FlowNet3D: Learning Scene Flow in 3D Point Clouds
Supplementary Material

Xingyu Liu*[1] Charles R. Qi*[2] Leonidas J. Guibas[1,2]
[1]Stanford University [2]Facebook AI Research

A. Overview

In this document, we provide more details to the main paper and show extra results on model size, running time and feature visualization.

In Sec. B we describe details in the FlyingThings3D experiments. In Sec. C, we provide more details on the baseline architectures (main paper Sec. 6.1). In Sec. D we describe how we prepared KITTI LiDAR scans for our evaluations (Sec. 6.2). In Sec. E and Sec. F we explain more details about the experiments for the two applications of scene flow (Sec. 6.3). Lastly in Sec. G we report our model size and run time and in Sec. H we provide more visualization results on FlyingThings3D and network learned features.

B. Details on FlyingThings 3D Experiments (Sec. 6.1)

The FlyingThings3D dataset only provides RGB images, depth maps and depth change maps. We constructed the point cloud scene flow dataset by popping up 3D points from depth map. The virtual camera intrinsic matrix is

$$K = \begin{bmatrix} f_x = 1050.0 & 0.0 & c_x = 479.5 \\ 0.0 & f_y = 1050.0 & c_y = 269.5 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$$

where (f_x, f_y) are the focal lengths and (c_x, c_y) is the location of principal point. We didn't use RGB images in point cloud experiments.

The Z values of background are significantly larger than the moving objects in the foreground of FlyingThings3D scenes. In order to prevent depth values from explosion and to focus on more apparent motion of foreground objects, we only use points whose Z is larger than a certain threshold t. We set $t = 35$ in all experiments.

We generate a mask for disappearing/emerging points due to: 1) change of field of view; 2) occlusion. Scene flow loss at the masked points are ignored during training but were used during testing (since we do not have masks at the test time).

C. Details on Baseline Architectures (Sec. 6.1)

FlowNet-C on depth and RGB-D images. This model is adapted from [4]. The original CNN model takes a pair of RGB images as input. To predict scene flow, we send a pair of depth images or RGB-D images into the network. Depth maps are transformed to XYZ coordinate maps. RGB-D images are six-channel maps where the first three channels are RGB images and the rest are XYZ maps. The model has the same architecture as FlowNet-C in [4] except that the input has six channels for RGB-D input.

The RGB values are scaled to $[0, 1]$. We use the same threshold t as point cloud experiments. Also, scene flow loss at positions where Z value is larger than t are ignored during training and testing.

EM-baseline. The model mixes two point clouds at input level. How to represent the input is not obvious though as two point clouds do not align/correspond. A possible solution is to append a one-hot vector (with length two) as an extra feature to each point, with $(1, 0)$ indicating the point is from the first set and $(0, 1)$ for the other set, which is adopted in our EM-baseline.

In Fig. 1, we illustrate our baseline architectures for the EM-baseline. For each *set conv* layer, r means radius for local neighborhood search, mlp means multi-layer perceptron used for point feature embedding, "sample rate" means how much we down-sample the point cloud (for example $1/2$ means we keep half of the original points). The *feature propagation* layer is originally defined in [6], where features from sub-sampled points are propagated to up-sampled points by 3D interpolation (with inverse distance weights). Specifically, for an up-sampled point its feature is interpolated by three k-NN points in the sub-sampled points. After this step, the interpolated features are then concatenated with the local features linked from the outputs of the set conv layers. For each point, its concatenated feature passes through a few fully connected layers, the widths of which are defined by $mlp\{l_1, l_2, ...\}$ in the block.

LM-baseline. The late mixture baseline (LM-baseline) mixes two point clouds at the global feature level, which makes it difficult to recover detailed local relations among the point clouds. In Fig. 2, we illustrate its architecture, which firstly computes global feature from each of the two point clouds, then concatenates the global features and further processes it with a few fully connected layers (mixture happens at global feature level), and finally concatenates the tiled global feature with local point feature from point cloud 1 to predict the scene flow.

DM-baseline. While our FlowNet3D model and the DM-baseline both belong to the deep mixture meta architecture, they share the same point feature learning modules to learn intermediate point features and then fix two points at this intermediate level. However they are different in two ways. First the DM-baseline does not adopt a flow embedding layer to "mix" the two point clouds (with XYZ coordinates and intermediate features). Instead The DM-baseline concatenates all feature distances and XYZ displacements into a long vector and passes it to a fully connected network before more set conv layers. This however results in sub-optimal learning because it is highly affected by the point orders. Specifically, given a point $p_i = (x_i, f_i)$ in the first point cloud's intermediate point cloud (the one to be mixed with the cloud from the second frame), its r radius neighborhood points in the second frame $\{q_j\}_{j=1}^{k}$ with $q_j = (y_j, g_j)$, the DM-baseline subsample points in the second frame so that k is fixed and then creates a long vector $v_i \in \mathbb{R}^{2k}$ by concatenation: $(y_j - x_i, d(f_i, g_j))$ for $j = 1, ..., k$. The function d is a cosine distance function to compute the feature distance of two points. The vector v_i is then processed with a few fully connected layers before feature propagation. Second, compared to FlowNet3D, the baseline just uses 3D interpolation (with skip links) for flow refinement, with interpolation of three nearest neighborhood with inverse distance weights as described in [6].

D. Details on KITTI Data Preparation (Sec. 6.2)

Ground removal. For our first evaluation on the KITTI dataset (Table 4 in the main paper), we evaluate on LiDAR scans with removed grounds, for two reasons. First, this is a more fair comparison with previous works that relied on ground segmentation/removal as a pre-processing step [3, 7]. Second, since our model is not trained on the KITTI dataset (due to the very small size of the dataset), it is hard to make it generalize to predicting motions of ground points because the ground is a large flat piece of geometry with little cue to tell its motion.

To validate we can effectively remove grounds in LiDAR point clouds, we evaluate two ground segmentation algorithms: RANSAC and GroundSegNet. RANSAC fits a tilted plane to point clouds and classify points close to the plane as ground points. GroundSegNet is a PointNet segmentation network trained to classify points (in 3D patches) to ground or non-ground (we annotated ground points in all 150 frames and used 100 frames as train and the rest as test set). Both methods can run in real time: 43ms and 57ms per frame respectively, and achieve very high accuracy: *94.02%* and *97.60%* averaged across test set. Note that for evaluation in the main paper Table 4, we used our annotated ground points for ground removal, to avoid dependency on the specific ground removal algorithm.

Method	RANSAC	GroundSegNet
Accuracy	94.02%	97.60%
Time per frame	43 ms	57 ms

Table 1: Evaluation for ground segmentation on KITTI Lidar scans. Accuracy is averaged across test frames.

Inference on large point clouds. On large KITTI scenes, we split the scene into multiple chunks. Chunk positions are the same for both frames. Each chunk has size of 5m×5m and is aligned with XY axes (considering Z is the up-axis). There are overlaps between chunks. In practice, neighboring chunks are off by 2.5m with a small noise (Gaussian with 0.3 std) in X or Y direction to each other.

We run the final FlowNet3D model on pairs of frame 1 chunk and frame 2 chunk that are at the same location. Points appearing in more than one chunk have their estimated flows averaged to get the final output.

E. Details on the Scan Registration Application (Sec. 6.3.1)

For this experiment we prepared a partial scan dataset by virtually scanning the ModelNet40 [8] CAD models with a rotated camera around the center axis of the object, with the same train/test split as for the classification task. The virtual scan tool is provided by the Point Cloud Library. In partial scans, parts of an object may appear in one scan but missing in the other, which makes registration/warping very challenging.

We finetuned our FlowNet3D model on this dataset, to predict the 3D warping flow from points in one partial scan to their expected positions in the second scan. Then at inference time, we predict the flow for each point in the first scan as its scene flow. Since the point moving distance can be very large in those partial scans, we iteratively regress twice for the scene flow (i.e. predict a flow from point cloud 1 to point cloud 2, and then predict a second residual flow from point cloud 1 + first flow to point cloud 2). Then the

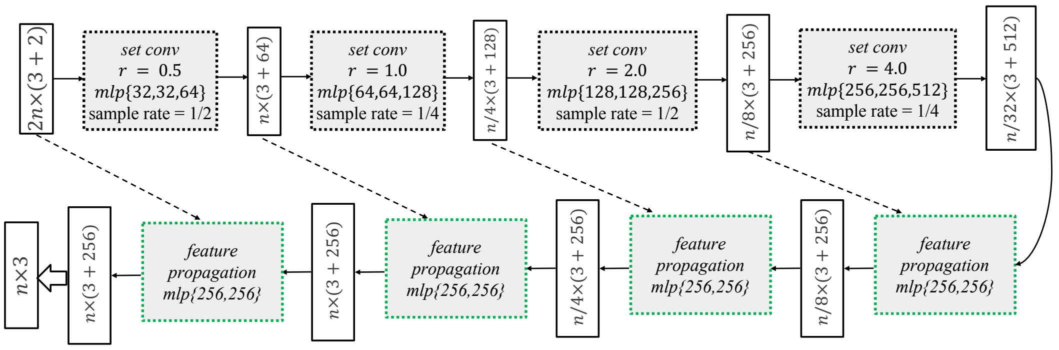

Figure 1: Architecture of the Early Mixture baseline model (EM-baseline).

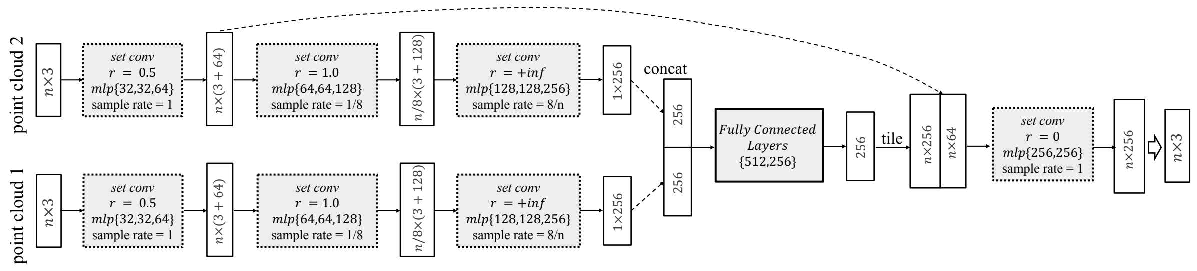

Figure 2: Architecture of the Late Mixture baseline model (LM-baseline).

final scene flow is the 1st flow + the residual flow (visualized in Fig. 6 main paper). To get a rigid motion estimation from the scene flow, we can fit a rigid transformation from the point cloud 1 to the point cloud 2 + scene flow, as they have one-to-one correspondences. Then the rigidly transformed point cloud 1 is the final estimation of our warping (shown in main paper Fig. 6 right while the warping error is reported in main paper Table 6).

F. Details on the Motion Segmentation Application (Sec. 6.3.2)

We first obtained the estimated scene flow with the method discussed in Sec. D. Then the flow is multiplied with a factor λ and is concatenated with coordinates of each point as a 6-dim vector $(x, y, z, \lambda d_x, \lambda d_y, \lambda d_z)$. Next based them we find connected components in the 6-dim space by setting two hyperparameters: a proper minimum cluster size and distance upper bound for forming a cluster.

G. Model Size and Run Time

FlowNet3D has a model size of 15MB, which is much smaller than most deep convolutional neural networks. In Table 2, we show the inference speed of the model on point clouds with different scales. For this evaluation we assume both point clouds from the two frames have the same number of points as specified by #points. We test the run time on a single NVIDIA GTX 1080 GPU with TensorFlow [1].

#Points	1K	1K	2K	2K	4K	4K	8K
Batch size	1	8	1	4	1	2	1
Time (ms)	18.5	43.7	36.6	58.8	101.7	117.7	325.9

Table 2: Run time of FlowNet3D with different input point cloud sizes and batch sizes. For this evaluation we assume the two input point clouds have the same number of points.

H. More Visualizations

Visualizing scene flow results on FlyingThings3D We provide results and visualization of our method on FlyingThings3D test set [5]. The dataset consists of rendered scenes with multiple randomly moving objects sampled from ShapeNet [2]. To clearly visualize the complex scenes, we provide the view of the whole scene from top. We also zoom in and view each object from one or more directions. The directions can be inferred from consistent XYZ coordinates shown in both the images and point cloud scene. We show points from frame 1, frame 2 and estimated flowed points in different colors. Note that local regions are zoomed in and rotated for clear viewing. To help find correspondence between images and point clouds, we used distinct colors for zoom-in boxes of corresponding objects. Ideal prediction would roughly align blue and green points. The results are illustrated in Figure 3-5.

Our method can handle challenging cases well. For ex-

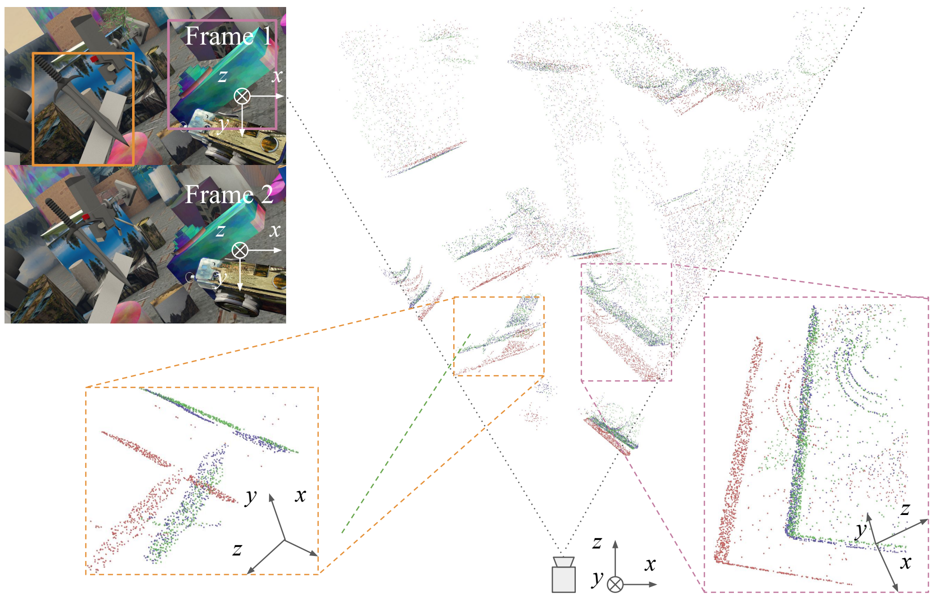

Figure 3: Scene flow results for TEST-A-0061-right-0013 of FlyingThings3D.

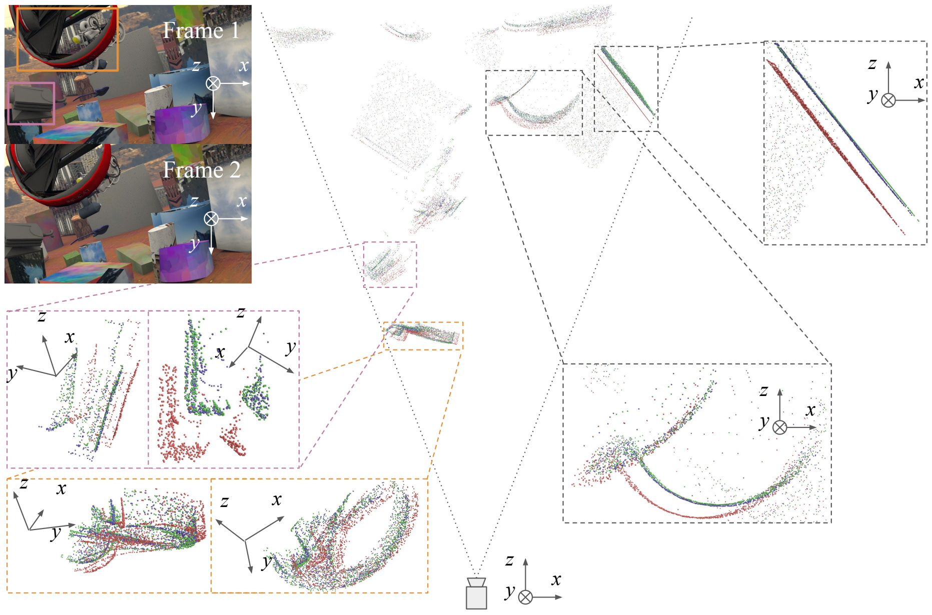

Figure 4: Scene flow results for TEST-A-0006-right-0011 of FlyingThings3D.

ample, in the zoom-in box of Figure 3, the gray box is occluded by the sword in both frames and our network can still estimate the motion of both the sword and visible part of the gray box well. There are also failure cases, mainly due to the change of visibility across frames. For example, in the orange zoom-in box of Figure 5, the majority of the wheel is visible in the first frame but not visible in the second frame. Thus our network is confused and the estimation of the motion for the non-visible part is not accurate.

Network visualization Fig. 6 visualizes the local point features our network has learned, by showing a heatmap of correlations between a chosen point in frame 1 and all points in frame 2. We can clearly see that the network has learned geometric similarity and is robust to partiality of the scan.

Fig. 7 shows what has been learned in a flow embedding layer. Looking at one neuron in the flow embedding layer, we are curious to know how point feature similar-

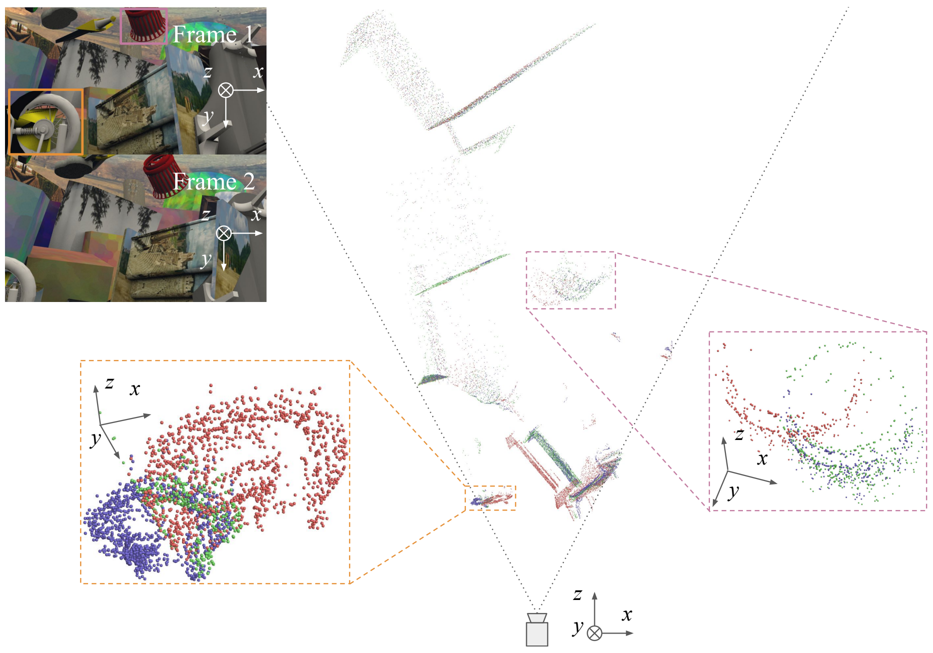

Figure 5: Scene flow results for TEST-B-0011-left-0011 of FlyingThings3D.

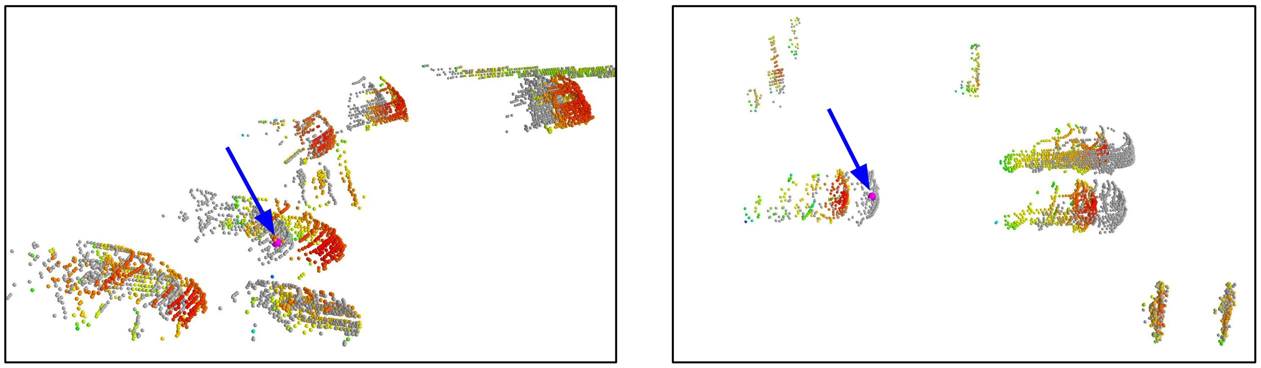

Figure 6: Visualization of local point feature similarity. Given a point P (pointed by the blue arrow) in frame 1 (gray), we compute a heat map indicating how points in frame 2 are similar to P in feature space. More red is more similar.

ity and point displacement affect its activation value. To simplify the study, we use a model trained with cosine distance function instead of network learned distance (through directly inputing two point feature vectors). We iterate distance values and displacement vector, and show in Fig. 7 that as similarity grows from -1 to 1, the activation becomes significantly larger. We can also see that this dimension is probably responsible for a flow along the positive Z direction.

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, 2016. 3

[2] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012, 2015. 3

[3] A. Dewan, T. Caselitz, G. D. Tipaldi, and W. Burgard. Rigid scene flow for 3d lidar scans. In *IROS*, 2016. 2

[4] A. Dosovitskiy, P. Fischery, E. Ilg, C. Hazirbas, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. In *ICCV*, 2015. 1

[5] N. Mayer, E. Ilg, P. Hausser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *CVPR*, 2016. 3

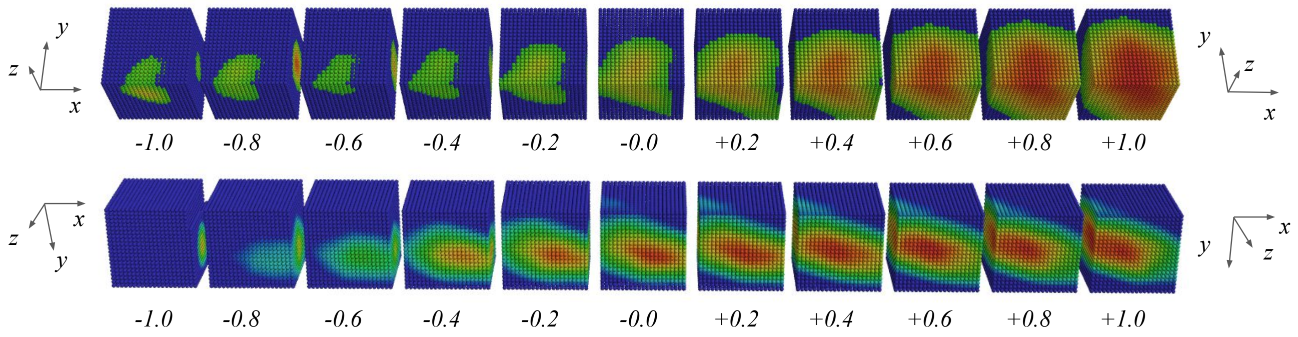

Figure 7: Visualization of flow embedding layer. Given a certain similarity score (defined by one minus cosine distance, at the bottom of each cube), the visualization shows which (x, y, z) displacement vectors in a $[-5, 5] \times [-5, 5] \times [-5, 5]$ cube activate one output neuron of the flow embedding layer.

[6] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*, 2017. 1, 2

[7] A. K. Ushani, R. W. Wolcott, J. M. Walls, and R. M. Eustice. A learning approach for real-time temporal scene flow estimation from lidar data. In *ICRA*, 2017. 2

[8] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *CVPR*, pages 1912–1920, 2015. 2