Supervised Fitting of Geometric Primitives to 3D Point Clouds

Lingxiao Li*[1] Minhyuk Sung*[1] Anastasia Dubrovina[1] Li Yi[1] Leonidas Guibas[1,2]
[1]Stanford University [2]Facebook AI Research

Abstract

Fitting geometric primitives to 3D point cloud data bridges a gap between low-level digitized 3D data and high-level structural information on the underlying 3D shapes. As such, it enables many downstream applications in 3D data processing. For a long time, RANSAC-based methods have been the gold standard for such primitive fitting problems, but they require careful per-input parameter tuning and thus do not scale well for large datasets with diverse shapes. In this work, we introduce Supervised Primitive Fitting Network (SPFN), an end-to-end neural network that can robustly detect a varying number of primitives at different scales without any user control. The network is supervised using ground truth primitive surfaces and primitive membership for the input points. Instead of directly predicting the primitives, our architecture first predicts per-point properties and then uses a differential model estimation module to compute the primitive type and parameters. We evaluate our approach on a novel benchmark of ANSI 3D mechanical component models and demonstrate a significant improvement over both the state-of-the-art RANSAC-based methods and the direct neural prediction.

1. Introduction

Recent 3D scanning techniques and large-scale 3D repositories have widened opportunities for 3D geometric data processing. However, most of the scanned data and the models in these repositories are represented as digitized point clouds or meshes. Such low-level representations of 3D data limit our ability to geometrically manipulate them due to the lack of structural information aligned with the shape semantics. For example, when editing a shape built from geometric primitives, the knowledge of the type and parameters of each primitive can greatly aid the manipulation in producing a plausible result (Figure 1). To address the absence of such structural information in digitized data, in this work we consider the conversion problem of mapping a 3D point cloud to a number of geometric primitives that best fit the underlying shape.

*equal contribution

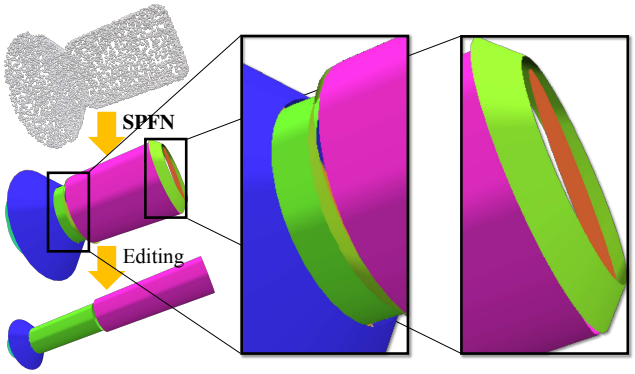

Figure 1: Our network SPFN generates a collection of geometric primitives that fit precisely to the input point cloud, even for tiny segments. The predicted primitives can then be used for structure understanding or shape editing.

Representing an object with a set of simple geometric components is a long-standing problem in computer vision. Since the 1970s [3, 19], the fundamental ideas for tackling the problem have been revised by many researchers, even until recently [31, 34, 9]. However, most of these previous work aimed at solving *perceptual* learning tasks; the main focus was on parsing shapes, or generating a rough abstraction of the geometry with bounding primitives. In contrast, our goal is set at precisely fitting geometric primitives to the shape *surface*, even with the presence of noise in the input.

For this primitive fitting problem, RANSAC-based methods [28] remain the standard. The main drawback of these approaches is the difficulty of finding suitable algorithm parameters. For example, if the threshold of fitting residual for accepting a candidate primitive is smaller than the noise level, over-segmentation may occur, whereas a too large threshold will cause the algorithm to miss small pieces primitives. This problem happens not only when processing noisy scanned data, but also when parsing meshes in 3D repositories because the discretization of the original shape into the mesh obscures the accurate local geometry of the shape surface. The demand for careful user control prevents RANSAC-based methods to scale up to a large number of categories of diverse shapes.

Such drawback motivates us to consider a *supervised* deep learning framework. The primitive fitting problem can

be viewed as a model prediction problem, and the simplest approach would be directly regressing the parameters in the parameter space using a neural network. However, the regression loss based on direct measurement of the parameter difference does not reflect the actual fitting error – the distances between input points and the primitives. Such misinformed loss function can significantly limit prediction accuracy. To overcome this, Brachmann *et al*. [4] integrated the RANSAC pipeline into an end-to-end neural network by replacing the hypothesis selection step with a differentiable procedure. However, their framework predicts only a single model, and it is not straightforward to extend it to predict *multiple* models (primitives in our case). Ranftl *et al*. [26] also introduced a deep learning framework to perform model fitting via inlier weight prediction. We extend this idea to predict weights representing per-point membership for *multiple* primitive models in our setting.

In this work, we propose Supervised Primitive Fitting Network (SPFN) that takes point clouds as input and predicts a varying number of primitives of different types with accurate parameters. For robust estimation, SPFN does not directly output primitive parameters, but instead predicts three kinds of per-point properties: point-to-primitive membership, surface normal, and the type of the primitive the point belongs to. Our framework supports four types of primitives: plane, sphere, cylinder, and cones. These types form the most major components in CAD models. Given these per-point properties, our *differentiable* model estimator computes the primitive parameters in an algebraic way, making the fitting loss fully backpropable. The advantage of our approach is that the network can leverage the readily available supervisions of per-point properties in training. It has been shown that per-point classification problems (membership, type) are suitable to address using a neural network that directly consumes a point cloud as input [24, 25]. Normal prediction can also be handled effectively with a similar neural network [2, 10].

We train and evaluate the proposed method using our novel dataset, ANSI 3D mechanical component models with $17k$ CAD models. The supervision in training is provided by parsing the CAD models and extracting the primitive information. In our comparison experiments, we demonstrate that our supervised approach outperforms the widely used RANSAC-based approach [28] with a big margin, despite using models from *separate* categories in training and testing. Our method shows better fitting accuracy compared to [28] even when we provide the latter with much higher-resolution point clouds as input.

Key contributions

- We propose SPFN, an end-to-end supervised neural network that takes a point cloud as input and detects a varying number of primitives with different scales.
- Our differentiable primitive model estimator solves a series of linear least-square problems, thus making the whole pipeline end-to-end trainable.
- We demonstrate the performance of our network using a novel CAD model dataset of mechanical components.

2. Related Work

Among a large body of previous work on fitting primitives to 3D data, we review only methods that fit primitives to *objects* instead of scenes, as our target use cases are scanned point clouds of individual mechanical parts. For a more comprehensive review, see survey [13].

RANSAC-based Primitive Fitting. RANSAC [8] and its variants [30, 20, 6, 14] are the most widely used methods for primitive detection in computer vision. A significant recent paper by Schnabel *et al*. [28] introduced a robust RANSAC-based framework for detecting multiple primitives of different types in a dense point cloud. Li *et al*. [17] extended [28] by introducing a follow-up optimization that refines the extracted primitives based on the relations among them. As a downstream application of the RANSAC-based methods, Wu *et al*. [32] and Du *et al*. [7] proposed a procedure to reverse-engineer the Constructive Solid Geometry (CSG) model from an input point cloud or mesh. While these RANSAC variants showed state-of-the-art results in their respective fields, their performance typically depends on careful and laborious parameter tuning for each category of shapes. In addition, point normals are required, which are not readily available from 3D scans. In contrast, our supervised deep learning architecture requires only point cloud data as input and does not need any user control at test time.

Network-based Primitive Fitting. Neural networks have been used in recent approaches to solve the primitive fitting problem in both supervised [34] and unsupervised [31, 29] settings. However, these methods are limited in accuracy with a restricted number of supported types. In the work of Zou *et al*. [34] and Tulsiani *et al*. [31], only cuboids are predicted and therefore can only serve as a rough abstraction of the input shape or image. CSGNet [29] is capable of predicting more variety of primitives but with low accuracy, as the parameter extraction is done by performing *classification* on a discretized parameter space. In addition, their reinforcement learning step requires rendering a CSG model to generate visual feedback for every training iteration, making the computation demanding. Our framework can be trained end-to-end and thus does not need expensive external procedures.

3. Supervised Primitive Fitting Network

We propose Supervised Primitive Fitting Network (SPFN) that takes an input shape represented by a point

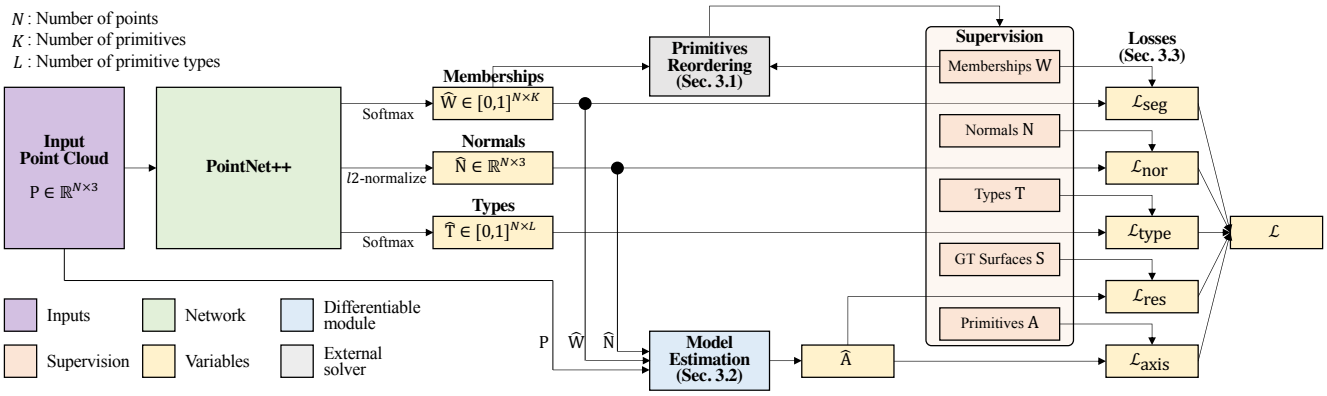

Figure 2: Network architecture. PointNet++ [25] takes input point cloud $\mathbf{P}$ and outputs three per-point properties: point-to-primitive membership $\hat{\mathbf{W}}$, normals $\hat{\mathbf{N}}$, and associated primitive type $\hat{\mathbf{T}}$. The order of ground truth primitives are matched with the output in the primitive reordering step (Section 3.1). Then, the output primitive parameters are estimated from the point properties in the model estimations step (Section 3.2). The loss is defined as the sum of five loss terms (Section 3.3).

cloud $\mathbf{P} \in \mathbb{R}^{N\times 3}$, where N is number of points, and predicts a set of geometric primitives that best fit the input. The output of SPFN contains the type and parameters for every primitive, plus a list of input points assigned to it. Our network supports $L = 4$ types of primitives: plane, sphere, cylinder, and cone (Figure 3), and we index these types by $0, 1, 2, 3$ accordingly. Throughout the paper, we will use notations $\{\cdot\}_{i,:}$ and $\{\cdot\}_{:,k}$ to denote i-th row and k-th column of a matrix, respectively.

During training, for each input shape with K primitives, SPFN leverages the following ground truth information as supervision: point-to-primitive membership matrix $\mathbf{W} \in \{0,1\}^{N\times K}$, unoriented point normals $\mathbf{N} \in \mathbb{R}^{N\times 3}$, and bounded primitive surfaces $\{\mathbf{S}_k\}_{k=1,\dots,K}$. For the membership matrix, $\mathbf{W}_{i,k}$ indicates if point i belongs to primitive k so that $\sum_{k=1}^{K} \mathbf{W}_{i,k} \leq 1$. Notice that $\mathbf{W}_{:,k}$, the k-th column of $\mathbf{W}$, indicates the *point segment* assigned to primitive k. We allow K to *vary* for each shape, and $\mathbf{W}$ can have zero rows indicating *unassigned* points (points not belonging to any of the K primitives; e.g. it belongs to a primitive of unknown type). Each $\mathbf{S}_k$ contains information about the type, parameters, and boundary of the k-th primitive surface, and we denote its type by $\mathbf{t}_k \in \{0, 1, \dots, L-1\}$ and its type-specific parameters by $\mathbf{A}_k$. We include the boundary of $\mathbf{S}_k$ in the supervision besides $\mathbf{P}$ because $\mathbf{P}$ can be noisy, and we do not discriminate against small surfaces in evaluating per-primitive losses (see Equation 17). For convenience, we define per-point type matrix $\mathbf{T} \in \{0,1\}^{N\times L}$ by $\mathbf{T}_{i,l} = \sum_{k=1}^{K} \mathbb{1}(\mathbf{W}_{i,k} = 1)\mathbb{1}(\mathbf{t}_k = l)$, where $\mathbb{1}(\cdot)$ is the indicator function.

The pipeline of SPFN at training time is illustrated in Figure 2. We use PointNet++ [25] segmentation architecture to consume the input point cloud $\mathbf{P}$. A slight modification is that we add three separate fully-connected layers to the end of the PointNet++ pipeline in order to predict the following per-point properties: point-to-primitive membership matrix $\hat{\mathbf{W}} \in [0,1]^{N\times K}$[1], unoriented point normals $\hat{\mathbf{N}} \in \mathbb{R}^{N\times 3}$, and per-point primitive types $\hat{\mathbf{T}} \in [0,1]^{N\times L}$. We use softmax activation to obtain membership probabilities in the rows of $\hat{\mathbf{W}}$ and $\hat{\mathbf{T}}$, and we normalize the rows of the $\hat{\mathbf{N}}$ to constrain normals to have l^2-norm 1. We then feed these per-point quantities to our *differentiable model estimator* (Section 3.2) that computes primitive parameters $\{\hat{\mathbf{A}}_k\}$ based on the per-point information. Since this last step is differentiable, we are able to backpropagate any kind of per-primitive loss through the PointNet++, and thus the training can be done end-to-end.

Notice that we do not assume a consistent ordering of ground truth primitives, so we do not assume any ordering of the columns of our predicted $\hat{\mathbf{W}}$. In Section 3.1, we describe the *primitive reordering* step used to handle such mismatch of orderings. In Section 3.2, we present our differential model estimator for predicting primitive parameters $\{\hat{\mathbf{A}}_k\}$. In Section 3.3, we define each term in our loss function. Lastly, in Section 3.4, we describe implementation details.

3.1. Primitives Reordering

Inspired by Yi *et al*. [33], we compute *Relaxed Intersection over Union* (RIoU) [15] for all pairs of columns from the membership matrices $\mathbf{W}$ and $\hat{\mathbf{W}}$. The RIoU for two indicator vectors $\mathbf{w}$ and $\hat{\mathbf{w}}$ is defined as follows:

$$\text{RIoU}(\mathbf{w}, \hat{\mathbf{w}}) = \frac{\mathbf{w}^{\mathbf{T}}\hat{\mathbf{w}}}{\|\mathbf{w}\|_1 + \|\hat{\mathbf{w}}\|_1 - \mathbf{w}^{\mathbf{T}}\hat{\mathbf{w}}}. \qquad (1)$$

The best one-to-one correspondence (determined by RIoU) between columns of the two matrices is then given by Hungarian matching [16]. We reorder the ground truth primi-

[1]For notational clarity, for now we assume the number of predicted primitives equals K, the number of ground truth primitives. See Section 3.4 for how to predict $\hat{\mathbf{W}}$ without prior knowledge of K.

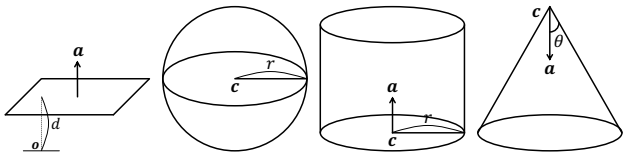

Figure 3: Primitive types and parameters. The boundary information in each $\mathbf{S}_k$, together with parameters $\mathbf{A}_k$, defines the (bounded) region of the primitive k. On the other hand, the point segment $\mathbf{W}_{:,k}$ provides an approximation to this bounded region.

tives according to this correspondence, so that ground truth primitive k is matched with the predicted primitive k. Since the set of inputs where a small perturbation will lead to a change of the matching result has measure zero, the overall pipeline remains differentiable almost everywhere. Hence we use an external Hungarian matching solver to obtain optimal matching indices, and then inject these back into our network to allow further loss computation and gradient propagation.

3.2. Primitive Model Estimation

In the model estimation module, primitive parameters $\{\mathbf{A}_k\}$ are obtained from the predicted per-point properties in a differentiable manner. As the parameter estimation for each primitive is independent, in this section we will assume k is a fixed index of a primitive. The input to the model estimation module consists of $\mathbf{P}$, the input point cloud, $\hat{\mathbf{N}}$, the predicted unoriented point normals, and $\hat{\mathbf{W}}_{:,k}$, the k-th column of the predicted membership matrix $\hat{\mathbf{W}}$. For simplicity, we write $\mathbf{w} = \hat{\mathbf{W}}_{:,k} \in [0,1]^N$ and $\hat{\mathbf{A}} = \hat{\mathbf{A}}_k$. For $\mathbf{p} \in \mathbb{R}^3$, let $D_l(\mathbf{p}, \mathbf{A})$ denote the distance from $\mathbf{p}$ to the primitive of type l and parameters $\mathbf{A}$. The differentiable module for computing $\hat{\mathbf{A}}$, given the primitive type, is illustrated below.

Plane. A plane is represented by $\mathbf{A} = (\mathbf{a}, d)$ where $\mathbf{a}$ is the normal of the plane, with $\|\mathbf{a}\| = 1$, and the points on the plane are $\{\mathbf{p} \in \mathbb{R}^3 : \mathbf{a}^\mathbf{T}\mathbf{p} = d\}$. Then

$$D^2_{\text{plane}}(\mathbf{p}, \mathbf{A}) = (\mathbf{a}^\mathbf{T}\mathbf{p} - d)^2. \tag{2}$$

We can then define $\hat{\mathbf{A}}$ as the minimizer to the *weighted* sum of squared distances as a function of $\mathbf{A}$:

$$\mathcal{E}_{\text{plane}}(\mathbf{A}; \mathbf{P}, \mathbf{w}) = \sum_{i=1}^{N} \mathbf{w}_i (\mathbf{a}^\mathbf{T}\mathbf{P}_{i,:} - d)^2. \tag{3}$$

By solving $\frac{\partial \mathcal{E}_{\text{plane}}}{\partial d} = 0$, we obtain $d = \frac{\sum_{i=1}^N \mathbf{w}_i \mathbf{a}^\mathbf{T}\mathbf{P}_{i,:}}{\sum_{i=1}^N \mathbf{w}_i}$. Plugging this into Equation 3 gives:

$$\mathcal{E}_{\text{plane}}(\mathbf{a}; \mathbf{P}, \mathbf{w}) = \|\text{diag}(\mathbf{w})\,\mathbf{X}\mathbf{a}\|^2, \tag{4}$$

where $\mathbf{X}_{i,:} = \mathbf{P}_{i,:} - \frac{\sum_{i=1}^N \mathbf{w}_i \mathbf{P}_{i,:}}{\sum_{i=1}^N \mathbf{w}_i}$. Hence minimizing $\mathcal{E}_{\text{plane}}(\mathbf{A}; \mathbf{P}, \mathbf{w})$ over $\mathbf{a}$ becomes a homogeneous least square problem subject to $\|\mathbf{a}\| = 1$, and its solution is given as the right singular vector $\mathbf{v}$ corresponding to the smallest singular value of matrix $\text{diag}(\mathbf{w})\,\mathbf{X}$. As shown by Ionescu *et al*. [11, 12], the gradient with respect to $\mathbf{v}$ can be backpropagated through the SVD computation.

Sphere. A sphere is parameterized by $\mathbf{A} = (\mathbf{c}, r)$, where $\mathbf{c} \in \mathbb{R}^3$ is the center and $r \in \mathbb{R}$ is the radius. Hence

$$D^2_{\text{sphere}}(\mathbf{p}, \mathbf{A}) = (\|\mathbf{p} - \mathbf{c}\| - r)^2. \tag{5}$$

In the sphere case (also in the cases of cylinder and cone), the squared distance is not quadratic. Hence minimizing the weighted sum of squared distances over parameters as done in the plane is only available via nonlinear iterative solvers [18]. Instead, we consider minimizing over the weighted sum of a different notion of distance:

$$\mathcal{E}_{\text{sphere}}(\mathbf{A}; \mathbf{P}, \mathbf{w}) = \sum_{i=1}^{N} \mathbf{w}_i (\|\mathbf{P}_{i,:} - \mathbf{c}\|^2 - r^2)^2. \tag{6}$$

Solving $\frac{\partial \mathcal{E}_{\text{sphere}}}{\partial r^2} = 0$ gives $r^2 = \frac{1}{\sum_{i=1}^N \mathbf{w}_i} \sum_{j=1}^N \mathbf{w}_j \|\mathbf{P}_j - \mathbf{c}\|^2$. Putting this back in Equation 6, we end up with a quadratic expression in $\mathbf{c}$ as a least square:

$$\mathcal{E}_{\text{sphere}}(\mathbf{c}; \mathbf{P}, \mathbf{w}, \mathbf{a}) = \|\text{diag}(\mathbf{w})\,(\mathbf{X}\mathbf{c} - \mathbf{y})\|^2, \tag{7}$$

where $\mathbf{X}_{i,:} = 2\left(-\mathbf{P}_{i,:} + \frac{\sum_{j=1}^N \mathbf{w}_j \mathbf{P}_{j,:}}{\sum_{j=1}^N \mathbf{w}_j}\right)$ and $\mathbf{y}_i = \mathbf{P}^\mathbf{T}_{i,:}\mathbf{P}_{i,:} - \frac{\sum_{j=1}^N \mathbf{w}_j \mathbf{P}^\mathbf{T}_{j,:}\mathbf{P}_{j,:}}{\sum_{j=1}^N \mathbf{w}_j}$. This least square can be solved via Cholesky factorization in a differentiable way [21].

Cylinder. A cylinder is parameterized by $\mathbf{A} = (\mathbf{a}, \mathbf{c}, r)$ where $\mathbf{a} \in \mathbb{R}^3$ is a unit vector of the axis, $\mathbf{c} \in \mathbb{R}^3$ is the center, and $r \in \mathbb{R}$ is the radius. We have

$$D^2_{\text{cylinder}}(\mathbf{p}, \mathbf{A}) = \left(\sqrt{\mathbf{v}^\mathbf{T}\mathbf{v} - (\mathbf{a}^\mathbf{T}\mathbf{v})^2} - r\right)^2, \tag{8}$$

where $\mathbf{v} = \mathbf{p} - \mathbf{c}$. As in the sphere case, directly minimizing over squared true distance is challenging. Instead, inspired by Nurunnabi et. al. [22], we first estimate the axis $\mathbf{a}$ and then solve a circle fitting to obtain the rest of the parameters. Observe that the normals of points on the cylinder must be perpendicular to $\mathbf{a}$, so we choose $\mathbf{a}$ to minimize:

$$\mathcal{E}_{\text{cylinder}}(\mathbf{a}; \hat{\mathbf{N}}, \mathbf{w}) = \left\|\text{diag}(\mathbf{w})\,\hat{\mathbf{N}}\mathbf{a}\right\|^2, \tag{9}$$

which is a homogeneous least square problem same as Equation 4, and can be solved in the same way.

Once obtaining the axis $\mathbf{a}$, we consider a plane $\mathscr{P}$ with normal $\mathbf{a}$ that passes through the origin, and notice the projection of the cylinder onto $\mathscr{P}$ should form a circle. Thus we can choose $\mathbf{c}$ and r to be the circle that best fits the projected points $\{\text{Proj}_\mathbf{a}(\mathbf{P}_{i,:})\}_{i=1}^N$, where $\text{Proj}_\mathbf{a}(\cdot)$ denotes the projection onto $\mathscr{P}$. This is exactly the same formulation as in the sphere case (Equation 6), and can thus be solved similarly.

Cone. A cone is parameterized by $\mathbf{A} = (\mathbf{a}, \mathbf{c}, \theta)$ where $\mathbf{c} \in \mathbb{R}^3$ is the apex, $\mathbf{a} \in \mathbb{R}^3$ is a unit vector of the axis from the apex into the cone, and $\theta \in (0, \frac{\pi}{2})$ is the half angle. Then

$$D^2_{\text{cone}}(\mathbf{p}, \mathbf{A})^2 = \left(\|\mathbf{v}\| \sin\left(\min\left(|\alpha - \theta|, \frac{\pi}{2}\right)\right)\right)^2, \quad (10)$$

where $\mathbf{v} = \mathbf{p} - \mathbf{c}, \alpha = \arccos\left(\frac{\mathbf{a}^\mathbf{T}\mathbf{v}}{\|\mathbf{v}\|}\right)$. Similarly with the cylinder case, we use a multi-stage algorithm: first we estimate $\mathbf{a}$ and $\mathbf{c}$ separately, and then we estimate the half-angle θ.

We utilize the fact that the apex $\mathbf{c}$ must be the intersection point of all tangent planes on the cone surface. Using the predicted point normals $\hat{\mathbf{N}}$, the multi-plane intersection problem is formulated as a least square similar with Equation 7 by minimizing

$$\mathcal{E}_{\text{cone}}(\mathbf{c}; \hat{\mathbf{N}}) = \left\|\text{diag}\,(\mathbf{w})\left(\hat{\mathbf{N}}\mathbf{c} - \mathbf{y}\right)\right\|^2, \quad (11)$$

where $\mathbf{y}_i = \hat{\mathbf{N}}^\mathbf{T}_{i,:}\mathbf{P}_{i,:}$. To get the axis direction $\mathbf{a}$, observe that $\mathbf{a}$ should be the normal of the plane passing through all $\mathbf{N}_i$ if point i belongs to the cone. This is just a plane fitting problem, and we can compute $\mathbf{a}$ as the unit normal that minimizes Equation 3, where we replace $\mathbf{P}_{i,:}$ by $\hat{\mathbf{N}}_{i,:}$. We flip the sign of $\mathbf{a}$ if it is not going from $\mathbf{c}$ into the cone. Finally, using the apex $\mathbf{c}$ and the axis $\mathbf{a}$, the half-angle θ is simply computed as a weighted average:

$$\theta = \frac{1}{\sum_{i=1}^N \mathbf{w}_i} \sum_{i=1}^N \mathbf{w}_i \arccos\left|\mathbf{a}^\mathbf{T} \frac{\mathbf{P}_{i,:} - \mathbf{c}}{\|\mathbf{P}_{i,:} - \mathbf{c}\|}\right|. \quad (12)$$

3.3. Loss Function

We define our loss function $\mathcal{L}$ as the sum of the following five terms without weights:

$$\mathcal{L} = \mathcal{L}_{\text{seg}} + \mathcal{L}_{\text{norm}} + \mathcal{L}_{\text{type}} + \mathcal{L}_{\text{res}} + \mathcal{L}_{\text{axis}}. \quad (13)$$

Each loss term is described below for a single input shape.

Segmentation Loss. The primitive parameters can be more accurately estimated when the segmentation of the input point cloud is close to the ground truth. Thus, we minimize $(1 - \text{RIoU})$ for each pair of a ground truth primitive and its correspondence in the prediction:

$$\mathcal{L}_{\text{seg}} = \frac{1}{K}\sum_{k=1}^K \left(1 - \text{RIoU}(\mathbf{W}_{:,k}, \hat{\mathbf{W}}_{:,k})\right). \quad (14)$$

Point Normal Angle Loss. For predicting the point normals $\hat{\mathbf{N}}$ accurately, we minimize the absolute cosine angle between ground truth and predicted normals:

$$\mathcal{L}_{\text{norm}} = \frac{1}{N}\sum_{i=1}^N \left(1 - |\mathbf{N}^\mathbf{T}_{i,:}\hat{\mathbf{N}}_{i,:}|\right). \quad (15)$$

The absolute value is taken since our predicted normals are unoriented.

Per-point Primitive Type Loss. We minimize cross entropy H for the per-point primitive types $\hat{\mathbf{T}}$ (unassigned points are ignored):

$$\mathcal{L}_{\text{type}} = \frac{1}{N}\sum_{i=1}^N \mathbb{1}(\mathbf{W}_{i,:} \neq \mathbf{0}) H(\mathbf{T}_{i,:}, \hat{\mathbf{T}}_{i,:}), \quad (16)$$

where $\mathbb{1}(\cdot)$ is the indicator function.

Fitting Residual Loss. Most importantly, we minimize the expected squared distance between $\mathbf{S}_k$ and the predicted primitive k parameterized by $\hat{\mathbf{A}}_k$ across all $k = 1, \ldots, K$:

$$\mathcal{L}_{\text{res}} = \frac{1}{K}\sum_{k=1}^K \mathbb{E}_{\mathbf{p}\sim U(\mathbf{S}_k)} D^2_{\mathbf{t}_k}(\mathbf{p}, \hat{\mathbf{A}}_k), \quad (17)$$

where $\mathbf{p} \sim U(\mathbf{S})$ means $\mathbf{p}$ is sampled uniformly on the bounded surface $\mathbf{S}$ when taking the expectation, and $D^2_l(\mathbf{p}, \hat{\mathbf{A}})$ is the squared distance from $\mathbf{p}$ to a primitive of type l with parameter $\hat{\mathbf{A}}$, as defined in Section 3.2. Note that every $\mathbf{S}_k$ is weighted equally in Equation 17 regardless of its *scale*, the surface area relative to the entire shape. This allows us to detect small primitives that can be missed by other unsupervised methods.

Note that in Equation 17, we use the ground truth type $\mathbf{t}_k$ instead of inferring the predicted type based on $\hat{\mathbf{T}}$ and then properly weighted by $\hat{\mathbf{W}}$. We do this because coupling multiple predictions can make loss functions more complicated, resulting in unstable training. At test time, however, the type of primitive k is predicted as

$$\hat{\mathbf{t}}_k = \underset{l}{\text{argmax}} \sum_{i=1}^N \hat{\mathbf{T}}_{i,l}\hat{\mathbf{W}}_{i,k}. \quad (18)$$

Axis Angle Loss. Estimating plane normal and cylinder/cone axis using SVD can become numerically unstable when the predicted $\hat{\mathbf{W}}$ leads to degenerate cases, such as when the number of points with a nonzero weight is too small, or when the points with substantial weights form a narrow plane close to a line during plane normal estimation (Equation 4). Thus, we regularize the axis parameters with a cosine angle loss:

$$\mathcal{L}_{\text{axis}} = \frac{1}{K}\sum_{k=1}^K \left(1 - \Theta_{\mathbf{t}_k}(\mathbf{A}_k, \hat{\mathbf{A}}_k)\right), \quad (19)$$

where $\Theta_t(\mathbf{A}, \hat{\mathbf{A}})$ denotes $|\mathbf{a}^\mathbf{T}\hat{\mathbf{a}}|$ for plane (normal), cylinder (axis), and cone (axis), and 1 for sphere (so the loss becomes zero).

3.4. Implementation Details

In our implementation, we assume a fixed number N of input points for all shapes. While the number of ground truth primitives varies across the input shapes, we choose an integer K_{max} in prediction to fix the size the output membership matrix $\hat{\mathbf{W}} \in \mathbb{R}^{N \times K_{\text{max}}}$ so that K_{max} is no less than

the maximum primitive numbers in input shapes. After the Hungarian matching in Section 3.1, unmatched columns in $\hat{\mathbf{W}}$ are ignored in the loss computation. At test time, we discard a predicted primitive k if $\frac{\sum_{i=1}^{N}\hat{\mathbf{W}}_{i,k}}{N} > \epsilon_{\text{discard}}$, where $\epsilon_{\text{discard}} = 0.005N$ for all experiments. This is just a rather arbitrary small threshold to weed out unused segments.

When evaluating the expectation $\mathbb{E}_{\mathbf{p}\sim U(\mathbf{S}_k)}(\cdot)$ in Equation 17, on-the-fly point sampling takes very long time in training. Hence the expectation is approximated as the average for M points on $\mathbf{S}_k$ that are sampled uniformly when preprocessing the data.

4. Experiments

4.1. ANSI Mechanical Component Dataset

For training and evaluating the proposed network, we use CAD models from American National Standards Institute (ANSI) [1] mechanical components provided by TraceParts [27]. Since there is no existing *scanned* 3D dataset for this type of objects, we train and test our network by generating noisy samples on these models. From 504 categories, we randomly select up to 100 models in each category for balance and diversity, and split training/test sets *by categories* so that training and test models are from disjoint categories, resulting in 13,831/3,366 models in training/test sets. We remark that the four types of primitives we consider (plane, sphere, cylinder, cone) cover 94.0% percentage of area per-model on average in our dataset. When generating the point samples from models, we still include surfaces that are not one of the four types. The maximum number of primitives per shape does not exceed 20 in all our models. We set $K_{\text{max}} = 24$ where we add 4 extra columns in $\hat{\mathbf{W}}$ to allow the neural net to assign a small number of points to the extra columns, effectively marking those points unassigned because of the threshold $\epsilon_{\text{discard}}$.

From the CAD models, we extract primitives information including their boundaries. We then merge adjacent pieces of primitive surfaces sharing exactly the same parameters; this happens because of the difficulty of representing boundaries in CAD models, so for instance a complete cylinder will be split into a disjoint union of two mirrored half cylinders. We discard tiny pieces of primitives (less than 2% of the entire area). Each shape is normalized so that its center of mass is at the origin, and the axis-aligned bounding box for the shape is included in $[-1, 1]$ range along every axis. In experiments, we first uniformly sample 8192 points over the entire surface of each shape as the input point cloud ($N = 8192$). This is done by first sampling on the discretized mesh of the shape and then projecting all points onto its geometric surface. Then we randomly apply noise to the point cloud along the surface normal direction in $[-0.01, 0.01]$ range. To evaluate the fitting residual loss $\mathcal{L}_{\text{res}}$, we also uniformly sample 512 points per primitive surface for approximating $\mathbf{S}_k$ ($M = 512$).

4.2. Evaluation Metrics

We design our evaluation metrics as below. Each quantity is described for a single shape, and the numbers are reported as the average of these quantities across all test shapes. For per-primitive metrics, we first perform primitive reordering as in Section 3.1 so the indices for predicted and ground truth primitives are matched.

- **Segmentation Mean IoU**:
  $\frac{1}{K}\sum_{k=1}^{K}\text{IoU}(\mathbf{W}_{:,k}, \mathcal{I}(\hat{\mathbf{W}}_{:,k}))$, where $\mathcal{I}(\cdot)$ is the one-hot conversion.
- **Mean primitive type accuracy**:
  $\frac{1}{K}\sum_{k=1}^{K}\mathbb{1}(\mathbf{t}_k = \hat{\mathbf{t}}_k)$, where $\hat{\mathbf{t}}_k$ is in Equation 18.
- **Mean point normal difference**:
  $\frac{1}{N}\sum_{i=1}^{N}\arccos\left(|\mathbf{N}^{\mathbf{T}}_{i,:}\hat{\mathbf{N}}_{i,:}|\right)$.
- **Mean primitive axis difference**:
  $\frac{1}{\sum_{k=1}^{K}\mathbb{1}(\mathbf{t}_k=\hat{\mathbf{t}}_k)}\sum_{k=1}^{K}\mathbb{1}(\mathbf{t}_k = \hat{\mathbf{t}}_k)\arccos\left(\Theta_{\mathbf{t}_k}(\mathbf{A}_k, \hat{\mathbf{A}}_k)\right)$.
  It is measured only when the predicted type is correct.
- **Mean/Std. $\{\mathbf{S}_k\}$ residual**:
  $\frac{1}{K}\sum_{k=1}^{K}\mathbb{E}_{\mathbf{p}\sim U(\mathbf{S}_k)}\sqrt{D^2_{\hat{\mathbf{t}}_k}(\mathbf{p}, \hat{\mathbf{A}}_k)}$. In contrast to the expression for loss $\mathcal{L}_{\text{res}}$, predicted type $\hat{\mathbf{t}}_k$ is used. The $\{\mathbf{S}_k\}$ residual standard deviation is defined accordingly.
- **$\{\mathbf{S}_k\}$ coverage**:
  $\frac{1}{K}\sum_{k=1}^{K}\mathbb{E}_{\mathbf{p}\sim U(\mathbf{S}_k)}\mathbb{1}\left(\sqrt{D^2_{\hat{\mathbf{t}}_k}(\mathbf{p}, \hat{\mathbf{A}}_k)} < \epsilon\right)$, where ϵ is a threshold.
- **P coverage**:
  $\frac{1}{N}\sum_{i=1}^{N}\mathbb{1}\left(\min_{k=1}^{K}\left(\sqrt{D^2_{\hat{\mathbf{t}}_k}(\mathbf{P}_{\mathbf{i},:}, \hat{\mathbf{A}}_k)}\right) < \epsilon\right)$, where ϵ is a threshold.

When the predicted primitive numbers is less than K, there will be less than K matched pairs in the output of the Hungarian matching. In this case, we modify the metrics of primitive type accuracy, axis difference, and $\{S_k\}$ residual mean/std. to average only over matched pairs.

4.3. Comparison to Efficient RANSAC [28]

We compare the performance of SPFN with Efficient RANSAC [28] and also hybrid versions where we bring in predictions from neural networks as RANSAC input. We use the CGAL [23] implementation of Efficient RANSAC with its default adaptive algorithm parameters. Following common practice, we run the algorithm multiple times (3 in our all experiments), and pick the result with highest input coverage. Different from our pipeline, Efficient RANSAC requires point normals as input. We use the standard jet-fitting algorithm [5] to estimate the point normals from the input point cloud before feeding to RANSAC.

Ind	Method	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	Eff. RANSAC [28]+J	43.68	52.92	11.42	7.54	0.072 ± 0.361	43.42	63.16	65.74	88.63
2	Eff. RANSAC [28]*+J*	56.07	43.90	6.92	2.42	0.067 ± 0.352	56.95	72.74	68.58	92.41
3	Eff. RANSAC [28]+J*	45.90	46.99	**6.87**	5.85	0.080 ± 0.390	51.59	67.12	72.11	92.58
4	Eff. RANSAC [28]+J*+$\hat{\mathbf{W}}$	69.91	60.56	**6.87**	2.90	0.029 ± 0.234	74.32	83.27	78.79	94.58
5	Eff. RANSAC [28]+J*+$\hat{\mathbf{W}}$+$\hat{\mathbf{t}}$	60.68	92.76	**6.87**	6.21	0.036 ± 0.251	65.31	73.69	77.01	92.57
6	Eff. RANSAC [28]+$\hat{\mathbf{N}}$+$\hat{\mathbf{W}}$+$\hat{\mathbf{t}}$	60.56	93.13	8.15	7.02	0.054 ± 0.307	61.94	70.38	74.80	90.83
7	DPPN (Sec. 4.4)	44.05	51.33	-	3.68	0.021 ± 0.158	46.99	71.02	59.74	84.37
8	SPFN-$\mathcal{L}_{\text{seg}}$	41.61	92.40	8.25	1.70	0.029 ± 0.178	50.04	62.74	62.23	77.74
9	SPFN-$\mathcal{L}_{\text{norm}}$+J*	71.18	95.44	**6.87**	4.20	0.022 ± 0.188	76.47	81.49	83.21	91.73
10	SPFN-$\mathcal{L}_{\text{res}}$	72.70	96.66	8.74	1.87	0.017 ± 0.162	79.81	85.57	81.32	91.52
11	SPFN-$\mathcal{L}_{\text{axis}}$	**77.31**	96.47	8.28	6.27	0.019 ± 0.188	80.80	86.11	86.46	94.43
12	SPFN ($\hat{\mathbf{t}} \rightarrow$ Est.)	75.71	95.95	8.54	1.71	0.013 ± 0.140	85.25	90.13	86.67	94.91
13	SPFN	77.14	**96.93**	8.66	**1.51**	**0.011 ± 0.131**	**86.63**	**91.64**	**88.31**	**96.30**

Table 1: Results of all experiments. +J indicates using point normals computed by jet fitting [5] from the input point clouds. The asterisk * indicates using high resolution ($64k$) point clouds. See Section 4.2 for the details of evaluation metrics, and Sections 4.3 to 4.5 for the description of each experiment. Lower is better in 3-5$^{\text{th}}$ metrics, and higher is better in the rest.

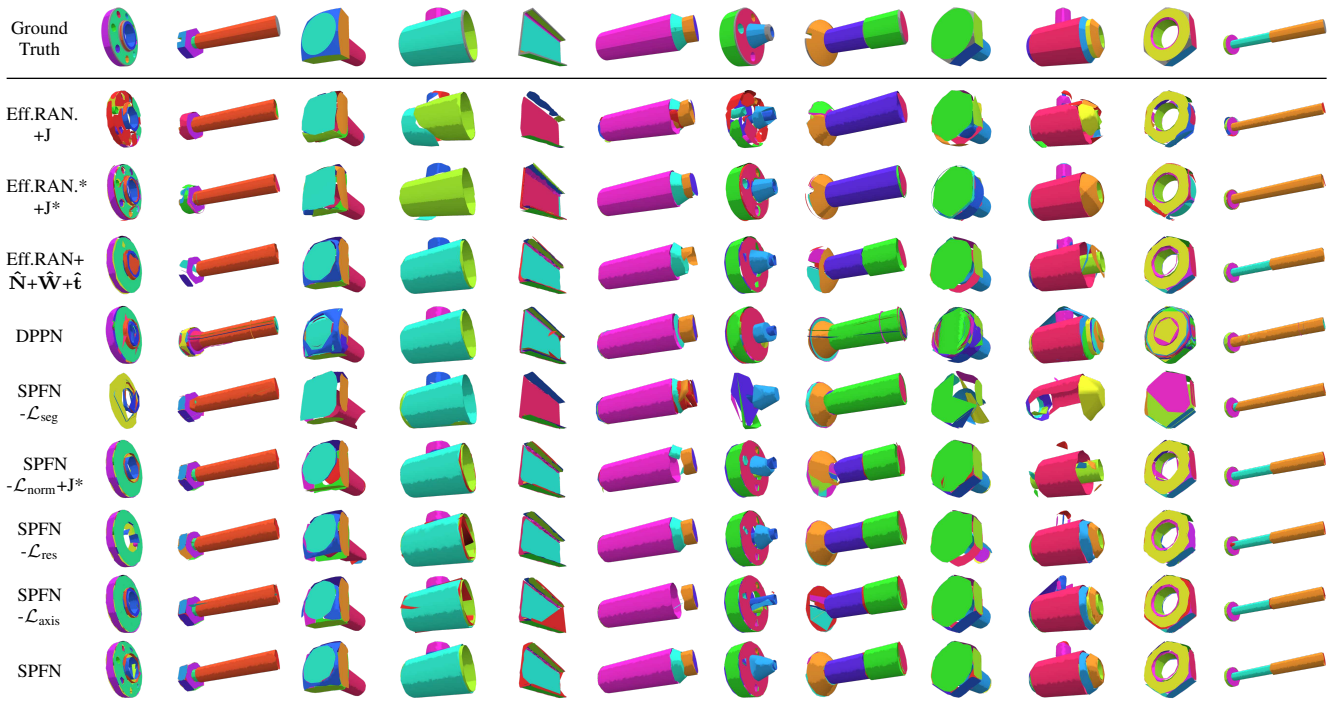

Figure 4: Primitive fitting results with different methods. The results are rendered with meshes generated by projecting point segments to output primitives and then triangulating them. Refer to Sections 4.3 to 4.5 for the details of each method.

We report the results of SPFN and Efficient RANSAC in Table 1. Since Efficient RANSAC can afford point clouds of higher resolution, we test it both with the identical $8k$ input point cloud as in SPFN (row 1), and with another $64k$ input point cloud sampled and perturbed in the same way (row 2). Even compared to results from high-resolution point clouds, SPFN outperforms Efficient RANSAC in all metrics. Specifically, both $\{\mathbf{S}_k\}$ and $\mathbf{P}$ coverage numbers with threshold $\epsilon = 0.01$ show big margins, demonstrating that our SPFN fits primitives more precisely.

We also test Efficient RANSAC by bringing in per-point properties predicted by SPFN. We first train SPFN with only $\mathcal{L}_{\text{seg}}$ loss, and then for each segment in the predicted membership matrix $\hat{\mathbf{W}}$ we use Efficient RANSAC to predict a single primitive (Table 1, row 4). We further add $\mathcal{L}_{\text{type}}$ and $\mathcal{L}_{\text{norm}}$ losses in training sequentially, and use the predicted primitive types $\hat{\mathbf{t}}$ and point normals $\hat{\mathbf{N}}$ in Efficient RANSAC (row 5-6). When the input point cloud is first segmented with a neural network, both $\{\mathbf{S}_k\}$ and $\mathbf{P}$ coverage numbers for Efficient RANSAC increase significantly, yet still lower than SPFN. Notice that the point normals and primitive types predicted by a neural network do not im-

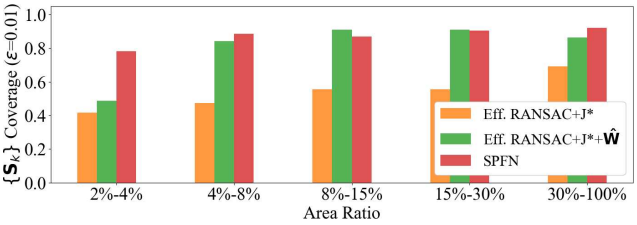

Figure 5: $\{\mathbf{S}_k\}$ coverage against scales of primitives.

prove the $\{\mathbf{S}_k\}$ and $\mathbf{P}$ coverage in RANSAC.

Figure 5 illustrates $\{\mathbf{S}_k\}$ coverage with $\epsilon = 0.01$ for varying scales of ground truth primitives. Efficient RANSAC coverage improves when leveraging the segmentation results of the network, but still remains low when the scale is small. In contrast, SPFN exhibits consistent high coverage for all scales.

4.4. Comparison to Direct Parameter Prediction Network (DPPN)

We also consider a simple neural network named Direct Parameter Prediction Network (DPPN) that directly predicts primitive parameters without predicting point properties as an intermediate step. DPPN uses the same PointNet++ [25] architecture that consumes $\mathbf{P}$, but different from SPFN, it outputs K_{max} primitive parameters for *every* primitive type (so it gives $4K_{\text{max}}$ sets of parameters). In training, the Hungarian matching to the ground truth primitives (Section 3.1) is performed with fitting residuals as in Equation 17 instead of RIoU. Since point properties are not predicted and the matching is based solely on fitting residuals (so the primitive type might mismatch), only $\mathcal{L}_{\text{res}}$ is used as the loss function. At test time, we assign each input point to the closest predicted primitive to form $\hat{\mathbf{W}}$.

The results are reported in row 7 of Table 1. Compared to SPFN, both $\{\mathbf{S}_k\}$ and $\mathbf{P}$ coverage numbers are far lower, particularly when the threshold is small ($\epsilon = 0.01$). This implies that supervising a network not only with ground truth primitives but also with *point-to-primitive* associations is crucial for more accurate predictions.

4.5. Ablation Study

We conduct ablation study to verify the effect of each loss term. In Table 1 rows 8-11, we report the results when we exclude $\mathcal{L}_{\text{seg}}$, $\mathcal{L}_{\text{norm}}$ (use jet-fitting normals computed from $64k$ points), $\mathcal{L}_{\text{res}}$, and $\mathcal{L}_{\text{axis}}$, respectively. The coverage numbers drop the most when the segmentation loss $\mathcal{L}_{\text{seg}}$ is not used (-$\mathcal{L}_{\text{seg}}$). When using point normals computed from $64k$ input point clouds instead of predicting them (-$\mathcal{L}_{\text{norm}}$+J*), the coverage also drops despite more accurate point normals. This implies that SPFN predicts point normals in a way to better fit primitives rather than to just accurately predict the normals. Without including the fitting residual loss (-$\mathcal{L}_{\text{res}}$), we see a drop in coverage and segmentation accuracy. Excluding the primitive axis loss $\mathcal{L}_{\text{axis}}$ not only hurts the axis accuracy, but also gives lower coverage numbers (especially $\{\mathbf{S}_k\}$ coverage). Row 12 ($\hat{\mathbf{t}} \rightarrow$ Est.) shows results when using predicted types $\hat{\mathbf{t}}$ in the fitting residual loss (Equation 17) instead of the ground truth types $\mathbf{t}$. The results are compatible but slightly worse than SPFN where we decouple type and other predictions in training.

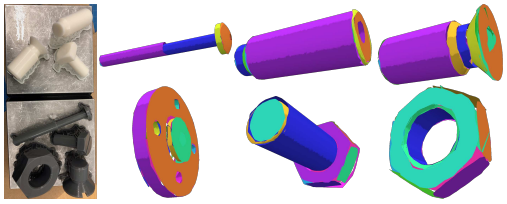

Figure 6: Results with real scans. Left are the 3D-printed CAD models from the test set.

4.6. Results with Real Scans

For testing with real noise patterns, we 3D-printed some test models and scanned the outputs using a DAVID SLS-2 3D Scanner. Notice that SPFN trained on synthesized noises successfully reconstructed all primitives including the small segments (Figure 6).

5. Conclusion

We have presented Supervised Primitive Fitting Network (SPFN), a fully differentiable network architecture that predicts a varying number of geometric primitives from a 3D point cloud, potentially with noise. In contrast to directly predicting primitive parameters, SPFN predicts per-point properties and then derive the primitive parameters using a novel differentiable model estimator. The strong supervision we provide allows SPFN to accurately predict primitives of different scales that closely abstract the underlying geometric shape surface, without any user control. We demonstrated in experiments that this approach gives significant better results compared to both the RANSAC-based method [28] and direct parameters prediction. We also introduced a new CAD model dataset, ANSI mechanical component dataset, along with a set of comprehensive evaluation metrics, based on which we performed our comparison and ablation studies.

Acknowledgments. The authors wish to thank Chengtao Wen and Mohsen Rezayat for valuable discussions and for making relevant data available to the project. Also, the authors thank TraceParts for providing ANSI Mechanical Component CAD models. This project is supported by a grant from the Siemens Corporation, NSF grant CHS-1528025 a Vannevar Bush Faculty Fellowship, and gifts from and Adobe and Autodesk. A. Dubrovina acknowledges the support in part by The Eric and Wendy Schmidt Postdoctoral Grant for Women in Mathematical and Computing Sciences.

References

[1] American National Standards Institute (ANSI). 6

[2] Aayush Bansal, Bryan Russell, and Abhinav Gupta. Marr revisited: 2D-3D alignment via surface normal prediction. *CVPR*, 2016. 2

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

[4] Eric Brachmann, Alexander Krull, Sebastian Nowozin, Jamie Shotton, Frank Michel, Stefan Gumhold, and Carsten Rother. DSAC - Differentiable RANSAC for camera localization. In *CVPR*, 2017. 2

[5] F. Cazals and M. Pouget. Estimating differential quantities using polynomial fitting of osculating jets. *Symposium on Geometry Processing (SGP)*, 2003. 6, 7

[6] Ondrej Chum and Jiri Matas. Matching with PROSAC - Progressive sample consensus. In *CVPR*, 2005. 2

[7] Tao Du, Jeevana Priya Inala, Yewen Pu, Andrew Spielberg, Adriana Schulz, Daniela Rus, Armando Solar-Lezama, and Wojciech Matusik. InverseCSG: Automatic conversion of 3D models to CSG trees. *SIGGRAPH Asia*, 2018. 2

[8] 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. 2

[9] Vignesh Ganapathi-Subramanian, Olga Diamanti, Soeren Pirk, Chengcheng Tang, Matthias Niessner, and Leonidas J. Guibas. Parsing geometry using structure-aware shape templates. In *3DV*, 2018. 1

[10] Paul Guerrero, Yanir Kleiman, Maks Ovsjanikov, and Niloy J. Mitra. PCPNET: Learning local shape properties from raw point cloud. *Eurographics*, 2018. 2

[11] Catalin Ionescu, Orestis Vantzos, and Cristian Sminchisescu. Matrix backpropagation for deep networks with structured layers. 2015. 4

[12] Catalin Ionescu, Orestis Vantzos, and Cristian Sminchisescu. Training deep networks with structured layers by matrix backpropagation. *CoRR*, abs/1509.07838, 2015. 4

[13] 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

[14] Zhizhong Kang and Zhen Li. Primitive fitting based on the efficient multiBaySAC algorithm. *PloS one*, 2015. 2

[15] Philipp Krähenbühl and Vladlen Koltun. Parameter learning and convergent inference for dense random fields. In *ICML*, 2013. 3

[16] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 1955. 3

[17] Yangyan Li, Xiaokun Wu, Yiorgos Chrysanthou, Andrei Sharf, Daniel Cohen-Or, and Niloy J. Mitra. Globfit: Consistently fitting primitives by discovering global relations. *SIGGRAPH*, 2011. 2

[18] Gabor Lukács, Ralph Martin, and Dave Marshall. Faithful least-squares fitting of spheres, cylinders, cones and tori for reliable segmentation. In *ECCV*, 1998. 4

[19] 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

[20] J. Matas and O. Chum. Randomized RANSAC with T(d,d) test. *Image and Vision Computing*, 2004. 2

[21] Iain Murray. Differentiation of the Cholesky decomposition, 2016. arXiv:1602.07527. 4

[22] A. Nurunnabi, Y. Sadahiro, and R. Lindenbergh. Robust cylinder fitting in three-dimensional point cloud data. In *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci.*, 2017. 4

[23] 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. 6

[24] 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

[25] 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. 2, 3, 8

[26] Renè Ranftl and Vladlen Koltun. Deep fundamental matrix estimation. In *ECCV*, 2018. 2

[27] TraceParts S.A.S. Traceparts. 6

[28] Ruwen Schnabel, Roland Wahl, , and Reinhard Klein. Efficient RANSAC for point-cloud shape detection. *Computer graphics forum*, 2007. 1, 2, 6, 7, 8

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

[30] P. H. S. Torr and A. Zisserman. MLESAC: A new robust estimator with application to estimating image geometry. *CVIU*, 2000. 2

[31] Shubham Tulsiani, Hao Su, Leonidas J. Guibas, Alexei A. Efros, and Jitendra Malik. Learning shape abstractions by assembling volumetric primitives. In *CVPR*, 2017. 1, 2

[32] Qiaoyun Wu, Kai Xu, and Jun Wang. Constructing 3D CSG models from 3D raw point clouds. *Symposium on Geometry Processing (SGP)*, 2018. 2

[33] Li Yi, Haibin Huang, Difan Liu, Evangelos Kalogerakis, Hao Su, and Leonidas Guibas. Deep part induction from articulated object pairs. *SIGGRAPH Asia*, 2018. 3

[34] Chuhang Zou, Ersin Yumer, Jimei Yang, Duygu Ceylan, and Derek Hoiem. 3D-PRNN: Generating shape primitives with recurrent neural networks. In *ICCV*, 2017. 1, 2