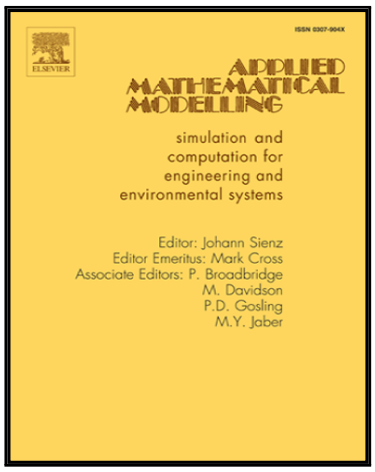


Improved Boundary Constrained Tetrahedral Mesh Generation by Shell Transformation

Jianjun Chen , Jianjing Zheng , Yao Zheng , Hang Si , Oubay Hassan , Kenneth Morgan

PII: S0307-904X(17)30449-3
DOI: 10.1016/j.apm.2017.07.011
Reference: APM 11861

To appear in: *Applied Mathematical Modelling*

Received date: 10 December 2015
Revised date: 21 June 2017
Accepted date: 3 July 2017

Please cite this article as: Jianjun Chen , Jianjing Zheng , Yao Zheng , Hang Si , Oubay Hassan , Kenneth Morgan , Improved Boundary Constrained Tetrahedral Mesh Generation by Shell Transformation, *Applied Mathematical Modelling* (2017), doi: 10.1016/j.apm.2017.07.011

Highlights

- A new flip is designed to reduce the usage of Steiner points in boundary recovery.
- Shell transformation searches for a local optimal mesh among more possibilities.
- The recursive scheme of shell transformation can perform flips on a large element set.
- Meshing examples for surface inputs mainly composed of stretched triangles are tested.
- Difficult boundary constrained meshing tasks in industrial applications are demonstrated.

Improved Boundary Constrained Tetrahedral Mesh Generation by Shell Transformation

Jianjun Chen[a,b*], Jianjing Zheng[a], Yao Zheng[a], Hang Si[c], Oubay Hassan[b], Kenneth Morgan[b]

[a] *Center for Engineering and Scientific Computation, and School of Aeronautics and Astronautics, Zhejiang University, Hangzhou 310027, China*

[b] *Zienkiewicz Centre for Computational Engineering, College of Engineering, Swansea University, Swansea SA2 8PP, Wales, U.K.*

[c] *Weierstrass Institute for Applied Analysis and Stochastics, Mohrenstrasse 39, 10117 Berlin, Germany*

SUMMARY

An excessive number of Steiner points may be inserted during the process of boundary recovery for constrained tetrahedral mesh generation, and these Steiner points are harmful in some circumstances. In this study, a new flip named *shell transformation* is proposed to reduce the usage of Steiner points in boundary recovery and thus to improve the performance of boundary recovery in terms of robustness, efficiency and element quality. Shell transformation searches for a local optimal mesh among multiple choices. Meanwhile, its recursive callings can perform flips on a much larger element set than a single flip, thereby leading the way to a better local optimum solution. By employing shell transformation properly, a mesh that intersects predefined constraints intensively can be transformed to another one with much fewer intersections, thus remarkably reducing the occasions of Steiner point insertion. Besides, shell transformation can be used to remove existing Steiner points by flipping the mesh aggressively. Meshing examples for various industrial applications and surface inputs mainly composed of stretched triangles are presented to illustrate how the improved algorithm works on difficult boundary constrained meshing tasks.

KEY WORDS: mesh generation; boundary recovery; shell transformation; Delaunay triangulation; Steiner points; tetrahedral meshes

1. INTRODUCTION

1.1 Reducing the usage of Steiner points in boundary recovery: why?

The Delaunay criterion provides a reasonable method to triangulate a given point set. However, boundary constraints may be lost in the resulting mesh, and either *conforming* or *constrained* methods are required to recover the lost constraints. For the conforming recovery method, Steiner points are inserted on the constraints and are not removed in the resulting meshes; thus, some of the lost constraints are recovered as concatenations of sub-constraints. For the constrained recovery method, the constraints are the same as the prescribed ones, and no Steiner points can remain on the constraints.

There is no guarantee to recover an edge or face in a tetrahedral mesh without adding Steiner points **[1]**. The typical failing examples are Schönhardt polyhedron **[2]** and Chazelle polyhedron **[3]**, which can only be tetrahedralized by adding Steiner points. Therefore, a robust three-dimensional boundary recovery algorithm must contain a *main procedure* that

* Corresponding author. E-mail: chenjj@zju.edu.cn

considers how to insert Steiner points **[4-19]**. For instance, Weatherill and Hassan **[4]** presented an algorithm for constructing 3D conforming triangulations, with Steiner points inserted on surface boundaries. Later, George *et al.* **[5]** and Du and Wang **[6]** presented a very similar point-splitting idea to attempt to remove all Steiner points from surface boundaries, which is successful in a large percentage (but < 100%) of application instances.

In **[15]**, we presented a boundary recovery algorithm that first inserts Steiner points at intersection positions between lost boundary constraints and the tetrahedral mesh to achieve a conforming recovery, and then removes these points from the surface to achieve the final constrained recovery. In the appendix of this paper, we provide the *theoretical* proofs to explain why this algorithm could output a constrained recovery result by calling a finite number of local operations on the tetrahedral mesh. Nevertheless, these proofs do not consider the round-off errors due to floating point numbers. Thus, the robustness of the algorithm presented in [15] could be challenged in the real world. It was observed that this algorithm likely fails when an excessive number of Steiner points are required during the boundary recovery procedure. This undesirable result occurs when the input surface contains a certain number of elements having high aspect ratios. In this circumstance, Steiner points are harmful to robustness and efficiency of boundary recovery and element quality.

(1) *Robustness*. Predicates such as those proposed by Shewchuk **[20]** can enhance the robustness of boundary recovery remarkably. However, the positions of Steiner points stored with floating-point numbers are essentially inaccurate due to round-off errors. These errors can accumulate if an excessive number of Steiner points are inserted. Predicates with these positions as inputs may return an undesirable value and collapse the entire boundary recovery procedure.

(2) *Efficiency*. For each Steiner point, massive time-consuming computations accompany with its creation, movement and suppression. Thus, the timing cost of a boundary recovery procedure is roughly proportional to the number of Steiner points.

(3) *Element quality*. Steiner points destroy local mesh size specifications and introduce elements having volumes close to zero. Various schemes have been proposed to improve the local mesh quality **[21]**, but these schemes may fail when many bad elements cluster in a local region where many Steiner points are located. This case usually happens near bad surface triangles, and the situation becomes worse under the combined influence of Steiner points and bad boundaries.

In addition, the above issues have the *locality* nature: one stretched element or small angle in the surface may introduce many Steiner points in its neighborhood; if several stretched elements and/or small angles are adjacent to each other, an excessive number of Steiner points may be inserted locally. Therefore, although the input surfaces in practical applications are mainly composed of well-shaped triangles, the above issues may appear occasionally if undesirable geometry features are neighbored with each other. This sort of local imperfection may exist due to many reasons, for instance, when the geometry itself contains undesirable features, or when the mesh gradation is out of control locally. In parallel mesh generation **[22-24]**, the domain decomposition approach may introduce undesirable artificial features on the inter-domain interfaces. In hybrid mesh generation for viscous simulations **[25-27]**, the boundary layer mesher may introduce low-quality faces that are parts of the inputs for the tetrahedral mesher. In simulations of moving boundary problems **[28-30]**, mesh faces were stretched in the mesh movement process, and some of them may appear in the boundary of the hole to be remeshed. To tackle the issue of minimizing the usage of Steiner points during boundary recovery is undoubtedly beneficial for these applications.

1.2 The role of mesh flip based schemes on boundary recovery

It is NP complete to predict whether a polyhedron can be tetrahedralized without adding Steiner points **[1]**, and the lower bound of the number of Steiner points is shown to be quadratic **[3]**. Due to these theoretical difficulties, many heuristic schemes are employed to reduce the number of Steiner points **[4-19]**. These schemes can be classified into *the preprocessing scheme* and *the postprocessing scheme*. The preprocessing schemes improve mesh topologies to reduce intersections between lost boundary constraints and mesh entities, before Steiner point insertion, while the postprocessing schemes suppress Steiner points directly after Steiner point insertion. For the same input, the numbers of Steiner points inserted by different boundary recovery algorithms may vary wildly.

Here, the preprocessing scheme must be highlighted. It produces a topologically improved mesh for the main procedure. The robustness and efficiency of the main procedure are mainly determined by the *quality* of this mesh. In previous studies, tremendous efforts have been made to tackle the main procedure. With respect to the preprocessing scheme, most of boundary recovery algorithms rely on a simple procedure that iteratively conducts the basic flips, i.e., 2-3, 3-2 and 4-4 flips (the numbers in these names denote the number of tetrahedra removed and created by the flips, respectively, see Figure 1a and 1b) **[31, 32]**. Nevertheless, one single flip calling may fail because it only involves a small number of elements. In the context of mesh improvement, Joe suggested improving the performance of flips by composing multiple basic flips **[31]**. In fact, Joe identified nine combinational operations; however, as pointed out by Shewchuk **[33]**, the most elaborate combinational operations can be expressed as one or two *edge removal* operations **[33, 34]**. Later, Liu and Baida **[10]** suggested adopting basic flips *recursively* for boundary recovery purposes. Although it was reported that their algorithm could perform robustly for some surface inputs described by highly stretched triangles, the numbers of inserted Steiner points were far more than minimally required.

Differently, Liu *et al.* proposed to resolve the boundary recovery problem by using a small polyhedron reconnection (SPR) routine **[14]**. Given the outskirt of a polyhedron, the SPR routine attempts to fill in the polygon with an *optimal* tetrahedral mesh by searching and comparing all the possible mesh configurations. In **[15, 35]**, the SPR based local transformation scheme is speeded up to handle a much bigger polyhedron than those managed by a single flip. Thus, it was reported that the SPR based boundary recovery algorithm could achieve better results than their flips based counterparts **[14, 15]**. The main issue of the SPR algorithm is its computing complexity, since the problem of meshing an empty polyhedron is NP hard. Thus, the size of the element set manageable by a single SPR calling is usually limited. In **[14, 15]**, it was recommended to limit the number of surface triangles of the polyhedron input to the SPR routine below 40.

1.3 Our contributions

In this study, a new flip named *shell transformation*[†] is proposed, which could be considered as an extension of the existing *edge removal* operation **[33, 35]**. The classic edge removal operation inputs a set of elements meeting at one edge, and attempts to remove this edge completely by triangulating the skirt polygon (see Figure 2, where the skirt polygon refers to the polygon $p_1p_2p_3p_4p_5p_6$). However, it is not uncommon that edge removal could not provide a covering mesh that better fits in the application purpose than the old one and thus fails to

[†]In the literature, a *shell* usually refers to a set of elements that meet at one edge, and the edge is referred to as *the supporting edge*, and the faces adjacent to this edge are referrred to as *supporting faces*. In this study, a shell refers to a polyhedron that can be filled up with a mesh composed of a set of elements that meet at one edge, and the mesh is called a *covering mesh* of the shell. Please keep the difference between the two definitions in mind while reading this article.

remove *e*. In some cases, the one shown in the bottom of Figure 2 might be a better option, where a *core* referring to the unmeshed part of the skirt polygon exists in the resulting mesh. In this case, we say the shell of *e* is partially reduced[‡]. Evidently, the remaining supporting faces must be removed to reduce the shell further. We know a supporting face *f* is bounded by *e* and two other edges (referred to as *link edges* hereafter). Obviously, if one of the *link edges* that bound *f* is removed, *f* will be removed accordingly. For the case shown in the bottom of Figure 2, *f* could be the face abp_5, and the link edge could be ap_5 or bp_5. If ap_5 is picked up for removal, the above transformation is called again to reduce the shell of ap_5. If the reduced shell of ap_5 does not contain the face abp_5 and any new supporting faces sharing *ab*, the shell of *ab* is reduced as well; otherwise, a process that attempts to remove the supporting faces around ap_5 is repeated.

Compared with Joe's combinational flips **[31]** that only considers the combination of a limited number of basic flips, the proposed scheme combines shell transformations to reduce neighbouring shells in a recursive manner. In this aspect, it is more general and more aggressive than Joe's approach. In real applications, hundreds of shells and thousands of elements could even be involved in one single calling of the recursive shell transformation routine, thereby leading the way to a better local optimum solution.

Based on shell transformation and its recursive scheme, we propose a preprocessing scheme and a postprocessing scheme for boundary recovery. Because the main procedure of boundary recovery inserts Steiner points in the intersections of lost boundary entities and mesh entities **[13, 15]**, the preprocessing scheme attempts to minimize the number of these intersections. The postprocessing scheme is a Steiner point suppression procedure, which provides a second chance to reduce the number of Steiner points.

We will choose surface inputs composed of many stretched elements to challenge the proposed boundary recovery algorithm and compare the resulting performance data with those of a commercial code and published results **[5, 10]**. The comparison reveals that the proposed algorithm not only achieves valid boundary constrained tetrahedral meshes for those failing examples of other algorithms, but also inserts much fewer Steiner points for those examples that other algorithms can also manage. Besides, the applicability of the developed algorithm for real simulations is demonstrated by various mesh generation tasks of computational aerodynamics simulations. In these tasks, the proposed preprocessing scheme recovers all boundaries without any Steiner points at negligible time costs.

The recursive flips idea is also incorporated in an open-source tetrahedral mesher named TetGen (version 1.5 or above) and introduced in **[19]** briefly (named *n*-to-*m* flips). Basically, an *n*-to-*m* flip reduces a shell in an arbitrary manner, while some quality functions are defined in our algorithm to choose the optimal outcome among multiple choices. Meanwhile, the proposed strategy of using the recursive flips is more aggressive than that adopted in TetGen. Consequently, it was observed that the proposed algorithm could recover more boundary constraints when Steiner points are not allowed. Nevertheless, when Steiner points are allowed, the constrained recovery results of the proposed algorithm and TetGen are comparable in terms of the number of finally survival Steiner points. Further suppression of these remaining Steiner points might be difficult since indecomposable polyhedra are observed around these points, as reported in **[36, 37]**.

The remainder of this article is organized as follows. Section 2 illustrates the general concept and implementation of recursive shell transformations. Section 3 details how to

[‡]Here, the *degree* of a covering mesh refers to the number of elements that share the supporting edge in this mesh. A shell is *reduced* if the degree of the new covering mesh is becoming smaller than that of the old mesh. In particular, if the degree of the new mesh becomes zero, the shell is *completely reduced*, and the new mesh is a *completely reduced mesh*; otherwise, the shell is *partially reduced*, and the new mesh is a *partially reduced mesh*.

reduce a shell *optimally* in the context of boundary recovery. Section 4 addresses the application of the proposed flips in a boundary constrained meshing algorithm. Section 5 provides various examples of numerical experiments demonstrating the effectiveness and efficiency of the improved algorithm. Section 6 concludes with outcomes of the study.

2. RECURSIVE SHELL TRANSFORMATIONS

2.1 An illustrative example

To help readers better understand the above recursive scheme, Figure 3 illustrates how the proposed scheme of recursive flips works on a local mesh composed of two shells (see Figure 3a), aimed at removing the edge *ab* from the mesh. Firstly, shell transformation is called on the shell of *ab*. Since the shell cannot be completely reduced, *ab* still exists in the output mesh (see Figure 3b). Nevertheless, the degree of the shell is reduced from 5 to 4. To reduce the shell further, the link edge *bh* is picked and shell transformation is called on the shell of *bh* and reduces this shell completely. Besides, the degree of the shell of *ab* is reduced from 4 to 3 after this step (see Figure 3c). Finally, shell transformation is called on the update shell of *ab* to remove *ab* by a single 3-2 flip (see Figure 3d. Note that the 3-2 flip is a special case of shell transformation).

2.2 A general implementation

Algorithm 1 presents a general implementation of the recursive shell transformations routine. Given an edge *e*, the calling **recursiveST(***e*, ∅, 0, $l_{\max}$**)** attempts to remove *e*, where $l_{\max}$ limits the maximally allowed recursive level. Given a face *f* and one of its boundary edges *e*, the calling **recursiveST(***e*, *f*, 0, $l_{\max}$**)** attempts to remove *f*.

Note that Algorithm 1 expands an *edge tree*, where the input edge *e* is the *root* of this tree, and those link edges (*e′*) inputted for further recursions are *children* of *e*. In this fashion, the tree is expanded recursively. If a tree node v_1 is the *ancestor* of another tree node v_2, we say the edge corresponding to v_1 is the *ancestor edge* of the edge corresponding to v_2. By this definition, the input edge *e* is the *ancestor edge* of all other edges.

Two external routines are employed in Algorithm 1, named **shellTransformation** and **pickRecursiveLinkEdge**, respectively. The routine **shellTransformation** attempts to reduce a shell and its implementation will be detailed in Section 3. Given a supporting face *f* in the shell of *e*, the routine **pickRecursiveLinkEdge** checks whether a further recursion is necessary. If yes, the routine returns a link edge between two possible candidates. To filter inefficient recursions, the implementation of this routine has been improved through the following guidelines:

(1) Do not return a boundary edge.
(2) If the tetrahedra sharing *f* overlap the shells of the ancestor edges of *e*, return nothing.
(3) Return a *reflex edge* only. In Figure 4, the faces ap_2p_3 and ap_3p_4 form a *reflex angle* if viewed from the point *b*; correspondingly, ap_3 is called a *reflex edge* of the face abp_3.

3. SINGLE CALLING OF SHELL TRANSFORMATIONS

3.1 A general implementation

3.1.1 Introduction and terms. The routine **shellTransformation** employed in Algorithm 1 attempts to reduce a shell. Its key step is the *optimal* triangulation of the skirt polygon. In

[33], Shewchuk presented a dynamic programming algorithm **[38]** to triangulate the skirt polygon *completely* and optimally. Nevertheless, the algorithm developed in this study finds a *partial* triangulation that corresponds to an *optimal* covering mesh.

Algorithm 1. The routine of recursive shell transformation

recursiveST(e, f, l, $l_{\max}$**)**
Inputs:
the supporting edge, denoted e
a face containing e that the routine attempts to remove, denoted f
the recursive level with an initial value of zero, denoted l
the maximally allowed recursive level, denoted $l_{\max}$
Variables:
the ending nodes of an edge, denoted $a(\cdot)$ and $b(\cdot)$
the skirt polygon of the shell of an edge, denoted $\mathrm{P}(\cdot)$
the set of elements containing an edge, denoted $\mathrm{S}(\cdot)$
a set of link faces contained in $\mathrm{S}(\cdot)$, denoted $\mathrm{F}(\cdot) = \{ f_1', f_2', \cdots, f_m'\}$, where $m = |\mathrm{F}(\cdot)|$

1. **if** $|\mathrm{S}(e)| <= 0$ **or** (f != ∅ **and** $f \notin \mathrm{F}(\mathrm{S}(e))$)
2. **return** success
3. **shellTransformation**($a(e)$, $b(e)$, f, $\mathrm{P}(e)$, $\mathrm{S}(e)$)
4. **if** $|\mathrm{S}(e)| <= 0$ **or** (f != ∅ **and** $f \notin \mathrm{F}(\mathrm{S}(e))$)
5. **return** success
6. **if** $l >= l_{\max}$ /* the recursive level is limited under $l_{\max}$. */
7. **return** fail
8. $m = |\mathrm{S}(e)|$ **/*** record the size of the shell $\mathrm{S}(e)$ ***/**
9. **for** $i = 1$ **to** m
10. e' = **pickRecursiveLinkEdge**(f_i') /* filters are set to avoid inefficient recursions */
11. **if** e' != ∅
12. **recursiveST(**e', f_i', l+1, $l_{\max}$**)** /* recursive calling */
13. **if** $|\mathrm{S}(e)| < m$ /* $\mathrm{S}(e)$ is reduced as well */
14. return **recursiveST(**e, f, l, $l_{\max}$**)** /* recursive calling */
15. **return** fail

Let the skirt polygon be defined by a set of consecutive mesh nodes

$$\mathrm{P} = \{p_1, \cdots p_m, p_{m+1} = p_1\} \ .$$

For each node $p_i (1 \le i \le m)$, p_{i+m} is its alias. $\mathrm{R}_{i,j}$ defines a *ring* of edges whose ending nodes are

$$\mathrm{P}_{i,j} = \begin{cases} \{p_i, p_{i+1}, \cdots p_j\} & \text{if } 1 \le i < j \le m \\ \{p_i, p_{i+1}, \cdots, p_{j+m}\} & \text{if } 1 \le j < i \le m \end{cases}.$$

Assuming $\mathrm{T}_{i,j}$ is the triangulation of $\mathrm{R}_{i,j}$, the part of covering mesh dual to $\mathrm{T}_{i,j}$ is defined by the set of elements

$$\mathrm{K}_{i,j} = \{\mathrm{conv}(t,a), \mathrm{conv}(t,b) \mid \forall t \in \mathrm{T}_{i,j}\}.$$

where $\mathrm{conv}(\cdot)$ refers to a tetrahedron formed by a face and a node.

Figure 5 depicts a general case of the partial triangulation of a skirt polygon, in which the core is defined by the set of mesh nodes

$$\mathrm{P}_c = \{p_{c_1}, \cdots p_{c_n}, p_{c_{n+1}} = p_{c_1}\},$$

and the triangulated part of the skirt polygon is depicted by

$$\mathrm{T}=\{\mathrm{T}_{c_j,c_{j+1}} \mid j=1,2,\cdots,n\}.$$

Consequently, the covering mesh that corresponds to this partial triangulation is

$$\begin{aligned}&\mathrm{K}=\mathrm{K}_t\cup\mathrm{K}_c\\&\mathrm{K}_t=\bigcup_{j=1}^{n}\mathrm{K}_{c_j,c_{j+1}}\\&\mathrm{K}_c=\{\mathrm{tetr}(p_{c_j},p_{c_{j+1}},a,b)\mid j=1,2,\cdots,n\}\end{aligned}\quad,\qquad(1)$$

where K_t and K_c refer to the sets of tetrahedra covering the triangulated region and the core region of the skirt polygon, respectively, and $\mathrm{tetr}(\cdot)$ refers to the tetrahedral element formed by four specified nodes.

Let $Q(\mathrm{K})$ be the *main* quality index of a covering mesh K. $Q(\mathrm{K})>0$ for a valid covering mesh. $Q(\mathrm{K})$ is larger, the mesh is better. By defining $Q(\mathrm{K})$ properly (see Section 3.3), we set up our problem to be a problem having optimal substructures such that an optimal solution can be constructed efficiently from optimal solutions of subproblems. Based on the partitioning of the covering mesh defined in Equation 1, shell transformation can thus be implemented as a routine including the following steps:

(1) Compute optimal solutions of $\mathrm{K}_{i,j}\ (1\le i,j\le m \text{ and } i\ne j)$ (denoted by $\mathrm{K}_{i,j}^{opt}$).
(2) Enumerate all *valid* partial triangulations, and pick up the one with maximal covering mesh quality.

3.1.2 Computing optimal solutions of $\mathrm{K}_{i,j}$. We define a matrix $\mathbf{M}_q$ to record the quality of $\mathrm{K}_{i,j}^{opt}$ (denoted by $\mathbf{M}_q(i,j)$). $\mathbf{M}_q(l,l+1)=\infty$ when $1\le l<m$ and $\mathbf{M}_q(m,1)=\infty$. Selecting a node $p_k\in\mathrm{P}_{i,j}$ other than p_i and p_j, the triangulation of $\mathrm{R}_{i,j}$ includes three parts (see Figure 6):

$$\mathrm{T}_{i,j}=\mathrm{T}_{i.k}\cup\mathrm{T}_{k,j}\cup\mathrm{T}_{i,k,j},\ \text{where}\ \mathrm{T}_{i,k,j}=\{\Delta p_i p_k p_j\}.$$

Evidently, $\mathrm{K}_{i,j}$ is a function of k, denoted by

$$\mathrm{K}_{i,j}(k)=\mathrm{K}_{i.k}\cup\mathrm{K}_{k,j}\cup\mathrm{K}_{i,k,j},\ \text{where}\ \mathrm{K}_{i,k,j}=\{\mathrm{conv}(\Delta p_i p_k p_j,\mathrm{a}),\mathrm{conv}(\Delta p_i p_k p_j,\mathrm{b})\}.$$

We define a new function f_Q to compute the quality of a mesh K by its subsets:

$$Q(\mathrm{K})=f_Q(n_K,Q(\mathrm{K}_1),Q(\mathrm{K}_2),\cdots,Q(\mathrm{K}_{n_K})).\qquad(2)$$

Here, $\mathrm{K}_i(i=1,2,\cdots,n_K)$ are nonoverlapping subsets of K and

$$\bigcup_{i=1}^{n_K}\mathrm{K}_i=\mathrm{K}.$$

Thus,

$$\mathbf{M}_q(i,j)=\max_{p_k\in\mathrm{P}_{i,j},p_k\ne p_i,p_j} f_Q(3,\mathbf{M}_q(i,k),\mathbf{M}_q(k,j),Q(\mathrm{K}_{i,k,j})).$$

Algorithm 2 presents a routine that fills in all useful elements of $\mathbf{M}_q$, which could be in either side of the main diagonal of $\mathbf{M}_q$. Besides, another matrix $\mathbf{M}_k$ is defined to record the values of k that maximize $Q(\mathrm{K}_{i,j}(k))$, and one diagonal of $\mathbf{M}_q$ and $\mathbf{M}_k$ is computed at a time in the increasing order of the size of $\mathrm{R}_{i,j}$ (i.e., number of vertices).

3.1.3 Enumerating all valid partial triangulation cases. Firstly, we introduce the concept of *triangulation graph*. It is a directed graph defined on a polygon that is bounded by a set of nodes $\mathrm{P}=\{p_1,p_2,\cdots p_m\}$. A *graph node* corresponds to a polygon node, and a graph edge $<p_i,p_j>$ exists if $\mathbf{M}_q(i,j)>0$, illustrating that a valid triangulations exists for the ring $\mathrm{R}_{i,j}$. Evidently, the core of the skirt polygon corresponds to a simple cycle of the

triangulation graph. The remaining issue is how to evaluate the quality of a simple cycle such that we could find an *optimal* one among all possible choices.

For a simple cycle denoted by $\mathrm{P}_c = \{p_{c_1}, \cdots p_{c_n}, p_{c_{n+1}} = p_{c_1}\}$, its dual covering mesh K could be reconstructed by Equation 1. In this study, a quality vector including three quality indices of K is introduced to evaluate the quality of P_c.

The first quality index is $Q(\mathrm{K})$, which is computed by

$$\begin{aligned} Q(\mathrm{K}) &= f_Q(2, Q(\mathrm{K}_c), Q(\mathrm{K}_t^{opt})) \\ &= f_Q(n{+}1, Q(\mathrm{K}_c), Q(\mathrm{K}_{c_1,c_2}^{opt}), \cdots, Q(\mathrm{K}_{c_n,c_1}^{opt})) \quad . \\ &= f_Q(n{+}1, Q(\mathrm{K}_c), \mathbf{M}_q(c_1, c_2), \cdots, \mathbf{M}_q(c_n, c_1)) \end{aligned} \tag{3}$$

In recursive shell transformations, a major goal of shell transformation is to remove a face adjacent to the parent of the present supporting edge. Denoting this face by *f*, the second quality index is

$$Q'(\mathrm{K}) = \begin{cases} 1 & \mathrm{K} \text{ does not include } f \\ 0 & \mathrm{K} \text{ includes } f \end{cases} .$$

The third quality index reflects the preference over a covering mesh with a smaller degree value, computed by

$$Q''(\mathrm{K}) = N_{big} - n,$$

where n is the size of the simple cycle (i.e., number of vertices), and N_{big} is a big enough value such that $Q''(\mathrm{K}) > 0$ always.

The priority of $Q(\mathrm{K})$, $Q'(\mathrm{K})$ and $Q''(\mathrm{K})$ in the quality vector may vary. To be general, the quality vector is denoted by

$$\mathbf{Q}_v(\mathrm{P}_c) = \mathbf{V}(Q(\mathrm{K}), Q'(\mathrm{K}), Q''(\mathrm{K})) . \tag{4}$$

To choose an optimal simple cycle, the quality vectors of different cycles are compared *lexicographically*.

Algorithm 2. Filling in $\mathbf{M}_q$ and $\mathbf{M}_k$

fillInMatrices(*a*, *b*, P, $\mathbf{M}_q$, $\mathbf{M}_k$)
Inputs:
a and *b*: the ending nodes of the supporting edge
P = {p_1, p_2, …, p_m}: the skirt polygon of a shell

```
1.  for d = 2 to m - 1
2.      for i = 1 to m
3.          j′ = i + d
4.          j = j′ > m ? j′ - m : j′
5.          for k′ = i + 1 to j′ – 1
6.              k = k′ > m ? k′ - m : k′
7.              q = f_Q(3, M_q(i, k), M_q(k, j), Q(K_{i,j}(k))))
8.              if k′ = i + 1 or q > M_q(i, j)
9.                  M_q(i, j) = q
10.                 M_k(i, j) = k
```

3.1.4 The shell transformation routine. Algorithm 3 presents the shell transformation routine. In general, the optimal simple cycle is not empty and its size (n) is smaller than the size of

the original skirt polygon (m). Nevertheless, two special cases may occur:

(1) If $n = m$, the core is the entire skirt polygon and no shell transformation is performed.

(2) If $n = 0$, the core is empty and the complete shell transformation is performed.

3.2 Mesh validity check

In Algorithm 3, the output mesh needs to meet two general validity conditions:

(1) All elements must have positive volumes.

(2) The mesh must not contain supporting faces around any *ancestor edge* of *ab*.

For boundary recovery purposes, additional validity conditions are defined for the output mesh:

(1) **BRC1**. The mesh must not intersect a given boundary edge.

(2) **BRC2**. The mesh must not intersect a given boundary face.

(3) **BRC3**. The mesh must not intersect boundary edges more than the input mesh.

(4) **BRC4**. The mesh must not intersect boundary faces more than the input mesh.

Note that these additional conditions are not mandatory. They are selectively enabled in accordance with the purposes of calling Algorithm 3.

Algorithm 3. A general routine of shell transformation

shellTransformation(*a, b, f,* P, K_{old})

Inputs:

a and *b*: the ending nodes of the supporting edge

f: a face containing *ab* that the routine attempts to remove

P = {$p_1, p_2, \ldots, p_m$}: the skirt polygon of the shell

K_{old}: the old covering mesh of the shell

1. **fillInMatrices**(*a*, *b*, P, $\mathbf{M}_q$, $\mathbf{M}_k$)
2. *G*: the triangulation graph with $\mathbf{M}_q$ as its matrix representation
3. $P_c = \{p_{c_1}, \cdots p_{c_n}, p_{c_{n+1}} = p_{c_1}\}$: an *optimal* simple cycle of *G*
4. K: the covering mesh dual to P_c, see Equation 1
5. **V(**K**)** & **V(**K_{old}**)**: the quality vectors of K & K_{old}, see Equation 4
6. Compare **V(**K**)** & **V(**K_{old}**)** lexicographically.
7. If K is better than K_{old}
8. Remesh the shell by replacing K_{old} with K

3.3 Quality functions

In this study, the $Q(\mathrm{K})$ of a valid mesh is computed by either of the two quality functions defined as below.

Let $q(t)$ be the quality function of a tetrahedron. The first measure of $Q(\mathrm{K})$ is evaluated by the worst element in K :

$$Q(\mathrm{K}) = Q_1(\mathrm{K}) = \begin{cases} -1 & \text{If K is invalid} \\ \min\limits_{t_i \in \mathrm{K}} \{q(t_i)\} & \text{otherwise} \end{cases}. \tag{5}$$

Thus,

$$f(n_K, Q(\mathrm{K}_1), Q(\mathrm{K}_2), \cdots, Q(\mathrm{K}_{n_K})) = \min_{i=1}^{n_K} \{Q(\mathrm{K}_i)\}.$$

The second measure of $Q(\mathrm{K})$ reflects the preference over a covering mesh that intersects the input boundary constraints (denoted by B_c) least, computed by

$$Q(\mathrm{K}) = Q_2(\mathrm{K}) \begin{cases} -1 & \text{If K is in} \\ N_{big} - I(\mathrm{K}, \mathrm{B}_c) & \text{otherw} \end{cases}, \tag{6}$$

where $I(\mathrm{K},\mathrm{B}_c)$ is the number of intersections between K and B_c, and N_{big} is a big enough value such that $N_{\mathrm{big}} - I(\mathrm{K},\mathrm{B}_c) > 0$ always.

Thus,

$$f(n_K, Q(\mathrm{K}_1), Q(\mathrm{K}_2), \cdots, Q(\mathrm{K}_{n_K})) = \begin{cases} \sum_{i=1}^{n_K} Q(\mathrm{K}_i) & Q(\mathrm{K}_i) > 0 \ \forall i = 1, 2, \cdots, n_K \\ -1 & \text{otherwise} \end{cases} \quad (7)$$

An issue here is that the intersections are defined at mesh edges and faces while Equation 7 is computed in an element-wise fashion. A resolution on this issue is presented as below:

(1) Let $I_f(f,\mathrm{B}_c)$ record how many times the interior of a face f is intersected by B_c. The share of $I_e(f,\mathrm{B}_c)$ on either element adjacent to f is $I_f(f,\mathrm{B}_c)/2$.

(2) Let $I_e(e,\mathrm{B}_c)$ record how many times the interior of an edge e is intersected by B_c. The share of $I_e(e,\mathrm{B}_c)$ on any element adjacent to e is $I_e(f,\mathrm{B}_c)/n_e$, where n_e is the total number of elements adjacent to e.

In the above computations, only those intersections defined at interior edges and faces of a shell will be considered since the outskirt of the shell remains unchanged in Algorithm 3. Besides, the n_e values of interior edges are computed as below.

(1) $n_e =$ size of the core for the supporting edge;

(2) $n_e = 3$ for the edges bounding the core (e.g., the edge p_2p_5 in the bottom of Figure 2);

(3) $n_e = 4$ for other interior edges (e.g., the edge p_3p_5 in the bottom of Figure 2).

4. THE IMPROVED CONSTRAINED MESHING APPROACH

4.1 The overall flowchart

Given a surface triangulation that defines the meshing domain, the tetrahedral meshing algorithm takes the following steps to generate a boundary constrained mesh **[15]**:

(1) Input the boundary triangulation.
(2) Construct an initial Delaunay triangulation with the outer box.
(3) Insert boundary points with the Bowyer-Watson algorithm **[39, 40]**.
(4) Recover boundaries with the proposed algorithm.
(5) Remove exterior elements.
(6) Insert field points with the modified Bowyer-Watson algorithm **[41]**.
(7) Improve mesh quality.

The main procedure of the boundary recovery step (Step 4) inserts Steiner points at the intersection positions of lost constraints and mesh entities **[13, 15]**. To reduce the usage of Steiner points in boundary recovery, a preprocessing step and a postprocessing step are placed before and after the main procedure, respectively.

(1) *Preprocessing step*. Recover edges and faces by recursive shell transformations. No mesh points are inserted or removed in this step.
(2) *Postprocessing step*. Suppress Steiner points by recursive shell transformations.

For constrained recovery, the remaining Steiner points after the postprocessing step are relocated to the domain interior **[15]**.

4.2 The preprocessing scheme

4.*2.1 Edge recovery*. A boundary edge is lost because it intersects some mesh edges or faces. If these intersecting entities are removed, the edge is recovered. Based on this idea, Algorithm

4 employs Algorithm 1 to remove entities intersecting a boundary edge individually. The timing performance of Algorithm 4 depends on $l_{\max}$. $l_{\max}$ is larger, more shell transformations are executed; thus, more edges are likely to be recovered. Algorithm 5 employs Algorithm 4 iteratively to recover edges by increasing $l_{\max}$ in each iterative step, and it ends if all edges are recovered, or the mesh quality is not improved in three consecutive iterative steps. In Algorithm 5, the routine **evalMesh** computes the mesh quality index with respect to edge recovery.

Algorithm 4. The edge recovery routine with a predefined maximal recursive level

recvBndEdgesByST(E, $l_{\max}$**)**
Inputs:
a set of lost boundary edges, denoted E = { e_1, e_2, ⋯, e_m}
the maximally allowed recursive level, denoted $l_{\max}$
Variables:
a set of mesh entities that intersect an edge, denoted G(·) = { g_1, g_2, ⋯, g_n}

```
1.  for i = 1 to |E|
2.     findInstEntsOfBE(e_i, &G(e_i)) /* find the set of entities intersecting e_i */
3.     if |G(e_i)| <= 0
4.        continue
5.     for j = 1 to |G(e_i)|
6.        if g_j is an edge
7.           recursiveST(g_j, ∅, 0, l_max)
8.        else if g_j is a face
9.           for k = 1 to 3
10.             if recursiveST(b_k, g_j, 0, l_max) succeeds /* b_k is a boundary edge of g_j */
11.                break
12. updateLostEdges(E)    /* update the list of lost boundary edges */
```

Algorithm 5 is called twice in the edge recovery procedure. The mesh validity condition BRC1 (see Section 3.2) is enabled in the first calling to ensure no new entities intersect the boundary edge that is being recovered. The routine **evalMesh** evaluates a mesh by the number of lost boundary edges. Besides, Equation 5 is adopted to define $Q(\mathrm{K})$, the vector used to evaluate the quality of a simple cycle is computed by

$$\mathbf{Q}_v(\mathrm{P_c}) = \mathbf{V}(Q(\mathrm{K}), Q'(\mathrm{K}), Q''(\mathrm{K}))=[Q'(\mathrm{K}), Q''(\mathrm{K}), Q(\mathrm{K})].$$

However, the validity condition BRC1 cannot prevent new entities from intersecting the boundary edges other than the one that is being recovered. To remove the intersections further, Algorithm 5 is restarted by replacing BRC1 with BRC3 as the mesh validity condition. The routine **evalMesh** evaluates a mesh by the number of intersections between all boundary edges and mesh entities. Besides, Equation 6 is adopted to define $Q(\mathrm{K})$, the vector used to evaluate the quality of a simple cycle is computed by

$$\mathbf{Q}_v(\mathrm{P_c}) = \mathbf{V}(Q(\mathrm{K}), Q'(\mathrm{K}), Q''(\mathrm{K}))=[Q(\mathrm{K}), Q'(\mathrm{K}), Q''(\mathrm{K})].$$

Nevertheless, it is time-consuming to implement BRC3 by computing intersections and counting their numbers in real time. To reduce the timing costs, a hash table is maintained to record the mesh entities intersected by lost boundary edges. In each item of the table, the number of intersections between lost boundary edges and a certain mesh edge or face is kept. The hash table is compact because only a few boundary edges are lost after the first calling of Algorithm 5.

For performance sake, the entities intersected by lost edges are not recorded once they are created; instead, they are recorded in the routine **updateLostEdges** (see Line 12 of Algorithm 4). This strategy may increase the number of intersections temporarily because some intersected entities created by shell transformation callings are not recorded in the table. However, if these entities survive the current calling of Algorithm 4, they will be inserted into the hash table afterwards and possibly removed in the next calling of Algorithm 4.

Note that both callings of Algorithm 5 never destroy recovered edges because the supporting edges in shell transformations are always non-boundary.

Algorithm 5. The edge recovery routine by increasing the maximal recursive level iteratively

recvBndEdgesByST_Wrap(E**)**
Inputs:
a set of lost boundary edges, denoted E
Variables:
the maximally allowed recursive level, denoted $l_{\max}$; initial value is 0
the number of consecutive unsuccessful iterations, denoted m_f; initial value is 0

1. **while** (|E| > 0 **and** m_f < 3)
2. **evalMesh(**EDG_RECV**,** &q_b**)**
3. **recvBndEdgesByST(**E, $l_{\max}$)
4. **evalMesh(**EDG_RECV**,** &q_a**)**
5. $q_a > q_b$ **?** $m_f = 0$ **:** m_f += 1
6. $l_{\max}$ += 1

4.2.2 Face recovery. A boundary face is lost because its boundary and/or its interior intersect mesh edges. The mesh edges intersecting face boundaries have been tackled in edge recovery; therefore, face recovery only needs to remove mesh edges intersecting face interiors.

The flowchart of face recovery is like edge recovery. It also contains two callings of a routine that removes the intersecting edges of boundary faces by increasing the allowed recursive levels iteratively. However, different validity conditions are enabled in the two callings. The first calling attempts to reduce the numbers of boundary faces whose interiors are intersected by mesh entities, and BRC2 is enabled as the validity condition. The second calling attempts to reduce the intersections in the interiors of boundary faces, and BRC4 is enabled as the validity condition.

4.3 The postprocessing scheme

Steiner points are moved to the domain interior after the main procedure of boundary recovery. For each Steiner point, the proposed scheme employs Algorithm 1 to remove the edges adjacent to this point. The point is removed directly when a mesh configuration shown in Figure 7 is formed (here, *a* is the point for removal). In **[32]**, such a mesh is called a *Christmas tree*.

The attempts of suppressing all surviving Steiner points are repeated a few times until no more benefits are expected.

5. RESULTS

We choose two groups of surfaces as inputs to demonstrate the proposed algorithm. The surfaces in the first group are mainly composed of highly stretched elements. The surfaces in the second group are some inputs for computational aerodynamics simulations.

5.1 Surface inputs composed of highly stretched triangles

The first group includes seven meshes. The B747 aircraft model was downloaded from the mesh database maintained by INRIA **[42]**, and the other six models are available from the repository in *edge.mcs.drexel.edu*. All these examples contain many badly shaped triangles. If no effective strategies are developed to reduce the number of Steiner points, boundary recovery may fail to tackle these examples, or succeed but insert an excessive number of Steiner points.

Figures 8 and 9 present the pictures of the Mohne and B747 meshes, respectively. The pictures of other models could be seen in the literature **[5, 9, 10]**.

5.1.1 Comparison of different algorithms. Table 1 compare the numbers of Steiner points inserted by the proposed boundary recovery algorithm with results published in **[10]** and those by TetGen (version number: 1.5) and a recent version of GHS3D (version number: 4.2-2; referred to as new GHS3D hereafter)[§] **[43]**. The new GHS3D was licensed in September 2010. The test data of both TetGen and the new GHS3D are provided by Dr. Hang Si. Those tests were executed on a MacBook Pro laptop (CPU: 2.8GHz; Memory: 4GB), i.e., the host computer of the license for GHS3D. The tests of the proposed algorithm were executed on a Sony laptop (CPU: 2.3GHz; Memory: 4GB).

The Mohne model used in **[10]** is slightly different with the model used by us. In **[10]**, it was reported that the model contains 2,763 nodes and 5,566 faces. The model used in our tests contains 2,760 nodes and 5,560 faces. The Thepart, Boeing_part and Thru-mazewheel models have exactly same surface data as those reported in **[10]**. For unknown reasons, the new GHS3D failed to tackle the Cami1 and B747 aircraft models. In **[5]**, it was reported that GHS3D can mesh the Cami1 model by inserting 14 Steiner points. For the Thru-mazewheel model, the new GHS3D inserts more Steiner points than its previous version (41 vs. 18).

In most of the tests, the numbers of Steiner points inserted by the proposed algorithm are minimal, apart from the test on the Boeing_part model, where TetGen requires 4 Steiner points for constrained recovery while the proposed algorithm inserts 5 Steiner points. In general, TetGen and the proposed algorithm produce comparable performance data.

In summary, the data listed in Table 1 reveal that the recursive flips are very useful to improve the topology of a mesh and thus benefit the subsequent boundary recovery procedure by reducing the usage of Steiner points. For instance, for 3 of 7 models (Cami1a, Thepart and B747), the proposed algorithm inserts no Steiner points. For the other four inputs, the numbers of inserted Steiner points range from 1 to 5. These numbers are much smaller than those by two versions of GHS3D and Liu's algorithm. For instance, the proposed algorithm inserts 5 Steiner points to mesh the Boeing_part model. However, the algorithm proposed by Liu and Baida **[10]** and the new GHS3D insert 25 and 22 Steiner points, respectively.

Minimizing the usage of Steiner points is helpful for enhancing the robustness of boundary recovery. It was reported that the old GHS3D failed to mesh the Mohne and Boeing_part models **[10]**. Although the reasons that the new GHS3D failed to mesh the Cami1 and B747 aircraft models are unknown, from our experience, vulnerable geometric computations regarding Steiner points may account for these failures.

Table 2 compares the timing performance of the proposed algorithm, TetGen and the new GHS3D. The results show that TetGen runs fastest in these tests, and both our algorithm and TetGen run much faster than GHS3D. By comparison with TetGen, the extra computations for optimal and partial shell transformations do affect the timing data of boundary recovery. An alternative approach would be to use simpler algorithms for initial recovery of missing

[§]It needs to be emphasised that the comparison between our code and GHS3D is not a comparison of the overall performance of two codes. The comparison presented here mainly focuses on the performance difference of two codes in terms of the number of Steiner points inserted in boundary recovery for a set of common models.

edges and faces, and then use partial and optimal shell transformations only for harder recovery of remaining missing edges and faces. Nevertheless, as to be demonstrated in Section 5.2, the timing cost consumed by boundary recovery usually takes a very small fraction of the entire meshing time cost in real applications. We leave the further optimization of efficiency of our algorithms for future works.

5.1.2 Performance of the proposed boundary recovery scheme. The proposed edge recovery scheme is of paramount importance for reducing the usage of Steiner points, which calls Algorithm 4 twice to reduce the number of lost edges (n_1) and the number of intersections (n_2) between lost edges and mesh entities, respectively. A lost boundary edge may intersect mesh entities more than once. The intersection number of a boundary edge records how many times this edge is intersected. A new index n_3 is introduced to record the maximal intersection numbers of all boundary edges. Taking the Cami1 model as an example, Figure 10 highlights the lost edges in different phases of the edge recovery process. Table 3 illustrates how n_1, n_2 and n_3 vary against the allowed recursive level ($l_{\max}$) that increases continuously in the main loop of Algorithm 5. The values when $l_{\max}$ = -1 correspond to the status before calling Algorithm 5.

In the first calling of Algorithm 5, 203 of 243 lost edges are recovered when $l_{\max}$ increases from -1 to 2. n_2 decreases by one order, from 1,505 to 145. n_3 decreases from 41 to 21, not so remarkably. With $l_{\max}$ increasing to 9, n_1 decreases stably to 13, and n_2 and n_3 also decrease to 39 and 12, respectively. However, the first calling of Algorithm 5 may create new entities that intersect the boundary edges other than the one is being recovered. Therefore, n_2 is observed to go up temporarily when $l_{\max}$ increases from 4 to 5. n_3 is observed to go up temporarily as well when $l_{\max}$ increases from 5 to 6.

In the second calling of Algorithm 5, a hash table is used to reduce the time-consuming intersection computations. The size of this table, n_4, is introduced in Table 3. n_2 goes up temporarily in the first iterative step because many new mesh entities that intersect lost edges are not recorded in the hash table. After they are recorded, n_2 is reduced stably. After the second calling, only 10 edges are lost, and each edge intersects one mesh entity once.

Finally, the main procedure of boundary recovery **[13, 15]** inserts 10 Steiner points, and only one Steiner point survives the shell transformation based suppression scheme.

5.2 Surface inputs for computational aerodynamics simulations

5.2.1 Sequential mesh generation of the F16 aircraft model. An F16 aircraft model is selected in this test (see Figure 11). Grid sources **[44]** are configured to refine local meshes where small element sizes are necessary. The mesh generation is conducted sequentially on the Sony laptop. Table 4 summarizes the performance data of the mesh generation process. The surface elements with minimal interior angles below 15 degrees are referred to as *thin triangles*, and the dihedral angle between neighboring surface elements are referred to as *small angles* if they are below 15 degrees.

All constraints are recovered by the proposed preprocessing scheme of boundary recovery. The Delaunay meshing procedure consumes about 30 seconds to generate nearly 4 million tetrahedral elements, 0.59 second of which is consumed by the boundary recovery procedure. The mesh improvement procedure **[21]** consumes about 227 seconds, about seven times slower than the Delaunay meshing procedures in this test.

5.2.2 Parallel mesh generation of the F16 aircraft model. The tested parallel mesh generation approach **[23, 24]** includes four main steps:

(1) *Domain decomposition*. The input surface mesh is decomposed into many subdomains by inserting inter-domain surface meshes inside the domain.
(2) *Mesh generation.* The subdomains are distributed and meshed individually in parallel.

(3) *Mesh repartitioning*. The mesh is repartitioned and then redistributed to balance the loads and minimise the communications.
(4) *Quality improvement*. The redistributed submeshes are improved individually.

The test was performed on a Dawning TC5000 Cluster composed of 30 computer nodes. Each node contains two six-core CPUs and the CPU frequency is 2.66GHz. The accessible memory size of each computer node is 24GB. In this test, the F16 aircraft model is selected, and 32 computer cores are used. 749 seconds are consumed totally to output a mesh composed of about 123 million tetrahedra. Table 5 presents the performance data.

Our focus is on Step 2, where subdomains are meshed by the proposed Delaunay mesher. Constrained boundary recovery is required to maintain the conformity of inter-domain meshes. The original surface mesh contains 27 thin triangles and 13 small angles. The domain decomposition step introduces 15 thin triangles and 1,843 small angles. In general, the surface quality of subdomains is much worse than that of the original input. Therefore, the parallel mesh generation challenges the boundary recovery algorithm more than its sequential counterpart. In our experience, the classic boundary recovery algorithm proposed in **[13, 15]** occasionally failed during the process of meshing some subdomains. However, the proposed algorithm behaved very reliable and effective in this test, where all boundary constraints are recovered in the preprocessing step.

The mesh generation step consumes 48.44 seconds, accounting for 6.5% of the total time cost. The boundary recovery only consumes 0.68 seconds averagely. Domain decomposition and quality improvement are two most time-consuming steps, accounting for 46.0% and 34.9% of the total time cost, respectively.

Figure 12 presents a coarser volume mesh of the F16 aircraft model. This mesh is created in parallel on 8 computer cores, containing about ten million tetrahedral elements.

5.2.3 Local remeshing of a wing/pylon/finned-store separation simulation. The last case is a wing/pylon/finned-store separation simulation executed on the Dawning TC5000 Cluster. The main loop of this simulation includes four main steps:

(1) *Flow computation.* Compute the unsteady flow by a finite volume solver.
(2) *Motion computation.* Compute aerodynamic forces and moments based on flow simulation results, with which as input, the positions of moving bodies in the next time step are determined using the six degrees-of-freedom equations of motion.
(3) *Mesh movement*. Move the mesh points to adapt the movement of mesh boundaries.
(4) *Local remeshing*. If mesh movement yields elements with unacceptable quality, the holes are formed by deleting these elements. Next, a new mesh is formed by merging undeleted elements and new elements filled in the holes. Finally, the solution is reconstructed by interpolation.

The computation conditions of this simulation can be found in **[45]**. The input volume mesh contains 1,111,001 tetrahedral elements and 194,754 nodes. Figure 13 shows the cut views of the volume meshes at different time steps (t_s), where Figure 13b and 13c compare the meshes before and after a local remeshing calling when t_s =0.38s. Figure 14 presents the contour plot of the computed pressure distributions during the separation.

The proposed Delaunay mesher is employed in the local remeshing step, where constrained recovery is mandatory to maintain the conformity of new elements and undeleted elements. The boundaries of holes usually contain a few thin triangles and small angles because interior volume elements are deformed in the mesh movement step. In previous studies, the local remeshing step is restarted by adding neighboring elements to expand the holes if the boundary recovery algorithm fails to mesh these holes. In our experiments, this backup scheme has never been employed. In the total 16 remeshing callings, all the boundary recovery processes end successfully in their preprocessing steps. 131,045 elements are generated averagely in each remeshing step. If not counting the computing time of quality

improvement, the Delaunay meshing procedure consumes 1.2 seconds averagely, 0.11 second of which is consumed by the boundary recovery procedure.

6. CONCLUSIONS

Minimizing the usage of Steiner points can improve the robustness and efficiency of boundary recovery and reduce the damage of Steiner points to element quality. In this study, this goal is achieved by a new flip named shell transformation. It searches the optimal mesh of a shell among multiple choices, and its recursive scheme could overcome the limits of a single flip. The boundary recovery algorithm is evidently improved by applying the new flip (and its recursive scheme) in the preprocessing and postprocessing procedures of boundary recovery. Various difficult boundary constrained meshing tasks are tested and the performance data are provided to demonstrate the effectiveness and efficiency of the improved algorithm.

Appendix

In **[15]**, we presented a boundary recovery algorithm that first inserts Steiner points at intersection positions between lost boundary constraints and the tetrahedral mesh to achieve a conforming recovery, and then removes these points from the surface to achieve the final constrained recovery. The problem of constructing a conforming tetrahedralization is much easier than the problem of constructing a constrained tetrahedralization. If all the possible intersection and subdivision cases **[4, 15]** are covered, it is not too hard to see that the conforming tetrahedralization procedures presented in **[4, 11, 15]** can construct the desired tetrahedralization. Interested readers are referred to **[11]** for the proofs. Nevertheless, the problem of constrained boundary recovery is much more difficult. The previous works **[5-6]** provide different levels of algorithmic and theoretical details. In this appendix, we will provide the *theoretical* proofs to explain why the algorithm presented in **[15]** could output a constrained recovery result after calling a finite number of local operations on the tetrahedral mesh. Besides, since these proofs do not consider the round-off errors due to floating point numbers, we will investigate some issues that may challenge the robustness of the algorithm in the real world.

A.1 Terms and definitions

Definition 1 (*visibility*) [6]

Let $\triangle ABC$ be a triangle with O as the centroid, and P be a point shown in Figure A.1. The vertices of $\triangle ABC$ are called *directionally oriented* if the outer normal direction $\overrightarrow{ON}$ is defined as $\overrightarrow{AB} \times \overrightarrow{BC}$. The triangle $\triangle ABC$ is said to be *visible* to P if $\overrightarrow{ON} \cdot \overrightarrow{OP} > 0$.

The visibility of $\triangle ABC$ to P is equivalent to the signed volume of the tetrahedron $ABCP$ being positive, or, the tetrahedron $ABCP$ being *valid*.

Definition 2 (*consistently oriented triangle set*)

Let $< f_1, f_2 >$ be a pair of triangles meeting at one common edge PQ. The vertices of both f_1 and f_2 are directionally oriented. If the direction of PQ in f_1 is opposite to its counterpart in f_2, the pair of triangles is called *consistently oriented*.

Let $\mathcal{F}=\{f_i \mid i=1,2,\cdots,n_f\}$ be a set of connected triangles. If an arbitrary pair of triangles meeting at one edge, denoted by $<f_i, f_j>(1\le i, j\le n_f,\ i\ne j)$, is consistently oriented, $\mathcal{F}$ is called *consistently oriented* as well, as shown in Figure A.2.

Let $r(f_i)(i=1,2,\cdots,n_f)$ be the same triangle as f_i but with opposite orders of vertices. Evidently, if $\mathcal{F}$ is *consistently oriented*, the triangle set

$\mathcal{R}(\mathcal{F})=\{r(f_i) \mid i=1,2,\cdots,n_f\}$

is consistently oriented as well. In brief, $\mathcal{R}(\mathcal{F})$ is called the reverse of $\mathcal{F}$.

Definition 3 (*ball*)

Let P be a point. The set of tetrahedra incident to P is called the *ball* of P, as shown in Figure A.3, denoted by

$\mathcal{B}(P)=\{e_i \mid i=1,2,\cdots,n_e\}$.

Definition 4 (*2-way cutting*) [6]

Let P and $\mathcal{B}(P)$ be defined as above. $\mathcal{B}^1(P)$ and $\mathcal{B}^2(P)$ are two sets of connected tetrahedra. The set $\{\mathcal{B}^1(P),\mathcal{B}^2(P)\}$ defines a *2-way cutting* of $\mathcal{B}(P)$ if

(1) $\mathcal{B}^1(P)\ne\varnothing$ and $\mathcal{B}^2(P)\ne\varnothing$;

(2) $\mathcal{B}^1(P)\subset\mathcal{B}(P)$ and $\mathcal{B}^2(P)\subset\mathcal{B}(P)$;

(3) $\mathcal{B}^1(P)\cap\mathcal{B}^2(P)=\varnothing$;

(4) $\mathcal{B}^1(P)\cup\mathcal{B}^2(P)=\mathcal{B}(P)$.

The set of triangles shared by $\mathcal{B}^1(P)$ and $\mathcal{B}^2(P)$ is referred to as the *poly-triangle separator* of $\mathcal{B}(P)$, denoted by $\mathcal{I}(\mathcal{B}^1,\mathcal{B}^2)$.

Figure A.4 presents a 2-way cutting, where

$\mathcal{B}^1(P)=\{PP_2P_6P_3, PP_3P_6P_5, PP_3P_5P_4, PP_4P_5P_1, PP_1P_5P_2, PP_2P_5P_6\}$,

$\mathcal{B}^2(P)=\{PP_7P_2P_3, PP_7P_3P_4, PP_7P_4P_1, PP_1P_2P_7\}$,

and the poly-triangle separator refers to

$\mathcal{I}(\mathcal{B}^1,\mathcal{B}^2)=\{PP_1P_2, PP_2P_3, PP_3P_4, PP_4P_1\}$.

Definition 5 (*n-way cutting*)

Let P and $\mathcal{B}(P)$ be defined as above. $\mathcal{B}^i(P)\ (i=1,2,\cdots,n,\ n>2)$ is a set of connected tetrahedra. The set $\{\mathcal{B}^i(P) \mid i=1,2,\cdots,n\}$ defines an *n-way cutting* of $\mathcal{B}(P)$ if

(1) $\mathcal{B}^i(P)\ne\varnothing\ (i=1,2,\cdots,n)$;

(2) $\mathcal{B}^i(P)\subset\mathcal{B}(P)\ (i=1,2,\cdots,n)$;

(3) $\mathcal{B}^i(P)\cap\mathcal{B}^j(P)=\varnothing\ (1\le i, j\le n, i\ne j)$;

(4) $\bigcup_{i=1}^{n}\mathcal{B}^i(P)=\mathcal{B}(P)$.

Let $\mathcal{I}(\mathcal{B}^i,\mathcal{B}^j)(i\ne j)$ be the set of triangles shared by $\mathcal{B}^i(P)$ and $\mathcal{B}^j(P)$. If $\mathcal{I}(\mathcal{B}^i,\mathcal{B}^j)$ is not empty, we say $\mathcal{I}(\mathcal{B}^i,\mathcal{B}^j)$ is a *poly-triangle half-separator* of $\mathcal{B}(P)$.

Figure A.5 presents a 4-way cutting, where

$\mathcal{B}^1(P)=\{PP_2P_5P_6, PP_2P_6P_3, PP_3P_6P_5, PP_3P_5P_4\}$,

$\mathcal{B}^2(P)=\{PP_4P_5P_1, PP_1P_5P_2\}$,

$\mathcal{B}^3(P)=\{PP_7P_4P_1, PP_1P_2P_7\}$,

$\mathcal{B}^4(P)=\{PP_7P_2P_3, PP_7P_3P_4\}$.

Accordingly, the sets of poly-triangle half-separators are denoted as below.

$\mathcal{I}(\mathcal{B}^1,\mathcal{B}^2)=\{PP_2P_5, PP_5P_4\}$,

$\mathcal{I}(\mathcal{B}^2,\mathcal{B}^3)=\{PP_1P_4, PP_2P_1\}$,

$\mathcal{I}(\mathcal{B}^3,\mathcal{B}^4)=\{PP_2P_7, PP_7P_4\}$,

$\mathcal{I}(\mathcal{B}^4,\mathcal{B}^1)=\{PP_2P_3, PP_3P_4\}$.

A.2 Algorithms and proofs

A.2.1 Constrained face recovery

Lemma 1

Let S be a point and $\mathcal{B}(S)$ be the ball of S. The set $\{\mathcal{B}^i(P)\,|\,i=1,2\}$ is an 2-*way cutting* of $\mathcal{B}(P)$, and the corresponding poly-triangle separator is denoted by

$\mathcal{I}(\mathcal{B}^1,\mathcal{B}^2)=\{SP_iP_{i+1}\,|\,1\le i\le m, P_{m+1}=P_1\}$.

Relabel the points set $\{P_i\,|\,1\le i\le m, P_{m+1}=P_1\}$ such that the boundary of $\mathcal{I}(\mathcal{B}^1,\mathcal{B}^2)$ is a polygon depicted by a set of end-to-end edges, denoted by

$\mathcal{P}=\{P_iP_{i+1}\,|\,1\le i\le m, P_{m+1}=P_1\}$.

Besides, let the exterior boundary of $\mathcal{B}^i(i=1,2)$ be

$\mathcal{C}(\mathcal{B}^i)=\mathcal{I}(\mathcal{B}^1,\mathcal{B}^2)\cup\mathcal{F}(\mathcal{B}^i)$,

where

$\mathcal{F}(\mathcal{B}^i)=\{f_j^i\,|\,1\le j\le n_i\}$

is the set of exterior faces of $\mathcal{B}^i$ bounding $\mathcal{B}(S)$.

If

(1) S is visible to all the faces in $\mathcal{F}(\mathcal{B}^1)\cup\mathcal{F}(\mathcal{B}^2)$;

(2) A valid triangulation exists for the polygon $\mathcal{P}$, and a consistently oriented triangle set defining this triangulation is

$\mathcal{I}'=\{f_j\,|\,1\le j\le m-2\}$,

where $\mathcal{I}'$ is oriented such that

$\begin{cases}\mathcal{C}'(\mathcal{B}^1)=\mathcal{I}'\cup\mathcal{F}(\mathcal{B}^1)\\ \mathcal{C}'(\mathcal{B}^2)=\mathcal{R}(\mathcal{I}')\cup\mathcal{F}(\mathcal{B}^2)\end{cases}$

are consistently oriented triangle sets;

(3) There exists a neighborhood of S (denoted by $U_1(S)=\{S^*\,|\,|S^*-S|<\varepsilon_1\}$) and two subsets of $U_1(S)$ that defines the respective parts of $U_1(S)$ separated by $\mathcal{I}(\mathcal{B}^1,\mathcal{B}^2)$ (denoted by $U_1^1(S)$ and $U_1^2(S)$) such that all the points belonging to $U_1^1(S)$ and $U_1^2(S)$ are visible to $\mathcal{I}'$ and $\mathcal{R}(\mathcal{I}')$, respectively,

there must be two points $S_i(i=1,2)$ such that

(1) S_1 is visible to all the faces in $\mathcal{F}(\mathcal{B}^1)$ and $\mathcal{I}'$, and

(2) S_2 is visible to all the faces in $\mathcal{F}(\mathcal{B}^2)$ and $\mathcal{R}(\mathcal{I}')$.

Proof:

Since S is visible to all the faces in $\mathcal{F}(\mathcal{B}^1)\cup\mathcal{F}(\mathcal{B}^2)$, there exists a neighborhood of *S*, (denoted by $U_2(S)=\{S^* \,|\, |S^*-S|<\varepsilon_2\}$) such that all the points belonging to $U_2(S)$ are visible to the faces to $\mathcal{F}(\mathcal{B}^1)\cup\mathcal{F}(\mathcal{B}^2)$.

Let $\varepsilon_{\min}=\min(\varepsilon_1,\varepsilon_2)$, $U_{\min}(S)=\{S^* \,|\, |S^*-S|<\varepsilon_{\min}\}$ defines the intersection of $U_1(S)$ and $U_2(S)$, i.e.,

$U_{\min}(S)=U_1(S)\cap U_2(S)$.

Evidently, $U_{\min}(S)$ could be subdivided into two parts that belong to the regions bounded by $\mathcal{C}'(\mathcal{B}^1)$ and $\mathcal{C}'(\mathcal{B}^2)$, denoted by $U^1_{\min}(S)$ and $U^2_{\min}(S)$, respectively.

$\forall S_1\in U^1_{\min}(S)$, we know S_1 must be visible to all the faces in $\mathcal{F}(\mathcal{B}^1)$ (since $S_1\in U_2(S)$) and $\mathcal{I}'$ (since $S_1\in U^1_1(S)$).

$\forall S_2\in U^2_{\min}(S)$, we know S_2 must be visible to all the faces in $\mathcal{F}(\mathcal{B}^2)$ (since $S_2\in U_2(S)$) and $\mathcal{R}(\mathcal{I}')$ (since $S_2\in U^2_1(S)$). □

Based on Lemma 1, we could develop an algorithm to remove a Steiner point from the interior of a boundary face by splitting this point into two points located in either side of the face (see Figure A.6).

Algorithm A.1: face-added-point splitting

Let S be a Steiner point at the interior of a lost boundary face f, and

$\mathcal{I}=\{SP_iP_{i+1}\,|\,1\le i\le m, P_{m+1}=P_1\}$

be the set of sub-faces of f adjacent to S (see Figure A.6, where $m=4$). Relabel the points set $\{P_i\,|\,1\le i\le m, P_{m+1}=P_1\}$ such that the boundary of $\mathcal{I}$ is a polygon depicted by a set of end-to-end edges, denoted by

$\mathcal{P}=\{P_iP_{i+1}\,|\,1\le i\le m, P_{m+1}=P_1\}$.

Evidently, $\mathcal{I}$ defines a poly-triangle separator of $\mathcal{B}(S)$. Let $\mathcal{B}^1(S)$ and $\mathcal{B}^2(S)$ be the two parts of $\mathcal{B}(S)$ separated by $\mathcal{I}$ and the exterior boundary of $\mathcal{B}^i(i=1,2)$ be

$\mathcal{C}(\mathcal{B}^i)=\mathcal{I}\cup\mathcal{F}(\mathcal{B}^i)$,

where

$\mathcal{F}(\mathcal{B}^i)=\{f^i_j\,|\,1\le j\le n_i\}$

is the set of exterior faces of $\mathcal{B}^i$ bounding $\mathcal{B}(S)$. Reorder the vertices of all faces in $\mathcal{C}(\mathcal{B}^1)$ and $\mathcal{C}(\mathcal{B}^2)$ is visible to S.

Now, we describe the general procedures included in the face-added-point splitting algorithm.

(1) Remove all the elements belonging to $\mathcal{B}(S)$.

(2) Retriangulate the polygon $\mathcal{P}$. Let the consistently oriented triangle set defining this triangulation be

$\mathcal{I}'=\{f_j\,|\,1\le j\le m-2\}$,

where $\mathcal{I}'$ is oriented such that

$$\begin{cases}\mathcal{C}'(\mathcal{B}^1)=\mathcal{I}'\cup\mathcal{F}(\mathcal{B}^1)\\ \mathcal{C}'(\mathcal{B}^2)=\mathcal{R}(\mathcal{I}')\cup\mathcal{F}(\mathcal{B}^2)\end{cases}$$

are consistently oriented triangle sets;

(3) Search two points (denoted by S_1 and S_2) that are respectively located at the interior of the regions bounded by $\mathcal{C}'(\mathcal{B}^1)$ and $\mathcal{C}'(\mathcal{B}^2)$ such that

(c.1) S_1 is visible to all the faces in $\mathcal{F}(\mathcal{B}^1)$ and $\mathcal{I}'$, and

(c.2) S_2 is visible to all the faces in $\mathcal{F}(\mathcal{B}^2)$ and $\mathcal{R}(\mathcal{I}')$.

(4) Connect S_1 to all the faces in $\mathcal{F}(\mathcal{B}^1)$ and $\mathcal{I}'$, and connect S_2 to all the faces in $\mathcal{F}(\mathcal{B}^2)$ and $\mathcal{R}(\mathcal{I}')$.

We have the following lemma for the face-added-point splitting algorithm.

Lemma 2

Let the face-added-point splitting algorithm be defined as above. If $\mathcal{B}(S)$ is a valid tetrahedralization, the algorithm could always output a valid tetrahedralization.

Proof:

Firstly, we prove that the algorithm could provide an output as defined in the finite number of steps. It is evident that this discussion on Steps (a) and (d) is unnecessary. With respect to Step (b), since the polygon $\mathcal{P}$ is planar, the algorithm such as the ear clipping algorithm could always provide a valid triangulation in at most $\mathcal{O}(m^2)$ steps.

With respect to Step (c), three conditions listed in Lemma 1 are met exactly.

(1) Since $\mathcal{B}(S)$ is a valid tetrahedralization, S is visible to all the faces in $\mathcal{F}(\mathcal{B}^1)\cup\mathcal{F}(\mathcal{B}^2)$.

(2) Since the polygon $\mathcal{P}$ is planar, a valid triangulation of $\mathcal{P}$ could always be provided as defined in the algorithm.

(3) Again, since the polygon $\mathcal{P}$ is planar, two half spaces could be defined as below

$$\begin{cases}U^+=\{P\mid P \text{ is visible to } \mathcal{I}'\}\\ U^-=\{P\mid P \text{ is visible to } \mathcal{R}(\mathcal{I}')\}\end{cases}.$$

Two nonempty point sets $U_1^1(S)$ and $U_1^2(S)$ could then be defined as blow.

$$\begin{cases}U_1^1(S)=U_1(S)\cap U^+\\ U_1^2(S)=U_1(S)\cap U^-\end{cases},$$

where $U_1(S)=\{S^*\mid\left|S^*-S\right|<\varepsilon_1\}$ is a neighborhood of *S*, and ε_1 could be an arbitrary positive real number.

According to Lemma 1, there exist qualified S_1 and S_2 that meet the conditions (c.1) and (c.2) listed in Step (c) of the algorithm. As a result, we know the output of the algorithm (i.e., $\mathcal{B}(S_1)\cup\mathcal{B}(S_2)$) is always valid.

Next, we prove the algorithm could be finished in the finite number of steps. Again, our focus is on Step (c).

Let $U_2(S)=\{S^*\mid\left|S^*-S\right|<\varepsilon_2\}$. Since ε_1 could be an arbitrary large number,

$\varepsilon_{\min}=\min(\varepsilon_1,\varepsilon_2)=\varepsilon_2$.

We could then search a qualified S_1 along the normal direction of f by using the bisectional scheme. In theory, this scheme could always be finished in $\log_2(l_0/\varepsilon_2)$ steps, where l_0 is the initial search length.

Likely, we could search a qualified S_2 in the finite number of steps. □

Furthermore, we could give the following theorem illustrating that we could achieve the final constrained recovery of a face by performing the face-added-point splitting algorithm for each Steiner point on f.

Theorem 1

Suppose f is a boundary face with its three edges being recovered but its interior being split into a union of triangles. We could then perform the finite number of the face-added-point splittings to achieve the final constrained recovery. Moreover, the recovery of f does not affect other existing or recovered constraints if the prescribed constraints contain no self-intersections.

Proof

We know from Lemma 2 that each face-added-point splitting reduces an added point. As the number of added points (denoted by *N*) is finite, eventually, we achieve the final constrained recovery of f by performing *N* splittings. As seen in Figure A.6, all the removed edges by the splitting algorithm are connected to the Steiner point. If the prescribed constraints contain no self-intersections, the recovery of f does not affect other existing or recovered constraints.

A.2.2 Constrained edge recovery

A.2.2.1 Constrained recovery of manifold edges. Here, manifold edges refer to those edges shared by two boundary faces. Based on Lemma 1, we could develop a similar algorithm as Algorithm A.1 to remove a Steiner point from the interior of a manifold edge (see Figure A.7).

Algorithm A.2: manifold-edge-added-point splitting

Let e be a conformingly recovered manifold edge, f_1 and f_2 are two faces meeting at e. S be a Steiner point at the interior of e, and

$$\mathcal{I} = \{SP_iP_{i+1} \mid 1 \le i \le m, P_{m+1} = P_1\}$$

be the set of sub-faces on f_1 and f_2 (see Figure A.7, where $m = 4$). $\mathcal{I}$ is subdivided into two parts by e and each part defines a set of sub-faces on f_1 or f_2, denoted by $\mathcal{I}_1$ and $\mathcal{I}_2$, respectively. Accordingly, the boundary polygon of $\mathcal{I}$ (i.e., $\mathcal{P}$) is subdivided into two parts by e, denoted by $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively.

Let other notations be defined as in Algorithm A.1. Algorithm A.2 includes the following steps.

(a) Remove all the elements belonging to $\mathcal{B}(S)$.

(b) Retriangulate the polygon $\mathcal{P}$ by triangulating $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively.

(c) Search two points S_1 and S_2 meeting the conditions (c.1) and (c.2) defined in Algorithm A.1.

(d) Connect S_1 to all the faces in $\mathcal{F}(\mathcal{B}^1)$ and $\mathcal{I}'$, and connect S_2 to all the faces in $\mathcal{F}(\mathcal{B}^2)$ and $\mathcal{R}(\mathcal{I}')$.

Like Lemma 2, we have the following lemma for Algorithm A.2.

Lemma 3
Let Algorithm A.2 be defined as above. If $\mathcal{B}(S)$ is a valid tetrahedralization, the algorithm could always output a valid tetrahedralization.

The proof of Lemma 3 is very similar to its counterpart of Lemma 2. The differences between these two proofs are listed as below.

(1) With respect to Step (b), a valid triangulation of $\mathcal{P}$ could always be provided by triangulating $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively.

(2) By performing the bisectional search along the average normal direction of f_1 and f_2, the qualified S_1 and S_2 could be obtained in the finite number of steps.

It is worth noting that the above differences do not change the theoretical roundness of Lemma 3.

A.2.2.2 Constrained recovery of non-manifold edges. Here, non-manifold edges refer to those edges shared by more than two boundary faces. To be concise, we directly present the algorithm (Algorithm A.3) for the removal of a Steiner point at the interior of a non-manifold edge (see Figure A.8) and then use a lemma (Lemma 4) to depict the property of Algorithm A.3.

Algorithm A.3: non-manifold-edge-added-point splitting

Let e be a conformingly recovered non-manifold edge, and the set of faces meeting at e be

$\mathcal{S} = \{f_i \mid i = 1, 2, \cdots, n, f_0 = f_n\}$.

Relabel $\mathcal{S}$ such that the dihedral angle between a pair of neighboring faces

$\mathcal{Q}_i = \mathcal{Q}_i(f_{i-1}, f_i)(i = 1, 2, \cdots, n, \mathcal{Q}_{n+1} = \mathcal{Q}_1)$

does not contain other faces. Reorder the vertices of each face such that the normal direction of $f_i (1 \le i \le n)$ points toward $\mathcal{Q}_{i+1}$ while that of f_{i-1} points outward $\mathcal{Q}_i$ (see Figure A.8(a)).

Let S be a Steiner point at the interior of e, and $\mathcal{I}$ be the set of triangles adjacent to S that are sub-faces of $f_i (1 \le i \le n)$. Denote the set of vertices of these sub-faces, excluding S, by $\mathcal{V}$.

$\mathcal{I}$ is subdivided into n parts by e and the set of sub-faces on f_i is denoted by $\mathcal{I}_i$. Accordingly, $\{\mathcal{B}^i(S) \mid 1 \le i \le n, \mathcal{B}^{n+1} = \mathcal{B}^1\}$ defines a n-way cutting of $\mathcal{B}(S)$, where $\mathcal{B}^i(S)$ is the set of tetrahedra located in $\mathcal{Q}_i$ (see Figure A.8(b)); $\{\mathcal{I}_i \mid 1 \le i \le n, \mathcal{I}_0 = \mathcal{I}_n\}$ defines a group of poly-triangle half-separators, i.e.,

$\mathcal{I}_i = \mathcal{I}(\mathcal{B}^i, \mathcal{B}^{i+1})(1 \le i \le n)$.

The boundary polygon of $\mathcal{I}_i$ could then be depicted by a set of end-to-end mesh edges

$\mathcal{P}_i = \{< P_j, P_k > \mid P_j, P_k \in \mathcal{V}\}$.

For instance, as seen in Figure A.8(b),

$\mathcal{P}_1 = \{< P_4, P_2 >, < P_2, P_5 >, < P_5, P_4 >\}$.

Let the exterior boundary of $\mathcal{B}^i$ be

$\mathcal{C}(\mathcal{B}^i) = \mathcal{I}_{i-1} \cup \mathcal{I}_i \cup \mathcal{F}(\mathcal{B}^i)(1 \le i \le n)$,

where $\mathcal{F}(\mathcal{B}^i)$ is the set of exterior faces of $\mathcal{B}^i$ bounding $\mathcal{B}(S)$.

Now, we describe the general procedures included in Algorithm A.3.

(a) Remove all the elements belonging to $\mathcal{B}(S)$.

(b) For each poly-triangle half-separator $\mathcal{I}_i$, retriangulate its boundary polygon $\mathcal{P}_i$. Let the consistently oriented triangle set defining this triangulation be $\mathcal{I}_i'$, where $\mathcal{I}_i'$ is oriented such that $\mathcal{C}'(\mathcal{B}^i) = \mathcal{I}_{i-1}' \cup \mathcal{R}(\mathcal{I}_i') \cup \mathcal{F}(\mathcal{B}^i)\ (1 \le i \le n)$ are consistently oriented triangle sets;

(c) For each empty polyhedron bounded by $\mathcal{C}'(\mathcal{B}^i)$, search a point (denoted by S_i) that is located in the interior of this polyhedron such that

(c.1) S_i is visible to all the faces in $\mathcal{C}'(\mathcal{B}^i)$.

(d) For each $S_i (1 \le i \le n)$, connect it to all the faces in $\mathcal{C}'(\mathcal{B}^i)$.

Like Lemma 3, we have the following lemma for Algorithm A.2.

Lemma 4

Let Algorithm A.3 be defined as above. If $\mathcal{B}(S)$ is a valid tetrahedralization, the algorithm could always output a valid tetrahedralization.

The proof of Lemma 4 is very similar to its counterpart of Lemma 3. The minor difference is that we now require $n (n > 2)$ Steiner points. Each Steiner point is located in the interior of the empty polyhedron bounded by $\mathcal{C}'(\mathcal{B}^i) (1 \le i \le n)$ and computed along the average normal direction of f_{i-1} and $\mathcal{R}(f_i)$ (i.e., the reverse of f_i) by the bisectional search scheme. Evidently, both this search scheme and Algorithm A.3 could be competed in the finite number of steps.

A.2.2.3 Edge-added-point splitting. For a boundary edge with its interior being split into a union of linear *segments*, Algorithm A.2 or Algorithm A.3 could then be employed to achieve its final constrained recovery. The pseudo-code form for this recovery procedure is simply given as:

Algorithm A.4: edge-added-point splitting

```
If the edge is a manifold edge
   Perform Algorithm A.2.
Else
   Perform Algorithm A.3.
End if
```

We now give the following theorem illustrating that we could achieve the final constrained recovery of an edge (denoted by e) by performing Algorithm A.4 for each Steiner point on e.

Theorem 2

Suppose e is a boundary edge with its interior being split into a union of linear segments. We could then perform the finite number of the edge-added-point splittings to achieve the final constrained recovery. Moreover, the recovery of e does not affect other existing or recovered constraints if the prescribed constraints contain no self-intersections.

Proof

We know from Lemma 3 and Lemma 4 that each edge-added-point splitting reduces an added point. As the number of added points (denoted by *N*) is finite, eventually, we achieve the final constrained recovery of f by performing *N* splittings. As seen in Figures A.7 and A.8, all the removed edges by the splitting algorithm are connected to the Steiner point. If the prescribed constraints contain no self-intersections, the recovery of e does not affect other existing or recovered constraints. □

A.2.3 Constrained boundary recovery: the overall algorithm

Now, given a set of boundary constraints that have been recovered conformingly, we could achieve the constrained recovery of these constraints by first treating Steiner points on boundary edges individually by Algorithm A.4, and then treating Steiner points on boundary faces individually by Algorithm A.1. The pseudo-code form for this recovery procedure is simply given as:

Algorithm A.5: constrained boundary recovery

```
For each edge containing Steiner points in their interiors (denoted by e)
   For each Steiner point on e (denoted by S)
      Perform edge-added point splitting (i.e., Algorithm A.4) for S.
   End for
End for
For each face containing Steiner points in their interiors (denoted by f)
   For each Steiner point on f (denoted by S)
      Perform face-added point splitting (i.e., Algorithm A.1) for S.
   End for
End for
```

We now give the following theorem illustrating that we could achieve the final constrained recovery of a set of boundary constraints that have been recovered conformingly by performing Algorithm A.5.

Theorem 3

Given a set of boundary constraints that have been recovered conformingly, we could then perform the finite number of the edge-added-point splittings and face-added-point splittings to achieve the final constrained recovery of these constraints by performing Algorithm A.5.

Proof

The proof is straightforward. Since the recovery of each edge and each face could be achieved in the finite number of steps, and the recovery procedure does not affect other existing or recovered constraints, plus the numbers of edges and faces for treatment are finite, Algorithm A.5 could always be finished in the finite number of steps. □

A.3 Implementation issues and remarks

In Section A.2, all the presented proofs do not consider the round-off errors due to floating point numbers. The robustness of Algorithm A.5, although theoretically it could provide a valid output in the finite number of steps (see Theorem 3), could be challenged in the real world.

For instance, there might be a very small neighbourhood of S (denoted by $U_2(S) = \{S^* \mid \left|S^* - S\right| < \varepsilon_2\}$ in the previous discussions) in which all the points are visibile to

the exterior boundary of $\mathcal{B}(S)$. This situation could be induced by a poorly shaped exterior boundary of $\mathcal{B}(S)$, such as with 1 or 2 small angles and/or with small or near-360-degree dihedral angles between two boundary triangles.

To improve the robustness, an idea is to enlarge the cavity for remeshing by adding more neighbouring elements, for instance, according to the Delaunay criteria **[6, 19]**. Nevertheless, caution must be taken to ensure the remeshing of the enlarged cavity does not affect other existing or recovered constraints **[6]**.

Note that in the proof for Lemma 2, for the simplicity of the proof, we suggest a simple bisectional search scheme along a specified direction. Evidently, this 1D search scheme could be problematic in the case of a very small $U_2(S)$. To improve the robustness, we **[15]** suggest a mesh optimization based scheme **[46]** to search the possible positions of Steiner points after splitting in the 3D space. In practice, to achieve a trade-off between computing time and robustness, we could first perform the 1D search, and switch to the time-consuming 3D search when the 1D search fails to provide a desirable output **[15]**.

Even if all the schemes for robustness mentioned above are incorporated, the boundary recovery procedure could still fail in the real work, when an excessive number of Steiner points are required for the recovery. This situation usually happens when the input surface contains a certain number of elements having high aspect ratios. For Algorithm A.5, at least two issues could be introduced by Steiner points. Firstly, the positions of Steiner points stored with floating-point numbers are essentially inaccurate due to round-off errors. These errors can accumulate if an excessive number of Steiner points are inserted. Predicates **[20]** with these positions as inputs may return an undesirable value and collapse the entire boundary recovery procedure. Secondly, the insertion of Steiner points is accompanied with the splitting of mesh elements in the conforming recovery procedure. The resulting poorly shaped elements could do harm to the robustness of both the conforming and constrained recovery procedures. Therefore, reducing the usage of Steiner points in boundary recovery could benefit a lot for the robustness of the boundary recovery algorithm like Algorithm A.5.

Acknowledgements: This research is funded by the National Natural Science Foundation of China (Grant No. 11432013, 11172267 and 10872182), the Joint Fund of the National Natural Science Foundation of China and China Academy of Engineering Physics (Grant No. U1630121) and Zhejiang Provincial Natural Science Foundation of China (Grant No. LR16F020002 and Y1110038). The authors acknowledge Dr. Dawei Zhao at Zhejiang University for his cooperation in preparing the results of parallel mesh generation. The first author would like to thank the financial support from Zhejiang University and China Scholarship Council during his research visit at Swansea University, UK. The authors appreciate the valuable comments and constructive suggestions from the anonymous reviewers, which help improve the work a lot.

References

[1] Ruppert J, Seidel R. On the difficulty of triangulating three-dimensional non-convex polyhedra. *Discrete and Computational Geometry* 1992; **7**:227-254.

[2] Schönhardt E. Über die zerlegung von dreieckspolyedern in tetraeder. *Mathematische Annalen* 1928; **98**:309-312.

[3] Chazelle B. Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm. *SIAM Journal On Scientific Computation* 1984; **13**:488-507.

[4] Weatherill NP, Hassan O. Efficient three-dimensional Delaunay triangulation with automatic point creation and imposed boundary constraints. *International Journal for Numerical Methods in Engineering* 1994; **37**:2005-2039.

[5] George PL, Borouchaki H, Saltel E. 'Ultimate' robustness in meshing an arbitrary polyhedron. *International Journal for Numerical Methods in Engineering* 2003; **58**:1061-1089.

[6] Du Q, Wang D. Constrained boundary recovery for three dimensional Delaunay triangulations. *International Journal for Numerical Methods in Engineering* 2004; **61**:1471-1500.

[7] George PL, Hecht F, Saltel E. Automatic mesh generator with specified boundary. *Computer Methods in Applied Mechanics and Engineering* 1991; **92**:269-288.

[8] Joe B. Three-dimensional boundary-constrained triangulations. *Proceedings of the 13th IMACS World Congress on Computation and Applied Mathematics*, Trinity College, Dublin, Ireland 1991; 215-222.
[9] Lewis RW, Zheng Y, Gethin DT. Three-dimensional unstructured mesh generation: part 3. volume meshes. *Computer Methods in Applied Mechanics and Engineering* 1996; **134**:285-310.
[10] Liu A, Baida M. How far flipping can go towards 3D conforming/constrained triangulation. *Proceedings of the 9th International Meshing Roundtable*, New Orleans, LA, USA, 2000; 307-315.
[11] Du Q, Wang D. Boundary recovery for three dimensional conforming Delaunay triangulation. *Computer Methods in Applied Mechanics and Engineering* 2004; **193**:2547-2563.
[12] Guan Z, Song C, Gu Y. The boundary recovery and sliver elimination algorithms of three-dimensional constrained Delaunay triangulation. *International Journal for Numerical Methods in Engineering* 2006; **68**:192-209.
[13] Chen J, Zheng Y. Redesign of a conformal boundary recovery algorithm for 3D Delaunay triangulation. *Journal of Zhejiang University SCIENCE A* 2006; **7**:2031-2042.
[14] Liu J, Chen B, Chen Y. Boundary recovery after 3D Delaunay tetrahedralization without adding extra nodes. *International Journal for Numerical Methods in Engineering* 2007; **72**:744-756.
[15] Chen J, Zhao D, Huang Z, Zheng Y, Gao S. Three-dimensional constrained boundary recovery with an enhanced Steiner point suppression procedure. *Computers and Structures* 2011; **89**:455-466.
[16] Shewchuk JR. Constrained Delaunay tetrahedralizations and provably good boundary recovery. *Proceedings of the 11th International Meshing Roundtable*, Ithaca, NY, USA, 2002; 193-204.
[17] Si H, Gärtner K. 3D boundary recovery by constrained Delaunay tetrahedralization. *International Journal for Numerical Methods in Engineering* 2011; **85**:1341-1364.
[18] Liu Y, Lo SH, Guan Z, Zhang H. Boundary recovery for 3D Delaunay triangulation. *Finite Elements in Analysis and Design* 2014; **84**:32-43.
[19] Si H. TetGen, a Delaunay-based quality tetrahedral mesh generator. *ACM Transactions on Mathematical Software* 2015; **41**: 11:1-11:36.
[20] Shewchuk JR. Robust adaptive floating-point geometric predicates. *Proceedings of the 12th Annual Symposium on Computational Geometry*, Philadelphia, PA, USA, 1996; 141-150.
[21] Freitag LA, Ollivier-Gooch C. Tetrahedral mesh improvement using swapping and smoothing. *International Journal for Numerical Methods in Engineering* 1997; **40**:3979-4002.
[22] Said R, Weatherill NP, Morgan K, Verhoeven NA. Distributed parallel Delaunay mesh generation. *Computer Methods in Applied. Mechanic Engineering* 1999; **177**:109-125.
[23] Larwood BG, Weatherill NP, Hassan O, Morgan K. Domain decomposition approach for parallel unstructured mesh generation. *International Journal for Numerical Methods in Engineering* 2003; **58**:177-188.
[24] Chen J, Zhao D, Huang Z, Zheng Y, Wang D. Improvements in the reliability and element quality of parallel tetrahedral mesh generation. *International Journal for Numerical Methods in Engineering* 2012; **92**:671-693.
[25] Hassan O, Probert EJ, Morgan K, Peraire J. Mesh generation and adaptivity for the solution of compressible viscous high speed flows. *International Journal for Numerical Methods in Engineering* 1995; **38**:1123-1148.
[26] Hassan O, Morgan K, Probert EJ, Peraire J. Unstructured tetrahedral mesh generation for three-dimensional viscous flows. *International Journal for Numerical Methods in Engineering* 1996; **39**:549-567.
[27] Ito Y, Murayama M, Yamamoto K, Shih AM, Soni BK. Efficient hybrid surface/volume mesh generation using suppressed marching-direction method. *AIAA Journal* 2013; **51**:1450-1461.
[28] Hassan O, Probert EJ, Morgan K. Unstructured mesh procedures for the simulation of three-dimensional transient compressible inviscid flows with moving boundary components. *International Journal for Numerical Methods in Fluids* 1998; **27**:41-55
[29] Tremel U, Sørensen KA, Hitzel S, Rieger H, Hassan O, Weatherill NP. Parallel remeshing of unstructured volume grids for CFD applications. *International Journal for Numerical Methods in Fluids* 2007; **53**:1361-1379.
[30] Zheng J, Chen J, Zheng Y, Yao Y, Li S, Xiao Z. An Improved Local Remeshing Algorithm for Moving Boundary Problems. *Engineering Applications of Computational Fluid Mechanics* 2016; **10**: 405-428.
[31] Joe B. Construction of three-dimensional improved quality triangulations using local transformations. *SIAM Journal On Scientific Computation* 1995; **16**:1292-1307.
[32] George PL, Borouchaki H. Back to edge flips in 3 dimensions. *Proceedings of the 12th International Meshing Roundtable*, Santa Fe, NM, USA, 2003; 393-402.
[33] Shewchuk JR. Two discrete optimization algorithms for the topological improvement of tetrahedral meshes. 2002. Unpublished manuscript. Sep-27-2015. URL: https://www.cs. berkeley.edu/~jrs/papers /edge.pdf.

[34] De l'Isle EB, George PL. Optimization of tetrahedral meshes. *IMA Volumes in Mathematics and its Applications* 1995; **75**:97–128.
[35] Liu J, Chen B, Sun S. Small polyhedron reconnection for mesh improvement and its implementation based on advancing front technique. *International Journal for Numerical Methods in Engineering* 2009; **79**: 1004-1018.
[36] Si H. On 3D indecomposable and irreducible polyhedra and the number of Steiner Points. 2002. May-17-2017. URL: www.wias-berlin.de/people/si/course/files/talk-numgrid2016.pdf.
[37] Goerigk N, Si H. On indecomposable polyhedra and the number of Steiner Points. *Proceedings of the 24th International Meshing Roundtable*, Austin, TX, USA, 2015; 343-355.
[38] Klincsek GT. Minimal tiangulations of polygonal dmains. *Annals of Discrete Mathematics* 1980; **9**:121-123.
[39] Bowyer A. Computing Dirichlet tessellations. *The Computer Journal* 1981; **24**:162-166.
[40] Watson D. Computing the n-dimensional Delaunay tessellation with application to Voronoï polytopes. *The Computer Journal* 1981; **24**:167-172.
[41] George PL. Improvements on Delaunay-based three-dimensional automatic mesh generator. *Finite Elements in Analysis and Design* 1997; **25**:297-317.
[42] 3D Meshes Research Database by INRIA GAMMA Group, Mar-05-2014. URL: https:// www.rocq.inria.fr/gamma/download/download.php.
[43] GHS3D: A Powerful Isotropic Tet-Mesher, Mar-05-2014. URL: https://www.rocq.inria.fr/gamma /gamma/ghs3d/.
[44] Xie L, Zheng Y, Chen J, Zou J. Enabling technologies in the problem solving environment HEDP. *Communications in Computational Physics* 2008; **4**:1170–1193.
[45] Snyder DO, Koutsavdis EK, Anttonen JS. Transonic Store Separation Using Unstructured CFD with Dynamic Meshing. *Proceedings of the 33rd AIAA Fluid Dynamics Conference and Exhibit*, Orlando, FL, USA, 2003. AIAA 2003-3919.
[46] Escobar JM, Montenegro R, Montero G, Rodríguez E, González-Yuste JM. Smoothing and local refinement techniques for improving tetrahedral mesh quality. *Computers and Structures* 2005; **83**:2423-2430.

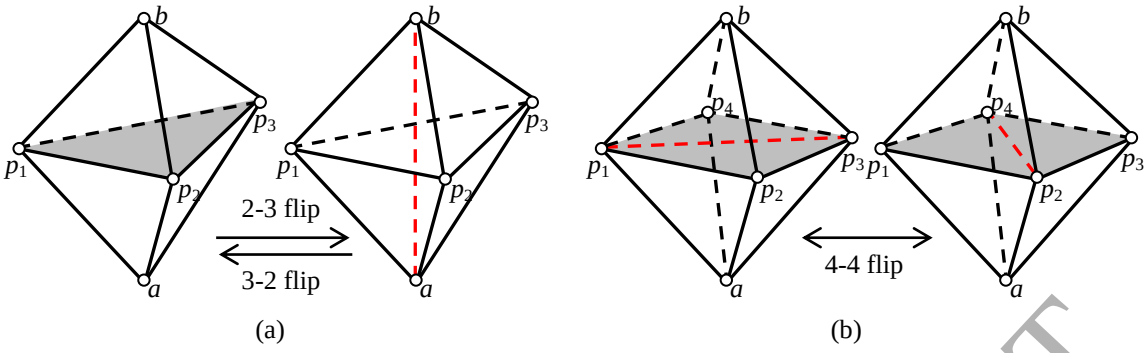


Figure 1. Basic flips for a tetrahedral mesh: (a) 2-3 flip and 3-2 flip; (b) 4-4 flip.

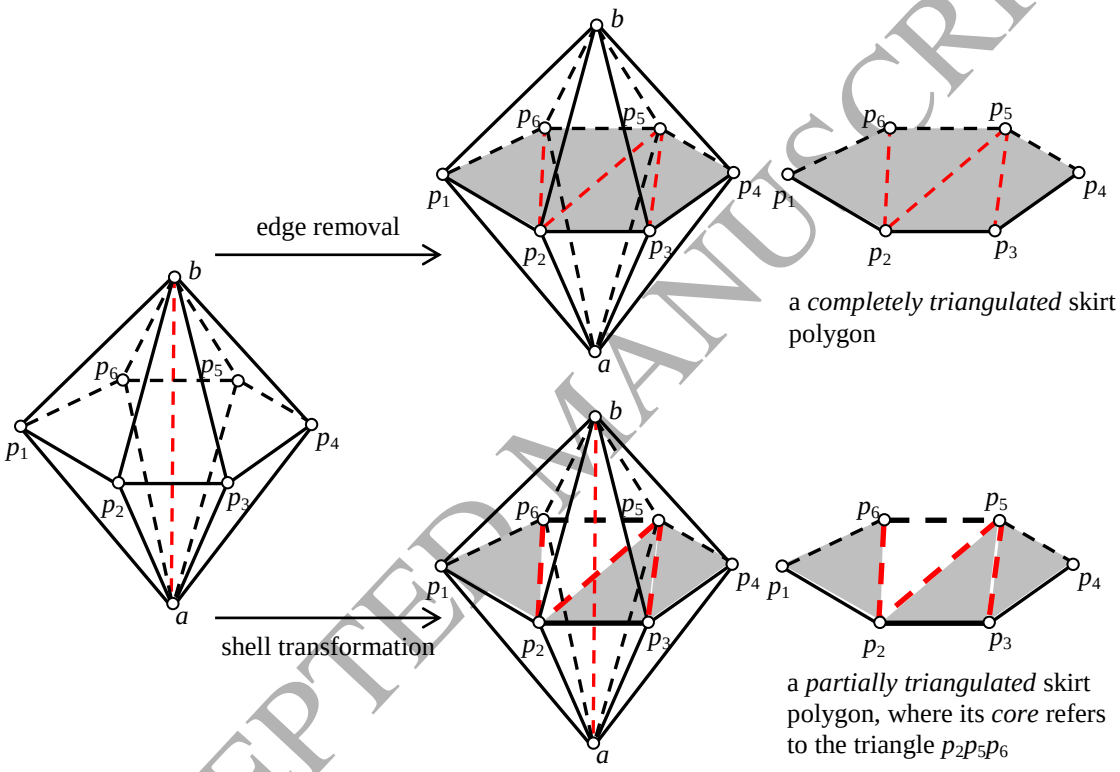


Figure 2. The difference between edge removal and a single calling of shell transformation. This difference enables shell transformation to be called recursively while edge removal cannot. This recursive ability is the main advantage of shell transformation technique.

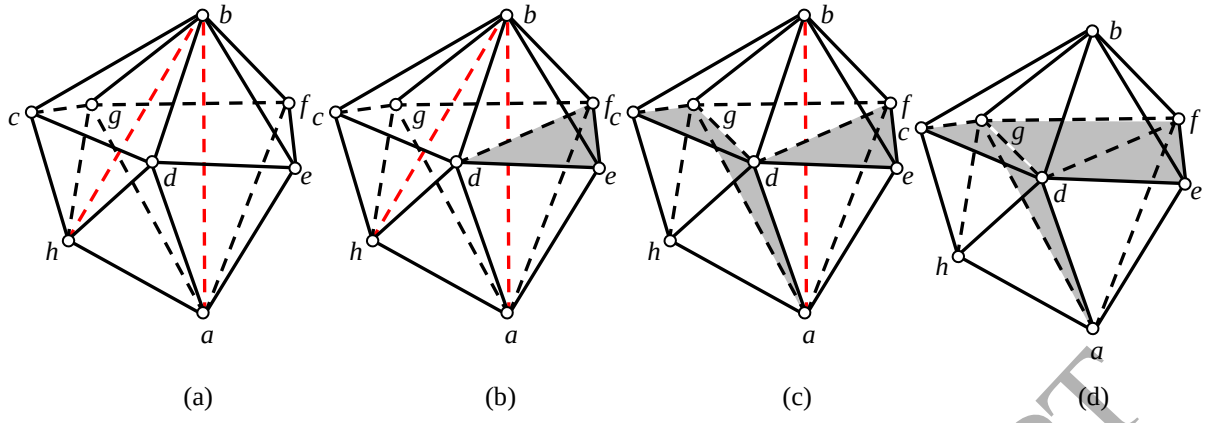


Figure 3. Illustration for the recursive scheme of shell transformation. (a) The input mesh, composed of two shells that are supported by the edges *ab* and *bh*, respectively. (b) The output after the first shell transformation calling on the shell of *ab*. (c) The output after the second shell transformation calling on the shell of *bh*. (d) The final output after the third shell transformation calling on the shell of *ab*.

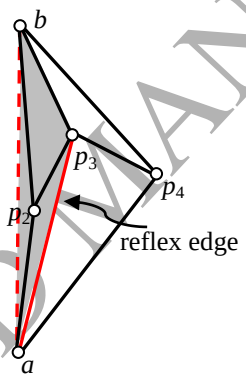


Figure 4. Illustrative case of a *reflex edge*.

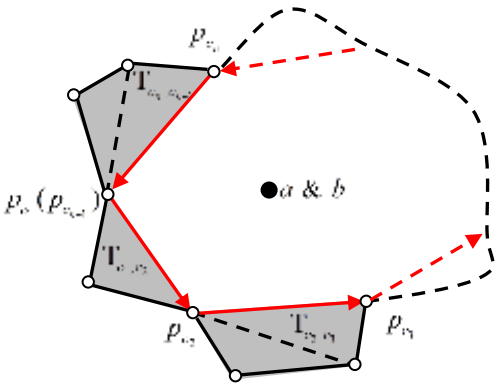
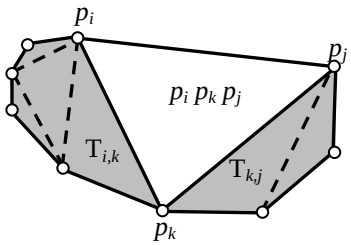


Figure 5. A general partial triangulation case of the skirt polygon.



$T_{i,j} = T_{i,k} \cup T_{k,j} \cup \{\Delta p_i\ p_k\ p_j\}$

Figure 6. Decomposing a triangulation optimisation problem into subproblems.

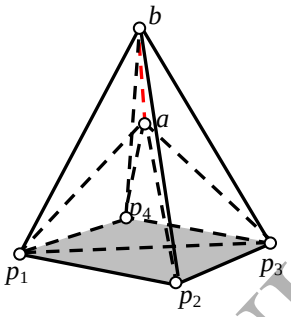


Figure 7. A Christmas tree type mesh.

Figure 8. (a) The surface mesh and (b) volume mesh after constrained recovery (cutting view) of the Mohne model.

Figure 9. (a) The surface mesh and (b) volume mesh after constrained recovery (cutting view) of the B747 aircraft model.

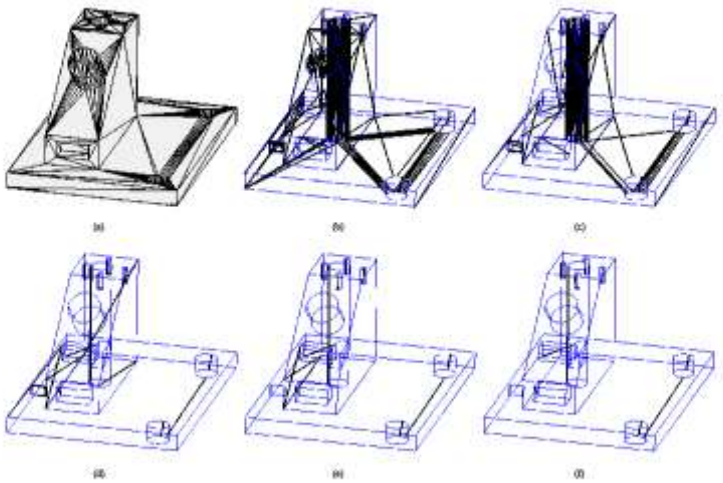


Figure 10. The edge recovery process of the Cami1 model. The surface mesh is shown in (a). 243 lost edges after inserting boundary points are shown in (b). 10 lost edges after edge recovery are shown in (f). (c), (d) and (e) highlight the lost edges in three different phases when calling Algorithm 5 for the first time. (c) l_{max} = 0; 127 lost edges. (d) l_{max} = 2; 40 lost edges. (e) l_{max} = 9; 13 lost edges.

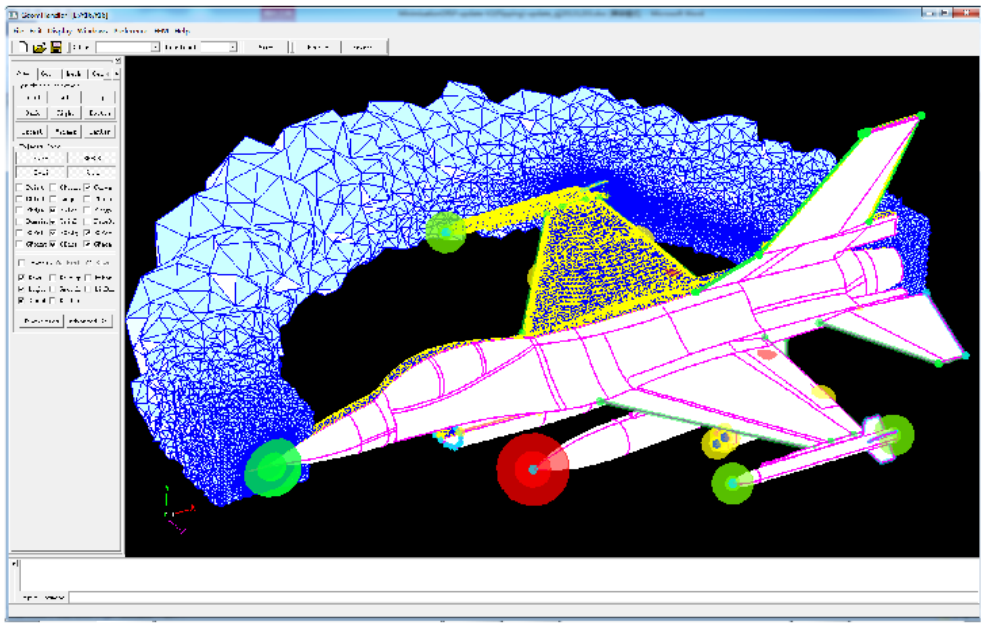


Figure 11. Visual representation of the geometry, the surface and volume mesh of the F16 aircraft model in the in-house mesh generation system HEDP/PRE.

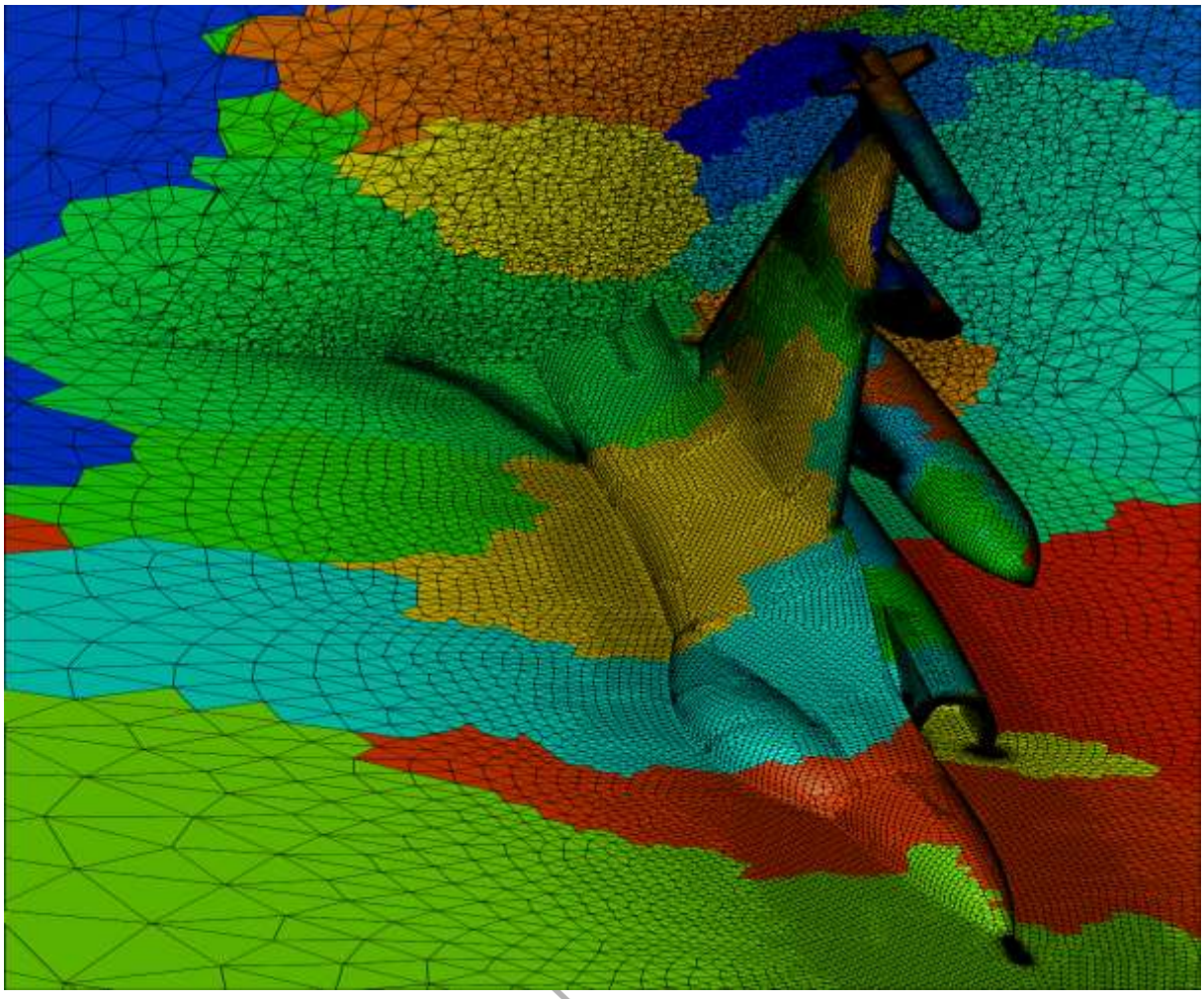


Figure 12. A mesh of the F16 aircraft model generated by the parallel mesher.

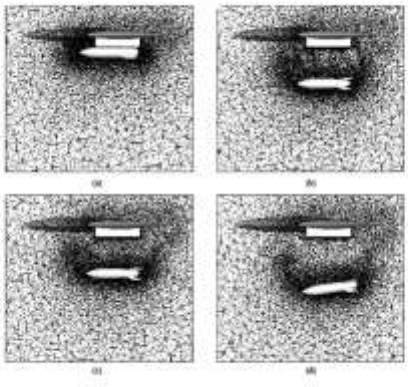


Figure 13. Cut views of the volume meshes for the wing/pylon/finned-store separation at different time steps (t_s): (a) t_s= 0s; (b) t_s= 0.38s, before local remeshing; (c) t_s= 0.38s, after local remeshing; (d) t_s= 0.5s.

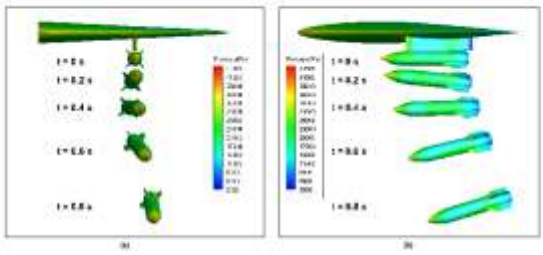


Figure 14. Contour plot of the computed pressure distributions during the separation: (a) front view; (b) side view.

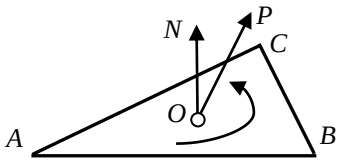


Figure A.1. Visibility of a face to a point.

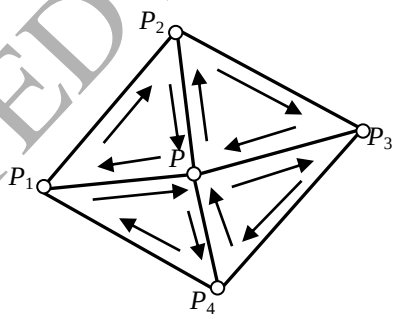


Figure A.2. A consistently oriented triangle set.

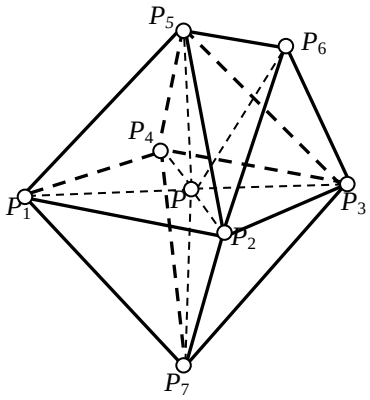


Figure A.3. A ball of a point P.

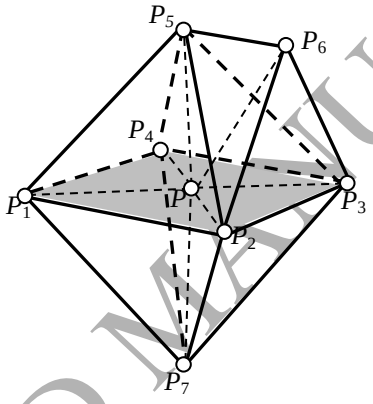


Figure A.4. A 2-*way cutting* and its corresponding *poly-triangle separator*.

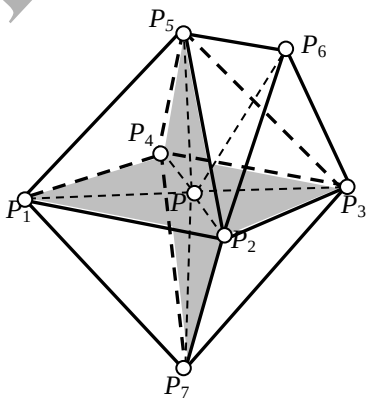


Figure A.5. A 4-*way cutting* and its corresponding *poly-triangle half-separators*.

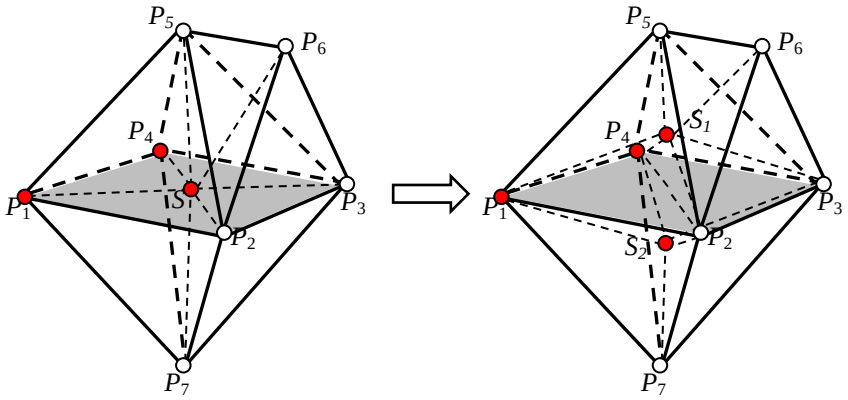


Figure A.6. An illustrative example for the face-added point splitting.

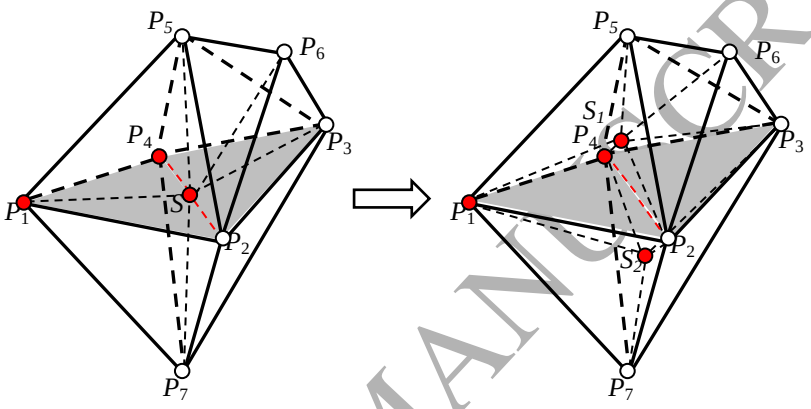
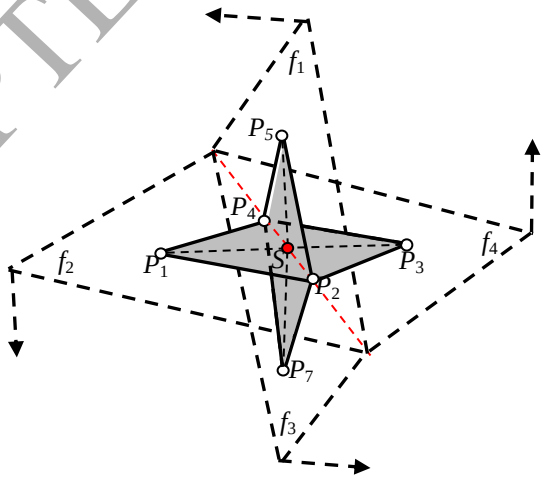


Figure A.7. An illustrative example for the manifold-edge-added point splitting.



(a)

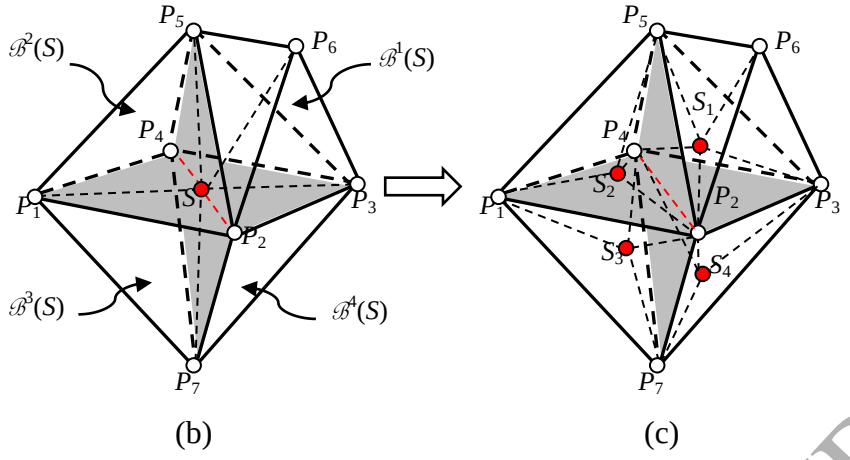


Figure A.8. An illustrative example for the non-manifold-edge-added point splitting.

Table 1. Comparison of different constrained recovery algorithms (NP: Not Reported). The performance data for the old GHS3D and Liu's algorithm are referred to **[10]**.

Examples	#Boundary faces	#Boundary nodes	Steiner points inserted in constrained boundary recovery				
			Our algorithm	TetGen 1.5	GHS3D 4.2-2 (New)	GHS3D (Old)	Liu's algorithm
Cami1a	832	408	0	1	13	NP	NP
Cami1	932	460	1	2	Fail	NP	NP
Mohne	5560	2760	1	3	35	Fail	26
Thepart	1992	994	0	0	4	24	6
Boeing_part	2472	1218	5	4	22	Fail	25
Thru-mazewheel	5622	2781	4	5	41	18	13
B747	5166	2560	0	0	Fail	NP	NP

Table 2. Timing data of the proposed algorithm and the new GHS3D (unit: second)

Algorithms	Cami1a	Cami1	Mohne	Thepart	Boeing_part	Thru-mazewheel	B747
Our Algorithm	0.61	0.48	6.27	0.39	0.79	0.59	0.4
Tetgen 1.5	0.048	0.067	0.33	0.088	0.16	0.19	0.1
GHS3D 4.2-2	4.99	Fail	11.9	1.35	3.89	7.46	Fail

Table 3. The mesh indices in the main loops of edge recovery

l_{max}		-1	0	1	2	3	4	5	6	7	8	9	10	11	12
1st calling of Algorithm 6	n_1	243	127	69	40	33	26	25	25	19	15	13	13	13	13
	n_2	1,505	866	243	145	117	82	87	82	63	48	39	41	38	37
	n_3	41	37	27	21	18	14	13	17	13	15	12	13	12	12
2nd calling of Algorithm 6	n_1	13	13	13	11	11	10	10	10	10					
	n_2	37	46	27	13	14	10	10	10	10					
	n_3	12	14	7	3	4	1	1	1	1					
	n_4	20	46	59	62	65	65	65	65	65					

Table 4. Performance data for the sequential mesh generation of the F16 aircraft model

Models	F16
#Boundary nodes	84,992
#Boundary faces	169,964
#Volume nodes	645,939
#Volume elements	3,904,675
#Thin faces	35
#Small angles	4
#Steiner points inserted	0
Time cost of the whole boundary recovery (s)	0.59
Time cost of mesh generation (s)	30.19
Time cost of mesh optimization (s)	226.6
Total time cost of all the steps (s)	256.79

Table 5. Performance data for the parallel mesh generation of the F16 aircraft model.

Models	F16
#Computer cores	32
#Subdomains	242
#Boundary nodes	1,156,192
#Boundary faces	2,312,364
#Volume nodes	21,701,930
#Volume elements	122,535,257
#Thin faces in the original surface	27
#Small angles in the original surface	13
#Thin faces after domain decomposition	15
#Small angles after domain decomposition	1,843
#Steiner points inserted	0
Average time cost of boundary recovery per computer core (s)	0.68
Time cost of all steps, wherein (s)	749.45
Time cost of domain decomposition (s)	344.87
Time cost of mesh generation (s)	48.44
Time cost of mesh repartitioning (s)	94.54
Time cost of quality improvement (s)	261.60