

Constraint-Aware Feature Learning for Parametric Point Cloud

Supplementary Material

A. Overview

This supplementary material provides additional details that supplement the main paper and includes more experimental results.

In Sec. B, we present the details of network architecture and data preparation in validation experiments. In Sec. C, we provide the data construction and static of Param20K dataset. In Sec. D offers data process for visualization and more results of constraint acquisition and classification experiments. In Sec. E, we show more details of our constraint representation and constraint learning methods.

B. Validation Experiment

B.1. Constraint Representation

This section introduces the constraint representation in the validation experiments.

The constraint representation in validation experiments is defined in a simple manner: for a point, its constraint representation is defined by the relationship between the plane it attached and the reference plane, as shown in Fig. 1. This calculation process is carried out by openCASCADE Technology (OCCT). The constraint representation is represented in a one-hot encoding, where $(1, 0, 0)$ indicates perpendicular, $(0, 1, 0)$ indicates parallel, and $(0, 0, 1)$ indicates other constraints. Each point in the point cloud is expressed as $(x, y, z, constraint)$. For example, if the coordinate of a point is $(0.1, 0.2, 0.3)$, and the plane it attached is perpendicular to the reference plane, it is expressed as $(0.1, 0.2, 0.3, 1, 0, 0)$, where $(0.1, 0.2, 0.3)$ represents the point's coordinate, and $(1, 0, 0)$ denotes the type of constraint.

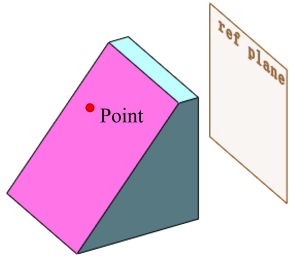


Figure 1. **Constraint representation in validation experiments.** The red point's constraint representation is the constraint type between the magenta plane and the reference plane, using one-hot encoding.

B.2. Dataset Generation

This section introduces the dataset generation procedures in the validation experiments.

The dataset generation for the validation experiments consists of the following steps:

1. Building the parametric templates for cuboids and prisms (with angles of $50^\circ, 60^\circ, 70^\circ, 80^\circ, 82^\circ, 85^\circ, 87^\circ, 89^\circ$). Afterwards determining the dimensional parameter range for each parameter. The parametric templates and the dimensional parameters are shown in Fig. 2.
2. Assigning random values to the parameters in parametric templates. Each parametric template is instantiated into 5,000 BRep format CAD shapes, which are stored in STEP files [8].
3. Using OCCT to read the STEP files and convert them into meshes. Poisson disk sampling is then applied to sample points from meshes.
4. For the sampled points, OCCT is used to calculate the plane to which each point is attached. The constraint type for each point is then determined based on the relationship between the attached plane and the reference plane.
5. Rotating the generated point clouds randomly along the XYZ axes, within a range of $[-25^\circ, 25^\circ]$.
6. Assigning the generated point clouds to each experiments. The validation experiments involve binary classification of cuboids and prisms with specific angles. The point clouds used in each of the eight experiments are as follows (80 % for training, and 20 % for testing):

- Experiment 1: Cuboid \times 5000, Prism $50^\circ \times$ 5000
- Experiment 2: Cuboid \times 5000, Prism $60^\circ \times$ 5000
- Experiment 3: Cuboid \times 5000, Prism $70^\circ \times$ 5000
- Experiment 4: Cuboid \times 5000, Prism $80^\circ \times$ 5000
- Experiment 5: Cuboid \times 5000, Prism $82^\circ \times$ 5000
- Experiment 6: Cuboid \times 5000, Prism $85^\circ \times$ 5000
- Experiment 7: Cuboid \times 5000, Prism $87^\circ \times$ 5000
- Experiment 8: Cuboid \times 5000, Prism $89^\circ \times$ 5000

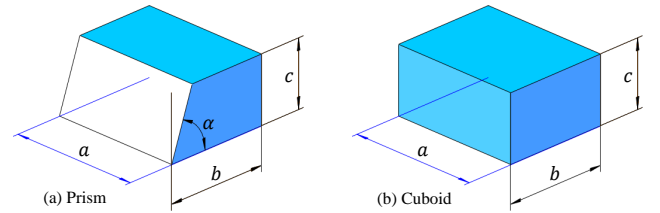


Figure 2. **Parametric templates for prisms and cuboids.** $a, b, c \in [0.3, 2.3]$, for prisms: $b > \frac{c}{\tan \alpha}$.

B.3. Model Design

This section introduces the structure of the constraint-aware model used in the validation experiments.

The constraint-aware model first employs a PointNet++ [10] backbone to predict point-wise constraint representations. These constraint representations are then concatenated with the point coordinates and passed through another PointNet++ backbone to determine the point cloud category. The model’s loss function is the sum of the classification loss and the constraint representation loss, as shown in Fig. 3

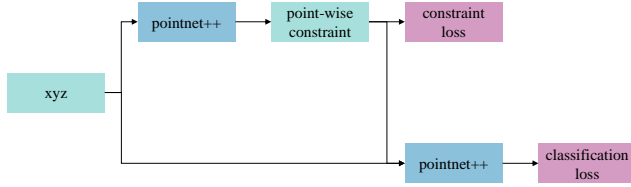


Figure 3. Constraint-aware model structure.

C. Param20K Dataset

C.1. Why build up the Param20K dataset?

The limited availability of BRep datasets poses barriers to deep learning on CAD shapes. Most existing CAD shape datasets consist of mesh files, such as MCB [3] and ESB [2]. While mesh files could approximate the appearance of CAD shapes, lack crucial boundary information. In contrast, BRep data [5, 16] serves as the native representation of CAD shapes and is therefore more suitable for dataset construction. However, labeled BRep datasets remain relatively scarce, for example, FabWave [1] includes only 2,133 BRep files. Although the ABC [4] contains a large number of BRep files, it remains unlabeled.

C.2. Dataset Construction

This section provide details of purification process and data acquisition of Param20K dataset.

Purification: The purification process involved removing broken, duplicated, or overly complex CAD shapes requiring more than 4,000 points for point cloud representation.

Point cloud: The point cloud data in Param20K dataset containing about 2,200 points, point constraints is also included, each point is expressed as (Coordinate, Primitive Type, Main Axis Direction, Adjacency), the meshes and point clouds are generated from BRep data.

C.3. Dataset Statics

This section presents the shapes count and proportion statistics for each category in the Param20K dataset. Additionally, shapes from each category are showcased.

Param20K dataset is divided into 80 % for training and 20 % for testing.

The class distribution is shown in Fig. 4.

The proportion of shapes in each category are shown in Fig. 5.

Shapes instantiated from the parametric templates designed by our team names begin with "Num".

Example shapes from each category are shown in Fig. 6.

D. Constraint Acquisition and Classification Experiments

D.1. Process to Visualize Main Axis Direction

The main axis direction of a point, denoted as $v = (x, y, z)$, is visualized as $R = \frac{x+1}{2}$, $G = \frac{y+1}{2}$, and $B = \frac{z+1}{2}$. Since the main axis direction is represented as a unit vector, $-1 \leq x, y, z \leq 1$, which results in $0 \leq \frac{x+1}{2}, \frac{y+1}{2}, \frac{z+1}{2} \leq 1$.

D.2. Constraint Visualization of CstBRep

More constraints extracted by the CstBRep module from BRep data are visualized in Fig. 7.

D.3. Constraint acquisition on MCB

We evaluated the pre-trained CstPnt module and ParSeNet [11], HPNet [15] on the CAD shape dataset MCB [3] to predict constraints. For the MCB consists of mesh files, it was not possible to extract Ground Truth constraints for comparison, therefore no ground truth presented in Fig. 8, and no numerical results are available. Based on the results visualized in Fig. 8, CstPnt demonstrated a high level of constraint prediction accuracy even on an unseen dataset.

D.4. Classification on MCB

We conducted evaluations on the CAD shape dataset MCB [3], with results presented in Tab. 1, from which our CstNet achieved the highest scores across all metrics, conclusions similar with experiments on Param20K dataset.

E. Constraint Learning Method Details

E.1. Defects of General Constraint Representation

The simple constraint representation in validation experiments is insufficient for a comprehensive definition, as selecting an appropriate reference plane for complex CAD shapes is challenging. Moreover, traditional constraint representations suffer from non-uniqueness, as shown in Fig. 9, defining the relations between three planes involves multiple constraint options, therefore, this approach is not well-suited for deep learning models.

The constraint is defined as a graph representation in some circumstances, where each node represents a primitive, and the edge between nodes represent their constraints.

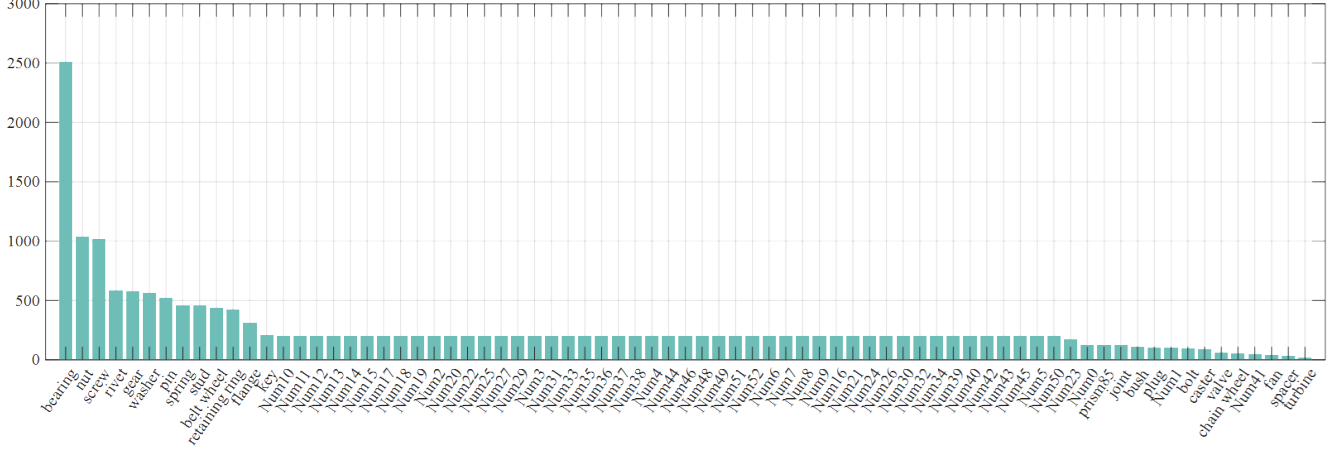


Figure 4. Quantity distribution of Param20K dataset.

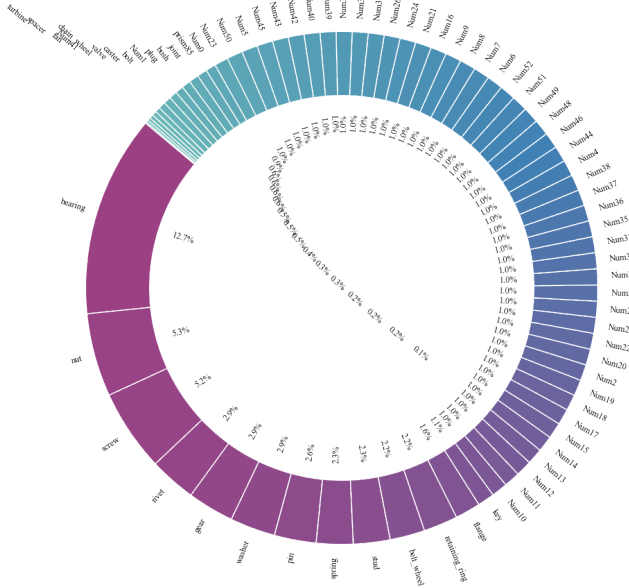


Figure 5. Shapes proportion of Param20K dataset.

However, this approach is not well-suited for point cloud analysis. First, in cases where only point clouds are provided as input, the number of primitives is hard to obtain. Second, due to the unordered nature of point cloud, it is difficult to associate points with the corresponding primitives, making label design very challenging. Additionally, the graph representation essentially records the constraint relationships between each pair of primitives, for the CAD shapes contain many primitives, recording the relative positions between every pair of primitives would result in an overwhelming data volume, making the representation of these relative positions impractical.

Table 1. Classification results on MCB. Acc: accuracy over instance %, Acc*: accuracy over class %, F1: F1-score, mAP: mean average precision %.

Method	Acc	Acc*	F1	mAP
PointCNN [6]	93.89	81.85	83.86	90.13
PointNet [9]	86.78	67.70	86.55	74.08
PointNet++ [10]	87.45	73.68	88.32	91.33
SpiderCNN [14]	93.59	79.70	81.30	86.64
PointConv [13]	93.25	80.24	71.31	82.19
DGCNN [12]	92.54	74.47	76.12	74.27
3DGCN [7]	93.71	78.71	84.59	84.35
Ours	96.87	89.21	89.85	93.17

E.2. Unique Process of MAD

This section provides the process of Main Axis Direction and example constraint representation.

The main axis direction is processed to ensure its uniqueness, as one axis can determine two unit vectors with opposite direction. For a given axis, we first randomly select one of the two vectors, denoted as $v(x, y, z)$, and then process it using the Algorithm 1. Examples shown in Fig. 10.

A point with constraint representation is expressed as Fig. 11.

E.3. CstBRep Module

This section explains why it is necessary to compute valid edges in the CstBRep module.

The CstBRep module is used to compute constraints from BRep data, and the computation process consists of the following steps:

Step 1: Point cloud generation (optional).

Step 2: Calculation of valid edges and the distance from each point to these valid edges. If the minimum distance

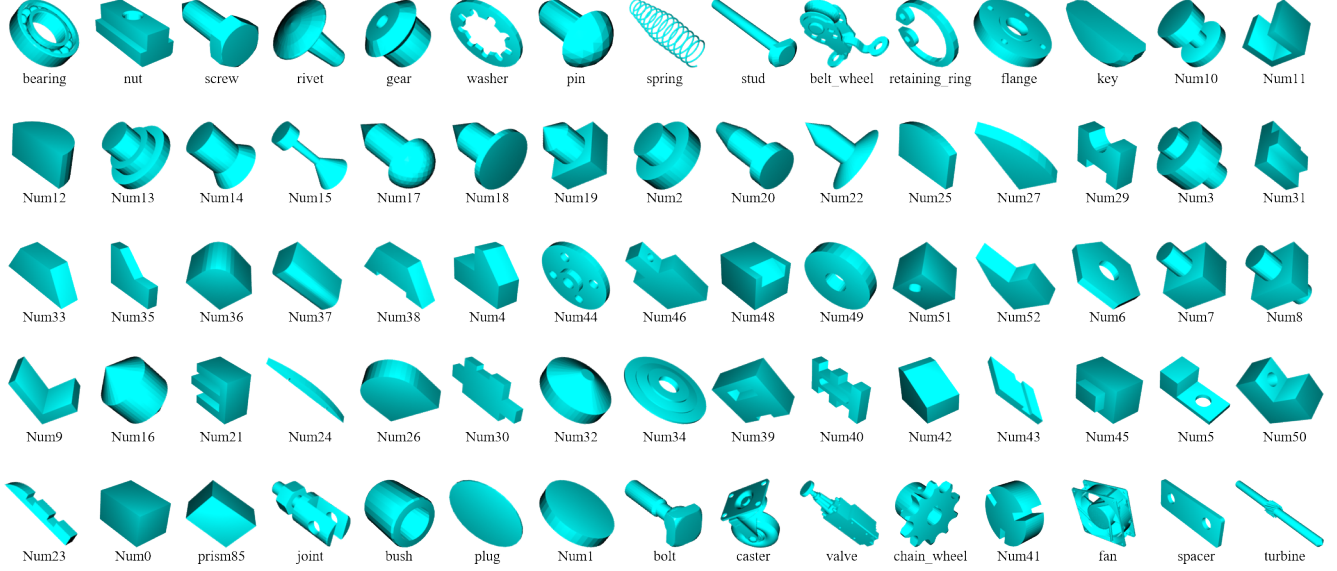


Figure 6. Example shapes for each category in Param20K dataset.

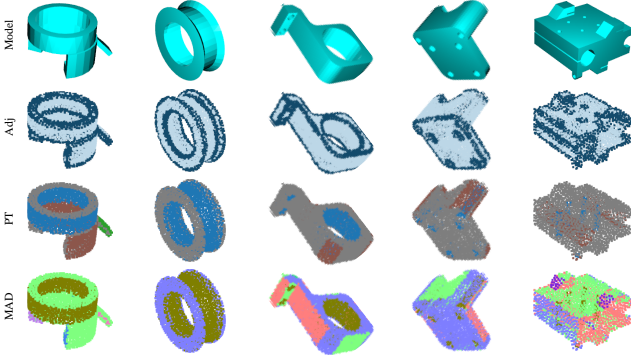


Figure 7. More constraint extraction results from the BRep data. MAD: Main Axis Direction, Adj: Adjacency, PT: Primitive Type.

is below a threshold, the point is classified as an edge-near point.

Step 3: Calculation of the primitives to which each point attached, followed by analysis to determine point’s Primitive Type and the Main Axis Direction.

In **Step 2**, valid edges are computed, as this helps to reduce the potential for data to mislead the deep learning model. The definition of valid edges is illustrated in Fig. 12. This definition is based on the observation that a complete cylinder in the BRep data stored in STEP files is often split into two half-cylinder faces. As a result, two generatrices form edges on the cylindrical surface, but the deep learning model cannot predict these edges, as shown in Fig. 13. Furthermore, in other cases, edges corresponding to smoothly

Algorithm 1: Unique Process of MAD

Data: Unit vector $v(x, y, z)$

Result: Unit vector after direction unification

```

1 if  $z < 0$  then
2    $v = -1 \times v$ ;
3   Return  $v$ ;
4 else if  $z == 0$  then
5   if  $y < 0$  then
6      $v = -1 \times v$ ;
7     Return  $v$ ;
8   else if  $y == 0$  then
9     if  $x < 0$  then
10        $v = -1 \times v$ ;
11       Return  $v$ ;
12 return  $v$ 

```

connected faces are also difficult to identify, leading to confusion for deep learning models.

E.4. CstPnt Module

This section provides a detailed explanation of the valid points in the computational steps that inspire the design of the CstPnt Module.

The CstPnt Module is used to predict constraint representation from point clouds. The design of the CstPnt Module is inspired by the following calculation process:

For a point p in point cloud, its constraint representation could be calculated by the following steps:

1. Identify neighbor points around p .


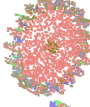
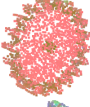
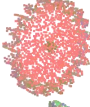
















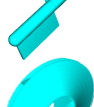
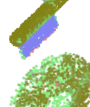
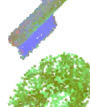
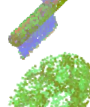






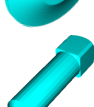
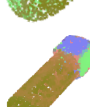






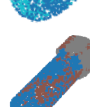
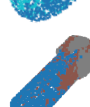

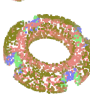

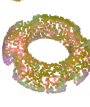
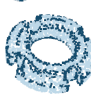
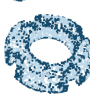
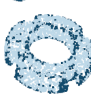

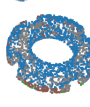
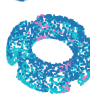




















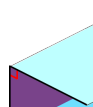

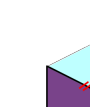
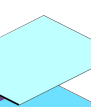

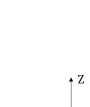
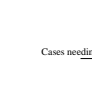
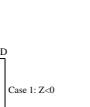
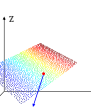
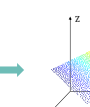
Model	MAD			Adj			PT		
	Ours	ParSeNet [11]	HPNet [15]	Ours	ParSeNet [11]	HPNet [15]	Ours	ParSeNet [11]	HPNet [15]
									
									
									
									
									
									
									
									

Figure 8. Constraint prediction on MCB.

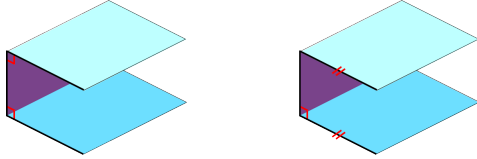


Figure 9. **Non-uniqueness of traditional constraint representation.** Left: Double vertical constraints, right: Vertical and parallel constraints.

2. From all neighbor points, identify those that belong to the same primitive as p ; these points are referred to as valid points.
3. Fit shapes such as cylinders or planes using valid points, and determine the primitive type with the smallest fitting error.
4. Based on the valid points and the primitive type, calculate the main axis direction and assess whether the point is near an edge.

In these steps, it is necessary to compute the valid points. We define the valid point as the neighbors that lie on the same primitive as the center point p , as visualized in Fig. 14.

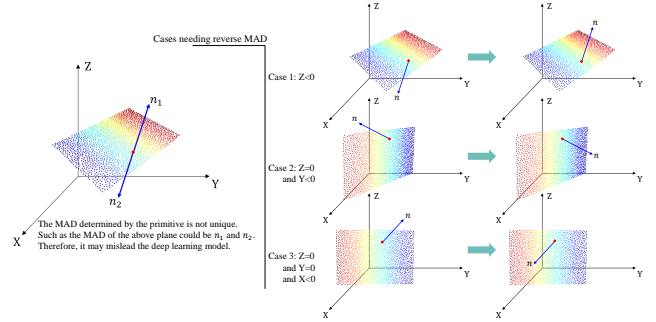


Figure 10. **Unique Process of MAD.**

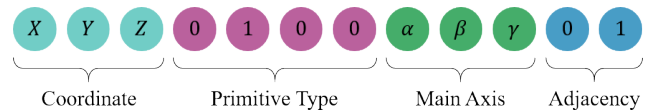


Figure 11. **Point with constraint representation.**

These valid points can be used to determine whether the center point is near an edge, as well as to calculate the main axis direction and primitive type of the attached primitive,

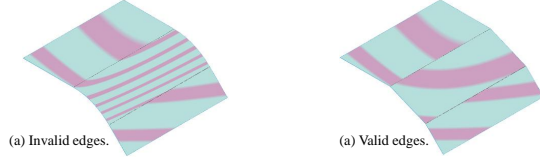


Figure 12. **Zebra stripes of primitives corresponding to invalid and valid edges.** Left: G1 or higher continuous at the invalid edges, right: G0 continuous at the valid edges.

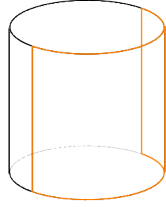


Figure 13. **Cylinder in BRep representation.** A complete cylindrical surface is represented as a combination of two half-cylinder surface, resulting in edges along the generatrices that cannot be recognized, thereby causing confusion for deep learning model.

while the invalid points do not contribute to these calculations.

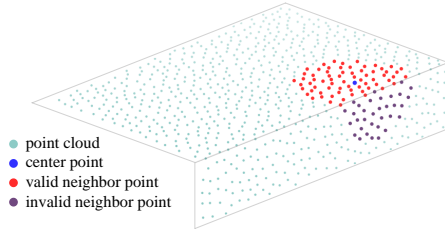


Figure 14. **Valid and invalid points for constraint representation calculation.**

E.5. SurfaceKNN

This section introduces the algorithm of SurfaceKNN.

SurfaceKNN based on the assumption that when KNN is applied with a small number of neighbors, the output points are searched along the shape's surface. This assumption has been validated in most cases. Leveraging this assumption, SurfaceKNN is accomplished by applying KNN with small number of neighbors iteratively. During each iteration, KNN is used to search a small set of neighboring points, which then serve as new central points for subsequent searches. This process continues until the desired number of points is obtained, as illustrated in Fig. 15.

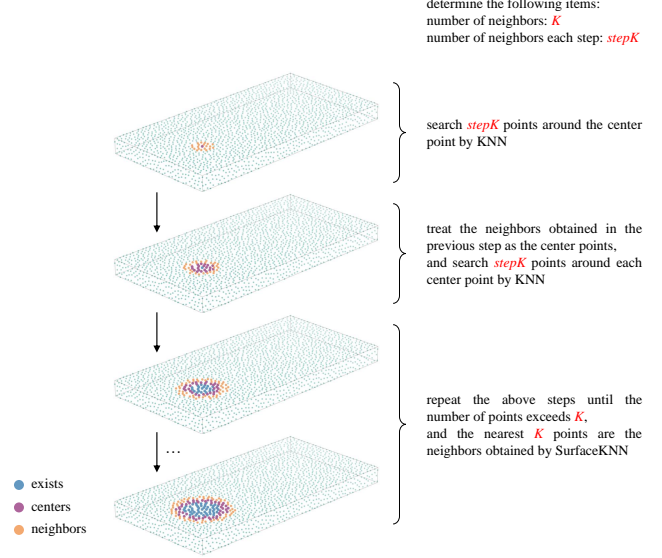


Figure 15. **Algorithm of SurfaceKNN.**

References

- [1] A. Angrish, B. Craver, and B. Starly. Fabsearch: A 3d cad model-based search engine for sourcing manufacturing services. *Journal of Computing and Information Science in Engineering*, 19(4), 2019. 2
- [2] S. Jayanti, Y. Kalyanaraman, N. Iyer, and K. Ramani. Developing an engineering shape benchmark for cad models. *CAD Computer Aided Design*, 38(9):939–953, 2006. 2
- [3] Sangpil Kim, Hyung-gun Chi, Xiao Hu, Qixing Huang, and Karthik Ramani. A large-scale annotated mechanical components benchmark for classification and retrieval tasks with deep neural networks. In *European conference on computer vision*, pages 175–191. Springer, 2020. 2
- [4] S. Koch, A. Matveev, Z. Jiang, F. Williams, A. Artemov, E. Burnaev, M. Alexa, D. Zorin, and D. Panozzo. Abc: A big cad model dataset for geometric deep learning. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 9593–9603, 2019. 2
- [5] J. G. Lambourne, K. D. D. Willis, P. K. Jayaraman, A. Sanghi, P. Meltzer, and H. Shayani. Brepnet: A topological message passing system for solid models. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 12768–12777, 2021. 2
- [6] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen. Pointcnn: Convolution on x-transformed points. In *Advances in Neural Information Processing Systems*, pages 820–830, 2018. 3
- [7] Z. H. Lin, S. Y. Huang, and Y. C. F. Wang. Learning of 3d graph convolution networks for point cloud analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(8):4212–4224, 2022. 3
- [8] Janusz Pobożniak. Algorithm for iso 14649 (step-nc) feature recognition. *Management and Production Engineering Review*, 4(4):50–58, 2013. 1
- [9] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep

- learning on point sets for 3d classification and segmentation. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, pages 77–85, 2017. [3](#)
- [10] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*, pages 5100–5109, 2017. [2](#), [3](#)
- [11] Gopal Sharma, Difan Liu, Subhransu Maji, Evangelos Kalogerakis, Siddhartha Chaudhuri, and Radomír Měch. *ParSeNet: A Parametric Surface Fitting Network for 3D Point Clouds*, page 261–276. Springer International Publishing, 2020. [2](#)
- [12] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics*, 38(5), 2019. [3](#)
- [13] W. Wu, Z. Qi, and L. Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 9613–9622, 2019. [3](#)
- [14] Y. Xu, T. Fan, M. Xu, L. Zeng, and Y. Qiao. Spidercnn: Deep learning on point sets with parameterized convolutional filters. In *Lecture Notes in Computer Science (including sub-series Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pages 90–105, 2018. [3](#)
- [15] Siming Yan, Zhenpei Yang, Chongyang Ma, Haibin Huang, Etienne Vouga, and Qixing Huang. Hpnet: Deep primitive segmentation using hybrid representations. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, page 2733–2742. IEEE, 2021. [2](#)
- [16] Long Zeng, Yong-Jin Liu, Sang Hun Lee, and Matthew Ming-Fai Yuen. Q-complex: Efficient non-manifold boundary representation with inclusion topology. *Computer-Aided Design*, 44(11):1115–1126, 2012. [2](#)