PI3DETR: Parametric Instance Detection of 3D Point Cloud Edges With a Geometry-Aware 3DETR

Fabio F. Oberweger* Michael Schwingshackl* Vanessa Staderini
AIT Austrian Institute of Technology
Center for Vision, Automation & Control
{fabio.oberweger, michael.schwingshackl, vanessa.staderini}@ait.ac.at

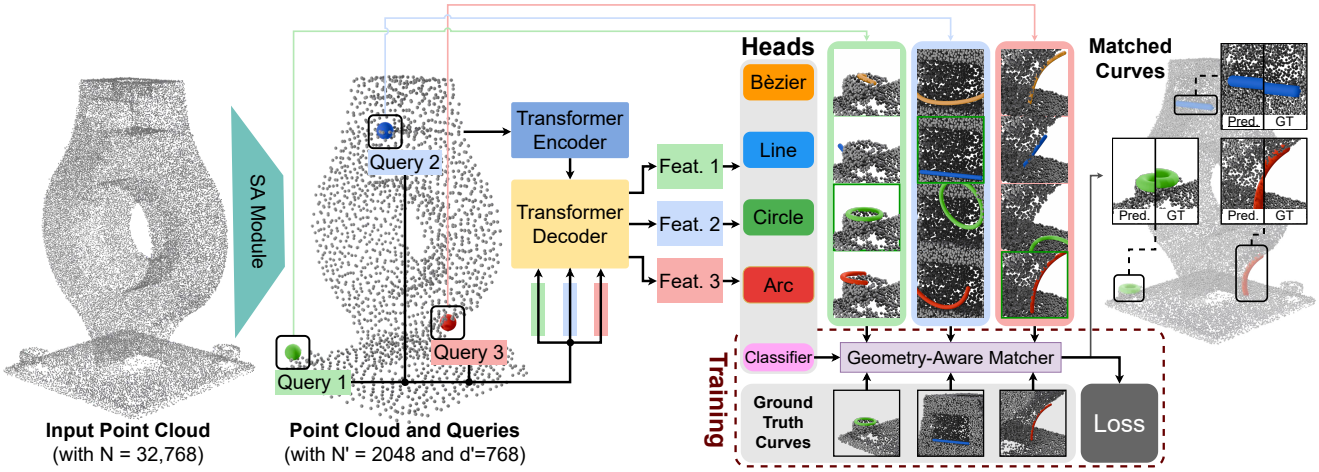


Figure 1. PI3DETR is an end-to-end pipeline that takes a point cloud as input. Like 3DETR [14], it uses an SAModule [16] to sample points and generate queries, and a Transformer [21] to extract query-specific features. Multiple heads predict parametric curves (cubic Bézier, line segments, circle, arc) with associated parameters, while a geometry-aware matcher aligns predictions with ground truth, removing the need for intermediate representations or post-processing. This visualization shows three sample queries for clarity.

Abstract

We present PI3DETR, an end-to-end framework that directly predicts 3D parametric curve instances from raw point clouds, avoiding the intermediate representations and multi-stage processing common in prior work. Extending 3DETR, our model introduces a geometry-aware matching strategy and specialized loss functions that enable unified detection of differently parameterized curve types, including cubic Bézier curves, line segments, circles, and arcs, in a single forward pass. Optional post-processing steps further refine predictions without adding complexity. This streamlined design improves robustness to noise and varying sampling densities, addressing critical challenges in real world LiDAR and 3D sensing scenarios. PI3DETR sets a new state-of-the-art on the ABC dataset and generalizes effectively to real sensor data, offering a simple yet powerful solution for 3D edge and curve estimation. Code: https://github.com/fafraob/pi3detr.

1. Introduction

Estimating 3D edges in point clouds is not only a compelling theoretical challenge but also a task with significant practical impact. In robotic vision, for example, edge information can be used to optimize grasp planning by determining precisely how and where an object should be gripped. In manufacturing, wireframe extraction enables more effective quality control, while in other domains, such as autonomous driving, CAD reconstruction, and various 3D data processing pipelines, edges serve as essential structural cues. In many of these applications, LiDAR (light detection and ranging) and other 3D sensing technologies are employed

*Equal contribution.

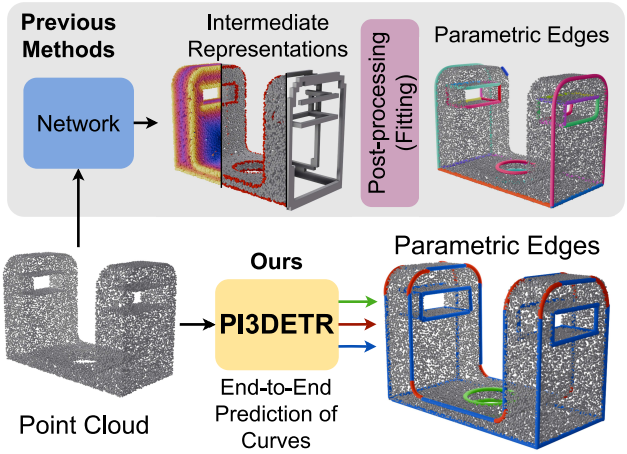


Figure 2. Unlike methods that rely on intermediate representations (distance fields, segmentations, voxels) and post-fitting, PI3DETR directly predicts parametric edge curves in an end-to-end manner.

to capture complex objects and environments. However, the resulting point clouds are rarely flawless as they often contain noise, exhibit non-uniform sampling densities, and suffer from occlusion-induced gaps. These imperfections make the development of robust and adaptable edge detection methods critical for ensuring reliable performance across diverse real-world scenarios.

Hence, we introduce PI3DETR, a streamlined end-to-end architecture that directly predicts parametric curves from raw point clouds, avoiding the complexity and modularity inherent in prior approaches. Existing methods often rely on intermediate representations such as voxel grids [25], range images [13], or 2D sketches [22], which introduce additional processing steps and impose assumptions on the input space. Others incorporate post-processing stages to fit curves to precomputed segmentations [23] or piece-wise linear primitives [25], or construct curve graphs using multi branch decoders [6]. In contrast, PI3DETR employs a unified architecture that directly infers parametric curve instances from point clouds, eliminating auxiliary representations, handcrafted pipelines, and multi stage reasoning. As illustrated in Fig. 2, our approach simplifies the overall pipeline while expanding capability. To achieve this, we adapt the 3DETR framework [13] and extend it with a novel geometry-aware matching strategy that enables joint detection of multiple curve types including cubic Bézier curves, line segments, circles, and arcs in a single forward pass. We further introduce tailored loss functions specifically designed for 3D edge estimation and propose optional post-processing steps that yield additional performance gains without being integral to the pipeline.

Another important aspect of our work is robustness to variations in point cloud sampling and noise. Both, our experiments and prior studies [23, 25], show that performance can degrade substantially when sampling densities change or noise is added. We hypothesize that our simplified end-to-end architecture enhances robustness by eliminating the dependency on intermediate representations or highly parameterized post-processing methods.

We evaluate PI3DETR on the ABC dataset [9] and compare it to the state-of-the-art approach NerVE [25]. Our results show that PI3DETR surpasses NerVE both qualitatively and quantitatively on this challenging benchmark. Furthermore, we collect a small dataset of real world sensor data to demonstrate the practical applicability of our method. Our main contributions are:

- PI3DETR, a streamlined architecture that directly predicts 3D curve instances with different parameterizations from raw point clouds.
- A geometry-aware prediction and matching strategy that enables detection of cubic Bézier curves, line segments, circles, and arcs within a single model.
- A simple yet effective optional post-processing procedure for parametric curve prediction.
- State-of-the-art performance on the ABC dataset for 3D parametric curve detection.

2. Related Work

As noted by Zhu *et al*. [25], a common strategy for parametric curve reconstruction is to first detect geometric primitives such as corners and edges and then then fit curves in a separate post-processing stage [6, 11, 13, 23]. PIE-NET [23] follows this paradigm using a PointNet++ [16] backbone to classify points into corner, edge, or background classes, after which a secondary network predicts curve proposals between detected corner pairs and estimates their parameters. The method supports lines, circles, and B-splines, and is trained on the ABC dataset [9].

EC-Net [24] addresses a related problem of edge-aware point cloud upsampling by jointly learning point consolidation and edge preservation. It predicts an edge confidence score to guide new point placement along sharp features, producing denser point clouds with crisper edges and corners compared to prior upsampling methods.

ComplexGen [6] discretizes point clouds into sparse voxels and applies a 3D CNN [4] to extract features, which are decoded via three DETR [3] decoder modules for vertices, edges, and faces. The predictions are then assembled into a CAD model by solving an integer linear program.

DEF [13] processes dense point clouds via a CNN U-Net [17] on range images to predict per point distance-to-feature fields. After thresholding, it estimates corners, segments curves, and constructs a connectivity graph that is optimized to produce parametric lines, circles, and B-splines.

Unlike keypoint based approaches, NerVE [25] performs volumetric edge detection by combining PointNet++ with a 3D CNN [4] to produce a voxel feature grid from the input

point cloud. Three encoder branches predict voxel occupancy by edge points, their 3D coordinates, and their connectivity to neighbors. From these predictions, piece-wise linear curves (PWLs) are assembled and refined through a dedicated post-processing step to form structured boundary representations.

Since the introduction of DETR [3] for 2D object detection, its transformer-based architecture with set prediction has been adapted to 3D tasks. 3DETR [14] extends DETR to 3D bounding box detection directly on point clouds, using a set aggregation module [16] for downsampling followed by Transformer [21] encoder and decoder blocks. In contrast to DETR's learned queries, 3DETR uses actual point locations as queries, grounding predictions in the input geometry.

Point2Primitive [22] addresses curve extraction as part of CAD reconstruction. It applies PointNet++ to cluster extrusion regions, then uses an adapted DETR to detect 2D sketch primitives such as lines, circles, and arcs within each extrusion. These sketches are combined with extrusion operations to reconstruct complete CAD models.

3. Method

PI3DETR takes a 3D point cloud as input and predicts 3D edges in the form of different curve types in a parameterized way. In the following, we describe the training data, the parameterization, and the model architecture.

3.1. Dataset and Ground Truth

The ABC dataset [9] is a standard benchmark for parametric edge detection [13, 23, 25] and CAD reconstruction [6, 22], providing paired CAD and mesh representations. Following prior work [13, 23, 25], we focus on sharp edges in the CAD models and apply the filtering protocol in Appendix A to remove erroneous or inconsistent entries. We further retain only models whose curves can be represented as cubic Bézier curves, line segments, circles, or arcs, and contain at most 100 curves. This yields 10,240 models, split into 80% training, 10% validation, and 10% test. To create our 3D point cloud dataset $\mathcal{P}$, we sample 32,768 surface points for each mesh in the remaining dataset. Triangle point picking [5] is used as the sampling method to promote generalization, in contrast to approaches that primarily rely on mesh vertices [25].

Parametrization and ground truth definition. Let $P \in \mathcal{P}$ denote an arbitrary point cloud from the dataset, and let M be the number of parametric curves associated with P. To adapt the curve representations of the CAD model underlying P for the learning task, we extract curve vertices and fit parametric primitives to each segment.

We denote each ground truth curve in P by an index $i \in \{1, \dots, M\}$. Each curve is assigned a class label

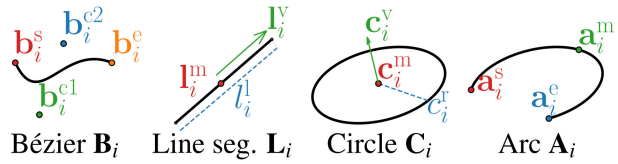


Figure 3. Parametric representations of the four curve types used for prediction: Bézier (B_i), Line Segment (L_i), Circle (C_i), and Arc (A_i).

$c_i \in \{1, 2, 3, 4\}$ indicating its type: 1 for a cubic Bézier curve, 2 for a line segment, 3 for a circle, and 4 for an arc. The parameters for each curve type are defined as follows, and are illustrated in Fig. 3: A cubic Bézier curve $\mathbf{B}_i = [\mathbf{b}_i^{\mathrm{s}}, \mathbf{b}_i^{\mathrm{c1}}, \mathbf{b}_i^{\mathrm{c2}}, \mathbf{b}_i^{\mathrm{e}}]$ ($c_i = 1$) is defined by four ordered control points $\mathbf{b}_i^{\mathrm{s}}, \mathbf{b}_i^{\mathrm{c1}}, \mathbf{b}_i^{\mathrm{c2}}, \mathbf{b}_i^{\mathrm{e}} \in \mathbb{R}^3$, corresponding to the start point, two control points, and the end point, respectively. A line segment $\mathbf{L}_i = [\mathbf{l}_i^{\mathrm{m}}, \mathbf{l}_i^{\mathrm{v}}, l_i^{\mathrm{r}}]$ ($c_i = 2$) is parameterized by a mid-point $\mathbf{l}_i^{\mathrm{m}} \in \mathbb{R}^3$, a unit direction vector $\mathbf{l}_i^{\mathrm{v}} \in \mathbb{R}^3$, and a length $l_i^{\mathrm{r}} \in \mathbb{R}^+$. A circle $\mathbf{C}_i = [\mathbf{c}_i^{\mathrm{m}}, \mathbf{c}_i^{\mathrm{v}}, c_i^{\mathrm{r}}]$ ($c_i = 3$) is defined by its mid-point $\mathbf{c}_i^{\mathrm{m}} \in \mathbb{R}^3$, a unit normal vector $\mathbf{c}_i^{\mathrm{v}} \in \mathbb{R}^3$ with $\|\mathbf{c}_i^{\mathrm{v}}\| = 1$, and a radius $c_i^{\mathrm{r}} \in \mathbb{R}^+$. Finally, an arc $\mathbf{A}_i = [\mathbf{a}_i^{\mathrm{s}}, \mathbf{a}_i^{\mathrm{m}}, \mathbf{a}_i^{\mathrm{e}}]$ ($c_i = 4$) is specified by three ordered points $\mathbf{a}_i^{\mathrm{s}}, \mathbf{a}_i^{\mathrm{m}}, \mathbf{a}_i^{\mathrm{e}} \in \mathbb{R}^3$, denoting the arc's start point, mid-point, and end point.

All curve parameters and input point clouds are uniformly normalized with respect to the longest axis of the mesh bounding box, preserving the geometric proportions of the shapes (e.g., preventing circles from being distorted into ellipses). After normalization, the point clouds, the point-, and vector-based curve parameters lie within $[-1, 1]^3$.

3.2. PI3DETR Overview

PI3DETR builds upon 3DETR [14] and extends its functionality to predict geometric objects of different shapes, in our case parametric curves, by incorporating a geometry-aware matching framework. The main differences from 3DETR lie in the design of the prediction heads, the loss functions, and the matching strategy, each of which is discussed in the corresponding paragraphs.

3DETR core. The input to PI3DETR is a point cloud $P \subset \mathbb{R}^3$ of N points with 3D coordinates only (no normals or colors). First, we use a set-aggregation module (SAModule) [16] with farthest point sampling (FPS) [16] to downsample to N' points, ensuring even spatial coverage. For each downsampled point, the SAModule extracts a d-dimensional local feature, which is linearly projected to d' and processed by Transformer encoder–decoder blocks [21]. Unlike 3DETR, we apply layer normalization *after* each block's operations instead of using pre-norm [8]. The encoder produces N' point features, which the decoder pro-

cesses together with K query embeddings $\{q_j^{\text{f}}\}_{j=1}^K$ to produce K output embeddings $\{o_j^{\text{f}}\}_{j=1}^K$ for curve prediction. Following Misra *et al*. [14], we use non-parametric queries obtained by FPS from the input cloud. Each query point q_j is encoded via sine positional encoding [21] and projected with a feed-forward network (FFN) to form q_j^{f}. The same positional encoding is applied to encoder outputs before the decoder, preserving spatial information lost during feature extraction.

Prediction feed-forward networks (FFNs). Following the parameterization in Sec. 3.1, we distinguish between two prediction types: 3D point regression (control points, centers, endpoints) and scalar regression (length, radius). Each prediction head is implemented as a FFN parameterized by $\boldsymbol{\theta}^{(\cdot)}$ and takes a decoder output embedding $o_j^{\text{f}} \in \{o_j^{\text{f}}\}_{j=1}^K$ as input.

For Bézier curves, $\boldsymbol{\theta}^{\mathbf{B}}$ maps o_j^{f} to four control points $\hat{\mathbf{B}}_j = [\hat{\mathbf{b}}_j^{\text{s}}, \hat{\mathbf{b}}_j^{\text{c1}}, \hat{\mathbf{b}}_j^{\text{c2}}, \hat{\mathbf{b}}_j^{\text{e}}] \in \mathbb{R}^{4\times 3}$. For line segments, $\boldsymbol{\theta}^{\mathbf{L}}$ outputs the midpoint $\hat{\mathbf{l}}_j^{\text{m}} \in \mathbb{R}^3$ and unit direction $\hat{\mathbf{l}}_j^{\text{v}} \in \mathbb{R}^3$, while $\boldsymbol{\theta}^{\ell}$ predicts the length $\hat{l}_j^{\text{r}} \in \mathbb{R}$. For circles, $\boldsymbol{\theta}^{\mathbf{C}}$ returns the center $\hat{\mathbf{c}}_j^{\text{m}} \in \mathbb{R}^3$ and unit normal $\hat{\mathbf{c}}_j^{\text{v}} \in \mathbb{R}^3$, and $\boldsymbol{\theta}^{r}$ estimates the radius $\hat{c}_j^{\text{r}} \in \mathbb{R}$. We summarize the line segment and circle predictions as $\hat{\mathbf{L}}_j = [\hat{\mathbf{l}}_j^{\text{m}}, \hat{\mathbf{l}}_j^{\text{v}}, \hat{l}_j^{\text{r}}]$ and $\hat{\mathbf{C}}_j = [\hat{\mathbf{c}}_j^{\text{m}}, \hat{\mathbf{c}}_j^{\text{v}}, \hat{c}_j^{\text{r}}]$, respectively. For arcs, $\boldsymbol{\theta}^{\mathbf{A}}$ predicts the start point, mid-point, and end point $\hat{\mathbf{A}}_j = [\hat{\mathbf{a}}_j^{\text{s}}, \hat{\mathbf{a}}_j^{\text{m}}, \hat{\mathbf{a}}_j^{\text{e}}] \in \mathbb{R}^{3\times 3}$. A classification head $\boldsymbol{\theta}^{\text{cls}}$ maps o_j^{f} to $\hat{p}_j \in [0,1]^5$ over the set {no-object, cubic Bézier, line segment, circle, arc}, selecting the curve type at inference. PI3DETR computes parameters for all curve types from every o_j^{f}, while a geometry-aware matcher assigns gradients only to the heads corresponding to the ground-truth type during training.

For simplicity, we describe all positions as if they were predicted directly. In practice, each point is obtained as an offset from its associated query location q_j, which stabilizes learning and improves localization accuracy.

Geometry-aware matching. We match predicted 3D curves to the ground truth using a cost-minimal bipartite assignment. Each decoder output $o_j^{\text{f}} \in \{o_j^{\text{f}}\}_{j=1}^K$ predicts parameters for all four curve types, producing $4 \cdot K$ curve hypotheses. For a prediction $j \in \{1, \dots, K\}$ and ground truth $i \in \{1, \dots, M\}$, the matching cost is

$$\mathcal{C}_{\text{match}}(j,i) = -\hat{p}_j(c_i) + \mathcal{L}_{\text{param}}(j,i), \tag{1}$$

where $-\hat{p}_j(c_i)$ is the negative class probability of query j being class c_i [3] and $\mathcal{L}_{\text{param}}$ is a geometry-specific parameter loss:

$$\begin{aligned}\mathcal{L}_{\text{param}}(j,i) = \\ \mathbb{1}_{\{c_i=1\}}\mathcal{L}_{\text{seq}}(\hat{\mathbf{B}}_j, \mathbf{B}_i) + \mathbb{1}_{\{c_i=2\}}\mathcal{L}_{\text{hybrid}}(\hat{\mathbf{L}}_j, \mathbf{L}_i) + \\ \mathbb{1}_{\{c_i=3\}}\mathcal{L}_{\text{hybrid}}(\hat{\mathbf{C}}_j, \mathbf{C}_i,) + \mathbb{1}_{\{c_i=4\}}\mathcal{L}_{\text{seq}}(\hat{\mathbf{A}}_j, \mathbf{A}_i).\end{aligned} \tag{2}$$

For cubic Bézier curves and arcs parameterized as ordered point sequences, we adopt an order-invariant ℓ_1-loss [7] to account for direction ambiguity:

$$\mathcal{L}_{\text{seq}}(\hat{\mathbf{X}}, \mathbf{X}) = \min\{\|\hat{\mathbf{X}} - \mathbf{X}\|_1,\ \|\text{rev}(\hat{\mathbf{X}}) - \mathbf{X}\|_1\}, \tag{3}$$

Here, $\text{rev}(\cdot)$ reverses the point order. For example, $\text{rev}(\hat{\mathbf{B}}) = [\hat{\mathbf{b}}^{\text{e}}, \hat{\mathbf{b}}^{\text{c2}}, \hat{\mathbf{b}}^{\text{c1}}, \hat{\mathbf{b}}^{\text{s}}]$.

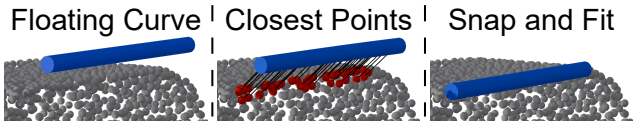
Lines $\mathbf{L} = [\mathbf{l}^m, \mathbf{l}^v, l^r]$ and circles $\mathbf{C} = [\mathbf{c}^m, \mathbf{c}^v, c^r]$ share a hybrid point–vector–scalar representation (mid-point/center, direction/normal, length/radius). The vector is sign-ambiguous, so we evaluate both orientations for this loss:

$$\begin{aligned}\mathcal{L}_{\text{hybrid}}(\hat{\mathbf{X}}, \mathbf{X}) = \min\{\|(\hat{\mathbf{x}}^{\text{m}}, \hat{\mathbf{x}}^{\text{v}}) - (\mathbf{x}^{\text{m}}, \mathbf{x}^{\text{v}})\|_1, \\ \|(\hat{\mathbf{x}}^{\text{m}}, \hat{\mathbf{x}}^{\text{v}}) - (\mathbf{x}^{\text{m}}, -\mathbf{x}^{\text{v}})\|_1\} + \|\hat{x}^{\text{r}} - x^{\text{r}}\|_1\,.\end{aligned} \tag{4}$$

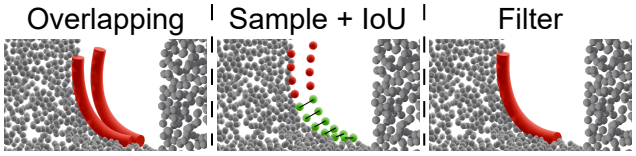
Following [3, 14], we solve for the minimal-cost permutation $\sigma \in \mathfrak{S}_K$ via the Hungarian algorithm [10], assigning each $i \in \{1, \dots, M\}$ to a unique $j = \sigma(i)$. The remaining $K - M$ predictions are matched to no-object entries ($c_{\sigma(i)} = 0$). More details regarding the matching are provided in B.1.

Loss function. Let $i \in \{1, \dots, K\}$ be the index of a ground truth curve from the extended list of ground truth curves that includes the no-object class, and let $j = \sigma(i) \in \{1, \dots, K\}$ be the index of a prediction that was matched with i in the matcher. The total loss for such a pair (j,i) is defined as

$$\begin{aligned}\mathcal{L}_{\text{total}}(j,i) = \mathcal{L}_{\text{CE}}(\hat{p}_j, c_i) + \mathbb{1}_{\{c_i \neq 0\}}\mathcal{L}_{\text{param}}(j,i) + \\ \mathbb{1}_{\{c_i \neq 0\}}\mathcal{L}_{\text{CD}}(S_j^{c_i}, S_i^{c_i}),\end{aligned} \tag{5}$$



(a) Snap & Fit (S&F): realigning slightly offset curves by finding the closest points in the point cloud and refitting the curve.



(b) IoU Filter: removing overlapping or nearby curves by computing the IoU of overlapping points and discarding the curve with lower confidence.

Figure 4. Optional post-processing steps: (a) S&F, (b) IoU Filter.

where $\mathcal{L}_{\text{CE}}(\hat{p}_j, c_i)$ is the cross-entropy loss between the predicted class probabilities $\hat{p}_j$ and ground truth class c_i, $\mathcal{L}_{\text{param}}$ is the geometry-aware parametric curve loss, and $\mathcal{L}_{\text{CD}}$ is the Chamfer distance (CD) loss. Here, $S_j^{c_i}$ and $S_i^{c_i}$ denote the 64 uniformly sampled 3D points from the curves of class c_i using the respective parameters of j and i.

Predictions matched to a placeholder ground truth instance with $c_i = 0$ contribute only to the class prediction branch through the cross entropy loss. We apply auxiliary losses after each decoder layer as in [3, 14] and since our dataset exhibits a class imbalance among curve types, we use class weights in the cross entropy loss. The weight computation and details of the CD loss are provided in Appendix B.3 and for completeness, CD is defined in Appendix C.

Optional post-processing. We optionally apply two post-processing steps to refine PI3DETR predictions and suppress duplicates, with their impact analyzed in the ablation study (Sec. 4.3). Snap & Fit (S&F) snaps predicted curves to the point cloud and refits them using its predicted class (Fig. 4a), while an IoU Filter removes overlapping curves of the same class based on sampled-point IoU and confidence (Fig. 4b). More details are provided in Appendix B.2.

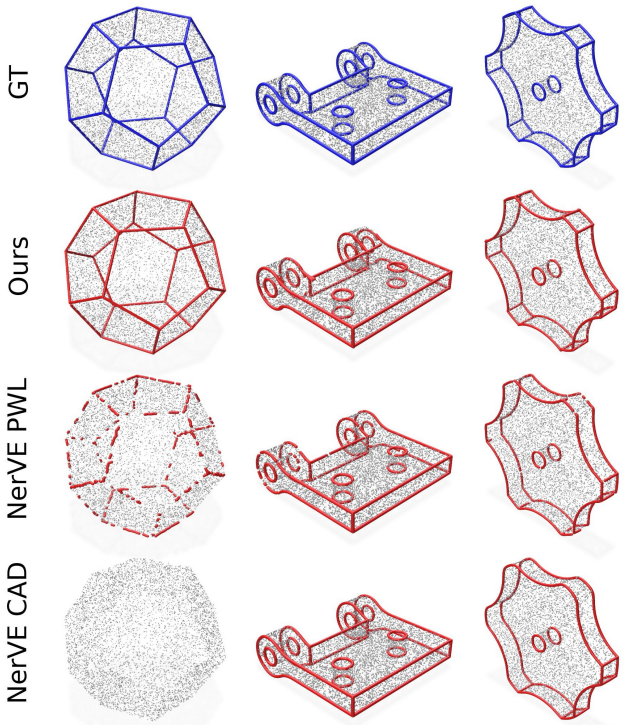


Figure 5. Qualitative comparison of predicted curves using $N = 32{,}768$. Our method and NerVE (CAD) predict parametric curves, while NerVE (PWL) produces piece-wise linear curves. NerVE (CAD) may produce invalid predictions, shown as empty.

3.3. Implementation Details

PI3DETR is implemented in PyTorch [15] and builds on the Transformer design of [3]. The SAModule [16] downsamples each input cloud to $N' = 2048$ points via FPS, computing $d = 256$ features from 64 neighbors within ℓ_2-radius 0.2. Features are projected to $d' = 768$ and fed to 3 Transformer encoder and 9 decoder layers, each with 8-head attention, a two-layer FFN (hidden dim 1024), and dropout [18] of 0.1. Sine positional encodings [21] are applied to XYZ coordinates of encoder outputs and queries. Prediction heads are four-layer FFNs (dim 768), except class, line length, and circle radius heads, which use three layers; all use LayerNorm [2] and ReLU [1]. Optimization uses AdamW [12] with learning rate warm-up from 10^{-10} to 10^{-4} over 30 epochs, then decayed to 10^{-5} at epoch 1230, and gradient clipping at ℓ_2-norm 0.2. Models are trained for 1700 epochs on 2 NVIDIA TITAN RTX GPUs with an effective batch size of 128 via gradient accumulation. Losses are normalized per-object. Data augmentation includes random point dropout (85% probability, retaining at least 20% points) and per-instance random 3D rotation.

4. Experiments

We evaluate PI3DETR through both qualitative and quantitative experiments, including comparisons with 3D edge detection baselines, point sampling and noise robustness tests, and ablation studies. Beyond baseline comparisons, we demonstrate PI3DETR's ability to directly regress parametric 3D curves, capturing rich geometric details both visually and quantitatively.

Configuration. Our unseen test set consists of 924 samples (see Sec. 3.1 for details). The data is normalized to $[-1, 1]^3$ for evaluation and uniformly formatted to ensure fairness and reproducibility. All runtime measurements are conducted on an NVIDIA RTX 4090 GPU. Unless otherwise specified, we use $N = 32{,}768$ input points and $K = 256$ point queries at inference (with training performed using $K = 128$) together with Snap & Fit (S&F) post-processing, as this configuration provides the most stable performance across metrics according to our ablation study (Sec. 4.3).

Metrics. Since no prior method uses the same curve classes as PI3DETR, standard instance detection mean average precision (mAP) metrics are not directly applicable. Let $i \in \{1, \dots, M\}$ be ground truth curve and $j \in \{1, \dots, K\}$ be prediction curve indices. For each curve, we sample points $\tilde{S}_i^{c_i}$ (ground truth) or $\tilde{S}_j^{\hat{c}_j}$ (prediction) at intervals of 0.01 along its type c_i or predicted type $\hat{c}_j$. The aggregated point sets $\bigcup_{i=1}^{M} \tilde{S}_i^{c_i}$ and $\bigcup_{j=1}^{K} \tilde{S}_j^{\hat{c}_j}$ are used to compute Chamfer (CD) and Hausdorff (HD) distances, defined

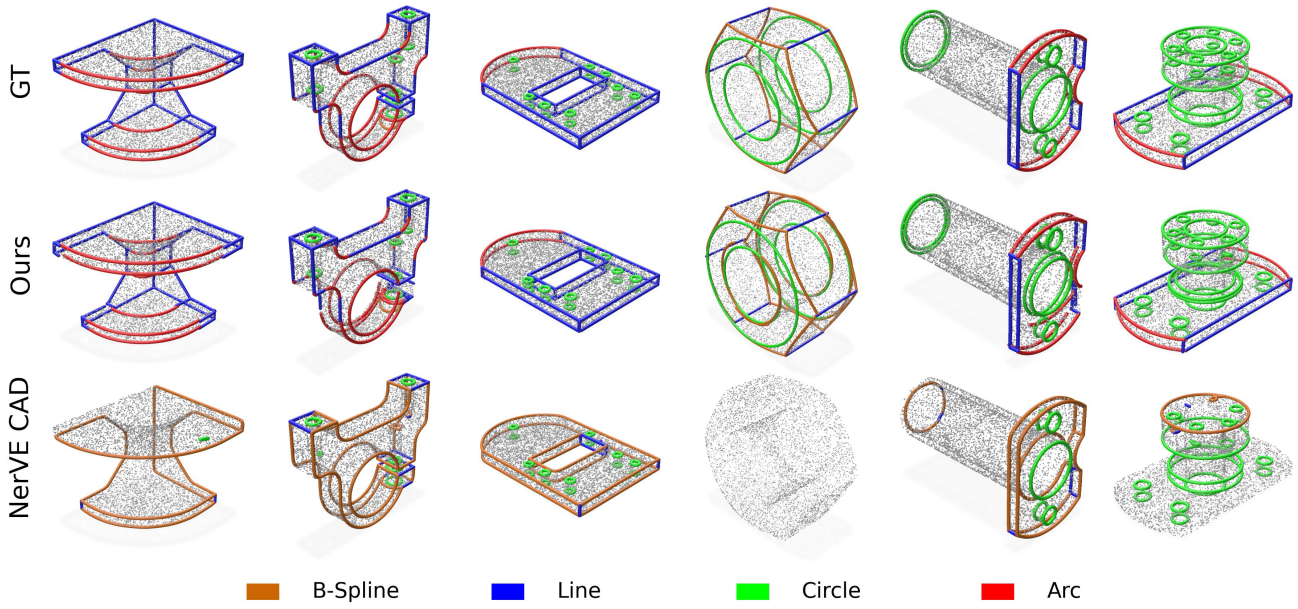


Figure 6. Qualitative comparison of curve instance predictions using $N = 32{,}768$. Our approach assigns the correct class at the appropriate locations, while NerVE often falls back to long B-splines and does not support arcs. Bézier (ours) and B-spline (NerVE) share the same color. NerVE may produce invalid predictions, shown as empty.

in Appendix C. We also report mAP for completeness and future comparison, defining a match (i, j) as a true positive if $c_i = \hat{c}_j$ and the CD between $\tilde{S}_i^{c_i}$ and $\tilde{S}_j^{\hat{c}_j}$ is below 0.005.

Baseline settings. Since NerVE [25] established the state-of-the-art and outperformed prior methods, we primarily compare PI3DETR against NerVE. For completeness, we also attempted to set up PIE-NET [23] and DEF [13], but PIE-NET has not released checkpoints for the second-stage curve fitting and DEF's curve extraction scripts were error-prone and required non-trivial modifications to run reliably. We do not consider segmentation-based approaches that operate directly on the point cloud, as our focus is on parametric curve reconstruction methods. We evaluate NerVE using its released checkpoint and configuration. All reported metrics are computed in $[-1, 1]^3$ space after transforming the outputs accordingly. NerVE predicts piece-wise linear (PWL) curves and fits parametric curves such as B-splines, line segments, and circles through a sensitive post-processing step that often fails to produce curves. In our tables, the number of such failures out of 924 test samples is shown as a superscript. For those cases, we exclude the failed samples from metric computation rather than penalizing them. This favors NerVE in comparison to PI3DETR, which always produced valid curves in the evaluations. Following our metrics protocol, we sample points at 0.01 intervals along the parametric curves or PWLs to compute metrics. We denote the raw PWL outputs as NerVE PWL and the post-processed curves as NerVE CAD. Although PWL outputs are more comparable to segmentation and tend to achieve higher metric scores due to voxel anchoring, we still report them for completeness.

4.1. Comparison

We evaluate PI3DETR's quantitative edge detection performance in Tab. 1. With a CD of 0.0024 and an HD of 0.0635, our method clearly outperforms both NerVE CAD and NerVE PWL. The performance gap is especially pronounced for HD, where higher values indicate severe local mismatches. For NerVE CAD, 77 cases fail during curve fitting from the PWL outputs. By omitting failed cases from NerVE CAD's evaluation, the comparison is biased in its favor, yet PI3DETR still significantly surpasses it. Beyond accuracy, we do not only outperform the PWL setting but also produce interpretable parametric curve instances directly, which the PWL outputs cannot provide.

For qualitative evaluation, Fig. 5 illustrates PI3DETR's

Metric	NerVE CAD	NerVE PWL	**Ours**
CD ↓	0.0401 (± 0.20) ^{77}	0.0046 (± 0.02)	**0.0024** (± **0.02**)
HD ↓	0.2478 (± 0.35) ^{77}	0.1534 (± 0.17)	**0.0635** (± **0.07**)
mAP ↑	–	–	**0.8090**

Table 1. Quantitative comparison with $N = 32{,}768$. Invalid predictions shown in superscript.

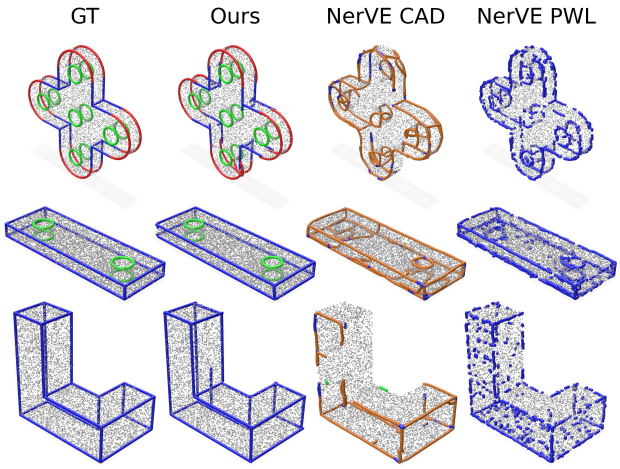


Figure 7. Qualitative comparison on $N = 32{,}768$ affected by noise ($\eta = s/2e^2$) to evaluate robustness.

strong prediction performance and also shows cases where NerVE's post-processing has difficulties converting PWL outputs into valid CAD curves. In Fig. 6, we further present our instance-level predictions and directly usable CAD outputs. Unlike NerVE CAD, which often produces overly long and connected B-spline curves instead of lines or circles, PI3DETR infers semantically more meaningful and structured curve instances.

4.2. Stress Testing

Point cloud density. To evaluate robustness, we vary point cloud density N by randomly sub-sampling the test set (Tab. 2). PI3DETR consistently achieves the lowest CD and HD across all settings, with performance only degrading at $N = 4{,}096$. It surpasses all NerVE variants at $N = 8{,}192$ for CD and $N = 4{,}096$ for HD. In the sparsest setting, NerVE CAD produces 347 invalid outputs, while our method remains fully valid. To further validate our end-to-end design, we report results for the same sparse case without the S&F post-processing, achieving a CD of 0.0049 and an HD of 0.0841, surpassing even the full pipeline.

Noise. To assess robustness against noise, we define three levels $\eta \in \frac{s}{1e^3}, \frac{s}{5e^2}, \frac{s}{2e^2}$, where η is the standard deviation of Gaussian noise added to the 3D coordinates and s denotes the largest axis-aligned span of the point cloud. Results in Tab. 3 show that PI3DETR consistently outperforms NerVE CAD across all noise levels for both CD and HD, and also surpasses NerVE PWL on HD. For CD, NerVE PWL performs better at lower noise levels, but PI3DETR achieves the best score once noise increases to $\eta = \frac{s}{2e^2}$. Without the post-processing step, PI3DETR attains a CD of 0.0138 and an HD of 0.1986, slightly weaker than the full pipeline yet still surpassing the NerVE baseline, further demonstrating its end-to-end robustness.

N	NerVE CAD	NerVE PWL	**Ours**
	Chamfer Distance (CD) ↓		
32,768	0.0401 (± 0.20) ^{77}	0.0046 (± 0.02)	**0.0024** (± **0.02**)
16,384	0.1134 (± 0.43) ^{58}	0.0061 (± 0.02)	**0.0025** (± **0.02**)
8,192	0.2882 (± 0.60) ^{113}	0.0167 (± 0.04)	**0.0027** (± **0.02**)
4,096	0.4562 (± 0.68) ^{347}	0.0984 (± 0.27)	**0.0050** (± **0.03**)
	Hausdorff Distance (HD) ↓		
32,768	0.2477 (± 0.35) ^{77}	0.1534 (± 0.17)	**0.0635** (± **0.07**)
16,384	0.3961 (± 0.49) ^{58}	0.2008 (± 0.20)	**0.0634** (± **0.07**)
8,192	0.6436 (± 0.66) ^{113}	0.2987 (± 0.25)	**0.0680** (± **0.08**)
4,096	0.8665 (± 0.74) ^{347}	0.4918 (± 0.42)	**0.0857** (± **0.10**)

Table 2. Robustness comparison under varying point cloud densities (N) from 32,768 to 4,086. Invalid predictions shown in superscript.

Noise	NerVE CAD	NerVE PWL	**Ours**
	Chamfer Distance (CD) ↓		
$\eta = s/1e^3$	0.0311 (± 0.21) ^{84}	**0.0061** (± **0.02**)	0.0113 (± 0.06)
$\eta = s/5e^2$	0.0194 (± 0.13) ^{199}	**0.0121** (± **0.04**)	0.0129 (± 0.06)
$\eta = s/2e^2$	0.0164 (± 0.04) ^{427}	0.0211 (± 0.05)	**0.0134** (± **0.06**)
	Hausdorff Distance (HD) ↓		
$\eta = s/1e^3$	0.2306 (± 0.30) ^{84}	0.2530 (± 0.22)	**0.1471** (± **0.11**)
$\eta = s/5e^2$	0.2743 (± 0.25) ^{199}	0.3581 (± 0.23)	**0.1684** (± **0.12**)
$\eta = s/2e^2$	0.3086 (± 0.20) ^{427}	0.3874 (± 0.23)	**0.1946** (± **0.12**)

Table 3. Robustness comparison under varying point cloud noise (η) from $s/1e^3$ to $s/5e^2$ where s is the point clouds span. Invalid predictions shown in superscript.

Although PI3DETR degrades relative to the noise-free setting, it remains consistently more robust than NerVE as noise increases. The stronger CD of NerVE PWL at low noise can be attributed to spurious PWLs detected in noisy point clouds, which lower CD but increase HD. This effect is visible in Fig. 7, which shows predictions of all methods for $\eta = \frac{s}{2e^2}$, and is consistent with the higher HD values reported in Tab. 3 or an invalid prediction percentage of 46%.

4.3. Model Ablation

Point queries. Since PI3DETR employs non-parametric point queries [14], the number of queries K can be varied at inference time. Tab. 4 presents an ablation study for $K \in \{128, 256, 512\}$, where results are reported after Snap & Fit (S&F) post-processing and runtimes are measured in seconds. For the main metrics, $K = 128$ achieves the best overall mAP, $K = 256$ yields the lowest Chamfer Distance (CD), and $K = 512$ performs best in terms of Hausdorff Distance (HD). Notably, $K = 128$ provides the strongest mAP across almost all primitives, with the excep-

	K=128	K=256	K=512
	Geometric Quality (CD, HD) ↓		
CD ↓	0.0026 (± 0.02)	**0.0024** (± **0.02**)	0.0025 (± 0.02)
HD ↓	0.0671 (± 0.08)	0.0635 (± 0.08)	**0.0629** (± **0.07**)
	Detection (AP) ↑		
Bézier ↑	0.6156	**0.6184**	0.5960
Line ↑	**0.9416**	0.9192	0.9203
Circle ↑	**0.9045**	0.8871	0.8824
Arc ↑	**0.8373**	0.8114	0.8063
mAP ↑	**0.8247**	0.8090	0.8012
	Timing (sec) ↓		
Model ↓	**0.1128** (± **0.02**)	0.1181 (± 0.03)	0.1337 (± 0.03)
S&F ↓	**0.0423** (± **0.05**)	0.0742 (± 0.10)	0.1323 (± 0.19)

Table 4. Evaluation metrics for different numbers of point queries K and $N = 32{,}768$. All results are reported with Snap & Fit (S&F) post-processing. Runtimes are measured in seconds.

Configuration	CD ↓	HD ↓	mAP ↑
No Post-processing	0.0025 (± 0.02)	0.0663 (± 0.07)	0.8043
+ Snap & Fit	**0.0024** (± **0.02**)	**0.0635** (± **0.07**)	**0.8090**
+ IoU Filter	**0.0024** (± **0.02**)	0.0636 (± 0.07)	0.7981

Table 5. Ablation study of the optional post-processing methods evaluated with $K = 256$ queries and $N = 32{,}768$.

tion of cubic Béziers where $K = 256$ slightly outperforms it. Overall, $K = 256$ emerges as the most stable configuration, balancing CD, HD, and mAP, whereas HD degrades significantly for $K = 128$. In terms of efficiency, $K = 128$ achieves the lowest runtime, while $K = 512$ incurs the highest due to processing more output curves. The end-to-end runtime (inference plus S&F) is 0.1551s, 0.1923s, and 0.2660s for $K = 128$, 256, and 512, respectively.

Post-processing. We also evaluate performance without post-processing. As shown in Tab. 5, even without refinement PI3DETR achieves strong results. Applying S&F offers slight improvements in CD and mAP, with a more noticeable gain in HD, suggesting that S&F effectively mitigates large errors. Interestingly, the IoU-based filtering step does not improve results, which may be explained by the frequent occurrence of thin structures in the dataset, where closely aligned curves risk being mistakenly suppressed.

4.4. Real World Evaluation

The ABC dataset does not contain any real-world object scans in the form of point clouds. To demonstrate our model's generalization to unseen, real-world data, we generated several CAD models of varying complexity that were not part of the ABC dataset. These models were 3D-printed and scanned with a 7-DOF robotic system equipped with a structured light sensor. Data acquisition followed the inspection plan proposed by Staderini *et al.* [19, 20]. The resulting dense point clouds were randomly sub-sampled to $N = 32{,}768$ points for inference. Fig. 8 shows the original mesh, the scanned point cloud, and our results, producing curves consistent with how the edges would be represented in a CAD model. NerVE fails on the right mesh and produces inconsistent, irregular curves on the others.

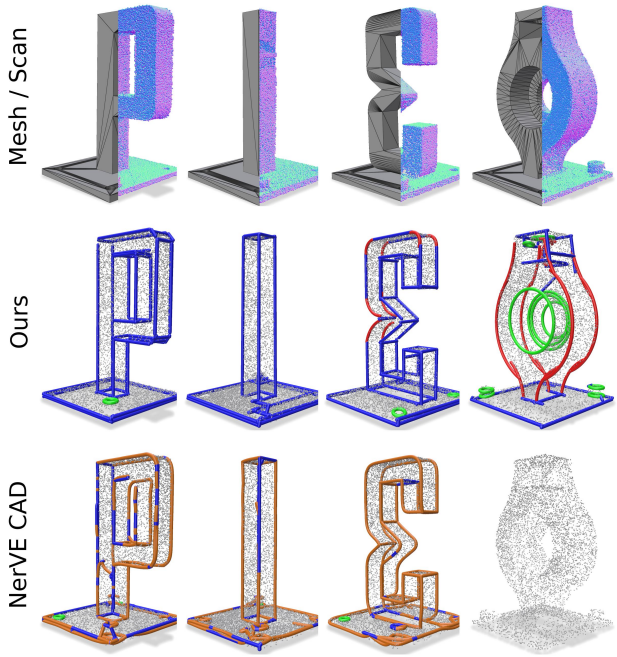


Figure 8. Qualitative comparison of edge detection results on real-world scanned CAD models unseen during training.

5. Conclusion

We presented PI3DETR, an end-to-end framework for detecting differently parameterized 3D curve instances in point clouds in a single forward pass. By incorporating a geometry-aware matching strategy, PI3DETR removes the need for intermediate representations or multi-stage processing and can optionally use simple post-processing steps. We achieve state-of-the-art performance, lowering CD on the ABC dataset by almost half compared to NerVE, and is particularly robust on sparse point clouds. PI3DETR's limitations are the need for extensive training to achieve optimal results, the lack of a fallback for misclassified curves, and the challenge of directly regressing curves without explicit point associations. Nevertheless, its simplicity makes it a powerful foundation for future advances in 3D curve detection and geometric reasoning, paving the way toward more structured and intelligent 3D understanding.

References

[1] Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018. 5

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 5

[3] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020. 2, 3, 4, 5, 1

[4] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3075–3084, 2019. 2

[5] Dawson-Haggerty et al. trimesh, 2019. 3

[6] Haoxiang Guo, Shilin Liu, Hao Pan, Yang Liu, Xin Tong, and Baining Guo. Complexgen: Cad reconstruction by b-rep chain complex generation. *ACM Transactions on Graphics (TOG)*, 41(4):1–18, 2022. 2, 3

[7] Yuan Hu, Zhibin Wang, Zhou Huang, and Yu Liu. Polyroad: Polyline transformer for topological road-boundary detection. *IEEE Transactions on Geoscience and Remote Sensing*, 62:1–12, 2024. 4, 1

[8] Guillaume Klein, Yoon Kim, and Yuntian Deng. Jean senel lart, and alexander rush. 2017. opennmt: Open source toolkit for neural machine translation. *Proceedings of ACL 2017, System Demonstrations*, pages 67–72, 2017. 3

[9] 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*, pages 9601–9611, 2019. 2, 3, 1

[10] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955. 4, 1

[11] Yujia Liu, Stefano D'Aronco, Konrad Schindler, and Jan Dirk Wegner. {PC}2wf: 3d wireframe reconstruction from raw point clouds. In *International Conference on Learning Representations*, 2021. 2

[12] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. 5

[13] Albert Matveev, Ruslan Rakhimov, Alexey Artemov, Gleb Bobrovskikh, Vage Egiazarian, Emil Bogomolov, Daniele Panozzo, Denis Zorin, and Evgeny Burnaev. Def: Deep estimation of sharp geometric features in 3d shapes. *ACM Transactions on Graphics*, 41(4), 2022. 2, 3, 6, 1

[14] Ishan Misra, Rohit Girdhar, and Armand Joulin. An end-to-end transformer model for 3d object detection. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 2906–2917, 2021. 1, 3, 4, 5, 7

[15] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019. 5

[16] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017. 1, 2, 3, 5

[17] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. 2

[18] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014. 5

[19] Vanessa Staderini, Tobias Glück, Philipp Schneider, Roberto Mecca, and Andreas Kugi. Surface sampling for optimal viewpoint generation. In *IEEE 13th International Conference on Pattern Recognition Systems*, pages 1–7, 2023. 8

[20] Vanessa Staderini, Tobias Glück, Philipp Schneider, and Andreas Kugi. Visual quality inspection planning: A model-based framework for generating optimal and feasible inspection poses. In *IEEE International Conference on Intelligent Robots and Systems*, pages 10799–10806, 2024. 8

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 1, 3, 4, 5

[22] Cheng Wang, Xinzhu Ma, Bin Wang, Shixiang Tang, Yuan Meng, and Ping Jiang. Point2primitive: Cad reconstruction from point cloud by direct primitive prediction. *arXiv preprint arXiv:2505.02043*, 2025. 2, 3

[23] Xiaogang Wang, Yuelang Xu, Kai Xu, Andrea Tagliasacchi, Bin Zhou, Ali Mahdavi-Amiri, and Hao Zhang. Pie-net: Parametric inference of point cloud edges. *Advances in neural information processing systems*, 33:20167–20178, 2020. 2, 3, 6, 1

[24] Lequan Yu, Xianzhi Li, Chi-Wing Fu, Daniel Cohen-Or, and Pheng-Ann Heng. Ec-net: an edge-aware point set consolidation network. In *Proceedings of the European conference on computer vision (ECCV)*, pages 386–402, 2018. 2

[25] Xiangyu Zhu, Dong Du, Weikai Chen, Zhiyou Zhao, Yinyu Nie, and Xiaoguang Han. Nerve: Neural volumetric edges for parametric curve extraction from point cloud. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13601–13610, 2023. 2, 3, 6