Journal of Zhejiang University SCIENCE A
ISSN 1009-3095 (Print); ISSN 1862-1775 (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

Redesign of a conformal boundary recovery algorithm for 3D Delaunay triangulation*

CHEN Jian-jun, ZHENG Yao‡

(*Center for Engineering and Scientific Computation, School of Computer Science, Zhejiang University, Hangzhou 310027, China*)

E-mail: zdchenjj@yahoo.com.cn; yao.zheng@zju.edu.cn

Received Mar. 5, 2006; revision accepted July 10, 2006

Abstract: Boundary recovery is one of the main obstacles in applying the Delaunay criterion to mesh generation. A standard resolution is to add Steiner points directly at the intersection positions between missing boundaries and triangulations. We redesign the algorithm with the aid of some new concepts, data structures and operations, which make its implementation routine. Furthermore, all possible intersection cases and their solutions are presented, some of which are seldom discussed in the literature. Finally, numerical results are presented to evaluate the performance of the new algorithm.

Key words: Boundary recovery, Delaunay triangulation, Mesh generation, Data structure
doi:10.1631/jzus.2006.A2031 **Document code:** A **CLC number:** TP393

INTRODUCTION

The Delaunay criterion provides a good way to triangulate a given point set. However, the predefined point connectivity is not certainly preserved during the triangulation, and some boundary constraints may be lost in the resulting triangulation. Therefore, the recovery of missing boundaries becomes an important topic.

2D boundary recovery problem turns out to be much easier to resolve in theory and practice than its 3D counterpart. It has been shown that there are certain polyhedrons, e.g. the Schönhardt polyhedron, that cannot be triangulated without adding Steiner points. Moreover, Ruppert and Seidel (1992) proved that it is an NP-complete problem to judge whether a polyhedron can be triangulated without adding Steiner points. Consequently, almost all practically useful boundary recovery algorithms have to consider the problem of where and how to add Steiner points.

Boundary constraints can be recovered in two ways: conformal and constrained. In the conformal recovery, Steiner points are inserted on the constraints, and not removed in the resulting volume meshes; thus some of the missing constraints are recovered as concatenations of sub-constraints. In the constrained recovery, the constraints are exactly the same as the prescribed ones, and no Steiner points are allowed to be left on them.

George *et al*.(1991) proposed a constrained boundary recovery algorithm in the early 1990s based on local transformation operators in conjunction with heuristic rules for inserting Steiner points into the inside of the problem domain. However, it suffers from robustness issues (Liu and Baida, 2000). George *et al*.(2003) presented an alternative constrained boundary recovery algorithm free of such problems. Interestingly, Du and Wang (2004) independently proposed an algorithm based on almost the same idea as that of George's new algorithm. Recently, we have successfully integrated an improved version of the algorithm into our parallel Delaunay mesh generator (Chen, 2006).

‡ Corresponding author

* Project (No. 60225009) supported by the National Natural Science Foundation of China through the National Science Fund for Distinguished Young Scholars

Weatherill and Hassan (1994) first investigated a conformal boundary recovery algorithm by adding points directly at the intersection position between missing boundaries and the current triangulation. Lewis *et al*.(1996) revisited the algorithm when implementing their 3D Delaunay mesh generator. Song *et al*.(2004) improved the algorithm by more detailed discussions about some intersection cases. In this paper, we follow the idea of the above algorithm, and redesign it for our 3D Delaunay mesh generator. Two main contributions of our efforts are presented. First, some new concepts, operations and data structures are designed to make its implementation rather routine, and greatly reduce the coding effort. Second, all intersection cases are examined systematically and their solutions are delivered, which is a prerequisite of a robust boundary recovery algorithm.

Our boundary recovery procedure is divided into two steps. First, missing edges are recovered, and then, missing facets are recovered. After defining some basic concepts, the two steps are described in detail. Finally, experimental results for evaluating the performance of the procedure are presented.

BASIC CONCEPTS

Ball, pipe, shell, and cluster

Define the set of all tetrahedra including a point P as the ball of the point, denoted as $Ball(P)$. Each element of the ball is called a ballel (ball+el).

Define the set of all tetrahedra cut through by an edge E as the pipe of the edge, denoted as $Pipe(E)$. Each element of the pipe is called a pipel (pipe+el).

Define the set of all tetrahedra including an edge E as the shell of the edge, denoted as $Shell(E)$. Each element of the shell is called a shellel (shell+el).

Define the set of all tetrahedra with one or more edges cutting through a facet F as the cluster of the facet, denoted as $Cluster(F)$. Each element of the cluster is called a clusterel (cluster+el). Especially, tetrahedra with one facet coplanar with F are also called clusterels in this paper.

Coding of points, edges, and facets of a tetrahedron

Geometric ingredients of a tetrahedron include 4 forming points, 6 edges and 4 facets, and they are coded for the convenience of programming. A point P is coded with the index that locates the point P in the forming point array of a tetrahedron t, and the code is denoted as $NCode(t,P)$. An edge E is coded according to the codes of its end points, and it is denoted as $ECode(t,E)$, where t is the tetrahedron containing E. The codes of the starting and end points of an edge are stored in No. 1~2 bits and No. 3~4 bits, respectively, see Table 1 for details. The code of a facet F of a tetrahedron t equals the code of the point that the facet does not contain, denoted as $FCode(t,F)$.

Table 1 Coding of edges of a tetrahedron

Edge codes	Edges
4	0→1
8	0→2
12	0→3
9	1→2
13	1→3
14	2→3

Given an ordered set of tetrahedra $T=\{t_1,t_2,\ldots,t_n\}$, denote a forming point of t_j with a capital α. If β is the lowercase of α, let

$$i\beta_j=NCode(t_j,\alpha). \quad (1)$$

Especially, abbreviate $i\beta_1$ as $i\beta$ while n=1.

S-type decomposition and Z-type decomposition of a triangular facet

If two edges of a triangular facet are divided by two Steiner points, there are two schemes to decompose the facet into three smaller triangles, as shown in Fig.1. They are called S-type decomposition and Z-type decomposition, respectively, for the three lines

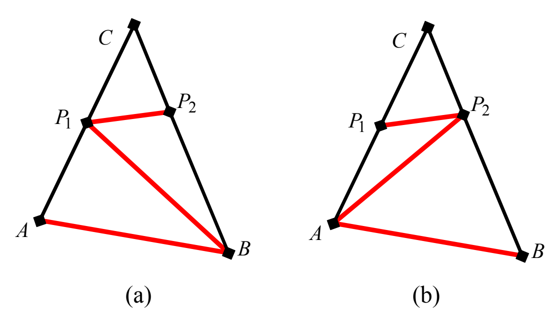

Fig.1 Decomposition schemes of a triangular facet
(a) S-type; (b) Z-type

P_1P_2, BP_1 (or AP_2) and AB constituting a geometry of S and Z shape, respectively. Here, denote the decomposition type of a facet F of a tetrahedron t containing F as $DType(t,F)$.

RECOVERY OF MISSING EDGES

Definition of a pipel

We define a pipel using the C programming language as follows:

```
typedef struct Pipel {
  int iEle;
  int type;
  int iNod1, iNod2;
  int iCod1, iCod2;
  int dectets[MAX_DEC_TETS];
  int nTets;
} Pipel;
```

here, *iEle* points to the position of the pipel in the global element array. *type* identifies the type of the pipel determined by the types of two intersection points between the missing edge and the pipel. The types of intersection points are classified as follows:

```
enum {DEG=0, NOD, EDG, FAC},
```

where *DEG* is a degenerate case of nonexisting intersection point; *NOD*, *EDG*, and *FAC* represent the cases when the intersection point lies in one of the forming points, edges, and facets of the pipel, respectively. According to the above classification for intersection points, there are 11 types of pipels as defined below:

```
#define NOD_NOD ((NOD<<2) | NOD)
#define EDG_NOD ((NOD<<2) | EDG)
#define FAC_NOD ((NOD<<2) | FAC)
#define NOD_EDG ((EDG<<2) | NOD)
#define EDG_EDG ((EDG<<2) | EDG)
#define FAC_EDG ((EDG<<2) | FAC)
#define NOD_FAC ((FAC<<2) | NOD)
#define EDG_FAC ((FAC<<2) | EDG)
#define FAC_FAC ((FAC<<2) | FAC)
#define NOD_DEG ((DEG<<2) | NOD)
#define EDG_DEG ((DEG<<2) | EDG)
```

From the definition, the types of the first and second intersection points of a pipel are stored in No. 1~2 bits and No. 3~4 bits of *type*, respectively. In addition, the decomposition types (S-type or Z-type) of 4 facets of a pipel are stored in No. 5~8 bits of *type*.

Among the 11 types of a pipel defined above, *NOD_NOD* and *NOD_DEG* are degenerate, and not allowed; *EDG_FAC* and *FAC_EDG*, *NOD_FAC* and *FAC_NOD*, *NOD_EDG*, *EDG_NOD* and *EDG_DEG* each can be merged into one type. Therefore, there are 5 types of pipels actually, as shown in Table 2.

Table 2 Five types of pipels

Cases	Type
Case 1	*NOD_EDG/EDG_NOD/EDG_DEG*
Case 2	*EDG_EDG*
Case 3	*NOD_FAC/FAC_NOD*
Case 4	*EDG_FAC/FAC_EDG*
Case 5	*FAC_FAC*

*iNod*1 and *iNod*2 point to the two intersection points, and *iNod*2 is invalid when *type=EDG_DEG*. *iCod*1 and *iCod*2 equal the codes of two geometrical entities (forming points, edges, or facets) of the pipel intersecting with the missing edge. For example, if *type=EDG_FAC*, *iCod*1 is an edge code, and *iCod*2 is a facet code.

dectets stores the indices of newly created tetrahedra in the global element array after decomposing the pipel, and *nTets* is the size of *dectets*.

Recovering a missing edge without adding Steiner points

There are two cases when the missing edge can be recovered by two basic edge/face swap operations, i.e. *Swap*23 and *Swap*44, without requiring Steiner points. These swap operations are shown in Fig.2, where *Swap*32 is for recovering a facet and will be discussed later.

Fig.2a illustrates the *Swap*23 operation, where AB is the missing edge. The key step of the swap is to update neighboring relations of newly created tetrahedra. Denote T an ordered set of tetrahedra composed of $ACDE$ and $BDCE$, Table 3 gives the details of this updating step for *Swap*23, where ic_1, ic_2, id_1,

id_2, ie_1, and ie_2 are codes of corresponding forming points in *ACDE* or *BDCE*. The nomenclature conforms to Eq.(1). For example, id_1=*NCode*(*ACDE*,*D*) as *ACDE* is the first element of *T*, and id_2=*NCode*(*BDCE*,*D*) as *BDCE* is the second element of *T*. Each title of columns 2~5 in Table 3, i.e. *Ni* (*i*=1, 2, 3, 4), indicates No. *i* neighbor of newly created tetrahedra. *NEIG*(*tet*,*k*) (*k*=1, 2, 3, 4) is a basic operation, which returns one of the neighboring tetrahedra of *tet*, and

$$FCode(tet, F)=k,$$

where *F* is the shared facet between *tet* and the returned tetrahedron.

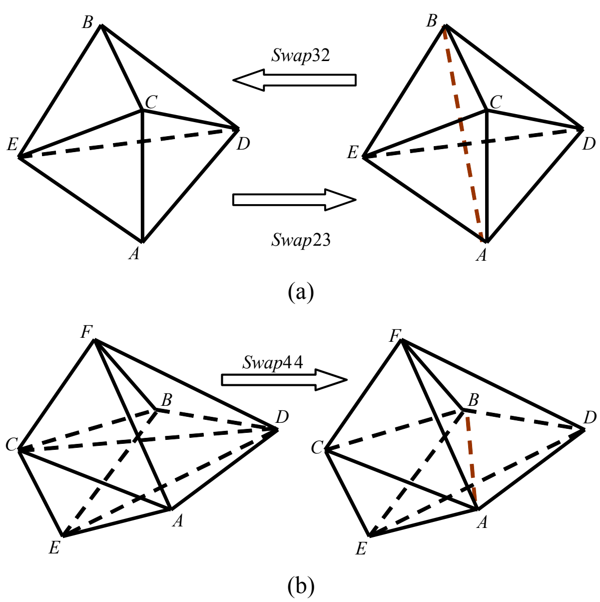

Fig.2 Three basic edge/face swap operations
(a) *Swap*23 and *Swap*32; (b) *Swap*44

Table 3 Update of neighboring relations: *Swap*23 (t_1=*ACDE*, t_2=*BDCE*; *NG*=*NEIG*)

New eles.	*N*1	*N*2	*N*3	*N*4
ACBE	$NG(t_2,id_2)$	*ABDE*	$NG(t_1,id_1)$	*ABCD*
ABCD	$NG(t_2,ie_2)$	$NG(t_1,ie_1)$	*ABDE*	*ACBE*
ABDE	$NG(t_2,ic_2)$	$NG(t_1,ic_1)$	*ACBE*	*ABCD*

Fig.2b explains another basic swap operation, *Swap*44, where *AB* is the missing edge, and *A*, *B*, *C*, and *D* are required to be coplanar. Four tetrahedra of *Pipe*(*AB*) are ordered as *ACDE*, *ADCF*, *BDCE*, and *BCDF*. Table 4 details the updating operations of neighboring relations of newly created tetrahedra for *Swap*44.

Table 4 Update of neighboring relations: *Swap*44 (t_1=*ACDE*, t_2=*ADCF*, t_3=*BDCE*, t_4=*BCDF*; *NG*=*NEIG*)

New eles.	*N*1	*N*2	*N*3	*N*4
ABDE	$NG(t_3,ic_3)$	$NG(t_1,ic_1)$	*ACBE*	*ADBF*
ACBE	$NG(t_3,id_3)$	*ABDE*	$NG(t_1,id_1)$	*ABCF*
ADBF	$NG(t_4,ic_4)$	*ABCF*	$NG(t_2,ic_2)$	*ABDE*
ABCF	$NG(t_4,id_4)$	$NG(t_2,id_2)$	*ADBF*	*ACBE*

Recovering a missing edge by adding Steiner points

While the basic swap operations cannot recover a missing edge, the edge recovery procedure can be done by decomposing all the pipels individually. Decomposition of a pipel will recover a sub-segment of the missing edge. Table 2 lists all types of pipels, and we will discuss their decomposition schemes one by one.

Consider Case 1 first. Here one edge of the pipel is cut through by the missing edge. The decomposition scheme for this case is shown in Fig.3a, and its updating operations of neighboring relations of newly created tetrahedra are listed in Table 5. Φ(*tet*,*vtx*,*fac*) is a basic operation with its execution route being as follows:

(1) Given a tetrahedron *tet*, *V* and *F* are one forming point and one facet of *tet* with codes *vtx* and *fac*, respectively. Let *iNeig*=*NEIG*(*tet*,*vtx*), and set the pipel with *iEle*=*iNeig* as *iPipel*.

(2) If *iNeig* points to a valid tetrahedron and the decomposition operation of *iPipel* has been completed, return the tetrahedron which lies in the array *dectets* of *iPipel* and shares *F* with *tet*; otherwise return a NULL tetrahedron.

Table 5 Update of neighboring relations for the decomposition of the pipel with one edge cut through by the missing edge (t=*ABCD*; *NG*=*NEIG*; $\varphi(vtx,fac)=\Phi(t,vtx,fac)$)

New eles.	*N*1	*N*2	*N*3	*N*4
ABCP	$\varphi(ia,BCP)$	*APCD*	$\varphi(ic,ABP)$	$NG(t,id)$
APCD	$\varphi(ia,PCD)$	$NG(t,ib)$	$\varphi(ic,APD)$	*ABCP*

The pipel of Case 2 has two edges cut through by the missing edge. Define two edges of a tetrahedron as opposite edges if they share no common vertex; otherwise, call them neighboring edges. Therefore, there are two subcases for Case 2, determined by whether the two edges of the pipel are opposite or not. Figs.3b and 3c give their decomposition schemes, named Subcases I and II, respectively. Table 6 details the updating operations of neighboring relations of newly created tetrahedra for Case 2. For Subcase II, ΔABD can be decomposed in the S-type manner or Z-type manner, as shown in Fig.3c. The adoption of the decomposition scheme for ΔABD is determined on the fly as follows:

Step 1: Let *iNeig*=*NEIG*(*ABCD*,*ic*), and set the pipel with *iEle*=*iNeig* as *iPipel*. If *iNeig* points to a NULL tetrahedron, go to Step 3; otherwise go to Step 2;

Step 2: If *iPipel* is not decomposed yet, go to Step 3; otherwise get the value of *DType*(*iNeig*,*ABD*). If it is S-type, return Z-type; otherwise return S-type.

Step 3: Both S-type and Z-type are OK!

For Case 3, one facet of the pipel is cut through by the missing edge. For Case 4, one edge and one facet are cut through by the missing edge. Figs.3d and 3e show their respective decomposition schemes. Table 7 and Table 8 detail their respective updating operations of neighboring relations of newly created tetrahedra.

Two alternative decomposition schemes are available for Case 5, where two facets of the pipel are cut through by the missing edge, as shown in

Table 6 Update of neighboring relations for the decomposition of the pipel with two edges cut through by the missing edge (*t*=*ABCD*; *NG*=*NEIG*; φ(*vtx*,*fac*)=Φ(*t*,*vtx*,*fac*))

Subcases	New eles.	*N*1	*N*2	*N*3	*N*4
I	ABP_2P_1	P_1BP_2D	AP_2CP_1	$\varphi(ic,ABP_1)$	$\varphi(id,BAP_2)$
	AP_2CP_1	P_1P_2CD	$\varphi(ib,AP_1C)$	ABP_2P_1	$\varphi(id,ACP_2)$
	P_1BP_2D	$\varphi(ia,BP_2D)$	P_1P_2CD	$\varphi(ic,P_1BD)$	ABP_2P_1
	P_1P_2CD	$\varphi(ia,P_2CD)$	$\varphi(ib,P_1DC)$	P_1BP_2D	AP_2CP_1
II (S-type)	$ABCP_1$	P_1BCP_2	$\varphi(ib,AP_1C)$	$\varphi(ic,ABP_1)$	$NG(t,id)$
	P_1BCP_2	$\varphi(ia,BCP_2)$	P_1P_2CD	$\varphi(ic,P_1BP_2)$	$ABCP_1$
	P_1P_2CD	$\varphi(ia,P_2CD)$	$\varphi(ib,P_1DC)$	$\varphi(ic,P_1P_2D)$	P_1BCP_2
II (Z-type)	$ABCP_2$	$\varphi(ia,BCP_2)$	AP_2CP_1	$\varphi(ic,ABP_2)$	$NG(t,id)$
	AP_2CP_1	P_1P_2CD	$\varphi(ib,AP_1C)$	$\varphi(ic,AP_2P_1)$	$ABCP_2$
	P_1P_2CD	$\varphi(ia,P_2CD)$	$\varphi(ib,P_1DC)$	$\varphi(ic,P_1P_2D)$	AP_2CP_1

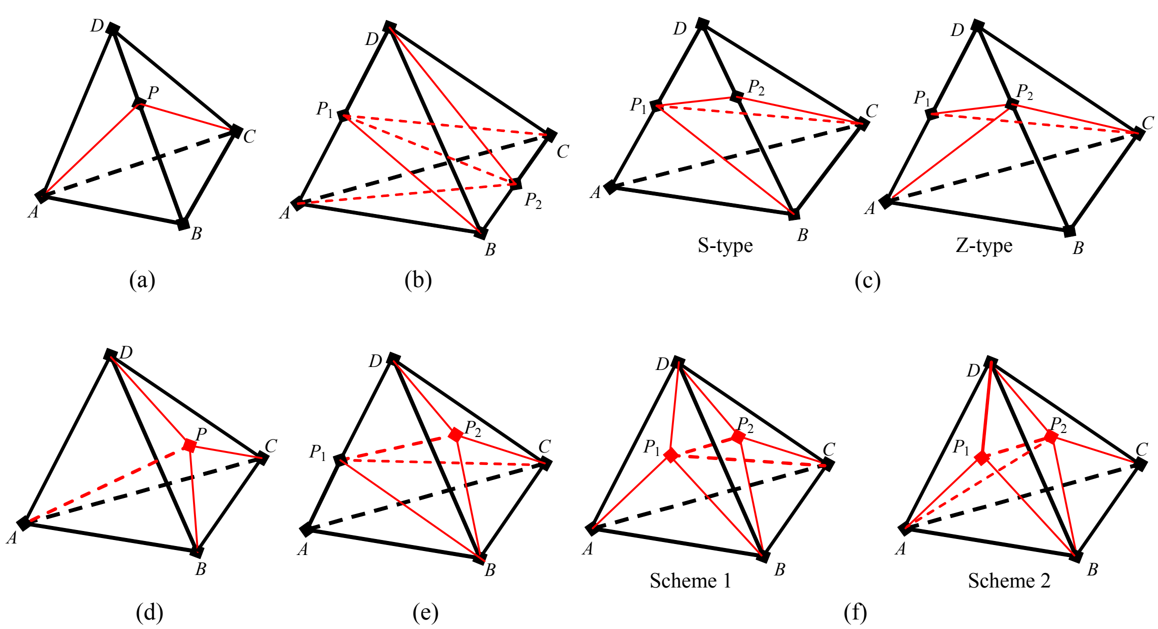

Fig.3 Decomposition schemes for the pipel with (a) one edge, (b) two opposite edges, (c) two neighboring edges, (d) one facet, (e) one edge and one facet, and (f) two facets cut through by the missing edge

Fig.3f. In our program, only Scheme 1, illustrated in the left of Fig.3f, is adopted. The updating operations of neighboring relations of newly created tetrahedra for Scheme 1 are detailed in Table 9.

Table 7 Update of neighboring relations for the decomposition of the pipel with one facet cut through by the missing edge (t=$ABCD$; NG=$NEIG$; $\varphi(vtx,fac)$=$\Phi(t,vtx,fac)$)

New eles.	$N1$	$N2$	$N3$	$N4$
$ABCP$	$\varphi(ia,BCP)$	$APCD$	$ABPD$	$NG(t,id)$
$APCD$	$\varphi(ia,CDP)$	$NG(t,ib)$	$ABPD$	$ABCP$
$ABPD$	$\varphi(ia,DBP)$	$APCD$	$NG(t,ic)$	$ABCP$

Table 8 Update of neighboring relations for the decomposition of the pipel with one edge and one facet cut through by the missing edge (t=$ABCD$; NG=$NEIG$; $\varphi(vtx,fac)$=$\Phi(t,vtx,fac)$)

New eles.	$N1$	$N2$	$N3$	$N4$
$ABCP_1$	P_1BCP_2	$\varphi(ib,AP_1C)$	$\varphi(ic,ABP_1)$	$NG(t,id)$
P_1BP_2D	$\varphi(ia,DBP_2)$	P_1P_2CD	$\varphi(ic,P_1BD)$	P_1BCP_2
P_1BCP_2	$\varphi(ia,BCP_2)$	P_1P_2CD	P_1BP_2D	$ABCP_1$
P_1P_2CD	$\varphi(ia,CDP_2)$	$\varphi(ib,CP_1D)$	P_1BP_2D	P_1BCP_2

Table 9 Update of neighboring relations for the decomposition of the pipel with two facets cut through by the missing edge (t=$ABCD$; NG=$NEIG$; $\varphi(vtx,fac)$=$\Phi(t,vtx,fac)$)

New eles.	$N1$	$N2$	$N3$	$N4$
$ABCP_1$	P_1BCP_2	AP_1CD	$\varphi(ic,ABP_1)$	$NG(t,id)$
P_1BCP_2	$\varphi(ia,BCP_2)$	P_1P_2CD	P_1BP_2D	$ABCP_1$
P_1P_2CD	$\varphi(ia,CDP_2)$	AP_1CD	P_1BP_2D	P_1BCP_2
P_1BP_2D	$\varphi(ia,DBP_2)$	P_1P_2CD	$\varphi(ic,BDP_1)$	P_1BCP_2
AP_1CD	P_1P_2CD	$NG(t,ib)$	$\varphi(ic,DAP_1)$	$ABCP_1$

.

RECOVERY OF MISSING FACETS

Definition of a clusterel

We define a clusterel using the C programming language as follows:

```
typedef struct Clusterel {
    int iEle;
    int type;
    int codes[4], ntypes[4];
    int nodes[4];
    DecTets dectets;
    int nTets;
} Clusterel;
```

here, *iEle* points to the position of the clusterel in the global element array. *type* identifies the type of the clusterel determined by the number of clusterel edges cutting through the missing facet. There are 5 cases for the type of a clusterel, defined using the C programming as follows:

enum {*CO_PLAN*=0, *ONE_EDG*, *TWO_EDG*, *THR_EDG*, *FOU_EDG*},

where *CO_PLAN* refers to the case that a facet of the clusterel is coplanar with the missing facet; *ONE_EDG*, *TWO_EDG*, *THR_EDG*, and *FOU_EDG* refer to the cases of 1, 2, 3, and 4 edges of the clusterel cutting through the missing facet, respectively.

codes stores the codes of edges cutting through the missing facet. *nodes* records the indices of the intersection points between the clusterel and the missing facets. *ntypes* are the types of intersection points, which have 5 cases, depicted as follows:

enum {*NOD_NUL*=0, *NOD_EXT*, *NOD_BEG*, *NOD_END*, *NOD_MID*};

where *NOD_NUL* refers to the case of nonexisting intersection point; *NOD_EXT*, *NOD_BEG*, *NOD_END*, and *NOD_MID* refer to the cases that the intersection points lie in the extension line, the starting point, the end point, and the middle of the edge cutting through the missing facet, respectively.

Similar to the definition of a pipel, here, *dectets* stores the indices in the global element array of newly created tetrahedra after decomposing the clusterel, and *nTets* is the size of *dectets*.

Recovering a missing facet without adding Steiner points

A missing facet can be recovered without adding Steiner points using the basic swap operation *Swap*32, shown in Fig.2a, where ΔECD is the missing facet, and the cluster before the swap operation consists of an ordered set of tetrahedra T={$ACBE$, $ABCD$, $ABDE$}.

Table 10 details the updating operations of neighboring relations of newly created tetrahedra for *Swap*32.

Table 10 Update of neighboring relations: *Swap*32 (t_1=*ACBE*, t_2=*ABCD*, t_3=*ABDE*; *NG*=*NEIG*)

New eles.	*N*1	*N*2	*N*3	*N*4
ACDE	*BDCE*	$NG(t_3,ib_3)$	$NG(t_1,ib_1)$	$NG(t_2,ib_2)$
BDCE	*ACDE*	$NG(t_1,ia_1)$	$NG(t_3,ia_3)$	$NG(t_2,ia_2)$

Recovering a missing facet by adding Steiner points

While a missing facet cannot be recovered using *Swap*32, all clusterels involved should be decomposed individually to recover the missing facet as a concatenation of sub-facets. As defined previously, there are 5 types of clusterels, determined by the values of *type* members. Fig.4 illustrates all the types of clusterels, where Δ*EFG* is the missing facet, and *ABCD* is a clusterel. The clusterel with *type*=*CO_PLAN* need not be decomposed for recovering Δ*ABC*, a sub-facet of Δ*EFG*. Note that the tetrahedron *NEIG*(*ABCD*,*id*) is also a clusterel with *type*=*CO_PLAN*. The decomposition schemes of clusterels with *type*=*ONE_EDG* and *type*=*TWO_EDG* are identical to those of pipels for Case 1 and Case 2, respectively. Therefore, we only need investigate the decomposition schemes for clusterels with 3 or 4 edges cutting through the missing facets.

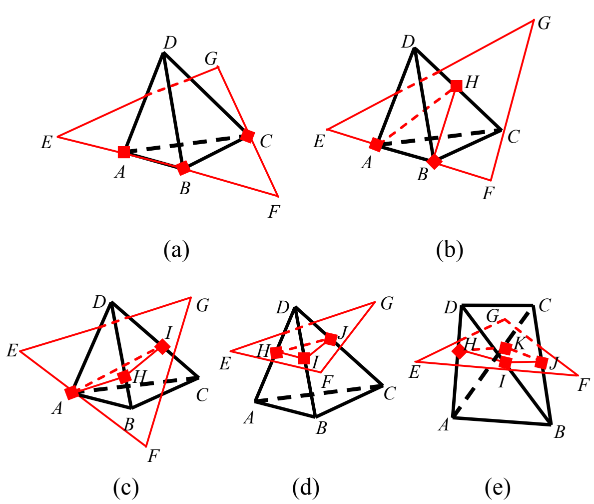

Fig.4 Five types of clusterels. (a) *type*=*CO_PLAN*; (b) *type*=*ONE_EDG*; (c) *type*=*TWO_EDG*; (d) *type*=*THR_EDG*; (e) *type*=*FOU_EDG*

Relabel the intersection points in Fig.4d as P_1, P_2, and P_3. Fig.5 presents 4 types of decomposition schemes for the clusterel with *type*=*THR_EDG*. They are named as *Si*/*Zj*, where *i* and *j* denote the numbers of facets decomposed with S-type and Z-type, respectively. It is obvious that the sum of *i* and *j* equals 3. The selection rule for the decomposition schemes for Δ*ABD*, Δ*BCD*, or Δ*CAD* is identical to those for Δ*ABD* while decomposing the pipel with two neighboring edges cut through by the missing edge, as described previously.

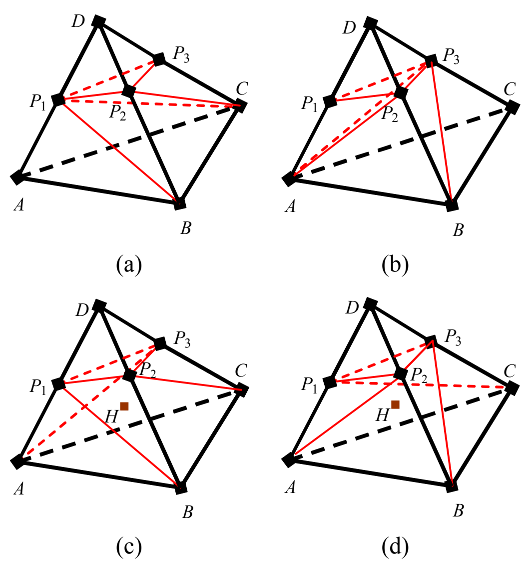

Fig.5 Decomposition schemes for the cluster with three edges cutting through the missing facet
(a) S2/Z1; (b) S1/Z2; (c) S3/Z0; (d) S0/Z3

As shown in Figs.5c and 5d, besides the intersection points, an extra Steiner point, named *H*, should be inserted for S3/Z0 and S0/Z3.

Table 11 details the updating operations of neighboring relations of newly created tetrahedra for decomposition schemes of clusterels with *type*=*THR_EDG*. The minor difference of the definition of *Φ*(*tet*,*vtx*,*fac*) here with that introduced in Tables 5~10 is that *tet* refers to a clusterel rather than a pipel.

There are 6 subcases for the decomposition of clusterels with *type*=*FOU_EDG*, named SSSS, ZSSS, ZZSS, ZSZS, ZZZS, and ZZZZ, respectively, and Fig.6 shows the corresponding decomposition schemes for them. For the name of each subcase, the character *X* (*X*=S or *X*=Z) in the position *i* (*i*=0~3) means that the facet numbered *i* in the clusterel adopts *X*-type decomposition scheme. Accordingly, in Fig.6, Δ*ABD*, Δ*BCD*, Δ*ACB*, and Δ*ADC* are numbered 0~3, respectively. The selection rule for the de-

Table 11 Update of neighboring relations for the decomposition of the clusterel with three edges cutting through the missing facet (*t*=*ABCD*; *NG*=*NEIG*; $\varphi(vtx,fac)=\Phi(t,vtx,fac)$)

Subcases	New eles.	$N1$	$N2$	$N3$	$N4$
S2/Z1	$ABCP_1$	P_1BCP_2	$\varphi(ib,AP_1C)$	$\varphi(ic,ABP_1)$	$NG(t,id)$
	P_1BCP_2	$\varphi(ia,BCP_2)$	$P_1P_2CP_3$	$\varphi(ic,P_1BP_2)$	$ABCP_1$
	$P_1P_2CP_3$	$\varphi(ia,P_2CP_3)$	$\varphi(ib,CP_1P_3)$	$P_1P_2P_3D$	P_1BCP_2
	$P_1P_2P_3D$	$\varphi(ia,P_2P_3D)$	$\varphi(ib,P_1DP_3)$	$\varphi(ic,P_1P_2D)$	$P_1P_2CP_3$
S1/Z2	$ABCP_3$	$\varphi(ia,BCP_3)$	$\varphi(ib,AP_3C)$	ABP_3P_2	$NG(t,id)$
	ABP_3P_2	$\varphi(ia,BP_3P_2)$	$AP_2P_3P_1$	$\varphi(ic,ABP_2)$	$ABCP_3$
	$AP_2P_3P_1$	$P_1P_2P_3D$	$\varphi(ib,AP_1P_3)$	$\varphi(ic,AP_2P_1)$	ABP_3P_2
	$P_1P_2P_3D$	$\varphi(ia,P_2P_3D)$	$\varphi(ib,P_1DP_3)$	$\varphi(ic,P_1P_2D)$	$AP_2P_3P_1$
S3/Z0	AP_1BH	P_1P_2BH	$ABCH$	AP_3P_1H	$\varphi(ic,ABP_1)$
	P_1P_2BH	BP_2CH	AP_1BH	$P_1P_3P_2H$	$\varphi(ic,P_1BP_2)$
	BP_2CH	CP_2P_3H	$ABCH$	P_1P_2BH	$\varphi(ia,BCP_2)$
	CP_2P_3H	$P_1P_3P_2H$	ACP_3H	BP_2CH	$\varphi(ia,P_2CP_3)$
	ACP_3H	CP_2P_3H	AP_3P_1H	$ABCH$	$\varphi(ib,AP_3C)$
	AP_3P_1H	$P_1P_3P_2H$	AP_1BH	ACP_3H	$\varphi(ib,AP_1P_3)$
	$ABCH$	BP_2CH	ACP_3H	AP_1BH	$NG(t,id)$
	$P_1P_3P_2H$	CP_2P_3H	P_1P_2BH	AP_3P_1H	$P_1P_2P_3D$
	$P_1P_2P_3D$	$\varphi(ia,P_2P_3D)$	$\varphi(ib,P_1DP_3)$	$\varphi(ic,P_1P_2D)$	$P_1P_3P_2H$
S0/Z3	AP_1P_2H	$P_1P_3P_2H$	AP_2BH	ACP_1H	$\varphi(ic,AP_2P_1)$
	AP_2BH	BP_2P_3H	$ABCH$	AP_1P_2H	$\varphi(ic,ABP_2)$
	BP_2P_3H	$P_1P_3P_2H$	CBP_3H	AP_2BH	$\varphi(ia,BP_3P_2)$
	CBP_3H	BP_2P_3H	CP_3P_1H	$ABCH$	$\varphi(ia,BCP_3)$
	ACP_1H	CP_3P_1H	AP_1P_2H	$ABCH$	$\varphi(ib,AP_1C)$
	CP_3P_1H	$P_1P_3P_2H$	ACP_1H	CBP_3H	$\varphi(ib,CP_1P_3)$
	$ABCH$	CBP_3H	ACP_1H	AP_2BH	$NG(t,id)$
	$P_1P_3P_2H$	BP_2P_3H	AP_1P_2H	CP_3P_1H	$P_1P_2P_3D$
	$P_1P_2P_3D$	$\varphi(ia,P_2P_3D)$	$\varphi(ib,P_1DP_3)$	$\varphi(ic,P_1P_2D)$	$P_1P_3P_2H$

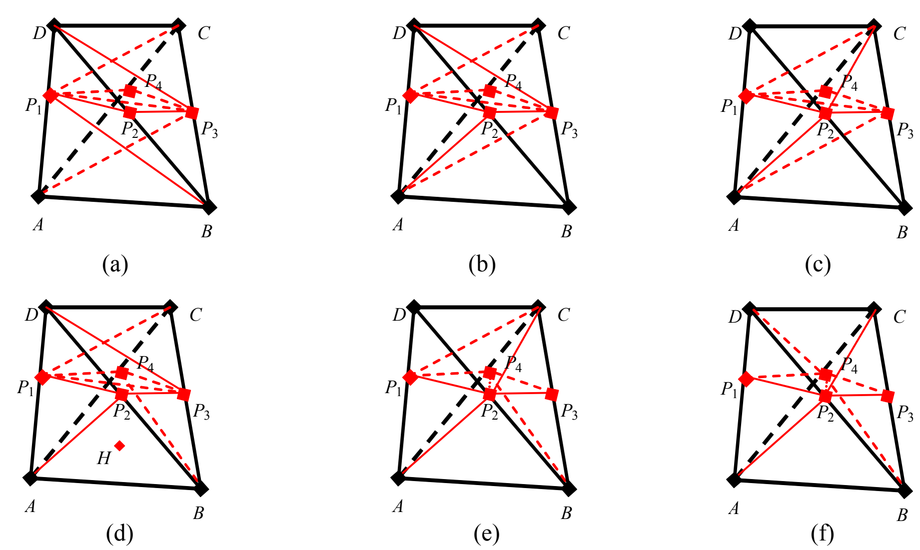

Fig.6 Decomposition schemes for the cluster with four edges cutting through the missing facet
(a) SSSS; (b) ZSSS; (c) ZZSS; (d) ZSZS; (e) ZZZS; (f) ZZZZ

composition scheme of each facet is also identical to that for Δ*ABD* when decomposing the pipel with two neighboring edges cut through by the missing edge, as described previously.

It is observed in Fig.6d that an extra Steiner point besides the intersection points is needed for ZSZS, where the decomposition types of a pair of opposite facets are both Z-type, and those of the other are both S-type.

Table 12 details the updating operations of neighboring relations of newly created tetrahedra for decomposition schemes of clusterels with *type*=*FOU_EDG*.

MISCELLANEOUS ISSUES

Smoothing operations of the surface

As reported in the literature, and also validated

Table 12 Update of neighboring relations for the decomposition of the clusterel with four edges cutting through the missing facet (t=$ABCD$; $\varphi(vtx,fac)=\Phi(t,vtx,fac)$)

Subcases	New eles.	$N1$	$N2$	$N3$	$N4$
SSSS	$P_1P_2P_3D$	$\varphi(ia,P_2P_3D)$	P_1P_3CD	$\varphi(ic,P_1P_2D)$	$P_1P_3P_2B$
	P_1P_3CD	$\varphi(ia,P_3CD)$	$\varphi(ib,P_1DC)$	$P_1P_2P_3D$	$P_1P_3P_4C$
	$P_1P_3P_4C$	$\varphi(id,P_3P_4C)$	$\varphi(ib,P_1CP_4)$	P_1P_3CD	$P_1P_4P_3A$
	$P_1P_3P_2B$	$\varphi(ia,P_3P_2B)$	$\varphi(ic,P_1BP_2)$	ABP_3P_1	$P_1P_2P_3D$
	$P_1P_4P_3A$	$\varphi(id,P_4P_3A)$	ABP_3P_1	$\varphi(ib,AP_1P_4)$	$P_1P_3P_4C$
	ABP_3P_1	$P_1P_3P_2B$	$P_1P_4P_3A$	$\varphi(ic,ABP_1)$	$\varphi(id,AP_3B)$
ZSSS	$P_1P_2P_3D$	$\varphi(ia,P_2P_3D)$	P_1P_3CD	$\varphi(ic,P_1P_2D)$	$P_1P_3P_2A$
	P_1P_3CD	$\varphi(ia,P_3CD)$	$\varphi(ib,P_1DC)$	$P_1P_2P_3D$	$P_1P_3P_4C$
	$P_1P_3P_4C$	$\varphi(id,P_3P_4C)$	$\varphi(ib,P_1CP_4)$	P_1P_3CD	$P_1P_4P_3A$
	$P_1P_3P_2A$	ABP_3P_2	$\varphi(ic,AP_2P_1)$	$P_1P_4P_3A$	$P_1P_2P_3D$
	$P_1P_4P_3A$	$\varphi(id,P_4P_3A)$	$P_1P_3P_2A$	$\varphi(ib,AP_1P_4)$	$P_1P_3P_4C$
	ABP_3P_2	$\varphi(ia,P_3P_2B)$	$P_1P_3P_2A$	$\varphi(ic,ABP_2)$	$\varphi(id,AP_3B)$
ZZSS	P_1P_2CD	$\varphi(ia,P_2CD)$	$\varphi(ib,P_1DC)$	$\varphi(ic,P_1P_2D)$	$P_1P_2P_3C$
	$P_1P_2P_3C$	$\varphi(ia,P_2P_3C)$	$P_1P_3P_4C$	P_1P_2CD	$P_1P_3P_2A$
	$P_1P_3P_4C$	$\varphi(id,P_3P_4C)$	$\varphi(ib,P_1CP_4)$	$P_1P_2P_3C$	$P_1P_4P_3A$
	$P_1P_3P_2A$	ABP_3P_2	$\varphi(ic,AP_2P_1)$	$P_1P_4P_3A$	$P_1P_2P_3C$
	$P_1P_4P_3A$	$\varphi(id,P_4P_3A)$	$P_1P_3P_2A$	$\varphi(ib,AP_1P_4)$	$P_1P_3P_4C$
	ABP_3P_2	$\varphi(ia,P_3P_2B)$	$P_1P_3P_2A$	$\varphi(ic,ABP_2)$	$\varphi(id,AP_3B)$
ZSZS	$P_1P_2P_3D$	$\varphi(ia,P_2P_3D)$	P_1P_3CD	$\varphi(ic,P_1P_2D)$	$P_1P_3P_2H$
	P_1P_3CD	$\varphi(ia,P_3CD)$	$\varphi(ib,P_1DC)$	$P_1P_2P_3D$	$P_1P_3P_4C$
	$P_1P_3P_4C$	$\varphi(id,P_3P_4C)$	$\varphi(ib,P_1CP_4)$	P_1P_3CD	$P_1P_4P_3H$
	$P_1P_4P_3H$	BP_3P_4H	$P_1P_3P_2H$	AP_4P_1H	$P_1P_3P_4C$
	$P_1P_3P_2H$	BP_2P_3H	AP_1P_2H	$P_1P_4P_3H$	$P_1P_2P_3D$
	AP_4P_1H	$P_1P_4P_3H$	AP_1P_2H	ABP_4H	$\varphi(ib,AP_1P_4)$
	BP_2P_3H	$P_1P_3P_2H$	BP_3P_4H	AP_2BH	$\varphi(ia,P_3P_2B)$
	AP_1P_2H	$P_1P_3P_2H$	AP_2BH	AP_4P_1H	$\varphi(ic,AP_2P_1)$
	AP_2BH	BP_2P_3H	ABP_4H	AP_1P_2H	$\varphi(ic,ABP_2)$
	ABP_4H	BP_3P_4H	AP_4P_1H	AP_2BH	$\varphi(id,P_4BA)$
	BP_3P_4H	$P_1P_4P_3H$	ABP_4H	BP_2P_3H	$\varphi(id,P_4P_3B)$
ZZZS	P_1P_2CD	$\varphi(ia,P_2CD)$	$\varphi(ib,P_1DC)$	$\varphi(ic,P_1P_2D)$	$P_1P_2P_4C$
	$P_1P_2P_4C$	$P_2P_3P_4C$	$\varphi(ib,P_1CP_4)$	P_1P_2CD	$P_1P_4P_2A$
	$P_2P_3P_4C$	$\varphi(id,P_3P_4C)$	$P_1P_2P_4C$	$\varphi(ia,P_2P_3C)$	$P_2P_4P_3B$
	$P_1P_4P_2A$	ABP_4P_2	$\varphi(ic,AP_2P_1)$	$\varphi(ib,AP_1P_4)$	$P_1P_2P_4C$
	$P_2P_4P_3B$	$\varphi(id,P_4P_3B)$	$\varphi(ia,P_3P_2B)$	ABP_4P_2	$P_2P_3P_4C$
	ABP_4P_2	$P_2P_4P_3B$	$P_1P_4P_2A$	$\varphi(ic,ABP_2)$	$\varphi(id,P_4BA)$
ZZZZ	$P_1P_2P_4D$	P_2CP_4D	$\varphi(ib,P_1DP_4)$	$\varphi(ic,P_1P_2D)$	$P_1P_4P_2A$
	$P_2P_3P_4C$	$\varphi(id,P_3P_4C)$	P_2CP_4D	$\varphi(ia,P_2P_3C)$	$P_2P_4P_3B$
	P_2CP_4D	$\varphi(ib,CP_4D)$	$P_1P_2P_4D$	$\varphi(ia,P_2CD)$	$P_2P_3P_4C$
	$P_1P_4P_2A$	ABP_4P_2	$\varphi(ic,AP_2P_1)$	$\varphi(ib,AP_1P_4)$	$P_1P_2P_4D$
	$P_2P_4P_3B$	$\varphi(id,P_4P_3B)$	$\varphi(ia,P_3P_2B)$	ABP_4P_2	$P_2P_3P_4C$
	ABP_4P_2	$P_2P_4P_3B$	$P_1P_4P_2A$	$\varphi(ic,ABP_2)$	$\varphi(id,P_4BA)$

by our experience, smoothing operations of the surface usually helps decrease the number of Steiner points added in the boundary recovery procedure, and hence to improve the element quality near the boundaries. Diagonal swap is frequently used, where the Delaunay criterion is employed for a quadrilateral composed of two neighboring triangular facets, as shown in Fig.7. Two conditions must be satisfied to perform a diagonal swap operation, i.e.

(1) ΔABC and ΔACD must be coplanar; and

(2) The circumcircle of ΔABC contains point D.

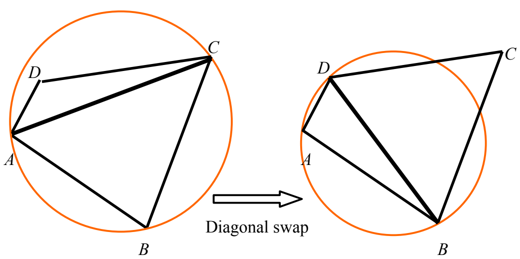

Fig.7 Diagonal swap

Removal of outer elements

A necessary step for Delaunay meshing algorithms is to remove tetrahedra outside of the problem domain after the boundary recovery, for which the coloring algorithm is usually adopted. However, one prerequisite for the coloring algorithm is that the tetrahedra sharing the prescribed surface facets, or sub-facets formed in the boundary recovery procedure, should be labeled as *OUTER* or *INNER*, representing the cases that the corresponding tetrahedra lie outside or inside the problem domain, respectively. The labeling operations are commonly performed concurrently with the recovery procedure for missing facets.

First, we assume two hypotheses in the following discussion.

(1) Forming points of prescribed surface facets are ordered such that the normal vectors of all facets calculated with the right-hand rule point outwards;

(2) Forming points of all tetrahedra are ordered, so that for each tetrahedron, the normal vector of the facet containing the previous 3 forming points calculated with the right-hand rule points to the 4th forming point of the tetrahedron.

Suppose ΔABC is a prescribed surface facet, and *ABCD* is one of the tetrahedra containing it, the *OUTER* or *INNER* property of *ABCD*, denoted with *etype*, can be computed with the following mapping relations, as shown in Table 13.

$$etype=FLAGE(ia,\ ib,\ ic,\ id), \tag{2}$$

where *ia*, *ib*, *ic* and *id* are the codes of the points *A*, *B*, *C*, and *D* in *ABCD*, respectively.

Table 13 Mapping relations between the *OUTER*/*INNER* property of a tetrahedron and codes of its forming points

id	*ia*	*ib*	*ic*	*etype*
0	1/3/2	2/1/3	3/2/1	*OUTER*
	3/1/2	2/3/1	1/ 2/3	*INNER*
1	0/2/3	3/0/2	2/3/0	*OUTER*
	2/0/3	3/2/0	0/3/2	*INNER*
2	0/3/1	1/0/3	3/1/0	*OUTER*
	3/0/1	1/3/0	0/1/3	*INNER*
3	0/1/2	2/0/1	1/2/0	*OUTER*
	1/0/2	2/1/0	0/2/1	*INNER*

Suppose ΔEFG is a prescribed surface facet, ΔABC is a recovered sub-facet of ΔEFG, and *ABCD* is one of the tetrahedra containing ΔABC, the *OUTER* or *INNER* property of *ABCD*, denoted with *etype*, can be computed with a two-step procedure. First, get an initial value of *etype* using Eq.(2); and then compare the normal vector of ΔABC with that of ΔEFG; if they are of opposite directions, reverse *etype*.

NUMERICAL EXPERIMENTS

The boundary recovery procedure presented above has been integrated into our 3D Delaunay mesh generator. It is fairly robust and efficient even for very complex geometries. Figs.8~10 show some volume mesh examples generated using this generator. Table 14 presents some statistics for the mesh examples and the boundary recovery procedure. It is observed that the basic swap operations, i.e. *Swap*23, *Swap*44, and *Swap*32, can recover most of the missing boundaries, however, they may fail for certain missing boundaries, for which Steiner points have to be added. Time performance data illustrate that most of time for our Delaunay mesh generator is spent on the procedures of inserting boundary nodes, creating field points and inserting them. Nevertheless time spent in the boundary recovery might not be omitted

Table 14 Statistics for the mesh examples and the boundary recovery procedure

Statistics	Example 1	Example 2	Example 3
No. of surface nodes	7443	4961	6708
No. of surface facets	14890	9962	13420
No. of volume mesh nodes	12858	7856	7932
No. of volume mesh elements	57704	33182	26868
No. of missing edges	112	39	60
No. of recovered edges by swapping (and its ratio to the total No. of missing edges (%))	101 (90.18)	33 (84.61)	49 (81.67)
No. of added points for the edge recovery	24	13	21
No. of missing facets	23	71	64
No. of recovered facets by swapping (and its ratio to the total No. of missing facets (%))	20 (86.96)	50 (70.42)	52 (81.25)
No. of added points for the facet recovery	6	37	18
Total elapsed time for the mesh generator (s)	3.984	2.140	8.562
Time for the boundary recovery (s) (and its ratio to the total elapsed time (%))	0.687 (17.24)	0.422 (19.72)	0.172 (2.01)

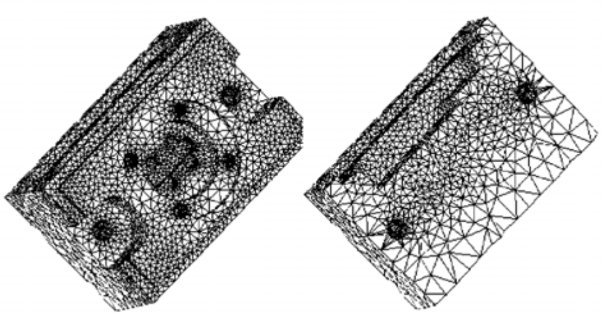

Fig.8 Mesh example 1

Fig.9 Mesh example 2

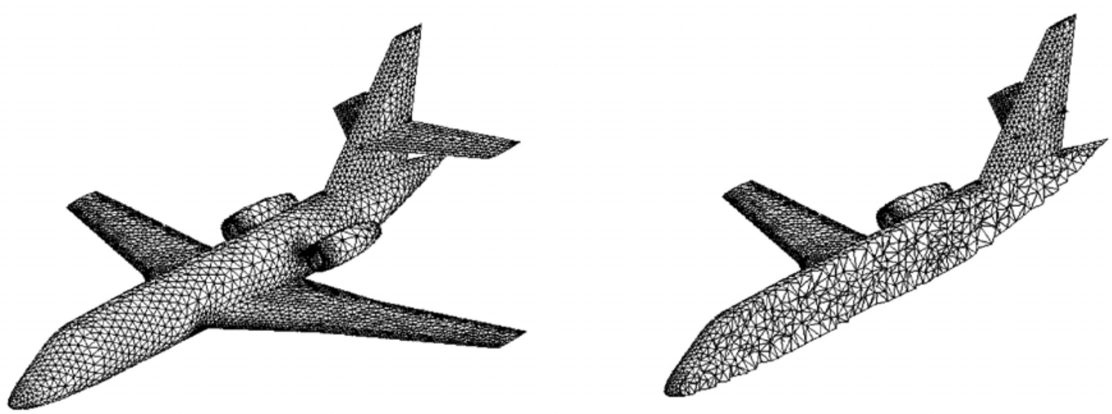

Fig.10 Mesh example 3

for some configurations, the ratio of which to the total elapsed time varies greatly from one configuration to another.

The surface mesh configuration for Example 1 (Fig.8) is smoothed, for which totally 30 Steiner points are added in the boundary recovery. However, the number grows to 59 if the mesh configuration is not smoothed. Here, only the diagonal swap is employed to smooth the surface mesh, and 214 swap operations are executed for this configuration.

CONCLUSION AND REMARKS

A classic conformal boundary recovery algorithm, where Steiner points are added directly in the intersection positions between missing boundaries and triangulations, is redesigned. Local transformation operations are integrated to improve boundary recovery results. The coding procedure of such an algorithm is usually dry and error-prone, however, it could become a rather routine and easy work with the help of some new concepts, data structures, and operations introduced in this paper. Moreover, all cases of Steiner point insertion are discussed, and their solutions are suggested, which highly enhances the robustness of our boundary recovery algorithm.

Element quality near boundaries is a key for the accuracy and/or convergence of the solution process for numerical simulations. It is our future work to improve it.

References

Chen, J.J., 2006. Unstructured Mesh Generation and Its Parallelization. Ph.D Thesis, College of Computer Science, Zhejiang University (in Chinese).

Du, Q., Wang, D.S., 2004. Constrained boundary recovery for three dimensional Delaunay triangulations. *International Journal for Numerical Methods in Engineering*, **61**(9):1471-1500. [doi:10.1002/nme.1120]

George, P.L., Hecht, F., Saltel, E., 1991. Automatic mesh generator with specified boundary. *Computer Methods in Applied Mechanics and Engineering*, **92**(3):269-288. [doi:10.1016/0045-7825(91)90017-Z]

George, P.L., Borouchaki, H., Saltel, E., 2003. 'Ultimate' robustness in meshing an arbitrary polyhedron. *International Journal for Numerical Methods in Engineering*, **58**(7):1061-1089. [doi:10.1002/nme.808]

Lewis, R.W., Zheng, Y., Gethin, D.T., 1996. Three-dimensional unstructured mesh generation: part 3. volume meshes. *Computer Methods in Applied Mechanics and Engineering*, **134**(3-4):285-310. [doi:10.1016/0045-7825(95)00918-3]

Liu, A., Baida, M., 2000. How Far Flipping Can Go towards 3D Conforming/Constrained Triangulations. Proceedings of the 11th International Meshing Roundtable, New Orleans, Louisiana, USA, p.307-315.

Ruppert, J., Seidel, R., 1992. On the difficulty of triangulating three-dimensional non-convex polyhedra. *Discrete and Computational Geometry*, **7**(3):227-254.

Song, C., Guan, Z.Q., Gu, Y.X., 2004. Boundary restore algorithm and sliver elimination of 3D constrained Delaunay triangulation. *Chinese Journal of Computational Mechanics*, **21**(2):169-176 (in Chinese).

Weatherill, N.P., Hassan, O., 1994. Efficient three-dimensional Delaunay triangulation with automatic point creation and imposed boundary constraints. *International Journal for Numerical Methods in Engineering*, **37**(12):2005-2039. [doi:10.1002/nme.1620371203]