CPFN: Cascaded Primitive Fitting Networks for High-Resolution Point Clouds

Eric-Tuan Lê[1]* Minhyuk Sung[2]* Duygu Ceylan[3] Radomir Mech[3] Tamy Boubekeur[3] Niloy J. Mitra[1,3]
[1]University College London [2]KAIST [3]Adobe Research

Abstract

Representing human-made objects as a collection of base primitives has a long history in computer vision and reverse engineering. In the case of high-resolution point cloud scans, the challenge is to be able to detect both large primitives as well as those explaining the detailed parts. While the classical RANSAC approach requires case-specific parameter tuning, state-of-the-art networks are limited by memory consumption of their backbone modules such as PointNet++ [27], and hence fail to detect the fine-scale primitives. We present Cascaded Primitive Fitting Networks (CPFN) that relies on an adaptive patch sampling network to assemble detection results of global and local primitive detection networks. As a key enabler, we present a merging formulation that dynamically aggregates the primitives across global and local scales. Our evaluation demonstrates that CPFN improves the state-of-the-art SPFN performance by $13 - 14\%$ *on high-resolution point cloud datasets and specifically improves the detection of fine-scale primitives by* $20 - 22\%$*. Our code is available at: https://github.com/erictuanle/CPFN*

1. Introduction

Representing 3D shapes with a compact set of atomic primitives is a well-established idea that has been evolved over the decades [2, 23]. While the idea has been mainly exploited for machine perception in a way to parse objects, most human-made objects are indeed modeled as a composition of geometric primitives. In CAD, modeling techniques such as Constructive Solid Geometry (CSG) [18] or building a binary tree of simple primitives, have been conventional practices. Hence, for scanned data of human-made objects, converting them into a form to reflect how they were modeled is important not only for the perception but also for enabling editing capabilities in downstream applications. The problem of precisely fitting primitives to the input scan is, however, more challenging than coarsely parsing and abstracting the shape.

For such a model fitting problem, RANSAC [7] is the de facto standard technique in computer vision. The algorithm of Schnabel *et al*. [31] or Li *et al*. [21] which iteratively runs RANSAC to find fitting primitives has been implemented in popular geometry processing libraries such as CGAL [24] and applied to solve the primitive fitting problem with many real scan data. However, such an unsupervised approach often suffers from the combinatorial complexity nature of the problem. From an optimization perspective, different primitive configurations can potentially result in similarly small fitting errors, although the iterative heuristic algorithm cannot take into account all the possible configurations. Furthermore, an undesired set of primitives can even result in a smaller fitting error due to noise in the input. While the RANSAC-based approach deals with the noise to some extent with some threshold parameters, the input-specific parameter tuning requires substantial manual effort.

To tackle the challenge, Li *et al*. [20] recently proposed a *supervised* framework called SPFN that learns the best configuration of primitives for each 3D scan from a large collection of CAD data. Instead of directly regressing the primitive parameters, their network employs PointNet++ [27] as an encoder of the input point cloud and predicts per-point information, including association from a point to a primitive, primitive type, and surface normal. A subsequent differentiable module computes the best primitive parameters minimizing the fitting error through an analytic formulation.

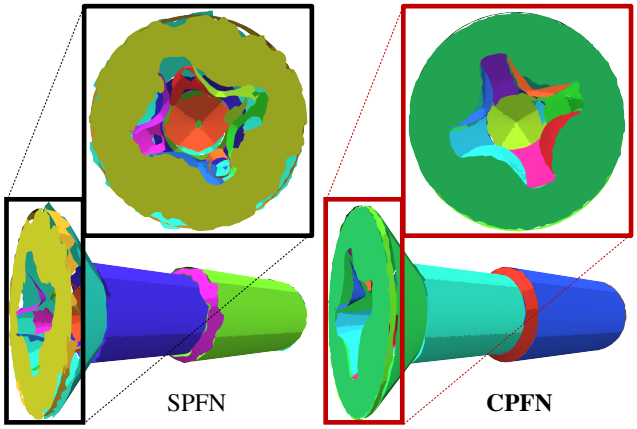

Figure 1. A side-by-side comparison between SPFN [20] and our CPFN. Our cascaded networks are designed to accurately detect and fit small primitives in a high-resolution point cloud.

*This work was partly done when E. Lê interned and M. Sung worked at Adobe Research.

While SPFN [20] demonstrated successful results, the challenge remains in handling *high-resolution* data. Even affordable 3D scanners are now capable of capturing local geometric details with high-resolution (e.g., point sets with 100k+ points). However, efficiently processing the high-resolution 3D data in neural networks raises a memory limit issue with consumer GPUs. Even with a simple point cloud processing architecture such as PointNet [26], the order of 10k points is the limit in training, whereas scans may include points in the order of 100k to 1M. Downsampling the input point cloud results in information loss for fine-scale details and thus fails to fit small primitives (see Figure 1 for example results with missed features by SPFN on typical high-resolution scans).

In this work, we propose a novel framework named Cascaded Primitive Fitting Networks (CPFN), which is particularly developed to capture local details in scans and fit small primitives. Our framework cascades two fitting networks: one for processing the *entire* input point cloud, and the other for processing *local* patches of the input. Both of them are SPFNs [20] but trained separately with global/local input data. Our design breaks the problem into three steps: first, adaptively sampling patches in regions of small details; suitably regressing primitives in (detected) regions of fine details; and merging the global and local primitives to get a multi-scale output.

The framework includes a *patch selection network* trained to detect regions with small primitives so that local patches fed to the *local fitting network* can be sampled in those regions at test time. Our key idea is in the merging algorithm that aggregates the per-point outputs of both networks and produces the final fitted primitives. The merging process is formulated as a binary program, although we empirically found that a Hungarian algorithm [17] can obtain a near-optimum solution in most cases. In experiments, we demonstrate that our cascaded networks outperform a single SPFN trained with downsampled point clouds in fitting primitives in all scales with a performance boost of $13-14\%$. The improvement reaches $20-22\%$ for smaller primitives. We also show that the fitting performance of the local fitting network can be improved when it takes global contextual information of the entire input point cloud from the global fitting network.

In summary, our key contributions are as follows.

- We propose CPFN, a primitive fitting framework leveraging two cascaded networks to adaptively detect both small and large primitives.
- Our merging algorithm ensembles the per-point information predicted by the two networks efficiently and produces the final fitted primitives.
- Our experiments demonstrate that the performance of the local fitting network benefits from feeding contextual information learned by the global fitting network.

2. Related Work

We review previous work leveraging neural networks in primitive fitting as well as recent work on processing high-resolution point clouds using neural networks. We refer the reader to the recent survey [15] for a discussion of classical approaches, particularly RANSAC-based methods.

Neural Geometric Primitive Fitting. Neural-network-based approaches have been widely investigated in decomposing 3D shapes into various types of primitives. To our knowledge, Tulsiani *et al*. [37] and Zou *et al*. [39] were the first proposing neural decomposition. They presented networks fitting cuboids to the input 3D shape represented as either voxels or a depth map. Subsequent works have extended this idea. For instance, Sun *et al*. [35] created an architecture predicting a hierarchical structure of cuboids; Smirnov *et al*. [34] suggested a new loss function based on distance fields and fitted rounded cuboids; while Lin *et al*. [22] introduced a reinforcement-learning-based approach fitting connected trapezoid boxes sequentially so that the final output becomes a scaffold mesh. Follow up work has also focused on fitting different types of primitives. As a generalization of cuboids, Paschalidou [25] used superquadrics as geometric primitives; Gadelha *et al*. [8] explored the use of sphere meshes in the context of learning a generative model of man-made shapes; while Chen *et al*. [5] and Deng *et al*. [6] concurrently proposed to represent input shapes as a convex set of planes that recursively partition the space. Genova *et al*. [10, 9] used implicit representations, namely Gaussians, to explore locality. The primitive types used in these works, however, have limited expressibility, and hence these works mostly focused on abstracting the input shapes with a coarse fitting.

Some notable efforts considered multiple primitive types in the fitting. Sharma *et al*. [32] and Kania *et al*. [16] introduced networks predicting a CSG structure from a raw geometry with various types of primitives. However, the precision of fitting was limited since the loss was defined with occupancies in a low-resolution voxel grid. Li *et al*. [20] proposed SPFN, a framework for more precisely fitting multiple primitives including plane, sphere, cylinder, and cone. It was further extended by Sharma *et al*. [33] to fit B-spline patches as well. While showing impressive results, both approaches employ PointNet++ [27], an off-the-shelf architecture, for encoding the input point cloud and thus are limited by the point cloud size (e.g., order of 8k points). Our method extends SPFN in a novel cascaded framework that can fit primitives at various scales to a high-resolution point cloud (e.g., order of 128k points).

Neural Networks for High-Resolution Point Clouds. Recent work has focused on processing high-resolution 3D data as input in neural networks, particularly to handle

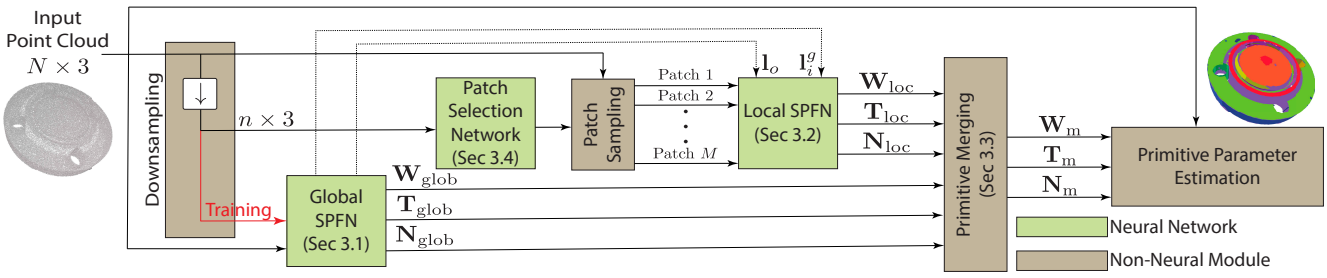

Figure 2. A diagram of CPFN. CPFN includes two SPFNs [20]: one for the entire object and the other for local patches. Contextual information is fed from the global SPFN (Section 3.1) to the local SPFN (Section 3.2). Patch Selection Network (Section 3.4) takes a downsampled point cloud as input and determines where the local patches should be sampled at test time. The per-point predictions from both SPFNs are integrated in the merging step (Section 3.3).

large-scale indoor/outdoor scans and detect and/or segment objects. In terms of voxel representations, OctNet [29] and SparseConvNet [11] are examples introducing efficient architectures to avoid computation in empty space. Particularly, SparseConvNet has shown the best performance in 3D indoor scene segmentation, as demonstrated by Han *et al*. [12]. The others introduced networks that internally voxelize input point clouds to utilize 3D convolutions [28, 4]. Although these architectures perform well in scene segmentation, they are not suitable for fitting problems since the voxelization causes significant discretization errors. As an alternative to voxelization, Tatarchenko *et al*. [36] proposed to exploit 2D convolutions by projecting points in a local region to a tangent plane. However, this architecture is sensitive to errors in surface normal estimation. Other previous work on neural point cloud processing proposed to either cluster points and generate object candidates for instance segmentation [19, 3, 14] or concatenate hierarchical downsampling and upsampling modules as an encoder-decoder architecture for semantic segmentation [13, 38]. Neither of these approaches are directly applicable to our problem since we need to jointly solve for both the semantic and instance segmentations. Our method operates directly at the point level and ensembles fitting results obtained from both the coarse global point cloud as well as high-resolution local patches in a novel merging step to solve the joint segmentation problem.

3. Method

In this section we describe our cascaded primitive fitting networks. Given an input point cloud with N points (where N is 128k in our experiments), our networks operate at two levels, namely global and local. We utilize a global primitive fitting network, SPFN [20], trained on downsampled versions of the input point clouds (i.e., trained on point clouds of size n where $n \ll N$) due to high memory footprints of point cloud processing backbone modules such as PointNet++. While the trained network can be tested on the original point cloud at inference time, it is likely to miss the small primitives representing fine scale details since they are lost in the downsampling process during training. Hence, we train an additional version of SPFN that operates on local patches of the high resolution point clouds (Section 3.2). Given the local predictions for patches and the global predictions for the rest of the point cloud, the core of our method is a novel merging step (Section 3.3) that consolidates all the predictions. In order to ensure that the capacity of the local network is utilized to learn the prediction of small primitives, at training stage we utilize a smart strategy to select the training patches from regions of the point cloud that contain such primitives. At inference time, we utilize a patch selection network (Section 3.4) that predicts the regions that are likely to contain small primitives and thus should be processed with the local network. In the following, we first provide a short summary of the SPFN architecture and discuss the different stages of our method in detail as presented in Figure 2.

3.1. Supervised Primitive Fitting Network [20]

SPFN [20], the Supervised Primitive Fitting Network, is an end-to-end network trained to detect the set of primitives (specifically, planes, spheres, cylinders, and cones) of an input 3D point cloud. SPFN first predicts three per-point properties; a segment label $\mathbf{W}$, normal $\mathbf{N}$, and primitive type $\mathbf{T}$. Given such predictions, the actual primitive parameters are estimated in a differentiable manner. Ground truth per-point primitive associations and primitive fitting errors are used as strong supervision. Specifically, the loss is composed of multiple terms: (i) segmentation loss $\mathcal{L}_{\text{seg}}$, (ii) normal loss $\mathcal{L}_{\text{norm}}$, (iii) primitive type loss $\mathcal{L}_{\text{type}}$, (iv) residual loss $\mathcal{L}_{\text{res}}$, i.e., fitting loss, and (v) axis loss $\mathcal{L}_{\text{axis}}$ that considers the normal of a plane or the axis of a cylinder or a cone:

$$\mathcal{L} = \mathcal{L}_{\text{seg}} + \mathcal{L}_{\text{norm}} + \mathcal{L}_{\text{type}} + \mathcal{L}_{\text{res}} + \mathcal{L}_{\text{axis}}. \quad (1)$$

When computing the loss, the predicted primitives are first mapped to the ground truth primitives using the Hungarian matching algorithm to find the pairs of primitives that maximize the intersection over union across the matched primitives. Per-point and per-primitive losses are computed

based on this correspondence. We refer the reader to the original paper [20] for details.

Our pipeline leverages the SPFN framework as it is in the global branch. In addition, we train a local version of SPFN that operates on local patches sampled from the high resolution point cloud. We provide additional contextual information to the local SPFN to boost the performance of the local predictions as we will discuss in Section 3.2.

3.2. Local SPFN

One of the key components of our pipeline is the local SPFN module, which aims to predict small primitives in fine-scale regions of the input point clouds. Given a patch sampled on the input point cloud, local SPFN predicts the same per-point features, i.e. the segment label $\mathbf{W}_p$, normal $\mathbf{N}_p$, and primitive type $\mathbf{T}_p$ as the global SPFN. While we keep the architecture of the original SPFN fixed, we provide additional global contextual information as input. Specifically, given the point cloud representing an object o, we first extract a latent vector, $\mathbf{l}_o$, for the entire point cloud using the global SPFN. Similarly, for each patch i, we extract patch features, $\mathbf{l}_i^g$, that we obtain for the seed point of the patch using the global SPFN. We concatenate both the object and patch features obtained from the global SPFN to the latent code of the patch $\mathbf{l}_i$ generated by the local SPFN encoder: $\mathbf{l}_i^{'} = [\mathbf{l}_i, \quad \mathbf{l}_o, \quad \mathbf{l}_i^g]$. We utilize $\mathbf{l}_i^{'}$ as input to the local SPFN decoder. Our experiments show that providing additional contextual information boosts the performance of the local SPFN as discussed in Section 4.4.

3.3. Segment Merging

Given a local patch, the local SPFN predicts a segmentation label for each point in the patch where each segment corresponds to a primitive. Our next step is to merge such local per-patch predictions with the predictions of the global SPFN to compute the final segmentation, i.e., primitive decomposition of the high resolution point cloud.

When each local SPFN predicts a maximum of K_{loc} segments, we represent the per-point segment label predictions of the i-th patch with a probability matrix, $\mathbf{W}_{\text{loc}}^i \in [0,1]^{N \times K_{\text{loc}}}$, which is defined over the *entire* N number of input points:

$$\mathbf{W}_{\text{loc}}^i = \begin{pmatrix} p_{1,1}^i & p_{1,2}^i & \cdots & p_{1,K_{\text{loc}}}^i \\ p_{2,1}^i & p_{2,2}^i & \cdots & p_{2,K_{\text{loc}}}^i \\ \vdots & \vdots & & \vdots \\ p_{N,1}^i & p_{N,2}^i & \cdots & p_{N,K_{\text{loc}}}^i \end{pmatrix} \tag{2}$$

with $p_{a,b}^i = \mathbb{P}(P_a \in \mathcal{S}_b^i), \quad a \in \{1,..,N\}, \quad b \in \{1,..,K_{\text{loc}}\}$, denoting the probability of point a belonging to segment b. Note that a point that does not belong to the patch has zero probability.

We represent the prediction results of the global SPFN which predicts a total of K_{glob} segments in a similar matrix representation, $\mathbf{W}_{\text{glob}}$. We then stack each of the segmentation matrices from M patches processed by the local SPFN and the global segmentation matrix:

$$\mathbf{W} = \begin{bmatrix} \mathbf{W}_{\text{loc}}^1 & \mathbf{W}_{\text{loc}}^2 & \cdots & \mathbf{W}_{\text{loc}}^M & \mathbf{W}_{\text{glob}} \end{bmatrix}. \tag{3}$$

The goal of the merging step is to compute the one-to-many relationship between the final set of primitives in the input and the individually predicted segmentations. Assuming there are a total of K_{m} primitives in the final decomposition, this relationship can be written as a binary matrix $\mathbf{C} \in \{0,1\}^{K_{\text{m}} \times (M \cdot K_{\text{loc}} + K_{\text{glob}})}$.

The optimum assignments between the individually predicted segmentations and the final set of primitives need to satisfy certain constraints. Specifically, each segment should be mapped to exactly one final primitive: $\mathbf{C}^T \mathbb{1}_{K_{\text{m}}} = \mathbb{1}_{M \cdot K_{\text{loc}} + K_{\text{glob}}}$. We further enforce that two segments predicted from the same local patch (or global SPFN) should not be merged under the assumption that the network will avoid over-segmentation: $\mathbf{CA} \leq \mathbb{1}_{K_{\text{m}} \times (M+1)}$, where $\mathbf{A} \in [0,1]^{(M \cdot K_{\text{loc}} + K_{\text{glob}}) \times (M+1)}$ is a matrix indicating the association between segments and patches (or global SPFN). Finally, we prefer to assign two segments $\mathcal{S}_k^i$ and $\mathcal{S}_l^j$ predicted from patches i and j to the same final primitive, i.e., merge them, if they have a significant amount of overlap measured as the number of points that belong to both segments. Since $\mathbf{I} = \mathbf{W}^T\mathbf{W}$ represents the intersections between segments as sums of joint probabilities for each point, we find $\mathbf{C}$ by maximizing $\sum_{i,j} \mathbf{I}_{ij} \left(\mathbf{C}^T\mathbf{C}\right)_{ij} = \text{tr}\left(\mathbf{I}\mathbf{C}^T\mathbf{C}\right)$, meaning that we maximize the intersections between the segments assigned to the same final primitive.

Finally, the final assignment task is formulated as the following binary quadratic programming problem:

$$\begin{aligned} \mathbf{C}^* = & \underset{\mathbf{C}}{\text{argmax}} \ \text{tr}\left(\mathbf{I}\mathbf{C}^T\mathbf{C}\right) \\ \textbf{s.t.} \quad & \mathbf{C}^T\mathbb{1}_{K_{\text{m}}} = \mathbb{1}_{M \cdot K_{\text{loc}} + K_{\text{glob}}} \\ & \mathbf{CA} \leq \mathbb{1}_{K_{\text{m}} \times (M+1)}. \end{aligned} \tag{4}$$

Finding the optimum $\mathbf{C}^T\mathbf{C}$ instead (a matrix indicating whether two segments are merged or not) also becomes a binary semidefinite programming problem. It typically takes a huge amount of time to solve either the binary quadratic or semidefinite programming. Empirically, we found that a simple heuristic based on Hungarian algorithm [17] can provide sufficiently good results while being significantly faster. In $\mathbf{I}$, we find the element that corresponds to the pair of segments with maximum intersection and mark the corresponding indices in $\mathbf{C}^T\mathbf{C}$ as one, i.e., the corresponding pair of segments is merged. We then zero out all other elements in $\mathbf{I}$ that violate the constraints, thus removing any conflicting segment pairs. These two steps are iterated until no segments can be further merged.

Once we recover $\mathbf{C}^*$, we obtain the final association between the individual points in the input point cloud and the final primitives by computing $\mathbf{W}|\mathbf{C}^{*T}|^\wedge$, where $\mathbf{X}^\wedge$ is a column-wise $l1$-normalization of $\mathbf{X}$. The output matrix can be considered as including association *scores* from each point to the final primitives, and thus the primitive with the highest score can be picked for each point. The per-point primitive type is also decided by summing probabilities across the patches including the point and choosing the type with the highest number. Also, in order to optimize for the corresponding primitive parameters, we need the per-point normals which we estimate as the average of the individual patch-based estimations (we recall that a point can potentially belong to multiple patches resulting in multiple type and normal estimations).

3.4. Patch Selection Network

As discussed previously, our method uses a global SPFN module to detect big primitives while relying on the local SPFN to capture small primitives that are likely to be missed in downsampled versions of the input point cloud. In order to ensure the capacity of the local SPFN focuses on small primitive regions, we propose a patch selection strategy both for training and test time. Since at training time we have access to ground truth primitives, we simply sample local patches that belong to small primitives as visualised by the heatmaps in Figure 3. We consider a primitive as small if it has less than $\eta \cdot N$ points with $0 < \eta < 1$ (we experiment with η values in the range $[1\%, 5\%]$). We randomly sample query points on any such small primitive and generate patches of n points using the n-nearest neighbour search for each query point.

At test time, areas that are likely to contain small primitives are unknown. One naive strategy is to randomly sample local patches to be processed by the local SPFN (see Section 4). Instead, we design a patch selection network that predicts a heatmap on the input point cloud to predict such areas. The network is trained using the binary cross-entropy loss:

$$\mathcal{L}_{\text{cross}} = -\sum_{i=1}^{n} (y_i \log(p_i) + (1 - y_i) \log(1 - p_i)), \quad (5)$$

where y_i is a binary value indicating if point i belongs to a small primitive or not and p_i the estimated probability. Similar to global SPFN, we train the patch selection network on downsampled point clouds. However, we still test the network on low-resolution point clouds which is the resolution at which query points are sampled.

At test time, we binarize the predicted heatmap and generate a pool of points that are likely to belong to small primitives. We randomly sample query points from this pool and generate a local patch. We repeat this process until each point in the pool is covered by at least one patch. Note that the local patches can potentially have overlaps.

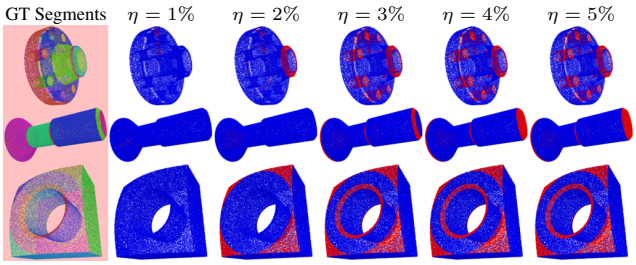

Figure 3. GT Heatmap. Increasing the threshold η enlarges the set of small primitives to sample patches from as shown in red.

4. Experiments

4.1. Dataset

We evaluate our proposed method on CAD models from American National Standards Institute (ANSI) [1] mechanical components provided by TraceParts [30]. SPFN [20] provides the pre-processed point clouds at two resolutions of 8k and 128k points. In our experiments, we use these high-resolution point clouds already provided if not stated otherwise, i.e., $N = 128$k.

The dataset consists of 504 categories, and we use the same training/test splits introduced in [20], providing $13\,831/3\,366$ models for each, respectively. Similar to the low-resolution version of the data, the high-resolution point clouds are preprocessed in a way to merge adjacent primitives sharing the same parameters and discard extremely tiny primitives, which have an area less than 0.5% of the entire object. (Note that the low-resolution version discarded primitives with less than 2% of the area, and thus losing more small primitives than the high-resolution version.) The point clouds are normalized to the unit sphere and also randomly perturbed with uniform noise $[-5\text{e-}3, 5\text{e-}3]$ along the ground truth normal direction. The maximum number of primitives in a given object is 28. Thus, we set $K_{\text{glob}} = 28$.

The input to our network is the noisy point cloud with the random noise, denoted by $\mathbf{P}$. The dataset also provides 512 points sampled on each of the ground-truth primitive surfaces (denoted by $\{\mathbf{S}_k\}$), which we use to compute the residual errors of our predictions similar to SPFN.

To train our global SPFN, we downsample the high-resolution point clouds to have $n = 8$k points using Furthest Point Sampling (FPS). This ensures that we also obtain points on the very tiny primitives preserved in the high-resolution data but discarded in the low-resolution version.

To train our local SPFN, we sample patches located in areas that contain small primitives (with less than $\eta \cdot N$ points) based on the ground truth primitive decomposition. Specifically, we first select a pool of points in the low-resolution point cloud that belong to any of the small primitives. Then, patches of n points are extracted from the high-resolution

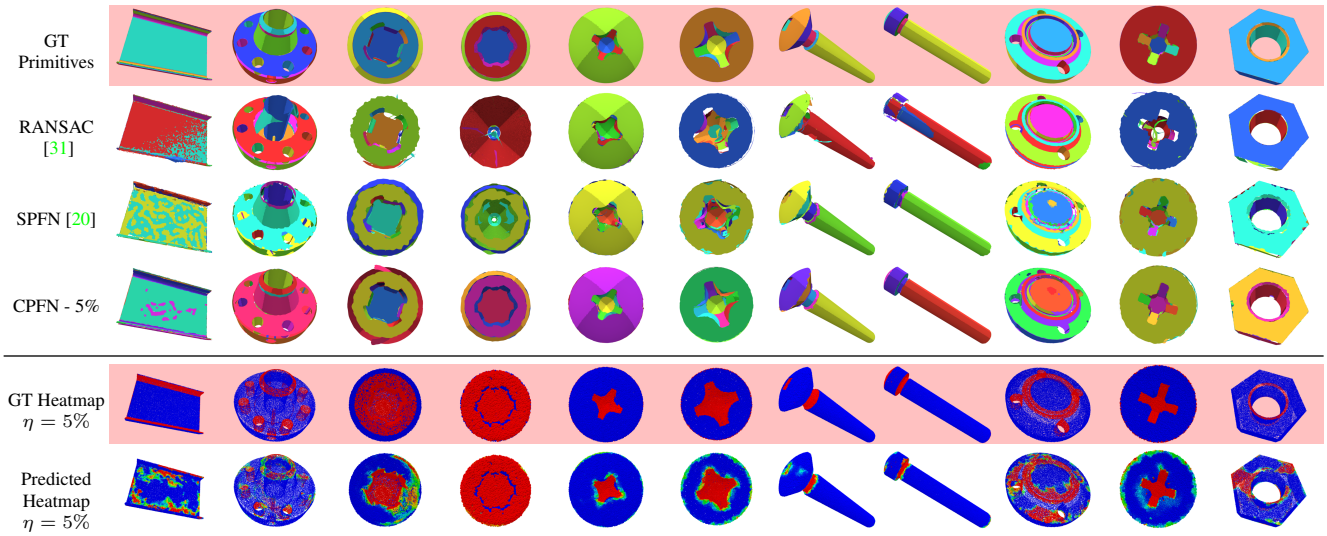

Figure 4. Primitive fitting results for RANSAC, plain SPFN and our CPFN networks. RANSAC and SPFN directly operate on the global object failing on the small primitives. Our CPFN pipeline estimates the heatmaps corresponding to small primitives at different scales and learns a better primitive decomposition on local patches sampled from such regions improving the detection of small primitives. Predicted heatmaps are displayed with the *Jet* color map going from blue to green to red. See the supplementary for more results.

point cloud using n-nearest neighbors by randomly sampling query points from our pool. The sampling process stops when all points in the pool are covered by a patch. Each patch is then centered at the origin and scaled to the unit sphere. Per-point normals and primitive parameters are modified to take into account these transformations. We set $K_{\text{loc}} = 21$ which is the maximum number of primitives observed on a single patch.

4.2. Evaluation metrics

We use the same seven evaluation metrics reported in [20]: (i) segmentation mean intersection over union (mIoU) in %, (ii) mean primitive type accuracy in %, (iii) mean point normal difference in degrees, °, (iv) mean primitive axis difference in degrees, °, (v) mean/std. $\{\mathbf{S}_k\}$ residual, (vi) $\{\mathbf{S}_k\}$ coverage in % and (vii) $\mathbf{P}$ coverage in %.

(i), (ii) and (iv) are computed for all primitives and are then averaged for each point cloud. As mentioned previously, the correspondence between the ground truth and predicted primitives is computed with the Hungarian matching algorithm so that the mIoU is maximized. In these metrics, we weight each primitive the same independently from their scales. (iii) is computed for each point and then averaged for all points within the point cloud.

(v) assesses the fitting error of each predicted primitive $\mathbf{S}_k$ using the 512 points pre-sampled on the assigned ground truth primitive surfaces. (vi) computes for each primitive the proportion of the pre-sampled points that are closer to the predicted primitive surface than an ϵ distance. Both (v) and (vi) are then averaged for each point cloud.

(vii) reports the proportion of points $\mathbf{P}$ that are closer to any predicted primitive surface than an ϵ distance.

We finally report the average of all of these metrics over all the point clouds in the test set.

4.3. Results

Comparison with Global SPFN. As a baseline, we first show the case when using only the global branch of our method. At test time, as the memory limitation due to PointNet++ is lifted, we evaluate the performance of the global SPFN on high-resolution point clouds. As shown in Table 1, the original SPFN (row 2) achieves an mIoU of 66.29% on the original test set [1]. As shown numerically in Table 2 and visually in Figure 4, the performance highly depends on the size of the primitives and significantly drops when only small primitives are considered. Specifically, for primitives that have a scale of less than 1%, i.e., primitives that contain less than $\eta \cdot N$ points, $\eta \leq 1\%$, mIoU drops to 44.25%. Similarly, for $1\% \leq \eta \leq 2\%$, mIoU is 55.53%.

CPFN. We train the local, patch-supervised primitive fitting network (local SPFN) with patches sampled on ground truth small primitives and test on patches identified by our patch selection network. During training, we consider 5 different scales, $\eta \in \{1\%, 2\%, 3\%, 4\%, 5\%\}$, to identify small primitives and hence train 5 different versions of our local SPFN. In Table 1, we report the accuracy of the complete pipeline, i.e., CPFN, using each of the local SPFN versions SPFN (rows 3-7). We further report the accuracy of each local SPFN version at the patch level in Table 3.

As shown by the quantitative numbers, we improve the accuracy of the original SPFN with respect to almost all the

[1] Note that we use a high-resolution version of the dataset that also includes more small primitives. Thus, the numbers reported here are different from the ones in [20].

Table 1. Quantitative comparisons across our CPFN, other baseline methods, and ablation cases. $\mathbf{l}_i$, $\mathbf{l}_o$, and $\mathbf{l}_i^g$ indicate patch features from Local SPFN, object features from global SPFN, patch features from global SPFN, respectively (see Section 3.2), and GP and PN stand for Global SPFN and Patch Selection Network (Section 3.4). If PN is not used, patches are sampled randomly to cover the whole object.

Id	Method	$\mathbf{l}_i$	$\mathbf{l}_o$	$\mathbf{l}_i^g$	GS	PN	Seg. (Mean IoU) (%) ↑	Primitive Type (%) ↑	Point Normal (°) ↓	Primitive Axis (°) ↓	$\{\mathbf{S}_k\}$ Residual Mean ± Std. ↓	$\{\mathbf{S}_k\}$ Coverage (%) ↑ $\epsilon = 0.01$	$\{\mathbf{S}_k\}$ Coverage (%) ↑ $\epsilon = 0.02$	$\mathbf{P}$ Coverage (%) ↑ $\epsilon = 0.01$	$\mathbf{P}$ Coverage (%) ↑ $\epsilon = 0.02$
1	RANSAC [31]						55.01	59.14	11.16	3.24	0.073 ± 0.039	61.88	67.66	78.93	86.64
2	SPFN [20]						66.29	89.50	10.59	1.25	$\mathbf{0.020 \pm 0.010}$	72.94	82.31	88.69	**94.57**
3	CPFN ($\eta = 1\%$)						70.11	93.49	9.37	1.37	0.026 ± 0.013	72.58	80.88	87.60	93.76
4	CPFN ($\eta = 2\%$)						75.85	95.55	7.79	**1.17**	0.034 ± 0.017	72.94	79.58	87.36	92.69
5	CPFN ($\eta = 3\%$)						78.45	96.22	6.87	1.39	0.033 ± 0.017	75.46	81.23	88.84	93.11
6	CPFN ($\eta = 4\%$)						79.09	96.37	6.63	1.48	0.032 ± 0.016	76.63	82.18	**89.19**	93.41
7	CPFN ($\eta = 5\%$)						**79.64**	**96.45**	**6.48**	1.44	0.030 ± 0.015	**76.64**	**82.54**	88.73	93.12
		$\mathbf{l}_i$	$\mathbf{l}_o$	$\mathbf{l}_i^g$	GS	PN	Ablation Study (CPFN, $\eta = 5\%$)								
8		✓			✓	✓	77.70	95.12	6.68	3.07	0.053 ± 0.029	64.55	70.60	85.21	90.53
9		✓	✓		✓	✓	78.30	95.96	6.61	1.70	0.033 ± 0.016	74.43	80.92	87.99	92.55
10		✓		✓	✓	✓	77.95	95.55	6.65	2.51	0.052 ± 0.026	64.72	70.92	86.00	91.37
11		✓	✓	✓			74.13	91.97	**5.15**	1.85	0.063 ± 0.022	71.09	75.73	87.15	91.61
12		✓	✓	✓	✓		78.44	95.47	5.28	1.67	0.052 ± 0.026	50.62	62.36	74.42	83.65
13		✓	✓	✓	✓	✓	**79.64**	**96.45**	6.48	**1.44**	$\mathbf{0.030 \pm 0.015}$	**76.64**	**82.54**	**88.73**	**93.12**
14	CPFN w/ GT Heatmap						80.94	96.74	6.83	1.45	0.028 ± 0.014	79.45	84.94	90.63	94.55

Table 2. The accuracy of CPFN, SPFN, and RANSAC in detecting primitives with varying scales in terms of mIoU (%). GP and PN stand for Global SPFN and Patch Selection Network (Section 3.4). Each scale bucket contains roughly the same number of primitives.

Scale	∼1%	1%∼2%	2%∼4%	4%∼12%	12%∼
RANSAC [31]	34.68	40.38	56.78	70.63	69.50
SPFN [20]	44.25	55.53	70.12	74.29	79.75
CPFN ($\eta = 1\%$)	56.21	61.93	71.91	76.04	80.02
CPFN ($\eta = 2\%$)	63.50	73.31	78.06	79.31	81.93
CPFN ($\eta = 3\%$)	64.89	76.19	82.85	81.62	83.54
CPFN ($\eta = 4\%$)	65.23	76.37	83.71	82.85	83.68
CPFN ($\eta = 5\%$)	**65.74**	**77.31**	**84.19**	**83.55**	**83.95**
GS / PN	Ablation Study (CPFN, $\eta = 5\%$)				
	52.38	66.81	64.36	79.24	**88.24**
✓	60.10	75.32	82.71	**83.83**	85.38
✓ ✓	**65.74**	**77.31**	**84.19**	83.55	83.95

metrics. Specifically, we achieve a significant improvement in terms of mIoU (+13.35%), type accuracy (+6.95%), point normal accuracy (decreasing difference by −38.81%), and $\{\mathbf{S}_k\}$ coverage (+3.70% at $\epsilon = 0.01$) for $\eta = 5\%$.

We note that the patches selected by our selection network do not solely contain small primitives but also a portion of larger primitives. Thus, CPFN has a positive impact both on small and larger primitives, especially improving the predicted normals in high curvature areas between the primitives. Also, by improving the segmentation on the smaller primitives (Figure 4), CPFN achieves a substantial boost in mIoU performance as the metric is independent to the size of the primitive. Differently, small primitives have a lower weight in the computation of $\mathbf{P}$ coverage thus limiting the improvement gain on this metric.

Increasing the scale of the primitives to be used for sampling training patches of local SPFN improves the overall performance. As shown in Tables 1 and 2, CPFN with $\eta = 5\%$ local SPFN generally outperforms the other versions. However, as we discuss in the ablation, training local SPFN on all patches without considering any threshold η results in lower accuracy. Our patch selection strategy ensures the capacity of local SPFN is utilized to improve the detection of small primitives, the main bottleneck of SPFN.

Comparison with RANSAC. We compare our method with the commonly used RANSAC-based approach for primitive fitting. Specifically, we use the efficient RANSAC [31] implementation provided by CGAL [24]. We use the default parameters except for the maximum point-to-surface distance for which we choose a value equal to twice the noise level $2 \cdot \nu = 0.01$.

As shown in Table 1, RANSAC (row 1) under-performs on all metrics compared to both SPFN and CPFN. It is specifically prone to either under- or over-segmentation (see Figure 4) due to noise resulting in a segmentation accuracy as low as 55.01% mIoU. Due to ambiguity in primitive types, it also has a low type prediction accuracy of 59.14%. Table 2 shows that, similar to SPFN, the performance of RANSAC significantly drops for smaller primitives (34.68% and 40.38% mIoU for $\eta \leq 1\%$ and $1\% \leq \eta \leq 2\%$ respectively). The poor segmentation performance has a direct impact on the prediction of primitive axes, residual loss, and coverage.

Effect of Point Cloud Resolution. To assess the performance of our method on different resolutions, we experimented with two additional resolutions, 16k and 64k, as shown in Table 4. The comparison between the results of SPFN and our CPFN shows that *higher resolutions* benefit more from our two-level prediction architecture.

Table 3. Performance of the local SPFNs trained with patches sampled from primitives with varying scales, $\eta \in \{1\%, 2\%, 3\%, 4\%, 5\%\}$, tested at patch level.

Method	Seg. (Mean IoU) (%) ↑	Primitive Type (%) ↑	Point Normal (°) ↓	Primitive Axis (°) ↓	$\{\mathbf{S}_k\}$ Residual Mean ± Std. ↓	$\{\mathbf{S}_k\}$ Coverage (%) ↑		$\mathbf{P}$ Coverage (%) ↑	
						$\epsilon = 0.01$	$\epsilon = 0.02$	$\epsilon = 0.01$	$\epsilon = 0.02$
Local SPFN ($\eta = 1\%$)	74.39	93.12	6.92	1.87	0.097 ± 0.063	56.96	65.26	87.70	93.20
Local SPFN ($\eta = 2\%$)	80.91	95.77	6.12	1.37	**0.079 ± 0.052**	**64.87**	**72.06**	90.65	94.54
Local SPFN ($\eta = 3\%$)	81.53	96.23	6.06	1.33	0.087 ± 0.057	63.50	70.48	90.90	94.78
Local SPFN ($\eta = 4\%$)	81.42	**96.32**	**6.01**	1.35	0.086 ± 0.058	63.72	70.70	**90.99**	**94.87**
Local SPFN ($\eta = 5\%$)	**81.60**	96.07	6.04	**1.29**	0.089 ± 0.059	62.77	69.86	90.53	94.58

Table 4. Quantitative comparison of SPFN [20] and CPFN with different resolutions of the point clouds.

Res.	16k		64k		128k	
Method	SPFN	CPFN	SPFN	CPFN	SPFN	CPFN
Mean IoU (%)	65.86	74.77	66.25	78.29	66.30	**79.64**

Generalization. To evaluate the generalization power of CPFN, we test how it performs on shapes that are acquired from different datasets than it was trained on. As shown in Figure 5, while the use of local patches improves the generalization capability of CPFN, as the shapes become significantly different than those seen during training, the performance degrades.

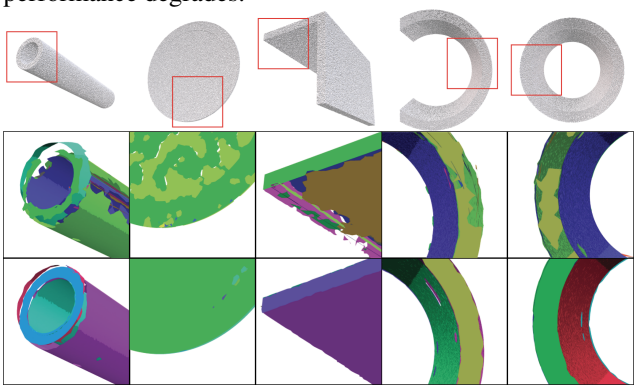

Figure 5. Generalization test results with ABC dataset. From top to bottom, inputs, SPFN [20] results, and CPFN results (in red box areas). Both networks are trained with TraceParts dataset.

4.4. Ablation Study

We motivate the different design choices of our method with a detailed ablation study. We first assess the effect of using additional contextual information as input to our local SPFN. Specifically, we provide both object level global features $\mathbf{l}_o$ and patch level local features $\mathbf{l}_i^g$, both extracted by the global SPFN, as additional input to the local features extracted from the local SPFN $\mathbf{l}_i$ (See Section 3.2). As shown in Table 1, the use of global features $\mathbf{l}_o$ has a positive impact on the patch segmentation accuracy ($+0.60\%$ mIoU, see row 9). More importantly, global features help to reduce the error in predicting primitive parameters that depend on global context such as primitive axis (44.63% improvement). This is further reflected by an improved $\{\mathbf{S}_k\}$ residual by 37.74% and a boost in $\{\mathbf{S}_k\}$ coverage ($+9.88\%$ at $\epsilon = 0.01$). The local features from the global SPFN $\mathbf{l}_i^g$ have a similar positive impact (see row 10). When both features are used in combination, we see a more significant improvement: an increase of $+1.94\%$ in segmentation mIoU, an increase of $+1.33\%$ in type prediction accuracy, a reduction of -53.09% in prediction of primitive axis error, and an increase of $+12.09\%$ in $\{\mathbf{S}_k\}$ coverage at $\epsilon = 0.01$.

We also test our CPFN without using the global SPFN predictions in the merging process (row 11). To cover the entire shape, here we train our local SPFN without the patch selection network (Section 3.4) but randomly sample patches over the entire input point cloud. Compared to our full CPFN pipeline (with $\eta = 5\%$), the overall mIoU drops by 5.51% with this alternative pipeline. We further evaluate CPFN without using the patch selection network (i.e., process all sampled patches by the local SPFN) but still using the global SPFN in the merging step (row 12). The segmentation mIoU slightly decreases compared to the full CPFN baseline (-1.09%). Lastly, we also evaluate the impact of the possible errors introduced by the patch selection network (row 14). With ground truth patch selection provided at test time, we see only a marginal improvement (an improvement of $+1.30\%$ in segmentation mIoU) showing that our method is not very sensitive to patch selection errors.

5. Conclusion

We presented CPFN, a cascaded primitive fitting network that focuses on fitting primitives to high-resolution point clouds obtained by scanning. Our approach consists of a cascade of global fitting network that operates on a downsampled version of the input point cloud as well as a local fitting network that processes local patches on the high resolution point cloud. We present a novel merging formulation that ensembles global and local predictions that outperform state-of-the-art fitting results, especially in regions of fine-scale details. In future work, we would like to explore developing a fully end-to-end trainable pipeline for all of the patch selection, patch-based prediction, and merging steps.

Acknowledgements. The ANSI mechanical component CAD models are originally provided by TraceParts and curated by Li *et al.* [20]. M. Sung acknowledges the support by the National Research Foundation (NRF) grant funded by the Korea government (MSIT) (2021R1F1A1045604).

References

[1] American National Standards Institute (ANSI). 5

[2] Thomas O. Binford. Visual perception by computer. In *IEEE Conference on Systems and Control*, 1971. 1

[3] J. Chen, B. Lei, Q. Song, H. Ying, D. Z. Chen, and J. Wu. A hierarchical graph network for 3d object detection on point clouds. In *CVPR*, 2020. 3

[4] S. Chen, S. Niu, T. Lan, and B. Liu. Pct: Large-scale 3d point cloud representations via graph inception networks with applications to autonomous driving. In *ICIP*, 2019. 3

[5] Z. Chen, A. Tagliasacchi, and H. Zhang. BSP-Net: Generating compact meshes via binary space partitioning. In *CVPR*, 2020. 2

[6] B. Deng, K. Genova, S. Yazdani, S. Bouaziz, G. Hinton, and A. Tagliasacchi. CvxNet: Learnable convex decomposition. In *CVPR*, 2020. 2

[7] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 1981. 1

[8] Matheus Gadelha, Giorgio Gori, Duygu Ceylan, Radomir Mech, Nathan Carr, Tamy Boubekeur, Rui Wang, and Subhransu Maji. Learning generative models of shape handles. In *CVPR*, 2020. 2

[9] K. Genova, F. Cole, A. Sud, A. Sarna, and T. Funkhouser. Local deep implicit functions for 3d shape. In *CVPR*, 2020. 2

[10] K. Genova, F. Cole, D. Vlasic, A. Sarna, W. Freeman, and T. Funkhouser. Learning shape templates with structured implicit functions. In *ICCV*, 2020. 2

[11] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3D semantic segmentation with submanifold sparse convolutional networks. In *CVPR*, 2018. 3

[12] L. Han, T. Zheng, L. Xu, and L. Fang. OccuSeg: Occupancy-aware 3d instance segmentation. In *CVPR*, 2020. 3

[13] Q. Hu, B. Yang, L. Xie, S. Rosa, Y. Guo, Z. Wang, N. Trigoni, and A. Markham. RandLA-Net: Efficient semantic segmentation of large-scale point clouds. In *CVPR*, 2020. 3

[14] L. Jiang, H. Zhao, S. Shi, S. Liu, C. W. Fu, and J. Jia. PointGroup: Dual-set point grouping for 3d instance segmentation. In *CVPR*, 2020. 3

[15] Adrien Kaiser, Jose Alonso Ybanez Zepeda, and Tamy Boubekeur. A survey of simple geometric primitives detection methods for captured 3D data. *Computer Graphics Forum*, 2018. 2

[16] Kacper Kania, Maciej Zięba, and Tomasz Kajdanowicz. UCSG-Net – unsupervised discovering of constructive solid geometry tree, 2020. arXiv:2006.09102. 2

[17] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 1955. 2, 4

[18] David H. Laidlaw, W. Benjamin Trumbore, and John F. Hughes. Constructive solid geometry for polyhedral objects. In *SIGGRAPH*, 1986. 1

[19] L. Landrieu and M. Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs. In *CVPR*, 2018. 3

[20] Lingxiao Li, Minhyuk Sung, Anastasia Dubrovina, Li Yi, and Leonidas Guibas. Supervised fitting of geometric primitives to 3D point clouds. In *CVPR*, 2019. 1, 2, 3, 4, 5, 6, 7, 8

[21] Yangyan Li, Xiaokun Wu, Yiorgos Chrysanthou, Andrei Sharf, Daniel Cohen-Or, and Niloy J. Mitra. GlobFit: Consistently fitting primitives by discovering global relations. In *SIGGRAPH*, 2011. 1

[22] Cheng Lin, Tingxiang Fan, Wenping Wang, and Matthias Nießner. Modeling 3D shapes by reinforcement learning. In *ECCV*, 2020. 2

[23] D. Marr and H. K. Nishihara. Representation and recognition of the spatial organization of three-dimensional shapes. *Proceedings of the Royal Society of London B: Biological Sciences*, 1978. 1

[24] Sven Oesau, Yannick Verdie, Clément Jamin, Pierre Alliez, Florent Lafarge, and Simon Giraudot. Point set shape detection. In *CGAL User and Reference Manual*. CGAL Editorial Board, 4.13 edition, 2018. 1, 7

[25] D. Paschalidou, A. O. Ulusoy, and A. Geiger. Superquadrics revisited: Learning 3d shape parsing beyond cuboids. In *CVPR*, 2019. 2

[26] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. In *CVPR*, 2017. 2

[27] Charles Ruizhongtai Qi, Ly Yi, Hao Su, and Leonidas J. Guibas. PointNet++: Deep hierarchical feature learning on point sets in a metric space. In *NIPS*, 2017. 1, 2

[28] Dario Rethage, Johanna Wald, Jürgen Sturm, Nassir Navab, and Federico Tombari. Fully-convolutional point networks for large-scale point clouds. In *ECCV*, 2018. 3

[29] G. Riegler, A. O. Ulusoy, and A. Geiger. OctNet: Learning deep 3d representations at high resolutions. In *CVPR*, 2017. 3

[30] TraceParts S.A.S. Traceparts. 5

[31] Ruwen Schnabel, Roland Wahl, and Reinhard Klein. Efficient RANSAC for point-cloud shape detection. *Computer Graphics Forum*, 2007. 1, 6, 7

[32] Gopal Sharma, Rishabh Goyal, Difan Liu, Evangelos Kalogerakis, and Subhransu Maji. CSGNet: Neural shape parser for constructive solid geometry. In *CVPR*, 2018. 2

[33] Gopal Sharma, Difan Liu, Evangelos Kalogerakis, Subhransu Maji, Siddhartha Chaudhuri, and Radomír Měch. ParSeNet: A parametric surface fitting network for 3d point clouds. In *ECCV*, 2020. 2

[34] D. Smirnov, M. Fisher, V. G. Kim, R. Zhang, and J. Solomon. Deep parametric shape predictions using distance fields. In *CVPR*, 2020. 2

[35] Chunyu Sun, Qianfang Zou, Xin Tong, and Yang Liu. Learning adaptive hierarchical cuboid abstractions of 3D shape collections. In *SIGGRAPH Asia*, 2019. 2

[36] M. Tatarchenko, J. Park, V. Koltun, and Q. Zhou. Tangent convolutions for dense prediction in 3d. In *CVPR*, 2018. 3

[37] Shubham Tulsiani, Hao Su, Leonidas J. Guibas, Alexei A. Efros, and Jitendra Malik. Learning shape abstractions by assembling volumetric primitives. In *CVPR*, 2017. 2

[38] Q. Xu, X. Sun, C. Y. Wu, P. Wang, and U. Neumann. Grid-GCN for fast and scalable point cloud learning. In *CVPR*, 2020. 3

[39] Chuhang Zou, Ersin Yumer, Jimei Yang, Duygu Ceylan, and Derek Hoiem. 3D-PRNN: Generating shape primitives with recurrent neural networks. In *ICCV*, 2017. 2