Point2Cyl: Reverse Engineering 3D Objects from Point Clouds to Extrusion Cylinders Supplementary Material

Mikaela Angelina Uy*[1] Yen-Yu Chang*[1] Minhyuk Sung[2] Purvi Goel[1]
Joseph Lambourne[3] Tolga Birdal[1] Leonidas Guibas[1]
[1]Stanford University [2]KAIST [3]Autodesk Research

This document supplements our submission Point2Cyl: Reverse Engineering 3D Objects from Point Clouds to Extrusion Cylinders. In particular, we differentiate our work with traditional primitive fitting (Sec. S.1), provide additional details and proofs and robustness of theorems from the main paper (Sec. S.2), network details for our Point2Cyl (Sec. S.3), implementation details of the baselines (Sec. S.4), data preprocessing (Sec. S.5) and visualization post-processing details (Sec. S.6), and additional experiment comparisons, ablations and failure cases (Sec. S.7).

S.1. Extrusion Cylinders vs. Traditional Primitives

Our work focuses on decomposing an object into *extrusion cylinders*, which are very common building blocks in CAD design, covering $> 80\%/70\%$ of the faces in existing datasets [6, 7]. Extrusion cylinders are described by any arbitrary closed loop and hence do not assume fixed/regular geometry as in existing work that handle traditional ***discrete/fixed*** primitives. Thus, they are not detectable by existing traditional primitive fitting works. For comparison, we ran a pre-trained model of SPFN [8] on our test set, which resulted to a fitting loss of 0.1111, compared to our 0.0305 in the main paper. By our primitive definition, our method alone cannot be used to recover primitives such as spheres and cones. An immediate solution is to integrate our method with existing primitive fitting work, *e.g.* SPFN [8], to jointly handle diverse types of primitives. We also make the distinction that our work recovers primitive *volumes* instead of surfaces, from works such as SPFN [8] that requires further stitching and cannot be used directly in boolean operations. See illustration of a nut on the right in Fig. S5-b. In SPFN, the single extrusion will be represented with nine *surfaces*: six side planes, two top/bottom planes, and one inner cylinder.

S.2. Theorem Proofs and Robustness

S.2.1 Proof of Theorem 2

We start by a short summary of Theorem 2 of the main paper:

Theorem 2 (Weighted recovery of **extrusion axis** from points)**.** *For a general weighted point set, the optimal **extrusion axis** is given by* $\hat{\mathbf{e}} = \arg\min_{\mathbf{e},||\mathbf{e}||=1}(\mathbf{e}^\top \mathbf{H}_{\mathbf{\Phi}}\mathbf{e})$*, where:*

$$\mathbf{H}_{\mathbf{\Phi}} = \mathbf{N}^\top \mathbf{\Phi}_{barr}^\top \mathbf{\Phi}_{barr}\mathbf{N} - \mathbf{N}^\top \mathbf{\Phi}_{base}^\top \mathbf{\Phi}_{base}\mathbf{N}. \quad (1)$$

where $\mathbf{\Phi}_{barr} = \mathrm{diag}(\phi_{barr})$, $\mathbf{\Phi}_{base} = \mathrm{diag}(\phi_{base}) \in \mathbf{R}^{N\times N}$. ϕ_{barr}/ϕ_{base} *indicate the barrel/base weights assigned to all points, respectively.*

Proof. First, we use the observation in Eq. 1 of the main paper to formulate an objective $\mathcal{L}_{\mathbf{e}}$, whose minimum is attained at the point where $\mathbf{e}$ is the best fitting extrusion axis. Next, we show that $\mathcal{L}_{\mathbf{e}}$ can be re-organized absorbing the weights into surface normals. Finally, we re-arrange the result into matrix form to obtain $\mathbf{H}_{\mathbf{\Phi}}$. These steps read:

$$\begin{aligned}
\mathcal{L}_{\mathbf{e}} &= \sum_{j\in\text{barrel}} w_j^2(\mathbf{n}_j^\top\mathbf{e})^2 - \sum_{i\in\text{base}} w_i^2(\mathbf{n}_i^\top\mathbf{e})^2 && (2)\\
&= \sum_{j\in\text{barrel}} (w_j\mathbf{n}_j^\top\mathbf{e})^\top(w_j\mathbf{n}_j^\top\mathbf{e}) - \sum_{i\in\text{base}} (w_i\mathbf{n}_i^\top\mathbf{e})^\top(w_i\mathbf{n}_i^\top\mathbf{e})\\
&= \sum_{i=1}^{N} (\phi_i^{\text{barr}}\mathbf{n}_i^\top\mathbf{e})^\top(\phi_i^{\text{barr}}\mathbf{n}_i^\top\mathbf{e}) - (\phi_i^{\text{base}}\mathbf{n}_i^\top\mathbf{e})^\top(\phi_i^{\text{base}}\mathbf{n}_i^\top\mathbf{e})\\
&= \sum_{i=1}^{N} \mathbf{e}^\top(\mathbf{n}_i^\top\phi^{\text{barr}^\top}\phi_i^{\text{barr}}\mathbf{n}_i - \mathbf{n}_i^\top\phi^{\text{base}^\top}\phi_i^{\text{base}}\mathbf{n}_i)\,\mathbf{e} && (3)\\
&= \mathbf{e}^\top\mathbf{H}_{\mathbf{\Phi}}\mathbf{e} && (4)
\end{aligned}$$

and therefore $\arg\min_{\mathbf{e}} \mathcal{L}_{\mathbf{e}} = \arg\min_{\mathbf{e}} \mathbf{e}^\top\mathbf{H}_{\mathbf{\Phi}}\mathbf{e}$. Here, scalars w_i and w_j can be viewed as matrices of size 1×1. The second equality follows from gathering all the weights into ϕ such that $\phi_i^{\text{barr}} = 0$ whenever i indicates a base point,

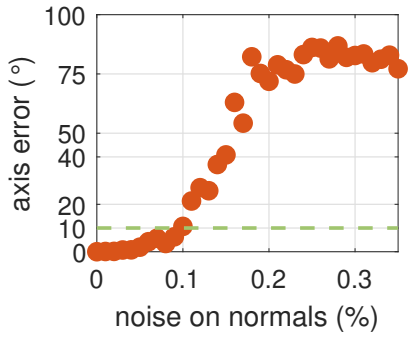

Figure S1. Robustness under noisy point normals.

and vice versa. We collect those into vectors $\boldsymbol{\phi}_{\text{barr}} = \{\phi_i^{\text{barr}}\}$, $\boldsymbol{\phi}_{\text{base}} = \{\phi_i^{\text{base}}\}$. Eq (2) clearly shows that the weights that control the contribution of each normal can be absorbed into the normals themselves. The last step in Eq (4) develops by virtue of this, *i.e.* we can define a *weighted normal i.e.* $\bar{\mathbf{n}}_i \triangleq \mathbf{n}_i \phi_i^{\text{barr}}$ and exploit Thm. 1 to write:

$$\mathbf{H}_{\boldsymbol{\Phi}} = (\bar{\mathbf{N}}_{\text{barr}}^\top \bar{\mathbf{N}}_{\text{barr}} - \bar{\mathbf{N}}_{\text{base}}^\top \bar{\mathbf{N}}_{\text{base}}) \tag{5}$$

$$= (\boldsymbol{\Phi}_{\text{barr}}\mathbf{N})^\top (\boldsymbol{\Phi}_{\text{barr}}\mathbf{N}) - (\boldsymbol{\Phi}_{\text{base}}\mathbf{N})^\top (\boldsymbol{\Phi}_{\text{base}}\mathbf{N}) \tag{6}$$

$$= \mathbf{N}^\top \boldsymbol{\Phi}_{\text{barr}}^\top \boldsymbol{\Phi}_{\text{barr}}\mathbf{N} - \mathbf{N}^\top \boldsymbol{\Phi}_{\text{base}}^\top \boldsymbol{\Phi}_{\text{base}}\mathbf{N}. \tag{7}$$

The second equality follows from individually weighting the normals using $\boldsymbol{\Phi}_{\text{barr}} = \text{diag}(\boldsymbol{\phi}_{\text{barr}}), \boldsymbol{\Phi}_{\text{base}} = \text{diag}(\boldsymbol{\phi}_{\text{base}}) \in \mathbf{R}^{N\times N}$. Note that this constructive approach also presents a perspective on the energy induced by our eigenvector problem. As $\|\mathbf{e}\| = 1$ is desired, the solution is given by the eigenvector corresponding to the smallest eigenvalue of $\mathbf{H}_{\boldsymbol{\Phi}}$. □

On a closer look, this approach resembles a graph partitioning where the cost is defined over the surface normals. We leave further analysis as a future work.

S.2.2 Proof of Theorem 3

Ideally, we would like to predict a minimal set of parameters for obtaining $\hat{\mathbf{M}}$, which contains $N \times 2K$ parameters. An obvious question arises when we consider the two outputs obtained through $\hat{\mathbf{M}}$: $\hat{\mathbf{W}}$ and $\hat{\mathbf{B}}$. In total these two matrices contain $N \times (K+2)$ unknowns. From this lens, it seems tempting to predict $\hat{\mathbf{W}}$ and $\hat{\mathbf{B}}$ instead. We now re-state Theorem 3 of the main paper arguing that knowing $\hat{\mathbf{W}}$ and $\hat{\mathbf{B}}$ is not sufficient to analytically compute $\hat{\mathbf{M}}$:

Theorem 3. *Matrix* $\hat{\mathbf{M}}$ *cannot be uniquely recovered from* $\hat{\mathbf{W}}$ *and* $\hat{\mathbf{B}}$.

Proof. We begin by assuming that recovering $\hat{M} \in \mathbb{R}^{N\times 2K}$ from $\hat{W} \in \mathbb{R}^{N\times K}$ and $\hat{B} \in \mathbb{R}^{N\times 2}$ would be possible. Hence, we assume the availability of $(\hat{W}, \hat{B})$, while $\hat{M}$ remains unknown and make use of the constraints at our disposal. First, we organize the relationship of $\hat{W}$ and $\hat{B}$ to $\hat{M}$ (as given in the main paper, Sec. 4.1) into matrix notation:

$$\hat{W} = \hat{M}\Delta_W \quad , \quad \hat{B} = \hat{M}\Delta_{\mathbf{B}} \tag{8}$$

where:

$$\Delta_W = \mathbf{I}_{K\times K} \otimes \begin{bmatrix}1\\1\end{bmatrix} \quad , \quad \Delta_B = \mathbf{1} \otimes \mathbf{I}_{2\times 2}. \tag{9}$$

$\mathbf{I}_{N\times N}$ denotes an $N \times N$ identity matrix and $\otimes$, the Kronecker product. Below, we show these matrices for the case of $K = 3$:

$$\Delta_W^{K=3} = \begin{bmatrix}1&0&0\\1&0&0\\0&1&0\\0&1&0\\0&0&1\\0&0&1\end{bmatrix} \quad , \quad \Delta_B^{K=3} = \begin{bmatrix}1&0\\0&1\\1&0\\0&1\\1&0\\0&1\end{bmatrix} \tag{10}$$

From the structure of $\mathbf{M}$, we also have $\mathbf{M1} = \mathbf{1}$. Note that, Δ_W and Δ_B are both non-square and hence non-invertible[1]. However, it is of interest to see whether these two constraints would be simultaneously sufficient to recover $\hat{\mathbf{M}}$. To this end, we convert the observations thusfar into the following constraints:

$$\|\hat{\mathbf{M}}\Delta_W - \hat{\mathbf{W}}\| \to \min \quad \|\hat{\mathbf{M}}\Delta_B - \hat{\mathbf{B}}\| \to \min\,. \tag{11}$$

To ensure those, we minimize a loss $\mathcal{L}_{\mathbf{M}}$:

$$\mathcal{L}_{\mathbf{M}} = \|\hat{\mathbf{M}}\Delta_W - \hat{\mathbf{W}}\|^2 + \|\hat{\mathbf{M}}\Delta_B - \hat{\mathbf{B}}\|^2 + \|\mathbf{M1} - \mathbf{1}'\|^2,$$

by letting $\nabla_{\mathbf{M}}\mathcal{L}_{\mathbf{M}} = 0$. After simple derivations and a re-arrangement, this yields:

$$\mathbf{M}(\Delta_W\Delta_W^\top + \Delta_B\Delta_B^\top + \mathbf{1}\mathbf{1}^\top) = \hat{\mathbf{W}}\Delta_W^\top + \hat{\mathbf{B}}\Delta_B^\top + \mathbf{1}'\mathbf{1}^\top$$

where $\mathbf{1}$ and $\mathbf{1}'$ denote one-vectors of different lengths. Note that the matrix $\mathbf{D} \triangleq (\Delta_W\Delta_W^\top + \Delta_B\Delta_B^\top + \mathbf{1}\mathbf{1}^\top)$ is known in advance and can be pre-computed. However, its rank will always be $\text{rank}(\mathbf{D}) = K + 1$ and therefore the number of linearly independent equations is insufficient to solve for $\mathbf{M}$. This concludes our proof that without further regularity, $\hat{\mathbf{M}}$ cannot be uniquely recovered from $\hat{\mathbf{W}}$ and $\hat{\mathbf{B}}$. □

S.2.3 Robustness under Noisy Point Normals

We include an experiment where the normals of the extrusion in Fig. 4 (main paper) is perturbed by an increasing amount of Gaussian noise. As shown on Fig. S1, our fit could tolerate (error $< 10°$) $\sim 10\%$ noise on the normals, which is a realistic setting.

[1] Using pseudoinverse does not give us the exact solution in this case.

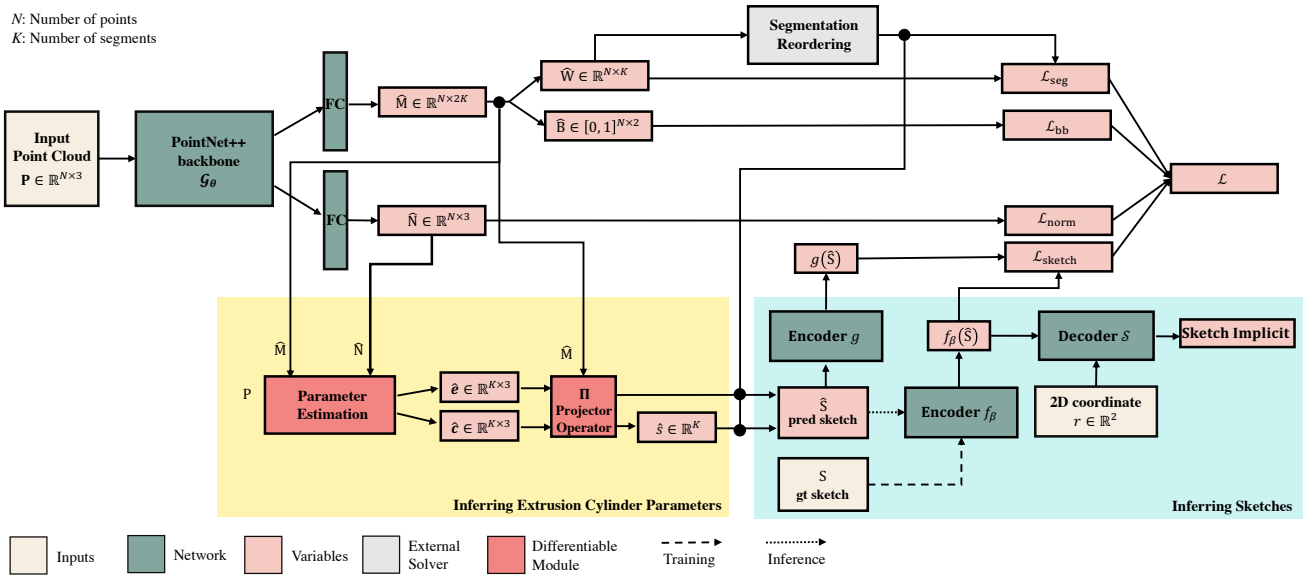

Figure S2. Network architecture of our Point2Cyl.

S.3. Additional Details for Our Point2Cyl

We provide additional implementation details for our Point2Cyl. Fig. S2 shows our overall pipeline. We randomly sample 8192 points for the point cloud inputs for networks taken from the underlying mesh for each model. We set $K = 8$ as the maximum number of extrusion segments and trained with a batch size of 4. When the number of extrusion cylinder segments in the ground truth label is less than K, the additional segments not matched with any of that in the ground truth are not included in the loss calculation. $\mathcal{S}$ was trained with normalized 2D point clouds with 2048 points, with a batch size of 8 with $\lambda_1 = 0.1, \lambda_2 = 1$. We use initial learning rate of 0.001 with a decay of 0.7. All models are trained for around 300 epochs or until convergence using the Adam optimizer.

We further clarify that our predicted extrusion cylinders can either be positive/negative volumes, and it can be inferred at post-processing. One can project (oriented) 3D point normals onto the sketch plane and compare it with the gradient of the predicted sketch implicit function to determine additive/subtractive cylinders.

S.4. Implementation Details of Baselines

S.4.1 Hough Voting for Extrusion Cylinders

While Hough voting and its generalized variants are used in simple primitive detection [1, 9, 10] or general object detection [2, 4, 11], their readily available adaptation to our problem remains unexplored. Hence, we create our Hough baseline by proposing a novel voting strategy for detection of extrusion cylinders.

In particular, we would like to model the *likelihood* of an extrusion axis given all the surface normals in the data. For an extrusion hypothesis, this reads:

$$p(\mathbf{e} \,|\, \mathbf{n}_1, \ldots, \mathbf{n}_N) = p(\mathbf{e}) \prod_{i=1}^{N} p(\mathbf{n}_i, \mathbf{e}) \tag{12}$$

$$= \frac{p(\mathbf{e}) \prod_{i=1}^{N} p(\mathbf{n}_i \,|\, \mathbf{e})}{p(\mathbf{n}_1, \ldots, \mathbf{n}_N)} \tag{13}$$

$$= \alpha \prod_{i=1}^{N} p(\mathbf{n}_i \,|\, \mathbf{e}). \tag{14}$$

The final equality follows from the assumption of a uniform prior and constant normalizing factor. Taking the logarithm of both sides, we obtain:

$$\log p(\mathbf{e} \,|\, \mathbf{n}_1, \ldots, \mathbf{n}_N) = \log \alpha + \sum_{i=1}^{N} \log p(\mathbf{n}_i \,|\, \mathbf{e}). \tag{15}$$

At this stage we propose to model $p(\mathbf{n}_i \,|\, \mathbf{e})$ as a *Gibbs measure*:

$$p(\mathbf{n}_i \,|\, \mathbf{e}) = \frac{1}{\beta} \exp(-\gamma v(\mathbf{e}, \mathbf{n}_i)). \tag{16}$$

with β being its normalizing constant. Plugging Eq (16) into Eq (15) leads to the Hough Transform for $\mathbf{e}$, $H(\mathbf{e}) \triangleq$

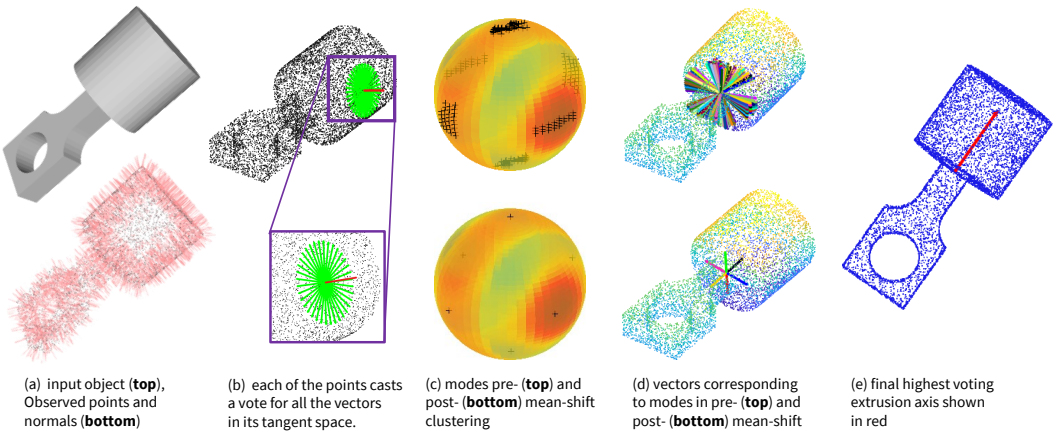

Figure S3. Steps of our Hough voting baseline for extrusion axis detection. Note that, our Point2Cyl algorithm can outperform this baseline. Nevertheless, it is still visible that this baseline can also tolerate clutter and confusing structures to a certain extent.

$\log p(\mathbf{e} \,|\, \mathbf{n}_1, \ldots, \mathbf{n}_N)$:

$$\log p(\mathbf{e} \,|\, \mathbf{n}_1, \ldots, \mathbf{n}_N) = \log \alpha - N \log \beta + \sum_{i=1}^{N} v(\mathbf{e}, \mathbf{n}_i)$$

$$H(\mathbf{e}) = c + \sum_{i=1}^{N} v(\mathbf{e}, \mathbf{n}_i) \tag{17}$$

where $c = \log \alpha - N \log \beta$ is a constant. Th is suggest that the *evidence* of an extrusion axis $\mathbf{e}$ can be obtained by interpreted as the sum of the *votes* cast per each normal $\mathbf{n}_i$. Note that, in practice, leaving c out leads to an *unnormalized log probability distribution* over the extrusion axis given surface normals, *i.e*. MAP (maximum a-posteriori) estimate trivializes to MLE (maximum likelihood estimation). This is what a Hough transform essentially computes.

The optionally *fuzzy* voting function $v(\mathbf{e}, \mathbf{n}_i)$ can be chosen to fit the geometric observation that the extrusion axis should lie in the tangent plane spanned by the surface normal, *i.e*. it has to be *normal to the normals*. With that, we can propose binary or non-binary voting functions:

$$v(\mathbf{e}, \mathbf{n}_i) \triangleq \delta(|\mathbf{e}_i^\top \mathbf{n}_i|) = \begin{cases} 0, & |\mathbf{e}_i^\top \mathbf{n}_i| < \epsilon \\ 1, & o.w. \end{cases} \tag{18}$$

At this point, one can obtain a point estimate by:

$$\mathbf{e}^\star = \underset{\mathbf{e} \in \mathbb{S}^2}{\arg\max}\, H(\mathbf{e}). \tag{19}$$

However, for the cases where multiple modes as well as noise are present, such approach would not yield a robust estimate. Therefore, we shift our attention to *mode seeking* and use the celebrated *mean-shift algorithm* [3] to discover the modes of the underlying distribution. To this end, in practice, for each point, we maintain a set of random vectors lying in its tangent space defined by the surface normal. This defines an empirical/discrete scalar field over the *Gaussian sphere* where each particle is a hypothesis for an extrusion axis. We then find the modes of this empirical distribution.

Different stages of our Hough voting baseline are shown in Fig. S3.

S.4.2 Direct Prediction (D.P.)

We also provide additional details for **D.P.**, a baseline neural network that takes in point cloud $\mathbf{P}$ and directly predicts the extrusion parameters without segmentation into generalized cylinders. Fig. S4 shows the network architecture for **D.P.**. We use a similar backbone as our network that encodes P into a latent feature, and then we directly predict K sets of extrusion parameters $(\hat{\mathbf{e}}, \hat{\mathbf{c}}, \hat{s}, g_{\text{DP}}(\hat{\mathbf{S}}_k))$ in four separate fully connected branches, where $g_{\text{DP}}(\hat{\mathbf{S}}_k)) \in \mathbb{R}^D$ is the latent code that conditions the sketch decoder $\mathcal{S}$. To train the network, we project the barrel points for each ground truth segment $\mathbf{P}^{\text{barrel}_k}$ using each of the directly predicted extrusion parameters, similar to $\mathcal{F}(\mathbf{P}, k)$ as introduced in Fit Cyl. in the Evaluation Metric section of the main paper. We use Hungarian matching on $\mathcal{F}(\mathbf{P}, k)$ to find correspondences with the ground truth extrusion parameters, and directly supervise each of the predicted parameters with their corresponding ground truth. We use the angular error (E.A.) for the extrusion axis, the distance/L2 loss for the extrusion center (E.C.), L1 for the extrusion scale, and $\mathcal{L}_{\text{sketch}}$ for g_{DP}.

S.5. Data Pre-Processing Details

We provide additional information on the data preprocessing used for both Fusion Gallery [12] and DeepCAD [13]. We used the Reconstruction subset for Fusion Gallery [12]. For clarification on the ABC dataset [6], DeepCAD dataset [13] is the subset of ABC dataset that collects the models constructed with sketch-extrude operations. Both datasets include json files that contain the step-by-step

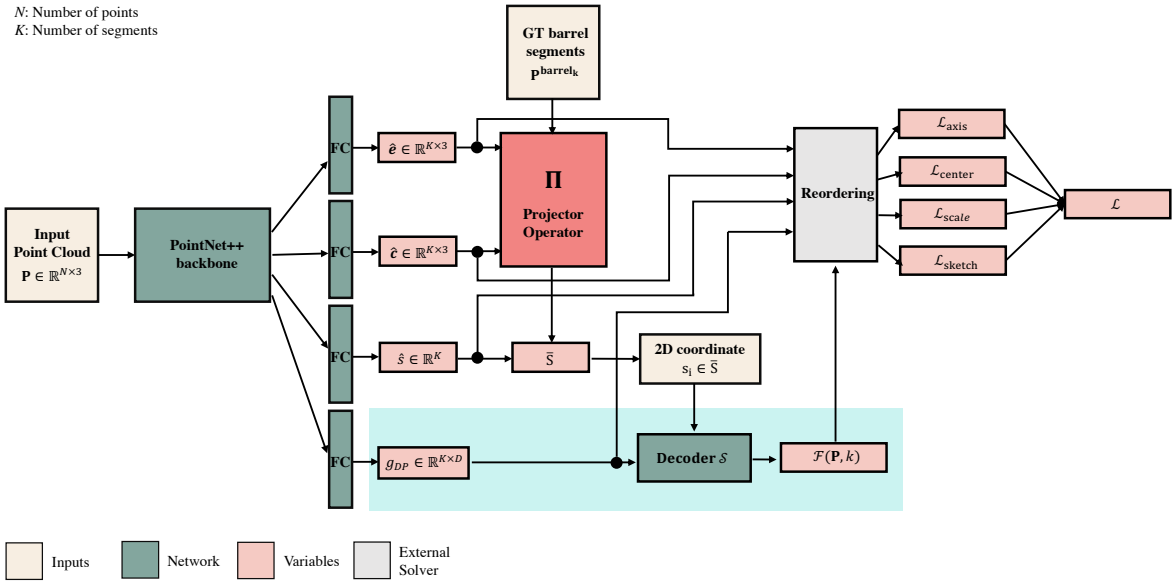

Figure S4. Network architecture of the direct prediction (D.P.) baseline.

construction sequence and the resulting triangulated output model, represented as a mesh. We obtain the extrusion cylinder segmentation labels for each face of a model by tracking which construction operation in the json file created each of mesh faces in the final geometry. For each construction operation of each model, we also extract the corresponding extrusion axis from the json file.

We uniformly sample 8192 points over each model's surface as our input point cloud along with each point's corresponding normal and segmentation label from its mesh face, as previously described. The point clouds are also normalized to fit a unit sphere. We classify each point as base/barrel by checking the dot product between its normal and the extrusion axis of the extrusion cylinder segment it belongs to. We obtain the ground truth extrusion center by taking the mean of all barrel points from each segment. We also represent the sketch of each segment of the model with a point cloud by sampling 8192 points along the barrel faces, and projecting them onto the plane perpendicular to the corresponding extrusion axis, centering them with the extrusion center, and finally normalizing to fit a unit circle.

We select a subset of all models that have $1-8$ extrusion cylinder segments. To balance our dataset, we also use a portion of the models with a single extrusion such that it only cover $\sim 20\%$ of our data. To increase the number of training/test models, we also use the intermediate models in each construction sequence. Moreover, we discard models that had tapered extrusions, an extrusion segment with surface area $< 2\%$ of the whole model, an extrusion segment that is too small whose extent is either too short (< 0.015) or had too few points (< 50). We will release our processed data upon publication.

S.6. Visualization Post-Processing Details

We also provide additional details on our post-processing step used for reconstruction refinement and the visualization of output models.

We first refine the segmentation output of our network as follows: i) Using the initial predicted segmentation, we use DBSCAN to cluster the points that belong to the same segment. If a segment results in more than one cluster, we unlabel the points belonging to the smaller clusters. ii) An unlabeled point is labeled with the consensus of its neighbors. iii) A point is relabeled if its neighbors have a high consensus with a different label, and the neighborhood consensus is used as the new label of the point.

We further use robust methods to estimate for scale and extent. To estimate for scale, we use RANSAC to randomly sample 1% of the barrel points for each extrusion segment, which are then used to estimate for the scale, as described in our main paper. The scale estimate of the segment is accepted if it explains more than 80% of all its barrel points. For extrusion extent, we use DBSCAN to cluster all the barrel points projected along the extrusion axis, and we calculate the extent based on the most dominant cluster.

Finally, we further optimize the sketch fitting by directly optimizing our sketch implicit network (modified from IGR [5] as described in the main paper), to directly fit the projected barrel points based on our network's output for each individual segment.

S.7. Additional Evaluations

S.7.1 Comparison with Conditional Generation Extension to DeepCAD [13]

We also analyze and compare our approach with a condition generation extension of DeepCAD [13] that is cast as a future work in their paper. DeepCAD [13] introduced a transformer-based generative model for CAD modeling sequences, similar to their proposed approach in their "Future Applications" section. We use the experimental code provided by the DeepCAD authors to encode point clouds using PointNet++ and map the resulting embeddings to the latent vector of their CAD sequence encoder from their original generative model. The PointNet++ encoder was trained for 100 epochs with batch size of 128 on the same DeepCAD training subset as Point2Cyl. We use the Adam optimizer with initial learning rate of 10^{-4} and decay the learning rate by 0.1 every 30 epochs.

At inference time, we use the Pointnet++ encoder to get the latent embedding of the point cloud and the CAD sequence decoder to obtain the reconstruction. We use the same DeepCAD test subset as for Point2Cyl. We used the released implementation for DeepCAD sequence reconstruction and found that the modeller fails to reconstruct 11.5% of the testing models from the output of the conditional DeepCAD generation. We report the fitting loss (Eq. 15 of the main paper) for DeepCAD (**0.0959**) vs. our model (**0.0758**) on the subset of models that [13] was able to successfully reconstruct. We take the midpoint of the tokenized outputs from [13] as the primitive parameters to obtain the corresponding extrusion cylinders that are used for loss computation. Fig. S6 shows some qualitative comparisons with reconstruction outputs from our Point2Cyl. These are examples where the DeepCAD reconstruction pipeline is able to output a solid. Results show that the outputs of DeepCAD does not always match the shape of the input geometry, and moreover, some examples show that DeepCAD often struggles to produce valid solid models in the output. In order to create a valid solid model, it is a requirement that the sketch profiles do not self-intersect. When self-intersecting profiles are extruded, the resulting solids will not define a closed and watertight volume. They will also fail the consistency checks of the solid modeling kernel and in some cases may fail to triangulate. Our Point2Cyl uses 2D implicits to define a sketch profile, and hence they will not intersect compared to DeepCAD, which can produce in self-intersecting geometry. Also, our Point2Cyl is trained to minimize the fitting error, while the encoding-decoding architecture of DeepCAD is not trained to make the output *fit* the input point cloud, and thus resulting in completely different shapes in some cases.

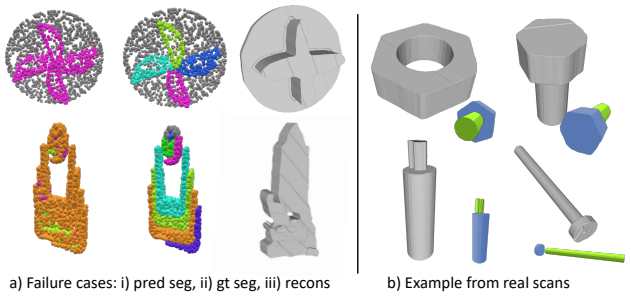

Figure S5. a) Failure cases. b) Real scan results.

S.7.2 Ablation on Noisy Data

We further experiment on adding noise to the input point clouds at both training and test time. We randomly perturb the points along the normal direction with a uniform noise between $[-\sigma, \sigma]$. Results are shown in Tab. S1, and we see that our Point2Cyl is able to tolerate noisy inputs without large performance drops. Experiments are on the Fusion Gallery dataset.

S.7.3 Ablation on Additional Loss Functions

We further ablate on adding additional loss functions. We experiment on adding additional loss functions that directly supervises for the extrusion axis (E.A.) and extrusion center (E.C.) based on the estimates from our parameter estimation module. Results are shown in Tab. S2. We see that adding these additional losses during training does not improve our performance. Experiments are on the Fusion Gallery dataset.

S.7.4 Failure Cases and examples on Real Scans

Fig. S5-a shows some examples of failure cases. Our approach is challenged by thinly separated extrusion cylinders (1st) and thin extrusion cylinders with few barrel points (2nd) resulting in poor reconstruction. Fig. S5-b shows examples of reconstructions from real scans of [8].

References

[1] Tolga Birdal, Benjamin Busam, Nassir Navab, Slobodan Ilic, and Peter Sturm. Generic primitive detection in point clouds using novel minimal quadric fits. *IEEE transactions on pattern analysis and machine intelligence*, 42(6):1333–1347, 2019. 3

[2] Tolga Birdal and Slobodan Ilic. Point pair features based object detection and pose estimation revisited. In *2015 International Conference on 3D Vision*, pages 527–535. IEEE, 2015. 3

[3] Yizong Cheng. Mean shift, mode seeking, and clustering. *IEEE transactions on pattern analysis and machine intelligence*, 17(8):790–799, 1995. 4

	σ	Seg.↑	Norm.(°) ↓	B.B.↑	E.A.(°) ↓	E.C.↓	Fit Cyl ↓	Fit Glob↓
H.V. + $\mathbf{N_J}$		0.409	12.264	0.595	58.868	0.1248	0.1492	0.0683
D.P.		-	-	-	30.147	0.1426	1.4132	0.4257
w/o $\mathcal{L}_{\text{sketch}} + \mathbf{N_J}$	0.00	0.699	12.264	**0.913**	14.169	0.0729	0.0828	0.0330
w/o $\mathcal{L}_{\text{sketch}}$		0.699	8.747	**0.913**	9.795	0.0727	0.0826	0.0352
Ours (**Point2Cyl**)		**0.736**	**8.547**	0.911	**8.137**	**0.0525**	**0.0704**	**0.0305**
H.V. + $\mathbf{N_J}$		0.395	13.894	0.594	58.907	0.1263	0.1662	0.0813
D.P.		-	-	-	38.786	0.1392	2.1174	0.7815
w/o $\mathcal{L}_{\text{sketch}} + \mathbf{N_J}$	0.01	0.625	13.894	0.869	16.544	0.0914	0.1100	0.0567
w/o $\mathcal{L}_{\text{sketch}}$		0.631	11.901	0.869	14.542	0.0916	0.1079	**0.0565**
Ours (**Point2Cyl**)		**0.698**	**10.347**	**0.897**	**11.629**	**0.0760**	**0.0967**	0.0606
H.V. + $\mathbf{N_J}$		0.398	17.617	0.590	59.327	0.1267	0.1780	0.0902
D.P.		-	-	-	45.907	0.1805	2.1277	1.6262
w/o $\mathcal{L}_{\text{sketch}} + \mathbf{N_J}$	0.02	0.659	17.617	0.889	14.976	0.0820	0.1077	0.0600
w/o $\mathcal{L}_{\text{sketch}}$		0.674	12.816	0.879	**11.553**	0.0796	0.1074	**0.0595**
Ours (**Point2Cyl**)		**0.679**	**12.691**	**0.891**	11.934	**0.0755**	**0.1056**	0.0621

Table S1. Experiment on noisy point clouds. $\sigma = 0.00$ corresponds to the results reported in our main paper.

	Seg.↑	Norm.(°) ↓	B.B.↑	E.A.(°) ↓	E.C.↓	Fit Cyl ↓	Fit Glob↓
w/ E.A. supervision	0.648	10.702	0.898	8.901	0.0735	0.1003	0.0490
w/ E.C. supervision	0.672	8.990	0.904	10.147	0.0733	0.0985	0.0556
w/ E.A. & E.C. supervision	0.633	10.508	0.897	9.166	0.0824	0.1069	0.0560
Ours (**Point2Cyl**)	**0.736**	**8.547**	**0.911**	**8.137**	**0.0525**	**0.0704**	**0.0305**

Table S2. Experiment on adding additional direct supervision losses during training.

[4] Bertram Drost, Markus Ulrich, Nassir Navab, and Slobodan Ilic. Model globally, match locally: Efficient and robust 3d object recognition. In *2010 IEEE computer society conference on computer vision and pattern recognition*, pages 998–1005. Ieee, 2010. 3

[5] Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. Implicit geometric regularization for learning shapes. In *ICML*. 2020. 5

[6] 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 Conference on Computer Vision and Pattern Recognition*, pages 9601–9611, 2019. 1, 4

[7] Joseph G. Lambourne, Karl D. D. Willis, Pradeep Kumar Jayaraman, Aditya Sanghi, Peter Meltzer, and Hooman Shayani. Brepnet: A topological message passing system for solid models. In *CVPR*, 2021. 1

[8] Lingxiao Li, Minhyuk Sung, Anastasia Dubrovina, Li Yi, and Leonidas Guibas. Supervised fitting of geometric primitives to 3d point clouds. In *CVPR*, 2019. 1, 6

[9] Ruwen Schnabel, Roland Wahl, and Reinhard Klein. Efficient ransac for point-cloud shape detection. In *Computer graphics forum*, 2007. 3

[10] Christiane Sommer, Yumin Sun, Erik Bylow, and Daniel Cremers. Primitect: Fast continuous hough voting for primitive detection. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8404–8410. IEEE, 2020. 3

[11] Alykhan Tejani, Danhang Tang, Rigas Kouskouridas, and Tae-Kyun Kim. Latent-class hough forests for 3d object detection and pose estimation. In *European Conference on Computer Vision*, pages 462–477. Springer, 2014. 3

[12] Karl D. D. Willis, Yewen Pu, Jieliang Luo, Hang Chu, Tao Du, Joseph G. Lambourne, Armando Solar-Lezama, and Wojciech Matusik. Fusion 360 gallery: A dataset and environment for programmatic cad reconstruction. *arXiv preprint arXiv:2010.02392*, 2020. 4

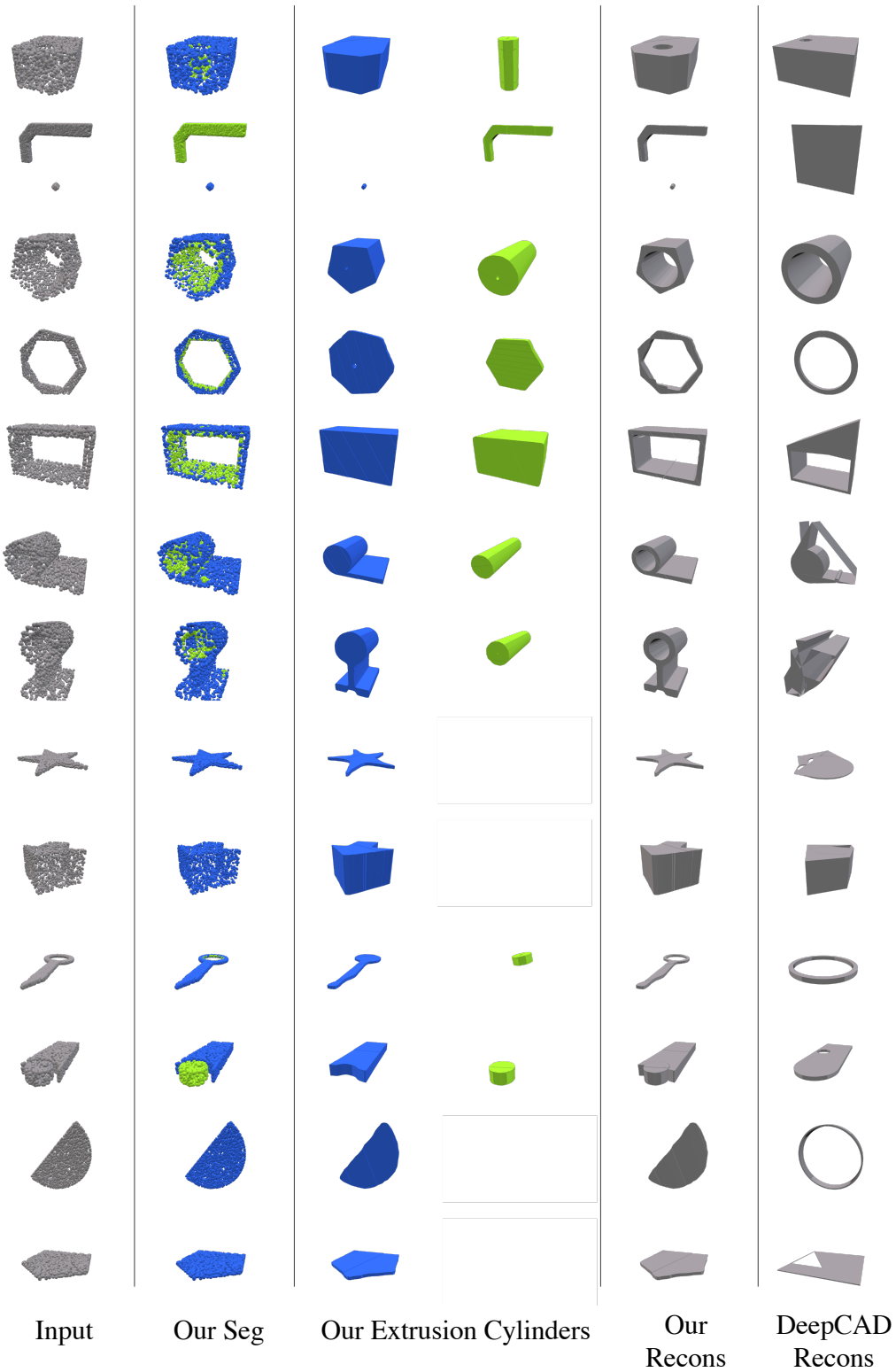

Figure S6. Additional qualitative examples from our Point2Cyl on the DeepCAD dataset. We also show comparisons with the conditional generation extension to DeepCAD [13] and show that our approach result in output models that better match the input.

[13] Rundi Wu, Chang Xiao, and Changxi Zheng. Deepcad: A deep generative network for computer-aided design models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2021. 4, 6, 8