ISSN 1995-0802, Lobachevskii Journal of Mathematics, 2022, Vol. 43, No. 10, pp. 2846–2852. © Pleiades Publishing, Ltd., 2022.

Self-Intersections Elimination for Unstructured Surface Computational Meshes

S. A. Freylekhman[1]* and A. A. Rybakov[1]**

(Submitted by A. M. Elizarov)

[1] *Joint Supercomputer Center of the Russian Academy of Sciences—Branch of Scientific Research Institute of System Analysis of the Russian Academy of Sciences, Moscow, 119334 Russia*

Received June 20, 2022; revised July 3, 2022; accepted July 15, 2022

Abstract—During the numerical solution of the problem of icing a three-dimensional body, the problem arises of representing the surface of this body in the process of an ice buildup formation. The surface of the streamlined body is described by an unstructured surface computational mesh, the cells of which are triangles. In the process of icing calculation, the ice buildup can take arbitrary shapes, and the surface mesh must be rebuilt in accordance with this. At the same time, in reality, situations are not uncommon when individual parts of the ice buildup can connect with each other, layer on each other, forming voids and cavities inside the ice massif. In the numerical solution, this situation is expressed in the appearance of self-intersections of the surface mesh. The occurrence of self-intersections of the computational mesh makes it impossible to continue calculations on the surface of the body, so self-intersections must be detected and eliminated. This work is devoted to a practical algorithm for finding and eliminating self-intersections of an unstructured surface computational mesh in solving the problem of icing a streamlined body.

DOI: 10.1134/S1995080222130133

Keywords and phrases: *icing of a streamlined body, unstructured surface computational mesh, surface rebuilding, self-intersection of the computational mesh, mesh refinement, mesh repairing.*

1. INTRODUCTION

Calculation of icing of a streamlined body surface is an important practical task necessary to ensure the functionality and safety of aircraft [1–4]. Computer modeling of icing processes can be performed using a wide range of tools, among which are FENSAP-ICE [5], LEWICE [6], OpenFOAM [7], AIPAC [8], IceVision [9], and others. The result of the body surface icing calculations are the values of the mass (or volume of ice) accumulated in the different cells of the computational mesh. In accordance with this information, the surface of the body must be rebuilt in such a way that the new surface reflects the nature and quantitative indicators of the ice buildup. Rebuilding the surface based on the values of the ice mass in the cells of the computational mesh is usually performed iteratively (the ice buildup is modeled sequentially using separate layers [10, 11]). Special methods of surface smoothing and ice mass redistribution are used to process complex surface areas, depressions and fractures [12, 13].

Despite all the efforts used in surface rebuilding, parts of real computational meshes that are used in practical problems can intersect each other during rebuilding. This corresponds to situations where different parts of the ice buildup collide with each other upon magnification, forming internal cavities. The intersection of computational meshes generates loops that are hidden under the outer surface, and the presence of such loops makes it impossible to continue the calculations correctly. Various approaches to the elimination of self-intersections of computational meshes can be found in [14–16]. The main problem faced by the authors of this article in the numerical solution of practical problems was

*E-mail: freysa@jscc.ru

**E-mail: rybakov@jscc.ru

the strongly broken shape of the surface of the formed ice cover and often highly elongated cells of the computational mesh (cells with at least one angle close to zero). In this article, an attempt was made to give the most simplified algorithm for eliminating self-intersections for such "bad" computational meshes.

2. GENERAL SCHEME OF THE ALGORITHM FOR ELIMINATING SELF-INTERSECTIONS OF THE COMPUTATIONAL MESH

Before defining the general scheme of the algorithm, let us describe the unstructured surface computational mesh. The objects of such a mesh are vertices (they are associated with three Cartesian coordinates x, y, z), edges and triangular cells. In this case, the obligatory conventions are fulfilled: each edge connects exactly two vertices, each triangular cell is incident to three vertices and three edges, and an edge can be incident to at most two cells. The last requirement allows us to separate the inner and outer regions of the surface formed by a pair of cells that converge in some edge.

Figure 1 shows 4 phases of the intersection elimination algorithm. At the first phase (Fig. 1a), all pairs of cells (triangles) that potentially intersect are searched. In the second phase (Fig. 1b) all triangles' intersection points are searched (this figure shows the intersection points of P and Q). In the third phase (Fig. 1c), intersecting triangles are split into smaller ones so that the resulting refined mesh contains only triangles adjacent to edges and vertices (there is no intersections at internal points). It is obvious that after the execution of the third phase of the algorithm, edges incident to more than two cells can form in the mesh (in Fig. 1, the PQ edge is incident to four cells at once). Therefore, at the fourth phase of the algorithm (Fig. 1d), extra mesh cells are removed (sets of cells that form internal mesh cavities are removed). Next, we consider in more detail each of the above phases of the algorithm separately.

3. FINDING PAIRS OF POTENTIALLY INTERSECTING TRIANGLES

First, to eliminate self-intersections of the mesh, it is necessary to find all pairs of intersecting (but not adjacent) triangles. Of course, this issue can be approached directly by considering all n triangles from the computational mesh and performing an intersection analysis of all $n(n-1)/2$ pairs. However, this approach will take a lot of time, since the number of triangles in the computational mesh is large, and the procedure for determining the intersection of two triangles is not trivial. Let's solve the problem in a different way.

First, instead of pairs of intersecting triangles, we will look for pairs of potentially intersecting triangles. To do this, we introduce the following concepts. For triangle ABC let's consider cuboid $[\min(A_x, B_x, C_x), \max(A_x, B_x, C_x)] \times [\min(A_y, B_y, C_y), \max(A_y, B_y, C_y)] \times [\min(A_z, B_z, C_z), \max(A_z, B_z, C_z)]$ and call it the box of this triangle. We will say that two triangles potentially intersect if their boxes intersect. Unlike the intersection of triangles, determining the intersection of rectangular boxes (with sides parallel to the coordinate axes) is a simple operation. By analogy with the box of a triangle, one can define a box of an arbitrary geometric figure, as well as of a set of such geometric figures. We will use a box for set of triangles.

Let's build an auxiliary object called a cloud of triangles. In this cloud, we include all the triangles of the computational mesh. As an additional data field of the triangle cloud, we will use its box. Now let's divide the cloud of triangles into two subclouds using a section by an arbitrary plane (Fig. 2 shows a two-dimensional illustration of this process). Each triangle of the original cloud will be assigned to the subcloud that corresponds to that part of the half-space in which the center of the triangle fell. Thus, for the cloud of triangles under consideration, two child subclouds are formed. Note that it is not necessary to divide a subcloud into exactly two child subclouds; the number of child subclouds can be arbitrary. Continuing the process of dividing the clouds of triangles further, we obtain a structure called a cloud tree, at the root of which is the entire set of triangles, and in the leaves—individual triangles.

With the help of the formed auxiliary tree of triangle clouds, we can quickly find potentially intersecting triangles. To do this, it suffices to use the following fact: if two clouds of triangles T_1 and T_2 are not potentially intersecting (that is, their boxes do not intersect), then any two triangles $t_1 \in T_1$ and $t_2 \in T_2$ are not potentially intersecting. When using a tree-like search for potentially intersecting triangles using nested triangle cloud boxes, the time it takes to search through all the triangles in the set changes from linear to logarithmic, which greatly speeds up the calculations.

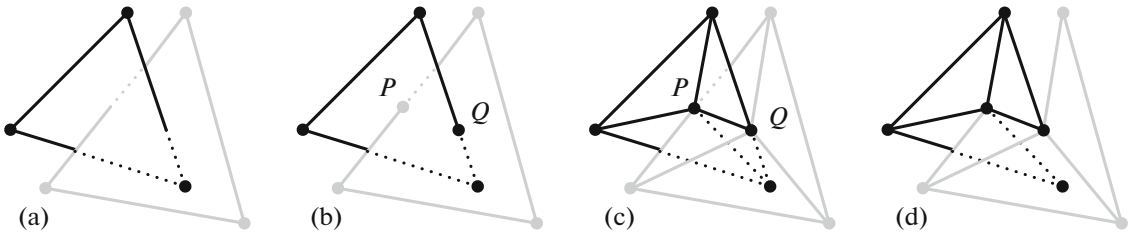

Fig. 1. Phases of self-intersections elimination.

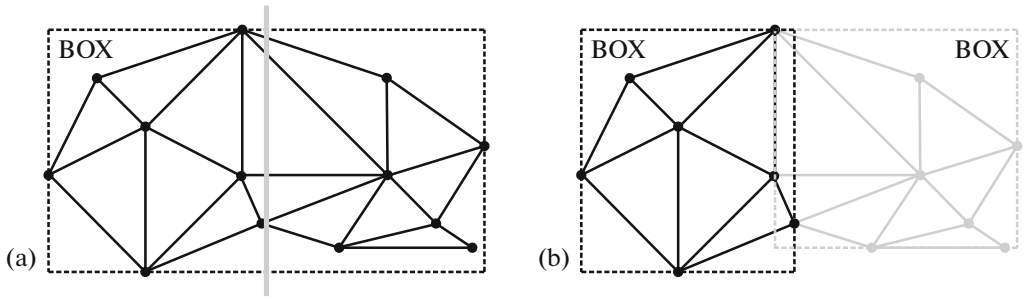

Fig. 2. Forming of a triangles cloud.

4. FINDING TRIANGLE INTERSECTION POINTS

After all pairs of potentially intersecting triangles have been determined, it is necessary to determine their intersection points. Note the following fact: if two triangles intersect in space, then at least one side of one triangle must intersect the other triangle. In other words, all intersection points of two triangles cannot be both interior points of one triangle and interior points of a second triangle. Therefore, the problem of finding the intersections of two triangles is reduced to the problem of the intersection of one triangle with the sides of the second one (and vice versa).

Based on this simple fact, to determine all points of intersection of triangles, it is enough to solve the following particular problems: establishing the fact that two vertices coincide (Fig. 3a), establishing the fact that a point lays inside a segment (Fig. 3b), establishing the intersection of two segments in space (Fig. 3c), establishing the fact of the intersection of a segment with a triangle (Fig. 3d).

Of the above particular problems, only the last two deserve attention. To solve them, we will use the representation of geometric objects in the form of the locus of points, namely: the segment PQ in space will be represented as the locus of points $P + \phi(Q - P), 0 \le \phi \le 1$, and the triangle ABC in space will be written as $A + \alpha(B - A) + \beta(C - A), \alpha \ge 0, \beta \ge 0, \alpha + \beta \le 1$ (record data are understood in vector form). Then, to find intersections of two segments AB and PQ in space, it suffices to solve the following system

$$\begin{cases} A_x + \psi(B_x - A_x) = P_x + \phi(Q_x - P_x), \\ A_y + \psi(B_y - A_y) = P_y + \phi(Q_y - P_y), \\ A_z + \psi(B_z - A_z) = P_z + \phi(Q_z - P_z), \\ 0 \le \psi \le 1, \\ 0 \le \phi \le 1. \end{cases} \tag{1}$$

A given system may have no solutions, exactly one solution, or infinitely many solutions. The case of an infinite number of solutions deserves attention. In this case, we are dealing with the imposition of two segments on top of each other, and the problem of finding intersections is reduced to the tasks of determining the coincidence of two points and determining whether a point lays inside a segment.

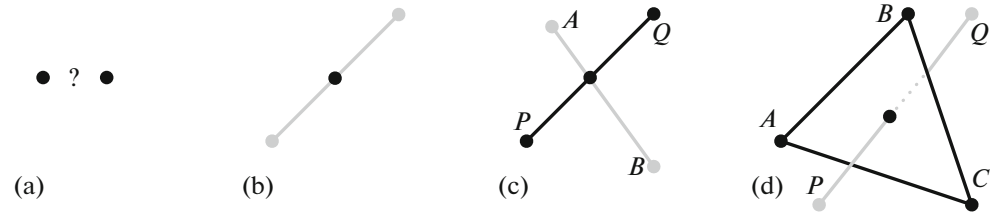

Fig. 3. Search for intersection points of different geometric primitives.

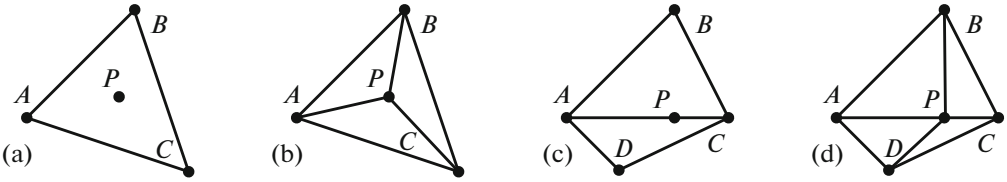

Fig. 4. Splitting of triangles by intersection points.

To search for intersections of the triangle ABC with the segment PQ, it is necessary to solve the following system of equations

$$\begin{cases} A_x + \alpha(B_x - A_x) + \beta(C_x - A_x) = P_x + \phi(Q_x - P_x), \\ A_y + \alpha(B_y - A_y) + \beta(C_y - A_y) = P_y + \phi(Q_y - P_y), \\ A_z + \alpha(B_z - A_z) + \beta(C_z - A_z) = P_z + \phi(Q_z - P_z), \\ \alpha \geq 0, \quad \beta \geq 0, \quad \alpha + \beta \leq 1, \quad 0 \leq \phi \leq 1. \end{cases} \tag{2}$$

A given system may have no solutions, have exactly one solution, or have an infinite number of solutions. In this case, the situation of an infinite number of solutions also requires special processing, in which the problem is reduced to determining the intersection points of two segments.

After searching for intersections of all geometric objects, it is necessary to place the found intersection points on the edges of the computational mesh (if the intersection occurred along the edge), or in cells (if the intersection occurred along the internal point of the triangle). After that, the transition to the next phase of the algorithm is performed.

5. SPLITTING TRIANGLES INTO SMALLER ONES

After the execution of the previous phase of the algorithm, self-intersection points were plotted on some elements of the computational mesh (edges and triangles). At this phase of the algorithm, it is necessary to refine the mesh in such a way that all such plotted points become nodes of the new refined mesh. This is the simplest procedure of all steps of the algorithm. If some plotted point P is inside the triangle ABC (Fig. 4a), then it is enough to split $ABC \to ABP, BCP, ACP$ (Fig. 4b). If several points fall inside the triangle at once, the partition is performed recursively. If some plotted point P falls on some edge AC of triangle ABC (Fig. 4c), then $ABC \to ABP, CBP$ must be split. In this case, if the edge AC is not the boundary of the computational mesh, then it is incident to another triangle, which must also be split along this point P (Fig. 4d).

It should be noted that with this approach, triangles with angles close to zero may appear. In order for the quality of the mesh not to decrease significantly, such triangles should be disposed of. For example, if the point P inside the triangle ABC is too close to the side AC (Fig. 4a), then the angle $\angle APC$ will be close to 180°, and in this In this case, to preserve the mesh quality, it is expedient to move the point P to the side AC.

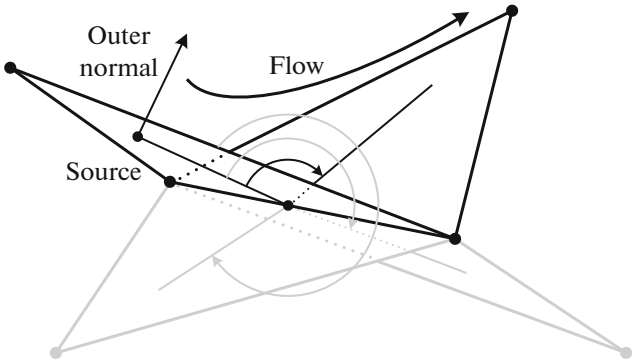

Fig. 5. Removing extra cells.

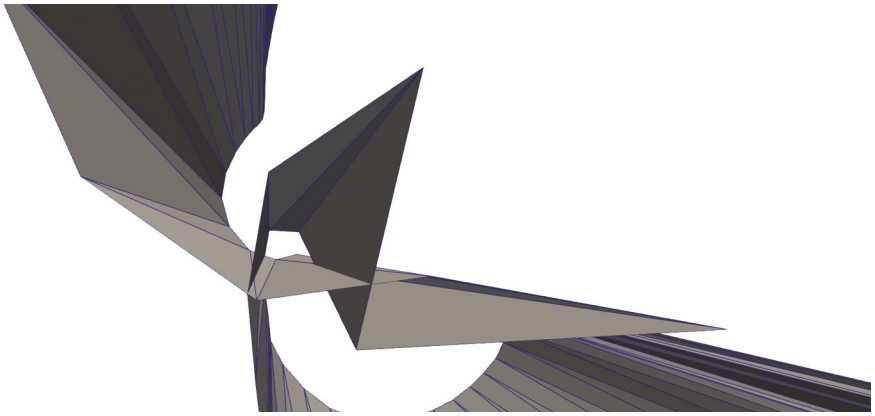

Fig. 6. An example of a computational mesh with self-intersection.

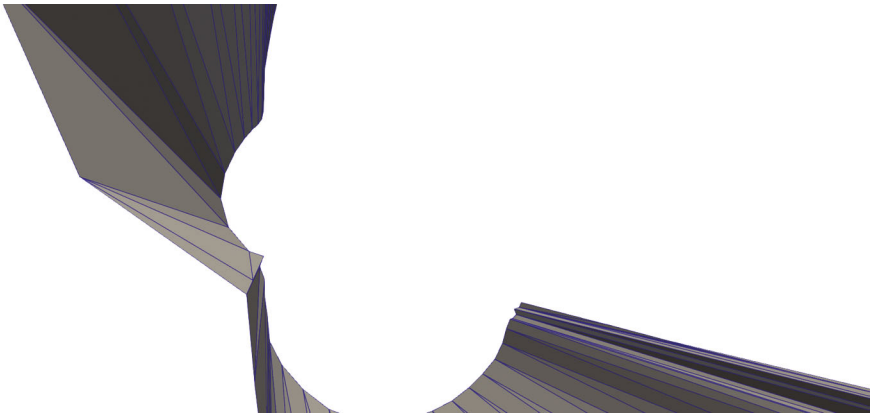

Fig. 7. The result of self-intersection elimination.

6. REMOVING EXTRA CELLS

The last step of the algorithm is to remove extra cells after mesh refinement. In places where the mesh self-intersects, after its refinement, edges are formed that can be incident to more than two cells (in the classical case, these are 4 cells, as shown in Fig. 5). In this case, for the correct representation of the surface, it is required that this edge be incident to only two cells (the remaining cells belong to the hidden self-intersection region and describe the cavity that must be cut out).

To determine extra cells, it is necessary to be able to identify at least one cell that is exactly related to the surface (in Fig. 5 the outward normal is indicated for such a cell). This can be achieved by traversing the surface from a region where there are definitely no self-intersections. That is, if it is known that

some cell of the computational mesh exactly refers to the surface and one has exactly one adjacent cell along some edge, then this adjacent cell also refers to the surface (it is uniquely determined from which to which cell the flow is flowing). Performing a traversal in this way, sooner or later we will either bypass all the cells (then, there are no self-intersections and the mesh does not need to be repaired), or we will come to an edge for which it is impossible to uniquely determine the flow direction through the edge (as shown in Fig. 5). For such edges, we will proceed as follows. A cell about which it is actually known that it belongs to the surface will be called "good." As the second contender for belonging to the surface, we choose the cell for which the angle of rotation of a good cell in the direction of the outer normal of a good cell before the coincidence will be minimal.

7. CONCLUSIONS

In the article, an algorithm for eliminating self-intersections of an unstructured surface computational mesh with triangular cells was considered. The main goal of this work was to determine the simplest possible sequence of actions that allows to reliably and fast eliminate mesh self-intersections in the process of calculating ice buildup on the surface of a streamlined body. The speed of this algorithm is of critical importance, since the presence of self-intersections of the computational mesh makes it impossible to continue the calculation, therefore, it is necessary to eliminate such anomalies as often as possible. The proposed algorithm contains simple operations for determining the intersections of geometric primitives, subdividing the mesh cells and identifying the actual new surface on which the liquid film flows. The use of this algorithm makes it possible to prevent accidental termination of icing calculations due to the occurrence of self-intersections on meshes with complex geometry that are formed during a long simulation time.

Figures 6 and 7 show an example of the occurrence of mesh self-intersection during the calculation and the result of removing the internal cavity using the proposed algorithm.

FUNDING

The work was carried out at the JSCC RAS as part of the government assignment (topic FNEF-2022-0016). Supercomputer MVS-10P was used in research.

REFERENCES

1. T. Myers, "Extension to the Messinger model for aircraft icing," AIAA J. **39**, 211–218 (2001).
2. P. Farzaneh and G. Bouchard, "Modeling a water flow on an icing surface," in *Proceedings of the International Workshop on Atmospheric Icing of Structures IWAIS XI, Montréal, 2005*.
3. W. Dong, J. Zhu, and X. Min, "Calculation of the heat transfer and temperature on the aircraft anti-icing surface," in *Proceedings of the 27th International Congress of the Aeronautical Sciences, 2010*.
4. H. Beaugendre, "A PDE-based approach to in-flight ice accretion," PhD Thesis (Dep. of Mech. Eng., McGill Univ., Montréal, Québec, 2003).
5. Y. Bourgault, H. Beaugendre, and W. Habashi, "Development of a shallow-water icing model in FENSAP-ICE," J. Aircraft **37**, 640–646 (2000).
6. W. Wright, P. Struck, T. Bartkus, and G. Addy, "Recent advances in the LEWICE icing model," SAE Int. Technical Paper (2015).
7. E. Beld, "Droplet impingement and film layer modeling as a basis for aircraft icing simulations in OpenFOAM," Internship Report (Eng. Fluid Dynamics Dep., Univ. Twente, 2013).
8. R. Domingos and S. Silva, "3D computational methodology for bleed air ice protection system parametric analysis," SAE Int. Technical Paper (SAE, 2015).
9. A. A. Aksenov, P. M. Byvaltsev, S. V. Zhluktov, K. E. Sorokin, A. A. Babulin, and V. I. Shevyakov, "Numerical simulation of ice accretion on airplane surface," AIP Conf. Proc. (2019).
10. S. Bourgault-Côté, K. Hasanzadeh, P. Lavoie, and E. Laurendeau, "Multi-layer icing methodologies for conservative ice growth," in *Proceedings of 7th European Conference for Aeronautics and Aerospace Sciences EUCASS, 2017*.
11. G. Fortin, A. Ilinca, J.-L. Laforte, and V. Brandi, "New roughness computation method and geometric accretion model for airfoil icing," J. Aircraft **41**, 119–1127 (2004).

12. D. Thompson, X. Tong, Q. Arnoldus, E. Collins, D. McLaurin, and E. Luke, "Discrete surface evolution and mesh deformation for aircraft icing applications," in *Proceedings of 5th AIAA Atmospheric and Space Environments Conference, 2013*.
13. X. Tong, D. Thompson, Q. Arnoldus, E. Collins, and E. Luke, "Three-dimensional surface evolution and mesh deformation for aircraft icing applications," J. Aircraft **54**, 1–17 (2016).
14. J. Charton, S. Baek, and Y. Kim, "Mesh repairing using topology graphs," J. Comput. Des. Eng. **8**, 251–267 (2021).
15. W. Jung, H. Shin, and B. K. Choi, "Self-intersection removal in triangular mesh offsettings," CAD J. **1**, 477–484 (2004).
16. V. Skorkovská, I. Kolingerová, and B. Benes, "A simple and robust approach to computation of meshes intersection," in *Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications VISIGRAPP, 2018*.