PARSENET: A Parametric Surface Fitting Network for 3D Point Clouds

Gopal Sharma[1], Difan Liu[1], Subhransu Maji[1],
Evangelos Kalogerakis[1], Siddhartha Chaudhuri[2,3], and Radomír Měch[2]
[1]{gopalsharma, dliu, smaji, kalo}@cs.umass.edu, [2]{sidch,rmech}@adobe.com

[1]University of Massachusetts Amherst [2]Adobe Research [3]IIT Bombay

Abstract. We propose a novel, end-to-end trainable, deep network called PARSENET that decomposes a 3D point cloud into parametric surface patches, including B-spline patches as well as basic geometric primitives. PARSENET is trained on a large-scale dataset of man-made 3D shapes and captures high-level semantic priors for shape decomposition. It handles a much richer class of primitives than prior work, and allows us to represent surfaces with higher fidelity. It also produces repeatable and robust parametrizations of a surface compared to purely geometric approaches. We present extensive experiments to validate our approach against analytical and learning-based alternatives. Our source code is publicly available at: https://hippogriff.github.io/parsenet.

1 Introduction

3D point clouds can be rapidly acquired using 3D sensors or photogrammetric techniques. However, they are rarely used in this form in design and graphics applications. Observations from the computer-aided design and modeling literature [4, 5, 15, 17] suggest that designers often model shapes by constructing several non-overlapping patches placed seamlessly. The advantage of using several patches over a single continuous patch is that a much more diverse variety of geometric features and surface topologies can be created. The decomposition also allows easier interaction and editing. The goal of this work is to automate the time-consuming process of converting a 3D point cloud into a piecewise parametric surface representation as seen in Figure 1.

An important question is how surface patches should be represented. Patch representations in CAD and graphics are based on well-accepted geometric properties: (a) *continuity* in their tangents, normals, and curvature, making patches appear smooth, (b) *editability*, such that they can easily be modified based on a few intuitive degrees of freedom (DoFs), *e.g.*, control points or axes, and (c) *flexibility*, so that a wide variety of surface geometries can be captured. Towards this goal, we propose PARSENET, a **par**ametric **s**urfac**e** fitting **net**work architecture which produces a compact, editable representation of a point cloud as an assembly of geometric primitives, including open or closed B-spline patches.

PARSENET models a richer class of surfaces than prior work which *only* handles basic geometric primitives such as planes, cuboids and cylinders [12,

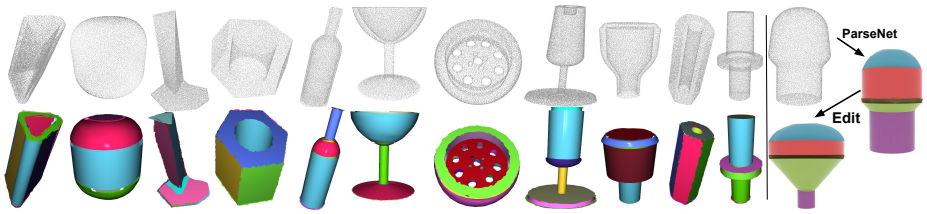

Fig. 1: ParSeNet decomposes point clouds (top row) into collections of assembled parametric surface patches including B-spline patches (bottom row). On the right, a shape is edited using the inferred parametrization.

13, 19, 26]. While such primitives are continuous and editable representations, they lack the richness and flexibility of spline patches which are widely used in shape design. ParSeNet includes a novel neural network (SplineNet) to estimate an open or closed B-spline model of a point cloud patch. It is part of a *fitting module* (Section 3.2) which can also fit other geometric primitive types. The fitting module receives input from a *decomposition module*, which partitions a point cloud into segments, after which the fitting module estimates shape parameters of a predicted primitive type for each segment (Section 3.1). The entire pipeline, shown in Figure 2, is fully differentiable and trained end-to-end (Section 4). An optional geometric postprocessing step further refines the output.

Compared to purely analytical approaches, ParSeNet produces decompositions that are more consistent with high-level semantic priors, and are more robust to point density and noise. To train and test ParSeNet, we leverage a recent dataset of man-made parts [9]. Extensive evaluations show that ParSeNet outperforms baselines (RANSAC and SPFN [12]) by 14.93% and 13.13% respectively for segmenting a point cloud into patches, and by 50%, and 47.64% relative error respectively for parametrizing each patch for surface reconstruction (Section 5).

To summarize, our contributions are:

- The first proposed end-to-end differentiable approach for representing a raw 3D point cloud as an assembly of parametric primitives *including* spline patches.
- Novel decomposition and primitive fitting modules, including SplineNet, a fully-differentiable network to fit a cubic B-spline patch to a set of points.
- Evaluation of our framework vs prior analytical and learning-based methods.

2 Related Work

Our work builds upon related research on parametric surface representations and methods for primitive fitting. We briefly review relevant work in these areas. Of course, we also leverage extensive prior work on neural networks for general shape processing: see recent surveys on the subject [1].

Parametric surfaces. A parametric surface is a (typically diffeomorphic) mapping from a (typically compact) subset of $\mathbb{R}^2$ to $\mathbb{R}^3$. While most of the geometric

primitives used in computer graphics (spheres, cuboids, meshes etc) can be represented parametrically, the term most commonly refers to curved surfaces used in engineering CAD modelers, represented as spline patches [4]. There are a variety of formulations – *e.g.* Bézier patches, B-spline patches, NURBS patches – with slightly different characteristics, but they all construct surfaces as weighted combinations of control parameters, typically the positions of a sparse grid of points which serve as editing handles.

More specifically, a B-spline patch is a smoothly curved, bounded, parametric surface, whose shape is defined by a sparse grid of control points $\mathbf{C} = \{\mathbf{c}_{p,q}\}$. The surface point with parameters $(u, v) \in [u_{\min}, u_{\max}] \times [v_{\min}, v_{\max}]$ and *basis functions* [4] $b_p(u)$, $b_q(v)$ is given by:

$$\mathbf{s}(u, v) = \sum_{p=1}^{P} \sum_{q=1}^{Q} b_p(u) b_q(v) \mathbf{c}_{p,q} \tag{1}$$

Please refer to supplementary material for more details on B-spline patches.

Fitting geometric primitives. A variety of analytical (*i.e.* not learning-based) algorithms have been devised to approximate raw 3D data as a collection of geometric primitives: dominant themes include Hough transforms, RANSAC and clustering. The literature is too vast to cover here, we recommend the comprehensive survey of Kaiser *et al.* [8]. In the particular case of NURBS patch fitting, early approaches were based on user interaction or hand-tuned heuristics to extract patches from meshes or point clouds [3,7,11]. In the rest of this section, we briefly review recent methods that *learn* how to fit primitives to 3D data.

Several recent papers [22,24,26,29] also try to approximate 3D shapes as unions of cuboids or ellipsoids. Paschalidou *et al.* [13,14] extended this to superquadrics. Sharma *et al.* [19,20] developed a neural parser that represents a test shape as a collection of basic primitives (spheres, cubes, cylinders) combined with boolean operations. Tian *et al.* [25] handled more expressive construction rules (*e.g.* loops) and a wider set of primitives. Because of the choice of simple primitives, such models are naturally limited in how well they align to complex input objects, and offer less flexible and intuitive parametrization for user edits.

More relevantly to our goal of modeling arbitrary curved surfaces, Gao *et al.* [6] parametrize 3D point clouds as extrusions or surfaces of revolution, generated by B-spline cross-sections detected by a 2D network. This method requires translational/rotational symmetry, and does not apply to general curved patches. Li *et al.* [12] proposed a supervised method to fit primitives to 3D point clouds, first predicting per-point segment labels, primitive types and normals, and then using a differential module to estimate primitive parameters. While we also chain segmentation with fitting in an end-to-end way, we differ from Li *et al.* in two important ways. First, our differentiable metric-learning segmentation produces improved results (Table 1). Second, a major goal (and technical challenge) for us is to significantly improve expressivity and generality by incorporating B-spline patches: we achieve this with a novel differentiable spline-fitting network. In a complementary direction, Yumer *et al.* [28] developed a neural network for fitting

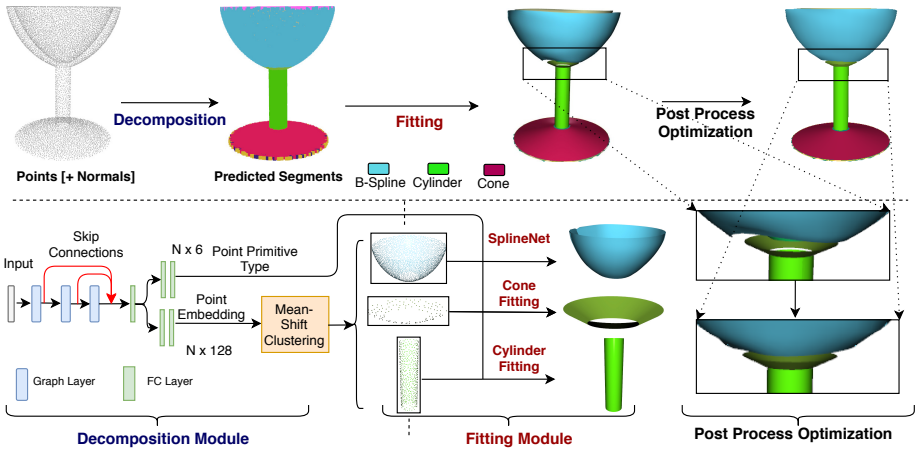

Fig. 2: **Overview of ParSeNet pipeline.** (1) The *decomposition module* (Section 3.1) takes a 3D point cloud (with optional normals) and decomposes it into segments labeled by primitive type. (2) The *fitting module* (Section 3.2) predicts parameters of a primitive that best approximates each segment. It includes a novel SplineNet to fit B-spline patches. The two modules are jointly trained end-to-end. An optional postprocess module (Section 3.3) refines the output.

a single NURBS patch to an unstructured point cloud. While the goal is similar to our spline-fitting network, it is not combined with a decomposition module that jointly learns how to express a shape with *multiple* patches covering different regions. Further, their fitting module has several non-trainable steps which are not obviously differentiable, and hence cannot be used in our pipeline.

3 Method

The goal of our method is to reconstruct an input point cloud by predicting a set of parametric patches closely approximating its underlying surface. The first stage of our architecture is a *neural decomposition module* (Fig. 2) whose goal is to segment the input point cloud into regions, each labeled with a parametric patch type. Next, we incorporate a *fitting module* (Fig. 2) that predicts each patch's shape parameters. Finally, an optional post-processing *geometric optimization* step refines the patches to better align their boundaries for a seamless surface.

The input to our pipeline is a set of points $\mathbf{P} = \{\mathbf{p}_i\}_{i=1}^{N}$, represented either as 3D positions $\mathbf{p}_i = (x, y, z)$, or as 6D position + normal vectors $\mathbf{p}_i = (x, y, z, n_x, n_y, n_z)$. The output is a set of surface patches $\{\mathbf{s}_k\}$, reconstructing the input point cloud. The number of patches is automatically determined. Each patch is labeled with a type t_k, one of: sphere, plane, cone, cylinder, open/closed B-spline patch. The architecture also outputs a real-valued vector for

each patch defining its geometric parameters, *e.g.* center and radius for spheres, or B-spline control points and knots.

3.1 Decomposition module

The first module (Fig. 2) decomposes the point cloud $\mathbf{P}$ into a set of segments such that each segment can be reliably approximated by one of the abovementioned surface patch types. To this end, the module first embeds the input points into a representation space used to reveal such segments. As discussed in Section 4, the representations are learned using metric learning, such that points belonging to the same patch are embedded close to each other, forming a distinct cluster.

Embedding network. To learn these point-wise representations, we incorporate edge convolution layers (EdgeConv) from DGCNN [27]. Each EdgeConv layer performs a graph convolution to extract a representation of each point with an MLP on the input features of its neighborhood. The neighborhoods are dynamically defined via nearest neighbors in the input feature space. We stack 3 EdgeConv layers, each extracting a 256-D representation per point. A max-pooling layer is also used to extract a global 1024-D representation for the whole point cloud. The global representation is tiled and concatenated with the representations from all three EdgeConv layers to form intermediate point-wise $(1024 + 256)$-D representations $\mathbf{Q} = \{\mathbf{q}_i\}$ encoding both local and global shape information. We found that a global representation is useful for our task, since it captures the overall geometric shape structure, which is often correlated with the number and type of expected patches. This representation is then transformed through fully connected layers and ReLUs, and finally normalized to unit length to form the point-wise embedding $\mathbf{Y} = \{\mathbf{y}_i\}_{i=1}^{N}$ (128-D) lying on the unit hypersphere.

Clustering. A mean-shift clustering procedure is applied on the point-wise embedding to discover segments. The advantage of mean-shift clustering over other alternatives (*e.g.*, k-means or mixture models) is that it does not require the target number of clusters as input. Since different shapes may comprise different numbers of patches, we let mean-shift produce a cluster count tailored for each input. Like the pixel grouping of [10], we implement mean-shift iterations as differentiable recurrent functions, allowing back-propagation. Specifically, we initialize mean-shift by setting all points as seeds $\mathbf{z}_i^{(0)} = \mathbf{y}_i, \forall y_i \in R^{128}$. Then, each mean-shift iteration t updates each point's embedding on the unit hypersphere:

$$\mathbf{z}_i^{(t+1)} = \sum_{j=1}^{N} \mathbf{y}_j g(\mathbf{z}_i^{(t)}, \mathbf{y}_j) / (\sum_{j=1}^{N} g(\mathbf{z}_i^{(t)}, \mathbf{y}_j)) \tag{2}$$

where the pairwise similarities $g(\mathbf{z}_i^{(t)}, \mathbf{y}_j)$ are based on a von Mises-Fisher kernel with bandwidth β: $g(\mathbf{z}_i, \mathbf{y}_j) = \exp(\mathbf{z}_i^T \mathbf{y}_j / \beta^2)$ (iteration index dropped for clarity). The embeddings are normalized to unit vectors after each iteration. The bandwidth for each input point cloud is set as the average distance of each point to its 150^{th} neighboring point in the embedding space [21]. The mean-shift iterations

are repeated until convergence (this occurs around 50 iterations in our datasets). We extract the cluster centers using non-maximum suppression: starting with the point with highest density, we remove all points within a distance β, then repeat. Points are assigned to segments based on their nearest cluster center. The point memberships are stored in a matrix $\mathbf{W}$, where $\mathbf{W}[i,k] = 1$ means point i belongs to segment k, and 0 means otherwise. The memberships are passed to the fitting module to determine a parametric patch per segment. During training, we use soft memberships for differentiating this step (more details in Section 4.3).

Segment Classification. To classify each segment, we pass the per-point representation $\mathbf{q}_i$, encoding local and global geometry, through fully connected layers and ReLUs, followed by a softmax for a per-point probability $P(t_i = l)$, where l is a patch type (*i.e.*, sphere, plane, cone, cylinder, open/closed B-spline patch). The segment's patch type is determined through majority voting over all its points.

3.2 Fitting module

The second module (Fig. 2) aims to fit a parametric patch to each predicted segment of the point cloud. To this end, depending on the segment type, the module estimates the shape parameters of the surface patch.

Basic primitives. Following Li *et al*. [12], we estimate the shape of basic primitives with least-squares fitting. This includes center and radius for spheres; normal and offset for planes; center, direction and radius for cylinders; and apex, direction and angle for cones. We also follow their approach to define primitive boundaries.

B-Splines. Analytically parametrizing a set of points as a spline patch in the presence of noise, sparsity and non-uniform sampling, can be error-prone. Instead, predicting control points directly with a neural network can provide robust results. We propose a neural network SplineNet, that inputs points of a segment, and outputs a fixed size control-point grid. A stack of three EdgeConv layers produce point-wise representations concatenated with a global representation extracted from a max-pooling layer (as for decomposition, but weights are not shared). This equips each point i in a segment with a 1024-D representation $\boldsymbol{\phi}_i$. A segment's representation is produced by max-pooling over its points, as identified through the membership matrix $\mathbf{W}$ extracted previously:

$$\boldsymbol{\phi}_k = \max_{i=1\ldots N} (\mathbf{W}[i,k] \cdot \boldsymbol{\phi}_i). \tag{3}$$

Finally, two fully-connected layers with ReLUs transform $\boldsymbol{\phi}_k$ to an initial set of 20×20 control points $\mathbf{C}$ unrolled into a 1200-D output vector. For a segment with a small number of points, we upsample the input segment (with nearest neighbor interpolation) to 1600 points. This significantly improved performance for such segments (Table 2). For closed B-spline patches, we wrap the first row/column of control points. Note that the network parameters to produce open and closed B-splines are not shared. Fig. 5 visualizes some predicted B-spline surfaces.

3.3 Post-processing module

SplineNet produces an initial patch surface that approximates the points belonging to a segment. However, patches might not entirely cover the input point cloud, and boundaries between patches are not necessarily well-aligned. Further, the resolution of the initial control point grid (20×20) can be further adjusted to match the desired surface resolution. As a post-processing step, we perform an optimization to produce B-spline surfaces that better cover the input point cloud, and refine the control points to achieve a prescribed fitting tolerance.

Optimization. We first create a grid of 40×40 points on the initial B-spline patch by uniformly sampling its UV parameter space. We tessellate them into quads. Then we perform a maximal matching between the quad vertices and the input points of the segment, using the Hungarian algorithm with L2 distance costs. We then perform an as-rigid-as-possible (ARAP) [23] deformation of the tessellated surface towards the matched input points. ARAP is an iterative, detail-preserving method to deform a mesh so that selected vertices (pivots) achieve targets position, while promoting locally rigid transformations in one-ring neighborhoods (instead of arbitrary ones causing shearing/stretching). We use the boundary vertices of the patch as pivots so that they move close to their matched input points. Thus, we promote coverage of input points by the B-spline patches. After the deformation, the control points are re-estimated with least-squares [15].

Refinement of B-spline control points. After the above optimization, we again perform a maximal matching between the quad vertices and the input points of the segment. As a result, the input segment points acquire 2D parameter values in the patch's UV parameter space, which can be used to re-fit any other grid of control points [15]. In our case, we iteratively upsample the control point grid by a factor of 2 until a fitting tolerance, measured via Chamfer distance, is achieved. If the tolerance is satisfied by the initial control point grid, we can similarly downsample it iteratively. In our experiments, we set the fitting tolerance to 5×10^{-4}. In Fig. 5 we show the improvements from the post-processing step.

4 Training

To train the neural decomposition and fitting modules of our architecture, we use supervisory signals from a dataset of 3D shapes modeled through a combination of basic geometric primitives and B-splines. Below we describe the dataset, then we discuss the loss functions and the steps of our training procedure.

4.1 Dataset

The ABC dataset [9] provides a large source of 3D CAD models of mechanical objects whose file format stores surface patches and modeling operations that designers used to create them. Since our method is focused on predicting surface patches, and in particular B-spline patches, we selected models from this dataset

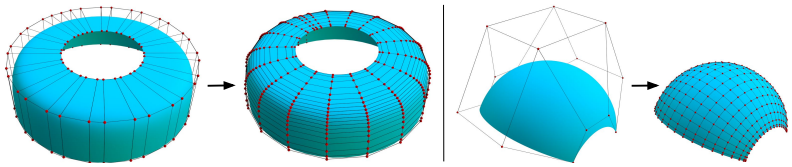

Fig. 3: **Standardization:** Examples of B-spline patches with a variable number of control points (shown in red), each standardized with 20×20 control points. Left: closed B-spline and Right: open B-spline. (Please zoom in.)

that contain at least one B-spline surface patch. As a result, we ended up with a dataset of 32K models (24K, 4K, 4K train, test, validation sets respectively). We call this ABCPARTSDATASET. All shapes are centered in the origin and scaled so they lie inside unit cube. To train SPLINENET, we also extract 32K closed and open B-spline surface patches each from ABC dataset and split them into 24K, 4K, 4K train, test, validation sets respectively. We call this SPLINEDATASET. We report the average number of different patch types in supplementary material.

Preprocessing. Based on the provided metadata in ABCPARTSDATASET, each shape can be rendered based on the collection of surface patches and primitives it contains (Figure 4). Since we assume that the inputs to our architecture are point clouds, we first sample each shape with 10K points randomly distributed on the shape surface. We also add noise in a uniform range $[-0.01, 0.01]$ along the normal direction. Normals are also perturbed with random noise in a uniform range of $[-3, 3]$ degrees from their original direction.

4.2 Loss functions

We now describe the different loss functions used to train our neural modules. The training procedure involving their combination is discussed in Section 4.3.

Embedding loss. To discover clusters of points that correspond well to surface patches, we use a metric learning approach. The point-wise representations $\mathbf{Z}$ produced by our decomposition module after mean-shift clustering are learned such that point pairs originating from the same surface patch are embedded close to each other to favor a cluster formation. In contrast, point pairs originating from different surface patches are pushed away from each other. Given a triplet of points (a, b, c), we use the triplet loss to learn the embeddings: c where τ the margin is set to 0.9. Given a triplet set $\mathcal{T}_\mathcal{S}$ sampled from each point set $\mathcal{S}$ from our dataset $\mathcal{D}$, the embedding objective sums the loss over triplets:

$$L_{emb} = \sum_{\mathcal{S} \in \mathcal{D}} \frac{1}{|\mathcal{T}_\mathcal{S}|} \sum_{(a,b,c) \in \mathcal{T}_\mathcal{S}} \ell_{emb}(a, b, c). \tag{4}$$

Segment classification loss. To promote correct segment classifications according to our supported types, we use the cross entropy loss: $L_{class} = -\sum_{i \in \mathcal{S}} \log(p_i^t)$

where p_t^i is the probability of the i^{th} point of shape $\mathcal{S}$ belonging to its ground truth type t, computed from our segment classification network.

Control point regression loss. This loss function is used to train SPLINENET. As discussed in Section 3.2, SPLINENET produces 20×20 control points per B-spline patch. We include a supervisory signal for this control point grid prediction. One issue is that B-spline patches have a variable number of control points in our dataset. Hence we reparametrize each patch by first sampling $M = 3600$ points and estimating a new 20×20 reparametrization using least-squares fitting [11,15], as seen in the Figure 3. In our experiments, we found that this standardization produces no practical loss in surface reconstructions in our dataset. Finally, our reconstruction loss should be invariant to flips or swaps of control points grid in u and v directions. Hence we define a loss that is invariant to such permutations:

$$L_{cp} = \sum_{\mathcal{S} \in \mathcal{D}} \frac{1}{|\mathcal{S}^{(b)}|} \sum_{s_k \in \mathcal{S}^{(b)}} \frac{1}{|\mathbf{C}_k|} \min_{\pi \in \Pi} ||\mathbf{C}_k - \pi(\hat{\mathbf{C}}_k)||^2 \tag{5}$$

where $\mathcal{S}^{(b)}$ is the set of B-spline patches from shape $\mathcal{S}$, $\mathbf{C}_k$ is the predicted control point grid for patch $\mathbf{s}_k$ ($|\mathbf{C}_k| = 400$ control points), $\pi(\hat{\mathbf{C}}_k)$ is permutations of the ground-truth control points from the set Π of 8 permutations for open and 160 permutations for closed B-spline.

Laplacian loss. This loss is also specific to B-Splines using SPLINENET. For each ground-truth B-spline patch, we uniformly sample ground truth surface, and measure the surface Laplacian capturing its second-order derivatives. We also uniformly sample the predicted patches and measure their Laplacians. We then establish Hungarian matching between sampled points in the ground-truth and predicted patches, and compare the Laplacians of the ground-truth points $\hat{\mathbf{r}}_m$ and corresponding predicted ones $\mathbf{r}_n$ to improve the agreement between their derivatives as follows:

$$L_{lap} = \sum_{\mathcal{S} \in \mathcal{D}} \frac{1}{|\mathcal{S}^{(b)}| \cdot M} \sum_{\mathbf{s}_k \in \mathcal{S}^{(b)}} \sum_{\mathbf{r}_n \in s_k} ||\mathcal{L}(\mathbf{r}_n) - \mathcal{L}(\hat{\mathbf{r}}_m)||^2 \tag{6}$$

where $\mathcal{L}(\cdot)$ is the Laplace operator on patch points, and $M = 1600$ point samples.

Patch distance loss. This loss is applied to both basic primitive and B-splines patches. Inspired by [12], the loss measures average distances between predicted primitive patch s_k and uniformly sampled points from the ground truth patch as:

$$L_{dist} = \sum_{\mathcal{S} \in \mathcal{D}} \frac{1}{K_{\mathcal{S}}} \sum_{k=1}^{K_{\mathcal{S}}} \frac{1}{M_{\hat{\mathbf{s}}_k}} \sum_{n \in \hat{\mathbf{s}}_k} D^2(\mathbf{r}_n, \mathbf{s}_k), \tag{7}$$

where $K_{\mathcal{S}}$ is the number of predicted patches for shape $\mathcal{S}$, $M_{\hat{\mathbf{s}}_k}$ is number of sampled points $\mathbf{r}_n$ from ground patch $\hat{\mathbf{s}}_k$, $D^2(\mathbf{r}_n, \mathbf{s}_k)$ is the squared distance from $\mathbf{r}_n$ to the predicted primitive patch surface $\mathbf{s}_k$. These distances can be computed analytically for basic primitives [12]. For B-splines, we use an approximation based on Chamfer distance between sample points.

4.3 Training procedure

One possibility for training is to start it from scratch using a combination of all losses. Based on our experiments, we found that breaking the training procedure into the following steps leads to faster convergence and to better minima:

- We first pre-train the networks of the decomposition module using ABCPARTSDATASET with the sum of embedding and classification losses: $L_{emb} + L_{class}$. Both losses are necessary for point cloud decomposition and classification.
- We then pre-train the SPLINENET using SPLINEDATASET for control point prediction exclusively on B-spline patches using $L_{cp} + L_{lap} + L_{dist}$. We note that we experimented training the B-spline patch prediction only with the patch distance loss L_{dist} but had worse performance. Using both the L_{cp} and L_{lap} loss yielded better predictions as shown in Table 2.
- We then jointly train the decomposition and fitting module end-to-end with all the losses. To allow backpropagation from the primitives and B-splines fitting to the embedding network, the mean shift clustering is implemented as a recurrent module (Equation 2). For efficiency, we use 5 mean-shift iterations during training. It is also important to note that during training, Equation 3 uses *soft* point-to-segment memberships, which enables backpropagation from the fitting module to the decomposition module and improves reconstructions. The soft memberships are computed based on the point embeddings $\{\mathbf{z}_i\}$ (after the mean-shift iterations) and cluster center embedding $\{\mathbf{z}_k\}$ as follows:

$$\mathbf{W}[i,k] = \frac{\exp(\mathbf{z}_k^T \mathbf{z}_i / \beta^2)}{\sum_{k'} \exp(\mathbf{z}_{k'}^T \mathbf{z}_i) / \beta^2)} \tag{8}$$

Please see supplementary material for more implementation details.

5 Experiments

Our experiments compare our approach to alternatives in three parts: (a) evaluation of the quality of segmentation and segment classification (Section 5.1), (b) evaluation of B-spline patch fitting, since it is a major contribution of our work (Section 5.2), and (c) evaluation of overall reconstruction quality (Section 5.3). We include evaluation metrics and results for each of the three parts next.

5.1 Segmentation and labeling evaluation

Evaluation metrics. We use the following metrics for evaluating the point cloud segmentation and segment labeling based on the test set of ABCPARTSDATASET:

- **Segmentation mean IOU** ("seg mIOU"): this metric measures the similarity of the predicted segments with ground truth segments. Given the ground-truth point-to-segment memberships $\hat{\mathbf{W}}$ for an input point cloud, and the predicted ones $\mathbf{W}$, we measure: $\frac{1}{K}\sum_{k=1}^{K} IOU(\hat{\mathbf{W}}[:,k], h(\mathbf{W}[:,k]))$
  where h represents a membership conversion into a one-hot vector, and K is the number of ground-truth segments.

Method	Input	seg iou	label iou	res (all)	res (geom)	res (spline)	P cover
NN	p	54.10	61.10	-	-	-	-
RANSAC	p+n	67.21	-	0.0220	0.0220	-	83.40
SPFN	p	47.38	68.92	0.0238	0.0270	0.0100	86.66
SPFN	p+n	69.01	79.94	0.0212	0.0240	0.0136	88.40
PARSENET	p	71.32	79.61	0.0150	0.0160	0.0090	87.00
PARSENET	p+n	81.20	87.50	0.0120	0.0123	0.0077	92.00
PARSENET + e2e	p+n	82.14	88.60	0.0118	0.0120	0.0076	92.30
PARSENET + e2e + opt	p+n	**82.14**	**88.60**	**0.0111**	**0.0120**	**0.0068**	**92.97**

Table 1: **Primitive fitting on** ABCPARTSDATASET**.** We compare PARSENET with nearest neighbor (NN), RANSAC [16], and SPFN [12]. We show results with points (p) and points and normals (p+n) as input. The last two rows shows our method with end-to-end training and post-process optimization. We report 'seg iou' and 'label iou' metric for segmentation task. We report the residual error (res) on all, geometric and spline primitives, and the coverage metric for fitting.

- **Segment labeling IOU** ("label mIOU"): this metric measures the classification accuracy of primitive type prediction averaged over segments: $\frac{1}{K}\sum_{k=1}^{K}\mathcal{I}\left[t_k = \hat{t}_k\right]$ where t_k and $\hat{t}_k$ is the predicted and ground truth primitive type respectively for k^{th} segment and $\mathcal{I}$ is an indicator function.

We use Hungarian matching to find correspondences between predicted segments and ground-truth segments.

Comparisons. We first compare our method with a nearest neighbor (NN) baseline: for each test shape, we find its most similar shape from the training set using Chamfer distance. Then for each point on the test shape, we transfer the labels and primitive type from its closest point in $\mathcal{R}^3$ on the retrieved shape.

We also compare against efficient RANSAC algorithm [16]. The algorithm only handles basic primitives (cylinder, cone, plane, sphere, and torus), and offers poor reconstruction of B-splines patches in our dataset. Efficient RANSAC requires per point normals, which we provide as the ground-truth normals. We run RANSAC 3 times and report the performance with best coverage.

We then compare against the supervised primitive fitting (SPFN) approach [12]. Their approach produces per point segment membership, and their network is trained to maximize relaxed IOU between predicted membership and ground truth membership, whereas our approach uses learned point embeddings and clustering with mean-shift clustering to extract segments. We train SPFN network using their provided code on our training set using their proposed losses. We note that we include B-splines patches in their supported types. We train their network in two input settings: (a) the network takes only point positions as input, (b) it takes point and normals as input. We train our PARSENET on our training set in the same two settings using our loss functions.

The performance of the above methods are shown in Table 1. The lack of B-spline fitting hampers the performance of RANSAC. The SPFN method with

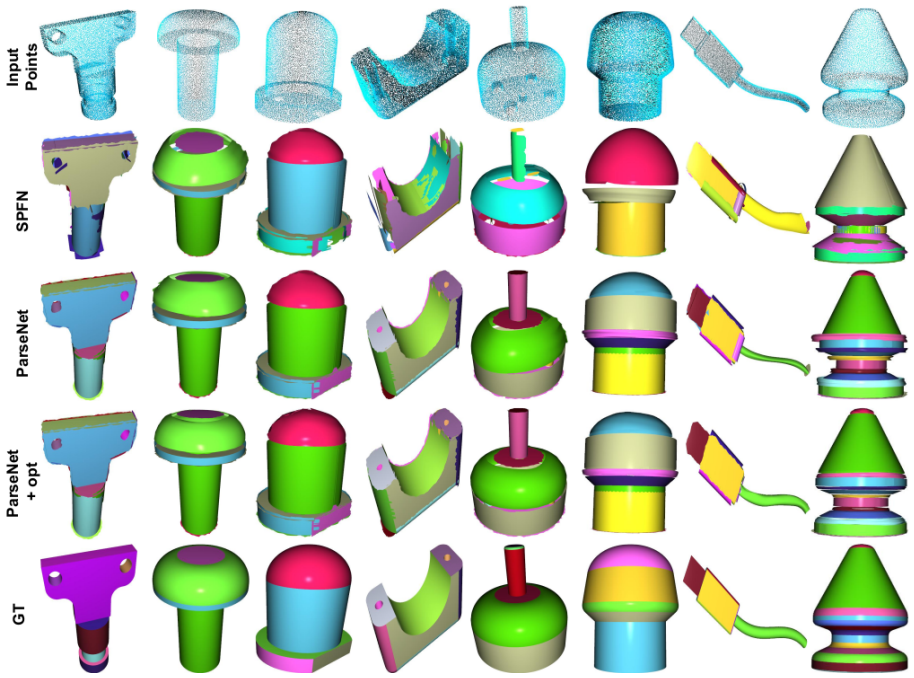

Fig. 4: Given the input point clouds with normals of the first row, we show surfaces produced by SPFN [12] (second row), ParSeNet without post-processing optimization (third row), and full ParSeNet including optimization (fourth row). The last row shows the ground-truth surfaces from our ABCPartsDataset.

points and normals as input performs better compared to using only points as input. Finally, ParSeNet with only points as input performs better than all other alternatives. We observe further gains when including point normals in the input. Training ParSeNet end-to-end gives 13.13% and 8.66% improvement in segmentation mIOU and label mIOU respectively over SPFN with points and normals as input. The better performance is also reflected in Figure 4, where our method reconstructs patches that correspond to more reasonable segmentations compared to other methods. In the supplementary material we evaluate methods on the TraceParts dataset [12], which contains only basic primitives (cylinder, cone, plane, sphere, torus). We outperform prior work also in this dataset.

5.2 B-Spline fitting evaluation

Evaluation metrics. We evaluate the quality of our predicted B-spline patches by computing the Chamfer distance between densely sampled points on the ground-truth B-spline patches and densely sampled points on predicted patches. Points are uniformly sampled based on the 2D parameter space of the patches. We use 2K samples. We use the test set of our SplineDataset for evaluation.

Loss				Open splines		Closed splines	
cp	dist	lap	opt	w/ ups	w/o ups	w/ ups	w/o ups
✓				2.04	2.00	5.04	3.93
✓	✓			1.96	2.00	4.9	3.60
✓	✓	✓		1.68	1.59	3.74	3.29
✓	✓	✓	✓	**0.92**	**0.87**	**0.63**	**0.81**

Table 2: **Ablation study for B-spline fitting.** The error is measured using Chamfer Distance (CD is scaled by 100). The acronyms "cp": control-points regression loss, "dist" means patch distance loss, and "lap" means Laplacian loss. We also include the effect of post-processing optimization "opt". We report performance with and without upsampling ("ups") for open and closed B-splines.

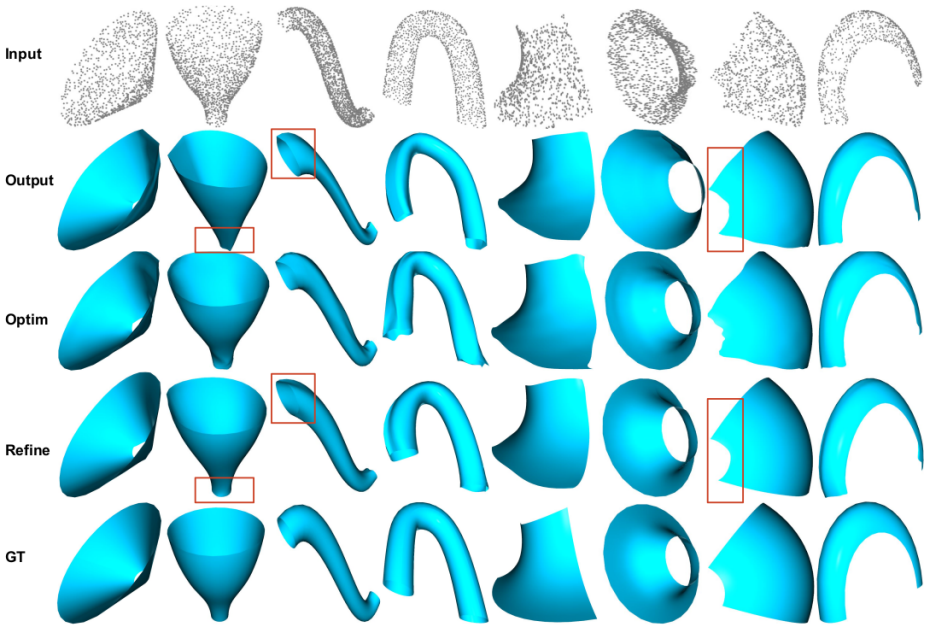

Fig. 5: **Qualitative evaluation of B-spline fitting.** From top to bottom: input point cloud, reconstructed surface by SplineNet, reconstructed surface by SplineNet with post-processing optimization, reconstruction by SplineNet with control point grid adjustment and finally ground truth surface. Effect of post process optimization is highlighted in red boxes.

Ablation study. We evaluate the training of SplineNet using various loss functions while giving 700 points per patch as input, in Table 2. All losses contribute to improvements in performance. Table 2 shows that upsampling is effective for closed splines. Figure 5 shows the effect of optimization to improve the alignment of patches and the adjustment of resolution in the control point grid. See supplementary material for more experiments on SplineNet's robustness.

5.3 Reconstruction evaluation

Evaluation metrics. Given a point cloud $\mathbf{P} = \{\mathbf{p}_i\}_{i=1}^{N}$, ground-truth patches $\{\cup_{k=1}^{K} \hat{\mathbf{s}}_k\}$ and predicted patches $\{\cup_{k=1}^{K} \mathbf{s}_k\}$ for a test shape in ABCPARTS-DATASET, we evaluate the patch-based surface reconstruction using the following:

- **Residual error** ("res") measures the average distance of input points from the predicted primitives following [12]: $L_{dist} = \sum_{k=1}^{K} \frac{1}{M_k} \sum_{n \in \hat{\mathbf{s}}_k} D(\mathbf{r}_n, \mathbf{s}_k)$ where K is the number of segments, M_k is number of sampled points $\mathbf{r}_n$ from ground patch $\hat{\mathbf{s}}_k$, $D(\mathbf{r}_n, \mathbf{s}_k)$ is the distance of $\mathbf{r}_n$ from predicted primitive patch $\mathbf{s}_k$.
- **P-coverage** ("P-cover") measures the coverage of predicted surface by the input surface also following [12]: $\frac{1}{N} \sum_{i=1}^{N} \mathbf{I}\left[\min_{k=1}^{K} D(\mathbf{p}_i, \mathbf{s}_k) < \epsilon\right]$ $(\epsilon = 0.01)$.

We note that we use the matched segments after applying Hungarian matching algorithm, as in Section 5.1, to compute these metrics.

Comparisons. We report the performance of RANSAC for geometric primitive fitting tasks. Note that RANSAC produces a set of geometric primitives, along with their primitive type and parameters, which we use to compute the above metrics. Here we compare with the SPFN network [12] trained on our dataset using their proposed loss functions. We augment their per point primitive type prediction to also include open/closed B-spline type. Then for classified segments as B-splines, we use our SPLINENET to fit B-splines. For segments classified as geometric primitives, we use their geometric primitive fitting algorithm.

Results. Table 1 reports the performance of our method, SPFN and RANSAC. The residual error and P-coverage follows the trend of segmentation metrics. Interestingly, our method outperforms SPFN even for geometric primitive predictions (even without considering B-splines and our adaptation). Using points and normals, along with joint end-to-end training, and post-processing optimization offers the best performance for our method by giving 47.64% and 50% reduction in relative error in comparison to SPFN and RANSAC respectively.

6 Conclusion

We presented a method to reconstruct point clouds by predicting geometric primitives and surface patches common in CAD design. Our method effectively marries 3D deep learning with CAD modeling practices. Our architecture predictions are editable and interpretable. Modelers can refine our results based on standard CAD modeling operations. In terms of limitations, our method often makes mistakes for small parts, mainly because clustering merges them with bigger patches. In high-curvature areas, due to sparse sampling, PARSENET may produce more segments than ground-truth. Producing seamless boundaries is still a challenge due to noise and sparsity in our point sets. Generating training point clouds simulating realistic scan noise is another important future direction.

Acknowledgements. This research is funded in part by NSF (#1617333, #1749833) and Adobe. Our experiments were performed in the UMass GPU cluster funded by the MassTech Collaborative. We thank Matheus Gadelha for helpful discussions.

References

1. Ahmed, E., Saint, A., Shabayek, A.E.R., Cherenkova, K., Das, R., Gusev, G., Aouada, D., Ottersten, B.E.: Deep learning advances on different 3D data representations: A survey. Computing Research Repository (CoRR) **abs/1808.01462** (2019)
2. Cuturi, M., Teboul, O., Vert, J.P.: Differentiable ranking and sorting using optimal transport. In: Advances in Neural Information Processing Systems 32 (2019)
3. Eck, M., Hoppe, H.: Automatic reconstruction of B-spline surfaces of arbitrary topological type. In: SIGGRAPH (1996)
4. Farin, G.: Curves and Surfaces for CAGD. Morgan Kaufmann, 5th edn. (2002)
5. Foley, J.D., van Dam, A., Feiner, S.K., Hughes, J.F.: Computer Graphics: Principles and Practice. Addison-Wesley Longman Publishing Co., Inc., USA, 2nd edn. (1990)
6. Gao, J., Tang, C., Ganapathi-Subramanian, V., Huang, J., Su, H., Guibas, L.J.: DeepSpline: Data-driven reconstruction of parametric curves and surfaces. Computing Research Repository (CoRR) **abs/1901.03781** (2019)
7. Hoppe, H., DeRose, T., Duchamp, T., Halstead, M., Jin, H., McDonald, J., Schweitzer, J., Stuetzle, W.: Piecewise smooth surface reconstruction. In: Proc. SIGGRAPH (1994)
8. Kaiser, A., Ybanez Zepeda, J.A., Boubekeur, T.: A survey of simple geometric primitives detection methods for captured 3D data. Computer Graphics Forum **38**(1), 167–196 (2019)
9. Koch, S., Matveev, A., Jiang, Z., Williams, F., Artemov, A., Burnaev, E., Alexa, M., Zorin, D., Panozzo, D.: ABC: A big cad model dataset for geometric deep learning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (June 2019)
10. Kong, S., Fowlkes, C.: Recurrent pixel embedding for instance grouping. In: 2018 Conference on Computer Vision and Pattern Recognition (CVPR) (2018)
11. Krishnamurthy, V., Levoy, M.: Fitting smooth surfaces to dense polygon meshes. In: SIGGRAPH (1996)
12. Li, L., Sung, M., Dubrovina, A., Yi, L., Guibas, L.J.: Supervised fitting of geometric primitives to 3d point clouds. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (June 2019)
13. Paschalidou, D., Ulusoy, A.O., Geiger, A.: Superquadrics revisited: Learning 3d shape parsing beyond cuboids. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 10336–10345 (2019)
14. Paschalidou, D., Gool, L.V., Geiger, A.: Learning unsupervised hierarchical part decomposition of 3d objects from a single rgb image. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (June 2020)
15. Piegl, L., Tiller, W.: The NURBS Book. Springer-Verlag, Berlin, Heidelberg, 2nd edn. (1997)
16. Schnabel, R., Wahl, R., Klein, R.: Efficient RANSAC for point-cloud shape detection. Computer Graphics Forum **26**, 214–226 (06 2007)
17. Schneider, P.J., Eberly, D.: Geometric Tools for Computer Graphics. Elsevier Science Inc., USA (2002)
18. Schulman, J., Heess, N., Weber, T., Abbeel, P.: Gradient estimation using stochastic computation graphs. In: Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2. p. 3528–3536. NIPS'15, MIT Press, Cambridge, MA, USA (2015)

19. Sharma, G., Goyal, R., Liu, D., Kalogerakis, E., Maji, S.: Csgnet: Neural shape parser for constructive solid geometry. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) pp. 5515–5523 (2018)
20. Sharma, G., Goyal, R., Liu, D., Kalogerakis, E., Maji, S.: Neural shape parsers for constructive solid geometry. Computing Research Repository (CoRR) **abs/1912.11393** (2019), http://arxiv.org/abs/1912.11393
21. Silverman, B.W.: Density Estimation for Statistics and Data Analysis. Chapman & Hall, London (1986)
22. Smirnov, D., Fisher, M., Kim, V.G., Zhang, R., Solomon, J.: Deep parametric shape predictions using distance fields. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (June 2020)
23. Sorkine, O., Alexa, M.: As-rigid-as-possible surface modeling. In: Symposium on Geometry Processing. pp. 109–116 (2007)
24. Sun, C., Zou, Q., Tong, X., Liu, Y.: Learning adaptive hierarchical cuboid abstractions of 3D shape collections. ACM Transactions on Graphics (SIGGRAPH Asia) **38**(6) (2019)
25. Tian, Y., Luo, A., Sun, X., Ellis, K., Freeman, W.T., Tenenbaum, J.B., Wu, J.: Learning to infer and execute 3d shape programs. In: International Conference on Learning Representations (2019)
26. Tulsiani, S., Su, H., Guibas, L.J., Efros, A.A., Malik, J.: Learning shape abstractions by assembling volumetric primitives. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (July 2017)
27. Wang, Y., Sun, Y., Liu, Z., Sarma, S.E., Bronstein, M.M., Solomon, J.M.: Dynamic graph cnn for learning on point clouds. ACM Transactions on Graphics **38**(5) (Oct 2019)
28. Yumer, M.E., Kara, L.B.: Surface creation on unstructured point sets using neural networks. Computer-Aided Design **44**(7), 644 – 656 (2012)
29. Zou, C., Yumer, E., Yang, J., Ceylan, D., Hoiem, D.: 3d-prnn: Generating shape primitives with recurrent neural networks. In: The IEEE International Conference on Computer Vision (ICCV) (2017)

7 Supplementary Material

In our Supplementary Material, we:

- provide background on B-spline patches;
- provide further details about our dataset, architectures and implementation;
- evaluate the robustness of SplineNet as a function of point density;
- evaluate our approach for reconstruction on the ABCPartsDataset;
- show more visualizations of our results; and
- evaluate the performance of our approach on the TraceParts dataset [12].

7.1 Background on B-spline patches.

A B-spline patch is a smoothly curved, bounded, parametric surface, whose shape is defined by a sparse grid of control points $\mathbf{C} = \{\mathbf{c}_{p,q}\}$. The surface point with parameters $(u, v) \in [u_{\min}, u_{\max}] \times [v_{\min}, v_{\max}]$ is given by:

$$\mathbf{s}(u, v) = \sum_{p=1}^{P} \sum_{q=1}^{Q} b_p(u) b_q(v) \mathbf{c}_{p,q} \tag{9}$$

where $b_p(u)$ and $b_q(v)$ are polynomial B-spline *basis functions* [4].

To determine how the control points affect the B-spline, a sequence of parameter values, or *knot vector*, is used to divide the range of each parameter into intervals or *knot spans*. Whenever the parameter value enters a new knot span, a new row (or column) of control points and associated basis functions become active. A common knot setting repeats the first and last ones multiple times (specifically 4 for cubic B-splines) while keeping the interior knots uniformly spaced, so that the patch interpolates the corners of the control point grid. A closed surface is generated by matching the control points on opposite edges of the grid. There are various generalizations of B-splines *e.g.*, with rational basis functions or non-uniform knots. We focus on predicting cubic B-splines (open or closed) with uniform interior knots, which are quite common in CAD [4, 5, 15, 17].

7.2 Dataset

The ABCPartsDataset is a subset of the ABC dataset obtained by first selecting models that contain at least one B-spline surface patch. To avoid over-segmented shapes, we retain those with up to 50 surface patches. This results in a total of $32k$ shapes, which we further split into training ($24k$), validation ($4k$), and test ($4k$) subsets. Figure 6 shows the distribution of number and type of surface patches in the dataset.

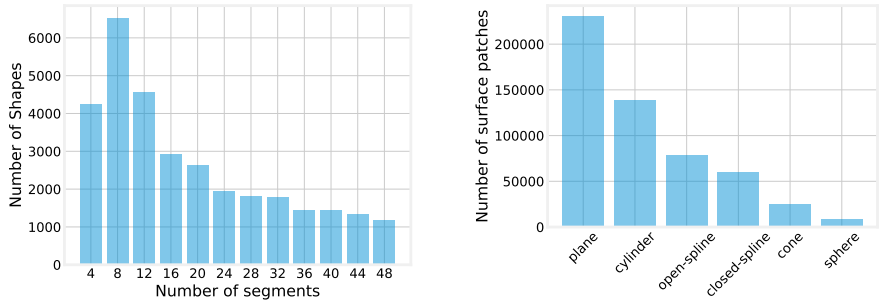

Fig. 6: **Histogram of surface patches in** ABCPartsDataset. Left: shows histogram of number of segments and Right: shows histogram of primitive types.

7.3 Implementation Details of ParSeNet

Architecture details. Our decomposition module is based on a dynamic edge convolution network [27]. The network takes points as input (and optionally normals) and outputs a per point embedding $Y \in R^{N\times 128}$ and primitive type $T \in R^{N\times 6}$. The layers of our network are listed in Table 3. The edge convolution layer (EdgeConv) takes as input a per-point feature representation $f \in \mathbb{R}^{N\times D}$, constructs a kNN graph based on this feature space (we choose $k = 80$ neighbors), then forms another feature representation $h \in \mathbb{R}^{N\times k\times 2D}$, where $h_{i,j} = [f_i, f_i - f_j]$, and i,j are neighboring points. This encodes both unary and pairwise point features, which are further transformed by a MLP ($D \to D'$), Group normalization and LeakyReLU (slope=0.2) layers. This results in a new feature representation: $h' \in \mathbb{R}^{N\times k\times D'}$. Features from neighboring points are max-pooled to obtain a per point feature $f' \in \mathbb{R}^{N\times D'}$. We express this layer which takes features $f \in \mathbb{R}^{N\times D}$ and returns features $f' \in \mathbb{R}^{N\times D'}$ as EdgeConv(f, D, D'). Group normalization in EdgeConv layer allows the use of smaller batch size during training. Please refer to [27] for more details on edge convolution network.
SplineNet is also implemented using a dynamic graph CNN. The network takes points as input and outputs a grid of spline control points that best approximates the input point cloud. The architecture of SplineNet is described in Table 4. Note that the EdgeConv layer in this network uses batch normalization instead of group normalization.

Training details. We use the Adam optimizer for training with learning rate 10^{-2} and reducing it by the factor of two when the validation performance saturates. For the EdgeConv layers of the decomposition module, we use 100 nearest neighbors, and 10 for the ones in SplineNet. For pre-training SplineNet on SplineDataset, we randomly sample $2k$ points from the B-spline patches. Since ABC shapes are arbitrarily oriented, we perform PCA on them and align the direction corresponding to the smallest eigenvalue to the $+x$ axis. This procedure does not guarantee alignment, but helps since it reduces the orientation variability in the dataset. For pre-training the decomposition module and SplineNet we

Index	Layer	out
1	Input	N × 3
2	EdgeConv(out(1), 3, 64)	N × 64
3	EdgeConv(out(2), 64, 64)	N × 64
4	EdgeConv(out(3), 64, 128)	N × 128
5	CAT(out(2), out(3), out(4))	N × (256)
6	RELU(GN(FC(out(5), 1024)))	N × 1024
7	MP(out(6), N, 1)	1024
8	Repeat(out(7), N)	N × 1024
9	CAT(out(8), out(5))	N × 1280
10	RELU(GN(FC(out(9), 512)))	N × 512
11	RELU(GN(FC(out(10), 256)))	N × 256
12	RELU(GN(FC(out(11), 256)))	N × 256
13	**Embedding**=Norm(FC(out(12), 128))	N × 128
14	RELU(GN(FC(out(11), 256))	N × 256
15	**Primitive-Type**=Softmax(FC(out(14), 6))	N × 6

Table 3: **Architecture of the Decomposition Module.** EdgeConv: edge convolution, GN: group normalization, RELU: rectified linear unit, FC: fully connected layer, CAT: concatenate tensors along the second dimension, MP: max-pooling along the first dimension, Norm: normalizing the tensor to unit Euclidean length across the second dimension.

augment the training shapes represented as points by using random jitters, scaling, rotation and point density.

Back propagation through mean-shift clustering. The W matrix is constructed by first applying non-max suppression (NMS) on the output of mean shift clustering, which gives us indices of K cluster centers. NMS is done externally *i.e.* outside our computational graph. We use these indices and Eq. 9 to compute the W matrix. The derivatives of NMS w.r.t point embeddings are zero or undefined (i.e. non-differentiable). Thus, we remove NMS from the computational graph and back-propagate the gradients through a partial computation graph, which is differentiable. This can be seen as a straight-through estimator [18]. A similar approach is used in back-propagating gradients through Hungarian Matching in [12]. Our experiments in Table 1 shows that this approach for end-to-end training is effective. Constructing a fixed size matrix W will result in redundant/unused columns because different shapes have different numbers of clusters. Possible improvements may lie in a continuous relaxation of clustering similar to differentiable sorting and ranking [2], however that is out of scope for our work.

7.4 Robustness analysis of SplineNet

Here we evaluate the performance of SPLINENET as a function of the point sampling density. As seen in Figure 7, the performance of SPLINENET is low

Index	Layer	Output
1	Input	N × 3
2	EdgeConv(out(1),3, 128)	N × 128
3	EdgeConv(out(2),128, 128)	N × 128
4	EdgeConv(out(3),128, 256)	N × 256
5	EdgeConv(out(4),256, 512)	N × 512
6	CAT(out(2), out(3), out(4), out(5)))	N × (1152)
7	RELU(BN(FC(out(6), 1024))	N × 1024
8	MP(out(7), N, 1)	1024
9	RELU(BN(FC(out(8), 1024))	1024
10	RELU(BN(FC(out(9), 1024))	1024
11	Tanh(FC(out(10), 1200))	1200
12	**Control Points** = Reshape(out(11), (20, 20, 3))	20 × 20 × 3

Table 4: **Architecture of** SplineNet**.** EdgeConv: edge convolution layer, BN: batch noramlization, RELU: rectified linear unit, FC: fully connected layer, CAT: concatenate tensors along second dimension, and MP: max-pooling across first dimension

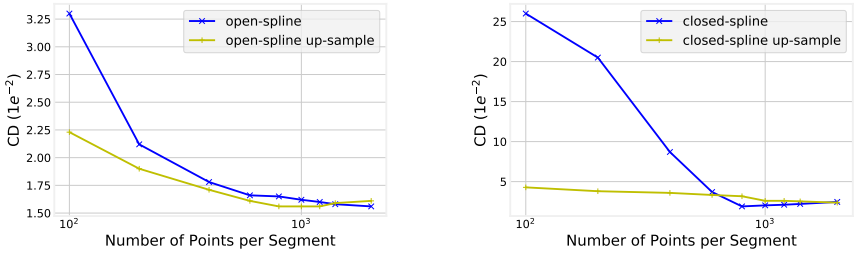

Fig. 7: **Robustness analysis of** SplineNet**.** Left: open B-spline and Right: closed B-spline. Performance degrades for sparse inputs (blue curve). Nearest neighbor up-sampling of the input point cloud to $1.6K$ points reduces error for sparser inputs (yellow curve). The horizontal axis is in log scale. The error is measured using Chamfer distance (CD).

when the point density is small (100 points per surface patch). SplineNet is based on graph edge convolutions [27], which are affected by the underlying sampling density of the network. However, upsampling points using a nearest neighbor interpolation leads to a significantly better performance.

7.5 Evaluation of Reconstruction using Chamfer Distance

Here we evaluate the performance of ParSeNet and other baselines for the task of reconstruction using Chamfer distance on ABCPartsDataset. Chamfer

Method	Input	p cover (1×10^{-4})	s cover (1×10^{-4})	CD (1×10^{-4})
NN	p	10.10	12.30	11.20
RANSAC	p+n	7.87	17.90	12.90
SPFN	p	7.17	13.40	10.30
SPFN	p+n	6.98	13.30	10.12
ParSeNet	p	6.07	12.40	9.26
ParSeNet	p+n	4.77	11.60	8.20
ParSeNet + e2e + opt	p+n	**2.45**	**10.60**	**6.51**

Table 5: **Reconstruction error measured using Chamfer distance on** ABCPartsDataset**.** 'e2e': end-to-end training of ParSeNet and 'opt': post-process optimization applied to B-spline surface patches.

distance between reconstructed points P and input points $\hat{P}$ is defined as:

$$p_{cover} = \frac{1}{|P|} \sum_{i \in P} \min_{j \in \hat{P}} \|i - j\|_2 \,,$$

$$s_{cover} = \frac{1}{|\hat{P}|} \sum_{i \in \hat{P}} \min_{j \in P} \|i - j\|_2 \,,$$

$$CD = \frac{1}{2}(p_{cover} + s_{cover}).$$

Here $|P|$ and $|\hat{P}|$ denote the cardinality of P and $\hat{P}$ respectively. We randomly sample $10k$ points each on the predicted and ground truth surface for the evaluation of all methods. Each predicted surface patch is also trimmed to define its boundary using bit-mapping with epsilon 0.1 [16]. To evaluate this metric, we use all predicted surface patches instead of the *matched* surface patches that is used in Section 5.3.

Results are shown in Table 5. Evaluation using Chamfer distance follows the same trend of residual error shown in Table 1. ParSeNet and SPFN with points as input performs better than NN and RANSAC. ParSeNet and SPFN with points along with normals as input performs better than with just points as input. By training ParSeNet end-to-end and also using post-process optimization results in the best performance. Our full ParSeNet gives 35.67% and 49.53% reduction in relative error in comparison to SPFN and RANSAC respectively. We show more visualizations of surfaces reconstructed by ParSeNet in Figure 8.

7.6 Evaluation on TraceParts Dataset

Here we evaluate the performance of ParSeNet on the TraceParts dataset, and compare it with SPFN. Note that the input points are normalized to lie inside a unit cube. Points sampled from the shapes in TraceParts [12] have a fraction of points not assigned to any cluster. To make this dataset compatible with our

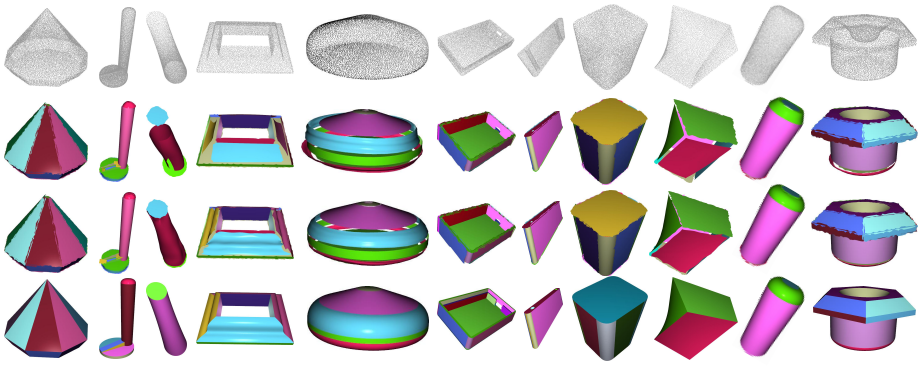

Fig. 8: Given the input point clouds with normals in the first row, we show surfaces produced by ParSeNet without post-processing optimization (second row), and full ParSeNet including optimization (third row). The last row shows the ground-truth surfaces from our ABCPartsDataset.

evaluation approach, each unassigned point is merged to its closest cluster. This results in evaluation score to differ from the reported score in their paper [12].

First we create a nearest neighbor (NN) baseline as shown in the Section 5.3. In this, we first scale both training and testing shape an-isotropically such that each dimension has unit length. Then for each test shape, we find its most similar shape from the training set using Chamfer Distance. Then for each point on the test shape, we transfer the labels and primitive type from its closest point in $\mathcal{R}^3$ on the retrieved shape. We train ParSeNet on the training set of TraceParts using the losses proposed in the Section 4.2 and we also train SPFN using their proposed losses. All results are reported on the test set of TraceParts.

Results are shown in the Table 6. The NN approach achieves a high segmentation mIOU of 81.92% and primitive type mIOU of 95%. Figure 9 shows the NN results for a random set of shapes in the test set. It seems that the test and training sets often contain duplicate or near-duplicate shapes in the TraceParts dataset. Thus the performance of the NN can be attributed to the lack of shape diversity in this dataset. In comparison, our dataset is diverse, both in terms of shape variety and primitive types, and the NN baseline achieve much lower performance with segmentation mIOU of 54.10% and primitive type mIOU of 61.10%.

We further compare our ParSeNet with SPFN with just points as input. ParSeNet achieves 79.91% seg mIOU compared to 76.4% in SPFN. ParSeNet achieves 97.39% label mIOU compared to 95.18% in SPFN. We also perform better when both points and normals are used as input to ParSeNet and SPFN.

Finally, we compare reconstruction performance in the Table 7. With just points as input to the network, ParSeNet reduces the relative residual error by 9.35% with respect to SPFN. With both points and normals as input ParSeNet reduces relative residual error by 15.17% with respect to SPFN.

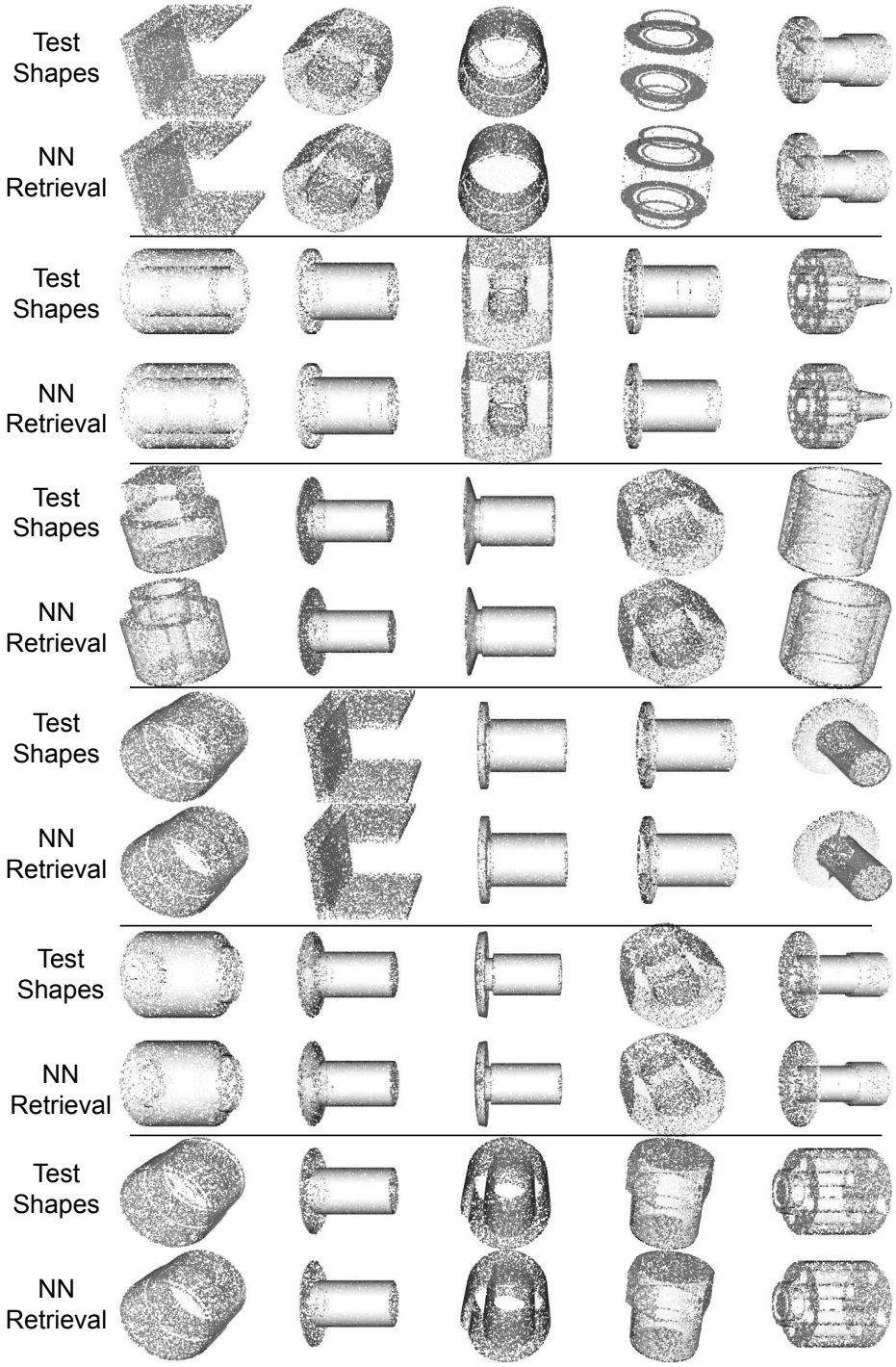

Fig. 9: **Nearest neighbor retrieval on the TracePart dataset** We randomly select 30 shapes from the test set of TraceParts dataset and show the NN retrieval, which reveals high training and testing set overlap. Shapes are an-isotropically scaled to unit length in each dimension. This is further validated quantitatively in Table 6.

Method	Input	seg mIOU	label mIOU
NN	p	81.92	95.00
SPFN	p	76.4	95.18
SPFN	p + n	88.05	98.10
ParseNet	p	79.91	97.39
ParseNet	p + n	**88.57**	**98.26**

Table 6: **Segmentation results on the TraceParts dataset.** We report segmentation and primitive type prediction performance of various methods.

Method	Input	res	P cover
NN	p	0.0138	91.90
SPFN	p	0.0139	91.70
SPFN	p + n	0.0112	**92.94**
ParseNet	p	0.0126	90.90
ParseNet	p + n	**0.0095**	92.72

Table 7: **Reconstruction results on the TraceParts dataset.** We report residual loss and P cover metrics for various methods.