# Single Image Mesh Reconstruction: Solving for 2D Data

Uday Kusupati* Abhinav Deep Singh*
The University of Texas at Austin
{uday, abhinav} @cs.utexas.edu

## Abstract

*Our goal is to develop a deep learning architecture that learns the topology and produces a 3D triangular mesh, just from a single image. In this work, we draw parallels between the three dimensional and its two dimensional counterpart. After, reducing the problem to two dimension, we propose a deep learning architecture that learns to produce multiple polygons. Our current method is based on graph convolutions along with reinforcement learning.*

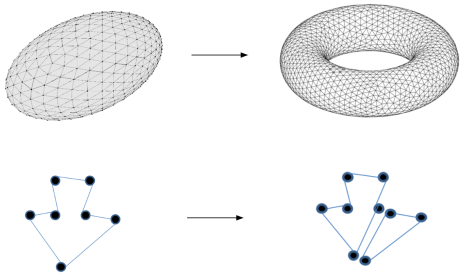

Figure 1. Topology in 3D and 2D

## 1. Introduction

Humans can easily infer the shape of an object from a single image. However, it can be challenging for the artificial systems. Recently, thanks to deep learning, single image 3D reconstruction has greatly improved. However, most of the methods are based on representations other than mesh, and therefore they lose important surface information. 3D triangular mesh representation is highly desirable as it provides a lot of information about shape and is used by a plethora of applications. Other methods which use mesh as their base representation, fail to learn the exact topology.

In this paper, we propose a network architecture that keeps the 3D triangular mesh representation preserved end-to-end, and also learns the topology. Similar to the Pixel2Mesh [17] implementation, our method learns to deform the basic shape, and add vertices. However, unlike Pixel2Mesh, we have a splitter module, that splits the shape based on the topology. This helps the network to also learn the topology of the object at the same time as learning the shape. The challenge here is to identify when and where to split the mesh. We use reinforcement learning for this. Our splitter network is an actor-critic network, trained with DDPG algorithm [12]. Finally, we deform the mesh again after splitting it, so that it learns the final shape. In order to see the feasibility of the approach, we first test this approach on two dimensional problem.

Before delving into details, we frist discuss the prior work in section 2. Then in section 3, we describe our two-dimensional toy dataset and also desribe how the problem is extendable to corresponding triangular 3D mesh. In Section 4, we describe our model architecture, and then we explain our losses. Section 5 deals with training methodology, and finally section 6 we show current results of our current method. We end with section 7, which deals with future work we plan to do as this research is currently a work in progress.

## 2. Prior Work

Most of the prior work on 3D reconstruction from single image rely on using the implicit or point cloud representations. Mesh representations are less structured and non-euclidean as they typically consist of sets of vertices and faces. Several works consider using neural networks and deep learning on the graph defined by the mesh. [15, 1, 13, 5] propose methods to use deep learning architectures on graphs and manifolds. Methods like [4] learn meshes not in an end-to-end fashion but as a post processing step. Most works focus on deforming a template to arrive at the target mesh[8, 17]. [16] introduces an approach to compactly represent meshes of similar objects and use the model to sample a good candidate template and deform it using techniques like FFD[10]. [6] uses an SMPL[14] model of human body to perform end-to-end recovery of human shape and pose.[18] and [7] deform templates of animal meshes for reconstruction of animals. More recently, [11] doesn't assume any template and performs mesh based inference to find the mesh topology as well as to reconstruct the entire mesh.

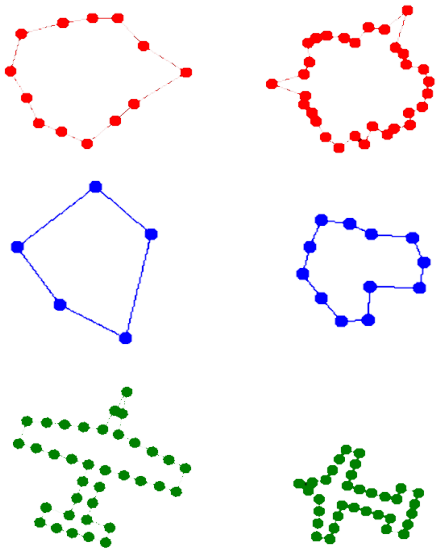

Figure 2. Toy Data: Polygons and Planes

## 3. Toy Data

We reduce the task to 2D first to perform different experiments on how neural networks perform on the task of reconstruction from a lower dimension. So the 2D image to 3D mesh reconstruction corresponds to 1D vectorized polygons/curves to 2D meshes/graphs. (See Fig. 1) We generate a synthetic dataset of 2D polygons/planes (See Fig.2) and their corresponding 1D vectorized representations. Along with the vertex positions and connectivities, we generate the normals at them. So, the task is now to deform a triangle or any pre-defined polygon to the desired polygonal output, given the input of the vector representation. But the desired output can contain multiple polygons which makes the task hard.

## 4. Model Architecture

Figure 3 shows the architecture we use. It consists of alternating deformer and split decision networks. The task of the deformer network is to deform the existing mesh to the target mesh using features from the input. The split decision block makes the decision of splitting a closed polygon and where to split too, so that the topology is changed. The alternating placement of the modules helps incremental learning of how the target mesh can be obtained by making fundamentally small changes like deform and split to the original mesh.

### 4.1. Deformer Block

#### 4.1.1 Block Architecture

The deformer block is inspired from the architecture from Pixel2Mesh[17]. It consists of a feature pooling network along with a graph convolutional network [2]. The key idea is to define a feature at each vertex. This feature is initialized with an embedding of the coordinates of the initial locations.

The feature pooling network fetches the relevant features for each vertex from the input, the vectorized polygon $s$, given the coordinates of the vertex. This obtained feature is concatenated with the existing feature on the vertex and is passed to the graph convolutional network which outputs the final feature and the deformed coordinates for each vertex. Graph convolutions are briefly explained as follows. Given a mesh $M$ represented by the vertices and edges $V = v_i$, $E = e_i$ and feature vectors defined on vertex $i$ as $F = f_i$, a graph convolution operation is defined as:

$$f_p^{l+1} = w_0 f_p^l + \sum_{q \in N(p)} w_1 f_q^l$$

where $l$ represents the graph convolutional layer and $w_0$, $w_1$ are the learnable parameters of the network. $N(p)$ is the neighbor set of vertex p. In our case the input feature at each vertex to the network is the existing feature concatenated with the pooled feature from the input vectorized polygon. We use fully connected networks in the graph convolution layers and stack $n$ layers on top of each other. The output of the deformer block are the new features $f_i$, at each vertex $v_i$ which are passed through a linear layer to obtain the deformed positions $c_i$ of the vertices.

The deformer block is built by stacking alternating graph convolution networks and vertex adders. Each vertex adder uniformly up-samples the vertices in the existing mesh by sampling a vertex at every edge center and appropriately creating new edges. It allows us to start with fewer vertices and learn a coarse to fine deformation. See Fig 4.

#### 4.1.2 Deformer Losses

The deformer is trained by different losses on the output deformation predicted. We use Chamfer Loss [3] to constrain the vertex positions, normal loss to enforce consistency of normals along with laplacian regularization and edge-length regularization for smooth, stable deformations and to prevent outliers respectively. Unless otherwise stated, we use $p$ to represent vertices on the predicted mesh and $q$ to refer ones on the ground truth mesh.

**Chamfer Loss:** The Chamfer distance is a metric to measure the similarity of point clouds. It is defined as

$$l_c = \sum_p min_q \|p - q\|_2^2 + \sum_q min_p \|p - q\|_2^2$$

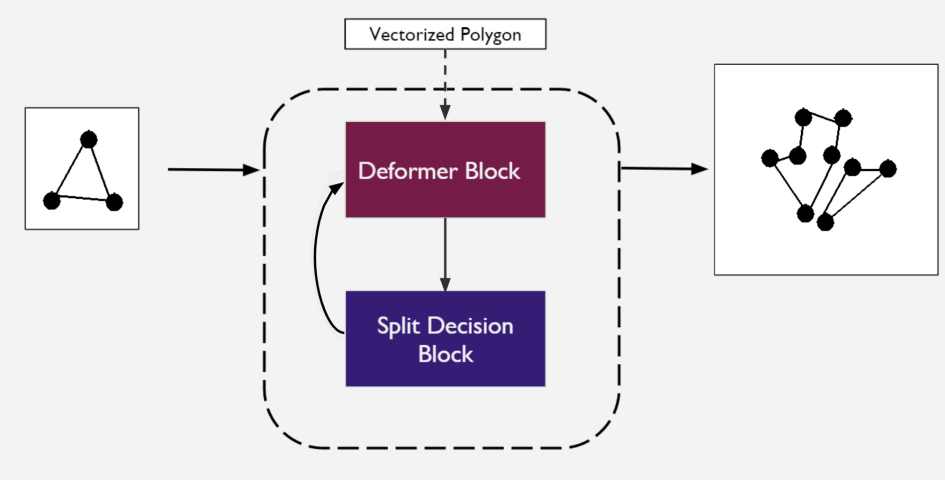


Figure 3. Toy Data: Polygons and Planes

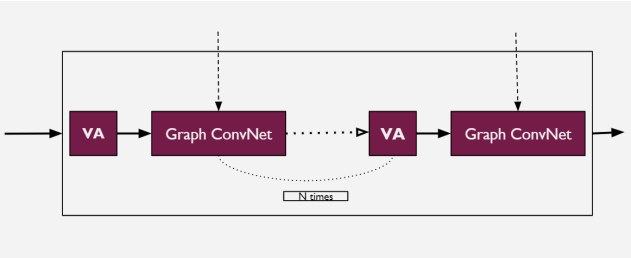


Figure 4. Deformer: Vertex Adder and Graph Convolutions

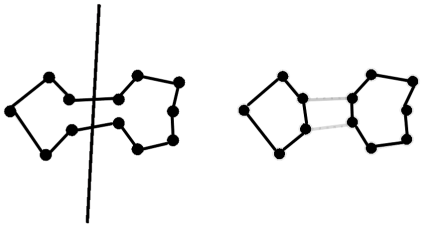

Figure 5. Splitting using line segment

Note that there is no notion of connectivity or anything related to mesh in this loss.

**Normal Loss:** A differentiable version of a normal loss that enforces the normals at the vertices to be consistent with those of the ground truth is,

$$l_c = \sum_p \sum_{q=argmin_q \|p-q\|_2^2} \|\langle p-k, n_q \rangle\|_2^2 \quad \text{s.t} \quad k \in N(p)$$

**Laplacian Regularization:** The above losses, in particular chamfer loss don't contain mesh relevant properties that causes the deformation learning to be unstable. To stabilise this we use a laplacian regularization that enforces that the deformations are smooth, in the sense that the laplacians at the input and output of a deformation block are not too different. Formally,

$$\delta_p = p - \sum_{k \in N(p)} \frac{1}{\|N(p)\|} k$$

$$l_{lap} = \sum_p \|\delta_p^{'} - \delta_p\|_2^2$$

where $\delta_p^{'}$ and $\delta_p$ are the laplacian coordinates of the vertex after and before the deformation block. **Edge Length Regularization:** To penalize long edges and flying vertices we add an edge length regularization,

$$l_{edge} = \sum_p \sum_{k \in N(p)} \|p-k\|_2^2$$

The overall loss is a weighted sum of the four losses, $l_{total} = l_c + \lambda_n l_n + \lambda_{lap} l_{lap} + \lambda_{edge} l_{edge}$ where $\lambda_n = 2$, $\lambda_{lap} = 1$, $\lambda_{edge} = 0.5$ are the fixed hyperparameters we use.

## 4.2. Splitter block

### 4.2.1 Block Architecture

The splitter block has to split a set of vertices into two sets, so that they can fit the ground truth better. One implementation of this block is to use hybrid representation, where the first mesh is transformed into the other representation, and then split is performed. Finally, after split, the output is transformed back to mesh. However, we want to keep mesh structure throughout the network, and therefore we use splitter in the following manner.

We model our splitter to predict a line segment which can be used to divide the polygon. See Fig. 5 for more details. Our splitter is an actor-critic network, which is trained using DDPG algorithm [12]. The input to the actor network

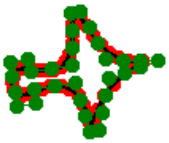

Figure 6. Results: Simple Deformer on Plane data (Green is ground truth; Red is learned representation)

is the averaged output of deformer as well as 1d projections of both the predicted mesh and the ground truth. Based on this, the actor network predicts 4 coordinates which represent a line segment. This line segment determines where to split the polygon. If the line intersects two edges of the same polygon (4 different vertices), such that no prior edge existed between the vertices of opposite edges, we cut the polygon there, and join the corresponding vertices to form 2 polygons.

The critic network is trained using the off-policy buffer. The reward for each action (predicted line segment) is calculated and stored in the buffer. More details about reward calculation are in section 4.2.2. The actor is trained based on the value learned by the critic, with the help of policy gradients.

The last part is decision network which outputs the probability whether a split should be made. During inference, we stop splitting the mesh, once the decision network gives low probability of split.

#### 4.2.2 Splitter Reward

We use a very basic reward to train the critic network. It consists of the following.

1) A reward $r_1$ if the predicted line segment lies in the bounding box encompassing the ground truth mesh and the predicted mesh. This limits the search space of the actor network.

2) A reward $r_2$, conditioned on $r_1$ if the predicted line segment $l$ intersects the predicted mesh $p$ at two points.

3) A reward $r_3$, conditioned on $r_2$, if $l$ doesn't intersect the ground truth mesh and there exist some ground truth vertices on either side of the line segment.

We use the values $r_1 = 5$, $r_2 = 5$, $r_3 = 10$ to train our network.

#### 4.2.3 Splitter Supervised Loss

The decision block is inspired from the encoder-decoder models, where the decoder also learns to predict the end token, to prevent further rollout. Similar to decoder, our decision network also predicts when to stop splitting. We provide decoder with ground truth for all iterations. It receives "true" label for all iterations except the last one, where it recieves "False", during training. At test time, we simply stop the whole process, once the decision block outputs low probability for split.

## 5. Training

Currently we don't train our model architecture in an end-to-end fashion. The main reason for this is that the gradients don't flow between the deformer and splitter blocks. Therefore, we can directly train the $n^{th}$ block after training all the previous $n-1$ blocks, so that the input received by the $n^{th}$ block is good. Hence, we iteratively train the two blocks in succession each for few epochs.

The deformer blocks are trained in a simple supervised manner. The Splitter blocks are trained by DDPG algorithm [12] by training the actor network on continous policy gradients, which are learned by the critic network.

We use the Adam optimizer[9], with learning rate 1e-5 for deformer blocks and 1e-4 for splitter blocks. Since, we use the topology as supervision, we iterate over deformer-splitter loop same number of the time as the topology during training. However, at test time, we stop after the splitter module predicts no more split actions.

## 6. Results

Fig. 6 shows how a model learns to fit single polygon data. We show the pipeline in Fig. 7. Since, we only have 2 polygons here, the network goes through Deformer $\rightarrow$ Splitter $\rightarrow$ Deformer. The Red is prediction while the Green is ground truth. The 1D projections which are sent to RL network can be also be seen in the bottom part of the image. Fig. 8 shows the pipeline for 3 polygons case.The deformer blocks share the weights. Currently the module has separate Splitter blocks, however we plan to merge them soon too.

Fig. 9 shows how the losses and rewards vary over iteration. Here, we trained the model(Deformer $\rightarrow$ Splitter $\rightarrow$ Deformer) for 900 epochs. Each block is trained for 300 epochs each. We notice that the losses which had plateaued after 300 epochs, again start to decrease after the network learns to split the polygon, when second deformer starts training.

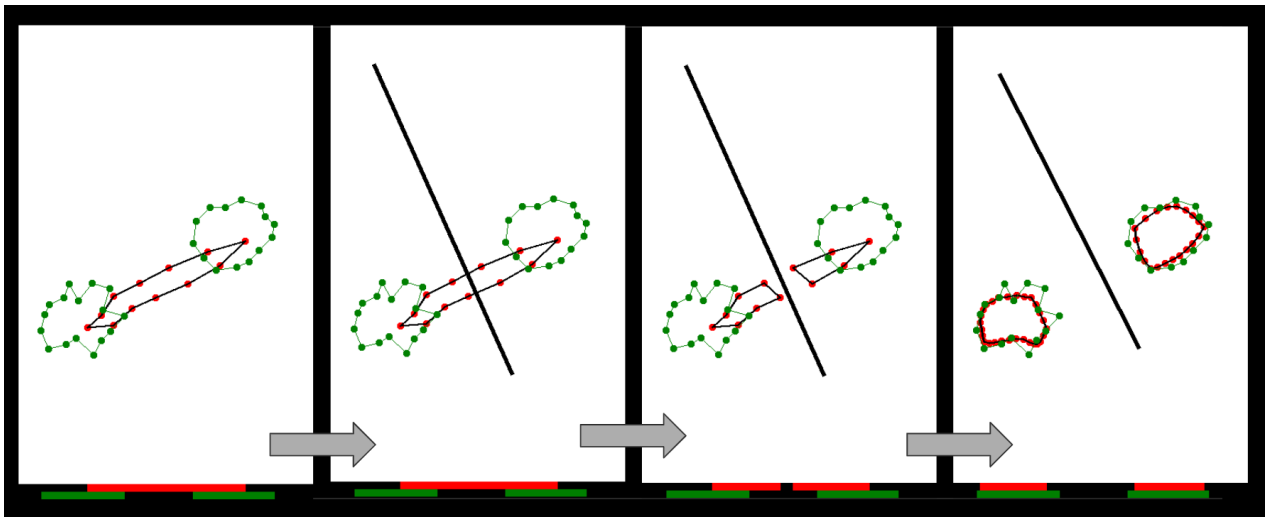

Figure 7. Final Result Pipeline: (Deformer → Splitter (2 frames) → Deformer

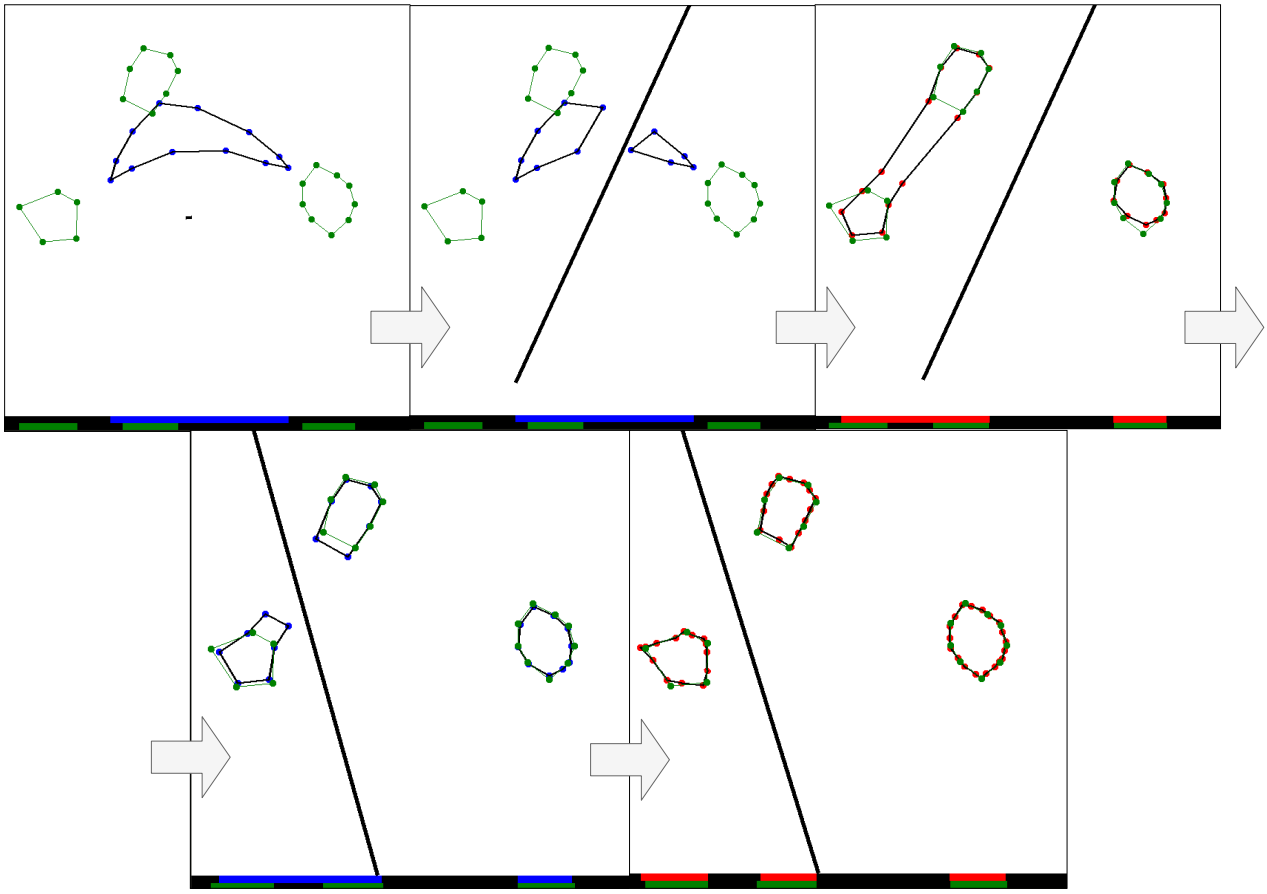

Figure 8. Results: Pipeline for 3 polygon: Deformer → Splitter → Deformer → Splitter → Deformer

## 7. Current Challenges and Future Work

This is a work in progress. We have a lot of aspects to figure out. Currently, our model is able to work perfectly for two polygons. For three polygons, the model is able to correctly learn the two splits, however, the final deformation is very slow. We believe that one reason could be the high number of vertices added by the last deformation. We plan to change the current vertex addition process to a better way,

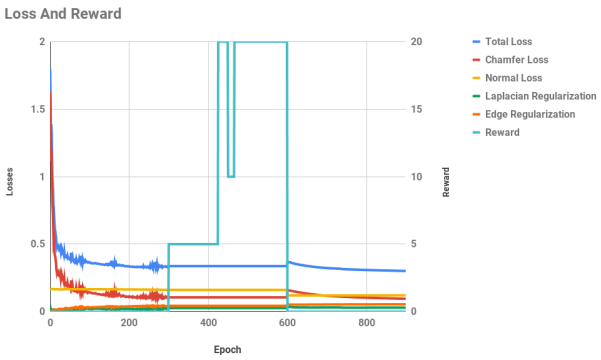


Figure 9. Results: Losses and rewards over iterations

where we only add vertices to the polygon that was split.

Also, as we discussed, our current training method is not straight forward. We haven't tried training our architecture in an end to end fashion. So we would also like to try that soon. Finally, we would like to soon move on from the toy dataset to the 3D dataset which is the end goal.

Another aspect we can perhaps explore is to removing decision block based supervision and use a merge block similar to split block. Also, we can look into Non-RL methods with hybrid representations.

## 8. Acknowledgements

We would like to thank Prof. Qixing Huang for providing feedback and guidance during the course of the work. We also thank Eddy Hudson.

## References

[1] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, July 2017.

[2] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[3] H. Fan, H. Su, and L. J. Guibas. A point set generation network for 3d object reconstruction from a single image. In *CVPR*, volume 2, page 6, 2017.

[4] T. Groueix, M. Fisher, V. G. Kim, B. C. Russell, and M. Aubry. Atlasnet: A papier-m\^ ach\'e approach to learning 3d surface generation. *arXiv preprint arXiv:1802.05384*, 2018.

[5] M. Henaff, J. Bruna, and Y. LeCun. Deep convolutional networks on graph-structured data. *CoRR*, abs/1506.05163, 2015.

[6] A. Kanazawa, M. J. Black, D. W. Jacobs, and J. Malik. End-to-end recovery of human shape and pose. In *Computer Vision and Pattern Regognition (CVPR)*, 2018.

[7] A. Kanazawa, S. Kovalsky, R. Basri, and D. Jacobs. Learning 3d deformation of animals from 2d images. *Computer Graphics Forum*, 35(2):365–374.

[8] A. Kanazawa, S. Tulsiani, A. A. Efros, and J. Malik. Learning category-specific mesh reconstruction from image collections. In *ECCV*, 2018.

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

[10] A. Kurenkov, J. Ji, A. Garg, V. Mehta, J. Gwak, C. Choy, and S. Savarese. Deformnet: Free-form deformation network for 3d shape reconstruction from a single image. *arXiv preprint arXiv:1708.04672*, 2017.

[11] Y. Liao, S. Donné, and A. Geiger. Deep marching cubes : Learning explicit surface representations. 2018.

[12] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[13] O. Litany, T. Remez, E. Rodolà, A. M. Bronstein, and M. M. Bronstein. Deep functional maps: Structured prediction for dense shape correspondence. *CoRR*, abs/1704.08686, 2017.

[14] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black. SMPL: A skinned multi-person linear model. *ACM Trans. Graphics (Proc. SIGGRAPH Asia)*, 34(6):248:1–248:16, Oct. 2015.

[15] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5425–5434, 2017.

[16] J. K. Pontes, C. Kong, S. Sridharan, S. Lucey, A. Eriksson, and C. Fookes. Image2mesh: A learning framework for single image 3d reconstruction. *arXiv preprint arXiv:1711.10669*, 2017.

[17] N. Wang, Y. Zhang, Z. Li, Y. Fu, W. Liu, and Y.-G. Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images. In *ECCV*, 2018.

[18] S. Zuffi, A. Kanazawa, and M. J. Black. Lions and tigers and bears: Capturing non-rigid, 3D, articulated shape from images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, 2018.