OXFORD CDE

Journal of Computational Design and Engineering, 2021, 8(1), 392–408

doi: 10.1093/jcde/qwaa086
Journal homepage: www.jcde.org
Advance Access Publication Date: 18 December 2020

RESEARCH ARTICLE

Unorganized point classification for robust NURBS surface reconstruction using a point-based neural network

Jinho Song [1], Junhee Lee[1], Kwanghee Ko [1,*], Won-Don Kim[2], Tae-Won Kang[2], Jeung-Youb Kim[3] and Jong-Ho Nam[4]

[1] The School of Mechanical Engineering, Gwangju Institute of Science and Technology, 61005, Gwangju, Republic of Korea; [2]Marine Tech-In, 48059, Busan, Republic of Korea; [3]ILJOO GnS Co LTD., 47866, Busan, Republic of Korea and [4] Division of Naval Architecture and Ocean Systems Engineering, Korea Maritime and Ocean University, 49112, Busan, Republic of Korea

*Corresponding author. E-mail: khko@gist.ac.kr http://orcid.org/0000-0001-7668-5796

Abstract

In this paper, a method for classifying 3D unorganized points into interior and boundary points using a deep neural network is proposed. The classification of 3D unorganized points into boundary and interior points is an important problem in the nonuniform rational B-spline (NURBS) surface reconstruction process. A part of an existing neural network PointNet, which processes 3D point segmentation, is used as the base network model. An index value corresponding to each point is proposed for use as an additional property to improve the classification performance of the network. The classified points are then provided as inputs to the NURBS surface reconstruction process, and it has been demonstrated that the reconstruction is performed efficiently. Experiments using diverse examples indicate that the proposed method achieves better performance than other existing methods.

Keywords: point classification; boundary point; corner point; neural network; surface reconstruction

1. Introduction

Reconstructing a surface from an unorganized point set is an important problem in computer-aided design, reverse engineering, geometric modeling, etc., and various studies have discussed this problem (Straßer, Klein, & Rau, 2012). In particular, finding a nonuniform rational B-spline (NURBS) surface that fits the points has been a core problem in reverse engineering.

The NURBS surface fitting problem requires parametrization and computation of the control points when knot vectors, degrees of the basis functions, and weights are given. Parametrization can be performed using methods such as the chord length (Hoschek & Lasser, 1993), centripetal (Hoschek & Lasser, 1993), and base surface methods (Ma & Kruth, 1995). The control points can be computed in the least squares sense using the parameter values of the points.

A NURBS surface can be defined as a map between a 2D parametric domain and a subset of 3D space. The map is given as a differentiable function of two parameters, which is defined over a rectangular domain in the 2D parametric space. In the parametrization step, it is important to know whether a point on the surface is on the boundary or at a corner because the boundary or corner points

Received: 11 August 2020; **Revised:** 17 November 2020; **Accepted:** 18 November 2020

define the domain of the surface in 3D space, and because the parameter values for those points should be on the boundary of the rectangular domain in the parametric space. A NURBS surface patch in 3D space with two parameters u and v is defined as a map from a parametric domain, the shape of which is a rectangle ($0 \leq u,\ v \leq 1$). Each boundary line of the domain corresponds to each boundary curve of the NURBS surface. For example, the points corresponding to ($u = 0,\ v$) are one of the boundary lines of the rectangular parametric domain, which maps a boundary curve of the surface in 3D space. Because the map is bijective, the boundary of the NURBS surface corresponds to the parametric domain boundary. Therefore, if points in 3D space are identified as boundary points, they should be mapped to the boundary of the rectangular domain.

The points on the NURBS surface can be classified into boundary and interior points. Let $\mathbf{S} \subset \mathbf{R}^3$ be a NURBS surface, where $\mathbf{R}^3$ is the 3D space.

Definition 1.1. The interior of $\mathbf{S}$, denoted as $int(\mathbf{S})$, is the set of points $\mathbf{p}$ in $\mathbf{R}^3$ such that the neighborhood of $\mathbf{p}$ contains points in $\mathbf{S}$.

Note that the neighborhood is homeomorphic to the plane in 2D space. Similarly, we present the following definition:

Definition 1.2. The boundary of $\mathbf{S}$, denoted as $\partial\mathbf{S}$, is defined as the set of points $\mathbf{p}$ in $\mathbf{R}^3$ such that the neighborhood has points in $\mathbf{S}$ and in $\mathbf{R}^3 - \mathbf{S}$.

Among the boundary points, there are four corner points that correspond to the four corners of the rectangular region in the parametric domain.

Suppose that $\mathbf{P}$ is a set of points sampled from $\mathbf{S}$. Then, the fitting problem is reduced to finding $\mathbf{S}$ from $\mathbf{P}$. However, it is difficult to obtain $\mathbf{S}$ from $\mathbf{P}$ because $\mathbf{P}$ is not identical to $\mathbf{S}$. Instead, we obtain $\mathbf{S}^*$, which approximates $\mathbf{S}$. When $\mathbf{S}^*$ is defined as a NURBS surface, the problem can be solved through the following three steps:

1. Classification of the corners, boundary points, and interior points in the set.
2. Parametrization that maps the classified points onto the parametric domain.
3. Computation of the control points and weights to create knot vectors fitting the given points.

The points in $\mathbf{S}$ can be classified as corners, boundary points, or interior points. Once each point is classified correctly, its parametrization can be properly performed over the parametric domain. The control points of $\mathbf{S}^*$ can then be computed in the least squares sense.

The first step is essential toward solving the problem because it provides the correct topological properties of each point that in turn define the topological structure of the NURBS surface. The reconstruction accuracy highly depends on the classification result because an incorrect classification of any corner, boundary, or interior point can negatively affect the parametrization step, leading to an undesirable fitting result. The point classification problem can be immediately solved if the point set is organized or contains well-defined meshes, because the boundaries of the points are already identified. However, when the points are not organized, all points should be classified based on their positions and distribution patterns.

Point classification has been used for the computation of the boundary points for a 2D or 3D input point set on the same plane, which is an important task in computational geometry. A convex hull algorithm is applied to find the smallest convex hull set of the input shape using various approaches, such as those proposed by Graham (1972), Jarvis (1973), and Barber, Dobkin, and Huhdanpaa (1996). It works well in most cases. However, it can only find a convex set from the boundary points, and cannot find all boundary points from a nonconvex set. To solve this problem, a generalization of the convex hull algorithm, called the alpha-shape, was introduced by Edelsbrunner, Kirkpatrick, and Seidel (1983). Suppose that a point set $\mathbf{P}$ is given. First, a generalized disk of radius $1/\alpha$ is defined. If $\alpha = 0$, the alpha-shape approach is reduced to the original convex hull algorithm. If $\alpha > 0$, the generalized disk becomes a closed disk of radius $1/\alpha$. Otherwise, the disk becomes the complement of the disk with a radius of $-1/\alpha$.

Then, for $\mathbf{p}_i \in \mathbf{P}$, the alpha-shape approach draws the generalized disks with a radius of $1/\alpha$, and the edge is drawn if the generalized disk contains no points in $\mathbf{P}$ and if $\mathbf{p}_i,\ \mathbf{p}_j \in \mathbf{P}$ lie on its boundary. This process is repeated for all points, and an alpha-shape graph is obtained, which contains the boundary points. The drawback of this method is that it highly depends on the input parameter α, and an optimal value for α is not provided in the literature. There exist some search algorithms for a concave hull set that use the same parameter α, the number of the neighborhood of a point K, or a length parameter l as the input parameters. However, they also share the same problem associated with the alpha-shape approach (Moreira & Santos, 2007; Duckham, Kulik, Worboys, & Galton, 2008; Asaeedi, Didehvar, & Mohades, 2017) and may yield different boundary point detection (BPD) results depending on their input parameters. Also, Lee and Bo (2016) proposed a feature curve extraction method from computer-aided design (CAD) models using a linear approximation technique and developable surface fitting, but all feature curves are not boundary points, and the method requires a predefined threshold, an angle to compare with the dihedral angle of the intersection segments for feature line segment extraction. In addition, the corner point detection (CPD) will fail for point sets on the plane since the algorithm defines corner points as intersection of at least three lines.

Recently, Mineo, Pierce, and Summan (2019) proposed an automated BPD algorithm that computes the boundary point candidates based on the local point cloud resolution β for each point and determines whether it is an interior point. The 30 points closest to each boundary point are projected onto the 2D plane and converted into polar coordinates. Here, each boundary point becomes the pole of the polar coordinates. Then, the algorithm attempts to create a path that surrounds the pole or the boundary point using an incremental approach. The algorithm evaluates and updates the angle α, which is spanned by the path that links the neighbor points around the pole. The algorithm stops when all neighborhood points are exhausted or the absolute sum of the angles $\sum \|\Delta\alpha\|$ reaches 2π. If the sum of the angles is $\sum \Delta\alpha > 2\pi$, the BPD algorithm concludes that the point being evaluated is an interior point. Otherwise, the algorithm considers it to be a boundary point. Mineo *et al.* explained that the final step of the BPD algorithm determines the boundary point of a hole in the point cloud when the hole's radius is β. This algorithm works well for general point sets. However, the final step of the algorithm can incorrectly identify an interior point as a boundary point of the hole when the points are distributed with a point cloud resolution equal to β. In addition, the algorithm can also fail for a sparse point cloud because it requires 30 neighborhood points for each boundary point to create a path, and may fail to include a sufficient number of neighborhood points to draw the correct path.

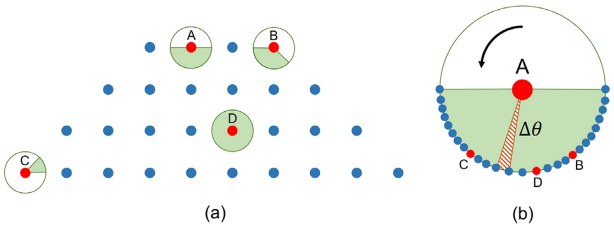

Figure 1: Illustration of the index I_c concept for an arbitrary point set. (a) A is a boundary point, B and C are corner points, and D is an interior point. (b) The magnified index circular sector of point A is represented by $\Delta\theta$ and the intersections of the other points are shown on the boundary of the circle.

This paper focuses on classifying 3D points defining the surface of a hull plate, the outer shell of a ship that is topologically equivalent to a rectangle. The underlying geometric entity defined by the points consists of four corner points, four boundary edges (boundary curves), and a face. The input points are sparse and unorganized. The proposed algorithm uses a part of the deep neural network PointNet (Qi, Su, Mo, & Guibas, 2017a) as the base neural network, which was proposed to solve the point segmentation problem. PointNet uses 3D coordinates and optionally RGB or normal values for the segmentation of a point cloud. However, an index value at each point is proposed for use in the network to improve classification performance in this study. The index value represents the topological property of each point. Therefore, it is invariant to rigid body transformations. Experiments with various data point sets indicate that the proposed method is superior to existing methods.

2. Methodology

In this section, the technical details of the proposed method are presented. We assume that the input is a sparse unorganized point set $\mathbf{P} = \{\mathbf{p}_i \mathbf{p}_i = (x_i, y_i, z_i),\ i = 0, \cdots, n\}$ that lies on the surface $\mathbf{S}$ in $\mathbf{R}^3$. The input point set may contain duplicate points that must be removed. Given two points $\mathbf{p}_i$ and $\mathbf{p}_j$, they are determined to be duplicated if $|\mathbf{p}_i - \mathbf{p}_j| < \delta$, where δ is a small number provided by the user.

The method for computing the index values at the points in $\mathbf{P}$ is presented along with the neural network used for 3D point classification.

2.1 Index computation and corner point classification

Given three distinct points $\mathbf{p}_i$, $\mathbf{p}_j$, and $\mathbf{p}_c$ in $\mathbf{P}$, the angle $\Delta\theta_{i,c,j} = arg(\mathbf{p}_i, \mathbf{p}_c, \mathbf{p}_j)$ is formed by the two vectors $\mathbf{p}_i - \mathbf{p}_c$ and $\mathbf{p}_j - \mathbf{p}_c$. We consider the points in the neighborhood of $\mathbf{p}_c \in \mathbf{P}$ including $\mathbf{p}_c$, denoted by $\mathbf{P}_c$. The *index* at $\mathbf{p}_c$ is then defined as follows:

$$I_c = \frac{1}{2\pi} \sum_{k=0}^{n-1} \Delta\theta_{k,c,k+1},\ \mathbf{p}_k \in \mathbf{P} - \{\mathbf{p}_c\}, \tag{1}$$

where $n+1$ is the number of points in $\mathbf{P}_c$. The index is inspired by the rotation index or topological degree (winding number), the number of complete rotations for a closed curve around the point on a plane, which was used in various mathematical applications (Peter 2018; Boult & Sikorski, 1989; Sakkalis, 1990; Ko, Sakkalis, & Patrikalakis, 2008).

Equation (1) can be computed as follows: Suppose that we have a plane Π on which the projected area of the convex hull of $\mathbf{P}_c$ is maximized. We project $\mathbf{P}_c$ onto Π to obtain $\mathbf{P}_c^{proj}$. We take one point $\mathbf{p}_c^{proj} \in \mathbf{P}_c^{proj}$, which is the projected point of $\mathbf{p}_c$, and the centroid $\mathbf{p}_0^{proj}$ of $\mathbf{P}_c^{proj}$. Then, we define a reference axis by connecting them. Next, we define a circle $\mathbf{C}$, whose respective radius and center are $\rho = |\mathbf{p}_c^{proj} - \mathbf{p}_0^{proj}|$ and $\mathbf{p}_c^{proj}$. We define a line $\mathbf{L}_k$ connecting $\mathbf{p}_k^{proj}$, where $\mathbf{p}_k^{proj} \in \mathbf{P}_c^{proj} - \{\mathbf{p}_c^{proj}\}$ and the centre point of a circle $\mathbf{p}_c^{proj}$. Then, we compute the intersection point between $\mathbf{C}$ and $\mathbf{L}_k$, which is denoted by $\hat{\mathbf{p}}_k^{proj}$. We repeat this process to compute all the points $\hat{\mathbf{p}}_k^{proj}$ and sort them based on $\Delta\theta_{0,c,k}$ in the counterclockwise direction from $k = 0$, as illustrated in Fig. 1. Here, we only consider the points $\hat{\mathbf{p}}_k^{proj}$ and $\hat{\mathbf{p}}_{k+1}^{proj}$ in the index computation so that the triangle formed by $\hat{\mathbf{p}}_k^{proj}$, $\hat{\mathbf{p}}_{k+1}^{proj}$, and $\hat{\mathbf{p}}_c^{proj}$ does not contain any point in $\mathbf{R}^3 - \mathbf{S}$, depicted by the shaded regions in Fig. 1. Then, we have

$$\begin{cases} I_c = 1, & if\ \mathbf{p}_c^{proj} \in int(\mathbf{S}) \\ 0 \le I_c < 1, & if\ \mathbf{p}_c^{proj} \in \vartheta\mathbf{S} \end{cases}. \tag{2}$$

It can be observed that the index values for A, B, and C are less than one, and that of D is equal to one, as shown in Fig. 1. The index value is a topological property and can be used for classification.

After the index values are computed, we select the four points having the smallest index values as corner points. This strategy works mostly because the corner points are surrounded by a lower number of points than other points, which we can verify from Fig. 1a. Note that the proposed CPD strategy can be extended to a n-vertex shape if the number of corner points is known because we can find n corner points that have the n smallest index values. The CPD method works for general shapes but may fail if a corner point is located at a concave region such as a dart (arrowhead) shape, as illustrated in Fig. 2. The index value at Point A becomes larger than that at Point B. Therefore, it is likely to be classified as a boundary point or an interior point. However, a dart (arrowhead) shape is not used in the target application. Therefore, this failure case can be ignored.

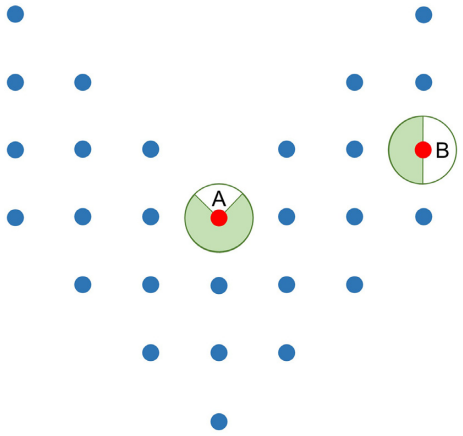

Figure 2: Failure case for CPD involving an arrowhead shape.

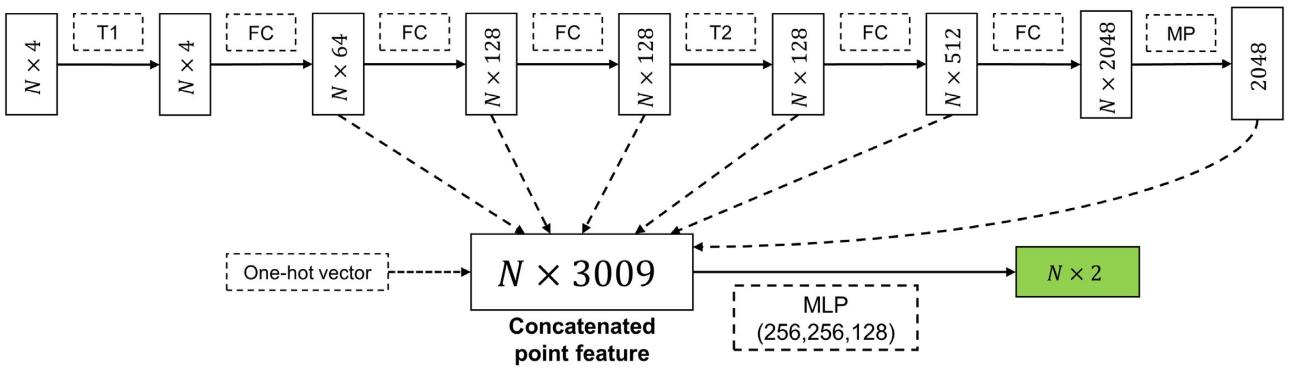

Figure 3: Part segmentation network architecture of PointNet, where T1 and T2 represent alignment/transformation networks; FC represents a fully connected layer, MP represents a max-pooling layer, and MLP represents a multilayer perceptron. The dotted arrows represent skip links that concatenate the outputs of the layers to the aggregated point features.

2.2 Deep neural network for 3D point classification

Neural networks are used in the classification of 3D points. 3D input points are converted into 3D voxel grids or images in order to feed them into deep learning neural networks such as multiview convolutional neural networks (CNNs; Su, Maji, Kalogerakis, & Learned-Miller, 2015; Qi *et al.*, 2016) or volumetric CNNs (Maturana & Scherer, 2015; Qi *et al.*, 2016). The problem with these methods is that point conversion causes a loss of geometric information and results in high computational complexity for volumetric representation. PointNet was proposed to address this problem (Qi *et al.*, 2017a). PointNet is a deep learning network that directly considers a 3D point set as its input for object classification or segmentation.

PointNet takes a set of 3D vectors as input. In addition, the normal vectors or RGB information at each point can be added to the vectors. PointNet uses a symmetric function in the max-pooling layer for unordered points, combines the local and global information to include point interaction information, and uses two alignment/transformation networks to make the points invariant to transformations in the network structure. As a result, the network can successfully perform classification or segmentation tasks. However, it does not capture the local structure well because it takes the whole point set as its input, which has since been addressed by PointNet++ (Qi, Yi, Su, & Guibas, 2017c). In addition, it treats PointNet as one layer and recursively uses PointNet to learn the local structure by taking subsampled point sets as input in the set abstraction levels. It produces point scores for the original input by interpolating the subsampled point sets and concatenating the point features from the set abstraction levels to achieve the part segmentation task. In this study, we use the part segmentation network of PointNet to classify 3D points (Fig. 3).

In fact, using the subsampled point sets obtained from the sampling and grouping layers in the set abstraction levels as input would make it difficult to train the network because the network would observe only part of the original point set when attempting to detect boundary points from the original input point set. Thus, we do not use the subsampling strategy for the input point sets to learn the local structure and instead take the whole point set as input for point classification. In addition, PointNet provides an option to use the normal or RGB information of each point. However, we do not use either option because neither RGB color nor normal information would be helpful for classification; RGB information is not helpful for point classification because most hull plates are single colored. Also, normal information is not helpful in classification. Unless there exist points representing the thickness of the plate, the normal vectors are distributed continuously. Therefore, they do not give a clue of whether a point belongs to a boundary or not. Instead, we use the index value as additional information to improve classification performance. PointNet is claimed to classify

3D points because it uses the local and global relationships of each point with respect to its neighborhood (Qi *et al.*, 2017a). However, such information is mostly based on the positions of the points.

The index value is a topological entity for each point, which adds information to the per-point feature and provides strong local and global information that shows the relationships with other points. Therefore, using the per-point feature with no modification required to PointNet can improve the classification performance.

The input of the part segmentation network of PointNet is a set of unorganized 4D point sets $\mathbf{P}^I = \{\mathbf{p}_1^I,\ \mathbf{p}_1^I,\ \cdots,\ \mathbf{p}_N^I\}$, where N is the number of points in $\mathbf{P}^I$, $\mathbf{p}_i^I = (\mathbf{p}_i,\ I_i)$, $\mathbf{p}_i = (x_i,\ y_i,\ z_i)$, and I_i is the index value at $\mathbf{p}_i$.

Then, the alignment/transformation network that predicts an affine transformation matrix is first applied to the input, so the input becomes invariant to various geometric transformations. Next, three fully connected layers are consecutively applied to the transformed point set to produce point sets with sizes $N \times 64$, $N \times 128$, and $N \times 128$. Second, the alignment/transformation network for feature transformation is applied to the $N \times 128$ point set, and two fully connected layers are consecutively applied to the transformed point features, resulting in a $N \times 512$ point set and a $N \times 2048$ point set. Then, the max-pooling layer, which is a symmetric function that aggregates feature information, is applied to produce a global point feature vector with a size of 2048. The output of the max-pooling layer is concatenated to the outputs of a fully connected layer, the second alignment/transformation network, and a one-hot vector that represents the class of the input. Because the number of classes is one, the size of the one-hot vector is also one, producing a concatenated point feature vector with a size of $N \times 3009$.

Finally, the multilayer perceptron with layer sizes (256, 256, and 128) is applied to each point of the concatenated output. Note that a combination of batch-norm and the rectified linear units (ReLU) activation function are used for all layers. The part segmentation network is designed to produce $N \times 2$ scores because we only classify each point as a boundary or interior point. The architecture of the part segmentation of PointNet is shown in Fig. 3. For more details regarding PointNet, please refer to the studies by Qi *et al.* (2017a) and Qi, Su, Mo, and Guibas (2017b).

2.2.1 *Training data preparation*

We synthetically generate curved plate models for training dataset as follows: First, we generate four non-collinear points to obtain four corner points. Then, we compute lines connecting two adjacent corner points to define the boundary of the surface. Next, we randomly sample the lines and generate the boundary points on the lines. The number of points for each boundary line is determined by the input grid size $n \times n$. We set the minimum grid size to 9×9, and the maximum grid size to 17×17. In the next step, we generate uniform interior points of $(n-2) \times (n-2)$, and similarly add points that are randomly sampled from the interior of the surface. As a result, we obtain the point set $\mathbf{P}^*$, which is composed of points $\mathbf{p}_i^* = (x_i,\ y_i,\ z_i)$ with additional random points, where z_i represents the depth value of the xy plane. Once the initial grid point set is generated, we modify its shape to one of four shape types: plane, elliptic paraboloid, hyperbolic paraboloid, and parabolic cylinder, which are general shapes of actual hull plates. If the selected shape is planar, we leave it as is. If we select other shapes, we modify the depth values z_i by using elliptic, hyperbolic paraboloid, or parabolic cylinder equations depending on the selected shape type. The applicable relationships are

$$\frac{z}{c} = \frac{x^2}{a^2} + \frac{y^2}{b^2},\quad \frac{z}{c} = \frac{y^2}{b^2} - \frac{x^2}{a^2},\quad \frac{z}{c} = x^2\ ,\ a, b \in [-1.0, 1.0]\,, c \in [0.0, 1.0]\,, \tag{3}$$

where a and b are random coefficients between -1.0 and 1.0, and c is a random scaling factor between 0.0 and 1.0. After modifying the plate shape, we apply a random rotation matrix to the point set $\mathbf{P}^*$ to obtain a curved hull plate with labels 0, 1, which, respectively, represent each point as a boundary or interior point. Then, we compute index values I_i for each point using equation (1) and add these to the generated point set $\mathbf{P}^*$. Fig. 4 shows the generation procedure for a synthetic hull plate model with a grid size of 13×13. We repeat this procedure to create the dataset, which is composed of the four shape types mentioned previously. We manually reviewed each generated hull plate and removed a plate if the shape of the generated plate was distorted. A total of 125 hull plates were created for each shape type, and for each shape, 25 hull plates were created for each grid size, which was composed of five different sizes: $9 \times 9, 11 \times 11, 13 \times 13, 15 \times 15$, and 17×17. Thus, the number of generated point sets was 500, and we added 72 actual hull plates from 9 ship hull models to the dataset, so that the total training dataset consisted of 572 examples. All plate data are normalized into a unit sphere as in Qi *et al.* (2017a) for better training. Fig. 5 shows some of the generated hull plates and Fig. 6 shows ship block models used for training and validation.

2.2.2 *Validation set selection and training procedure*

In order to select a validation set from the training set, K-fold cross-validation is used (James, Witten, Hastie, & Tibshirani, 2013). K-fold cross-validation is a widely used evaluation method in statistics to show the performance of a model. The procedure of the validation is as follows: First, we randomly shuffle the entire dataset and set K to 5 since it is a commonly used K parameter. The dataset is split into 5 folds of similar or equivalent size. We choose 1 fold as a validation set and use the remaining 4 $(K-1)$ folds as a training set for training. After one training is finished, we obtain accuracies and initialize the model. Then, we choose the next fold as a validation and use the remaining folds as a training set for another training. The training procedure is repeated 5 times since there are 5 folds, and we collect the accuracies to compute the average for evaluation.

For each training, a GeForce RTX 2080Ti GPU was used, requiring approximately 30 minutes to process a given dataset. The batch size was 5, with an epoch size of 50. The Adam optimizer was used with a weight decay rate of 0.0001 and a learning rate of 0.001, which was decreased to 0.00025, as the learning rate was divided by 2 every 20 epochs (Kingma & Ba, 2014). The size of the input data was fixed at 400.

Table 1 shows the results of 5-fold cross-validation for each fold, where T_{ACC} and V_{ACC} represent the train and validation accuracies, respectively, and T_{Loss} and V_{Loss} represent the final loss function values for the training and validation sets. As we can observe from Table 1, all accuracy values are around 97%, which shows high performance. In addition, all train and validation accuracies are almost identical to each other. There is almost no difference between the train and validation accuracies, which indicates that no overfitting

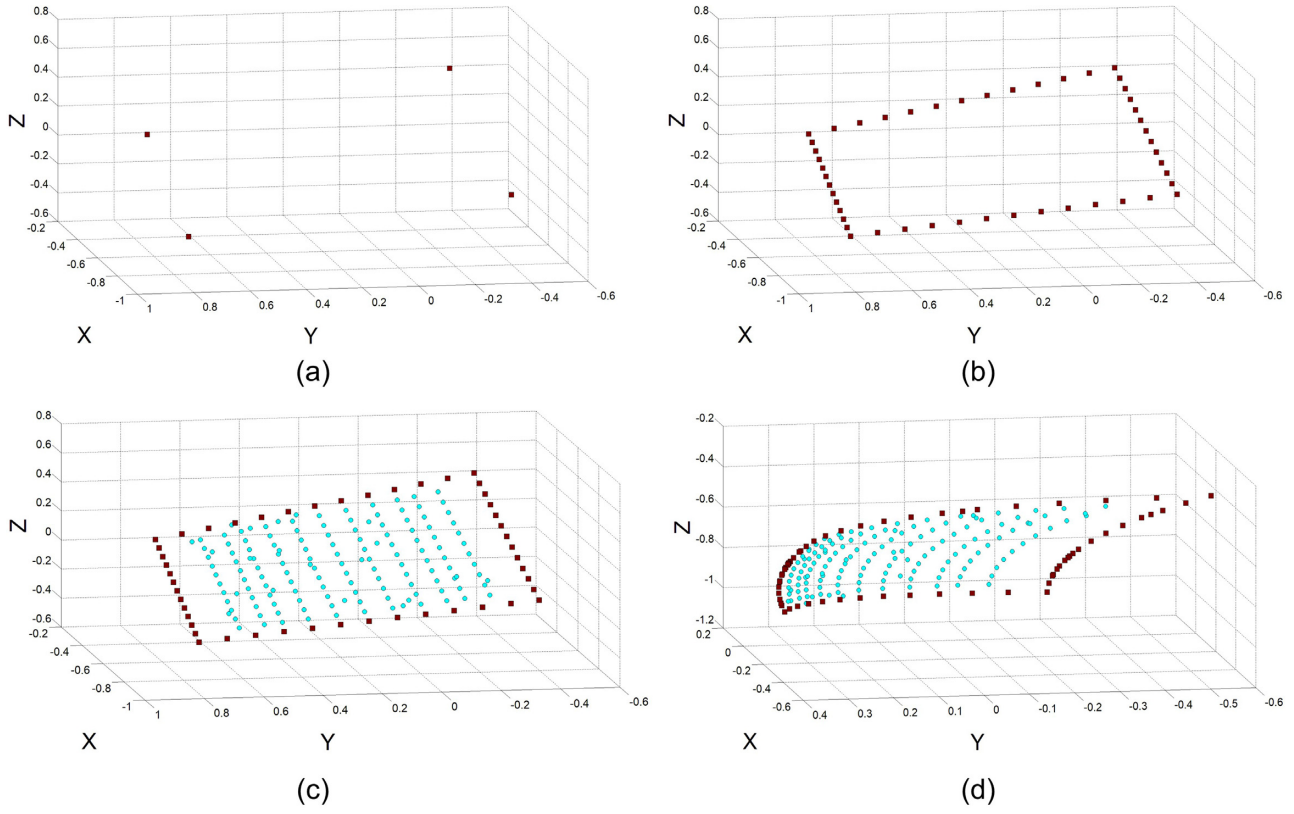

Figure 4: Example of curved hull plate generation. (a) Corner point generation, (b) boundary generation with random points, (c) interior point generation with random points, and (d) transformation to an elliptic paraboloid.

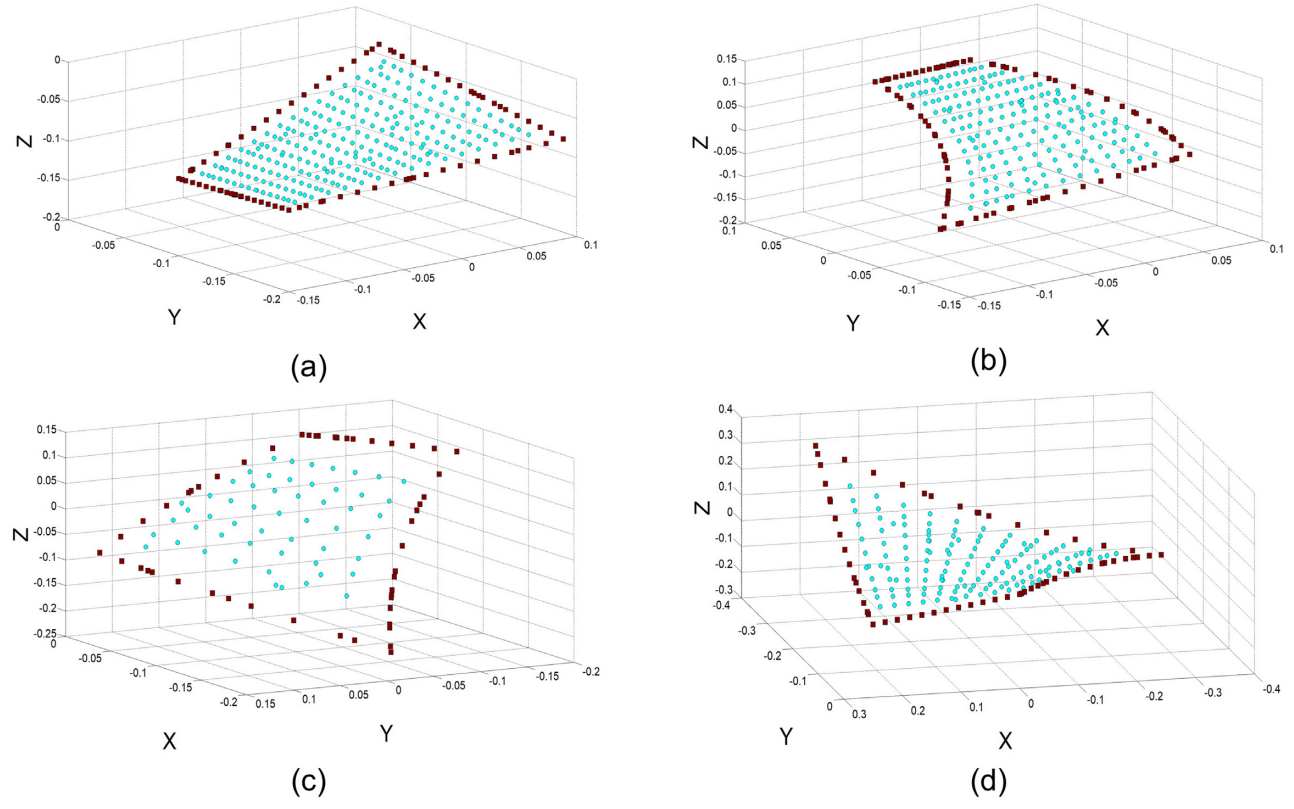

Figure 5: Examples of generated hull plate data. (a) Planar, (b) parabolic, (c) elliptic, and (d) hyperbolic shapes.

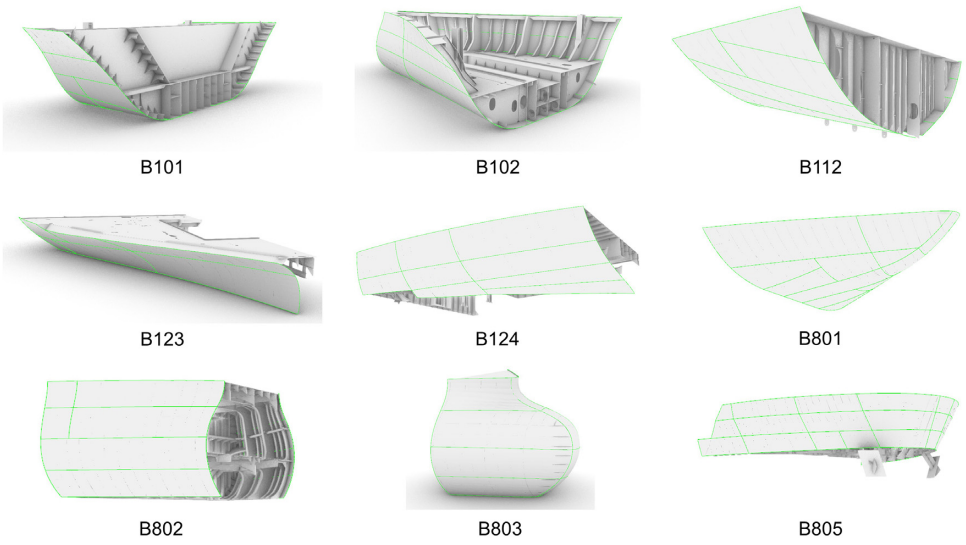

Figure 6: Examples of ship hull CAD models used in training and validation dataset.

Table 1: Evaluation result of 5-fold cross-validation.

Fold	T_{ACC}	V_{ACC}	T_{Loss}	V_{Loss}
1	97.64	97.98	0.073	0.064
2	97.35	98.68	0.078	0.048
3	97.75	97.73	0.074	0.057
4	97.81	97.60	0.070	0.067
5	97.70	97.75	0.076	0.054
AVG	97.65	97.95	0.074	0.058

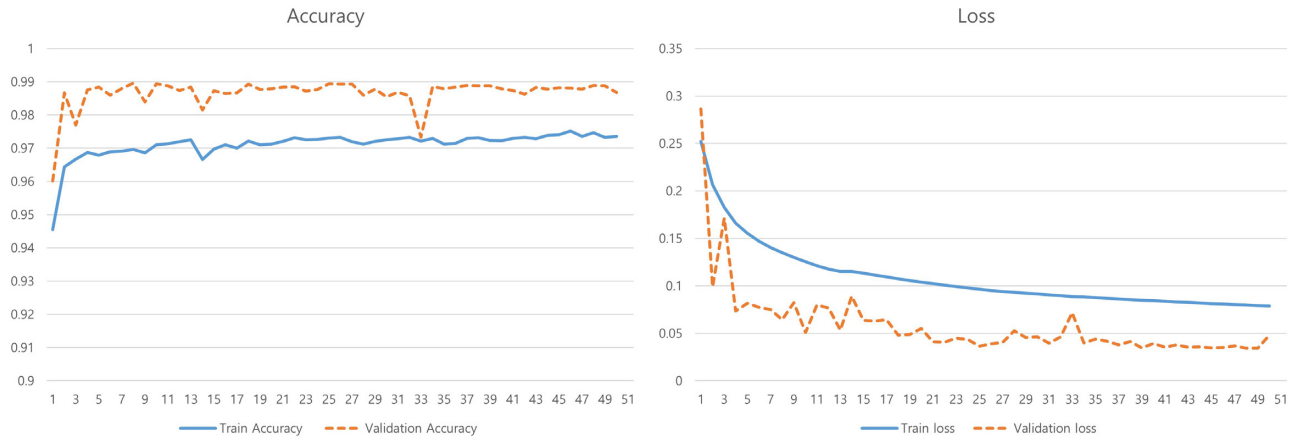

Figure 7: Graphs of accuracy and loss versus epochs for training and validation for the second fold.

and underfitting occurred during each training. Similarly, all loss function values are almost identical to each other, and there is also almost no difference between the train and validation loss values. Therefore, the network model is well trained, achieving the average train and validation accuracies of 97.65% and 97.95%, respectively, and the dataset is reliable to use since all the accuracies and loss values are similar to each other, as shown in Table 1. We choose the network model of the second fold, which includes 115 plates for the validation set and 457 plates for the training set since it shows the highest average accuracy of training and testing among the 5 folds, which is 98.01%.

Fig. 7 shows the changes in accuracy during training and validation versus the number of epochs for the second fold. The loss function values during training are shown in the right image of Fig. 7, which indicates that the overall trend of the loss function decreases over time. Fig. 7 implies that the network has been properly trained for point classification.

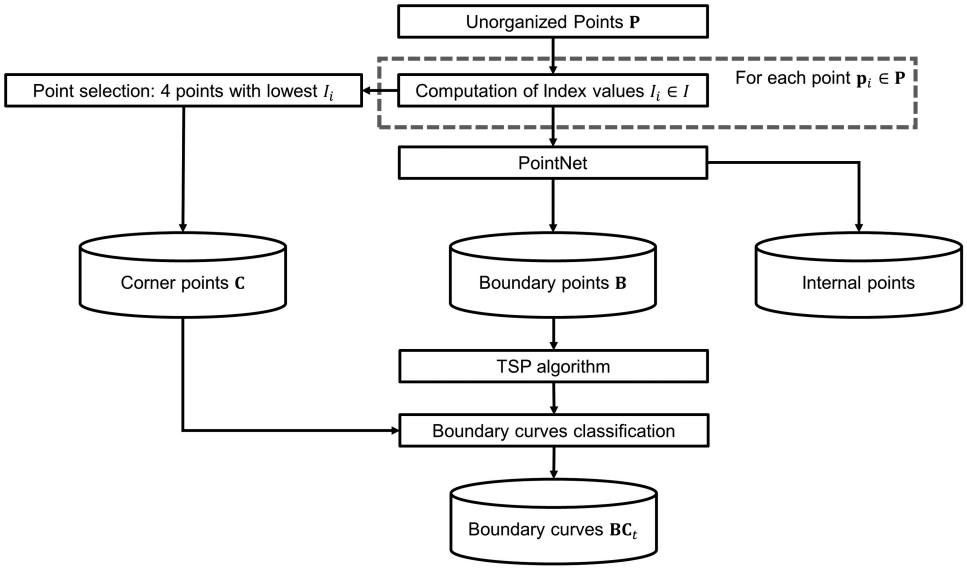

Figure 8: Flowchart of the proposed framework for point classification.

2.3 Boundary curve classification

The network only determines whether a point is a boundary point or an interior point. A NURBS surface consists of four corner points and four boundary curves. Therefore, it is necessary to further process the classified boundary points **B** and corner points **c** to determine which boundary the points belong to for NURBS surface reconstruction.

First, we organize **B** and **c** by ordering them. In this study, the traveling salesman problem algorithm is used. When K unorganized cities and the distances between them are given, this algorithm finds the shortest possible route that crosses each city exactly once and returns to the origin city (Schrijver, 2005). The cities are the boundary points in our case. Note that the simulated annealing method is used to solve the problem (Kirkpatrick, Gelatt, & Vecchi, 1983; Kirkpatrick 1984). After the boundary points are ordered, we can immediately obtain four ordered boundary curves $\mathbf{BC}_t$ $(t = 0, 1, 2, 3)$ based on the corner points. Fig. 8 shows the flowchart for the boundary and CPD process.

2.4 NURBS surface reconstruction

Given the corner, boundary, and interior points, a NURBS surface approximating the points can be constructed. The target NURBS surface $\mathbf{S}(u, v)$ is of degree 3 in both parametric directions, having $m \times m$ control points. The value of m is determined using

$$m = \begin{cases} \lfloor \sqrt{N-K} \rfloor + 2, & \text{if } \sqrt{N-K} \text{ is an integer} \\ \lfloor \sqrt{N-K} \rfloor + 1, & \text{otherwise,} \end{cases} \tag{4}$$

where K is the number of boundary points.

The overall process for surface reconstruction is as follows: We construct four NURBS boundary curves from $\mathbf{BC}_t$ in the least squares sense. The chord length parametrization method is used (Hoschek & Lasser, 1993) for parametrization. The number of control points of each boundary curve is m, and the degree is 3. Therefore,

$$\mathbf{BC}(t) = \sum_{l}^{m} \mathbf{Q}_l N_{l,4}(t), \tag{5}$$

where $N_{l,4}(t)$ is a NURBS basis function of degree 3 with the parameter t $(0 \le t \le 1)$ and with knot vector $\mathbf{T}$.

Next, we compute a NURBS surface from the boundary curves using the Coons patch algorithm and update the control points of the surface so that the surface approximates the interior points as closely as possible (Farin, 2001). A Coons patch is a parametric surface with u and $(0 \le u, v \le 1)$, which is obtained from the four boundary curves using

$$\mathbf{S}(u, v) = \sum_{i=0}^{n_u-1} \sum_{j=0}^{n_v-1} \mathbf{Q}_{i,j} N_{i,4}(u) \; N_{j,4}(v), \tag{6}$$

where $\mathbf{T}_u$ and $\mathbf{T}_v$ are knot vectors for u and v, respectively.

The control points of the boundary curves are $\{\mathbf{Q}_{0,j}\}_{j=0}^{n_v-1}$ for $u = 0$, $\{\mathbf{Q}_{n_u-1,j}\}_{j=0}^{n_v-1}$ for $u = 1$, $\{\mathbf{Q}_{i,0}\}_{j=0}^{n_u-1}$ for $v = 0$, and $\{\mathbf{Q}_{i,n_v-1}\}_{j=0}^{n_u-1}$ for $v = 1$, where $n_u, n_v = m$. Then, we compute the control points $\mathbf{Q}_{i,j}$ $(1 \le i \le n_u - 2)$ and $(1 \le j \le n_v - 2)$ of the base surface by applying the Coons patch algorithm to the boundary curves (Farin, 2001).

$$\mathbf{Q}_{i,j} = \begin{bmatrix} 1-u_i & u_i \end{bmatrix} \begin{bmatrix} \mathbf{Q}_{0,j} \\ \mathbf{Q}_{n_u-1,j} \end{bmatrix} + \begin{bmatrix} \mathbf{Q}_{i,0} & \mathbf{Q}_{i,n_v-1} \end{bmatrix} \begin{bmatrix} 1-v_j \\ v_j \end{bmatrix} - \begin{bmatrix} 1-u_i & u_i \end{bmatrix} \begin{bmatrix} \mathbf{Q}_{0,0} & \mathbf{Q}_{0,n_v-1} \\ \mathbf{Q}_{n_u-1,0} & \mathbf{Q}_{n_u-1,n_v-1} \end{bmatrix} \begin{bmatrix} 1-v_j \\ v_j \end{bmatrix} \tag{7}$$

Here, u_i and v_j are defined as

$$u_i = \frac{i}{n_u - 1}\ (i = 0, \ldots, n_u), \quad v_j = \frac{j}{n_v - 1}\ (j = 0, \ldots, n_v). \tag{8}$$

The computed control points $\mathbf{Q}_{i,j}$ can then be obtained from the boundary curves. The surface $\mathbf{S}(u, v)$ must be refined to represent the given points by adjusting the control points. Here, we assume that the control points on the boundary remain fixed because they are obtained by interpolating the given boundary points. Instead, we update the control points $\mathbf{Q}^{in} = \{\mathbf{Q}_{i,j}\}_{i=1,\ j=1}^{n_u-2,\ n_v-2}$ to approximate the interior points.

$\mathbf{p}_k$ is an interior point classified by the network so that

$$\mathbf{p}_k = \mathbf{S}(u_k, v_k) = \sum_{i=0}^{n_u-1}\sum_{j=0}^{n_v-1} N_{i,4}(u_k)\ N_{j,4}(v_k)\,\mathbf{Q}_{i,j}, \tag{9}$$

where u_k and v_k are the parametric values for $\mathbf{p}_k$. The parametric value for $\mathbf{p}_k$ is assumed to be given. In this study, we are given the control points corresponding to the boundary curves of $\mathbf{S}$. Therefore, equation (9) is reduced to

$$\mathbf{p}_k - \mathbf{H}(u_k, v_k) = \sum_{i=1}^{n_u-2}\sum_{j=1}^{n_v-2} N_{i,4}(u_k)\ N_{j,4}(v_k)\,\mathbf{Q}_{i,j}, \tag{10}$$

where

$$\mathbf{H}(u_k, v_k) = \sum_{j=0}^{n_v-1} N_{0,4}(u_k)\,N_{j,4}(v_k)\,\mathbf{Q}_{0,j} + \sum_{j=0}^{n_v-1} N_{n_u-1,4}(u_k)\,N_{j,4}(v_k)\,\mathbf{Q}_{n_u-1,j} + \sum_{i=0}^{n_u-1} N_{i,4}(u_k)\,N_{0,4}(v_k)\,\mathbf{Q}_{0,j} + \sum_{j=0}^{n_u-1} N_{i,4}(u_k)\,N_{n_v-1,4}(v_k)\,\mathbf{Q}_{n_v-1,j}. \tag{11}$$

We can represent the right side of equation (10) as the multiplication of two matrices $\mathbf{M}$ and $\mathbf{Q}^{in}$:

$$\mathbf{p}_k - \mathbf{H}(u_k, v_k) = \mathbf{M}\mathbf{Q}^{in}, \tag{12}$$

where $\mathbf{Q}^{in}$ is a $(n_u - 2)(n_v - 2) \times 1$ matrix representation of the control points in equation (10), and $\mathbf{M}$ is defined as a $1 \times (n_u - 2)(n_v - 2)$ row matrix.

$$\mathbf{M} = [N_{1,4}(u_k)\,N_{1,4}(v_k)\cdots N_{n_u-2,4}(u_k)\,N_{n_v-2,4}(v_k)] \tag{13}$$

Then, for $\mathbf{p}_k \in \mathbf{P}^{in}$, we solve equation (12). However, a proper parametric value should be assigned to each point in $\mathbf{p}_k$. In this study, we use the base-surface parametrization method (Ma & Kruth, 1995). We have a surface obtained from the boundary curves that can be used as a base surface for the parametrization method, and the geometric shape is not complex.

$$\mathbf{P}^{in} = \begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \vdots \\ \mathbf{p}_{(n_u-2)(n_v-2)} \end{bmatrix}, \quad \mathbf{H} = \begin{bmatrix} \mathbf{H}(u_1, v_1) \\ \mathbf{H}(u_1, v_2) \\ \vdots \\ \mathbf{H}\left(u_{(n_u-2)(n_v-2)}, v_{(n_u-2)(n_v-2)}\right) \end{bmatrix},$$

$$\mathbf{M} = \begin{bmatrix} N_{1,4}(u_1)\,N_{1,4}(v_1) & \cdots & N_{(n_u-2),4}(u_1)\,N_{(n_v-2),4}(v_1) \\ \vdots & \ddots & \vdots \\ N_{1,4}\left(u_{(n_u-2)(n_v-2)}\right) N_{1,4}\left(v_{(n_u-2)(n_v-2)}\right) & \cdots & N_{n_u-2,4}\left(u_{(n_u-2)(n_v-2)}\right) N_{n_v-2,4}\left(v_{(n_u-2)(n_v-2)}\right) \end{bmatrix} \tag{14}$$

Then, a matrix equation for $\mathbf{P}^{in}$ can be used to compute $\mathbf{Q}^{in}$ for approximating the points on the NURBS surface $\mathbf{S}(u, v)$.

$$\mathbf{P}^{in} - \mathbf{H} = \mathbf{M}\mathbf{Q}^{in} \tag{15}$$

Equation (15) can be solved for $\mathbf{Q}^{in}$ using the singular value decomposition algorithm. The computed $\mathbf{Q}^{in}$ values are used as the new control points for $\mathbf{S}(u, v)$. Next, the error between $\mathbf{P}$ and $\mathbf{S}^{\text{updated}}(u, v)$ is computed to determine if the approximation error is within the user-defined tolerance τ.

The root mean square error (RMSE) between $\mathbf{P}$ and $\mathbf{S}^{\text{updated}}(u, v)$ is computed as an error measure. $\mathbf{P}$ is orthogonally projected onto $\mathbf{S}^{\text{updated}}(u, v)$ to produce $\mathbf{P}^{\text{proj}}$ and the RMSE is computed as follows:

$$R = \sqrt{\frac{1}{N}\sum_{k=0}^{N-1} \left\| \mathbf{p}_k - \mathbf{p}_k^{\text{proj}} \right\|}. \tag{16}$$

If $R < \tau$, then $\mathbf{S}^{\text{updated}}(u, v)$ is used as a reconstructed surface. Otherwise, we estimate a new set of parametric values for $\mathbf{P}^{in}$ using $\mathbf{S}^{\text{updated}}(u, v)$, compute a new $\mathbf{Q}^{in}$, and calculate the RMSE error. This process repeats until $R < \tau$ is satisfied.

3. Experimental Results and Discussion

Two types of experiments were performed to evaluate the performance of the proposed methods. First, the effectiveness of the index values was tested. Two sets of training data were prepared: one set containing the position vectors of each point and the other set containing the position vectors and index values of each point. Both types were used to train the neural network PointNet, and the results were compared. Second, the proposed method was compared with the method PO_I that only uses index values without a

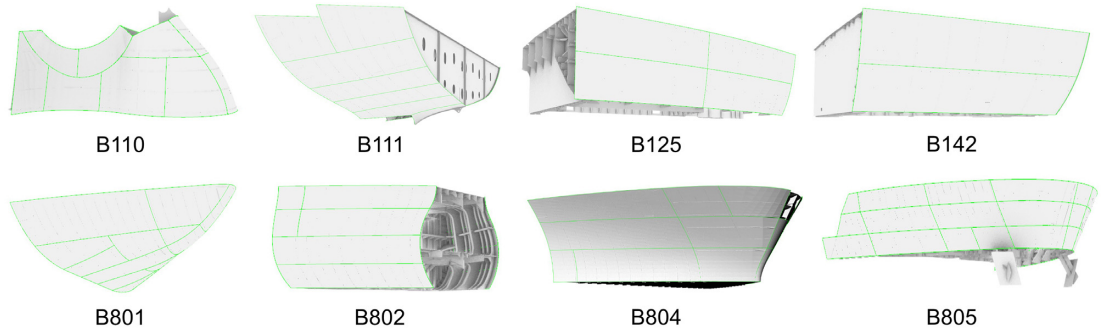

Figure 9: Examples of ship hull CAD models used in the experiments.

Table 2: Recall, precision, and accuracy results of PN_{xyz} and PN_I on the test set.

Method	PN_{xyz}	PN_I
Recall	0.9479	**0.9997**
Precision	0.9410	**1.0**
Accuracy	0.9370	**0.9998**

neural network and the recent boundary detection algorithm, the BPD method (Mineo *et al.*, 2019). PO_I computes the mean of index values for a plate, and classifies the points with the index values smaller than the mean value of them as boundary points.

For the experiments, a dataset consisting of 52 hull plates from 8 actual ship hull models was used, as shown in Fig. 9. Some hull plates are extracted from the same ship hull models used in the training and validation datasets such as B801, B802, or B805 models, but they are obtained from different locations of the models. Each plate model was represented by 72–181 points. These plates were not used for training. For normal and index computations, all points around each point were used for index computation and 30 points in the nearest neighborhood of each point were used for normal computation. For evaluation measures, recall, precision, and accuracy are used to compute the point classification performance evaluation (Powers, 2011). When an input point set is given, if an algorithm correctly classifies a point as a boundary point, it is defined as true positive (TP). If the algorithm classifies a point as an interior point, it is true negative (TN). Similarly, if the algorithm classifies a point as a boundary point but it is an interior point according to the ground truth label, it is false positive (FP). If the algorithm classifies a point as an interior point but it is a boundary point, it is false negative (FN). These four values (TP, TN, FP, and FN) are used to compute recall, precision, and accuracies where the mathematical definitions are defined in equation (17).

$$\text{Recall} = \frac{\text{TP}}{\text{TP}+\text{FN}}, \qquad \text{Precision} = \frac{\text{TP}}{\text{TP}+\text{FP}}, \qquad \text{Accuracy} = \frac{\text{TP}+\text{TN}}{\text{TP}+\text{TN}+\text{FP}+\text{FN}} \tag{17}$$

The test hardware consisted of an Intel Core i7-6700K CPU and 32GB RAM on the 64-bit Windows 10 platform. Visual Studio 2015 (C++) and Python were used for the implementation.

3.1 First test: index value analysis using PointNet

PointNet results without and with index values are denoted by PN_{xyz} and PN_I, respectively. The same parameters and training sets were used in this test. The training set for PN_I contained the additional index values. The training and validation accuracies achieved by PN_{xyz} were 89.16% and 89.55%, which were lower than the train accuracy of 97.35% and validation accuracy of 98.68% achieved by PN_I. Also, Table 2 shows the recall, precision, and accuracy results of PN_{xyz} for the test set that are lower than the results of PN_I. In fact, PN_{xyz} only correctly classified 5 out of 52 test cases, and Fig. 10 shows some examples of misclassifications. In addition, PN_{xyz} did not correctly classify the case in which two plates are connected consecutively. For example, Fig. 11 shows the point classification results for two elliptic hull plates connected along an edge, as marked by the rectangle in the figure. Each plate was used for training and validation. In Fig. 11a, some boundary points were missed by PN_{xyz}, and some interior points were incorrectly identified as boundary points. On the other hand, PN_I correctly classified all points, as shown in Fig. 11b. This implies that PN_I performs better than PN_{xyz}.

The effectiveness of the index values was further tested with the examples used in the study by Mineo *et al.* (2019). The first point set is a curved surface defining one fan blade of an aircraft turbine engine. The blade contains three holes. The second point set is the Stanford bunny model (Turk & Levoy, 1994), a model commonly used in computer graphics experiments. It was cut in half and used as input in the demo program presented by Mineo *et al.* (2019). The number of points in the two examples are 7242 and 20 443, respectively. For the normal and index computations, 30 points in the nearest neighborhood of each point were used. Note that both point sets possess a more complex shape than typical hull plates. The point density is higher than that of the dataset that was used to train the network.

Fig. 12 shows the results generated by PN_{xyz} and PN_I for the two test sets. For the turbine blade surface, PN_{xyz} failed to find various boundary points. It misidentified a large number of interior points as boundary points near the corners and misidentified a large number of boundary points, as shown in Fig. 12a. In Fig. 12c, PN_{xyz} also failed to detect various boundary points. It does not work

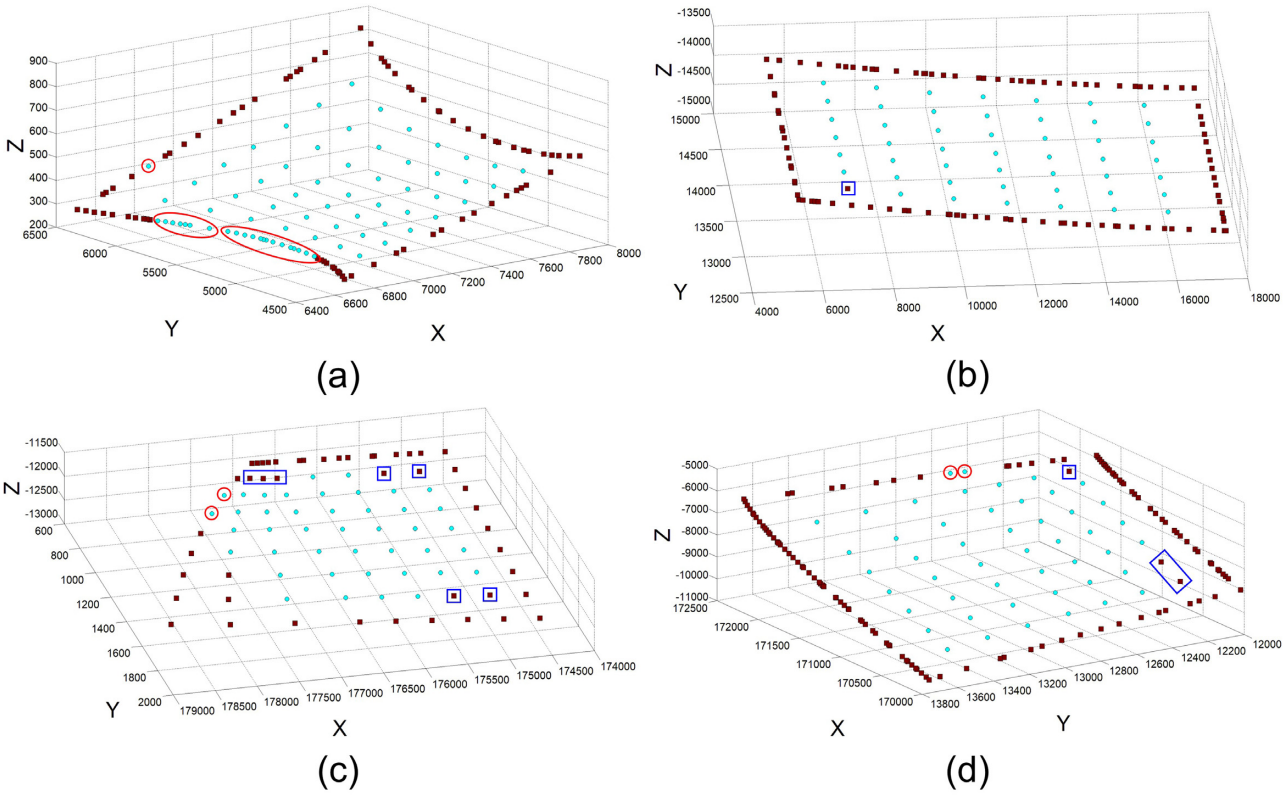

Figure 10: Some failure cases of PN_{xyz} for the test dataset, where square points represent boundary points, and circles and boxes represent regions of misidentified points.

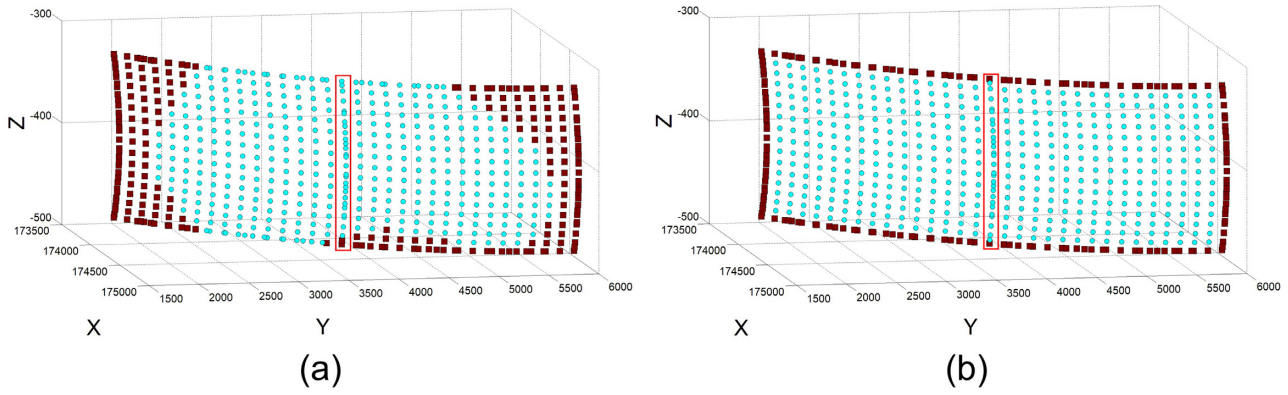

Figure 11: Point classification results of the two combined elliptic hull plates, (a) PN_{xyz} and (b) PN_I. The two plates are connected along the edge shown by the rectangle. The square points represent boundary points or misidentified boundary points.

well in regions where the point density is high, and in convex regions such as the ears and feet. On the other hand, PN_I found more boundary points than PN_{xyz}, as shown in Fig. 12b. In the half-bunny example, PN_I also correctly identified more boundary points. Some internal boundary points are identified as boundary points, which are marked in the box, as shown in Fig. 12d. The region in the box in Fig. 12d is treated as a hole by the network because of the distribution pattern of the points (they seem to form a hole because of the sparse point density), and it seems that some of the points on the boundary of the hole are identified. The proposed method is not trained to detect internal boundary points of a hole. However, it does detect some of them as shown in Fig. 12b because the proposed method determines whether a point belongs to a boundary or not by considering the local distribution pattern of the points in its neighborhood, not taking into account the global point distribution or topological information. As shown in Fig. 12b, the boundary points of the hole in the lower part of the blade are identified as the boundary points because the density of the points is high, and in a local sense the distribution pattern of the points is sufficiently close to a straight line or a convex. Similarly, some of the internal boundary points are detected for the holes in the upper part of the blade, as shown in Fig. 12b. The difference between PN_{xyz} and PN_I is the existence of index information that provides the topology distribution of the point set. These results imply that index values do assist the deep learning network in classifying correct boundary points and in improving overall performance.

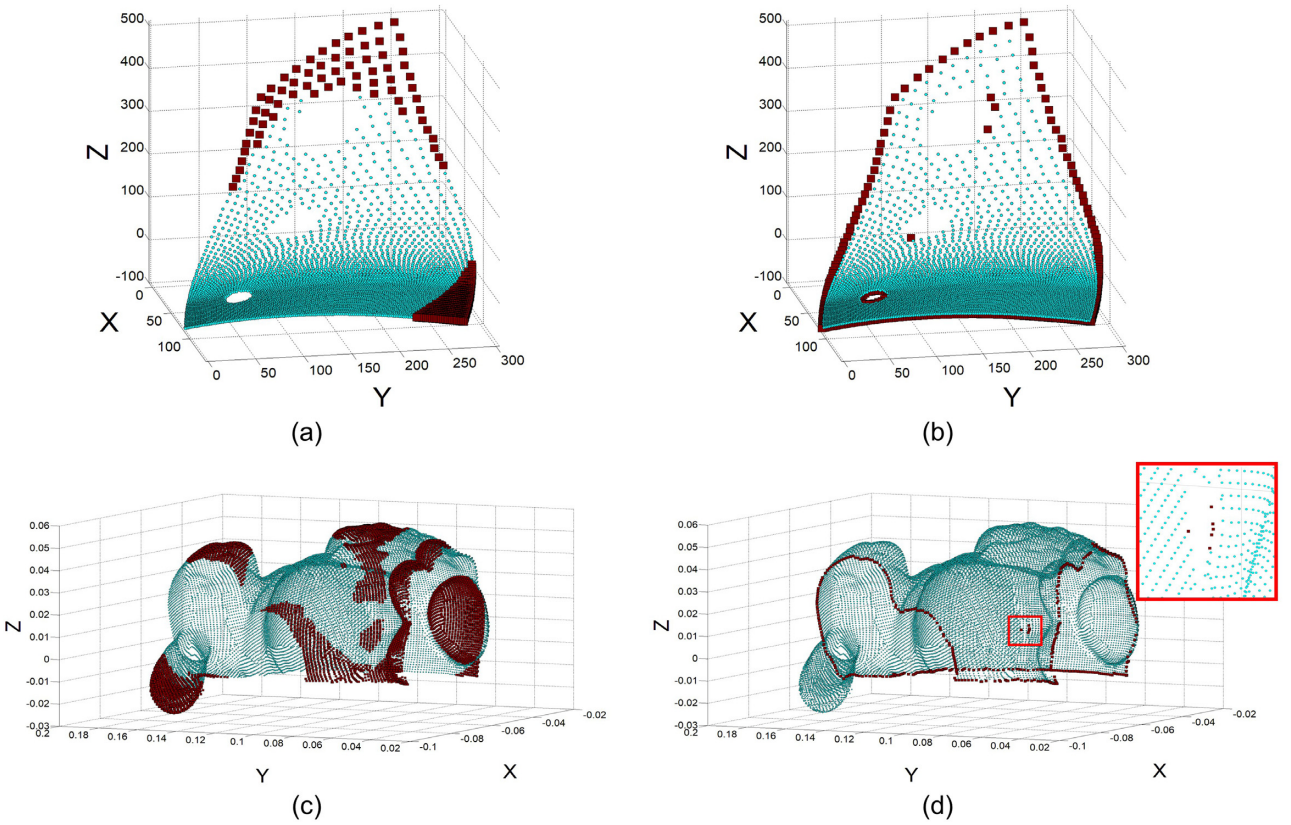

Figure 12: Point classification results of a turbine blade surface and the Stanford half-bunny. (a) and (c) are the results using PN_{xyz}. (b) and (d) are the results using PN_I. The rectangular area in (d) is magnified for better visualization.

Table 3: Recall, precision, and accuracy results of the BPD method, PO_I, and the proposed method on the test examples.

Method	BPD	PO_I	PN_I
Recall	0.9997	**1.0**	0.9997
Precision	0.9583	0.9949	**1.0**
Accuracy	0.9754	0.9971	**0.9998**

3.2 Second test: comparison to PO_I and a state-of-the-art algorithm

In this section, we compare the performance of PN_I and the BPD method with PO_I. PO_I only uses an index value without the network. The test datasets used for comparison are the point data of various curved plates obtained from ship hull models in Fig. 9. The recall, precision, and accuracy results for the plates in the test dataset are summarized in Table 3, where the numbers in bold indicate the best scores for recall, precision, and accuracy. Here, the total number of the points for the test set used in the experiment is 5857. Here, the total numbers of the ground truth boundary points and interior points are 3288 and 2569, respectively. Confusion matrices, which show the number of the prediction points (TP, NP, FP, and FN) for each method, are given in Fig. 16.

The algorithm PO_I found the correct boundary points for 44 out of the 52 curved hull plates. It misidentified interior points as boundary points for eight hull plates, where the misidentified points are located close to the boundary of the input shape as shown in Fig. 13.

The BPD method found the correct boundary points for 25 out of the 52 curved hull plates, achieving the worst prediction result among the tested algorithms. It also sometimes incorrectly identified interior points as boundary points.

The BPD method consists of two steps. In the first step, the boundary points are determined based on the local point cloud resolution β. Then, in the second step, the algorithm determines whether it is an interior point by drawing a path based on the neighborhood of the point for each boundary point. When the input shape is long and thin, the second step of the BPD method does not work properly owing to the low point distribution. As shown in Fig. 14, the BPD method misidentified various interior points as boundary points. In particular, it classified 24 interior points as boundary points for a plate of B805 model in Fig. 14, which cannot be used for a sufficiently accurate surface reconstruction.

The proposed algorithm found the correct boundary points for 51 out of 52 curved hull plates. It only missed one boundary point for B805 No. 1, which is also one of the models that the BPD method failed to handle correctly. In this case, whether the point is

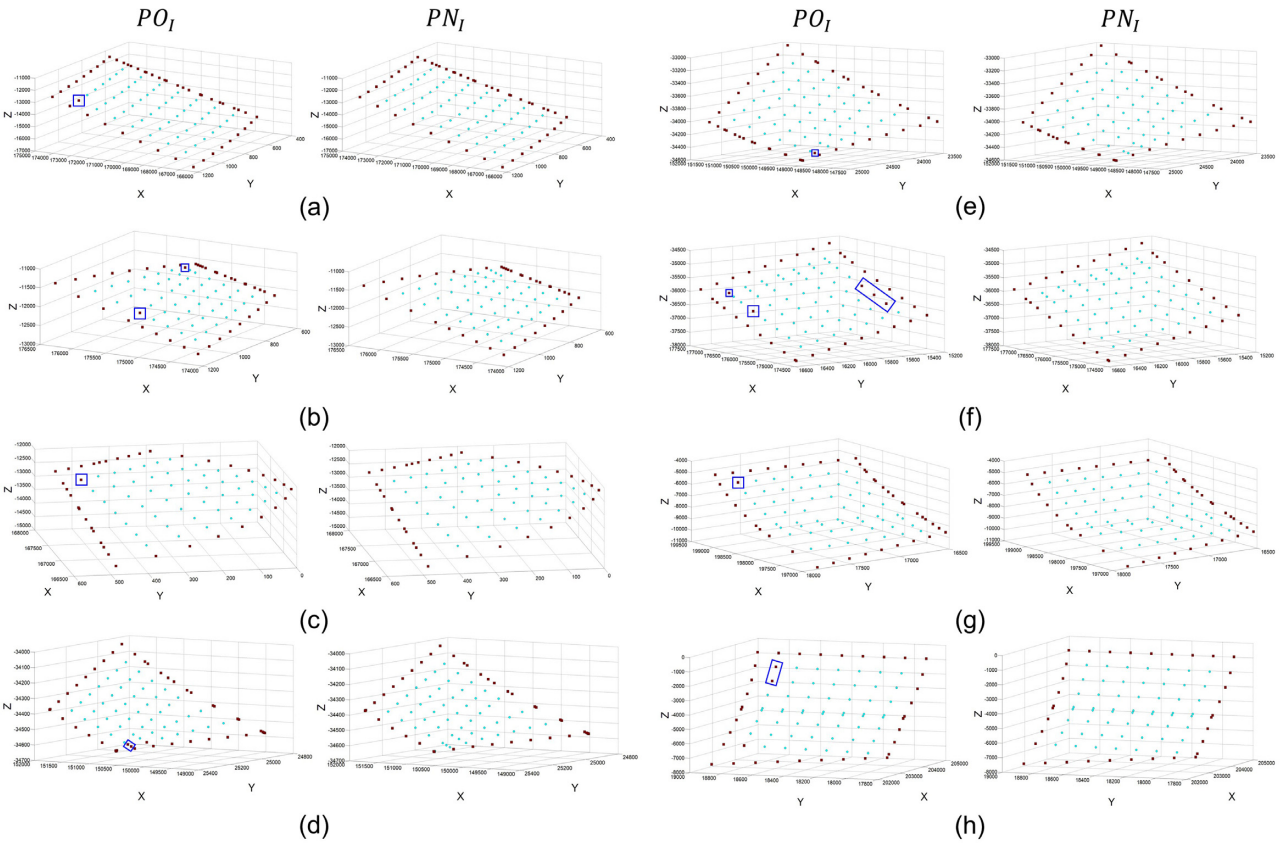

Figure 13: All failure cases of PO_I and classification results of the proposed method PN_I for the same inputs. The square points represent the boundary points and boxes represent regions of misidentified points.

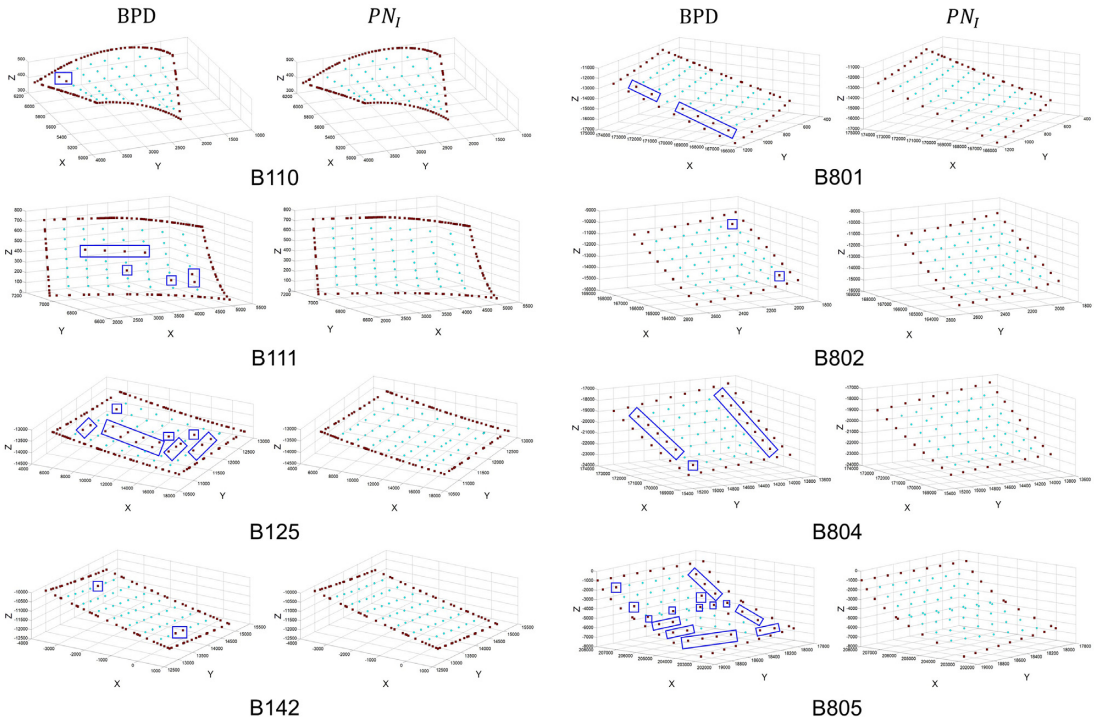

Figure 14: Failure cases of the BPD method and the results of the proposed method PN_I for the same inputs of the BPD method. The square points represent the boundary points. The points in the boxes are misidentified points.

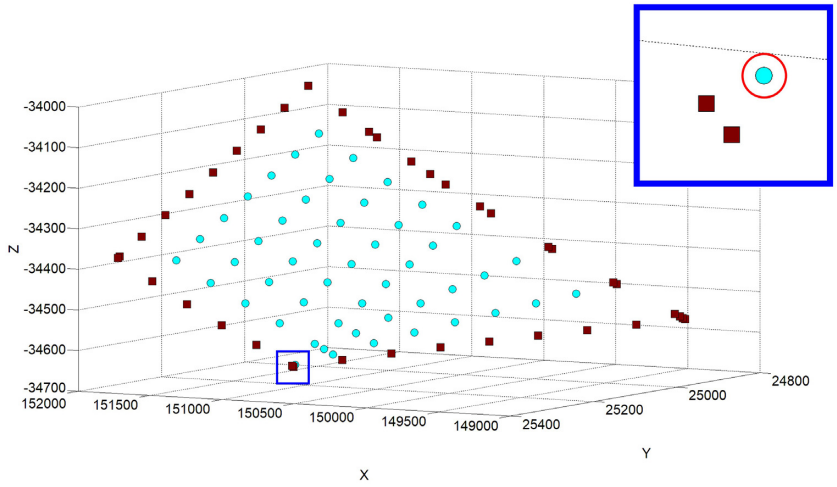

Figure 15: Failure case of the proposed algorithm for B805 No. 1, where the square points represent the boundary points. The zoomed-in area in the box shows boundary and interior point locations. The circle in the box represents the region involving a misidentified boundary point.

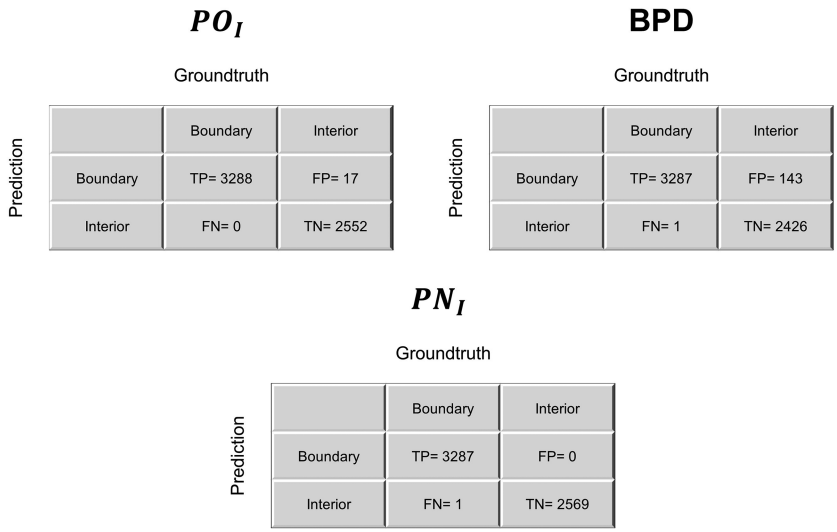

Figure 16: Confusion matrices for PO_I, BPD, and PN_I methods.

a boundary point or not is somewhat ambiguous, because it is located very close to the boundary of the input shape, as shown in Fig. 15.

This experiment demonstrates that PN_I is superior to the PO_I and BPD methods for the detection of the boundary points of curved hull plates. Furthermore, all corner points are correctly identified by the proposed algorithm for all the test cases in the experiment.

3.2.1 *Discussion*

The proposed point classification method works well for curved hull plates. However, it has a few drawbacks. First, if an interior point is too close to any of the boundary points, the trained network PN_I cannot correctly differentiate whether it is an interior point or a boundary point, even when an index value is provided. Second, if a corner point of the input plate is located on a concave line, it may not be able to detect it, as shown in Fig. 2. Third, an object containing holes may not be properly handled by the algorithm, as shown in Fig. 12. Plates with a hole are not considered in this work because the hull plates of a ship usually have no hole inside. However, we believe that such problems might be caused by insufficient examples provided in the training dataset, which can be overcome by using more test cases representing diverse scenarios.

We performed experiments to check the performance of the proposed method when noise is added to the input points. We chose an actual hull plate from B804 model, and added zero-mean Gaussian noise with different variances σ^2 to the input points. We found that the proposed algorithm correctly classified the boundary points, even though variance σ^2 is increased up to 20.0 as shown in Fig. 17. Therefore, our proposed algorithm is robust to some amount of noise.

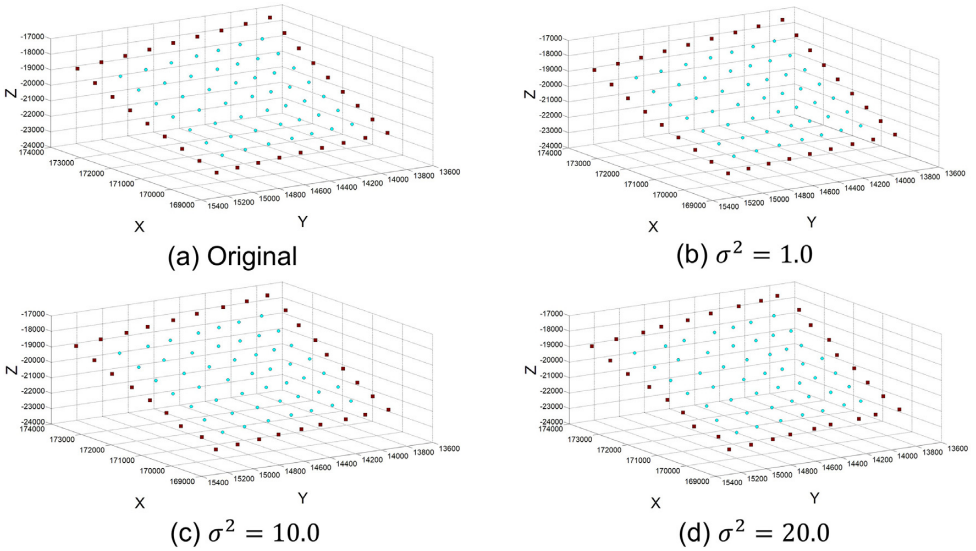

Figure 17: Point classification results of B804 plate with zero-mean Gaussian noise of different σ values for (a) original input, (b) $\sigma = 1.0$, (c) $\sigma = 10.0$, and (d) $\sigma = 20.0$.

Table 4: NURBS surface reconstruction results based on ship hull data (RMSE in mm).

Model	t (s)	RMSE	$Q_n \times Q_n$	Model	t (s)	RMSE	$Q_n \times Q_n$
B110				**B142**			
1	8.27	0.038	9	1	7.81	0.115	7
2	9.34	0.175	9	2	7.82	0.025	9
3	9.11	1.594	7	**B801**			
4	8.59	0.328	8	1	7.45	0.015	9
5	9.01	0.797	6	2	8.21	0.59	7
6	9.18	0.735	7	3	7.49	0.011	9
B111				4	7.3	0.042	9
1	8.87	0.092	9	5	7.68	0.055	9
2	7.55	0.066	9	**B802**			
3	7.56	0.023	9	1	7.22	0.002	9
4	8.2	0.036	9	2	7.07	0.001	9
5	7.75	0.01	9	3	7.21	0.002	9
6	7.58	0.022	9	4	6.91	0.001	9
B125				5	7.57	0.017	9
1	7.73	0.079	9	6	7.35	0.017	9
2	8.36	0.224	6	**B805**			
3	7.92	0.198	7	2	7.75	0.115	7
4	8.42	0.038	9	3	10.93	0.016	9
B804				4	7.21	0.075	9
1	6.98	0.283	8	5	6.94	0.179	8
2	6.06	0.513	7	6	7.1	0.038	8
3	6.74	0.002	9	7	7.17	0.002	9
4	6.94	0.002	9	8	7.28	0.868	8
5	8.11	0.061	8	9	7.12	1.693	8
6	8.48	0.23	8	10	7.18	0.78	8
7	9.64	1.464	8	11	7.18	0.408	8
8	9.52	1.183	8	12	7.33	1.018	6
9	8.09	0.219	9	13	7.02	1.929	7
10	9.29	0.231	9	**AVG**	7.86	0.327	-

3.3 NURBS surface reconstruction results

In this section, we present the NURBS surface reconstruction results for the hull plates contained in the test dataset. We reconstructed NURBS surfaces for 51 hull plates, those in which the boundary points were correctly classified by the network PN_l. To estimate the accuracy of the reconstruction, equation (16) was computed in millimeters (mm). The total computation time t was estimated in seconds, and the final grid size of the control points $Q_n \times Q_n$ was included. Table 4 lists the NURBS reconstruction results using the test data except No. 1 of B805 that is not included because it is a failure case of PN_l. The average computation time of the proposed

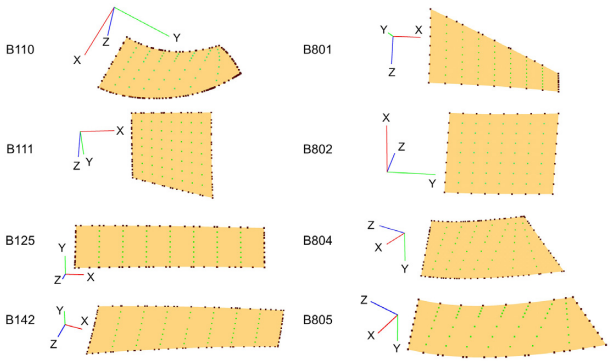

Figure 18: Input point sets in which bold points represent boundary points and the corresponding NURBS reconstruction results for each ship hull model.

framework was 7.86 s, and the average RMSE was 0.327 mm. Fig. 18 shows examples of reconstructed NURBS surfaces based on ship hull data, including the input boundary points.

4. Conclusions

In this paper, we proposed a new point classification method for unorganized 3D points using a neural network, which classifies them into boundary, corner, and interior points. Thereafter, we presented a procedure for reconstructing a NURBS surface using the classified points. With respect to hull plates, the input point set is generally sparse and unorganized. Therefore, classification is a challenging task that existing methods cannot properly deal with. To overcome this issue, we incorporated an index value for each point (a topological property providing geometric position) and trained a network with this information to improve the classification performance.

We based this study on PointNet, an existing deep neural network model used for point segmentation. However, we introduced topological information that has not been previously considered in the segmentation problem, which was demonstrated to improve classification performance in our target application.

However, the network does not provide a complete solution in that it cannot properly handle certain special cases, although we believe these issues can be overcome by introducing more diverse training examples. The proposed algorithm can be easily extended to a wide spectrum of applications where an unorganized point set is given as input and classification of the points into boundary and interior points is a main problem. For each application, additional information specific to each problem should be introduced. Moreover, the current network structure can be analysed and redesigned for optimal performance of different target applications, which is a topic recommended for future work.

Acknowledgement

This study was supported by the Technology Innovation Program (or Industrial Strategic Technology Development Program) (20000208, Development of master data system of lead time for precision enhancement of shipbuilding production management) funded by the Ministry of Trade, Industry & Energy (MOTIE, Korea).

Conflict of interest statement

None declared.

References

Asaeedi, S., Didehvar, F., & Mohades, A. (2017). α-Concave hull, a generalization of convex hull. *Theoretical Computer Science*, *702*, 48–59.

Barber, C. B., Dobkin, D. P., & Huhdanpaa, H. (1996). The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software (TOMS)*, *22*, 469–483.

Boult, T., & Sikorski, K. (1989). An optimal complexity algorithm for computing the topological degree in two dimensions. *SIAM Journal on Scientific and Statistical Computing*, *10*(4), 686–698.

Duckham, M., Kulik, L., Worboys, M., & Galton, A. (2008). Efficient generation of simple polygons for characterizing the shape of a set of points in the plane. *Pattern Recognition*, *41*(10), 3224–3236.

Edelsbrunner, H., Kirkpatrick, D., & Seidel, R. (1983). On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, *29*(4), 551–559.

Farin, G. (2001). *Curves and surfaces for CAGD: A practical guide*. (5th ed.). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Graham, R. L. (1972). An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, *1*(4), 132–133.

Hoschek, J., & Lasser, D. (1993). *Fundamentals of computer aided geometric design*. Wellesley, MA, USA: AK Peters.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An Introduction to statistical learning*. New York, NY: Springer.

Jarvis, R. (1973). On the identification of the convex hull of a finite set of points in the plane. *Information Processing Letters*, *2*(1), 18–21.

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic gradient descent. In *ICLR: International Conference on Learning Representations*.
Kirkpatrick, S. (1984). Optimization by simulated annealing: Quantitative studies. *Journal of Statistical Physics*, *34*(5–6), 975–986.
Kirkpatrick, S., Gelatt, C., & Vecchi, M. (1983). Optimization by simulated annealing. *Science (New York, N.Y.)*, *220*(4598), 671–680.
Ko, K. H., Sakkalis, T., & Patrikalakis, N. M. (2008). A reliable algorithm for computing the topological degree of a mapping in R^2. *Applied Mathematics and Computation*, *196*(2), 666–678.
Lee, K. W., & Bo, P. (2016). Feature curve extraction from point clouds via developable strip intersection. *Journal of Computational Design and Engineering*, *3*(2), 102–111.
Maturana, D., & Scherer, S. (2015). Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*(pp. 922–928).
Ma, W., & Kruth, J. P. (1995). Parameterization of randomly measured points for least squares fitting of B-spline curves and surfaces. *Computer-Aided Design*, *27*, 663–675.
Mineo, C., Pierce, S. G., & Summan, R. (2019). Novel algorithms for 3D surface point cloud boundary detection and edge reconstruction. *Journal of Computational Design and Engineering*, *6*(1), 81–91.
Moreira, A., & Santos, M. (2007). Concave hull: A k-nearest neighbours approach for the computation of the region occupied by a set of points, In *GRAPP 2007, Proceedings of the 2nd International Conference on Computer Graphics Theory and Applications*(pp. 61–68).
Peter, M. (2018). A Note Regarding Hopf's Umlaufsatz, *The American Mathematical Monthly*, *125*(6), 541–544.
Powers, D. M. (2011). Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation. *Journal of Machine Learning Technologies*, *2*(1), 37–63.
Qi, C. R., Su, H., Mo, K., & Guibas, L. J. (2017a). Pointnet: Deep learning on point sets for 3D classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*(pp. 652–660).
Qi, C. R., Su, H., Mo, K., & Guibas, L. J. (2017b). Pointnet: Deep learning on point sets for 3D classification and segmentation supplementary material.
Qi, C. R., Su, H., Nißener, M., Dai, A., Yan, M., & Guibas, L. J. (2016). Volumetric and multi-view CNNs for object classification on 3D data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*(pp. 5648–5656).
Qi, C. R., Yi, L., Su, H., & Guibas, L. J. (2017c). Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*(pp. 5099–5108).
Sakkalis, T. (1990). The Euclidean algorithm and the degree of the Gauss map. *SIAM Journal on Computing*, *19*(3), 538–543.
Schrijver, A. (2005). On the history of combinatorial optimization (till 1960). *Handbooks in Operations Research and Management Science*, *12*, 1–68.
Straßer, W., Klein, R., & Rau, R. (2012). *Geometric modeling: Theory and practice: The state of the art*. Berlin, Germany: Springer Science & Business Media.
Su, H., Maji, S., Kalogerakis, E., & Learned-Miller, E. G. (2015). Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE International Conference on Computer Vision*(pp. 945–953).
Turk, G., & Levoy, M. (1994). The Stanford bunny, the Stanford 3D scanning repository.