Research and Application on Model Repairing Algorithm of 3D Modelling Technology

Jixin Tan [1, 2, a] and Jianxun Chen [1, 2, b]

[1]College of Computer Science and Technology, Wuhan University of Science and Technology, Wuhan, 430065, China

[2]Hubei province key laboratory of intelligent information processing and Real-time Industrial System, Wuhan University of Science and Technology, Wuhan, 430065, China

[a]tjx1021@qq.com, [b]cjxwh@wust.edu.cn

Keywords: Half-Edge data structure; STL File; 3D solid model; Topology structure; Normal vector

Abstract. In order to repair the errors of 3D solid model, this paper proposed methods that based on half-edge data structure. First, constructed the half-edge data structure through the STL file of 3D solid model. If the topological information is wrong, find the error vertices based on the feature that all of the edges opposite the vertex can constitute several loops by linking them, and separate the error vertices logically. If the normal vector of 3D solid model is wrong, find a triangle facet, whose normal vector is correct, and repair its adjacent facets, and then repair all of the adjacent facets in turns. At last, the algorithms are implemented and experiments are carried out to verify its correctness and effectiveness.

Introduction

Three-dimensional (3D) model plays a vital role in many industrial and scientific fields, such as product design and manufacturing, gaming and simulation. Visualization and computation of 3D solid model is the key of application and research. [1,2].

In the field of 3D modeling, STL (Stereo Lithography) file is widely used to store the 3D information of 3D solid model [3,4]. STL file format is created by the United States 3D Systems Corporation in 1987. It describes 3D solid model surface by small triangle facets dispersedly and approximately. The format STL file has four main principles: the first is the rule of common vertex, which claims that every two adjacent triangle facet must share two vertices; the second is the orientation rule, which claims the normal vector of a single facet follows the right-hand rule and its normal vector must point outside of the solid; the third rule is that the 3D solid, formed by all of the facets, must be closed; and the last rule is that the coordinate values of each vertex must be non-negative [5,6].

3D solid modeling usually use half-edge data structure [7] to represent the topological relations of the vertices, edges and facets. The half-edge data structure divides each one edge into two half-edges, which are in opposite orientation. As the Fig. 1, one half-edge can obtain its opposite half-edge, the previous half-edge and the next, and it can also obtain the vertices and the facets associated with it. Similarly, each vertex can get the half-edges who point it, and each facet can also get its associated half-edges. In this paper, the half-data structure is being used to describe the 3D solid model, and then repair the errors.

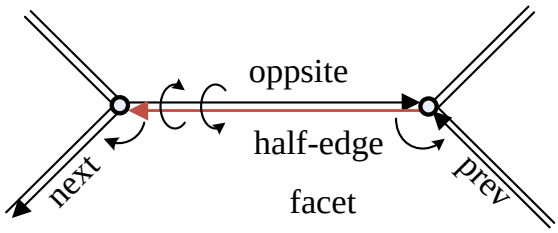


Figure 1. Half-edge structure

Analysis of Errors

STL files sometimes have some errors, and those errors can cause some errors in 3D solid model visualization and computation, which requires strict correctness and quality. [8] According to the error characteristics of the STL file, the errors can be divided into the following categories:

1. Holes and cracks: some triangle facets are missing.
2. Overlap: several triangle facets overlap.
3. Dislocation: the vertices who should overlap do not overlap;
4. Redundant: there exist isolated vertices, isolated edges, isolated facets, hanging edges, hanging facets. [9-11]

This paper focus on repairing two types of STL files errors:

1. Topology error. Two or more closed 3D solid models share the same vertex or edge, so that it causes the non-manifold error. [12] In these cases, the method to repair is to split the error vertex or edge; as a result, one solid model will be split into two models. The method in this paper repairs the topology error of 3D solid model by correcting the topology information of half-edge data structure.

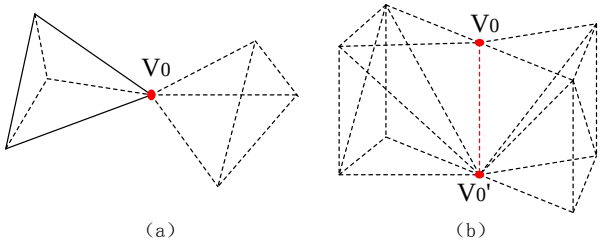


Figure 2. 3D solid models with topology error

2. Normal vector error. One or more triangle facets run counter to the second principle of STL file, that is, the normal vector of the triangle facets point to the inner of the 3D solid model. As a result, the 3D solid model is not closed. In the STL file, each triangle facet consists of indices of three vertices. The order of the indices, which follows the right-hand rule, determines the normal vector of the facet. The normal vector is correct while it points to the inner of 3D solid model, whereas it is wrong while it points to the outside. Repairing the normal vector, in fact, is to reorder the vertex indices of triangle facet.

Repairing Methods

Repairing of Topology Error. For the reason that the repairing operation is based on the half-edge data structure, so the half-edge data structure should be constructed first. This step comprises the following sub-steps. First, construct vertex structure, which includes coordinates information of vertices and an associated half-edge index array. Then construct half-edge structure, which includes a head vertex index, a tail vertex index, an opposite half-edge index, a previous half-edge index, a next half-edge index and the associated facet index. Finally, construct the facet structure, which includes the associated half-edge index array. So vertex, half-edge and facet can get their data each other via half-edge structure.

Before splitting the error vertices or edges, them must be located. There is a truth that the opposite edges of a normal vertex can formed only one loop end to end, but the opposite edges of a defective vertex can formed more. As shown in Fig. 2, the opposite edges of vertex V0 in (a) formed two loops and so do V0 in (b). According to this characteristic, the error vertices or edges can be located. The located and repair method just as follows steps:

Step 1. Traverse the vertices of the 3D solid model.

Step 2. If the amount of adjacent vertices of vertex V_0 (Fig. 3) is greater than or equal to 6, just record the number of adjacent vertices of vertex V_0. And further computation is needed to judge whether the topology information of V_0 is correct.

Step 3. Traverse and check the adjacent vertices of V_0 as following way: select an adjacent vertex

V_1 randomly, and add V_1 to an empty list called L. Then get the next adjacent vertex V_2, which follows the rule that $V_0V_1V_2$ is enclosed as an adjacent facet of V_0, and add V_2 to the end of L. When getting the next adjacent vertex V_3, V_3 also must meet the requirement that $V_0V_2V_3$ is an adjacent facet of V_0. Next, add other vertices in turns. Each time before adding a vertex V_i to the list, the judgement whether V_i is in L is to be operated.

1. If V_i is not in L, V_i should be added to the end of L, then repeat step 3.
2. If V_i is in L, check whether all adjacent points have been visited.

a. If all adjacent vertices have been visited, V_0 is the normal vertex. Then repeating step 2 to step 3 until all the operations are completed.

b. If there are adjacent vertices that has not been visited, a new vertex V_n, whose data and information is same to V_0 will be created. Then substitute V_n for all the V_0, who are in the facets that are made up of V_0 with V_i and vertices after V_i in L. And next, delete V_i and those vertices after V_i in L. Finally, check the vertex next to V_i whether it has been visited. If it has been checked, V_n will not be added to the L, otherwise, V_n will be added to the end of L. And repeat step 3.

Step 4. Repairing is completed. When repairing completed, one or more vertices will be added to the STL file.

Fig. 3 is an example that shows how to split a vertex. The following describes the changes of list L.

L: V_1 → (add vertices) → L: $V_1V_2V_3$

→ (V_1 is in L and there are some vertices that not visited. So create V_n to substitute the V_0 that linked with V_1, V_2 and V_3. And delete V_1, V_2 and V_3 from L. Finally, V_1 will not be added to L because the next vertex V_2 has been visited before.)

→ L: V_4 → (add vertices) → L: $V_4V_5V_6$

→ (V_4 is in L and all adjacent vertices of V_0 has been visited, so clear L and repairing completed.) → L: (NULL)

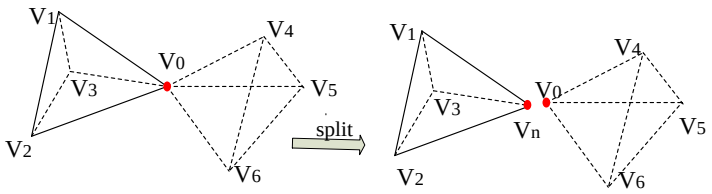


Figure 3. Split vertex

The case of that two or more 3D solid model share the same edge can be transformed to the case of share the same vertex. Obviously, after V_0 has been split, just split V_0' can split the edge so as to repair the error of sharing same edge.

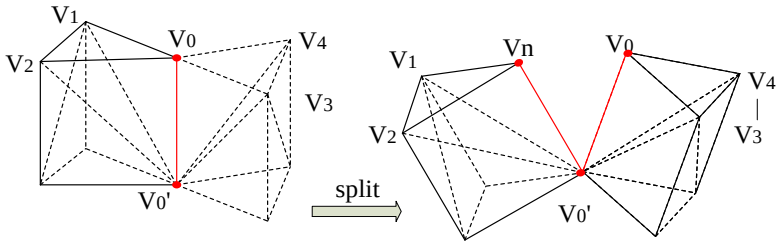


Figure 4. Split edge

Repairing of Normal Vector Error. To repair the normal vector, it must be judged whether it is correct or not first. Since the process of judgement is time-consuming, it is unnecessary to judge every facet, but just judge only one facet that can be confirmed its correctness. If the normal vector of the facet called F can be confirmed, just repair the normal vector of the adjacent facets of F via the properties of the half-edge structure. And then, repair all the adjacent facets of repaired facets in turns until all facets are correct.

As the above, the algorithm will find a correct triangles facet via the spatial relationship between

point and polyhedron. [13]

Step 1. Traverse the triangle facets of the 3D solid model, and check the normal vectors.

Step 2. Select a facet called F and calculate the normal vector called V of F. And then, obtain the space linear called L, who is perpendicular to F and meet in central point of F called P0 with F.

Step 3. Calculate all intersection points (P1, P2, … Pn) of L and the other facets of the model. Then calculate the vectors ($\overrightarrow{P_0P_1}$, $\overrightarrow{P_0P_2}$, ..., $\overrightarrow{P_0P_n}$) that P0 to the intersection points. Next, compare all vectors.

1. If $\overrightarrow{P_0P_1}$, $\overrightarrow{P_0P_2}$, ..., $\overrightarrow{P_0P_n}$ are all in the same orientation but opposite to V (Fig. 5 - a), the *V* is correct and finish to check other facets.
2. If $\overrightarrow{P_0P_1}$, $\overrightarrow{P_0P_2}$, ..., $\overrightarrow{P_0P_n}$ are all in the same orientation and coincident to *V* (Fig. 5 - b), the *V* direct to the inside of its 3D solid model. So reorder the vertices of *F* to make the normal vector direct to the outside.
3. If any two of $\overrightarrow{P_0P_1}$, $\overrightarrow{P_0P_2}$, ..., $\overrightarrow{P_0P_n}$ are in the opposite orientation (Fig. 5 - c), the normal vector *V* of *F* cannot be determined. Therefore, repeat step 2 to step 3 to continue checking the next facet.

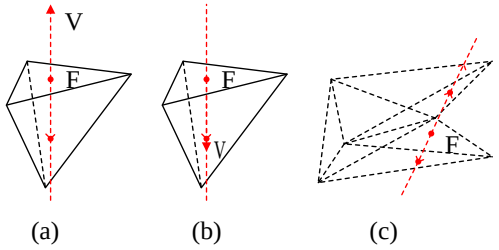


Figure 5. (a) Opposite orientation; (b) Coincident orientation; (c) Both orientation

Suppose that F is correct or has been repaired, the repair operation of other facets will be carried out as the following.

Step 4. Add F to a queue called Q, who recodes those facets whose adjacent facet will be checked, then traverse the adjacent facets of F and check them whether they are correct or not. If an adjacent facet called F1 of F has been checked, continue to check the next adjacent facet. Otherwise, check the half-edge of F1 that is along F. If the half-edge in F is $\overrightarrow{P_1P_2}$ while its opposite half-edge in F1 is also $\overrightarrow{P_1P_2}$, the normal vector of F1 is in wrong orientation so that reorder the vertices of F1 to repair its normal vector. If the normal vector of F1 is correct or has been repaired, just add F1 to Q and repeat step 4 to check the next adjacent facet of F.

Step 5. After all of adjacent facet of F has been checked, pop F from Q and repeat step 4 to step 5 to check the adjacent facets of the facet that is in the head of Q.

Step 6. Repairing completed.

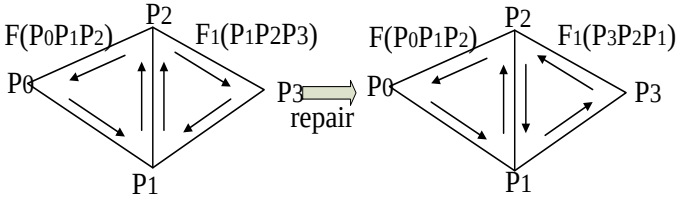


Figure 6. Repairing of normal vector error

Experiments

The repairing algorithms have been developed by C++, and the input and output are both STL files. The Computational Geometry Algorithms Library (CGAL) is used to verify the validity of algorithms and the graphical API is OpenGL, who is used to visualize the 3D solid model.

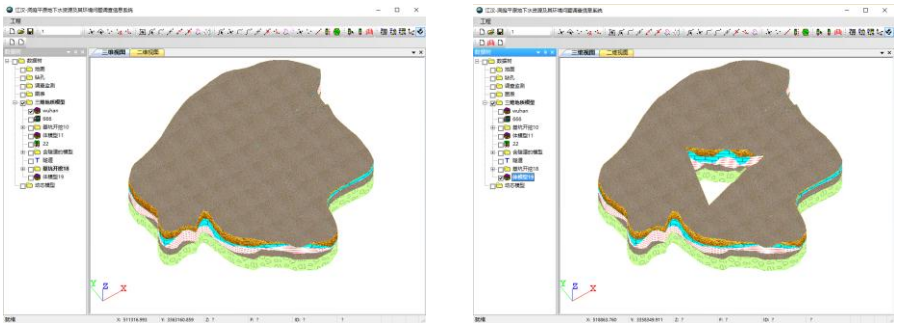
The object of Polyhedron_3 class of CGAL, which can be constructed by STL file, has three states: *is closed*, *is not closed* and *is empty*. They indicate the 3D solid model is correct, unclosed and empty respectively. If the 3D solid model has topology error, it will be in *is empty* state. If it has normal vector error, it will be in *is not closed* state. And the CGAL provides a Boolean operation algorithm, which only can work correctly on closed Polyhedron_3 objects, so as to it can be used to verify, too. If unclosed or empty Polyhedron_3 objects are processed by the Boolean operation algorithm, the result will be empty.

Several STL files have been chose as samples. After the incorrect files are processed by the algorithm, the vertex amount of them will increase (Table 1).

Table 1 Vertex amount and state of 3D solid model with topology error

STL files	before		after	
	vertices	state	vertices	state
heisha2-4	29	*is_empty*	31	*is_closed*
heisha11	1356	*is_empty*	1357	*is_closed*
han23-11	1466	*is_closed*	1466	*is_closed*

Fig. 7 shows the repaired 3D solid model can be processed by the Boolean operation algorithm successfully.



(a) (b)

Figure 7. (a): The repaired 3D solid model; (b): Result of the Boolean operation algorithm

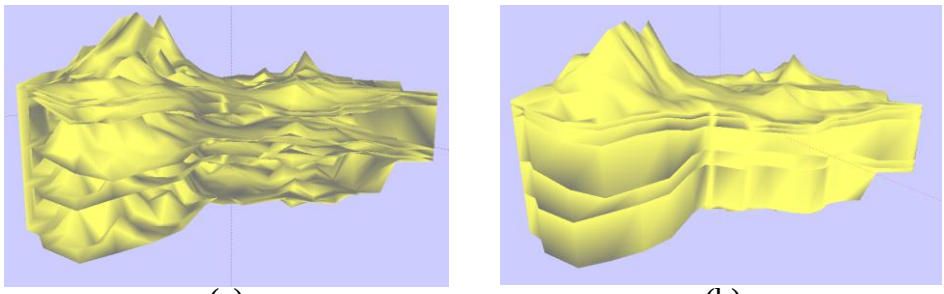
Table 2 State of 3D solid model with the normal vector error

STL files	before	after
layer-1	*is_not_closed*	*is_closed*
layer-2	*is_not_closed*	*is_closed*
layer-3	*is_not_closed*	*is_closed*

Fig. 8 (a) shows the 3D solid model with the normal vector error. For the normal vector error, the model drew by OpenGL is weird. Fig. 8 (b) shows the 3D solid model is correct after repairing.

Conclusions

Half-edge data structure can completely store the topology information of 3D solid model so as to it can be used to judge its correctness and repair it. This paper analyzed the topology error and normal vector error of 3D solid model constructed by STL file, and proposed the methods of repairing them. Algorithms have been developed in the experiment and the experiment verified the algorithms can repair the errors correctly.



(a) (b)

Figure 8. (a):before repairing; (b): after repairing

References

[1] Tao Ju. Fixing Geometric Errors on Polygonal Models: A Survey [J]. Journal of Computer Science and Technology: English, 2009, 24(1):19-29.

[2] Marco Attene, Marcel Campen, Leif Kobbelt. Polygon Mesh Repairing: An Application Perspective [J]. ACM Computing Surveys (scheduled to appear), 2013, 45(2):1-38.

[3] Liu Su. Visualize method and application of 3D geological model [J]. China High-Tech Enterprises, 2015, (20):133-134.

[4] Quang Xinghuai, LIU Ping, WANG Zhitao. 3D Geological Modeling and Application [J]. Electric Power Survey & Design, 2015, (z1):136-140.

[5] Zhou Huamin, Cheng Xuewen, Liu Fen, Li Dequn. Research on Repair Algorithms for STL Files [J]. JOURNAL OF COMPUTER-AIDED DESIGN & COMPUTER GRAPHICS, 2005, 17(4):761-767.

[6] Li Jiangfeng, Zhong Yuexian, Li Diansheng. Research on errors identifying and repairing of STL file [J]. Machine Design and Manufacturing Engineering, 2002, (3):40-42.

[7] Dan Englesson. THE HALF-EDGE DATA STRUCTURE: MODELING AND ANIMATION [EB/OL]. http://www.danenglesson.com/images/portfolio/MoA/halfedge.pdf. 2011.

[8] Sun Quanxin. Study of Three-dimensional Geological Model Construction for Mine Based on B-Rep [D]. Xi'an: Xi'an University of Architecture and Technology, 2013.

[9] Jin Fengzanm, Ying Shen, Li Lin. Validation rules and repair true 3D solid [J]. Geomatics and Information Science of Wuhan University, 2015, 45(2):258-262.

[10] Chen Ping, Zhang Zhenyan, Li Guojin. The algorithm of repairing holes based on STL [J]. Journal of Hubei University of Technology, 2012, 27(4):12-14.

[11] Yang Yang, Pan Mao, Wu Gengyu, Liu Yin. Sealed Geological Block Modeling in 3D Geological Structural Model [J]. JOURNAL OF COMPUTER-AIDED DESIGN & COMPUTER GRAPHICS, 2015, (10):1929-1935.

[12] Wang Zengbo. Fast topological reconstruction algorithm for a STL file [J]. Journal of Computer Applications, 2014, 34(9):2720—2724.

[13] Zhai Yan, Xu Weiya, Zhang Qiang. Judgment of topological relation between point and polygon or polyhedron [J]. Computer Engineering and Design, 2015, 36(4):972—976.