An end-to-end geometric deficiencies elimination algorithm for 3D meshes

Ma, Bingtao ; Liu, Hongsen ; Nan, L.; Tang, Xu; Fan, Huijie ; Cong, Yang

DOI
10.1109/YAC51587.2020.9337658

Publication date
2021

Document Version
Final published version

Published in
Proceedings of the 35th Youth Academic Annual Conference of Chinese Association of Automation

An End-to-End Geometric Deficiencies Elimination Algorithm for 3D Meshes

Bingtao Ma[1,2,3], Hongsen Liu[1,2,3], Liangliang Nan[4], Xu Tang[1,2], Huijie Fan[1,2] and Yang Cong[1,2,*]
[1]*State Key Laboratory of Robotics, Shenyang Institute of Automation, Chinese Academy of Sciences, Shenyang, 110016, China.*
[2]*Institutes for Robotics and Intelligent Manufacturing, Chinese Academy of Sciences, Shenyang, 110016, China.*
[3]*University of Chinese Academy of Sciences,100049, China.*
[4]*Delft University of Technology, Delft, 2628BL, Netherlands.*
mabingtao93@gmail.com, congyang81@gmail.com

2020 35th Youth Academic Annual Conference of Chinese Association of Automation (YAC) | 978-1-7281-7684-0/20/$31.00 ©2020 IEEE | DOI: 10.1109/YAC51587.2020.9337658

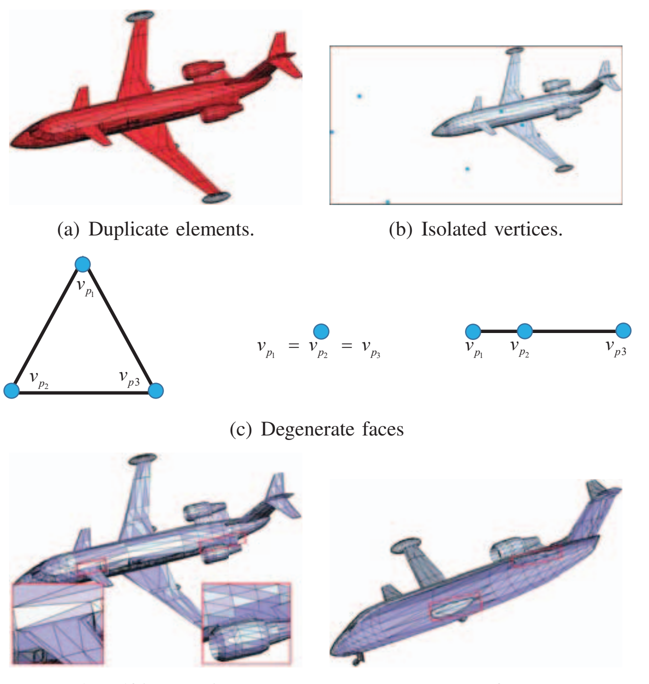
***Abstract*—The 3D mesh is an important representation of geometric data. It is widely used in computer graphics and has attracted more attention in computer vision community recently. However, in the generation of mesh data, geometric deficiencies (*e.g.*, duplicate elements, degenerate faces, isolated vertices, self-intersection, and inner faces) are unavoidable. Geometric deficiencies may violate the topology structure of an object and affect the use of 3D meshes. In this paper, we propose an end-to-end algorithm to eliminate geometric deficiencies effectively and efficiently for 3D meshes in a specific and reasonable order. Specifically, *duplicate elements* can be first eliminated by assessing appear times of vertices or faces. Then, *degenerate faces* can be removed according to the outer product of two edges. Next, since *isolated vertices* do not appear in any face vertices, they can be deleted directly. Afterward, *self-intersecting faces* are detected and remeshed by using an AABB tree. Finally, we detect and remove an *inner face* according to whether multiple random rays shooted from a face can reach infinity. Experiments on ModelNet40 dataset illustrate that our method can eliminate the deficiencies of 3D meshes thoroughly.**

***Index Terms*—3D Mesh; Geometric deficiencies; Mesh repair.**

I. Introduction

Since 3D meshes have many advantages (*e.g.*, rich topological information, lightweight, compact, and easier geometric transformation), 3D computer vision applications begin to pay more attention to mesh data, such as object reconstruction [1] [2], scene reconstruction [3] [4] and object recognition [5] [6] [7]. However, most meshes have a lot of geometric deficiencies when they are generated, *e.g.*, ModelNet40 [8] and ShapeNet [9] datasets. Geometric deficiencies may violate the topology structure of an object, waste computing and storage resources, and affect make use of the advantages of 3D meshes. The common geometric deficiencies are presented in Fig. 1: (a) Duplicate elements are some vertices or faces that appear repeatedly. They make some structures of the mesh that should be topologically connected just overlaid together, *i.e.*, topology connections erroneous. (b) Isolated/Unreferenced vertices are not used by any faces, which may affect the normalization of 3D meshes. If an isolated vertex is very far away from 3D meshes, the center of the bounding box is not the center of 3D meshes after normalization, which causes the mesh far away from the screen center and may become very small to be invisible. (c) Degenerate faces consist of three same vertices or three collinear vertices. Degenerate faces and isolated vertices can be regarded as noise, which cannot provide any useful information for describing a shape. In addition, they waste computing and storage resources. (d) Self-intersecting faces

The corresponding author is Prof. Yang Cong. This work is supported by National Nature Science Foundation of China under Grant (61722311, U1613214, 61821005, 61533015) and Cooperation Projects of CAS & ITRI (CAS-ITRI201905).



(a) Duplicate elements. (b) Isolated vertices.

(c) Degenerate faces

(d) Self-intersection. (e) Inner faces.

Fig. 1. The common deficiencies of 3D meshes, which can be eliminated by the proposed method. (a) Duplicate elements are some vertices or faces appear repeatedly that marked in red. (b) Isolated vertices are highlighted in blue, which are not used by any faces. (c) Normal triangular face (left) and degenerate faces (middle and right) whose three vertices degenerate to a vertex (*i.e.*, for a face t_i, $v_{p1} = v_{p2} = v_{p3}$, where v_{p1}, v_{p2}, and v_{p3} are three vertices of a triangular face) or a line segment. (d) Self-intersection means their intersection is not an edge of the mesh. We visualize them in zoomed parts. (e) In the cross-section view of a mesh model, the marked faces are inner faces.

are not a realistic representation of geological structures [10], and the intersections of these faces are not edges of 3D meshes. (e) Inner surfaces are unnecessary in general computer graphic and 3D computer vision applications that need the outer contour of 3D meshes. Although inner surfaces generally cannot affect the outer contour of 3D meshes, they affect extract features from the outer contour. Above geometric deficiencies overwhelm the advantages of meshes and become obstacles for utilizing meshes.

There are some existing methods to deal with meshes, such as mesh denoising [11] [12], mesh surface simplification [13] and inside-outside classification of 3D meshes (*e.g.*, ray casting-based method [14], winding number [15], and a signed distance field-based method [16]). These methods only process one type of many deficiencies and cannot eliminate most deficiencies in an end-to-end manner. Specially, they cannot eliminate deficiencies that affect mesh topology.

To address these issues and better make use of advantages of 3D meshes, we propose an end-to-end mesh processing algorithm to eliminate five mentioned deficiencies thoroughly. To be specific, each type of deficiency is eliminated by a corresponding operation. These key operations are organized in a specific and reasonable order, *i.e.*, removing duplicate elements, degenerate faces, isolated vertices, self-intersecting faces, and inner faces. Experiment results on mesh model dataset demonstrate the effectiveness of our algorithm.

The main contributions can be summarized as follows:

- An end-to-end algorithm that organizes key operations in a specific and reasonable order is proposed to eliminate these mentioned key deficiencies, which can make us utilize advantages of 3D meshes and reduce storage and computing resources.
- We propose simple and effective methods to remove degenerate faces and self-intersecting faces according to the outer product of any two edges and intersection of two faces respectively.
- We propose a new method to remove inner faces of 3D meshes effectively by combining ray-casting and voting strategies, which simulate shoot multiple random rays from a face and accumulate the number of rays that can reach infinity, then decide whether to delete the face.

II. The Proposed Method

In this section, we first introduce corresponding operation of removing each deficiency (*i.e.*, removing duplicate elements, isolated vertices, self-intersecting faces, and inner faces), then introduce the overall scheme of our end-to-end geometric deficiency elimination algorithm.

A 3D mesh model can be regarded as a graph that consists of vertices, edges and triangular faces, defined as $\mathcal{M} = \{\mathcal{V}, \mathcal{T}\}$. $\mathcal{V} = \{v_i\}_i^{N_v}$ represents the aggregation of N_v vertices; $\mathcal{T} = \{t_i\}_{i=1}^{N_t}$ represents the aggregation of N_t triangular faces, each face is represented as a 3-tuple of vertex indices $t_i = (p_1, p_2, p_3)$, $1 \leq p_1, p_2, p_3 \leq N_v$ and consists of three edges $e_i^1 = v_{p_2} - v_{p_1}$, $e_i^2 = v_{p_3} - v_{p_2}$, and $e_i^3 = v_{p_3} - v_{p_2}$.

Algorithm 1 Correct Face Orientation

```
Input: $\mathcal{M} = \{\mathcal{V}, \mathcal{T}\}$, desired total rays $N_{\max}$, minimum rays
    $N_{\min}$ of each face;
Output: A repaired 3D mesh model $\mathcal{M}^{'}$;
1:  Compute area $s_i$ of per face $t_i$;
2:  $S = \sum_{i=1}^{T} s_i$;
3:  for $i = 1, ..., T$ do
4:      $c^i_{\text{front}} = 0$, $c^i_{\text{back}} = 0$;
5:      $n_i = \max(s_i/S * N_{\max}, N_{\min})$;
6:      Randomly sample rays $\{r^j_{\text{front}}, r^j_{\text{back}}\}_{j=1}^{n_i}$ from both
        sides of $t_i$. $r^j_{\text{front}}$ and $r^j_{\text{back}}$ have same origin and shoot
        in opposite directions;
7:      for $j = 1, ..., n_i$ do
8:          if $r^j_{\text{front}}$ can reach infinity then
9:              $c^i_{\text{front}} = c^i_{\text{front}} + 1$;
10:         end if
11:         if $r^j_{back}$ can reach infinity then
12:             $c^i_{\text{back}} = c^i_{\text{back}} + 1$;
13:         end if
14:     end for
15:     if $c^i_{\text{front}} < c^i_{\text{back}}$ then
16:         flip $t_i$;
17:     else
18:         pass;
19:     end if
20: end for
    return $\mathcal{M}^{'}$;
```

Remove duplicate elements. Duplicate elements are duplicate vertices and duplicate faces. A 3D mesh $\mathcal{M}$ may contain some vertices that satisfy:

$$v_i = v_j,\ i \neq j,\ v_i, v_j \in \mathcal{V}, \tag{1}$$

then v_i and $V_{\text{dup}} = \{v_j\}$ are duplicate vertices. The duplicate faces are different faces $t_i = (p_1^i, p_2^i, p_3^i)$ and $t_j = (p_1^j, p_2^j, p_3^j)$ which have the same 3-tuple of vertex indices, but the order may be different, *i.e.*,

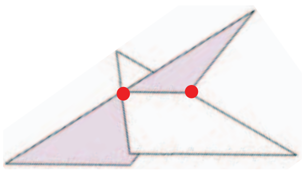
$$\{p_1^i, p_2^i, p_3^i\} = \{p_1^j, p_2^j, p_3^j\}. \tag{2}$$

After duplicate elements are detected, we leave only one from the duplicate elements. The corresponding operation is denoted as f_{dedup}.

New duplicate faces may produce after removing the duplicate vertices, therefore, duplicate vertices should be removed before detecting and removing duplicate faces. The vertices indices list changes when a vertex is removed, then each 3-tuple of vertex indices for $\mathcal{T}$ needs to be updated simultaneously when remove each duplicate vertex.

Remove degenerate faces. As illustrated in Fig. 1(c), degenerate faces are some faces which degenerate into a point or a line segment. They can be detected according to the following formula:

$$e_i^j \times e_i^k = 0,\ \exists\ j, k \in t_i,\ j \neq k, \tag{3}$$


(a) Intersection of two faces.

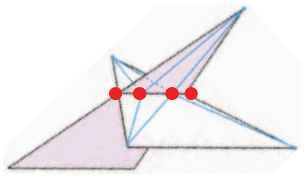

(b) Remesh self-intersecting faces.

Fig. 3. Illustration of our method for detecting and remeshing self-intersecting faces. (a) Detect self-intersecting faces: we get two endpoints (*i.e.,* two red points) of the intersection line of two faces firstly, then self-intersection faces satisfy that one of two endpoints is not the vertex of the two faces. (b) Remesh self-intersecting faces: we take an appropriate number of points in equal intervals on the intersection line of two faces, then triangulate two faces separately.

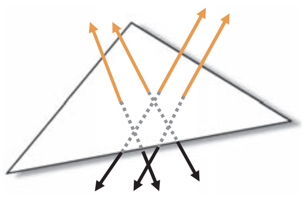


Fig. 4. Simulating rays shooted from a faces. Each orange ray is shooted from the front side of a face, which has a black opposite ray is shooted from the another side.

i.e., as for a degenerate face, the outer product of any two edges is zero. After detecting this deficiency, we remove the 3-tuple t_i and keep the vertices of the degenerate face. Because these vertices may be used by other faces, and they can be removed as isolated vertices if they are not used by other faces. This operation is denoted as $f_{\text{remove\_deg}}$.

Remove isolated vertices. Isolated vertex is not indexed by any face, *i.e.*, vertex v_i satisfy :

$$i \notin t_i = (p_1, p_2, p_3), \ \forall \, t_i \in \mathcal{T}. \tag{4}$$

These vertices can be deleted directly. We denote this operation as $f_{\text{remove\_iso}}$. Similar to removing duplicate vertices, when removing each isolated vertex, each 3-tuple of vertex indices for $\mathcal{T}$ needs to be updated simultaneously.

Remesh self-intersections. The self-intersecting faces can be repaired by our remeshing method. This method firstly detects the intersection line of two faces is a common edge of two faces or not by using an AABB tree. To be specific, as shown in Fig. 3(a), we get two endpoints (*i.e.,* two red points) of the intersection line of two faces, then no-common edge satisfys that one of two endpoints is not the vertex of the two faces. If it is not a common edge (*i.e.*,two faces are self-intersecting), we take an appropriate number of points at equal intervals on the intersection line and triangulate two faces separately, as depicted in Fig .3(b). If it is a common edge of two faces, we detect next pair faces. We denote this operation as $f_{\text{remesh\_si}}$.

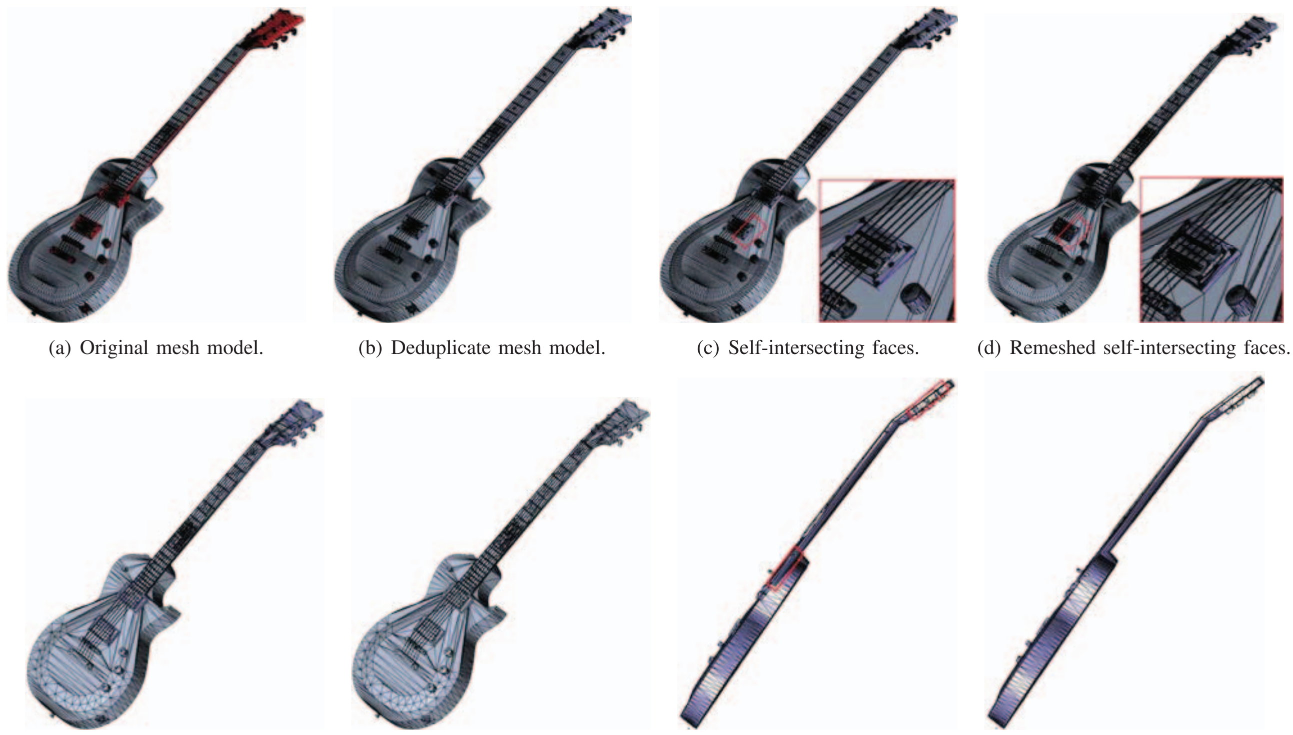
Remove inner faces. The orientation of some faces are incorrect after the operation f_{dedup} or are incorrect original, for instance, the mesh model of Fig. 1(d), where gray indicates outside and light blue indicates inside. Before removing the inner faces of 3D meshes, correcting facets orientation is necessary and can make visualization better. The method of [17] can solve it well, which is summarized in **Algorithm 1**. Specifically, for each face t_i, this method randomly samples a large number of rays, whose origin and direction are both random and number proportional to the area of the face. Then, as shown in Fig. 4, two rays are shooted in opposite directions for each ray origin, *i.e.*, r^j_{front} and r^j_{back}. The corresponding counter c^i_{front} (resp. c^i_{back}) is incremented if a ray shooted from the front side (resp. back) of the face and does not intersect with any other faces. After shooting all the rays, the face t_i is flipped if $c^i_{\text{front}} < c^i_{\text{back}}$, *i.e.*, $t_i = (p_1, p_2, p_3) \rightarrow t_i = (p_1, p_3, p_2)$.

Algorithm 2 Remove Inner Faces

```
Input: $\mathcal{M} = \{\mathcal{V}, \mathcal{T}\}$, desired total rays $N_{\max}$, minimum rays
    $N_{\min}$ of each face;
Output: A repaired and 3D mesh model $\mathcal{M}'$;
 1: Compute area $s_i$ of per face $t_i$;
 2: $S = \sum_{i=1}^{T} s_i$;
 3: for $i = 1, ..., T$ do
 4:   $c^i_{\text{inf}} = 0$;
 5:   $n_i = \max(s_i/S * N_{\max}, N_{\min})$;
 6:   Randomly sample rays $\{r_j\}_{j=1}^{n_i}$ from the outside of $t_i$;
 7:   for $j = 1, ..., n_i$ do
 8:     if $r_j$ can reach infinity then
 9:       $c^i_{\text{inf}} = c^i_{\text{inf}} + 1$;
10:     end if
11:   end for
12:   if $c^i_{\text{inf}} < 0.05 n_i$ then
13:     delete $t_i$;
14:   end if
15: end for
16: $\mathcal{M}' = f_{\text{remove\_iso}}(\mathcal{M})$;
    return $\mathcal{M}'$;
```

For detecting and removing inner faces of 3D meshes, a method based on ray casting and voting strategies inspired by [17], which is summarized in **Algorithm 2**. The main idea is to judge each face whether visible from the outside or not, which is decided by randomly shooting many rays from the outside of the face (*i.e.*, orange rays of Fig. 4). Specifically, for every face t_i, we randomly sample n_i ($n_i \geq 100$) points proportional to the face's area from the outside of the face as the ray origins, and the direction of each ray is also randomly sampled. If a ray does not intersect with any other faces in 3D meshes, *i.e.*, it can reach infinity, which indicates the corresponding face is an outer face, the corresponding score counter c^i_{inf} is incremented. After shooting all rays, the face can be regarded as an inner face if $c^i_{\text{inf}} < 0.05 n_i$. Then 3-tuple t_i can be removed, while keeping its vertices that may be used by outer faces. After removing all inner faces, $f_{\text{remove\_iso}}$ is conducted again to remove those vertices that only used by inner faces.

Overall Scheme. The overall processing scheme is shown in **Algorithm 3**. First, operations f_{dedup}, $f_{\text{remove\_deg}}$ and



(a) Original mesh model. (b) Deduplicate mesh model. (c) Self-intersecting faces. (d) Remeshed self-intersecting faces.

(e) Simplified mesh model. (f) After correcting faces' orientation. (g) Inner faces. (h) After removing inner faces.

Fig. 2. Visualization of the intermediate processing results for guitar_0205 in ModelNet40 [8] dataset. Gray face indicates outside and light blue face indicates inside. Processing sequence: (a) Original mesh model (red part is duplicate elements). (b) Mesh model processed via f_{dedup}, $f_{\text{remove\_deg}}$ and $f_{\text{remove\_iso}}$. Light blue face means its orientation is incorrect. (c) A local patch of self-intersecting faces. (d) Self-intersecting faces are repaired by remeshing self-intersections algorithm. (e) We simplify the mesh model to speed up the processing of subsequent steps. (f) Mesh model of previous steps exists wrong face orientation, which is corrected via **Algorithm 1**. (g) A cross-section view of the model to reveal its inner structure. (h) A cross-section view of the model after removing inner faces, which proves that we can remove the inner faces thoroughly.

$f_{\text{remove\_iso}}$ are conducted one by one to remove duplicate elements, degenerate faces, and isolated vertices successively. Second, we normalize 3D meshes model into a unit sphere for the convenience of visualization in subsequent steps. Afterward, $f_{\text{remesh\_si}}$ is conducted to remesh self-intersecting faces. $f_{\text{remesh\_si}}$ may produce duplicate elements, so operation f_{dedup} is conducted again. Some mesh models may have a large number of vertices, which costs much processing time. Thus 3D meshes are simplified to the expected number of vertices by a mesh simplification algorithm [13], which can speed up the processing of subsequent steps and is denoted as $f_{\text{simplify}}(\mathcal{M}, N_d)$, where N_d is the expected number. Finally, the face orientation is corrected via **Algorithm 1** and the undesired inner faces are removed via **Algorithm 2**.

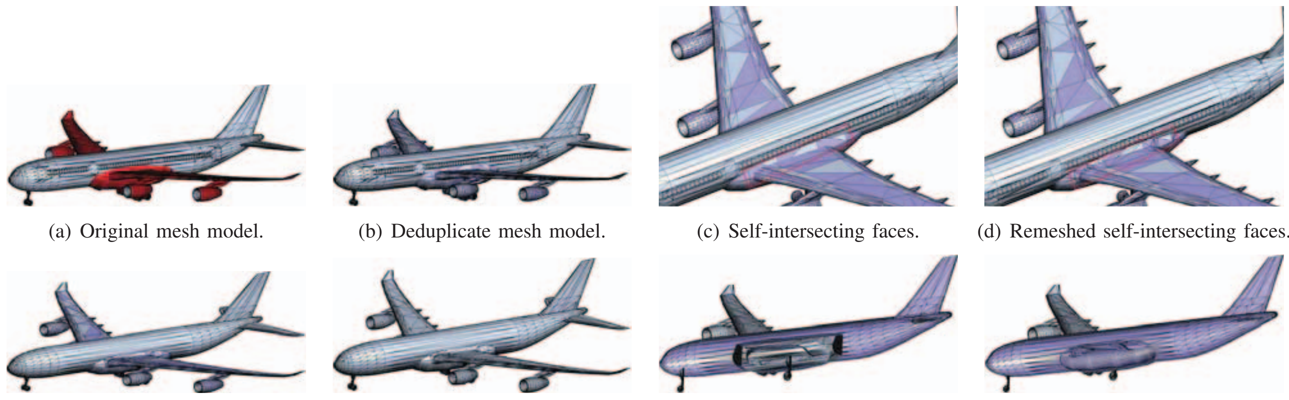
Algorithm 3 Eliminate Geometric Deficienciey of a 3D Mesh

```
Input: $\mathcal{M} = \{\mathcal{V}, \mathcal{T}\}$, expected number $N_d$;
Output: A repaired and simplified 3D mesh $\mathcal{M}^{'}$;
1: $\mathcal{M} = f_{\text{dedup}}(\mathcal{M})$;
2: $\mathcal{M} = f_{\text{remove\_deg}}(\mathcal{M})$;
3: $\mathcal{M} = f_{\text{remove\_iso}}(\mathcal{M})$;
4: Normalize $\mathcal{M}$ into a unit sphere;
5: $\mathcal{M} = f_{\text{remesh\_si}}(\mathcal{M})$;
6: $\mathcal{M} = f_{\text{dedup}}(\mathcal{M})$;
7: $\mathcal{M} = f_{\text{simplify}}(\mathcal{M}, N_d)$;
8: Correct each facet's orientation Algorithm 1, get $\mathcal{M}$;
9: Remove the inner structure via Algorithm 2, get $\mathcal{M}^{'}$;
   return $\mathcal{M}^{'}$;
```

III. Experiment Results

We implement the overall processing scheme in C++ based on the Libigl library [18], and verify the validity of the proposed method on ModelNet40 [8] dataset that has 12,311 3D mesh models. The intermediate processing results of guitar_0205 and airplane_0635 via **Algorithm 3** are shown in Fig. 2 and Fig. 5 respectively.

Then, we introduce the intermediate processing results of airplane_0635 in detail. The original mesh model has 25,085 vertices and 31,222 faces and is illustrated in Fig. 5(a), and the red part is duplicate elements. Fig. 5(b) shows the mesh model processed via f_{dedup}, $f_{\text{remove\_deg}}$ and $f_{\text{remove\_iso}}$ (*i.e.*, step 1-3 of **Algorithm 3**), and the number of vertices and faces decrease to 11,492 and 22,528 respectively. Comparing with the original mesh model, the number of vertices reduces more than half. The results illustrate that the original mesh may contain massive duplicate elements, degenerate faces and isolated vertices that waste much storage and computing resources



(a) Original mesh model. (b) Deduplicate mesh model. (c) Self-intersecting faces. (d) Remeshed self-intersecting faces.

(e) Simplified mesh model. (f) After correcting faces' orientation. (g) Inner faces. (h) After removing inner faces.

Fig. 5. Visualization of the intermediate processing results for airplane_0635 in ModelNet40 [8] dataset. Gray face indicates outside and light blue face indicates inside. Processing sequence: (a) Original mesh model (red part is duplicate elements). (b) Mesh model is processed via f_{dedup}, $f_{\mathrm{remove\_deg}}$ and $f_{\mathrm{remove\_iso}}$. Light blue face means its orientation is incorrect. (c) A local patch of self-intersecting faces. (d) Self-intersecting faces are repaired by remeshing self-intersections algorithm. (e) We simplify the mesh model to speed up the processing of subsequent steps. (f) Mesh model of previous steps exists wrong face orientation, which is corrected via **Algorithm 1**. (g) A cross-section view of the model to reveal its inner structure. (h) A cross-section view of the model after removing inner faces, which proves that we can remove the inner faces thoroughly.

when saving it to disk or conduct some computing tasks on it. Then, self-intersecting faces are remeshed. Comparing Fig. 5(c) and Fig. 5(d), the self-intersecting faces can be remeshed well, which provides a guarantee for effective removing of the inner faces. Then we conduct f_{dedup} and f_{simplify}, and the result is shown in Fig. 5(e). Contrasting Fig. 5(e) and Fig. 5(d), we can find some small and unimportant faces (*e.g.*, the windows of the airplane) are simplified/disappeared, which can speed up subsequent processing steps. In addition, the orientation of some faces is wrong in Fig. 5(b) - Fig. 5(e) (*i.e.*, gray indicates outside and light blue indicates inside). So **Algorithm 1** is used to correct the orientation of these faces, and the result is shown in Fig. 5(f). Then we remove the inner structure via **Algorithm 2**. The slices before and after removing the inner faces are respectively shown in Fig. 5(g) and 5(h), which proves the proposed **Algorithm 2** can effectively remove the inner faces. Specially, the mesh model owns 10,000 vertices and 23,252 faces before removing inner faces (*i.e.*, Fig. 5(g)), and owns 8,564 and 16,189 faces after removing inner faces (*i.e.*, Fig. 5(h)), which indicates remove inner faces can save resources for other tasks that consume mesh. In the future, 3D meshes processed by our method could be efficiently applied into many computer vision applications. *e.g.*, object recognition [19], [20], domain adaptation [21].

IV. Conclusion

In this paper, we propose an end-to-end processing algorithm to effectively eliminate key geometric deficiencies of 3D meshes, then we can reduce storage and computing resources and utilize advantages of deficiency-free meshes in other tasks (*e.g.*, object reconstruction, scene reconstruction, and object recognition). Specifically, we remove degenerate faces and self-intersecting facessimply and effectively according to the outer product of any two edges and intersection of two faces respectively. In addition, we remove inner faces of 3D meshes effectively by combining ray-casting and voting strategies. Intermediate processing results demonstrate the effectiveness of our method. In the future, we plan to research 3D mesh object recognition and 3D mesh scene understanding, for which an effective preprocessing algorithm and high quality 3D meshes are necessary.

References

[1] N. Wang, Y. Zhang, Z. Li, Y. Fu, W. Liu, and Y.-G. Jiang, "Pixel2mesh: Generating 3d mesh models from single rgb images," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 52–67.

[2] A. Kundu, Y. Li, and J. M. Rehg, "3d-rcnn: Instance-level 3d object reconstruction via render-and-compare," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 3559–3568.

[3] E. Piazza, A. Romanoni, and M. Matteucci, "Real-time cpu-based large-scale three-dimensional mesh reconstruction," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1584–1591, 2018.

[4] A. Rosinol, T. Sattler, M. Pollefeys, and L. Carlone, "Incremental visual-inertial 3d mesh generation with structural regularities," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8220–8226.

[5] J. Xie, G. Dai, F. Zhu, E. K. Wong, and Y. Fang, "Deepshape: Deep-learned shape descriptor for 3d shape retrieval," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 7, pp. 1335–1345, 2016.

[6] M. Aubry, U. Schlickewei, and D. Cremers, "The wave kernel signature: A quantum mechanical approach to shape analysis," in *2011 IEEE international conference on computer vision workshops (ICCV workshops)*. IEEE, 2011, pp. 1626–1633.

[7] B. Ma, Y. Cong, H. Liu, and X. Tang, "Tpn: Topological perception network for 3d mesh representation," in *2020 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2020, pp. 2711–2715.

[8] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1912–1920.

[9] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su *et al.*, "Shapenet: An information-rich 3d model repository," *arXiv preprint arXiv:1512.03012*, 2015.

[10] G. Caumon, P. Collon-Drouaillet, C. L. C. De Veslud, S. Viseur, and J. Sausse, "Surface-based 3d modeling of geological structures," *Mathematical Geosciences*, vol. 41, no. 8, pp. 927–945, 2009.

[11] P.-S. Wang, Y. Liu, and X. Tong, “Mesh denoising via cascaded normal regression.” *ACM Trans. Graph.*, vol. 35, no. 6, pp. 232–1, 2016.

[12] S. K. Yadav, U. Reitebuch, and K. Polthier, “Robust and high fidelity mesh denoising,” *IEEE transactions on visualization and computer graphics*, vol. 25, no. 6, pp. 2304–2310, 2018.

[13] M. Garland and P. S. Heckbert, “Surface simplification using quadric error metrics,” in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, 1997, pp. 209–216.

[14] F. S. Nooruddin and G. Turk, “Interior/exterior classification of polygonal models,” in *Proceedings Visualization 2000. VIS 2000 (Cat. No. 00CH37145)*. IEEE, 2000, pp. 415–422.

[15] A. Jacobson, L. Kavan, and O. Sorkine-Hornung, “Robust inside-outside segmentation using generalized winding numbers,” *ACM Transactions on Graphics (TOG)*, vol. 32, no. 4, pp. 1–12, 2013.

[16] H. Xu and J. Barbič, “Signed distance fields for polygon soup meshes,” in *Proceedings of Graphics Interface 2014*. Citeseer, 2014, pp. 35–41.

[17] K. Takayama, A. Jacobson, L. Kavan, and O. Sorkine-Hornung, “A simple method for correcting facet orientations in polygon meshes based on ray casting,” *Journal of Computer Graphics Techniques*, vol. 3, no. 4, p. 53, 2014.

[18] A. Jacobson *et al.*, “libigl,” https://github.com/libigl/libigl.

[19] Y. Cong, D. Tian, Y. Feng, B. Fan, and H. Yu, “Speedup 3-d texture-less object recognition against self-occlusion for intelligent manufacturing,” *IEEE transactions on cybernetics*, vol. 49, no. 11, pp. 3887–3897, 2018.

[20] H. Liu, Y. Cong, G. Sun, and Y. Tang, “Robust 3-d object recognition via view-specific constraint,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2020.

[21] J. Dong, Y. Cong, G. Sun, and D. Hou, “Semantic-transferable weakly-supervised endoscopic lesions segmentation,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 10 712–10 721.