Feature line detection of noisy triangulated CSG-based objects using deep learning

Martin Denk[1], Kristin Paetzold[2], Klemens Rother[1]

[1] Institute for Material and Building Research;
Munich University of Applied Sciences (MUAS), Germany

[2] Institute of Technical Product Development (ITPE),
University of the German Federal Armed Forces Munich, Germany

Abstract

Feature lines such as sharp edges are the main characteristic lines of a surface. These lines are suitable as a basis for surface reconstruction and reverse engineering [1]. A supervised deep learning approach based on graph convolutional networks on estimating local feature lines will be introduced in the following. We test this deep learning architecture on two provided data sets of which one covers sharp feature lines and the other arbitrary feature lines based on unnoisy meshed constructive solid geometry (CSG). Furthermore, we use a data balancing strategy by classifying different feature line types. We then compare the selected architecture with classical machine learning models. Finally, we show the detection of these lines on noisy and deformed meshes.

Keywords: Feature line, geometric deep learning, reverse engineering

1 Introduction

Various applications such as animation design, topology optimization or 3D laser scanning use discrete triangulated meshes to represent the surface of a 3D geometry. On the other hand, engineering applications used for optimization or design modification require a parametric representation of the geometry [2] [3], such as a constructive solid geometry (CSG). Computer aided design (CAD) tools provide a CSG format, by combining or subtracting multiple simple objects such as spheres, cylinders, or cubes. Still as state of the art, engineers have to convert these triangular meshes manually into parametric CSG models, which is extremely time consuming and thus inefficient. Moreover, manual reconstruction itself can lead to inaccurate approximation of the surfaces by human preparation and interpretation, especially for noisy and deformed triangular meshes. As a solution, the robust detection of important lines called feature line will allow an automatic reconstruction of such discrete surfaces [4]. Then a proper reconstruction of the mesh allows the CAD geometry to be updated by using the surface of a deformed finite element mesh, that has been analyzed [5] [6] [7]. Furthermore, the parametric model or the feature lines of the meshed model itself support a subsequent for shape optimization or structural analysis. In addition, it offers modern geometric modeling techniques based for the engineering purpose, such as sculpting in animation design.

2 State of the Art

A proper reverse engineering deals with a parametrically meaningful surface reconstruction to enable further modifications. The following figure roughly illustrates the reverse engineering process derived from the methods in [6] [2] [7] [5] [3] for mesh data and point clouds. By mesh segmentation and line detection method, reasonable surface segments [8] [2] and line segments [2] [9] [10] can be found. With these geometric features, shape parametrization can interpret surface types like primitive surfaces [2] [3] [11] or B-Splines surfaces [7] [6]. These parametric surfaces leads to a boundary-represented or CSG based model. Our focus in this thesis is on the detection of feature lines such as sharp edges for meshed geometry on CSG models.

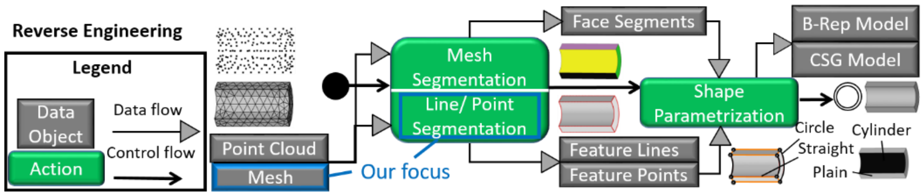


Figure 1: Reverse engineering and our contribution of line detection

Several detection methods summarized in [4] identify feature lines using local surface curvature and normal estimation. For discrete surfaces, there is only an approximation of the local curvature and vertex normal vectors. Therefore, there are several different approaches calculating the local curvature, such as averaging through the neighbourhood of the face or interpolating a local polynomial patches [4]. In particular [11] uses a mesh curvature histogram to determine sharp and smooth edges. We investigate a deep learning approach, since the learning process receives a detection rule directly from the data set and not by approximating the local geometrical properties. Changing the scope, data set, or feature line type is manageable by changing or adding training data. There is no need to redesign the algorithm. In several recent publications, machine learning was used for feature line detection [2] [9] [10] [1], which can divide the 3D objects into meaningful surface patches. Since feature lines can be determined locally [4], a local detection by selecting a local patch similar to [10] [9] [1] is basis for our approach. In particular, [10] uses a feature line detection of sharp edges based on point clouds using a deep learning model trained on meshed data. The authors use a k-nearest neighbour graph to locally determine a sharp edge. In [2] multiple energy terms that capture face and edges information are combined. In particular, for edges, they detect feature lines by using a Support Vector Machine to find edges and the alignment of these edges. The most similar approach to our work is the local feature line detection in [9] and the sharpness field extraction in [1]. To determine feature lines, the authors of [9] presents a detection by a Support Vector Machine. As input for the learning algorithm, [9] selects the local curvature, the dihedral angle, the global curvature and a shape diameter on the neighbourhood of an edge. Unlike [9], we use a deep learning approach that processes on angular sizes around the edges without interpreting a form factor or a global curve. Furthermore [9] applies their feature line detection on a data set consisting organic and mechanical meshes. We focus the detection on mechanical CSG data. [1] uses a deep learning architecture containing CNN (Convolutional neuronal networks) layers to determine sharp features on point clouds. As the neighbourhood [1] uses a planar radial grid unfolded at the surface. They select the angle and normal values along the radial grid lines for the input vector. For our process, we use a scale, rotation and translation invariant value similar to [1] inform of scalar products. But unlike [1] we restrict to use a fixed neighbourhood of an edge. So that we can estimate small local features independently of a selected radius or number of points.

Due to the rapid development in deep learning, models of [10] [1] [8] can perform a classification task for different kind of 3D geometries [12] [13]. Especially the use of networks CNN allows a high degree of parallelisation. CNN layers typically are used for Euclidean data structure such as 2D pixels or 3D voxels [12]. While deep learning methods on Euclidean data such as voxels or pixels are widely established, the current research areas summarized in [13] or [12] aims to use CNN for non-Euclidean data such as graphs, polygonised surfaces and point clouds. Our selected convolutional layers are most similar to [14]. [14] used a graph based approach for processing non-Euclidian data. In [14] the adjacency matrix for normalizing the learnable weight values for each specific graph is selected. We use only one learnable weight parameter and bias weight for each kernel of the graph convolutional layer. Our input size and structure remains the same, so that we do not have to normalize the weights.

To summarize our main contribution, in chapter 3 we train a deep learning model for local feature line detection on triangulated surfaces of primitive models such as cylinder, cube or sphere. We first provide a line and surface categorization for a CSG data set balancing the training data as described in chapter 3.1. In chapter 3.2, we feed an input vector with the usage of the local gradient and angle information into a model containing convolutional graph based simplified layers of [14]. In chapter 4, we validate the approach by dividing the data set into train and validation. In addition, we use a CSG object visualizing the influence on deformation and noise.

3 Method

For the classification task, a supervised CSG data set consisting of various triangulated primitives and labelled edges of the triangles is generated. A mesh-generating algorithm creates discrete surfaces with different resolutions of these CSG objects and stores correspondences between the edges of the triangle and the contours of the CSG object. In addition, we balance the data that the amount of data is similar for each selected line type and surface.

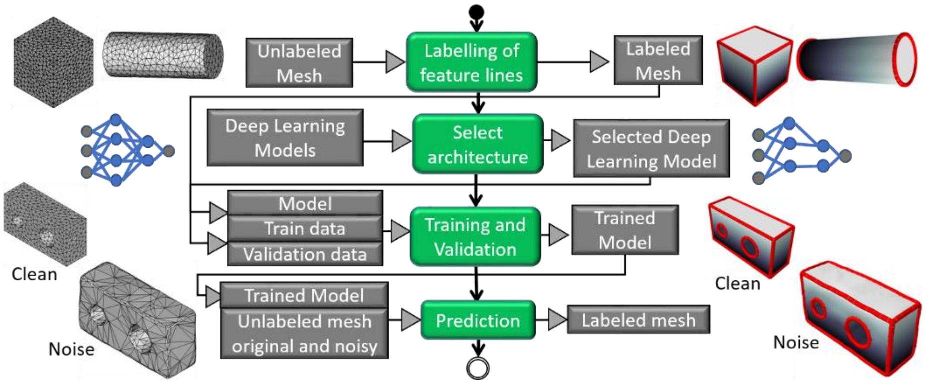


Figure 2: Activity diagram of labelling, training and prediction.

The activity diagram of figure 2 covers the different steps (Action) and data transformations (Data objects) to select and train a deep learning model using two objects from the CSG data set as examples. At the end, the diagram shows the feature line detection on unmarked clean and noisy mesh data applying the deep learning model.

3.1 Generation and balancing of trainings data

For the data set, we categorize the edges of the triangle mesh by using their parametric definition of the underneath surface of feature line. As categorization criteria we use the primitive surface type S and line type l with

$$S = \{'Cone','Cylinder','Plain','Torus','Sphere'\}; \quad L = \{'Circle','Line'\}.$$

We balance the data set by using a categorization of the supervised surface type and feature line type of the triangulated mesh, so that different edge type occur in the same amount for training. The following figure shows several primitive objects and the results of surface and edge classification.

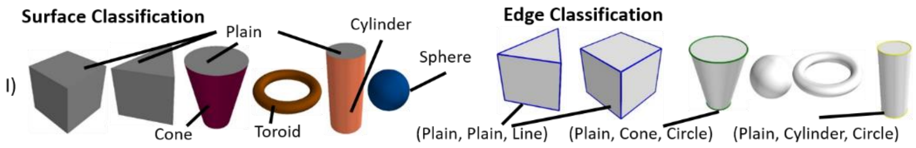


Figure 3: Classification of primitives

A valid feature line has two different neighbour surfaces or the same surface on both sides. We classify each feature line through the neighbour surface and line type like in figure 2 $('Plain','Cone','Circle')$, so that we get the numerical categories for feature lines

$$l = \{x \mid x \leq |L|\binom{|S|}{2} \; ; \; x \in \mathbb{N}^+\}. \tag{1}$$

Additionally we classify the non-feature lines according to their corresponding surface type such as $('Cylinder')$. So that we add an additional numerical class for each surface type with

$$s = \{x \mid |L|\binom{|S|}{2} < x \leq |L|\binom{|S|}{2} + |S| \; ; \; x \in \mathbb{N}^+\}. \tag{2}$$

We add some additional objects into the training set of figure 2 as shown in the following figure. We first use a data set consisting of sharp edges of the primitives in figure 3 I) and in figure 4 II) for training. Then we extend the data set with III) consisting of smoothed rounded and bevelled edges. The object in IV) serves as a test object.

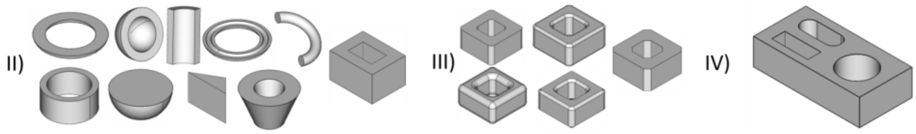


Figure 4: II) Data for training and validation III) extended data with detailed edges IV) test model

The amount of different edge types of the selected objects differs. Therefore, we use these categories (1) and (2) for balancing the data set, so that same amount of each surface and line type is covered for training. This balancing strategy can be seen in the following figure.

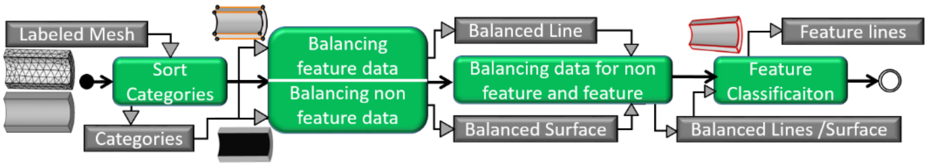


Figure 5: Balancing of the data and binary feature classification

We separate the data into surface and line categories. Some categories are not presented in the selected objects. These categories are ignored. Then we balance the data. Additionally to the balanced data, we use an original, deformed and two noisy triangulated surfaces for visual testing the feature line detection. The following figure represents the selected test models a) and different mesh resolutions b). Our data consists of several different mesh resolutions, but we limit the minimum resolution so that the primitive shape will be preserved. Therefore, the low resolution model in b) is invalid for our dataset.

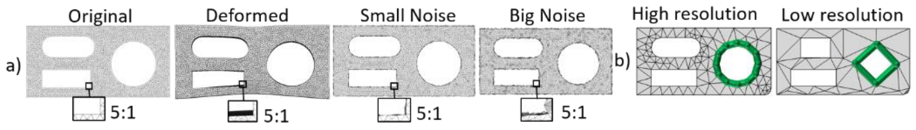


Figure 6: 2D view of the model of IV) a) Original, deformed and noisy meshes for the visual test b) Shape preserving and not preserving resolution

3.2 Deep learning architecture and feature vector

For the classification on the geometric mesh data, an input vector based on scalar products is used for our learning model, which is scale, rotational and translational invariant. Additionally only local information are used, so that the size of the input vector is the same for each edge classification. For training, angles of the triangles and their adjoining surroundings are used as input. The selected scalar products and neighbourhood for the input vector are shown in the following figure.

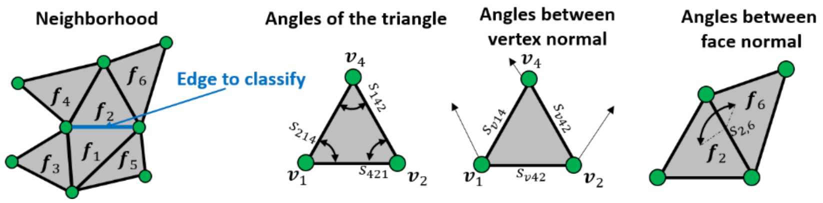


Figure 7: Structure of the input data for training.

For the normal vector $\boldsymbol{n}_v$ of the vertices v, we use a uniformly weighted approach of [15] over the mean value of the surrounding face normal $\boldsymbol{n}_i$ with

$$\boldsymbol{n}_v = \frac{1}{n}\sum_{i}^{n} \boldsymbol{n}_i. \tag{3}$$

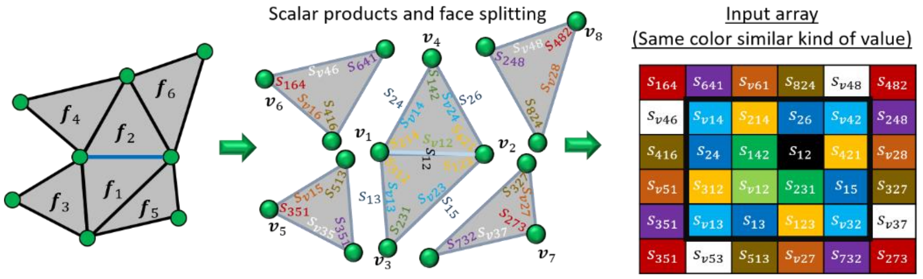
We calculate the normalized scalar product s_{ikj} between the vertices $\boldsymbol{v}_i$, $\boldsymbol{v}_j$ and $\boldsymbol{v}_k$ the edges with

$$s_{ikj} = \frac{1}{\|\boldsymbol{v}_i - \boldsymbol{v}_k\|_2 \; \|\boldsymbol{v}_j - \boldsymbol{v}_k\|_2} \langle \boldsymbol{v}_i - \boldsymbol{v}_k, \boldsymbol{v}_j - \boldsymbol{v}_k \rangle \tag{4}$$

The scalar product of the edges $s_{v,ij}$ and the scalar product between faces s_{mn} with the point i and j and the faces m and n will be calculated with

$$s_{v,ij} = \frac{1}{\|\boldsymbol{n}_i\|_2 \; \|\boldsymbol{n}_j\|_2} \langle \boldsymbol{n}_i , \boldsymbol{n}_j \rangle; \quad s_{mn} = \frac{1}{\|\boldsymbol{f}_m\|_2 \; \|\boldsymbol{f}_n\|_2} \langle \boldsymbol{f}_m , \boldsymbol{f}_n \rangle. \tag{5}$$

According to the non-Euclidian data structure, we use a neighbourhood with a fixed size, which is available for each edge in the same manner. In the following figure, the composition of input vector is visualized.



Scalar products and face splitting
Input array
(Same color similar kind of value)
f_1 f_2 f_3 f_4 f_5 f_6
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8

s_{164}	s_{641}	s_{v61}	s_{824}	s_{v48}	s_{482}
s_{v46}	s_{v14}	s_{214}	s_{26}	s_{v42}	s_{248}
s_{416}	s_{24}	s_{142}	s_{12}	s_{421}	s_{v28}
s_{v51}	s_{312}	s_{v12}	s_{231}	s_{15}	s_{327}
s_{351}	s_{v13}	s_{13}	s_{123}	s_{v32}	s_{v37}
s_{351}	s_{v53}	s_{513}	s_{v27}	s_{732}	s_{273}

Figure 8: Face correspondence, scalar products and input vector

We build a matrix consisting of the different scalar products. Values with similar meaning are marked with the same color. For classification, we use the input vector at each edge separately. We therefore calculate the input matrix for each edge and feed it into the neuronal network, which contains convolution layers and fully interconnected layers. At the end, we get the probability for the feature and non-feature line by a Softmax function. To increase generality for noisy networks, we use Gaussian noise with a standard deviation of 0.1 for the input layer during training and multiple dropouts.

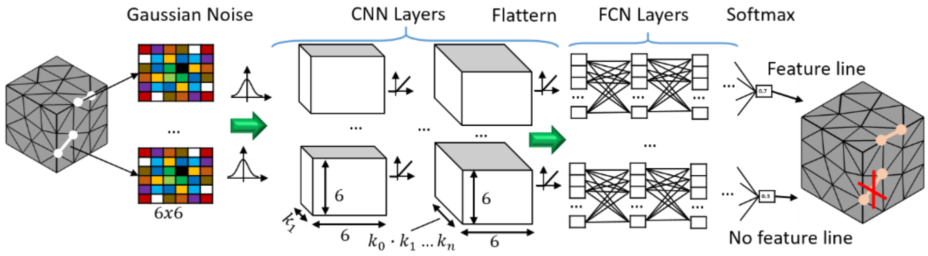


Figure 9: Architecture “FL-Net” for the feature line classification.

The use of convolution layers can detect patterns and interactions between the input matrices. Due to the non-Euclidian data we only use a kernel size of $1x1$ for the convolutional layer with k_i separated kernels for each layer. Therefore, a correlation between the neighbour entries of the input matrix is not covered. In addition, the value of the input matrix itself has dissimilarities. The angles in a triangle have a different meaning than the normal values of the face. Due to the $1x1$ kernel size the entire location of the values is independent.

Therefore, we use the simple propagation rule

$$\boldsymbol{h}^{(l+1)} = \sigma(w_1\boldsymbol{h}^l + w_0) \tag{6}$$

with $\boldsymbol{h}^{(l+1)}$ the output of the current layer l an activation function σ and the weight values w_1 and w_0 for each kernel k. As our activation function, we use a rectifier linear unit (ReLU) $\sigma = max(x, 0)$. After several convolutional layers with ReLU we flatten the feature maps and provide the flattened array into a fully connected layers. For binary classification, we use a Softmax function.

4 Results and Discussion

By balancing the data sets, we reduce the amount of training data. The following table shows the reduction of total edges by balancing. The data set of I), II) is smaller than the data set of I), II), III). By a new line type with a low occurrence in III) the balancing reduces the amount of data in I), II), III), so that the balanced data set is smaller than in the balanced data set in I), II).

Table 1: Data balancing

Data set type	Data I, II	Balance I, II	Data I, II, III	Balance I, II, III
Line data Surface data	82472 5389390	25056 2743135	192744 7978422	21120 2863875
Balanced Surface/Line data	**25056/25056**		**21120/21120**	

The balancing strategies reduce the training data so that a huge amount is necessary. We use the balanced data and divide it into validation and training data. For the train and validation data, we split the surface and line data into 80% training and 20% validation. We now select several machine learning a) Linear SVM (penalty =1), b) Decision Tree (depth=5) and c) Multi Layer Perceptron (Layers = 100, $\sigma = ReLU$) model and compare them with our architecture d) FL-Net. By providing the balanced data of (I, II) all selected methods obtain an accuracy of 100%. By providing the balanced extended data set with smooth edges (I, II, III) the method a), b), c) and d) obtains an accuracy of 84,4%, 90,1%, 87,0% and 92,8%. Our architecture performs best compared to a), b), c) but the accuracy drops to 92,8% compared to the model trained with sharp edge data set. Therefore, the chosen data structure and learning model is not able to detect smooth edges of the data set III) per 100%.

We test the selected architecture and training scenario on the test model IV). Figure 10 shows the result of using sharp edges for training data with an accuracy of 100%.The results of the model trained on sharp edges show a meaningful feature line detection for the original, deformed and (small) noisy mesh. If the noise of the mesh increases, accuracy drops.

I, II)

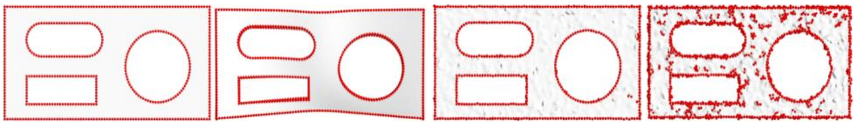


Figure 10: Using the data set with sharp feature line

The results of the model trained with additionally smooth edges show a lower accuracy in figure 11 than in figure 10. There are quite more miss classifications compared to the results in figure 10 in each test subject. The inclusion of smooth edges thus reduces the ability to detect sharp edges.

I, II, III)

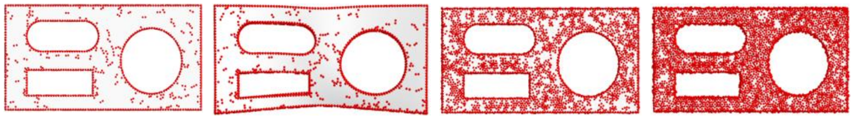


Figure 11: Using the data set with arbitrary feature line

To detect other feature lines types (not sharp), we need to expand the scope of the input feature vector. The neighbourhood based on six faces is not enough to detect smooth edges accurate enough. In addition, we have to use noisy data for training to increase the robustness of the model.

5 Conclusion and Outlook

In our research feature line detection using a data driven approach is covered. First, we create a categorization of the feature lines according to their surface and line correspondence. Secondly, we use these categories for a meaningful balancing of the data set. A feature vector based on scalar products based on the local geometry of the neighbourhood is used. This vector is fed into a model containing convolutional and fully connected layers to perform a binary classification into "Feature" and "No Feature". The proposed method establishes the detection of sharp edges of deformed meshes with low noise. However, this method lacks the ability to detect smooth feature lines. This leads to a further improvement by extending the neighbourhood or by using additionally curvature values. We can increase robustness by additionally using noisy mesh data for supervised learning. Therefore, the noise is contained not only in the learning model, it is also covered by data.

Acknowledgement

This work is part of the project ANGORA which is supported by "Federal Ministry for Economic Affairs and Energy" with the funding indicator ZF4428401BZ7 in the context of the research network IraSME and ZIM. Responsibility for the content of this publication lies with the authors. The authors would like to thank all those involved in the project.

Literatur

[1] P. Raina, S. Mudur and T. Popa, "Sharpness fields in point clouds using deep learning," *Computers & Graphics,* vol. 78, pp. 37-53, 2019.

[2] V. Vidal, C. Wolf and F. Dupont, "Mechanical Mesh Segmentation and Global 3D Shape Extraction," 2014.

[3] R. Bénière, G. Subsol, G. Gesquière, F. L. Breton and W. Puech, "A comprehensive process of reverse engineering from 3D meshes to CAD models," *Computer-Aided Design,* vol. 45, pp. 1382-1393, 2013.

[4] W. Quan, W. Meng and X. Zhang, "The Extraction of Feature Lines on 3D Models: A Survey," in *2014 International Conference on Virtual Reality and Visualization*, 2014.

[5] A. B. Makhlouf, B. Louhichi, M. A. Mahjoub and G. Subsol, "Approach for CAD model Reconstruction from a deformed mesh," in *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*, 2017.

[6] B. Louhichi, G. N. Abenhaim and A. S. Tahan, "CAD/CAE integration: updating the CAD model after a FEM analysis," *The International Journal of Advanced Manufacturing Technology,* vol. 76, pp. 391-400, 01 1 2015.

[7] A. Ben Makhlouf, B. Louhichi, M. Mahjoub and D. Deneux, "Reconstruction of a CAD model from the deformed mesh using B-

spline surfaces," *International Journal of Computer Integrated Manufacturing,* pp. 1-13, 4 2019.

[8] Y. Feng, Y. Feng, H. You, X. Zhao and Y. Gao, "MeshNet: Mesh Neural Network for 3D Shape Representation," *AAAI 2019,* 2018.

[9] H. Benhabiles, G. Lavoué, J.-P. Vandeborre and M. Daoudi, "Learning Boundary Edges for 3D-Mesh Segmentation," *Comput. Graph. Forum,* vol. 30, pp. 2170-2182, 2011.

[10] L. Yu, X. Li, C.-W. Fu, D. Cohen-Or and P.-A. Heng, "EC-Net: an Edge-aware Point set Consolidation Network," *CoRR,* vol. abs/1807.06010, 2018.

[11] S. Gauthier, W. Puech, R. Bénière and G. Subsol, "Analysis of digitized 3D mesh curvature histograms for reverse engineering," *Computers in Industry,* Vols. 92--93, pp. 67-83, 11 2017.

[12] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam and P. Vandergheynst, "Geometric deep learning: going beyond Euclidean data," *CoRR,* vol. abs/1611.08097, 2016.

[13] E. Ahmed, A. Saint, A. E. R. Shabayek, K. Cherenkova, R. Das, G. Gusev, D. Aouada and B. E. Ottersten, "Deep Learning Advances on Different 3D Data Representations: A Survey," *CoRR,* vol. abs/1808.01462, 2018.

[14] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," *CoRR,* vol. abs/1609.02907, 2016.

[15] S. Jin, R. R. Lewis and D. West, "A comparison of algorithms for vertex normal computation," *The Visual Computer,* vol. 21, pp. 71-82, 01 2 2005.