Lastly, we want to show how our model over-fits over the epochs. We can see this by looking at how the loss function converges to zero, and Figure 5 shows this very clearly.

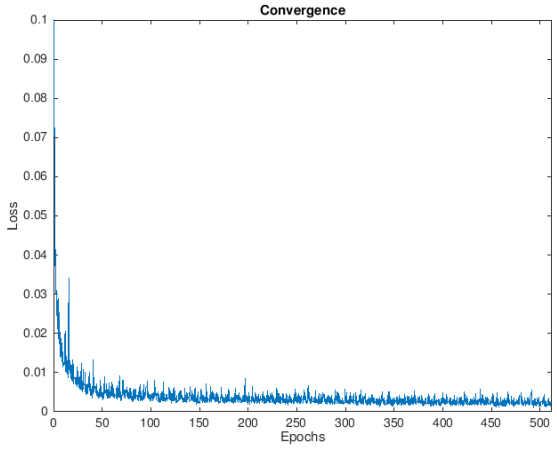
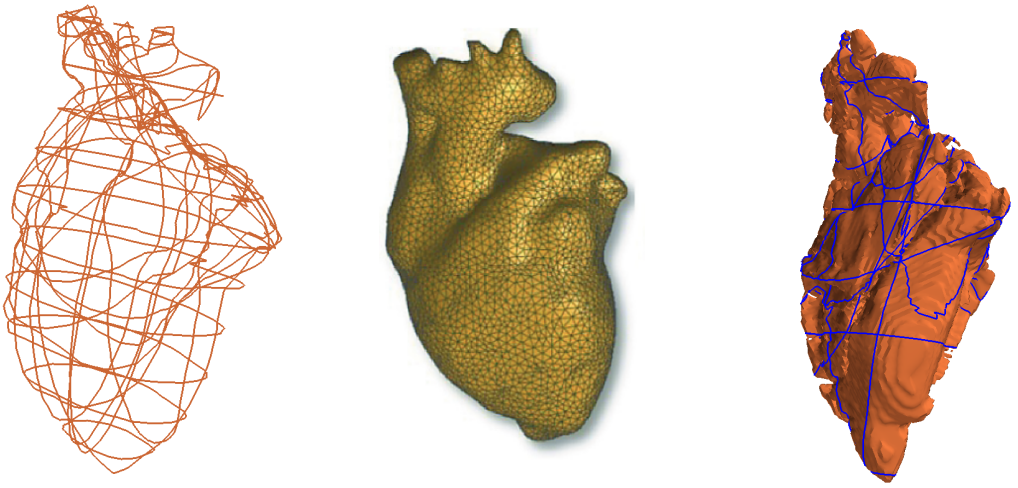


Figure 5: A function of the loss value, shown against the number of epochs passed



(a) Input cross-sections.

(b) Bermano et al. (2011) reconstruction

(c) Our Reconstruction

Figure 6: A heart reconstruction

5.6 Comparing to Bermano et al. (2011)

In this section, we are comparing our algorithm to [Bermano et al. (2011)]. We reconstructed of a Heart mesh with a set of non-parallel cross-sections with 216×216 2D samples and a $350 \times 350 \times 350$ 3D samples, and 512 epochs. Figure 6c shows the ability of our method to recreate the overall shape of the heart; however, in this reconstruction shows a lack of features missing compared to [Bermano et al.

(2011)] reconstruction. For example, the missing information in the vicinity of the arteries is not reconstructed. Instead, our method creates a lump type effect in that area. However, this might be created by the way sample in 2D, since these regions complex in terms of many different slices where they intersect with each other.

We will like to note that the bottom part of the heart in our reconstruction is reconstructed as a pointy end but should reconstruct a smoother edge, we would like to point out that [Bermano et al. (2011)] apply to smooth as a post-process.

Looking at our other results we can see that our method is capable of reconstructing finer details of the 3D object, however, choose the correct parameters takes time, and knowledge of the data set, and this is not ideal. We explore this problem in the next section.

5.7 Failing to find right configurations

On Figure 7, we can see our result of reconstructing a kitten mesh, using only 12 slices to do so. This particular reconstruction was challenging to make since we had to try different parameters to try the right configuration to get a satisfying reconstruction, and we fail to do so.

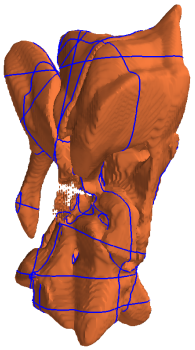
First, our first attempt was to reconstruct it with the same amount of 2D resolution as the other examples (216×216), we can see the results in Figure 7a. This result is less than optimal since we can barely see the shape of the kitten and a little of its tail.

We decided to give the network needed more resolution for it to learn more about the kitten mesh. So we did just that in the results shown in Figure 7b, we did obtain better results for the overall shape of the kitten, but it has multiple artefacts, for example, the one in its face. It seemed that some information was missing on these areas, we opt to increase the number of epochs to see if the network could better understand its shape and fill in the missing information, however, we came across with the same artefact but worse, and an elongated cat.

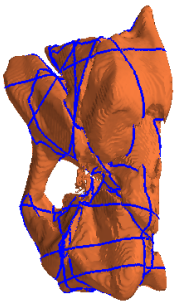
This three example shows how the algorithm is sensitive of how much data it is provided to the training data, not only that but where is that data. If our network only has information about ears, it would not be able to create the rest of the head body and legs.

We can see that without previous knowledge of the data is hard to find the correct configuration to recreate the optimal reconstruction. This might be fixable using a better way to sample each plane and finding the correct labels.

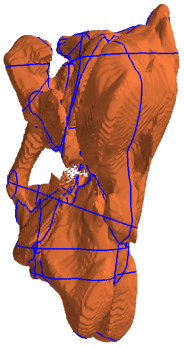
(a) Input cross-sections.



(b) Under-sampled cat(216×216)



(c) Sample each planed by 300×300



(d) Same samples as c, but had 1024 epochs

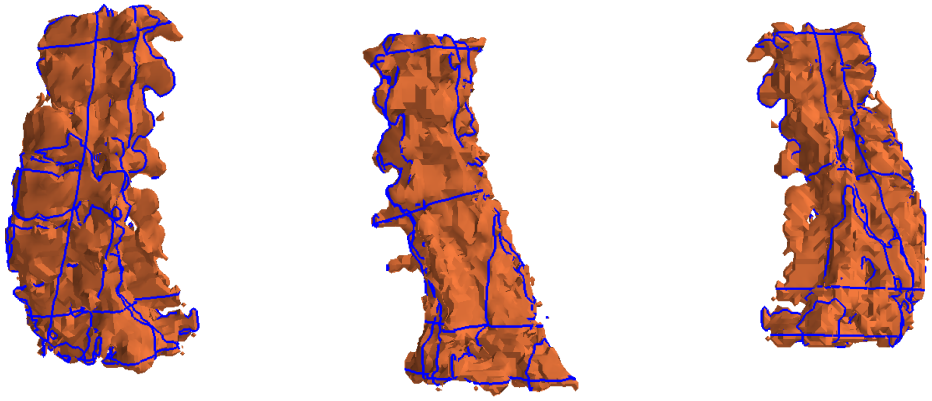
Figure 7: Reconstruction of a kitten mesh varying the amount of 2D samples and epochs on the network.

5.8 General Results

Here we show more examples of our algorithm running, discuss overall results and some artefacts that we found along the way.

We reconstructed a spine mesh made out 32 slices, the resolution that we created a reconstruction of a human spine, using mostly non-parallel slices, seen in Figure 9. We can see that our method creates a robust reconstruction of very complex data; we can see that it creates clear single vertebrae. However, we can see some detach floating polygon; this might be an artefact of how we find the polygon mesh with Marching cubes.

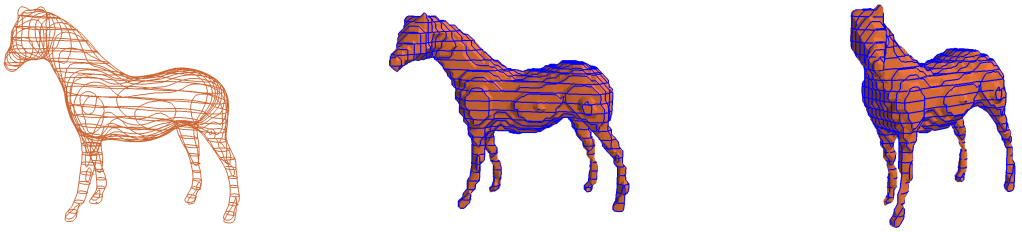
Reconstructing a horse out of 40 slices seems an easy task for our model since it



(a) Side view (b) Frontal view (c) Side view

Figure 8: Reconstruction of a spine mesh made by sampling 216×216 each plane and 3D sampling a box by $300 \times 300 \times 300$

has enough information to create a robust representation of this horse. We can see overall the voxel effect thought the surface of the object, but we can see that the ears, mouth, body and legs are reconstructed similarly.



(a) Input slices (b) Side view (c) Frontal view

Figure 9: Reconstruction of a spine mesh made by sampling 216×216 each plane and 3D sampling a box by $300 \times 300 \times 300$.

Table 1 shows that our method speed of training is directly correlated with the number of epochs we use and how many training samples give it as input. Our method seems to run quite slow, even though we run all multidimensional arrays in our *GPU*.

Table 1: Runtimes (in hours)

Object	Slices	# Epochs	# of training samples	Training time (hours)
Eight	20	1024	1213056	4.25
Armadillo	23	512	1353024	2.34
Skull	16	512	1026432	2.1
Human	4	512	1102500	1.8
Heart	25	512	1446336	2.44
Vertebrae	36	512	1959552	3.12
Kitten	12	512	839808	1.5
Horse	40	512	2099520	4.50

6 Discussion and Conclusion

We have described an algorithm which introduces deep binary classification to the the problem of reconstruction from slices, it creates a continuous scalar field, that is used to produce a robust interpolation surface corresponding to a set of cross-sections. This solves the problem of reconstruction from the arbitrarily oriented cross-section. The algorithm is easy to understand and implement. Also, it could have multiple applications outside of computer science, for example, it could be applicable to create 3D reconstructions of organs using MRI scans, this could be easy to implement since they take multiple slices of the same object, and are mostly parallel to each other, its a matter of data and image processing.

Limitations. Overall, we found that our method reconstructs robust 3D objects, and it can learn complex shapes, implicitly. However, we found the following limitations:

- Sampling over each plane influences not only how much details are extracted for our reconstructions, but as well how much time it takes for our network to create a reliable indicator function that is both continuous and over-fitted to a specific 3D object. This is unnecessary since we are sampling too much in regions that are uniform in indicator value, and that is what causes the algorithm to be very slow.
- The need to use Marching cubes as a way to extract a polygonal mesh. Since this algorithm uses a three-dimensional discrete scalar field to do so, and this is wasting our continuous scalar field (i.e. our indicator function).
- The architecture of our neural network is too big, this creates an easy way to overfit the network, however it makes our training really slow since it creates

a lot of parameters.

Future research. The direction of future research is quite clear since we have several aspects that need to be improved.

An exciting direction would be to understand the relationship between the number of slices and the quality of our reconstruction in addition to understanding the impact of using different optimization's methods, against the ADAM optimizer.

Furthermore, creating a more reliable way to sample 2D space and therefore, more reliable training data for our networks seems like a critical element to improve our algorithm. Another critical element is searching for a better algorithm to better use our continuous indicator function, instead of using a discrete algorithm as Marching Cubes. This improve our polygon mesh and therefore our reconstruction.

Generalizing our network to fit multiple labels sounds like a genetic improvement and comfortable addition to implementing since it would reconstruct multi-material polygon meshes. And creating a more sophisticated neural network architecture that can overfit, as much as possible , to the input data.

Another interesting approach to future research is to investigate what type of impact has the regularization method that we employ in our model.

Conclusion Our choice of using deep classification networks to extract a continuous indicator function finds a new and exciting solution to this problem. And there is a clear direction of future research that would improve our algorithm.

References

Achlioptas, P., Diamanti, O., Mitliagkas, I., and Guibas, L. (2018). Learning representations and generative models for 3d point clouds.

Atzmon, M., Maron, H., and Lipman, Y. (2018). Point convolutional neural networks by extension operators. *arXiv preprint arXiv:1803.10091*.

Barequet, G. and Vaxman, A. (2009). Reconstruction of multi-label domains from partial planar cross-sections. In *Computer Graphics Forum*, volume 28, pages 1327–1337. Wiley Online Library.

Bermano, A., Vaxman, A., and Gotsman, C. (2011). Online reconstruction of 3d objects from arbitrary cross-sections. *ACM Transactions on Graphics (TOG)*, 30(5):113.

Boissonnat, J.-D. and Memari, P. (2007). Shape reconstruction from unorganized cross-sections. In *Symposium on geometry processing*, pages 89–98.

Braude, I., Marker, J., Museth, K., Nissanov, J., and Breen, D. (2007). Contour-based surface reconstruction using mpu implicit models. *Graphical models*, 69(2):139–157.

Delanoy, J., Aubry, M., Isola, P., Efros, A. A., and Bousseau, A. (2017). 3d sketching using multi-view deep volumetric prediction. *arXiv preprint arXiv:1707.08390*.

Han, X., Gao, C., and Yu, Y. (2017a). Deepsketch2face: a deep learning based sketching system for 3d face and caricature modeling. *ACM Transactions on Graphics (TOG)*, 36(4):126.

Han, X., Li, Z., Huang, H., Kalogerakis, E., and Yu, Y. (2017b). High-resolution shape completion using deep neural networks for global structure and local geometry inference. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)*.

Henaff, M., Bruna, J., and LeCun, Y. (2015). Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*.

Huang, Z., Zou, M., Carr, N., and Ju, T. (2017). Topology-controlled reconstruction of multi-labelled domains from cross-sections. *ACM Transactions on Graphics (TOG)*, 36(4):76.

Huang, Z. Y., Holloway, M., Carr, N., and Ju, T. (2018). Repairing inconsistent curve networks on non-parallel cross-sections. In *Computer Graphics Forum*, volume 37, pages 25–35. Wiley Online Library.

Ju, T., Zhou, Q.-Y., and Hu, S.-M. (2007). Editing the topology of 3d models by sketching. *ACM Transactions on Graphics (TOG)*, 26(3):42.

Karras, T., Laine, S., and Aila, T. (2018). A style-based generator architecture for generative adversarial networks. *arXiv preprint arXiv:1812.04948*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Li, C.-L., Zaheer, M., Zhang, Y., Poczos, B., and Salakhutdinov, R. (2018). Point cloud gan. *arXiv preprint arXiv:1810.05795*.

Marker, J., Braude, I., Museth, K., and Breen, D. E. (2006). Contour-based surface reconstruction using implicit curve fitting, and distance field filtering and interpolation. In *Volume Graphics*, volume 2006, pages 1–9.

Mescheder, L., Oechsle, M., Niemeyer, M., Nowozin, S., and Geiger, A. (2019). Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4460–4470.

Monti, F., Boscaini, D., Masci, J., Rodola, E., Svoboda, J., and Bronstein, M. M. (2017). Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proc. CVPR*, volume 1, page 3.

Sajjadi, M. S., Vemulapalli, R., and Brown, M. (2018). Frame-recurrent video super-resolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6626–6634.

Sharf, A., Lewiner, T., Shamir, A., Kobbelt, L., and Cohen-Or, D. (2006). Competing fronts for coarse–to–fine surface reconstruction. In *Computer Graphics Forum*, volume 25, pages 389–398. Wiley Online Library.

Sharma, O. and Agarwal, N. (2017). Signed distance based 3d surface reconstruction from unorganized planar cross-sections. *Computers & Graphics*, 62:67–76.

Turk, G. and O'brien, J. F. (2005). Shape transformation using variational implicit functions. In *ACM SIGGRAPH 2005 Courses*, page 13. ACM.

Wang, W., Huang, Q., You, S., Yang, C., and Neumann, U. (2017). Shape inpainting using 3d generative adversarial network and recurrent convolutional networks. *arXiv preprint arXiv:1711.06375*.

Welzl, E. and Wolfers, B. (1994). Surface reconstruction between simple polygons via angle criteria. *J. Symb. Comput.*, 17(4):351–369.

Williams, F., Schneider, T., Silva, C., Zorin, D., Bruna, J., and Panozzo, D. (2018). Deep geometric prior for surface reconstruction. *arXiv preprint arXiv:1811.10943*.

Yeh, R. A., Chen, C., Yian Lim, T., Schwing, A. G., Hasegawa-Johnson, M., and Do, M. N. (2017). Semantic image inpainting with deep generative models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5485–5493.

Yi, Z., Zhang, H., Tan, P., and Gong, M. (2017). Dualgan: Unsupervised dual learning for image-to-image translation. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2868–2876. IEEE.

Yin, K., Huang, H., Zhang, H., Gong, M., Cohen-Or, D., and Chen, B. (2014). Morfit: interactive surface reconstruction from incomplete point clouds with curve-driven topology and geometry control. *ACM Trans. Graph.*, 33(6):202–1.

Zeng, Y., Samaras, D., Chen, W., and Peng, Q. (2008). Topology cuts: A novel min-cut/max-flow algorithm for topology preserving segmentation in n–d images. *Computer vision and image understanding*, 112(1):81–90.