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

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

In this supplementary, we first evaluate the ability of our approach to process real scan data with real noise pattern (Section S.1). Then, we visualize zoomed-in renderings of CPFN outputs to demonstrate its ability to capture smaller primitives compared to the baselines (Section S.2). We further report the performance of CPFN in terms of memory and computational complexity (Section S.3). Then, we provide an individual evaluation of both the patch selection network accuracy (Section S.4) and the merging strategy against a widely used commercial solver Gurobi [1] (Section S.5).

We then further evaluate CPFN by varying the input resolution (Section S.6) and the value of the primitive scale threshold η (Section S.7). We also evaluate the impact of the value of the target number of primitives K_{glob} and K_{loc} on shapes with fewer primitives (Section S.8). Then, we report the performance of the global SPFN using a new sampling strategy based on curvature to create the low resolution point clouds (Section S.9). We also assess alternative patch selections by sampling patches either on parts with poor detection with the global SPFN (Section S.10) or with high curvature areas (Section S.11).

We finally provide additional details on our architecture (Section S.12). Our implementation in Pytorch is publicly available on our GitHub page: https://github.com/erictuanle/CPFN

S.1. Qualitative Evaluation on Real Scan Data

We tested our CPFN with the real scans provided by [3]. The resolution of the provided scans is much lower than our synthetic dataset: only 20k points compared to 128k points used in our experiments. The pattern and the scale of the noise are also significantly different with our previous experiments. Yet, CPFN provides reasonable results as shown in Figure S1.

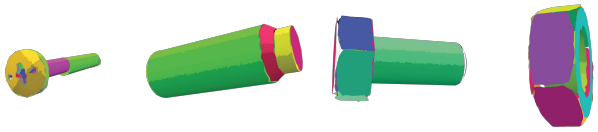

Figure S1. CPFN results on real scans (from [3]).

S.2. Qualitative Evaluation of Small Primitive Detection and Fitting

To visually assess the efficiency of CPFN in detecting small primitives, we visualize in Figure S2 zoomed-in renderings of the predictions for smaller primitives. As shown, our CPFN pipeline improves significantly the detection and the fitting of the smaller primitives compared to the RANSAC [7] and SPFN [3] baselines.

S.3. Memory and Computation Complexity

To run SPFN [3] at training time, one would need to double the average consumer GPU memory for a batch size of 8 for the high resolution Traceparts dataset while our method could still be applicable for even higher resolutions.

At test time, CPFN only requires three forward passes - through the patch selection network ($\sim$0.02s) and the global ($\sim$0.25s) and local ($\sim$0.22s) SPFN respectively - followed by the merging step ($\sim$0.43s). Note that our heuristic merging step takes less than one second per scan, which is incomparably faster than running a commercial optimization solver as described in Section S.5. Both SPFN [3] and CPFN have similar memory footprint at test time.

S.4. Quantitative Evaluation of the Patch Selection Network

We assess the accuracy, precision and recall of our patch selection network trained to identify the regions which are likely to contain small primitives at test time. We report the quantitative results in Table S1 for varying scales of primitives, i.e., $\eta \in \{1\%, 2\%, 3\%, 4\%, 5\%\}$ (see Section 3.4).

By increasing the value of η, we enlarge the set of primitives considered as small and as such, we also increase the number of local patches to retrieve. The accuracy of our patch selection network varies between 87.96% and 98.46%

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

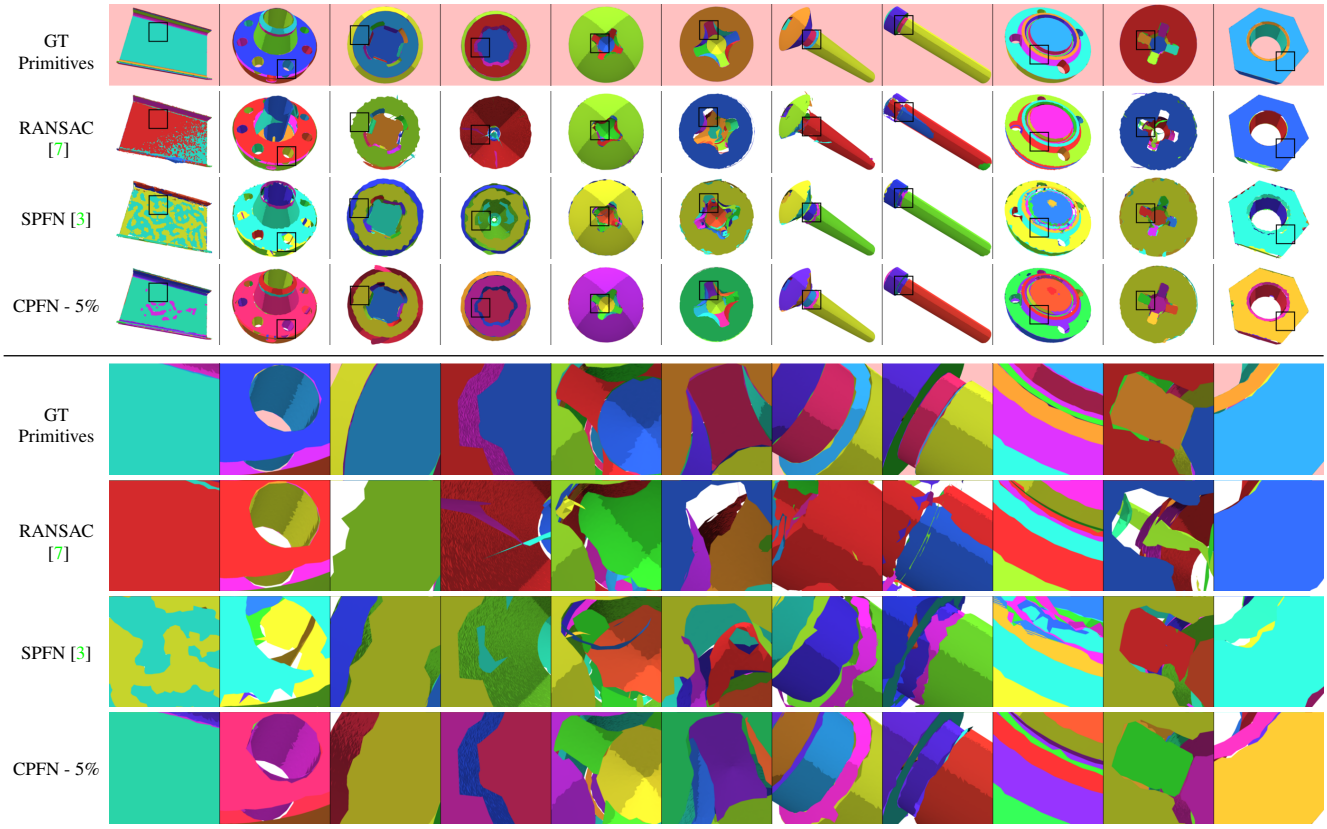

Figure S2. Primitive fitting results for RANSAC, plain SPFN and our CPFN networks (Rows 1-4) and zoomed-in visualizations on smaller primitives (Rows 5-8). RANSAC and SPFN directly operate on the global object failing on the small primitives. Our CPFN pipeline estimates the heatmaps corresponding to small primitives at different scales and learns a better primitive decomposition on local patches sampled from such regions improving the detection of small primitives.

but slightly decreases with the scale. The average precision and recall is 57.21% and 52.75% respectively emphasizing that identifying fine-scale regions in a low-resolution point cloud is a challenging task. Nevertheless, our pipeline is not that sensitive to the accuracy of the patch selection stage. In fact, in our ablation studies (Table 1, row 14 in the paper), we show that a perfect patch selection has a marginal impact on the output performance.

Table S1. Accuracy and precision/recall trade-off of our patch selection network trained to identify small primitives, i.e. primitives with less than $\eta \cdot N$ points (see Section 3.4).

Scale η	Accu.	Prec.	Recall	TP	FP	TN	FN
1%	**98.46**	**60.85**	37.14	0.66	**0.43**	**97.79**	**1.12**
2%	96.27	57.65	52.82	2.29	1.68	93.98	2.05
3%	91.98	54.65	53.25	4.70	3.90	87.28	4.12
4%	88.99	54.23	59.35	7.20	6.08	81.79	4.93
5%	87.96	58.65	**61.21**	**8.99**	6.34	78.79	5.70

S.5. Quantitative Evaluation of our Merging Heuristic Compared to a Binary Programming Solver

As explained in Section 3.3, our segment merging problem is a quadratic binary programming problem. Thus, instead of using our Hungarian-algorithm-based merging method, we tried a commercial solver, Gurobi [1] (using a branch-and-bound algorithm), and compared the performance with ours in terms of both the computation time and the segmentation accuracy after the merging. Since the branch-and-bound method may take a huge amount of time, we set the time limit to 10 mins. When we tested both methods with 26 randomly picked test cases, in 22 cases out of them, Gurobi failed to find the optimum solution in 10 mins. Contrarily, our method took 0.43 secs on average. The resulting mIoU of Gurobi for the random 26 cases was 50.54%, while our heuristic achieved 81.42%. This gap in performance highlights that even after 10 mins, the last solution found by Gurobi is still far away from optimum. Even when Gurobi managed to get a better solution than ours (with a lower energy in the optimization) before the 10 mins limit, the improvement on mIoU was marginal:

+0.09%.

S.6. Quantitative Evaluation of CPFN with Different Point Cloud Resolutions

We report the per-primitive-type mIoU in Table S2 at two different input resolutions - 16k and 128k. The comparison between the results of SPFN [3] and our CPFN at 16k and 128k shows that higher resolutions take more benefits from our two-level prediction architecture.

Also CPFN performs better than SPFN [3] for all primitive types. The gap between the highest (Sphere) and the lowest (Cone) mIoUs is also smaller compared to SPFN [3], meaning that our performance is more balanced across the different primitive types.

Table S2. Percentages of each primitive type (top row) and mean IoUs of SPFN [3] and CPFN for each type at 16k and 128K input resolutions.

Type		Cone	Cylinder	Plane	Sphere
Pct. Primitives		26.95	29.23	39.98	3.84
16k	SPFN	56.51	67.29	66.11	82.70
	CPFN	68.85	75.94	74.83	86.68
128k	SPFN	57.13	67.54	66.38	83.24
	CPFN	**76.46**	**77.85**	**80.57**	**87.79**

S.7. Impact of the Primitive Scale Threshold η

We analyze the effect of an increase for the value of the primitive scale threshold η (see Section 3.4), which is the maximum scale that can be selected by our patch selection network, in Table S3. Doubling η from 5% to 10% brings a slight increase in the performance for the bigger primitives but reduces the ability to detect small primitives.

Table S3. Segmentation accuracy of CPFN at various primitive scales with two different η values 5% and 10%. Each scale bucket contains roughly the same number of primitives

Scale	∼1%	1%∼2%	2%∼4%	4%∼12%	12%∼
RANSAC [4]	34.68	40.38	56.78	70.63	69.50
SPFN [3]	44.25	55.53	70.12	74.29	79.75
CPFN - 5%	**65.74**	77.31	84.19	83.55	83.95
CPFN - 10%	65.58	**77.49**	**84.31**	**85.16**	**84.66**

S.8. Impact of the Values of the Maximum Number of Primitives Predicted by each SPFN K_{glob} and K_{loc}

We set reasonably big numbers for K_{glob} and K_{loc} (see Section 3.3) to handle all test cases in our dataset. But, obviously we cannot set very huge numbers due to the GPU memory issue — many neural networks including SPFN [3] have the same technical issue. If the input scan has a huge number of primitives, as an alternative, we can consider using only local SPFN with small size patches. Table S4 shows how the performance of our CPFN changes when the number of primitives in an input varies for the fixed values of $K_{\text{glob}} = 28$ and $K_{\text{loc}} = 21$. One worthwhile observation is that setting big numbers to K_{glob} and K_{loc} does not affect the case of having a few primitives in the input scan

Table S4. Segmentation accuracy (%) of CPFN with respect to the number of primitives in the input shape.

Nb Primitives	3∼8	8∼12	12∼14	14∼25
CPFN ($\eta = 5\%$)	78.58	81.36	83.17	74.61

S.9. Curvature-based Importance Sampling for the Global SPFN

Instead of generating the low-resolution point cloud with FPS sampling only, we evaluate a new importance sampling strategy based on curvature. We first estimated point curvatures directly from the point cloud via jet-fitting using CGAL [4] with default parameters. (We clipped outlier values due to numerical instabilities.) We trained SPFN using low-resolution point clouds sampled in two steps: (i) half of the points were sampled using FPS sampling, and (ii) the remaining half was randomly sampled proportionally to mean curvature. For the Traceparts [6] dataset, we obtain an mIoU of $\mathbf{65.49}\%$, which is lower than $\mathbf{66.29}\%$ with the full FPS based sampling and also $\mathbf{79.64}\%$ with our CPFN pipeline.

S.10. Patch Selection Based on Global SPFN IoUs

In order to assess our sampling strategy described in Section 3.4, we compare two types of heatmaps: (i) the heatmap highlighting areas with low global SPFN mIoU and (ii) the heatmap estimated by our patch selection network to identify small primitives. As shown in Figure S3, both heatmaps have high values in similar areas, meaning that sampling patches from either heatmap will produce similar patch samples. The last column in the figure (column 11) is an exceptional case when the global SPFN fails to properly segment large primitives due to their proximity.

S.11. Patch Selection Based on Curvature

We also tried estimating the point heatmaps based on the previously computed mean curvature. We sampled patches from the top 20% points with the highest mean curvature at training and test time. This approach did not perform as good as our original CPFN pipeline: $\mathbf{76.00}\%$ mIoU compared with $\mathbf{79.64}\%$ of ours as high curvature areas do not cover all of the small primitives.

S.12. Implementation and Training Details

We provide a more detailed description of our architecture and parameters used in our experiments. We first explain the design of both the global and local SPFN in

Section S.12.1 and the patch selection network in Section S.12.2. We then detail how we pre-processed the dataset (Section S.12.3) and the optimization parameters used for the training (Section S.12.4). We finally explain how the full pipeline operates at training (Section S.12.5) and test time (Section S.12.6).

S.12.1 Supervised Primitive Fitting Network (SPFN)

Throughout our experiments, we use our re-implementation of the SPFN [3] pipeline in Pytorch.

SPFN [3] is itself based on a single-scale PointNet++ [5] for which the default hyperparameters are used. The default PointNet++ implementation is designed as an encoder-decoder architecture, which progressively decreases the point cloud resolution with depth, from the input resolution to 512, 128 and finally to a single point vector. The decoder symmetrically upsamples the point cloud to the input resolution. After a common linear layer, the last PointNet++ [5] layers is replaced to produce three per-point outputs for the segmentation $\mathbf{W}$, the type $\mathbf{T}$ and the normal $\mathbf{N}$ from three dense layer heads. On top of PointNet++ backbone, SPFN [3] (see Section 3.1) computes the primitive parameters based on the network outputs and supervises the training on five different losses: (i) segmentation loss $\mathcal{L}_{\text{seg}}$, (ii) normal loss $\mathcal{L}_{\text{norm}}$, (iii) primitive type loss $\mathcal{L}_{\text{type}}$, (iv) residual loss $\mathcal{L}_{\text{res}}$, i.e., fitting loss, and (v) axis loss $\mathcal{L}_{\text{axis}}$.

We keep unchanged almost all implementation details from the original Tensorflow implementation. All hyperparameters are kept identical for both of our SPFN-based sub-networks, i.e., global (Section 3.1) and local SPFN (Section 3.2). We train separately the global and local SPFN since they act on different scales of the input point cloud. As explained in the paper, we modify the local SPFN to use the global $\mathbf{l}_o$ and local context $\mathbf{l}_i^g$ extracted from the global SPFN by feeding an augmented latent code $\mathbf{l}_i^{'} = [\mathbf{l}_i, \quad \mathbf{l}_o, \quad \mathbf{l}_i^g]$ to the decoder. The first module from the decoder is thus modified to accept the additional input channels. Another difference is the maximum number of primitives each SPFN instance can predict. As global objects are likely to contain more primitives than patches, the global SPFN is trained to be able to predict more primitives (K_{glob}) than the local SPFN counterpart (K_{loc}). This boils down to change the number of output channels of the segmentation head to the desired number of primitives. For the TraceParts [6] dataset, we fix $K_{\text{glob}} = 28$ and $K_{\text{loc}} = 21$ which are respectively the maximum number of primitives in a single object and in a single patch found in the dataset.

Both SPFNs are trained using ground truth information extracted from the CAD models, either at the object- or the patch-level: (i) point-to-primitive assignment, (ii) point normals, (iii) primitive types and (iv) primitive axis (except for spheres).

S.12.2 Patch Selection Network

The patch selection network (Section 3.4), also based on the same default PointNet++ [5] implementation, produces a single binary classification tensor and is trained using the binary cross-entropy loss. Thus, we replace the last linear layer of the backbone PointNet++ [5] by a new one with 2 output channels. The GT small-primitive heatmaps for supervision are based on the primitives which area are smaller than a threshold η.

S.12.3 Dataset Parameters

The Traceparts [6] dataset is pre-processed to merge adjacent primitives sharing the same parameters and discard extremely tiny primitives. Differently to [3], only primitives with an area smaller than 0.5% (instead of 2%) of the entire object are discarded to make sure smaller primitives are included in the dataset. Higher resolution point clouds allow our approach to capture accurately small primitives that were originally discarded. The point clouds are normalized to the unit sphere and randomly perturbed with uniform noise $[-5e-3, 5e-3]$ along the ground truth normal direction. For comparison, the average sampling distance ($d_s \approx 5e-3$) is equivalent to that noise level.

S.12.4 Optimization Parameters

For all SPFN modules, we use a batch size of 16 samples with both (i) a learning decay (initial learning rate 10^{-3} with staircase learning decay 0.7) and (ii) a batch norm decay (initial batch norm decay 0.5 with staircase decay 0.5). For optimization, we use Adam [2] with β coefficients of 0.9 and 0.999 for the gradient and its square respectively. For the patch selection network, the same set of training parameters are used except the batch size which is increased to 32.

The global SPFN and the patch selection network are trained with 100 epochs. The local SPFN input dataset is much bigger than the original dataset as multiple patches will be sampled on the same object. Thus, we fix the number of epochs for the local SPFN so that all trainings have the same number of iterations.

S.12.5 CPFN Pipeline at Training Time

Our full CPFN pipeline is trained as a sequential cascaded process. We first train the global SPFN (Section 3.1) and the patch selection network (Section 3.4) on downsampled point clouds $n = 8\,192$. The output of the global SPFN provides an initial primitive decomposition of the input object with poor accuracy on small primitives. The patch selection network is thus trained to learn how to sample patches in those smaller primitive scale areas. The patch selection

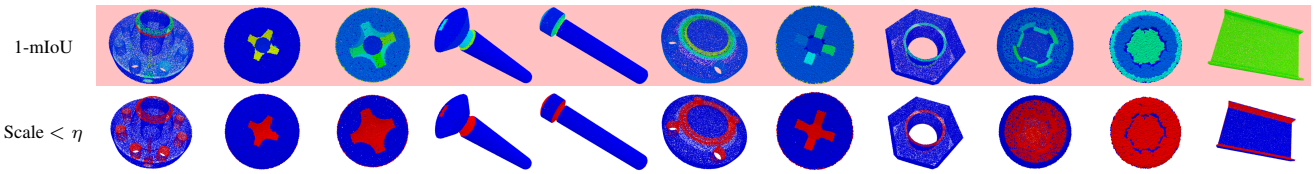

Figure S3. Comparison between the heatmap of global SPFN IoUs and the heatmap highlighting small primitive areas with $\eta = 5\%$. Both heatmaps highlight similar areas for most of the objects, showing that sampling patches from either heatmap will produce similar patch samples. Heatmaps are displayed with the *Jet* color map going from blue to green to red.

network is only used at test time when primitive information is not available. To improve on the initial global SPFN outputs, we train in turn our local SPFN (Section 3.2) on patches of $n = 8\,192$ points randomly samples in GT small primitive areas. The patches are sampled sequentially to cover as much as possible all small primitives. At each iteration, a new patch is sampled on small primitives from a random query point that has not yet been covered by any of the previous patches. We limit the maximum number of patches for a given object to 32. To provide both local and global context to this local SPFN, we augment the patch latent vector with the object latent vector $\mathbf{l}_o$ and the patch centroid's features $\mathbf{l}_i^g$, both extracted via the trained global SPFN.

S.12.6 CPFN Pipeline at Test Time

At test time, we run in parallel the global SPFN and the patch selection network to generate respectively the initial primitive decomposition and the small primitive heatmap. Contrary to the training, the global SPFN is tested on the high-resolution version of the point cloud. However, the patch selection network is still tested on the low-resolution point cloud. The heatmap - produced by the patch selection network - generates values in $[0, 1]$ for each of the $n = 8\,192$ points - higher values meaning higher chances for the point to be part of a small primitive. Then patches are randomly sampled in areas where the predicted value is above $\theta = 0.5$ so that all such points are covered by at least one patch. The local SPFN is then run on those newly sampled patches to refine the fitting on the smaller primitives. Finally, the segments from both the global and the local SPFNs are merged to produce the final primitive decomposition with improved performance on small primitives by following the process explained in Section 3.3 of the paper.

References

[1] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2020. 1, 2

[2] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. 4

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

[4] Sven Oesau, Yannick Verdie, Clément Jamin, Pierre Alliez, Florent Lafarge, and Simon Giraudot. Point set shape detection. In *CGAL User and Reference Manual*. CGAL Editorial Board, 4.13 edition, 2018. 3

[5] Charles Ruizhongtai Qi, Ly Yi, Hao Su, and Leonidas J. Guibas. PointNet++: Deep hierarchical feature learning on point sets in a metric space. In *NIPS*, 2017. 4

[6] TraceParts S.A.S. Traceparts. 3, 4

[7] Ruwen Schnabel, Roland Wahl, and Reinhard Klein. Efficient RANSAC for point-cloud shape detection. *Computer Graphics Forum*, 2007. 1, 2