A Robust Algorithm to Remove the Self-intersection of 3D Mesh Data without Changing the Original Shape

Jiang Zhu¹, Yurio Hosaka¹, Hayato Yoshioka²

¹ Department of Mechanical Engineering, Tokyo Institute of Technology
² Laboratory for Future Interdisciplinary Research of Science and Technology, Tokyo Institute of Technology
zhuj@mech.titech.ac.jp, 11-35, 2-12-1 O-okayama, Meguro, Tokyo, Japan

Abstract. Self-intersection is a serious problem in computational geometry and digital manufacturing. Because intersected parts do not exist in the physical world, the self-intersection must be removed for actually applications such as 3D printing or CNC machining. In the conventional method, such intersected parts are usually deleted, which causes the change of the original shape. In this research, a robust algorithm to remove the self-intersection of 3D mesh model is proposed. It could automatically detect the self-intersection, and remove the self-intersection without changing the original shape. The proposed algorithm has four steps. (1) Detect the intersected facets in the mesh model. (2) Divide the intersected facets along the crossing lines. (3) Find the outside facets. (4) Delete the inner facets and burying facets. Different models are used to do experiments, and the experimental results prove the validity and practicability of the proposed algorithm.

1. Introduction
Currently, 3D mesh models are widely used in modern manufacturing system. With the rapid development of 3D sensing technology and additive manufacturing technology, it becomes easier to acquire large scale 3D digital data of a physical objects using the 3D scanning technology, and the advanced 3D printing technology can easily build a physical model based on the acquired or designed 3D digital data. In such processes, 3D triangle mesh model is the most widely used method to represent the 3D models [1, 2]. For example, the STL (Stereolithography) file format is the most commonly used file format standard for 3D printing, and it is supported by almost all commercial CAD/CAM, and reverse engineering software.

In actual application of 3D mesh data such as additive manufacturing or FEM (finite element method) analysis, the quality of the mesh data is very important. A manifold triangle mesh should be a closed triangle mesh without topological errors and intersecting triangles. However, due to the limitation of the 3D sensing devices or triangulation algorithms, there are many kinds of non-manifold mesh exist, such as hole, zero thickness and self-intersection. Since such non-manifold meshes do not exist in physical world, they must be fixed in advance for actual usage.

In conventional reverse engineering process, the non-manifold meshes are fixed by the operator manually. The increasing of the mesh complexity makes such process a very time-consuming task. In order to realize the automation of 3D sensing and 3D printing, automatic detection and correction of the non-manifold mesh became an essential task to solve. In the past decades, there are many researcher proposed different algorithms to fix the non-manifold mesh [3-5]. However, there are few algorithms
developed to remove the self-intersection of the 3D mesh data. In the commercial reverse engineering software, the self-intersected triangles are firstly deleted, then re-triangulation is carried out to fill the resulted hole. However, in such process, the original boundary shape of 3D model will be changed due to the delete operation. It will cause the low accuracy of the 3D digital model.

In this research, a robust self-intersection removal algorithm for 3D mesh model is proposed. It could automatically detect the intersected parts in the 3D mesh model, and it could remove the self-intersected parts without changing the original boundary shape of the 3D model. The experimental result proofs the effectiveness of this proposed algorithm.

2. Algorithm to remove the self-intersection of 3D mesh model

2.1. Outline of the proposed algorithm
In this proposed algorithm, the basic idea is only the facets on the outer surface represent the boundary shape of the 3D model. The facets of a self-intersected mesh model could be classified into three class, facets on the outer surface, facets inside the surface, and intersected facets. The intersected facets can be either on the outer surface or insider the surface. In this algorithm, the intersected facets are divided into a set of sub-facets along the crossing line. Such sub-facets could be finally classified into facets on the outer surface, or facets inside the surface. Based on this idea, the proposed algorithm can be outlined as the following four steps.

![Figure 1. Overview of the proposed method](image)

(a). Detect self-intersected facets  
(b). Divide the intersected facets  
(c). Find the outside facets  
(d). Delete the inner and burying facets

(1) Detect intersected facets. In this algorithm it first computes the intersected facets in the mesh model, and calculates the crossing lines of the intersected facets, as shown in figure 1(a).

(2) Divide the intersected facets. In the intersected facets, some parts may be on the outer surface, while some parts may be inside the surface. Therefore, the intersected facets are divided into a set of sub-facets along the crossing line, as shown in figure 1(b).

(3) Find the outside facets. In this process, the algorithm determines whether a facet is on the outer surface or inside the surface, as shown in figure 1(c). All the facets on the outer surface represent the boundary of the original shape.

(4) Delete the inner facets and burying facets. The inner facets and the burying facets lies inside the surface of the 3D model. They are deleted in this step, as shown in figure 1(d).
2.2. Intersected facets detection and crossing line calculation
Before removing the self-intersection of the mesh, the intersected triangles and the crossing lines need to be found out. The self-intersection problem can be considered as a set of triangle-to-triangle intersection (TTI) problems, and it can be solved by the known TTI techniques [6]. In this research, in order to increase the processing speed, an approach based on tracing neighbour of intersected triangles (TNOIT) is used to calculate the intersected facets and crossing lines [7].

2.3. Intersected facets subdivision based on Delaunay triangulation with constraint
The intersected facets calculated in the previous step need to be divided into a set of sub-facets along the crossing line. In this algorithm, a modified Delaunay triangulation with constraint is developed for intersected facets subdivision. The vertices of the intersected triangles and the vertices of the crossing lines are used as the point set for Delaunay triangulation. The crossing lines calculated in the previous steps are used as the constraint lines. The details of Delaunay triangulation could be found in [8], here in this paper only the constraint will be explained. Suppose \( p_i \) is a new point to be added for Delaunay triangulation, and triangle \( T_{ABC} \) is the in circle triangle of point \( p_i \), the triangulation constraints can be expressed as the following rules, which are illustrated in figure 2.

1. If line \( L_{AB} \) is the constraint line, the new triangle will be \( T_{ABC} \) and \( T_{ABp_i} \).
2. If line \( L_{Cp_i} \) is the constraint line, and line \( L_{Cp_i} \) is intersected with line \( L_{AB} \), the new triangle will be \( T_{ACp_i} \) and \( T_{BCp_i} \).
3. If line \( L_{Cp_i} \) is the constraint line, while line \( L_{Cp_i} \) is not intersected with line \( L_{AB} \), the new triangle will be \( T_{ABC} \), \( T_{ABp_i} \), and \( T_{BCp_i} \).

![Figure 2. Illustration of constraints for Delaunay triangulation.](image)

2.4. Outside facets determination
After Delaunay triangulation along the crossing lines, connected sub-divided triangles replaced the previous intersected triangles. However, the resulted mesh is still a non-manifold mesh, because there are four triangles connected with each other on the crossing lines. Since the boundary shape of a 3D model is represented only by the facets on the outside surface, the facets inside the model should be found and eliminated. If one triangle on the surface is intersected with other triangles, usually the intersected triangle can be divided into part on the outside surface and part inside the body along the crossing line. In such cases, the outside facets is determined by examining the normal vectors of the facets on the non-manifold edge [8]. However, in some cases if the triangle on the surface is intersected with other triangles on multiple levels, method based on normal vectors will be lack of accuracy. Some fake outside part may be detected as outside part, while it is actually inside the body of the mesh model. In order to fix this problem, a 3D winding number method is developed in this research for outside facet detection.

Winding number method [9] is usually used to detect whether a point is inside a closed area in 2D plane. Based on the same idea, the 3D winding number algorithm is developed. Given a triangle \( T \) in a closed mesh \( M \). \( N \) is the normal vector of triangle \( T \), and triangle \( T \) belongs to plane \( P \). The winding number \( WN_T \) of triangle \( T \) can be calculated as the following steps:
(1) Find any point \( p \) inside triangle \( T \), draw a half-line in any direction in the positive half-space of the plane. The positive direction is defined as the direction, which normal vector \( N \) points to. Here for simplicity, the half-line \( HL \) in the normal direction, which passes the gravity center of the triangle is used.

(2) Compute the intersection between half-line \( HL \) and the other triangle facets of mesh \( M \). If half-line \( HL \) intersects with triangle \( T_i \) from the positive half-space, intersection direction value \( ID_i \) is assigned a value of +1. If half-line \( HL \) intersects with triangle \( T_i \) from the negative half-space, intersection direction value \( ID_i \) is assigned a value of -1.

(3) Winding number \( WN_T \) of triangle \( T \) is counted as the summary value of all the intersection direction values.

If triangle \( T \) is on the outer surface of mesh \( M \), winding number \( WN_T \) will be a value equal to 0. On the contrary, if triangle \( T \) is not on the outer surface of mesh \( M \), winding number \( WN_T \) will be a non-0 value as shown in figure 3(a).

2.5. Inner facets and burying facet deletion

Till this step all the triangles in the mesh model can be classified into either facets on the outer surface or facets inside the surface. Not only intersected triangles, some non-intersected triangles which lies inside the model as shown in figure 3(b) also need to be eliminated, because such facets does not exists in the physical world.

3. Experimental result

The proposed self-intersection removal algorithm was implemented on a computer (CPU: CoreI5 4200M, RAM: DDR3-1333 8GB, GPU: GTX860M) using C++ and OpenGL. STL file format is used as input and output. Many objects have been tested using this developed algorithm.

Figure 4 shows a simple Pyramid-in-Cube model. The model is generated by simply putting a pyramid model into a cube model, the outside view and inside view of the model is shown in figure 4(a) and (b) respectively. Figure 4(c) shows the result of intersected facet detection, and figure 4(d) shows the result of intersected facet subdivision. From the result shown in figure 4(d), it can be confirmed that all the intersected triangles are divided into small triangles along the crossing lines. Figure 4(e) and figure 4(f) show the result after self-intersection removal. From the inside view of the processed mesh model, as shown in figure 4(f), it can be confirmed that all the inside facets are successfully removed using the proposed method. The processed model is composed of facets only on the outer surface of the model.
The second model to test is intersected balls model, which is composed of ten balls which intersected with each other, each ball is composed of 300 triangle facets. The shade view of the model is shown in Figure 5. Processed result of intersected balls model.

Figure 4. Processed result of Pyramid-in-Cube model

Figure 5. Processed result of intersected balls model

The second model to test is intersected balls model, which is composed of ten balls which intersected with each other, each ball is composed of 300 triangle facets. The shade view of the model is shown in
figure 5(a). Figure 5(b) shows the wireframe view of the model, and the computed crossing lines, which are highlighted in red color. Figure 5(c) shows the intersection result which is processed by a well-known commercial reverse engineering software. Although we do not know the details about the algorithm in that software, it can be assumed that it adopted the logic of deleting the intersected parts, and filling the resulted holes from the experimental results. It can be clearly noticed that the original shape of the model is dramatically changed. Figure 5(d) shows the wireframe view of the model processed by our developed algorithm, the final output model have totally 3,552 triangle facets. From the enlarged view it can be found that the intersected triangles are successfully removed, and there is no intersection detected in the processed result by using commercial software CATIA.

The third model to test is the scanned data of a Nobel medal. The model is acquired using a 3D scanner (Next Engine HD Pro), and it contains 2,656 vertices and 5,067 triangle facets. Since in this proposed algorithm, the mesh model should be a closed model for outside facets detection, the input model is pre-processed using our previously developed algorithms to fill the holes and remove the zero-volume facets. Figure 6 shows the processed result of self-intersection removal. Totally, there are 33 intersected triangles detected, but all of the intersected triangles are very small. From the enlarged view of the result, it can be determined that the tiny self-intersected parts are successfully detected and replaced by the divided facet. The processed model is used for 3D printing by a 3D printer (Prodigy Plus, Stratasys), and no error occurs in the printing process.

4. Conclusion
In this research, a robust algorithm to remove the self-intersection of the mesh model is presented. This proposed algorithm could automatically detect the intersected parts in the 3D mesh model, and it could remove the self-intersected parts without changing the original boundary shape of the 3D model. The proposed method can be summarized into four steps. (1) Detect the intersected facets in the mesh model. (2) Divide the intersected facets along the crossing lines. (3) Find the outside facets. (4) Delete the inner facets and burying facets. The experimental result indicates that comparing with the commercial software, this proposed method is more effective to remove the self-intersected parts in the 3D model, while keep the original shape unchanged. It could be used to fix the non-manifold mesh problems in 3D printing.
printing or FEM analysis. In addition, this method can also be modified for Boolean operation of 3D mesh models.

References

[1] Zhu J, Tanaka T, Saito Y, 2007, Multi-resolution Mesh for Sculptured Surface Machining, Journal of Solid Mechanics and Material Engineering, Vol.1, No.1, pp.69-80.

[2] Roy M, Foufou S, Truchetet F., 2010, Recent advances in multiresolution analysis of 3D meshes and their applications, IS&T/SPIE Electronic Imaging, International Society for Optics and Photonics, California, USA, pp. 17-21.

[3] Jung WH, Shin HY, Choi BK, 2004, Self-intersection removal in triangular mesh offsetting, CAD’04 conference, Thailand, pp. 477-484.

[4] Chen Y, Wang H, Rosen DW, 2005, A point-based offsetting method of polygonal meshes, ASME Journal of Computing and Information Science in Engineering, pp.1-21.

[5] Habtamu MA, Min HK, 2015, Self-intersection avoidance for 3-D triangular mesh model, the 2015 world congress on advances in structural engineering and mechanics, Korea, pp.25-29.

[6] Moeller T, 1997, A fast triangle-triangle intersection test, Journal of graphics tools, Vol. 2, pp. 25-30

[7] Zhu J, Tanaka T, Saito Y, 2007, A rough cutting model generation algorithm based on multi-resolution mesh for sculptured surface machining, Journal of advanced mechanical design, systems, and manufacturing, Vol. 1, No. 5, pp. 628-639.

[8] Cignoni P, C. Montani, R. Scopigno, 1998, A fast divide and conquer Delaunay triangulation algorithm in E3, Computer-Aided Design, Vol. 30, No. 5, pp. 333-341.

[9] Alec J, Ladislav K, Olga SH, 2013, Robust inside-outside segmentation using generalized winding numbers, ACM Transactions on graphics, Vol. 32, Issue 4, Article No. 33.