BPNet: Bézier Primitive Segmentation on 3D Point Clouds

Rao Fu[1,2], **Cheng Wen**[3], **Qian Li**[1,*], **Xiao Xiao**[4] and **Pierre Alliez**[1]
[1]Inria, France
[2]Geometry Factory, France
[3]The University of Sydney, Australia
[4]Shanghai Jiao Tong University, P. R. China
{rao.fu, qian.li, pierre.alliez}@inria.fr, cwen6671@uni.sydney.edu.au, xiao.xiao@sjtu.edu.cn

Abstract

This paper proposes BPNet, a novel end-to-end deep learning framework to learn Bézier primitive segmentation on 3D point clouds. The existing works treat different primitive types separately, thus limiting them to finite shape categories. To address this issue, we seek a generalized primitive segmentation on point clouds. Taking inspiration from Bézier decomposition on NURBS models, we transfer it to guide point cloud segmentation casting off primitive types. A joint optimization framework is proposed to learn Bézier primitive segmentation and geometric fitting simultaneously on a cascaded architecture. Specifically, we introduce a soft voting regularizer to improve primitive segmentation and propose an auto-weight embedding module to cluster point features, making the network more robust and generic. We also introduce a reconstruction module where we successfully process multiple CAD models with different primitives simultaneously. We conducted extensive experiments on the synthetic ABC dataset and real-scan datasets to validate and compare our approach with different baseline methods. Experiments show superior performance over previous work in terms of segmentation, with a substantially faster inference speed.

1 Introduction

Structuring and abstracting 3D point clouds via segmentation is a prerequisite for various computer vision and 3D modeling applications. Many approaches have been proposed for semantic segmentation, but the finite set of semantic classes limits their applicability. 3D instance-level segmentation and shape detection are much more demanding, while this literature lags far behind its semantic segmentation counterpart. Finding a generalized way to decompose point clouds is essential. For example, man-made objects can be decomposed into canonical primitives such as planes, spheres, and cylinders, which are helpful for visualization and editing. However, the limited types of canonical primitives are insufficient to describe objects' geometry in real-world tasks. We are looking for a generalized way of decomposing point clouds. The task of decomposing point clouds into different geometric primitives with corresponding parameters is referred to as *parametric primitive segmentation*. Parametric primitive segmentation is more reasonable than semantic instance segmentation for individual 3D objects, which unifies the 3D objects in the parametric space instead of forming artificially defined parts. However, the task is quite challenging as 1) there is no exhaustive repertoire of canonical geometric primitives, 2) the number of primitives and points belonging to that primitive may significantly vary, and 3) points assigned to the same primitive should belong to the same type of primitive.

Inspired by the fact that Bézier decomposition, where NURBS models can be divided into canonical geometric primitives (plane, sphere, cone, cylinder, etc.) and parametric surfaces into rational Bézier patches, we propose to learn Bézier decomposition on 3D point clouds. We focus on segmenting point clouds sampled from individual objects, such as CAD models. Departing from previous primitive segmentation, we generalize different primitive types to Bézier primitives, making them suitable for end-to-end and batch training. To the best of our knowledge, our method is the only work to learn Bézier decomposition on point clouds. To summarize our contributions:

1. We introduce a novel soft voting regularizer for the relaxed intersection over union (IOU) loss, improving our primitive segmentation results.
2. We design a new auto-weight embedding module to cluster point features which is free of iterations, making the network robust to real-scan data and work for axis-symmetric free-form point clouds.
3. We propose an innovative reconstruction module where we succeed in using a generalized formula to evaluate points on different primitive types, enabling our training process to be fully differential and compatible with batch operations.
4. Experiments demonstrate that our method works on the free-form point clouds and real-scan data even if we only train our model on the ABC dataset. Furthermore, we present one application of Bézier primitive segmentation to reconstruct the full Bézier model while preserving the sharp features. The code is available at: https://github.com/bizerfr/BPNet.

*Corresponding author

2 Related Work

Bézier primitive segmentation involves parametric fitting, instance segmentation, and multi-task learning. We now provide a brief review of these related research areas.

Primitive segmentation. Primitive segmentation refers to the search and approximation of geometric primitives from point clouds. Primitives can be canonical geometric primitives, such as planes or spheres, or parametric surface patches, such as Bézier, BSpline, or NURBS. We can classify primitive segmentation methods into two lines of approaches: geometric optimization and machine learning. Popular geometric optimization-based methods include RANSAC [Fischler and Bolles, 1981; Schnabel *et al.*, 2007], region growing [Marshall *et al.*, 2001] and Hough transforms [Rabbani *et al.*, 2007]. We refer to [Kaiser *et al.*, 2019] for a comprehensive survey. One limitation of geometric optimization-based methods is that they require strong prior knowledge and are hence sensitive to parameters. In order to alleviate this problem, recent approaches utilize neural networks for learning specific classes of primitives such as cuboids [Zou *et al.*, 2017; Tulsiani *et al.*, 2017]. The SPFN supervised learning approach [Li *et al.*, 2019] detects a wider repertoire of primitives such as planes, spheres, cylinders, and cones. Apart from the canonical primitives handled by SPFN, ParSeNet [Sharma *et al.*, 2020] and HPNet [Yan and Yang, 2021] also detect open or closed BSpline surface patches. Nevertheless, different types of primitives are treated separately with insufficient genericity. This makes them unsuitable for batch operations, thus suffering long inference times. Deep learning-based methods are less sensitive to parameters but often support a limited repertoire of primitives. Our work extends SPFN, ParSeNet, and HPNet with more general Bézier patches.

Instance segmentation. Instance segmentation is more challenging than semantic segmentation as the number of instances is not known a priori. Points assigned to the same instance should fall into the same semantic class. We distinguish between two types of methods: proposal-based [Yi *et al.*, 2019; Yang *et al.*, 2019; Engelmann *et al.*, 2020] and proposal-free methods [Wang *et al.*, 2018; Jiang *et al.*, 2020; Huang *et al.*, 2021]. On the one hand, proposal-based methods utilize an object-detection module and usually learn an instance mask for prediction. On the other hand, proposal-free methods tackle the problem as a clustering step after semantic segmentation. We refer to a recent comprehensive survey [Guo *et al.*, 2020]. The significant difference between instance segmentation and primitive segmentation is that instance segmentation only focuses on partitioning individual objects where primitive fitting is absent.

Patch-based representations. Patch-based representations refer to finding a mapping from a 2D patch to a 3D surface. Previous works including [Groueix *et al.*, 2018; Yang *et al.*, 2018; Deng *et al.*, 2020; Bednarik *et al.*, 2020] learn a parametric 2D mapping by minimizing the Chamfer distance [Fan *et al.*, 2017]. One issue with Chamfer distance is that it is not differentiable when using the nearest neighbor to find matched pairs. We learn the uv mapping instead. Learning uv parameters enables us to re-evaluate points from our proposed generalized Bézier primitives, making our training process differentiable and supporting batch operations.

Multi-task learning. Multi-task learning aims to leverage relevant information contained in multiple related tasks to help improve the generalization performance of all the tasks [Zhang and Yang, 2021]. Compared to single-task learning, the architectures used for multi-task learning—see, e.g., [Zhang *et al.*, 2014; Dai *et al.*, 2016]—share a backbone to extract global features, followed by branches that transform the features and utilize them for specific tasks. Inspired by [Dai *et al.*, 2016], we use a cascaded architecture for our joint optimization tasks.

3 Method

Figure 1 shows an overview of the proposed neural network. The input to our method is a 3D point cloud $P = \{p_i | 0 \leq i \leq N-1\}$, where p_i denotes the point coordinates (with or without normals). The output is the per-point patch labels $\{P_k | \cup_{k=0} P_k = P\}$, where each patch corresponds to a Bézier primitive. The network will also output patch degree (d_u-by-d_v) and weighted control points $C = \{\mathbf{c}_{kmn} = (x, y, z, w) | 0 \leq m \leq d_u, 0 \leq n \leq d_v, 0 \leq k \leq K-1\}$, where K denotes the number of patches. We constrain the maximum degree to be $M_d * N_d$. We let our network output a maximum number of K Bézier patches for all CAD models, and we use $\hat{K}$ to denote the ground-truth number of patches which is smaller than K and varies for each CAD model.

3.1 Architecture

Our architecture consists of two components: a backbone for extracting features and a cascaded structure for joint optimization. The backbone is based on three stacked *EdgeConv* [Wang *et al.*, 2019] layers and extracts a 256D pointwise feature for each input point. Let $\mathbf{P} \in \mathbb{R}^{N \times D_{in}}$ denote the input matrix, where each row is the point coordinates (D_{in} is three) with optional normals (D_{in} is six). Let $\mathbf{X} \in \mathbb{R}^{N \times 256}$ denote the 256D pointwise feature matrix extracted from the backbone. We use a cascaded structure to optimize the per-point degree probability matrix $\mathbf{D} \in \mathbb{R}^{N \times (M_d * N_d)}$, the soft membership matrix $\mathbf{W} \in \mathbb{R}^{N \times K}$, the UV parameter matrix $\mathbf{T} \in \mathbb{R}^{N \times 2}$, and the weighted control points tensor $\mathbf{C} \in \mathbb{R}^{K \times (M_d+1) \times (N_d+1) \times 4}$ jointly. Because $\mathbf{D}$, $\mathbf{W}$, $\mathbf{T}$, and $\mathbf{C}$ are coupled, it is natural to use a cascaded structure to jointly optimize them. Here, the cascaded structure is similar to [Dai *et al.*, 2016], where the features are concatenated and transformed for different MLP branches.

3.2 Joint Optimization

We have four modules: decomposition, fitting, embedding, and reconstruction. They are coupled to optimize $\mathbf{D}$, $\mathbf{W}$, $\mathbf{T}$ and $\mathbf{C}$ jointly by using our proposed four modules.

Decomposition Module

Degree classification. We use Bézier primitive with different degrees to replace classical primitives, including plane, sphere, plane, BSpline, etc. For the sake of the classification of degrees, the straightforward idea would be to use a cross-entropy loss: $\mathrm{CE} = -\log(p_t)$, where p_t denotes the possibility of the true degree labels. However, the degree type is

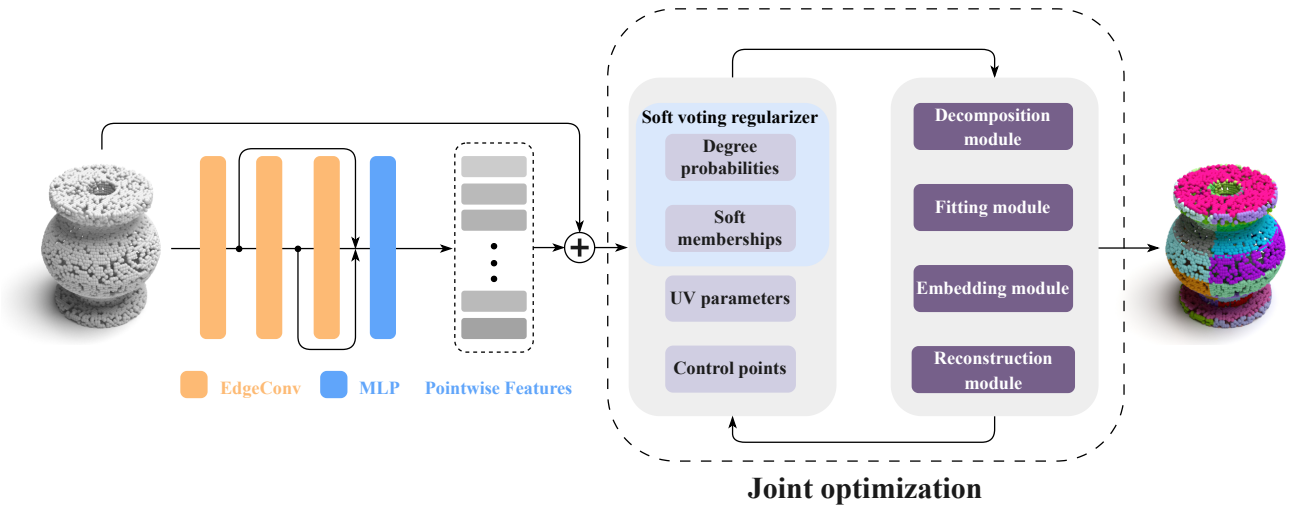


Figure 1: Overview of the proposed pipeline. The network takes a point cloud as input. It outputs pointwise features followed by four modules to predict Bézier geometry and topology: (1) The decomposition module decomposes point clouds into multiple patches. (2) The fitting module regresses uv parameters for each point and control points for each Bézier patch. (3) The embedding module clusters pointwise features assigned to the same patch. (4) The reconstruction module re-evaluates the input point clouds from the above predictions.

highly imbalanced. For example, surfaces of degree type 1-by-1 represent more than 50%, while 3-by-2 surfaces are rare. To deal with the imbalance, we utilize the multi-class focal-loss [Lin *et al.*, 2017]: $\mathrm{FL} = -(1-p_t)^{\gamma}\log(p_t)$, where γ denotes the focusing parameter. Then the degree type classification loss is defined as:

$$L_{\text{deg}} = \frac{1}{N}\sum_{i=0}^{N-1}\mathrm{FL}(\mathbf{D}_{i,:}) \tag{1}$$

Primitive segmentation. The output of primitive segmentation is a soft membership indicating per-point primitive instance probabilities. Each element w_{ik} is the probability for a point p_i to be a member of primitive k. Since we can acquire pointwise patch labels from our data pre-processing, we use a relaxed IOU loss [Krähenbühl and Koltun, 2013; Yi *et al.*, 2018; Li *et al.*, 2019] to regress the $\mathbf{W}$:

$$L_{\text{seg}} = \frac{1}{\hat{K}}\sum_{k=0}^{\hat{K}-1}\left[1 - \frac{\mathbf{W}_{:,k}^{T}\hat{\mathbf{W}}_{:,\hat{k}}}{\|\mathbf{W}_{:,k}\|_1 + \|\hat{\mathbf{W}}_{:,\hat{k}}\|_1 - \mathbf{W}_{:,k}^{T}\hat{\mathbf{W}}_{:,\hat{k}}}\right], \tag{2}$$

where $\mathbf{W}$ denotes the output of the neural network and $\hat{\mathbf{W}}$ is the one-hot encoding of the ground truth primitive instance labels. The best matching pairs $(k, \hat{k})$ between prediction and ground truth are found via the Hungarian matching [Kuhn, 1955]. Please refer to [Li *et al.*, 2019] for more details.

Soft voting regularizer. Since we learn $\mathbf{D}$ and $\mathbf{W}$ separately, points belonging to the same primitive instance may have different degrees, which is undesirable. To favor degree consistency between points assigned to the same primitive, we propose a soft voting regularizer that penalizes pointwise degree possibilities. We first compute a score for each degree case for all primitive instances by $\mathbf{S} = \mathbf{W}^{T}\mathbf{D}$, where each element s_{kd} denotes the soft number of points for degree d in primitive instance k. We then perform L_1-normalization to convert $\mathbf{S}$ into primitive degree distributions $\hat{\mathbf{S}}$:

$$\hat{\mathbf{S}} = \left[\frac{1}{\sum_{d=0} S_{kd}}\right] \odot \mathbf{S}, \tag{3}$$

where the first term denotes the sum of each column and $\odot$ denotes the element-wise product. Finally, we utilize a focal loss to compute the primitive degree voting loss:

$$L_{\text{voting}} = \frac{1}{\hat{K}}\sum_{k=0}^{\hat{K}-1}\mathrm{FL}(\hat{\mathbf{S}}_{k,:}), \tag{4}$$

where FL denotes the focal loss. The global loss for the decomposition module is defined as: $L_{\text{dec}} = L_{\text{deg}} + L_{\text{seg}} + L_{\text{voting}}$.

Fitting Module

Parameter regression. Through Bézier decomposition we obtain the ground truth labels for the (u, v) parameters and record all parameters into matrix $\hat{\mathbf{T}}$. We regress the uv parameters using a mean squared error (MSE) loss:

$$L_{\text{para}} = \frac{1}{N}\sum_{i=0}^{N-1}\|\mathbf{T}_{i,:} - \hat{\mathbf{T}}_{i,:}\|_2^2 \tag{5}$$

Control point regression. We select a maximum number of primitive instances K for all models. As the ground truth primitive instance $\hat{K}$ varies for each model, we reuse the matching pairs directly from the Hungarian matching already computed in the primitive segmentation step. Note that as the predicted degree (d_u, d_v) may differ from the ground truth $(\hat{d_u}, \hat{d_v})$, we align the degree to compute the loss via a maximum operation as $(max(d_u, \hat{d_u}), max(d_v, \hat{d_v}))$. The network always outputs $(M_d + 1) \times (N_d + 1)$ control points

for each primitive corresponding to the predefined maximum degree in U and V direction, and these control points will be truncated by the aligned degree. Furthermore, if the ground-truth degree is smaller than the prediction, we can pad "fake" control points that are zero for the ground-truth patch; otherwise, we just use the aligned degree, which is the maximum of the predicted and the ground truth. Finally, the control point loss is defined as:

$$L_{\text{ctrl}} = \frac{1}{N_{\mathbf{c}}} \sum_{t=0}^{N_{\mathbf{c}}-1} \|\mathbf{c}_t - \hat{\mathbf{c}}_t\|_2^2, \tag{6}$$

where $\mathbf{c}_t$ and $\hat{\mathbf{c}}_t$ denote the matched control points, and $N_{\mathbf{c}}$ is the number of matched control point pairs. Finally, we define the L_{fit} loss as: $L_{\text{fit}} = L_{\text{para}} + L_{\text{ctrl}}$.

Embedding Module

We use the embedding module to eliminate over-segmentation by pulling point-wise features toward their center and pushing apart different centers. Unlike ParSeNet and HPNet, 1) we do not need a mean-shift clustering step which is time-consuming; 2) we calculate the feature center in a weighted manner rather than simply averaging. The weights are chosen as $\mathbf{W}$ and will be automatically updated in the decomposition module; 3) $\mathbf{W}$ will be further optimized to improve the segmentation. Moreover, our embedding module is suitable for batch operations even though the number of primitive instances for each CAD model and the number of points for each primitive varies. Otherwise, one has to apply mean-shift for each primitive, which deteriorates timing further.

To be specific, we use $\mathbf{W}$ to weight $\mathbf{X}$ to obtain primitive features for all candidate primitive instances. Then, we reuse $\mathbf{W}$ to weigh all the primitive instance features to calculate a "soft" center feature for each point. We favor that each point feature embedding should be close to its "soft" center feature, and each primitive instance feature embedding should be far from each other. The primitive instance-wise feature matrix $\mathbf{X}_{\text{ins}}$ is defined as:

$$\mathbf{X}_{\text{ins}} = [\frac{1}{\sum_{i=0}^{N-1} w_{ik}}] \odot (\mathbf{W}^T\mathbf{X}), \tag{7}$$

where each row of $\mathbf{X}_{\text{ins}}$ denotes the instance-wise features for each patch. We then compute the "soft" center feature matrix $\mathbf{X}_{\text{center}}$ as: $\mathbf{X}_{\text{center}} = \mathbf{W}\mathbf{X}_{\text{ins}}$, where each row denotes the "soft" center for each point.

Then we define L_{pull} as:

$$L_{\text{pull}} = \frac{1}{N} \sum_{i=0}^{N-1} \text{Relu}(\|\mathbf{X}_{i,:} - (\mathbf{X}_{\text{center}})_{i,:}\|_2^2 - \delta_{\text{pull}}), \tag{8}$$

and we define L_{push} as:

$$L_{\text{push}} = \frac{1}{2K(K-1)} \sum_{k_1<k_2} \text{Relu}(\delta_{\text{push}} - \|(\mathbf{X}_{\text{ins}})_{k_1,:} - (\mathbf{X}_{\text{ins}})_{k_2,:}\|_2^2). \tag{9}$$

Finally, the total embedding loss L_{emb} is defined as: $L_{\text{emb}} = L_{\text{pull}} + L_{\text{push}}$.

Reconstruction Module

The reconstruction module is designed to reconstruct points from the predicted multiple Bézier primitives, i.e., rational Bézier patches, and further jointly optimize $\mathbf{W}$. One difficulty is that each CAD model has various numbers of primitives, and the degree of each primitive is also different. Therefore, we seek a generalized formula to support tensor operations on re-evaluating points for a batch of CAD models. The straightforward approach would be to compute a synthesizing score for all degree types. Assume the maximum number of primitive instances is K, and we have $M_d * N_d$ types of different degrees. The total number of combinations is $K * M_d * N_d$. We define a synthesizing score for each case in Einstein summation form: $(s_w)_{kci} = w_{ik} * s_{kc}$, where w_{ik} denotes the probability of point p_i to belong to primitive instance k and s_{kc} denotes the degree score for degree type m-by-n indexed with $c = M * (m-1) + (n-1)$ for primitive instance k coming from $\mathbf{S}$. Then, we need to normalize $(s_w)_{kdi}$ such that $\sum_{k,d,i}(s_w)_{kdi} = 1$. Finally, the reconstructed point coordinates p_i are defined as:

$$\begin{pmatrix} x_i' \\ y_i' \\ z_i' \end{pmatrix} = \sum_{k,m,n} (s_w)_{kci} \mathbf{R}_{kmn}(u_i, v_i), \tag{10}$$

where parameter (u_i, v_i) for point p_i is shared for all combinations. Such a formulation makes extending the formula in matrix form easy and avoids resorting to loop operations. However, such an approach is too memory-intensive. We thus truncate the degree from the degree probability matrix by redefining the Bernstein basis function for degree d as:

$$(B_M)_d^l(t) = \begin{cases} \binom{d}{l} t^l (1-t)^{d-l}, & l \le d \\ 0, & l > d \end{cases}, \tag{11}$$

where $0 \le l \le M$, and M is the maximum degree. Then, the reconstructed point coordinates for p_i for a degree m-by-n patch k is:

$$\begin{pmatrix} x_i' \\ y_i' \\ z_i' \end{pmatrix} = \frac{\sum_{m_i}^{M_d} \sum_{n_i}^{N_d} (B_{M_d})_m^{m_i}(u)(B_{N_d})_n^{n_i}(v)\mathbf{c}_{m_i n_i}(c_w)_{m_i n_i} w_{ik}}{\sum_{m_i,n_i} (B_{M_d})_m^{m_i}(u)(B_{N_d})_n^{n_i}(v)(c_w)_{m_i n_i} w_{ik}}, \tag{12}$$

where $\mathbf{c}_{m_i n_i}$ denotes the control point coordinates and $(c_w)_{m_i n_i}$ denotes its weight, and w_{ik} is the element of $\mathbf{W}$.

If we also input the normal $(n_{x_i}, n_{y_i}, n_{z_i})$ for point p_i, we can also reconstruct the normal $(n'_{x_i}, n'_{y_i}, n'_{z_i})$ by:

$$\begin{pmatrix} n'_{x_i} \\ n'_{y_i} \\ n'_{z_i} \end{pmatrix} = \begin{pmatrix} \frac{\partial x_i'}{\partial u} \\ \frac{\partial y_i'}{\partial u} \\ \frac{\partial z_i'}{\partial u} \end{pmatrix} \times \begin{pmatrix} \frac{\partial x_i'}{\partial v} \\ \frac{\partial y_i'}{\partial v} \\ \frac{\partial z_i'}{\partial v} \end{pmatrix}, \tag{13}$$

where $\times$ denotes the cross product.

$\mathbf{p}_i$ denotes the input point coordinates. $\mathbf{p}_i^*$ denotes the reconstructed point coordinates. $\mathbf{n}_{p_i}$ denotes the input point normals. $\mathbf{n}_{p_i}^*$ denotes the reconstructed normals. The coordinate loss is defined as:

$$L_{\text{coord}} = \frac{1}{N} \sum_{i=0}^{N-1} \|\mathbf{p}_i - \mathbf{p}_i^*\|_2^2. \tag{14}$$

If we also input the normals, the normal loss is defined as:

$$L_{\text{norm}} = \frac{1}{N}\sum_{i=0}^{N-1}(1 - |\mathbf{n}_{p_i}^T \mathbf{n}_{p_i}^*|). \tag{15}$$

The loss for the reconstruction module is defined as:

$$L_{\text{recon}} = \begin{cases} L_{\text{coord}}, & \text{without normals,} \\ L_{\text{coord}} + L_{\text{norm}}, & \text{with normals.} \end{cases} \tag{16}$$

Total Loss

The total loss is defined as the sum of decomposition, fitting, embedding, and reconstruction losses: $L = L_{\text{dec}} + L_{\text{fit}} + L_{\text{emb}} + L_{\text{recon}}$. We do not use different weights for each loss item because all point clouds are normalized into a unit sphere. Moreover, the uv parameters are outputted directly from a $sigmoid$ layer, and the control points are outputted directly by a $tanh$ layer. Thus, each loss item is almost at the same scale, so we do not need different weights for each loss item. Furthermore, we use different learning rates for different modules to balance the training. Specific training details are listed in section 4.2.

4 Experiments

4.1 Dataset Pre-Processing

We evaluate our approach on the ABC dataset [Koch *et al.*, 2019]. However, the ABC dataset does not have the annotations to learn Bézier decomposition on point clouds. Therefore, we do a pre-processing step. Specifically, we utilize the CGAL library [CGAL, 2009] and OpenCascade library [OpenCascade, 2018] to perform Bézier decomposition on STEP files directly and perform random sampling on the surface to obtain the following labels: point coordinates, point normals, point uv parameters, surface patch indices of the corresponding points, surface patch degrees, and surface patch control points. Finally, we use 5,200 CAD models for training and 1,300 CAD models for testing. Each CAD model contains randomly sampled $8,192$ points (non-uniform) with annotations.

4.2 Training Details

We train a multi-task learning model. The learning rates differ depending on the MLP branch. The learning rate for the backbone, soft membership, and uv parameters is set to 10^{-3}, while the learning rate for the degree probabilities and control points is set to 10^{-4}. As we have several learning tasks that are not independent, we set a lower learning rate for loss items, such as degree probabilities which converges faster. We set γ as 3.0 for the focal loss, and δ_{pull} as 0 and δ_{push} as 2.0 for the embedding losses. We employ ADAM to train our network. The model is then trained using 150 epochs.

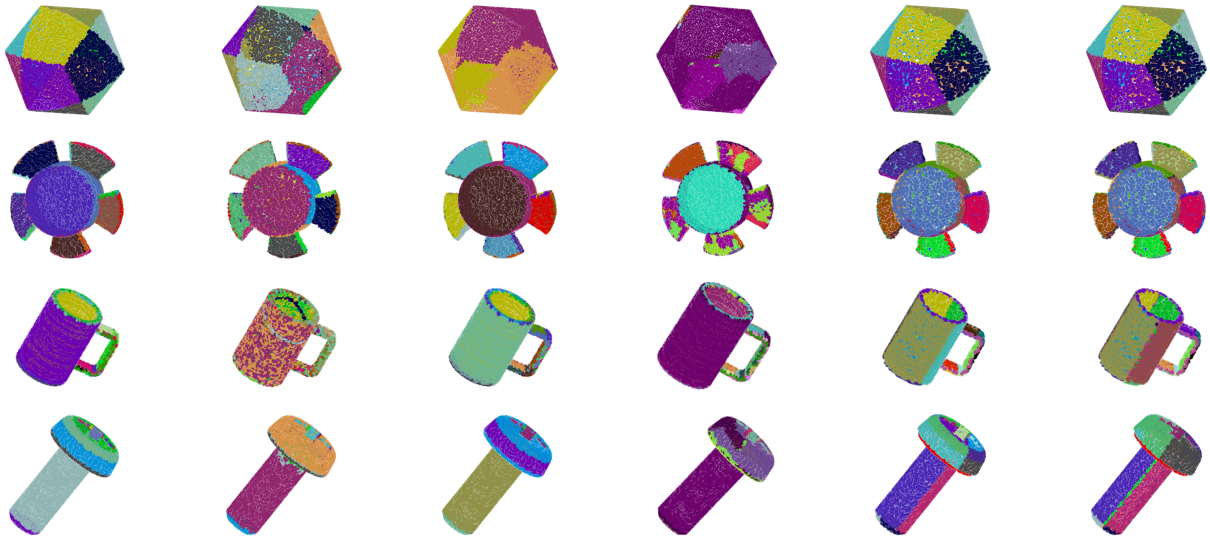
4.3 Comparisons

We compare our algorithm with SPFN, ParSeNet, and HPNet [Li *et al.*, 2019; Sharma *et al.*, 2020; Yan and Yang, 2021]. We use both points and normals for training all the algorithms. Since SPFN only supports four types of canonical primitives (plane, sphere, cone, and cylinder), we consider points belonging to other primitives falling out of the supported canonical primitive types as the "unknown" type. To make fair comparisons, we modify SPFN to let the network take point coordinates and normals as input for training. For ParSeNet, we only train the segmentation module on the ABC dataset. We use their pre-trained fitting model (SplineNet) directly. For HPNet, we also use the pre-trained fitting model directly, which is the same as ParSeNet. We observed that the output of HPNet is very sensitive to the number of points. In order to use HPNet at its best, we down-sample the point clouds to 7k points for training and testing. We choose the following evaluation metrics:

1. **Primitive Type Accuracy** ("Acc"):
   $\frac{1}{K}\sum_{k=0}^{K-1}\mathbb{I}(t_k == \hat{t}_k)$, where t_k and $\hat{t}_k$ are predicted primitive type and ground truth type, respectively. This is used to measure the type accuracy. Note that our primitive types differ from other baselines.
2. **Rand Index** ("RI"):
   $\frac{a+b}{c}$, where c is $\binom{N}{2}$ denoting the total possible pairs for all points, and a denotes the number of pairs of points that are both in the same primitive of prediction and ground truth, while b denotes the number of pairs of points that are in a different primitive of prediction and ground truth. Rand index is a similarity measurement between two instances of data clustering, and a higher value means better performance [Chen *et al.*, 2009; Yi *et al.*, 2018].
3. **Normal Error** ("Err"):
   $\frac{1}{N}\sum_{i=0}^{N-1}\arccos(|\mathbf{n}_{p_i}^T \mathbf{n}_{p_i}^*|)$, where $\mathbf{n}_{p_i}$ and $\mathbf{n}_{p_i}^*$ are ground truth and predicted unit normal, respectively.
4. **Inference Time** ("Time"):
   The inference time on the whole test dataset.
5. **Average Primitive Number** ("Num"):
   The predicted average number of primitives on the whole test data set.

We record these evaluation metrics in table 1 and 2. Figure 2 shows visual depictions of the results. Our results show the best performance regarding primitive type accuracy, normal fitting error, and inference time. Our method is much faster for inference because it uses a general formula for different primitive types, and the embedding module is free of iterations. Other methods treat primitives with different equations, and ParSeNet and HPNet need a mean-shift step. Even though our approach may lead to more segmented primitives by the nature of Bézier decomposition, the evaluation metrics of primitive type accuracy and normal fitting error are computed in a point-wise manner. Thus, over-segmentation and under-segmentation will not lead to smaller or bigger errors due to fewer or more segmented primitives.

We also show the performance of all the methods without normals as input. For our method and SPFN, we only input point coordinates into the neural networks but use normals as supervision. Since ParSeNet does not regress normals, we cannot use normals as supervision. We train ParSeNet without normals as input to test its performance. HPNet uses the network to regress the normals from the input and also utilizes the ground truth normals to construct an affinity matrix



(a) GT-Basic (b) HPNet (c) ParSeNet (d) SPFN (e) GT-Ours (f) Ours

Figure 2: Comparisons on the ABC dataset, where GT-Ours denotes the ground truth for Bézier decomposition, and GT-Basic denotes the ground truth for HPNet, ParSeNet, and SPFN.

Method	Acc(%)(↑)	RI(%)(↑)	Err(rad)(↓)	Time(min)(↓)	Num
HPNet	94.09	97.76	0.1429	1120.31	13.75
ParSeNet	94.98	97.18	-	252.01	14.06
SPFN	83.20	93.03	0.1452	11.52	21.02
Ours	**96.83**	95.68	**0.0522**	**4.25**	19.17

Table 1: Evaluation for primitive instance segmentation on the ABC-decomposition dataset. "rad" denotes the radian. Here we input both point coordinates and normals.

Method	Acc(%)(↑)	RI(%)(↑)	Err(rad)(↓)	Time(min)(↓)	Num
HPNet	90.45	96.04	0.2256	1165.95	22.08
ParSeNet	90.31	94.55	-	206.74	15.39
SPFN	72.90	80.76	0.3309	10.46	27.61
Ours	**92.34**	90.54	**0.2003**	**4.05**	33.67

Table 2: Here we only input point coordinates into the neural network.

as a post-processing step for clustering. We modify HPNet to let the affinity matrix be constructed from the regressed normals instead of the ground-truth normals. Table 2 records the evaluation metrics of each method. From the experiments, we deduce that normals are important for the task of parametric primitive segmentation.

4.4 Ablation Studies

We first conduct experiments to verify the usefulness of the soft voting regularizer. The soft voting regularizer favors point primitive type consistency for each primitive instance, i.e., points assigned to the same primitive instance should have the same primitive type. From our experiment, we find that the soft voting regularizer not only improves the primitive type accuracy but also accelerates training relaxed IOU. Please refer to figure 3 and the last two rows of table 3.

Module	Acc(%)(↑)	RI(%)(↑)	Err(rad)(↓)	Num
D	97+0.10	96+0.83	1.0982	24.46
D+E	97+0.26	96−0.36	0.9834	22.54
D+F	97+0.19	96+0.52	0.5424	23.45
D+E+F	97−0.02	96−0.39	0.4884	20.85
D+F+R	97+0.19	96+0.48	0.0819	23.08
No-voting	97−1.32	96−0.44	0.0547	19.45
Full-module	97−0.17	96−0.32	0.0522	19.17

Table 3: Ablation study. D denotes the decomposition module. E denotes the embedding module. F denotes the fitting module. R denotes the reconstruction module. No-voting denotes using all modules without the soft voting regularizer. Full-module denotes using all the modules plus the soft voting regularizer.

We also verify the functionalities of each module. If we only use the decomposition module, the result is not good even though the "Acc" and "RI" are slightly higher because the decomposition module ignores the fitting, limiting the segmentation applicable to specific datasets. The reconstruction module reduces the "Err" significantly compared to the fitting module because the reconstruction module controls how "well-fitted" a predicted Bézier primitive is to the input point clouds. In contrast, the fitting module only regresses the control points and uv parameters. The embedding module is designed to eliminate small patches that contain few points, seeing the "Num" column. Therefore, experimenting with the embedding module results in fewer patch numbers than its counterpart. To conclude, training with all the modules yields the best results.

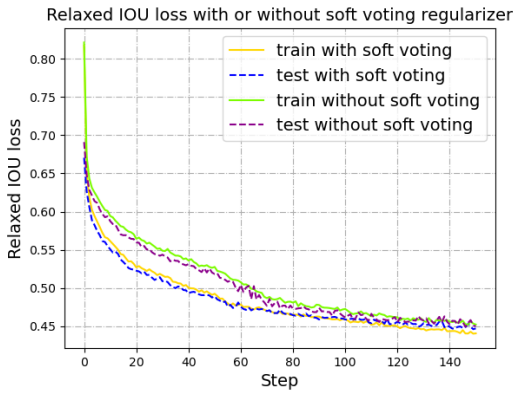


Figure 3: Relaxed IOU loss with or without soft voting loss.

(a) HPNet (b) ParSeNet (c) SPFN (d) Ours

Figure 4: Stress tests on real-scan data. The above two rows are CAD point clouds, and the last two are free-form point clouds.

4.5 Stress Tests

To test whether our algorithm can work in real-world scenarios, we show more results from the real-scan data from the Aim@Shape dataset [Falcidieno, 2004]. The sampling is non-uniform, with missing data and measurement noise compared to the ABC dataset. Besides, We cannot train the network on those data directly because they lack ground-truth labels. Instead, we use the models trained on the ABC dataset and test the performance on real-scan data. Our algorithm still works, while other methods are sensitive. Another positive aspect is that our algorithm could decompose the axis-symmetric free-form point clouds with much smoother boundaries of different patches. Please refer to figure 4.

We also test the performance of our network by adding Gaussian white noise. Specifically, we apply different scales of Gaussian white noise to the point coordinates after normalizing them into a unit sphere. The noise scale denotes the standard deviation of the Gaussian white noise. It ranges from 0.01 to 0.05. We train our network on noise-free data but test the network with Gaussian white noise. Please refer to table 4.

Noise scale	Acc(%)(↑)	RI(%)(↑)	Err(rad)(↓)	Num
No-noise	**96.83**	**95.68**	**0.0522**	19.17
0.01	96.75	94.27	0.0525	20.38
0.02	96.63	93.48	0.0529	21.68
0.03	96.34	92.73	0.0538	22.76
0.04	96.15	92.04	0.0552	23.68
0.05	96.07	91.34	0.0559	24.38

Table 4: Evaluation of our algorithm at different noise scales.

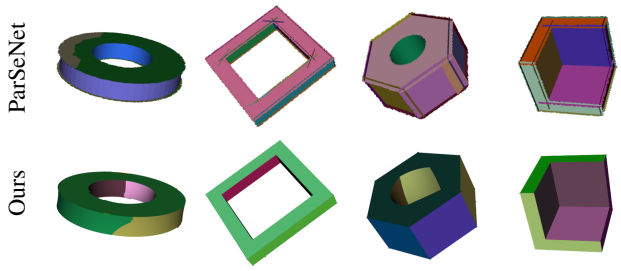


Figure 5: Reconstruction of the full Bézier model and visual comparison with ParSeNet.

4.6 Applications

We can reconstruct the full Bézier model from the Bézier primitive segmentation. We do not follow ParSeNet to pretrain a model that outputs a fixed control point size. Instead, we reuse the rational Bézier patch to refit the canonical Bézier patch. We treat the degrees of the canonical Bézier patch the same as the rational Bézier patch. As a result, we fetch the segmentation and degrees of each patch predicted from the network. Then, we use the parameterization [Lévy *et al.*, 2002] to recompute uv parameters and least squares to refit control points for each patch. Each patch is expanded by enlarging the uv domain to guarantee intersections with its adjacent patches. After that, we use the CGAL co-refinement package [CGAL, 2009] to detect intersecting polylines for adjacent tessellated patches and trim the tessellated patch with the intersected polylines. Our reconstructed full Bézier model can preserve the sharp features, while the boundaries of ParSeNet for different primitives are jaggy and thus fail to preserve the sharp features. Please refer to figure 5.

5 Conclusion

This paper presents an end-to-end method to group points by learning Bézier decomposition. In contrast to approaches treating different geometric primitives separately, our method uses a general formulation for different primitive types. Regarding limitations, Bézier decomposition may naturally generate overly complex segmentations. In addition, we choose the rational Bézier patch as the primitive type. As the formulation is not linear, fitting the parametric patch is not direct. In future work, we wish to use the neural network to directly regress the canonical Bézier patch.

Acknowledgements

This research is part of a project that has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No. 860843. The work of Pierre Alliez is also supported by the French government, through the 3IA Côte d'Azur Investments in the Future project managed by the National Research Agency (ANR) with the reference number ANR-19-P3IA-0002.

References

[Bednarik *et al.*, 2020] Jan Bednarik, Shaifali Parashar, Erhan Gundogdu, Mathieu Salzmann, and Pascal Fua. Shape reconstruction by learning differentiable surface representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4716–4725, 2020.

[CGAL, 2009] CGAL. The computational geometry algorithms library. https://www.cgal.org/, 2009. Accessed: 2023-05-15.

[Chen *et al.*, 2009] Xiaobai Chen, Aleksey Golovinskiy, and Thomas Funkhouser. A benchmark for 3d mesh segmentation. *ACM Transactions on Graphics (TOG)*, 28(3):1–12, 2009.

[Dai *et al.*, 2016] Jifeng Dai, Kaiming He, and Jian Sun. Instance-aware semantic segmentation via multi-task network cascades. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3150–3158, 2016.

[Deng *et al.*, 2020] Zhantao Deng, Jan Bednařík, Mathieu Salzmann, and Pascal Fua. Better patch stitching for parametric surface reconstruction. In *2020 International Conference on 3D Vision (3DV)*, pages 593–602. IEEE, 2020.

[Engelmann *et al.*, 2020] Francis Engelmann, Martin Bokeloh, Alireza Fathi, Bastian Leibe, and Matthias Nießner. 3d-mpa: Multi-proposal aggregation for 3d semantic instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9031–9040, 2020.

[Falcidieno, 2004] Bianca Falcidieno. Aim@shape project presentation. In *Proceedings of Shape Modeling International (SMI)*, pages 329–329. IEEE Computer Society, 2004.

[Fan *et al.*, 2017] Haoqiang Fan, Hao Su, and Leonidas J Guibas. A point set generation network for 3d object reconstruction from a single image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 605–613, 2017.

[Fischler and Bolles, 1981] 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*, 24(6):381–395, 1981.

[Groueix *et al.*, 2018] Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. A papier-mâché approach to learning 3d surface generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (TPAMI)*, pages 216–224, 2018.

[Guo *et al.*, 2020] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. Deep learning for 3d point clouds: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2020.

[Huang *et al.*, 2021] Jingwei Huang, Yanfeng Zhang, and Mingwei Sun. Primitivenet: Primitive instance segmentation with local primitive embedding under adversarial metric. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 15343–15353, 2021.

[Jiang *et al.*, 2020] Li Jiang, Hengshuang Zhao, Shaoshuai Shi, Shu Liu, Chi-Wing Fu, and Jiaya Jia. Pointgroup: Dual-set point grouping for 3d instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4867–4876, 2020.

[Kaiser *et al.*, 2019] Adrien Kaiser, Jose Alonso Ybanez Zepeda, and Tamy Boubekeur. A survey of simple geometric primitives detection methods for captured 3d data. *Computer Graphics Forum (CGF)*, 38(1):167–196, 2019.

[Koch *et al.*, 2019] Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. 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.

[Krähenbühl and Koltun, 2013] Philipp Krähenbühl and Vladlen Koltun. Parameter learning and convergent inference for dense random fields. In *International Conference on Machine Learning (ICML)*, pages 513–521. PMLR, 2013.

[Kuhn, 1955] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[Lévy *et al.*, 2002] Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérome Maillot. Least squares conformal maps for automatic texture atlas generation. *ACM Transactions on Graphics (TOG)*, 21(3):362–371, 2002.

[Li *et al.*, 2019] Lingxiao Li, Minhyuk Sung, Anastasia Dubrovina, Li Yi, and Leonidas J Guibas. Supervised fitting of geometric primitives to 3d point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2652–2660, 2019.

[Lin *et al.*, 2017] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017.

[Marshall *et al.*, 2001] David Marshall, Gabor Lukacs, and Ralph Martin. Robust segmentation of primitives from range data in the presence of geometric degeneracy. *IEEE Transactions on pattern analysis and machine intelligence (TPAMI)*, 23(3):304–314, 2001.

[OpenCascade, 2018] OpenCascade. Open cascade technology occt. https://www.opencascade.com/, 2018. Accessed: 2023-05-15.

[Rabbani *et al.*, 2007] Tahir Rabbani, Sander Dijkman, Frank van den Heuvel, and George Vosselman. An integrated approach for modelling and global registration of point clouds. *ISPRS journal of Photogrammetry and Remote Sensing*, 61(6):355–370, 2007.

[Schnabel *et al.*, 2007] Ruwen Schnabel, Roland Wahl, and Reinhard Klein. Efficient ransac for point-cloud shape detection. *Computer Graphics Forum (CGF)*, 26(2):214–226, 2007.

[Sharma *et al.*, 2020] Gopal Sharma, Difan Liu, Subhransu Maji, Evangelos Kalogerakis, Siddhartha Chaudhuri, and Radomír Měch. Parsenet: A parametric surface fitting network for 3d point clouds. In *European Conference on Computer Vision (ECCV)*, pages 261–276. Springer, 2020.

[Tulsiani *et al.*, 2017] Shubham Tulsiani, Hao Su, Leonidas J Guibas, Alexei A Efros, and Jitendra Malik. Learning shape abstractions by assembling volumetric primitives. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2635–2643, 2017.

[Wang *et al.*, 2018] Weiyue Wang, Ronald Yu, Qiangui Huang, and Ulrich Neumann. Sgpn: Similarity group proposal network for 3d point cloud instance segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2569–2578, 2018.

[Wang *et al.*, 2019] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 38(5):1–12, 2019.

[Yan and Yang, 2021] Siming Yan and Zhenpei Yang. Hpnet: Deep primitive segmentation using hybrid representations. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[Yang *et al.*, 2018] Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian. Foldingnet: Point cloud auto-encoder via deep grid deformation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 206–215, 2018.

[Yang *et al.*, 2019] Bo Yang, Jianan Wang, Ronald Clark, Qingyong Hu, Sen Wang, Andrew Markham, and Niki Trigoni. Learning object bounding boxes for 3d instance segmentation on point clouds. *Advances in neural information processing systems (NIPS)*, 32, 2019.

[Yi *et al.*, 2018] Li Yi, Haibin Huang, Difan Liu, Evangelos Kalogerakis, Hao Su, and Leonidas Guibas. Deep part induction from articulated object pairs. *ACM Transactions on Graphics (TOG)*, 37(6):1–15, 2018.

[Yi *et al.*, 2019] Li Yi, Wang Zhao, He Wang, Minhyuk Sung, and Leonidas J Guibas. Gspn: Generative shape proposal network for 3d instance segmentation in point cloud. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3947–3956, 2019.

[Zhang and Yang, 2021] Yu Zhang and Qiang Yang. A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering*, 34(12):5586–5609, 2021.

[Zhang *et al.*, 2014] Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Facial landmark detection by deep multi-task learning. In *European Conference on Computer Vision (ECCV)*, pages 94–108. Springer, 2014.

[Zou *et al.*, 2017] Chuhang Zou, Ersin Yumer, Jimei Yang, Duygu Ceylan, and Derek Hoiem. 3d-prnn: Generating shape primitives with recurrent neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 900–909, 2017.