# NerVE: Neural Volumetric Edges for Parametric Curve Extraction from Point Cloud – *Supplementary Material*

Xiangyu Zhu[12*] Dong Du[1*] Weikai Chen[3] Zhiyou Zhao[1]
Yinyu Nie[4] Xiaoguang Han[12†]

[1]SSE, CUHKSZ [2]FNii, CUHKSZ [3]Tencent America [4]Technical University of Munich

In this supplementary material, we provide additional details of our method in Sec. A, data collection setup in Sec. B, and additional ablation study, numerical and visual results in Sec. C.

## A. Details of Method

In this section, we present network specifications for learning NerVE and list the post-processing details for the PWL curves refinement and the final parametric curve fitting.

### A.1. Network Details

**Encoder.** We use a simplified dense PointNet++ [6] as our encoder. Specifically, given a point cloud of shape $(N, 3)$ ($N$ is the point number), we first find k-nearest neighbors ($k = 8$ as in [1]) of each point and reform them into a tensor of shape $(N, 8, 3)$ as input. Then we apply a network consisting of 4 layers of MLP, where the latent size is 128 and the output shape is $(N, 8, 128)$. We finally obtain point features of $(N, 128)$ by using a max pooling function in the second dimension. After that, the point features are fused as cube features $(32, 32, 32, 128)$ (32 is the grid resolution) by average pooling when multiple points appear in the same cube. Three 3D-convolution layers are then applied to the cube features, where the kernel size, stride, and padding are 3, 1, and 1 respectively. We use Leaky ReLU as the activation function. The final shape of the feature grid is $(32, 32, 32, 128)$.

**Decoder.** We adopt a 5-layer MLP as the decoder to predict edge occupancy, orientations, and edge point position. The latent layer size of the MLP is 128. Each cube in the feature grid has a feature size of 128, which is directly decoded by the occupancy decoder into a one-dimensional scalar. It is decoded by a position decoder into three floats. For the orientations decoder, as shown in Figure 2 in our paper, we first concatenate the cube feature with its three adjacent cube features, which means the input to the orientations decoder is of shape $(3, 128 + 128)$. The orientations decoder takes the concatenated feature as the input and produces three one-dimensional scalars. Sigmoid activation is used as the last layer in both the occupancy decoder and the orientations decoder. The output of our position decoder is clipped into $[-1, 1]^3$ to be consistent with the coordinate system of the input point cloud (see *Input Point Cloud Pre-processing* in Sec. B for more details on coordinate transform).

### A.2. Parametric Curve Extraction

We introduce the processing details in parametric curve extraction, including the refinement of PWL curves and parametric fitting. Note that the whole procedure of extraction is fast, where the parametric curve extraction takes only 0.018 seconds per shape on average from all shapes in the test set (472 shapes), where the average vertex number on shape curves is about 22000.

**PWL Curves Refinement.** To refine the predicted PWL curves from our network for parametric curve extraction, we propose several post-processing steps. In the following settings, the predicted PWL curves are regarded as an undirected graph, where the definition of vertex degree is the same as in general graph theory.

**Step 1: Point Reconnection.** We first find all vertices with degree 1 and denote such vertices as 1-d vertices. Then we add an edge between two 1-d vertices if the distance of these vertices is smaller than a given threshold $\delta_r$ meanwhile, their tangent vectors should be consistent enough. See Fig. 1 for an illustration of reconnection as well as the definition of consistency between tangent vectors. Take Fig. 1 (b) as an example, the consistency of tangent vectors is defined by $\vec{t}_1 \cdot \vec{t}_2 + \vec{t}_2 \cdot \vec{t}_3$. If $\|\overrightarrow{p_1 p_2}\| < \delta_r$ and $\vec{t}_1 \cdot \vec{t}_2 + \vec{t}_2 \cdot \vec{t}_3 > \sqrt{2}$ ($\sqrt{2}$ is fixed, which works well in all our experiments), we

*Xiangyu Zhu and Dong Du contribute equally.
†Corresponding author's email: hanxiaoguang@cuhk.edu.cn

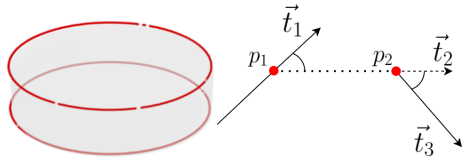


Figure 1. (a) An example that requires curve reconnection. (b) Let $p_1, p_2$ be two close 1-d vertices. $\vec{t}_1, \vec{t}_3$ are tangent vectors for $p_1, p_2$ (defined as in the PWL curves) and $\vec{t}_2 = \overrightarrow{p_1p_2}/\|\overrightarrow{p_1p_2}\|$.

can connect $p_1$ and $p_2$.

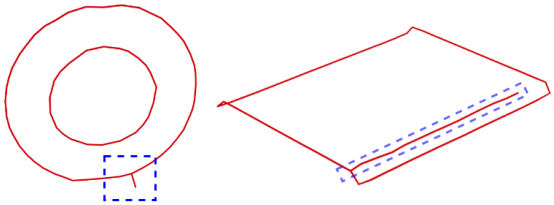


Figure 2. Examples of extra edges. (a) A short protruding edge will be removed if its vertex number is less than $N_p$ ($N_p = 5$ in our experiments). (b) A long extra edge is also removed because of the B-Rep constraints. Such long extra edges usually appear around non-sharp edges of shapes (referring to Fig. 4), and it is difficult for neural networks to distinguish them from sharp edges.

**Step 2: Extra Edges Removal.** As shown in Fig. 2, there could be extra edge segments in the space. To remove these outliers, we first find all paths which start with a 1-d vertex. A path on the PWL curves graph is defined by adding constraints: $\deg(V_{\text{in}}) = 2, \deg(V_{\text{end}}) \neq 2$, where $V_{\text{in}}$ is interior vertex and $V_{\text{end}}$ is the end vertex of the path and $\deg(V)$ means the degree of vertex $V$ on PWL curves graph. Then we remove one from these paths if its vertex number is less than $N_p$. In this way, short extra edges can be removed, and there remain long extra edges that failed to reconnect in *Step 1*. We provide an option on keeping long protruding edges according to users' preference on conforming to the B-Rep constraints. In particular, long protruding edges are removed if B-Rep constraints are strictly required and kept otherwise. In our experiments, we choose to impose the B-Rep constraints for final parametric curves and remove such long protruding edges.

**Step 3: Multi-paths Handling.** In our method, all vertices with degree $> 2$ are regarded as endpoints of curves. However, there could be multiple close paths between two endpoints as shown in Fig. 3, When curves are close to edges or faces of the cube, the network might predict the additional point causing multiple paths. To handle this issue, we can randomly select one of the paths and delete others since all paths are close in geometry, and we finally fit the paths in the sense of least squares. Two paths from the same pair of

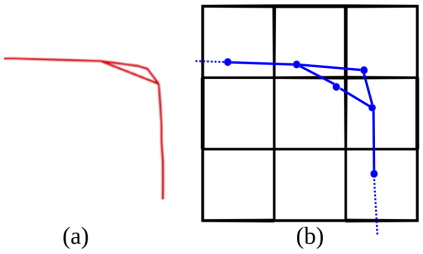


Figure 3. (a) There are multiple paths between two degree$>2$ vertices(endpoints). (b) 2D illustration for generation of multi-paths. When a curve is near the edges or faces of the cube, the network might predict the additional point, which can cause multiple paths between two endpoints.

endpoints are defined to be close if the Chamfer distance of these two paths is lower than a given threshold $\delta_p$. We repeat this process until all extra paths are eliminated.

| $\delta_r$ | Value | $2l$ | $3l$ | $4l$ | $5l$ |
|---|---|---|---|---|---|
| | CD↓ | 0.0112 | 0.0066 | **0.0051** | 0.0121 |
| | HD↓ | 0.2246 | 0.1924 | **0.1844** | 0.2870 |
| $N_p$ | Value | 3 | 4 | 5 | 6 |
| | CD↓ | 0.0054 | 0.0054 | **0.0051** | 0.0061 |
| | HD↓ | 0.2046 | 0.2010 | **0.1844** | 0.1847 |
| $\delta_p$ | Value | $0.5l$ | $2l$ | $4l$ | $\infty$ |
| | CD↓ | 0.00509 | **0.00508** | 0.00509 | 0.00512 |
| | HD↓ | 0.18442 | 0.18442 | 0.18442 | 0.18442 |

Table 1. Different CD and HD errors of final parametric curves when $\delta_r, N_p, \delta_p$ taking different values. $l = 2/r$ is the edge length of a cube in the grid ($r = 64$ for resolution $64^3$). By default, $\delta_r = 4l, N_p = 5, \delta_p = 2l$. In the experiments, one parameter value changes and other parameters remain the default values. For $\delta_p$, the value of $\infty$ means we handle all possible multi-paths cases without checking the Chamfer distance.

**Choices of Parameters.** Three parameters are discussed in PWL curves refinement: $\delta_r, N_p, \delta_p$. To choose appropriate values, we compare the CD and HD errors of the final parametric curves in different parameter settings, as shown in Table 1. Based on the comparison, we choose $\delta_r = 4l, N_p = 5, \delta_p = 2l$, where $l = 2/r$ ($r = 64$ for resolution $64^3$) is the edge length of a cube in the grid.

**Parametric Curve Fitting.** After obtaining paths between pairs of endpoints or closed paths, we can use an off-the-shelf spline fitting library for parametric curve fitting on these paths. Specifically, we use the function *make_lsq_spline* from *Scipy*, which can fit given points in the sense of least squares with BSpline functions. For our setting of BSpline functions, the order of spline is 3; the number of knots (except the knots for start and end points) is half of the number of path vertices; knots are uniformly sampled in $[0, 1]$. Note that the positions of all endpoints are fixed during fitting.

For closed paths, we first try direct 3D circle fitting on one closed path since most closed curves in the ABC dataset are circles [7]. If the fitting error is large, which means the closed path is probably not a circle, we simply apply the previous spline fitting to it. Here, the threshold for the fitting error is fixed as 0.001, which works fine in all our experiments. For 3D circle fitting, we use a feasible and simple approach. We first use principal component analysis (PCA) on points and then convert it to a 2D circle fitting problem, which can be easily solved, finally we map the fitted 2D circle into $\mathbb{R}^3$ by PCA and obtain the 3D circle.

## B. Details of Dataset

In this section, we provide more details about data processing, including data cleaning of raw ABC dataset [4], ground truth data preparation of NerVE attributes, and preprocessing for input point clouds.

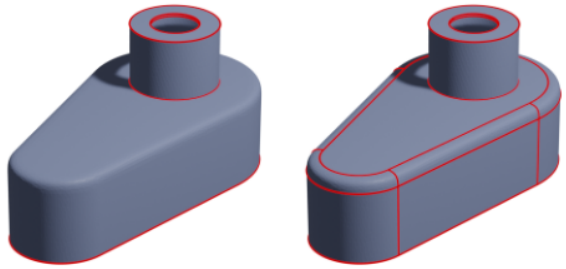

Figure 4. Differences between sharp edge and non-sharp edge. We highlight the sharp edges on the left figure, while highlight both sharp and non-sharp edges on the right figure.

**Data Cleaning.** As discussed in our paper, the dataset has to be cleaned due to data missing, shape repetition, and lack of sharp edges. For those shapes with little difference in geometry or structure, we regard them as repeated. As for the lack of sharp edges, there are cases where a shape does not possess any sharp edges (e.g. a sphere) or only has a few sharp edges. Here we use the sharp edges originally defined in the ABC dataset [4] and Fig. 4 shows the difference. To filter out the mentioned shapes, we manually examined all shapes from the first chunk and obtained 2,364 shapes for network learning.

**Ground Truth Preparation.** To obtain the ground truth of cube attributes in NerVE, we first obtain the PWL curves from the GT parametric curves by uniform sampling and denote the PWL curves as the GT PWL curves. In a grid of $32^3$ cubes (here we assume the grid resolution is 32), all cubes intersected with the GT PWL curves are labeled True, which are occupied cubes. Similarly, all faces intersected with the GT PWL curves are labeled True. Intersections of GT PWL curves with cubes can be easily calculated by considering each line segment in GT PWL curves. In the occupied cubes, we take the midpoint of the truncated GT PWL curve inside the cube as the point position.

**Input Point Cloud Pre-processing.** To benefit network training, we normalize the input point cloud and GT point positions of cubes. For the input point cloud, we first subtract the point position from its $k$ nearest neighbors for each point, then multiply by a factor of $r$ ($r = 32$ for resolution $32^3$). For the GT global point positions, we convert it to the local coordinates of its cube, where the center of a cube is the new origin, and the axes are scaled by $r$ ($r = 32$ for resolution $32^3$). In this way, the range of the GT point position becomes $[-1, 1]^3$.

## C. More Details of Experiments

In this section, we give more details about the experiments, such as baseline settings in the comparison and a detailed explanation of cube point choice in the ablation study. Finally, we show more results of our method in Fig. 6.

**Baseline Settings.** In our experiments, we adopt VCM [5], EC-NET [8], and PIE-NET [7] as baselines to evaluate our proposed method. Specifically, we use the implementation of VCM in the CGAL library [2]. Given a point cloud for testing, we compute the Voronoi covariance at each point, where the offset radius is 0.2, the convolution radius is 0.25, and the pareto-optimal threshold is 0.24. When a Voronoi covariance value is larger than the threshold, we consider the corresponding point to belong to an edge. For EC-NET and PIE-NET, we utilize the released source codes and pretrained models to test the input point cloud with their specified normalization, and then transform the outputs to align with the ground truth for a fair evaluation.

**Time Consumption.** The average inference times of VCM, EC-Net, PIE-Net, and Ours are 2.06, 0.84, 0.52, and 0.15 seconds, respectively. For post-processing, our method takes 0.02s on average, which is more efficient than the post-processing of PIE-Net (3.01s). It can be explained by using masks, which can only choose surface cubes for the calculation to reduce consumption and make it more efficient than other approaches.

| | | VCM [5] | EC-NET [8] | PIE-NET [7] | Ours |
|---|---|---|---|---|---|
| Clean | | 0.194 | 0.128 | 0.132 | **0.071** |
| Noise | $\sigma = l/4$ | 0.200 | 0.222 | 0.289 | **0.097** |
| | $\sigma = l/2$ | 0.270 | 0.278 | 0.301 | **0.110** |
| #Sample Points | 16384 | 0.185 | 0.135 | 0.179 | **0.095** |
| | 8192 | 0.192 | 0.164 | 0.238 | **0.112** |
| | 4096 | 0.203 | 0.212 | 0.246 | **0.146** |

Table 2. Edge estimation errors (HD, the smaller the better) of four methods on noisy or resampled inputs. $l = 2/32$ is the edge length of a cube in grid.

**HD Results of Stress Tests.** Table 2 shows additional quantitative results of edge estimation in stress tests. Here the metric is HD instead of CD used in the main paper.

**Ablation Study for Cube Point Choice .** We show details for the discussion of cube point choice. In Dual Contouring [3], the point position in the cube is calculated by minimizing a quadratic error function (QEF) on Hermite data of the surface, which are intersection points of the surface with the cube edges and their corresponding normal vectors. Let $p$ be the point position, it can be calculated by the following minimization:

$$p = \arg\min_x \sum_i (n_i \cdot (x - p_i))^2. \quad (1)$$

where $p_i$ is one of the intersection points with the cube edges and $n_i$ is its normal vector. The surface is approximated as a plane at each intersection point, and the formulation minimizes all the distances from $p$ to all planes in the sense of least squares. One natural counterpart for a 3D curve is to consider the intersection points with cube faces and their tangent vectors. Similarly, we approximate the curve as a line at each intersection point $p_i$ with the cube face, and minimize all distances from $p$ to all lines in the sense of least squares. Let $t_i$ be the tangent vector of $p_i$ (the direction does not need to be specified), QEF for the curve can be formulated as:

$$\min_{x,\forall\alpha_i} \sum_i \|x - p_i - \alpha_i t_i\|_2^2 + \lambda \sum_i \alpha_i^2. \quad (2)$$

where $\alpha_i$ is proposed to enable the problem to be solved by a linear system and $\lambda$ is a weight to balance the two terms. Notice that we only need to solve for $x$, and the system can be easily reformed as a linear system of order 3 to solve for $x$. However, such a type of point position definition does not perform well in the curve restoration as shown in our main paper. Therefore, we finally choose a simple and accurate definition of the point position, which adopts the midpoint of the truncated curve inside the cube. See Fig. 5 for a visual comparison of these two definitions.

**Performance in Higher Resolution.** To evaluate the performance of NerVE in a higher resolution, we perform experiments using resolution $128^3$ and compare the results with lower resolutions, as shown in Table 3. Using resolution $128^3$ brings more accurate PWL outputs (CD, HD), with slightly fewer scores of $R_o$, $P_o$, and $C_e$. The average inference times of Ours-$32^3$, Ours-$64^3$, and Ours-$128^3$ are 0.15, 0.21, and 0.57 seconds, respectively. On the whole, the performance of NerVE can scale well with the increase of voxel grid resolutions.

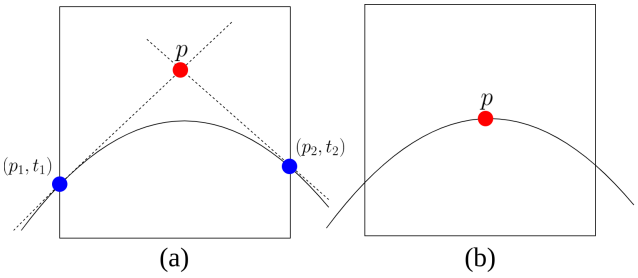


Figure 5. 2D illustration for two types of cube point definitions. (a) $p$ is obtained by minimizing a QEF. (b) $p$ is obtained by taking the midpoint of the truncated curve inside the cube.

| | $R_o\uparrow$ | $P_o\uparrow$ | $C_e\uparrow$ | $D_p\downarrow$ | CD↓ | HD↓ |
|---|---|---|---|---|---|---|
| Reso $32^3$ | **0.965** | **0.965** | **0.940** | 0.003 | 0.0012 | 0.0714 |
| Reso $64^3$ | 0.958 | 0.960 | **0.940** | 0.001 | 0.0009 | 0.0523 |
| Reso $128^3$ | 0.945 | 0.947 | 0.914 | **0.0005** | **0.0008** | **0.0484** |

Table 3. NerVE performance in different resolutions.

**Ablation on Choice of $k$ in Point Encoder.** We choose $k = 8$ empirically based on the experiments of resolution $32^3$. For the higher resolution $64^3$, using a larger $k$ may improve the accuracy but incur much more calculation cost, as shown in Table 4. It is a trade-off of accuracy and efficiency to apply $k = 8$ in our experiments.

| | $32^3, k=4$ | $32^3, k=8$ | $32^3, k=16$ | $64^3, k=4$ | $64^3, k=8$ | $64^3, k=16$ |
|---|---|---|---|---|---|---|
| CD↓ | 0.0017 | 0.0012 | 0.0012 | 0.0015 | 0.0009 | 0.0008 |
| HD↓ | 0.0833 | 0.0714 | 0.0669 | 0.0680 | 0.0523 | 0.0491 |

Table 4. Ablation study on choices of k in the point encoder.

**Ablation on Using PointNet++ or 3DCNN Features.** We conducted an ablation study with or without using PointNet++ and 3DCNN features. The results of the network predictions and the PWL curves are reported in Table 5. As shown, both the PointNet++ and 3DCNN features can promote the performance of NerVE.

| | $R_o\uparrow$ | $P_o\uparrow$ | $C_e\uparrow$ | $D_p\downarrow$ | CD↓ | HD↓ |
|---|---|---|---|---|---|---|
| wo PointNet++ | 0.9358 | 0.9494 | 0.8517 | 0.0041 | 0.0025 | 0.0982 |
| wo 3DCNN | 0.9383 | 0.9460 | 0.9127 | 0.0040 | 0.0020 | 0.0942 |
| Ours | **0.9649** | **0.9650** | **0.9437** | **0.0030** | **0.0012** | **0.0714** |

Table 5. Ablation study on using PointNet++ and 3DCNN blocks.

**Ablation Study on Post-processing.** The quantitative results of parametric curves with/without post-processing are CD: 0.008/0.067, HD: 0.224/0.225. Thus, the post-processing is necessary numerically for better parametric extraction.

**Ablation Study on data splits.** Our experiments are all based on the same random split. Here, we test on other two different random splits with resolution $32^3$, the CD results

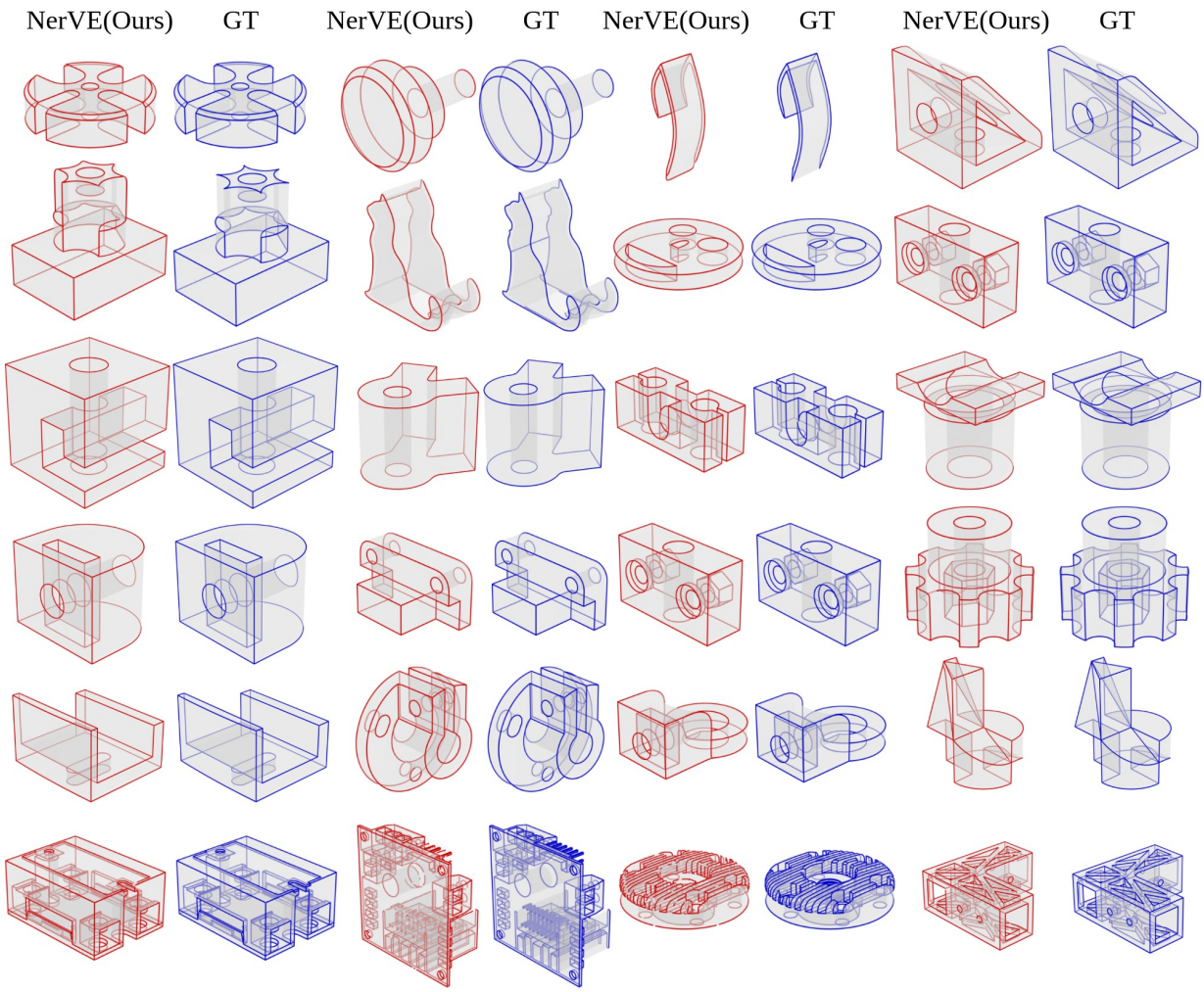


Figure 6. More results of our method. The last row shows the predicted results of our method on 4 complicated cases.

are 0.0013 and 0.0016 (the number in our main paper is 0.0012). As shown, the differences are insignificant.

**More Results.** More results of our method are shown in Fig. 6. Our method can produce reasonable results even for complicated cases, as shown in the last row of Fig. 6.

## References


[1] Zhiqin Chen, Andrea Tagliasacchi, Thomas Funkhouser, and Hao Zhang. Neural dual contouring. *arXiv preprint arXiv:2202.01999*, 2022. 1

[2] Andreas Fabri and Sylvain Pion. Cgal: The computational geometry algorithms library. In *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*, pages 538–539, 2009. 3

[3] Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. Dual contouring of hermite data. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 339–346, 2002. 4

[4] 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. 3

[5] Quentin Mérigot, Maks Ovsjanikov, and Leonidas J Guibas. Voronoi-based curvature and feature estimation from point clouds. *IEEE Transactions on Visualization and Computer Graphics*, 17(6):743–756, 2010. 3

[6] 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

[7] 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. 3

[8] 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. 3