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

1 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 [4].

1.1 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:

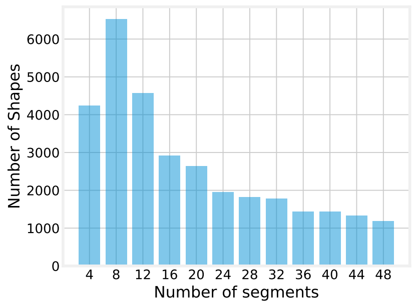
$$\mathbf{s}(u, v) = \sum_{p=1}^{P} \sum_{q=1}^{Q} b_p(u) b_q(v) \mathbf{c}_{p,q} \tag{1}$$

where $b_p(u)$ and $b_q(v)$ are polynomial B-spline *basis functions* [2].

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 [2,3,5,7].

1.2 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 1 shows the distribution of number and type of surface patches in the dataset.



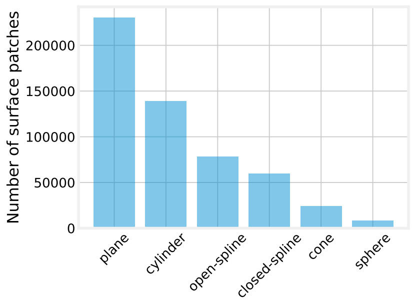


Fig. 1: **Histogram of surface patches in** ABCPartsDataset. Left: shows histogram of number of segments and Right: shows histogram of primitive types.

1.3 Implementation Details of ParSeNet

Architecture details. Our decomposition module is based on a dynamic edge convolution network [9]. 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 1. 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 [9] 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 2. Note that the EdgeConv layer in this network uses batch normalization instead of group normalization.

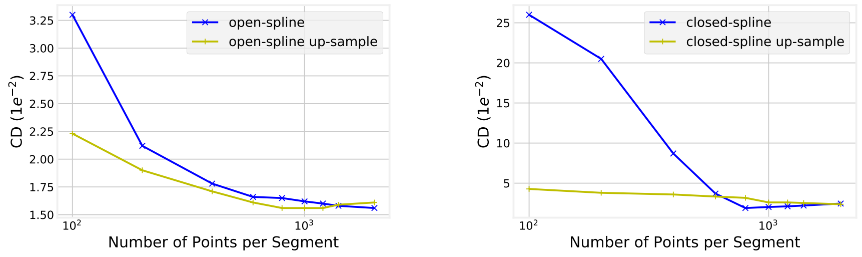


Fig. 2: **Robustness analysis of SplineNet.** Left: open B-spline and Right: closed B-spline. Performance degrades for sparse inputs (blue curve). Nearest neighbor up-sampling of the input point cloud to $1.6K$ points reduces error for sparser inputs (yellow curve). The horizontal axis is in log scale. The error is measured using Chamfer distance (CD).

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 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 [8]. A similar approach is used in back-propagating gradients through Hungarian Matching in [4]. 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 [1], however that is out of scope for our work.

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 1: **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.

1.4 Robustness analysis of SplineNet

Here we evaluate the performance of SplineNet as a function of the point sampling density. As seen in Figure 2, the performance of SplineNet is low when the point density is small (100 points per surface patch). SplineNet is based on graph edge convolutions [9], 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.

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

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

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

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

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 2: **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

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 [6]. 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 3. 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 3.

1.6 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 [4] have a fraction of points not assigned to any cluster. To make this dataset compatible with our 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 [4].

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

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 3: **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.

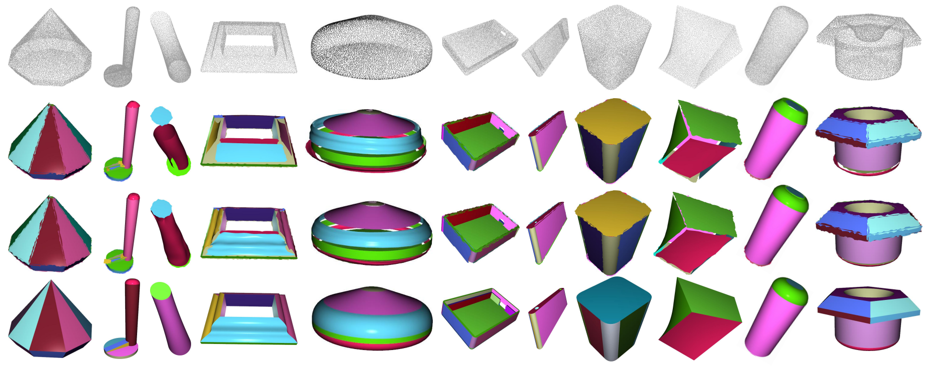


Fig. 3: Given the input point clouds with normals in the first row, we show surfaces produced by ParSeNet without post-processing optimization (second row), and full ParSeNet including optimization (third row). The last row shows the ground-truth surfaces from our ABCPartsDataset.

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 4. The NN approach achieves a high segmentation mIOU of 81.92% and primitive type mIOU of 95%. Figure 4 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 5. 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.

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 4: **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 5: **Reconstruction results on the TraceParts dataset.** We report residual loss and P cover metrics for various methods.

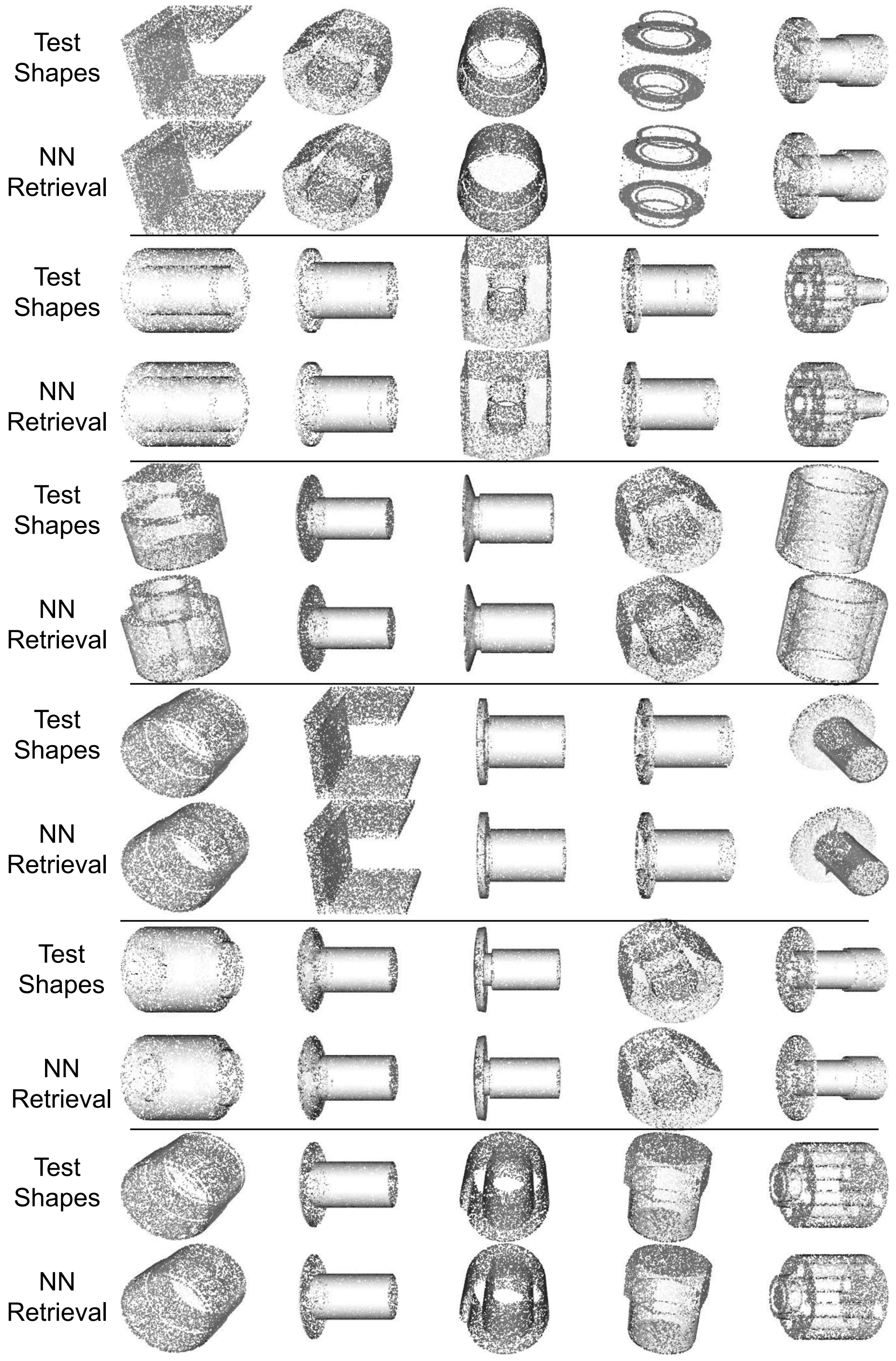


Fig. 4: **Nearest neighbor retrieval on the TracePart dataset** We randomly select 30 shapes from the test set of TraceParts dataset and show the NN retrieval, which reveals high training and testing set overlap. Shapes are an-isotropically scaled to unit length in each dimension. This is further validated quantitatively in Table 4.

References

1. Cuturi, M., Teboul, O., Vert, J.P.: Differentiable ranking and sorting using optimal transport. In: Advances in Neural Information Processing Systems 32, pp. 6861–6871. Curran Associates, Inc. (2019), http://papers.nips.cc/paper/8910-differentiable-ranking-and-sorting-using-optimal-transport.pdf
2. Farin, G.: Curves and Surfaces for CAGD. Morgan Kaufmann, 5th edn. (2002)
3. 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)
4. 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)
5. Piegl, L., Tiller, W.: The NURBS Book. Springer-Verlag, Berlin, Heidelberg, 2nd edn. (1997)
6. Schnabel, R., Wahl, R., Klein, R.: Efficient RANSAC for point-cloud shape detection. Computer Graphics Forum **26**, 214–226 (06 2007)
7. Schneider, P.J., Eberly, D.: Geometric Tools for Computer Graphics. Elsevier Science Inc., USA (2002)
8. Schulman, J., Heess, N., Weber, T., Abbeel, P.: Gradient estimation using stochastic computation graphs. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems 28, pp. 3528–3536. Curran Associates, Inc. (2015), http://papers.nips.cc/paper/5899-gradient-estimation-using-stochastic-computation-graphs.pdf
9. 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)