Depth-Aware Pixel2Mesh

Rohin Manvi Kabir Jolly Julian Quevedo
Stanford University
{rohinm, kjolly, julianq}@stanford.edu

Abstract

Currently, there are existing methods that are able to construct 3D meshes of objects when given 2D image inputs. However, there is room to improve upon these models by adding in a component of object recognition that we as humans use to perceive the world around us: depth. Our goal is to build upon the Pixel2Mesh work by expanding its capacity from utilizing RGB input images to RGB-D input images. These RGB-D images were created using the MiDaS model, and were fed into a modified Pixel2Mesh to create 3D meshes. While our initial approach involving leveraging differentiable rendering was initially unsuccessful, we found that when using the depth-aware Pixel2Mesh, training results exhibited faster convergence and validation results indicated stronger total loss minimization and comparable voxel loss performance.

1. Introduction

Current advances in the computer vision space have become increasingly accurate in the detection of objects when given 2D inputs. Tools are readily available for accurately and efficiency performing 2D semantic and instance segmentation. However, the world around us lies in 3D, and there is still much work to be done moving forward in developing a computational understanding of 3D shapes and objects.

Currently, there has been prior work done to reconstruct a mesh - which is a collection of vertices, edges, and faces - of a single object or multiple objects located within a 2D image. While these models currently exist, the goal of our work is to enhance such methods to more accurately construct these meshes. We aim to reconstruct better 3D meshes using 2D image inputs by taking depth information into consideration to build up an accurate representation of how the object takes form in the real world.

With the ubiquity and rising prevalence, as well as reliance on automated robotic systems, creating models that are both practical and viable upon implementation has been extremely important. The idea of autonomous driving and robotic assistants seems not too far out, and the component of vision that drives many of its key functionalities plays a crucial role to their success as an innovation.

Mesh R-CNN is one big research advancement in this space, offering the capability of constructing topologically accurate 3-dimensional meshes given a 2-dimensional RGB image [5]. The idea is that it is possible to build on top of Mask R-CNN (a 2D object recognition system) and create voxel representations of 3 dimensional objects within images, and eventually translate these voxel representations into a mesh using a GNN-based approach.

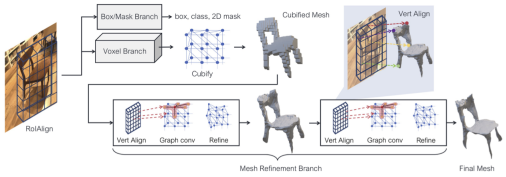

Figure 1. Overview diagram of the Mesh R-CNN process. Figure taken from Gkioxari, Malik, and Johnson [5]

In this project, we seek to expand upon existing models, namely Mesh R-CNN, by working towards enhancing its performance through depth aware inputs. The goal is to expand the original 3 channel RGB input image to a 4 channel RGB-D input image and log potential changes in performance as a result.

2. Related Work

Our work can be seen as an extension to the work done in creating Pixel2Mesh [14]. In this paper, the researchers aimed to reconstruct a 3D shape using a triangular mesh when given a single RBG input image. This work was unique in representing the 3D mesh using a graph CNN, producing the desired geometry by taking an ellipsoid shape and progressively deforming it to best match the desired 3D shape.

The results from this paper both qualitatively and quantitatively outperform prior work, such as representing the output of the neural network as a volume as done by Choi

et al. [2] or as a point cloud as done by Fan et al [4]. To understand why Pixel2Mesh outperformed prior work, we see that the approach adopts the idea of producing progressively more vertices, which generates a smoother output that better captures 3D details and also results in better loss minimization than a volume or point cloud based approach.

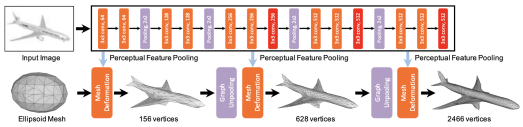

Figure 2. Diagram depicting the Pixel2Mesh approach of progressively generating the desired mesh. Figure taken from Wang et al. [14]

Similar to Mesh R-CNN and Pixel2Mesh, PIFuHD offers high-fidelity 3d reconstruction of clothed humans from a single image. The resolution high enough to recover details in the anatomy such as fingers, facial features and even the details in clothes. This system used Pixelaligned Implicit Function(PIFu) in addition to pixel-aligned prediction module and an occupancy probability field prediction module. [12]

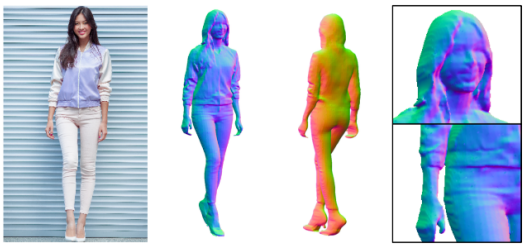

Figure 3. Given a high-resolution single image of a person, PIFuHD recovers highly detailed 3D reconstructions of clothed humans at 1k resolution. Figure taken from Saito et al. [12]

We also build on top of the work done by Ranftl et al. They focused on improving monocular depth estimation, ensuring that there was heightened performance regardless of depth range and scale of the object in consideration [10]. This was done by pretraining encoders, learning with multiple objectives on varied datasets to produce a very high capacity for model generalization. The final model, MiDaS, is established as the state of the art when looking at zero-shot performance in comparison to other work with similar objectives.

More traditional methods of 3D reconstruction have used binomial stereo vision. They imitate the human visual system and generate depth maps as results. The idea of using methods that are more similar to human vision have been explored. This is usually accomplished using additional hardware that augments the data. [16]

Figure 4. Outputs of running MiDaS depth estimation model on a random subset of single-view input images. Figure taken from Ranftl et al. [10]

3D for Free is a crossmodal Transfer Learning method using HD Maps. It uses a large unlabeled dataset of images and LiDAR. The dataset was manually mined using a LiDAR based object detector. The model was constructed to produce 3D cuboids with varying confidence. The results seemed to show that this was a promising approach to the problem. [15]

Du2Net: Learning Depth Estimation from Dual-Cameras and Dual-Pixels combines dual camera stereo and dual-pixel sensor stereo. The authors' work provides substantial improvements over similar previous works. They do this by avoiding the inherent ambiguity caused by certain aperture problems. They also make the stereo baseline orthogonal to the dual-pixel baseline. [17]

3. Methods

3.1. Differentiable Rendering

In order to provide further supervision on our generated meshes, we considered differentiable rendering with PyTorch3D [11]. Once a mesh has been generated by our model, we can use a differentiable rasterizer to render a depth map of it. The error between the rendered depth map and the "ground-truth" D channel of the RGB-D image can then be measured. Importantly, as in MiDaS, this loss must be scale-invariant. This would allow our model to take further advantage of the RGB-D images by creating a mesh that has the same depth characteristics as the input depth map.

In order to meaningfully compare the rendered depth maps with the ones from the input images, we need to render them from the same camera positions. So, we parsed the image metadata from 3D-R2N2 and extracted camera elevation, azimuth, and distance for each training example, which could then be passed in to PyTorch3D's differentiable renderer.

Although we were able to render depth maps for each generated mesh, we discovered more difficulties and were unable to fully implement the scale-invariant depth loss. First, MiDaS was trained on natural images and hallucinates a ground plane beneath the ShapeNet renderings. On

the other hand, the PyTorch3D differentiable renderer simply marks all background points as -1, a sentinel value for when there is no mesh face in a given pixel. Second, MiDaS outputs an *inverse* depth map, where the closes pixels are assigned the highest value. In contrast, PyTorch3D outputs a proper depth map, where closer pixels get smaller values. We hope to investigate solving both of these problems simultaneously in future work.

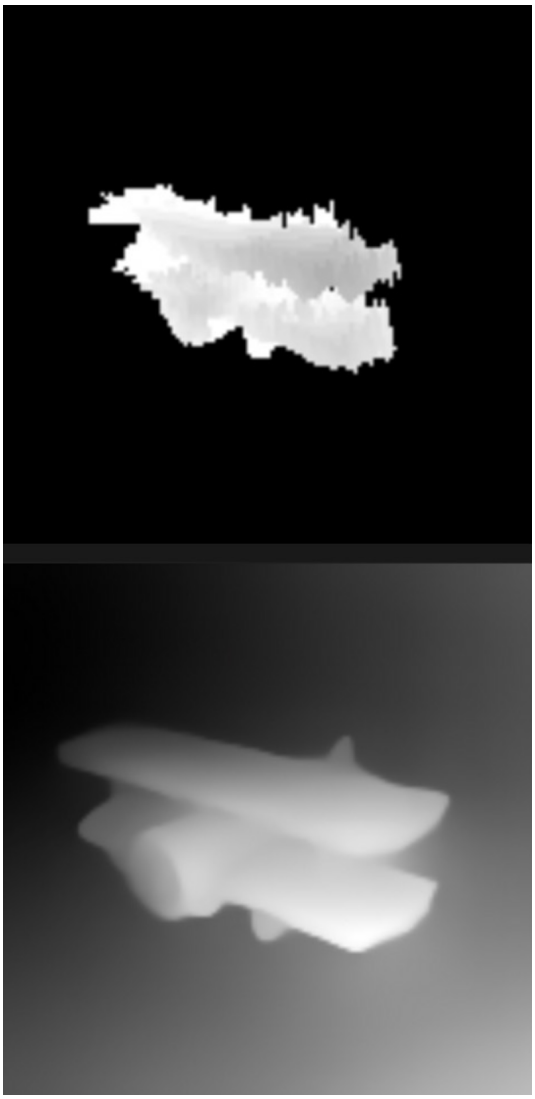

Figure 5. Result of mesh reconstruction (top) compared to inverse depth estimation image (bottom)

3.2. RGB-D Backbone

With regards to the coding libraries and packages used, we employed PyTorch for creating the model architectures [9], NumPy for data manipulation [6], and Matplotlib for generating the figures used for both qualitative and quantitative analysis [7].

Index	Inputs	Operation	Output shape
(1)	Input	Image	$137 \times 137 \times 3$
(2)	(1)	ResNet-50 `conv2_3`	$35 \times 35 \times 256$
(3)	(2)	ResNet-50 `conv3_4`	$18 \times 18 \times 512$
(4)	(3)	ResNet-50 `conv4_6`	$9 \times 9 \times 1024$
(5)	(4)	ResNet-50 `conv5_3`	$5 \times 5 \times 2048$
(6)	(5)	Bilinear interpolation	$24 \times 24 \times 2048$
(7)	(6)	Voxel Branch	$48 \times 48 \times 48$
(8)	(7)	`cubify`	$\|V\| \times 3, \|F\| \times 3$
(9)	(2), (3), (4), (5), (8)	Refinement Stage 1	$\|V\| \times 3, \|F\| \times 3$
(10)	(2), (3), (4), (5), (9)	Refinement Stage 2	$\|V\| \times 3, \|F\| \times 3$
(11)	(2), (3), (4), (5), (10)	Refinement Stage 3	$\|V\| \times 3, \|F\| \times 3$

Figure 6. High-level architecture of the ShapeNet version of Mesh R-CNN

As a baseline, used the version of the Mesh R-CNN model that was trained on the ShapeNet dataset. The backbone feature extractor is ResNet-50 pretrained on ImageNet which has 4 different convolutional layers. The voxel branch of the model receives a bilinearly interpolated feature map from the last ResNet convolutional layer and predicts a voxel grid. The VertAlign operator concatenates features from all of the ResNet-50 convolutional layers before projecting to a single vector. The mesh refinement branch has three parts. Each part has six graph convolution layers organized into three residual blocks.

Index	Inputs	Operation	Output shape
(1)	Input	`conv2_3` features	$35 \times 35 \times 256$
(2)	Input	`conv3_4` features	$18 \times 18 \times 512$
(3)	Input	`conv4_6` features	$9 \times 9 \times 1024$
(4)	Input	`conv5_3` features	$5 \times 5 \times 2048$
(5)	Input	Input vertex features	$\|V\| \times 128$
(6)	Input	Input vertex positions	$\|V\| \times 3$
(7)	(1), (6)	`VertAlign`	$\|V\| \times 256$
(8)	(2), (6)	`VertAlign`	$\|V\| \times 512$
(9)	(3), (6)	`VertAlign`	$\|V\| \times 1024$
(10)	(4), (6)	`VertAlign`	$\|V\| \times 2048$
(11)	(7),(8),(9),(10)	Concatenate	$\|V\| \times 3840$
(12)	(11)	Linear$(3840 \to 128)$	$\|V\| \times 128$
(13)	(5), (6), (12)	Concatenate	$\|V\| \times 259$
(14)	(13)	ResGraphConv$(259 \to 128)$	$\|V\| \times 128$
(15)	(14)	$2\times$ ResGraphConv$(128 \to 128)$	$\|V\| \times 128$
(16)	(15)	GraphConv$(128 \to 3)$	$\|V\| \times 3$
(17)	(16)	Tanh	$\|V\| \times 3$
(18)	(6), (17)	Addition	$\|V\| \times 3$

Figure 7. Mesh R-CNN mesh refinement stage architecture

The voxel loss is the binary cross-entropy between the predicted voxel occupation probabilities and the true voxel occupancies. The mesh loss is defined over a finite set of points. The mesh is represented by a point cloud. Pointclouds P and Q are sampled from the ground truth and the intermediate mesh predictions from the model. Chamfer distance and the normal distance between two pointclouds is defined as follows:

$$\mathcal{L}_{\text{cham}}(P, Q) = |P|^{-1} \sum_{(p,q) \in \Lambda_{P,Q}} \|p - q\|^2 + |Q|^{-1} \sum_{(q,p) \in \Lambda_{Q,P}} \|q - p\|^2 \quad (1)$$

$$\mathcal{L}_{\text{norm}}(P, Q) = -|P|^{-1} \sum_{(p,q)\in\Lambda_{P,Q}} |u_p \cdot u_q| - |Q|^{-1} \sum_{(q,p)\in\Lambda_{Q,P}} |u_q \cdot u_p|. \quad (2)$$

These are used as the losses for the mesh. This model was trained for 25 epochs using Adam with learning rate 10^{-4} and 32 images per batch on 8 Tesla V100 GPUs.

To allow Mesh R-CNN to take RGB-D images as input, we changed the first ResNet layer to learn four-channel filters instead of three-channel filters. To take advantage of pretraining, we will do so by copying over the weights of for the first three channels and only train the fourth from scratch. This allows us to build off the existing weights, iteratively improving them as the model is further trained on the RGB-D input images.

3.3. Mesh Refinement Head

After passing the RGB-D image through the backbone, we use the learned image features to produce a mesh. After predicting a rough voxelization of the object, we convert to a mesh representation using Mesh R-CNN's `cubify`. Then, three "message-passing" graph neural network (GNN) layers are used to refine the mesh vertices to match the target shape. It does so by updating node representations with the following update rule:

$$f_i' = (W_0 f_i + \sum_{j\in N(i)} W_1 f_j,$$

where W_0 and W_1 are learned weight matrices and $N(i)$ indicates the neighborhood of node i.

4. Dataset and Features

Figure 8. ShapeNet renderings from the chair, laptop, bench, and airplane categories

We will train our model on two datasets: ShapeNet Core [1] (along with renderings from R2N2 [3]) and Pix3D [13]. ShapeNet Core consists of over 50,000 3D meshes, which R2N2 provides rendered images of. Pix3D provides another 10,000 3D meshes of IKEA furniture, each paired with a real-life photo. Because of the in-the-wild nature of Pix3D's images, it serves as a more difficult benchmark than ShapeNet, whose images are purely synthetic.

One of our main goals is to assess the impact of depth information in 3D mesh reconstruction. However, the images in these datasets are purely RGB. Thus, as a preprocessing step, we use the Tiefenrausch method [8] to predict each image's depth map. We then stack the depth maps onto the original images, resulting in four-channel RGB-D images.

Figure 9. Natural images of IKEA meshes from Pix3D

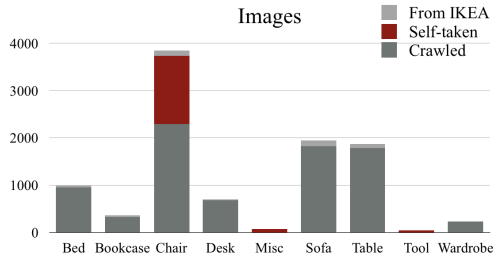

Figure 10. Pix3D image distribution across categories

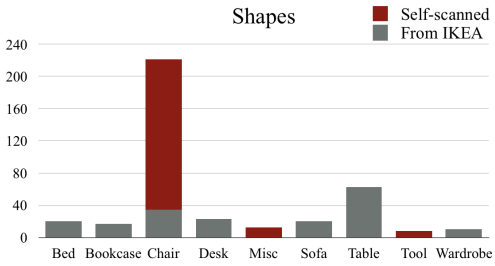

Figure 11. Pix3D mesh distribution across categories

5. Results

Our initial unsuccessful mesh reconstruction involved creating a custom loss that tracked the accuracy of the generated depth estimation predictions. However, the generated images were qualitatively and quantitatively innaccurate, and the model exhibited lower loss the less weight we assigned to the depth loss term. This is because the loss

Figure 12. RGB-D image channels, where the depth channel is predicted by the Tiefenrausch method

stagnation meant that the depth loss was actually detracting from the performance of the model when utilizing the differentiable rendering approach, and the lower weighting was performing better simply because it caused the depth loss to approach zero (hence having less/negligible impact on the ouptut).

As a result, we switched our approach to tracking the performance of the depth-aware Pixel2Mesh model. After preprocessing the dataset (as described earlier), we were able to compare our baseline and experimental models. The baseline model was simply running Pixel2Mesh on regular RGB input images, replicating the work done by Wang et al [14]. Our custom depth-aware Pixel2Mesh model had the ability to take in 4 channels, one of which was generated by the MiDaS model, and was trained accordingly.

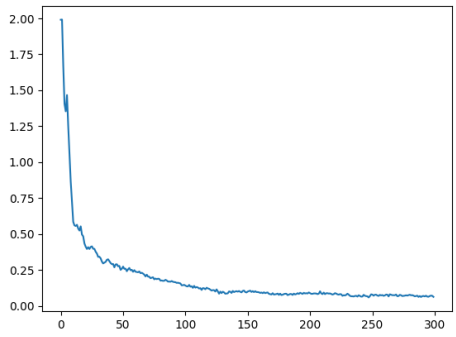

Figure 13. Total loss plotted over 300 batch updates using RGB input images.

As evidenced by the above figures, we see that when adding the depth channel to the input images, the model was able to converge slightly faster when training. The final validation losses after training over 300 epochs was 0.51 for the baseline Pixel2Mesh model and 0.39 for the depth-aware Pixel2Mesh model, showing that not only was there faster convergence, but lower overall loss on the unseen data as well.

We see a similar trend when plotting the voxel loss. Once again, adding the depth channel resulted in a slightly faster convergence, however for the validation loss, the difference

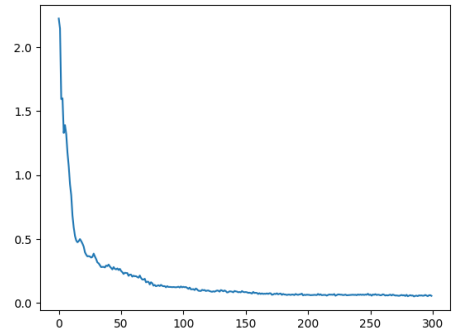

Figure 14. Total loss plotted over 300 batch updates using RGB-D input images.

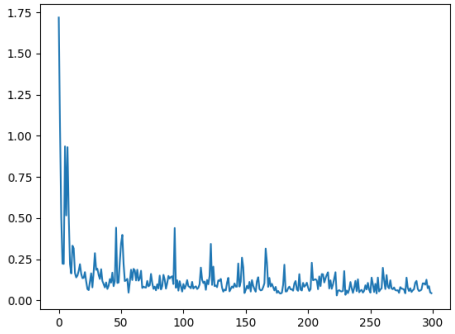

Figure 15. Voxel loss plotted over 300 update steps using RGB input images.

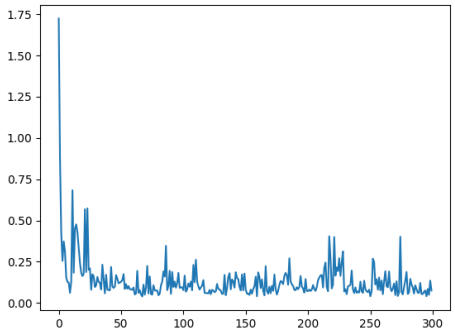

Figure 16. Voxel loss plotted over 300 update steps using RGB-D input images.

between the baseline and depth-aware models was negligible and thus performance was considered to be comparable.

Model	Chamfer Dist. (with RGB)	(with RGB-D)
Pixel2Mesh	4.915	**3.707**
Mesh R-CNN	**4.729**	4.791

Table 1. Total test set Chamfer distance after training for 4 epochs on train set

6. Conclusion and Future Work

Another possible extension to our research is to construct *colored* meshes that accurately represent the textures and materials that appear in the images. Since we represent the meshes as graphs, our idea is to incorporate color information as an additional node feature. Thus, as supervision for our color predictions, we need the meshes in the datasets to have vertex colorings. However, the ShapeNet and Pix3D meshes are colored in a variety of ways, including via face colorings and texture images. To account for this, we could use Blender to "bake" vertex colors into each mesh. To enable Mesh R-CNN to predict the color of each vertex, we plan to increase the dimension of the node features predicted by the mesh refinement stage. Instead of predicting 3-dimensional features, we will predict 6-dimensional features, where the first three correspond to the vertex coordinate and the second three correspond to RGB values. Using this approach, we could reconstruct 3D meshes that better represent how the object appears in the real world.

We also plan to continue the unfinished work on using differentiable rendering and the depth images during training. We hope to overcome the aforementioned roadblocks and hypothesize that due to the additional information fed in during training time, it is likely to outperform the results exhibited by the current depth-aware Pixel2Mesh.

7. Contributions and Acknowledgements

KJ worked on creating the custom loss function that incorporated depth. RM worked on modifying the Mesh R-CNN model to take a depth dimension in addition to RGB. KJ, RM, and JQ wrote the paper. JQ write MiDaS depth estimation scripts to compute RGB-D images and correspondingly re-designed the Mesh R-CNN dataset class, wrote training and evaluation scripts as well as differentiable rendering code.

Our approach was built on top of code from Ranftl et al. and Gkioxari et al.

References

[1] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015. 4

[2] Christopher B. Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. *CoRR*, abs/1604.00449, 2016. 2

[3] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2016. 4

[4] Haoqiang Fan, Hao Su, and Leonidas J. Guibas. A point set generation network for 3d object reconstruction from a single image. *CoRR*, abs/1612.00603, 2016. 2

[5] Georgia Gkioxari, Jitendra Malik, and Justin Johnson. Mesh r-cnn, 2019. 1

[6] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020. 3

[7] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. 3

[8] Johannes Kopf, Kevin Matzen, Ocean Quigley Suhib Alsisan, Francis Ge, Yangming Chong, Josh Patterson, Jan-Michael Frahm, Shu Wu, Matthew Yu, Peizhao Zhang, Zijian He, Peter Vajda, Ayush Saraf, and Michael Cohen. One shot 3d photography. 39(4), 2020. 4

[9] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 3

[10] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer, 2019. 2

[11] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv:2007.08501*, 2020. 2

[12] Shunsuke Saito, Tomas Simon, Jason M. Saragih, and Hanbyul Joo. Pifuhd: Multi-level pixel-aligned implicit function for high-resolution 3d human digitization. *CoRR*, abs/2004.00452, 2020. 2

[13] Xingyuan Sun, Jiajun Wu, Xiuming Zhang, Zhoutong Zhang, Chengkai Zhang, Tianfan Xue, Joshua B Tenenbaum,

and William T Freeman. Pix3d: Dataset and methods for single-image 3d shape modeling. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 4

[14] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images. In *ECCV*, 2018. 1, 2, 5

[15] Benjamin Wilson, Zsolt Kira, and James Hays. 3d for free: Crossmodal transfer learning using hd maps, 2020. 2

[16] Yuyang Wu. Monocular instance level 3d object reconstruction based on mesh r-cnn. In *2020 5th International Conference on Information Science, Computer Technology and Transportation (ISCTT)*, pages 1–6, 2020. 2

[17] Yinda Zhang, Neal Wadhwa, Sergio Orts-Escolano, Christian Häne, Sean Fanello, and Rahul Garg. Du^{2}net: Learning depth estimation from dual-cameras and dual-pixels, 2020. 2