

# An approach to achieving optimized complex sheet inflation under constraints

## Abstract

Sheet inflation is an enhanced and more general version of the classic pillowing procedure[1] used to modify hexahedrdal meshes. The flexibility of sheet inflation makes it a valuable tool for hex mesh generation, modification and topology optimization. However, it is still difficult to generate self-intersecting sheet within a local region while assuring the mesh quality. This paper proposes an approach to achieving optimized complex sheet inflation under various constraints. To determine a qualified inflatable quad set, the approach firstly constructs the boundary loop and the initial quad set by using max-flow-min-cut algorithm, and then improves the quality of the initial quad set by a chord-based optimization method. Our approach can generate complex sheets that intersect themselves more than once and guarantee the quality of the resultant mesh. We successfully apply this approach to mesh matching and mesh boundary optimizing.

**Keywords:** hex mesh; sheet inflation; mesh optimization; mesh modification; mesh matching

## 1. Introduction

In finite element analysis, hex meshes are usually preferred to tetrahedral meshes due to higher accuracy, faster convergence and lighter storage[2, 3]. Therefore, many researchers have been committed to the research of hex meshing for decades. Although there has been tremendous progress in this area, perfect solutions are still eluding for hex mesh generation, modification and topological optimization. The main reason for the difficulties is the inherent global connectivity which makes any local modification often inevitably propagate to the whole mesh[4, 5, 6, 7].

In recent years, sheet operations attract more and more attention and have been applied in different situations for their effectiveness in dealing with global structures. The most common sheet operations include sheet inflation, sheet extraction, dicing and column collapse. Unlike other operations that only deal with existing sheets, sheet inflation can be utilized to create new ones with various shapes and topologies, making it irreplaceable in hex mesh generation, especially in topological modification and optimization. The common procedures of sheet inflation are shown in Fig. 1.

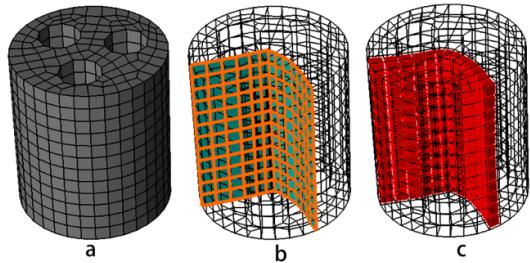


Figure 1: Sheet inflation procedures: (a) the original hex mesh; (b) the quad set; (c) the new sheet inflated from the quad set.

Due to its versatile ability to create various types of sheets

with different shapes and topologies, it is critical for sheet inflation to be controllable in order to meet various mesh modification demands. In practice, the controllability can be fulfilled by allowing the user to setup constraints on the mesh to guide the inflation process. These constraints are usually specified by a set of boundary edges and a set of hexahedra. The former determines the positions where the new sheet should appear on the mesh boundary, and the latter delimits the region where the new sheet can propagate. As shown in Fig. 2(b), a set of boundary edges(green) and a set of hex(yellow) are specified to control the shape and position of the new sheet. One satisfactory sheet is shown in Fig. 2(d).

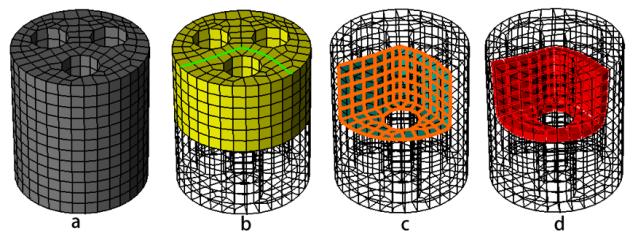


Figure 2: Sheet inflation under user-specified constraints: (a) the original hex mesh; (b) the user-specified boundary edges (green) and hex set (yellow); (c) the internal quad set; (d) the new sheet.

As seen in Fig. 1 and Fig. 2, the quad set is critical to sheet inflation since it determines both the position and topology of the sheet to be inflated. Therefore, to achieve the qualified sheet inflation, the main task, which is also the most difficult, is to find an internal quad set that satisfies the constraints. As shown in Fig. 2, after the user specifies the constraints in Fig. 2(b), the most critical step to generate the qualified sheet is to determine the internal quad set within the specified hex set that intersects with the mesh boundary exactly at the specified edges, as shown in Fig. 2(c).

In recognizing the importance of sheet inflation, many re-

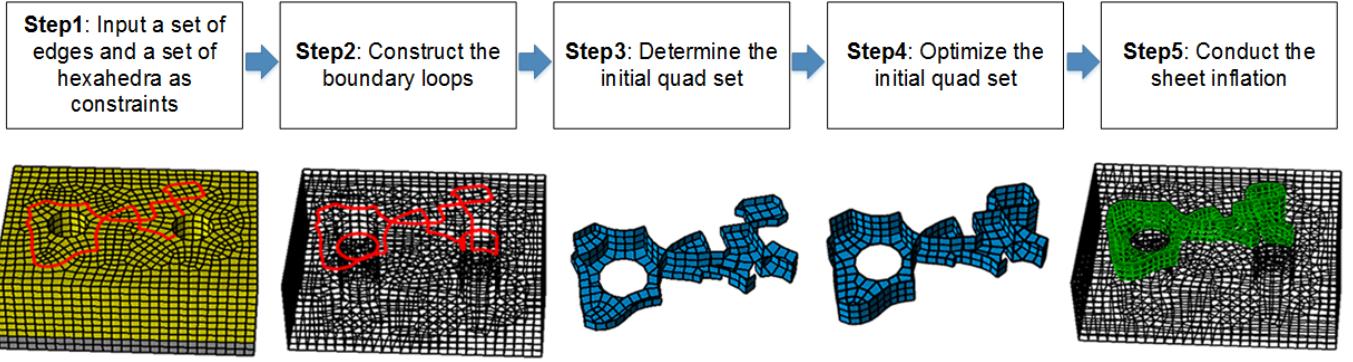


Figure 3: Overview of the approach.

46 researchers have been investigating algorithms for decades and  
 47 have achieved great progress. Mitchell et al. proposed the Pil-  
 48 lowing algorithm to deal with the doublet problem[1]. This  
 49 algorithm involves inflating a new sheet guided by a hex set  
 50 called shrinking set. Its simplicity and effectiveness makes Pil-  
 51 lowing prevalent in mesh modification, especially in mesh qual-  
 52 ity improvement. Although it can be adapted to satisfy simple  
 53 constraints, the Pillowing lacks the ability to generate complex  
 54 sheets such as self-intersecting sheets.

55 Suzuki et al. introduced a method to construct interior sheet  
 56 surfaces when the boundary dual cycles are given[8]. They used  
 57 this method to firstly construct dual structures and then deduce  
 58 the primal hex meshes from these dual structures. While in  
 59 the paper they illustrated the topological structures of sheet sur-  
 60 faces in the dual space, particularly self-intersecting sheet sur-  
 61 faces, they didn't mention how to determine the quad set and  
 62 generate the corresponding sheets on the primal hex mesh.

63 Staten et al. proposed the General Sheet Inflation method  
 64 which explained in detail how to do sheet inflation on hex  
 65 meshes when boundary mesh edges are specified[9]. This  
 66 method can create normal, self-touching and self-intersecting  
 67 sheets. However, determining the quad sets for sheet inflation  
 68 in this method seems to rely on the sweeping structures of the  
 69 mesh. Meanwhile it is not clear whether the method can create  
 70 self-intersecting sheets within a local region.

71 Chen et al. proposed a new approach to inflate sheets when  
 72 conducting mesh matching on complex interfaces[10]. This  
 73 method firstly constructs the boundary loop of the quad set, and  
 74 then determines the interior quad set. Although it can locally  
 75 generate self-intersecting sheets under constraints, it allows the  
 76 sheets to intersect themselves once at most. Meanwhile, the op-  
 77 timization method it applies on quad set cannot guarantee the  
 78 quality improvement.

79 Despite these achievements, to inflate complex sheets un-  
 80 der various constraints, especially to generate self-intersecting  
 81 sheets within a local region, is still very difficult. Furthermore,  
 82 how to assure the mesh quality for complex sheet inflation is  
 83 also a challenging problem. In this paper we propose a new  
 84 approach to achieving complex sheet inflation. By firstly deter-  
 85 mining the boundary loop and then utilizing the max-flow-min-  
 86 cut algorithm, we construct an initial quad set that fulfills all the

87 constraints. By applying a chord-based optimization method,  
 88 we improve the quality of the quad set and thus the mesh qual-  
 89 ity after sheet inflation is guaranteed.

90 The rest of this paper is organized as below: we will show  
 91 the outline of the approach in the second chapter, explain the  
 92 procedures of constructing the boundary loops of quad sets in  
 93 the third chapter, provide the details of the determination and  
 94 optimization of the initial quad set in the fourth and fifth chap-  
 95 ters respectively, and then give some results in the sixth chapter  
 96 and list conclusions and future work in the last chapter.

## 97 2. Overview of the Approach

98 In this paper, an approach is proposed to achieving com-  
 99 plex sheet inflation under various constraints while assuring the  
 100 mesh quality. It takes a set of boundary mesh edges and a set  
 101 of hexahedra as input. The edges and hexahedra specify the  
 102 boundary position and local region of the sheet. Under these  
 103 constraints, it constructs a qualified quad set and then creates  
 104 the new sheet as output. To avoid the difficulty of determining  
 105 a qualified inflatable quad set directly, our approach involves  
 106 three major steps: (a) determining the boundary loops for the  
 107 quad set that satisfy the boundary constraints; (b) constructing  
 108 the initial quad set based on the boundary loops; (c) optimiz-  
 109 ing the initial quad set. Figure 3 shows the overview of our  
 110 approach.

111 Our approach needs to solve two critical problems. The first  
 112 problem is how to construct the valid boundary loop and ini-  
 113 tial quad set satisfying various boundary constraints. Bound-  
 114 ary constraint specified by the user can be a loop or just a  
 115 set of edges, be non-self-intersecting or intersecting itself more  
 116 than once. The second problem is how to effectively optimize  
 117 the initial quad set when a global optimal result can hardly be  
 118 reached.

119 To solve these two problems, our basic ideas are:

- 120 1. We use path searching and max-flow-min-cut algorithms  
 121 to construct the valid boundary loop and initial quad set  
 122 that meet the various boundary constraints. In accordance  
 123 with the inherent characteristics of valid boundary loops  
 124 and quad sets, loops can be formed by the path searching  
 125 algorithm if the boundary constraints are not loops, and

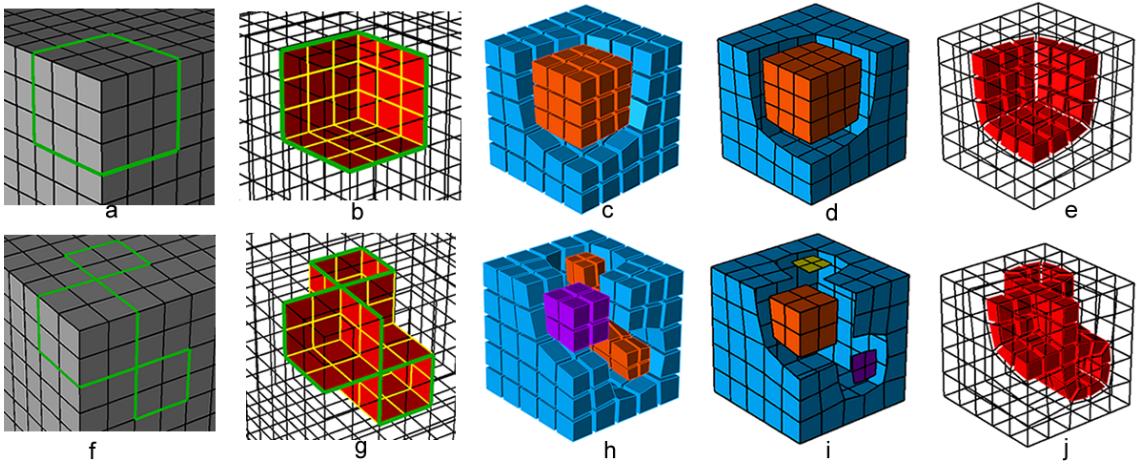


Figure 4: Characteristics of valid boundary loops: (a) a non-self-intersecting boundary loop; (b) the non-self-intersecting quad set; (c) two hex sets separated by the quad set; (d) two boundary quad sets separated by the boundary loop; (e) the new non-self-intersecting sheet; (f) a self-intersecting boundary loop; (g) the self-intersecting quad set; (h) three hex sets separated by the quad set; (i) four boundary quad sets separated by the boundary loop; (j) the new self-intersecting sheet.

126 intersecting lines on the quad set can also be determined  
 127 by the path searching algorithm. The valid boundary loop  
 128 and initial quad set can be effectively determined by the  
 129 max-flow-min-cut algorithm due to the fact that the deter-  
 130 mination of these structures on the mesh is analogous to  
 131 finding a cut set on a graph;

132 2. We optimize the initial quad set by optimizing its chords.  
 133 The chords, the dual structures of the quad set, are more  
 134 global than individual edges while less constrained than  
 135 the whole quad set. By optimizing the chords, we can ef-  
 136 fectively improve the quality of the quad set while avoid-  
 137 ing the difficulty in optimizing the quad set as a whole.

138 In the following sections specific details of the major steps  
 139 are provided.

### 140 3. Determination of the Boundary Loops

141 It is quite difficult to find a qualified quad set satisfying all  
 142 the constraints directly. The quad set's boundary loop is ac-  
 143 cordingly constructed at first. This section firstly discusses the  
 144 characteristics of valid boundary loops and quad sets, and then  
 145 presents the method to construct a valid boundary loop under  
 146 the boundary constraints.

147 By observation of the structures of valid quad sets and the  
 148 process of sheet inflation, an inflatable quad set and its bound-  
 149 ary loop have two characteristics:

150 1. An inflatable quad set separates its local hex set into  $2 + N$   
 151 subsets, where  $N$  stands for how many times the quad  
 152 set intersects itself. For example, the non-self-intersecting  
 153 quad set in Fig. 4(b) separates the local hex set into two parts (Fig. 4(c)). The self-intersecting quad set in Fig.  
 154 4(g) separates the local hex set into 3 subsets (Fig. 4h). Mean-  
 155 while, the boundary loop of the quad set separates the bound-  
 156 ary of the local hex set into  $2 + n$  subsets, where  
 157  $n$  is the number of self-intersecting nodes on the boundary

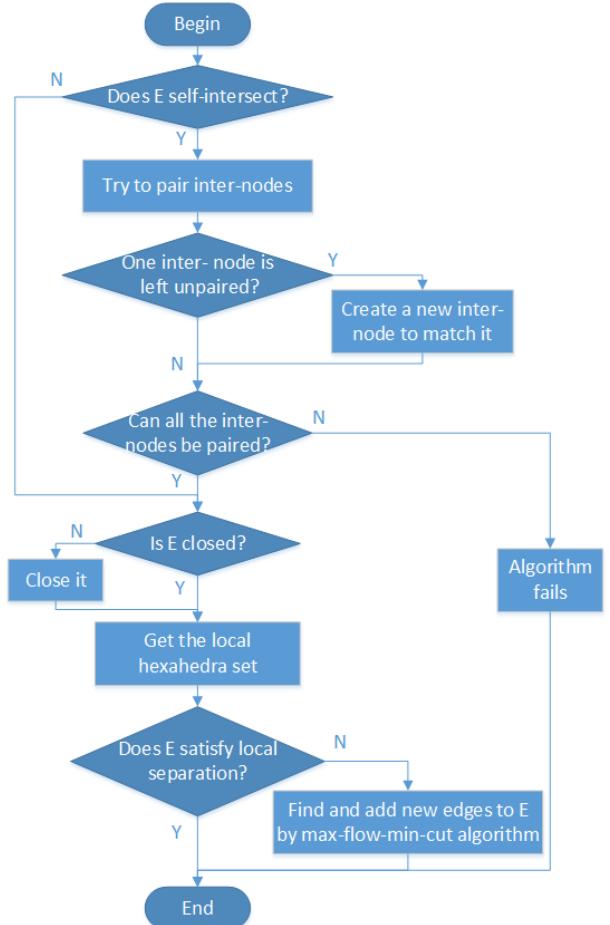


Figure 5: Flowchart of the determination of boundary loops.

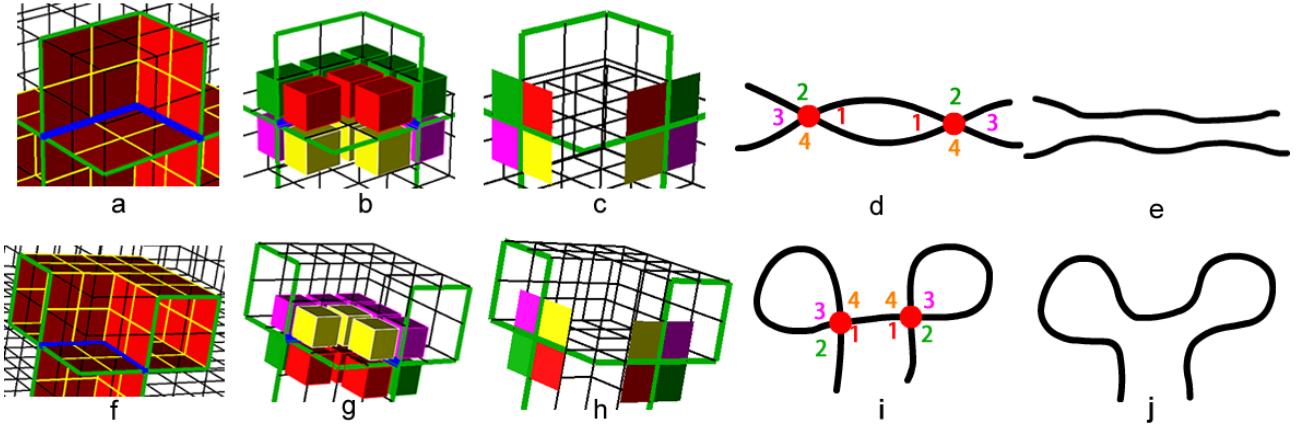


Figure 6: Two local templates for pairing intersecting nodes: (a) the 1st local intersecting structure of the quad set; (b) int-4-hex-sets around the 1st local intersecting structure of the quad set; (c) the associated local structure of the boundary loop; (d) the 1st template; (e) intersecting nodes are resolved after pairing; (f) the 2nd local intersecting structure of the quad set; (g) int-4-hex-sets around the 2nd local intersecting structure of the quad set; (h) the associated local structure of the boundary loop; (i) the 2nd template; (j) intersecting nodes are resolved after pairing.

loop. The non-self-intersecting boundary loop in Fig. 4(a) separates the boundary of the local hex set into two subsets (Fig. 4(d)), and the self-intersecting boundary loop in Fig. 4(f) separates the boundary of the local hex set into 4 subsets (Fig. 4(i)).

2. Intersecting nodes on the boundary loop can be grouped into pairs. An intersecting line on the quad set can be found between two nodes in each pair.

The first characteristic, which is also called local separation, is necessary for a quad set to be inflatable, because the sheet inflation operation is actually done by separating the hex subsets and filling the gaps with new hexahedra, as shown in Fig. 4(c), Fig. 4(e), Fig. 4h and Fig. 4(j). It also indicates that the boundary loops must be closed otherwise new hexahedra on the mesh boundary will not be correctly created. The second characteristic also indicates that each intersecting node on the boundary loop needs another intersecting node to be paired in order to determine an intersecting line.

Based on the above analysis, we check whether the specified constraint edges are eligible to be a valid boundary loop. If not, we use the path searching algorithm and max-flow-min-cut algorithm to determine a valid boundary loop. The flowchart of the determination of boundary loops is shown in Fig. 5. The rest of this section will provide detailed description about the method of the determination of boundary loops.

bring out two templates for pairing intersecting nodes. By recursively applying these two templates, a proper pairing between intersecting nodes can be achieved.

These two templates come from two simple self-intersecting quad sets as shown in Fig. 6(a) and Fig. 6(f). These two quad sets both have one intersecting line and two intersecting nodes on their boundary loops. Topological structures of their boundary loops form the two templates as shown in Fig. 6(d) and Fig. 6(i). If a part of the boundary loop has the same topological structure as either of the two templates, there must be a quad set with the same topological structure of the corresponding simple self-intersecting quad set. This template-based pairing method thus can effectively avoid improper pairing result mentioned in [8]. To recursively find next pair of intersecting nodes, the local topological structures are modified accordingly as shown in Fig. 6(e) and Fig. 6(j).

Some structures near the intersecting lines or intersecting nodes are very important for determining the quad set. The hexahedra adjacent to the intersecting lines are separated into four subsets by the local structures of quad sets, as shown in Fig. 6(b) and Fig. 6(g). These hex subsets are called int-4-hex-sets of the intersecting line. The associated local structures of boundary loops are shown in Fig. 6(c) and Fig. 6(h). The quads in same color mean they are adjacent to the same int-4-hex-set. The numbers in different colors on the two templates in Fig. 6(d) and Fig. 6(i) indicate the corresponding int-4-hex-sets.

Two examples in Fig. 7 are presented to explain how to pair intersecting nodes by these two templates.

The pairing process for the 1st example is shown in Fig. 8. Firstly the intersecting nodes A and B are paired by the 1st template. After topological structure is modified, C and D are also paired by the 1st template.

The pairing process for the 2nd example is shown in Fig. 9. Firstly the intersecting nodes A and B are paired by the 2nd template. After topological structure is modified, the nodes C and D are paired by the 1st template.

After recursively applying these two templates, if there is still

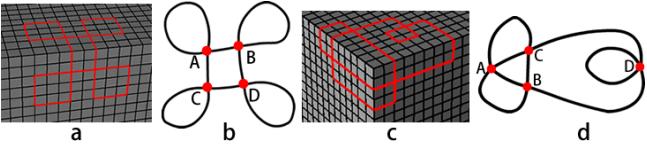


Figure 7: Two examples of pairing intersecting nodes: (a) the boundary loop of the 1st example; (b) the topological structure of the boundary loop of the 1st example; (c) the boundary loop of the 2nd example; (d) the topological structure of the boundary loop of the 2nd example.

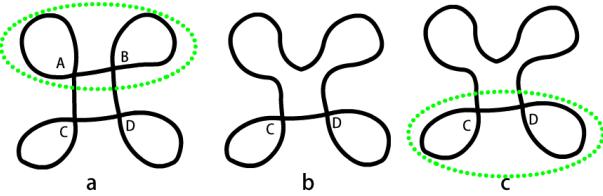


Figure 8: The pairing process of the 1st example: (a) node A and B are paired; (b) node A and B are resolved; (c) node C and D are paired.

231 one intersecting node left unpaired, we use the templates to cre-  
232 ate a new intersecting node at a appropriate position to be paired  
233 with this intersecting node. For example, in Fig. 10(a), the in-  
234 tersecting node A is unpaired. We use the 2nd template to create  
235 a new intersecting node B to be paired with A as shown in Fig.  
236 10(b).

237 Our algorithm would fail if some intersecting nodes are  
238 unable to be paired by the two templates. These boundary  
239 loops usually determine non-orientable quad sets[8] which will  
240 largely degenerate the mesh quality after sheet inflation. There-  
241 fore we do not handle such kind of boundary loops in this paper.

### 242 3.2. Forming Closed Boundary Loops

243 The ability to do local separation requires the constraint  
244 edges to form closed loops. Sometimes the constraint edges,  
245 which are specified by the user according to specific mesh mod-  
246 ification requirements, may not be closed loops. When this hap-  
247 pens, we need to close the constraint edges by adding new edges  
248 on the mesh boundary. This is done by connecting the dangling  
249 edges on the constraint edges using the A\* path searching algo-  
250 rithm.

251 A\* path searching is a one-source-one-target path searching  
252 algorithm which runs very efficiently. When applying A\* al-  
253 gorithm, in addition to considering the count of mesh edges on  
254 the path, we also take the angles between adjacent edges on the  
255 path into consideration in order to get a smooth path. Figure

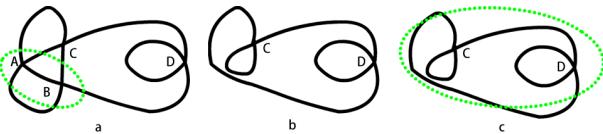


Figure 9: The pairing process of the 2nd example: (a) node A and B are paired; (b) node A and B are resolved; (c) point C and D are paired.

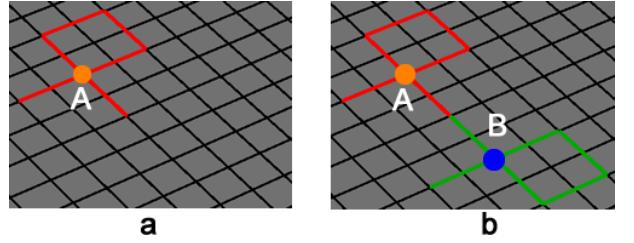


Figure 10: Handling of the situation when single intersecting node is unpaired: (a) intersecting node A is unpaired; (b) intersecting node B is created to be paired with A.

256 11(b) shows the result when the angles between adjacent edges  
257 are not considered, and Fig. 11(c) shows a better result when  
258 the angles are taken into consideration.

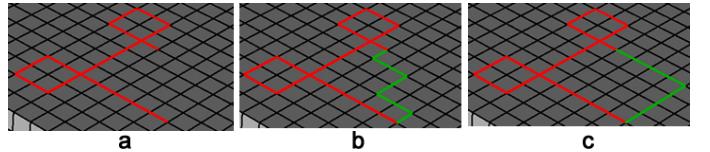


Figure 11: Closing the boundary loop: (a) the unclosed boundary loop; (b) new edges determined by A\* algorithm without considering the turning angles; (c) new edges determined by A\* algorithm considering the turning angles.

### 259 3.3. Making the Boundary Loop Able to Do Local Separation

260 After previous sections, the constraint edges specified by the  
261 user become a closed boundary loop with intersecting nodes  
262 being paired. According to the discussion in the beginning of  
263 Section 3, a valid boundary loop should be able to do local sep-  
264 aration. Sometimes the closed boundary loop may fail to do  
265 local separation due to the existence of through holes on the  
266 hex mesh. To fix that, we use the max-flow-min-cut algorithm  
267 to add necessary edges to the boundary loop to make it able to  
268 do local separation. Suppose the boundary loop is E, the algo-  
269 rithm flowchart is illustrated in Fig. 12.

270 The following example illustrates the process. Figure 13(a)  
271 shows the hex mesh containing a through hole and the bound-  
272 ary loop E (red). To check whether E is able to do local sep-  
273 aration, we firstly get E's adjacent hex set H (Fig. 13(b)) and  
274 iteratively add H's adjacent hexahedra to H and test whether  
275 the new boundary of H can be separated by E into two subsets  
276 until reaching the maximum iteration times (Fig. 13(c)). How-  
277 ever, E is still unable to do local separation on H since there  
278 are still only one quad set (blue in Fig. 13(d)) on H's bound-  
279 ary. Hence, we need to use the max-flow-min-cut algorithm to  
280 add other necessary mesh edges to E to make it able to do local  
281 separation.

282 Max-flow-min-cut algorithm is an effective and efficient tool  
283 to find the minimal cut set in a directed graph. As shown in  
284 Fig. 14(a), to use the max-flow-min-cut algorithm, a directed  
285 graph should be constructed with an *s* node and a *t* node, and  
286 the weights of the edges should also be set before calculation.

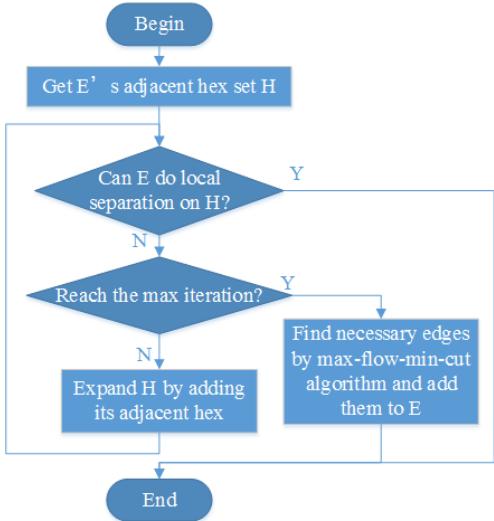


Figure 12: Flowchart of the process of making the boundary loop able to do local separation.

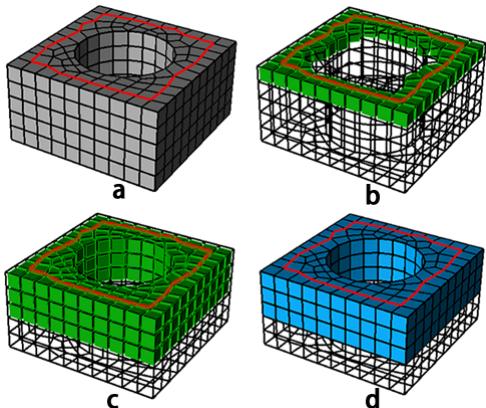


Figure 13: An example of boundary loops that are unable to do local separation: (a) the boundary loop  $E$ ; (b)  $E$ 's adjacent hexahedra  $H$ (green); (c) expand  $H$  by iteratively adding its adjacent hexahedra; (d)  $E$  still cannot separate  $H$ 's boundary into two subsets when reaching maximum expanding iteration times.

287 The algorithm will find a minimal cut set that determines the  
288 max flow from  $s$  to  $t$ .

289 If we take the boundary of  $H$  as a graph where the nodes  
290 stand for a quad or a quad set and the edges stand for the ad-  
291 jacency relationship between the quads or quad sets, then to be  
292 able to do local separation is similar to find a cut set on this  
293 graph. Hence, the max-flow-min-cut algorithm can be used to  
294 help us to find necessary edges to make the boundary loop sat-  
295 isfy the local separation.

296 To use the algorithm, we need to construct the directed graph  
297 at first, including constructing the  $s$  and  $t$  nodes and deciding  
298 the edges' weights. As the current boundary loop  $E$  must be  
299 part of the final boundary loop, the two quad sets on its two  
300 sides just can be the  $s$  and  $t$  nodes as shown in Fig. 14(b). Since  
301 all of the edges on the final boundary loop should be on the  
302 boundary of the hex mesh, the quads shared by  $H$  and the rest  
303 of the hex mesh, which is shown as  $Q_{in}$  in Fig. 14(b), should be  
304 accordingly grouped into a single node in the graph to guarantee  
305 that  $Q_{in}$  won't be separated by the min cut. Similarly, in order to  
306 forbid the mesh edges on sharp CAD model edges to be added  
307 in to the boundary loop, which usually results in poor mesh  
308 quality, we group the quads sharing these mesh edges into a  
309 single node, e.g.  $q_2$  and  $q_3$  in Fig. 14(c).

310 After constructing the nodes, we assign weights to the graph  
311 edges according to how many mesh edges are shared by the  
312 two nodes. For example,  $q_1$  in Fig. 14(c) shares two edges with  
313 node  $t$ , so the weight of the directed edges in the graph between  
314  $q_1$  and  $t$  is 2. All of the edges in the graph are bi-directional  
315 except the edges adjacent to  $s$  and  $t$ .

316 The final directed graph is shown in Fig. 14(d). After cal-  
317 culation, new edges are found as shown in Fig. 14(e), and now  
318  $E$  is able to do local separation after adding these new edges as  
319 shown in Fig. 14(f).

320 When a self-intersecting boundary loop is unable to do local  
321 separation, we also need to apply max-flow-min-cut algorithm  
322 to find necessary new edges to make it able to do local separa-  
323 tion. The procedures are quite similar to non-self-intersecting  
324 boundary loop discussed previously. We treat a self-intersecting  
325 boundary loop as several non-self-intersecting boundary loops  
326 and handle each of them individually. The self-intersecting  
327 boundary loop in Fig. 15(a) consists of three sub loops  $loop_1$ ,  
328  $loop_2$  and  $loop_3$ . The local hex set is shown in Fig. 15(b). Due  
329 to the existence of three through holes on the hex mesh, the  
330 boundary loop is unable to do local separation. After applying  
331 max-flow-min-cut algorithm to  $loop_1$  and  $loop_3$ , new edges are  
332 found as shown in Fig. 15(c).

#### 333 4. Determination of the Initial Quad Set

334 This section provides specific details of the determination of  
335 the initial quad set based on the boundary loop. The intersect-  
336 ing lines are firstly constructed according to the pairing of the  
337 intersecting nodes. The initial quad set is then constructed us-  
338 ing the max-flow-min-cut algorithm. Since the boundary loops  
339 satisfy the boundary constraints and the max-flow-min-cut al-  
340 gorithm is conducted within the local region specified by the  
341 user, the initial quad set fulfills all the constraints.

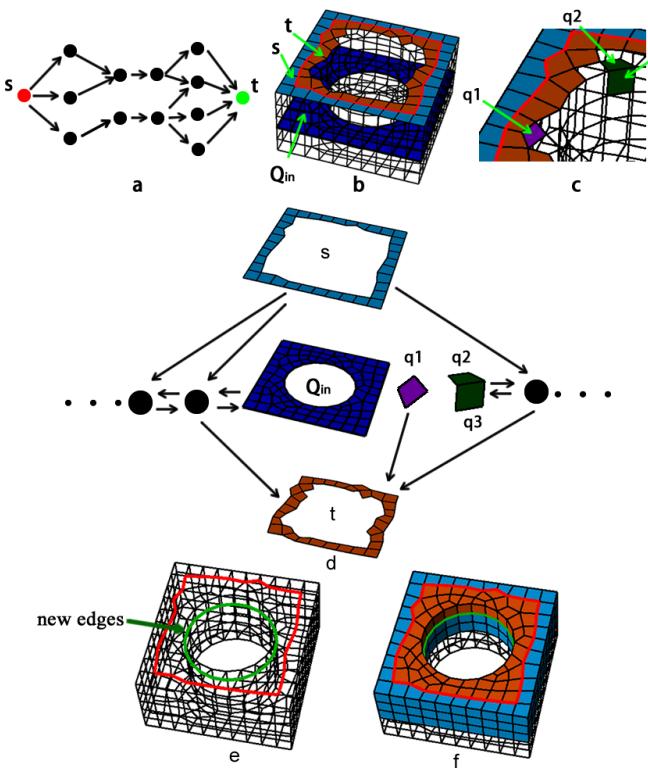


Figure 14: Using max-flow-min-cut algorithm to find necessary edges for local separation: (a) directed graph used for ordinary max-flow-min-cut problem; (b) the quad sets on  $E$ 's two sides can be  $s$  and  $t$  nodes; (c)  $q_1$  shares two edges with  $t$ ,  $q_2$  and  $q_3$  share an edge on the geometric hard edge; (d) the directed graph for max-flow-min-cut algorithm; (e) new edges are found (green); (f)  $E$  is able to do local separation after adding new edges.

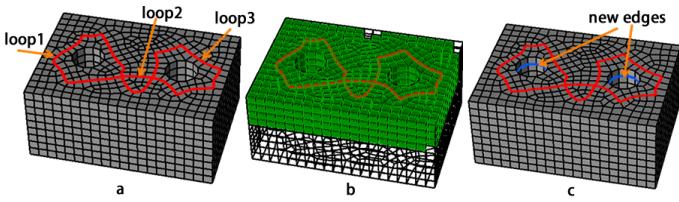


Figure 15: Making self-intersecting boundary loop able to do local separation: (a) the self-intersecting boundary loop with three sub loops; (b) the local hex set; (c) new edges (blue) are found by using max-flow-min-cut algorithm.

#### 342 4.1. Determination of Intersecting Lines

343 Intersecting lines are very important for defining the struc-  
344 tures of self-intersecting quad sets. Before determining the ini-  
345 tial quad set, we need to construct the intersecting lines first. In  
346 Section 3.1 the two templates not only make two intersecting  
347 nodes be paired but also setup the corresponding relationships  
348 between the int-4-quad-sets of the two intersecting nodes. In  
349 this section, we use this information to construct the intersect-  
350 ing lines and the int-4-hex-sets of these intersecting lines.

351 The following example shows the process of constructing the  
352 intersecting line and its int-4-hex-sets. In Fig. 16(a), the blue  
353 nodes are two intersecting nodes paired by the 1st template (in  
354 this example they can also be paired by the 2nd template), and  
355 the numbers and different colors denote the correspondence be-  
356 tween the int-4-quad-sets of the two points. We use A\* algo-  
357 rithm to determine the intersecting line between the two inter-  
358 secting nodes. Similar to Section 3.2, in order to get a smooth  
359 path, we take the turning angles between adjacent edges in the  
360 path into consideration while applying the A\* searching. The  
361 intersecting line is shown in Fig. 16(b). Next, we get all the  
362 hexahedra adjacent to the intersecting line, and use the corre-  
363 spondence between the int-4-quad-sets to construct the int-4-  
364 hex-sets as shown in Fig. 16(c).

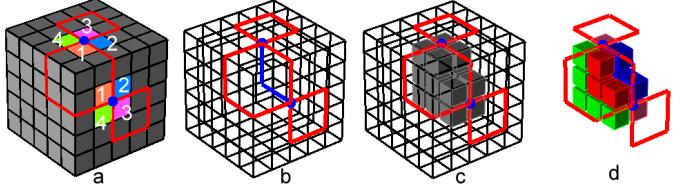


Figure 16: Process of constructing the intersecting line and its int-4-hex-sets: (a) two intersecting nodes paired by the 1st template; (b) the intersecting line determined by A\* algorithm; (c) the hexahedra adjacent to the intersecting line; (d) the int-4-hex-sets of the intersecting line.

#### 365 4.2. Determination of Initial Quad Set by Max-flow-min-cut Al- 366 gorithm

367 After constructing the intersecting lines and their int-4-hex-  
368 sets, we now construct the initial quad set. The idea is similar to  
369 that mentioned in Section 3.3, we convert the problem from be-  
370 ing able to do local separation to acquiring min cut on a graph.  
371 If we see the hexahedra as nodes and the adjacency between the  
372 hexahedra as edges of the graph, the quad set is actually a cut set  
373 of the graph. Hence, max-flow-min-cut algorithm can be used  
374 to efficiently get a quad set that is able to do local separation.  
375 This section provides specific details about this process.

376 As the boundary loop separates the boundary of the local hex  
377 set into  $2 + n$  subsets ( $n$  is the number of intersecting nodes), we  
378 can get  $2 + n$  hex sets that are adjacent to these  $2 + n$  quad sets.  
379 From these  $2 + n$  hex sets and the int-4-hex-sets of the inter-  
380 secting lines, we merge hex sets that share common hexahedra.  
381 This will result in  $2 + N$  hex sets ( $N$  is the number of intersecting  
382 lines). We then apply the max-flow-min-cut algorithm multiple  
383 times to get the initial quad set. The pseudo-codes are provided  
384 in Algorithm 1.

---

**Algorithm 1** Determination of the Initial Quad Set

---

**Input:** The boundary loop  $L$  and the local hex set  $H_{local}$ ;

**Output:** The initial quad set  $Q_{init}$ ;

- 1:  $n$  = the count of self-intersecting nodes on  $L$ ;
- 2:  $N$  = the count of intersecting lines;
- 3: Get the quad subsets on the boundary of  $H_{local}$  as  $Q_{bound} = \{Q_{b1}, Q_{b2}, \dots, Q_{b2+n}\}$ ;
- 4:  $H_{bound} = \{H_{b1}, H_{b2}, \dots, H_{b2+n}\}$  where  $H_i$  is the hex set adjacent to  $Q_i, i = 1, 2, \dots, 2 + n$ ;
- 5:  $H_{int} = \{H_{in1}^1, H_{in1}^2, H_{in1}^3, H_{in1}^4, \dots, H_{inN}^1, H_{inN}^2, H_{inN}^3, H_{inN}^4\}$  where  $\{H_{in1}^1, H_{in1}^2, H_{in1}^3, H_{in1}^4\}$  is the int-4-hex-sets of the  $i$ th intersecting line;
- 6:  $H_{merge} = H_{bound} \cup H_{int}$ ;
- 7: **while**  $\exists H_{mi}, H_{mj} \in H_{merge}, i \neq j$  that  $H_{mi} \cap H_{mj} = \emptyset$  **do**
- 8:   Merge  $H_{mi}$  and  $H_{mj}$ ;
- 9: **end while**
- 10:  $Q_{init} = \emptyset$ ;
- 11: **for** each  $H_{mi} \in H_{merge}$  **do**
- 12:    $H_{rest} = \bigcup\{H_{merge} - H_{mi}\}$ ;
- 13:   Let  $H_{mi}$  be the  $s$  node and  $H_{rest}$  be the  $j$  node;
- 14:   Perform the max-flow-min-cut algorithm and get the quad set  $Q_i$ ;
- 15:    $Q_{init} = Q_{init} \cup Q_i$ ;
- 16: **end for**

---

385 Two examples are provided to illustrate the procedures of Al-  
386 gorithm 1. The first example is a non-self-intersecting boundary  
387 loop as shown in Fig. 17.  $n$  and  $N$  are both 0 for this example.  
388 The two quad subsets on the boundary of the local hex set  $H$   
389 are  $Q_{b1}$  and  $Q_{b2}$  as shown in Fig. 17(c). We get the two hex  
390 sets adjacent to  $Q_{b1}$  and  $Q_{b2}$  as  $H_{b1}$  and  $H_{b2}$  (Fig. 17(d)). We  
391 construct the directed graph by taking  $H_{b1}$  as  $s$  node and  $H_{b2}$  as  
392  $t$  node, taking other hexahedra as normal nodes, and assigning  
393 weights to the edges according to the number of quads shared  
394 by two hex sets. After running the max-flow-min-cut algorithm,  
395 we get the initial quad set  $Q_{init}$  as shown in Fig. 17(e).

396 The second example is a self-intersecting boundary loop as  
397 shown in Fig. 18. The boundary loop (red in Fig. 18(a))  
398 intersects itself once, hence  $n = 2, N = 1$ . Firstly we  
399 get the int-4-hex-sets  $i4h_1-i4h_4$  (Fig. 18(b)). The bound-  
400 ary of the local hex set  $H$  is separated into  $Q_{b1}-Q_{b4}$  (Fig.  
401 18(c)), and the hex sets adjacent to these quad subsets are  
402  $H_{b1}-H_{b4}$  (Fig. 18(d) and Fig. 18(e)). Therefore  $H_{merge} =$   
403  $H_{int} \cup H_{bound} = \{i4h_1, \dots, i4h_4, H_{b1}, \dots, H_{b4}\}$ . After merging,  
404  $H_{merge} = \{H_{m1}, H_{m2}, H_{m3}\}$  (Fig. 18(f) and Fig. 18(g)). We then  
405 conduct the max-flow-min-cut algorithm twice. For the first  
406 time, we take  $H_{m1}$  as  $s$  node and  $H_{m2} \cup H_{m3}$  as  $t$  node (Fig.  
407 18(h)), and get the quad set  $Q_1$  (Fig. 18(i)). For the second  
408 time, we take  $H_{m2}$  as  $s$  node and  $H_{m1} \cup H_{m3}$  as  $t$  node (Fig.  
409 18(j)), and get the quad set  $Q_2$  (Fig. 18(k)). Finally the initial  
410 quad set is  $Q_{init} = Q_1 \cup Q_2$  (Fig. 18(l)).

## 5. Optimization of the Initial Quad Set

412 Although the initial quad set determined by previous section  
413 satisfies all the constraints specified by the user, the quality of

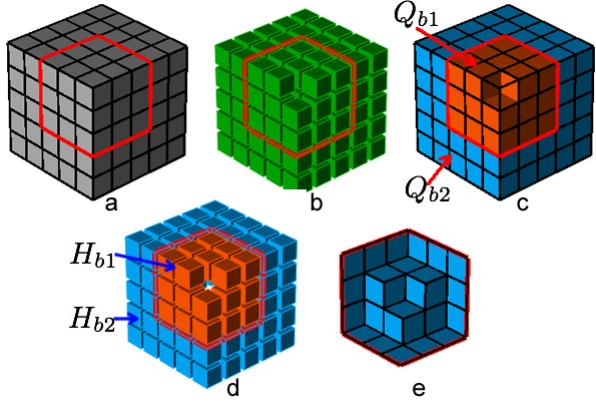


Figure 17: Process of the determination of the initial quad set from a non-intersecting boundary loop: (a) the non-intersecting boundary loop; (b) the local hex set; (c) the two boundary quad subsets  $Q_{b1}$ (orange) and  $Q_{b2}$ (blue); (d) two hex sets  $H_{b1}$ (orange) and  $H_{b2}$ (blue); (e) the initial quad set  $Q_{init}$ .

414 the mesh after sheet inflation is usually not good enough. In  
415 this paper we propose a new optimization method for the quad  
416 set based on the quad set's dual structures. In this section we  
417 discuss the details of this optimization method.

### 418 5.1. Quality Evaluation of the Quad Set

419 In a hex mesh, the valence of a mesh edge stands for the count  
420 of its adjacent quads. A mesh edge is called a regular edge if  
421 it is inside the mesh or on the mesh boundary and its valence  
422 equals 4 or 3 respectively. Otherwise the edge is called an ir-  
423 regular edge. The irregular degree is the difference of valences  
424 between one edge and its corresponding regular edge. Suppose  
425  $e$  stands for an edge and  $v_e$  stands for its valence, the irregular  
426 degree of  $e$  is computed as Equation 1.

$$427 ID(e) = \begin{cases} \|v_e - 4\| & \text{if } e \text{ is inside the mesh,} \\ \|v_e - 3\| & \text{if } e \text{ is on the mesh boundary.} \end{cases} \quad (1)$$

428 Staten et al. showed that the edge valence has direct impact  
429 on the mesh quality in [11]. Therefore in this paper we use the  
430 edges' valences and irregular degree to evaluate the quality of  
431 the quad set.

432 When creating new sheets, different quad sets have different  
433 impact on the quality of the mesh. In Fig. 19(a), before sheet  
434 inflation,  $ID(e_1) = 0$ ; after sheet inflation,  $e_1$  is splitted into  
435 two edges  $e_2$  and  $e_3$  and  $ID(e_2) + ID(e_3) = 0$  as shown in Fig.  
436 19(c). However, if  $Q_2$  (Fig. 19(d)) is inflated, although  $e_4$  is a  
437 regular edge, the two edges splitted from  $e_4$  are irregular edges  
438 as shown in Fig. 19(f), and  $ID(e_5) + ID(e_6) = 2$ . The inherent  
439 reason for the difference of irregular degrees is that  $Q_1$  and  $Q_2$   
440 separate the adjacent hex sets into different configurations as  
441 shown in Fig. 19(b) and Fig. 19(e).

442 Suppose  $e$  is an edge on quad set  $Q$ , and  $Q$  separates  $e$ 's ad-  
443 jacent hexahedra into two subsets  $H_1$  and  $H_2$ , then the variation  
444 between the irregular degrees of  $e$  and two edges splitted from  
445  $e$  can be computed by Equation 2.  $\|H_1\|$  and  $\|H_2\|$  represent the

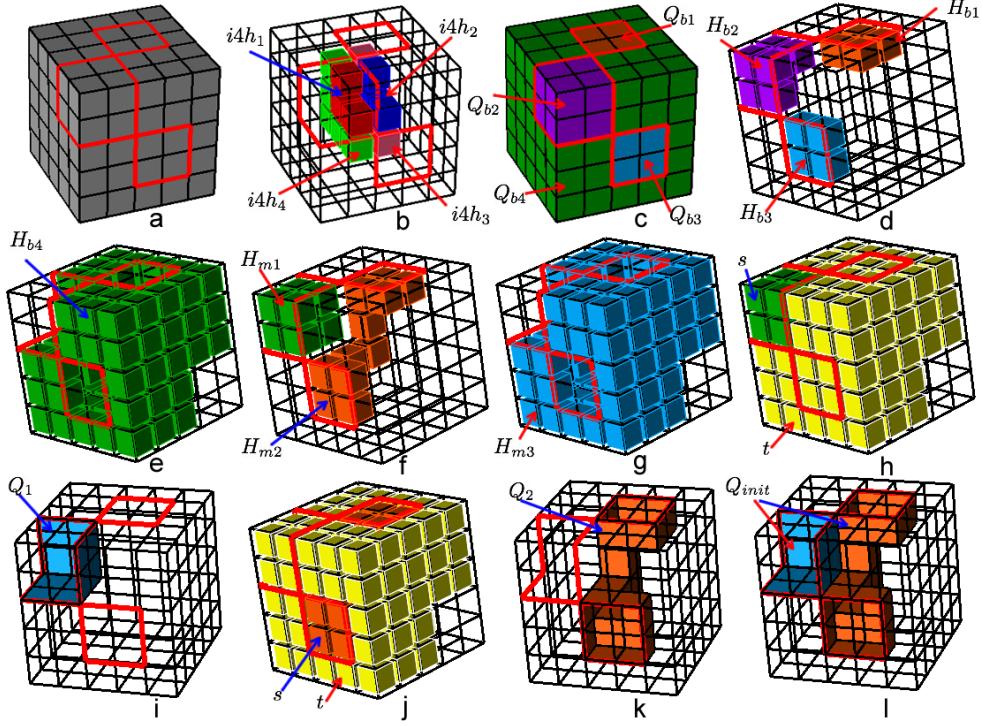


Figure 18: Process of the determination of the initial quad set from a self-intersecting boundary loop: (a) the self-intersecting boundary loop; (b) the int-4-hex-sets  $i4h_1-i4h_4$  of the intersecting line; (c) the four quad subsets  $Q_{b1}-Q_{b4}$  on the boundary of the local hex set separated by the boundary loop; (d) the hex sets  $H_{b1}-H_{b3}$  adjacent to  $Q_{b1}-Q_{b3}$  respectively; (e) the hex set  $H_{b4}$  adjacent to  $Q_{b4}$ ; (f) the hex set  $H_{m1}$  merging from  $i4h_1$  and  $H_{b2}$  and  $H_{m2}$  merging from  $i4h_3$ ,  $H_{b1}$  and  $H_{b3}$ ; (g) the hex set  $H_{m3}$  merging from  $i4h_2$ ,  $i4h_4$  and  $H_{b4}$ ; (h) take  $H_{m1}$  as  $s$  node and  $H_{m2} \cup H_{m3}$  as  $t$  node; (i) the quad set  $Q_1$  determined by the max-flow-min-cut algorithm; (j) take  $H_{m2}$  as  $s$  node and  $H_{m1} \cup H_{m3}$  as  $t$  node; (k) the quad set  $Q_2$  determined by the max-flow-min-cut algorithm; (l) the initial quad set  $Q_{init} = Q_1 \cup Q_2$ .

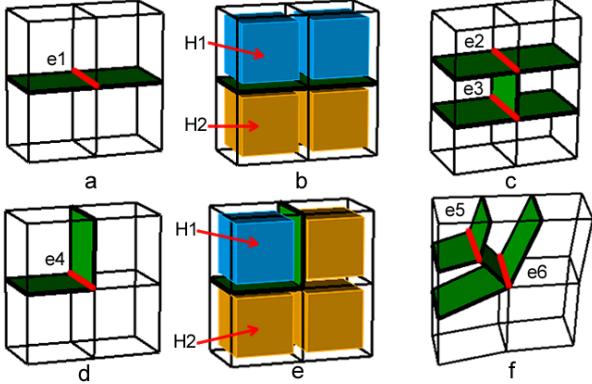


Figure 19: Different impacts on the mesh quality of different quad sets: (a) quad set  $Q_1$  (green) and one of its mesh edges  $e_1$  (red); (b) the hex set adjacent to  $e_1$  is separated into two subsets  $H_1$  and  $H_2$  by  $Q_1$ ; (c)  $e_1$  is splitted into two edges  $e_2$  and  $e_3$  after inflation; (d) quad set  $Q_2$  and one of its mesh edges  $e_4$ ; (e) the hex set adjacent to  $e_4$  is separated into two subsets  $H_1$  and  $H_2$  by  $Q_2$ ; (f)  $e_4$  is splitted into two edges  $e_5$  and  $e_6$  after inflation.

numbers of hexahedra in  $H_1$  and  $H_2$  respectively. A negative  $\Delta V_e$  means the mesh quality near  $e$  has been improved by the inflation, otherwise the mesh quality becomes worse.

$$\Delta V_e = \|H_1\| + \|H_2\| - ID(e) \quad (2)$$

If  $Q$  is non-self-intersecting,  $E = \{e_1, e_2, \dots, e_n\}$  is the set of edges on  $Q$  where  $n$  is the count of the edges,  $Q$ 's variation of irregular degrees  $\Delta V_Q$  can then be computed by Equation 3.

$$\Delta V_Q = \sum_{i=1}^n \Delta V_{e_i} \quad (3)$$

### 5.2. Chord-based Optimization of the Quad Set

To improve the quality of the quad set, it needs to adjust the structure of the quad set to make it contain as less edges with large variation of irregular degrees as possible. It is, however, not a trivial task since any changes applied on one edge will inevitably impact its adjacent edges. Theoretically, if we evaluate all the possible quad sets then we can find the optimal quad set with least variation of irregular degrees. Nevertheless it is almost impossible due to the large searching space. Chen et al. proposed an optimization method which locally handles concave or convex edges one by one[10]. This method avoids the difficulty of global optimization by handling the irregular edges on the quad set in a greedy-based manner. However, it cannot guarantee the mesh quality due to the interaction between edges

465 on the quad set, and sometimes it may be even not convergent.  
 466 In this paper, we propose an optimization method based on the  
 467 dual structure of the quad set. Our method not only avoids the  
 468 difficulty in global optimization, but also guarantees the mesh  
 469 quality after sheet inflation.

470 On a quad mesh, starting from an edge, we can recursively  
 471 get all the edges that are topologically parallel to this edge.  
 472 These edges and the adjacent quads form a chord, the dual  
 473 structure of the quad set[4, 12]. Similar to the quad mesh, the  
 474 quad set used for sheet inflation also consists of chords. All  
 475 of the mesh edges on the boundary loop pairwise belong to one  
 476 chord. For example, in Fig. 20(b),  $e_1$  and  $e_2$  belong to the chord  
 $c_1$ , and  $e_3$  and  $e_4$  belong to the chord  $c_2$ . The difference between  
 478 a quad mesh and a quad set for sheet inflation is that there is a  
 479 sheet associated with each chord on the quad set. For example,  
 480 in Fig. 20(c),  $e_3$ ,  $e_4$  and their chords are contained in the sheet  $s$   
 481 (red). This relationship reveals that if two edges on the bound-  
 482 ary loop belong to a same chord, then we can find other chords  
 483 connecting these two edges within the corresponding sheet.

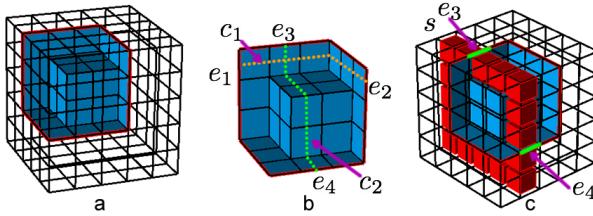


Figure 20: The chords of the quad set: (a) the quad set; (b) two chords  $c_1$  and  $c_2$  on the quad set; (c) the sheet  $s$  that contains  $e_3$  and  $e_4$ .

484 We improve the quality of the quad set by iteratively im-  
 485 proving the quality of the chords that contain the edges on the  
 486 boundary loop. The chords are neither too constrained as the  
 487 whole quad set nor too local as one single concave or convex  
 488 edge. By optimizing the chords, we can effectively improve the  
 489 quality of the quad set while avoiding the difficulty of optimiz-  
 490 ing the quad set as a whole. Suppose the quad set is  $Q$  and  
 491 its edge set is  $E$ , the details of the optimization procedures are  
 492 explained:

- 493 1. Divide the edges on the boundary loop into groups accord-  
 494 ing to whether they are in the same sheet. For example,  $e_3$   
 495 and  $e_4$  are grouped in Fig. 21(a), and  $e_1 - e_4$  are grouped  
 496 in Fig. 22(a);
- 497 2. Select a group of edges that have not been processed. Get  
 498 all of the edges in the corresponding sheet and the quads  
 499 adjacent to these edges. These edges and quads com-  
 500 pose the searching space for new chords that connect these  
 501 edges. Suppose the edge group is  $g = \{e_3, e_4\}$ , Fig. 21(b)  
 502 shows the edges that are in the same sheet of  $g$  and the  
 503 adjacent quads;
- 504 3. Since the edges in the group pairwise connect to one  
 505 chord, we use A\* algorithm to search for the new chords.  
 506 While searching, we take the edges' variation of irregular  
 507 degrees into consideration. For example, in Fig. 21(c),  
 508 there are three quads adjacent to  $q_1$ . If we select  $q_2$  or  $q_4$

509 as the next forward step, the variation of irregular degrees  
 510 of  $e_5$  will be 2; if we select  $q_3$  as the next forward step,  
 511 the variation of irregular degrees  $e_5$  becomes 0. Hence we  
 512 select  $q_3$ . The new chord connecting  $e_3$  and  $e_4$  is shown in  
 513 Fig. 21(d). If there are more than two edges in the group,  
 514 it needs deciding which two edges should be paired. For  
 515 example, four edges  $e_1 - e_4$  in Fig. 22(b) are in the same  
 516 group. If we connect  $e_1$  and  $e_3$  and get the new chord  $c_3$   
 517 (blue in Fig. 22(c)), then  $e_2$  and  $e_4$  cannot be connected  
 518 without intersecting with  $c_3$  which is not allowed for keep-  
 519 ing the quad set valid as shown in Fig. 22(d). So we search  
 520 new chords between  $e_1$  and  $e_4$ ,  $e_2$  and  $e_3$ . The two new  
 521 chords are shown in Fig. 22(e);

- 522 4. The new chords and the original chords encompass a hex  
 523 set on the sheet. For example, the hex set  $h$  in Fig. 21(e)  
 524 is encompassed by the two chords  $c_2$  and  $c_4$ , and the hex set  
 525  $h$  in Fig. 22(f) is encompassed by the four chords  $c_1, c_2, c_4$   
 526 and  $c_5$ . We get the boundary quad set  $Q_h$  of  $h$ , and let  $Q$   
 527 be the symmetric difference between  $Q$  and  $Q_h$ , i.e.  $Q = Q \cup Q_h - Q \cap Q_h$ . The updated  $Q$  is shown in Fig. 21(f)  
 528 and Fig. 21(g);
- 529 5. Repeat the process of Step 2-4 until all of the edge groups  
 530 are handled. The final quad sets for the two examples are  
 531 shown in Fig. 21(g) and Fig. 22(g).

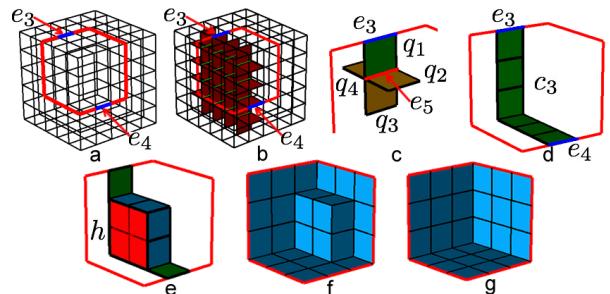


Figure 21: The 1st example of the quad set optimization: (a) two edges  $e_3$  and  $e_4$  on the boundary loop; (b) the edges on the sheet (green) and adjacent quads (red); (c)  $q_1$ 's three adjacent quads  $q_2$ ,  $q_3$  and  $q_4$ ; (d) the new chord connecting  $e_3$  and  $e_4$ ; (e) the hex set  $h$  encompassed by the two chords; (f)  $Q$  is updated by getting the symmetric difference between  $Q$  and  $Q_h$ ; (g) the final quad set.

533 The variation of irregular degrees  $\Delta V_Q$  of the initial quad set  
 534 in Fig. 20(a) is 42. After optimization, the variation of irregular  
 535 degrees of the quad set in Fig. 21(g)  $\Delta V'_Q$  becomes 18, which is  
 536 24 smaller than  $\Delta V_Q$ .

537 The variation of irregular degrees  $\Delta V_Q$  of the initial quad set  
 538 in Fig. 22(a) is 76, and it becomes 56 after optimization which  
 539 is 20 smaller.

## 540 6. Examples

541 We implement our approach in C++ on a 32-bit Windows  
 542 7 platform, using Visual Studio 2010. The approach has been  
 543 tested on different meshes under various constraints. This sec-  
 544 tion presents two practical applications of the approach: one is  
 545 complex sheets generation for mesh matching and the other one

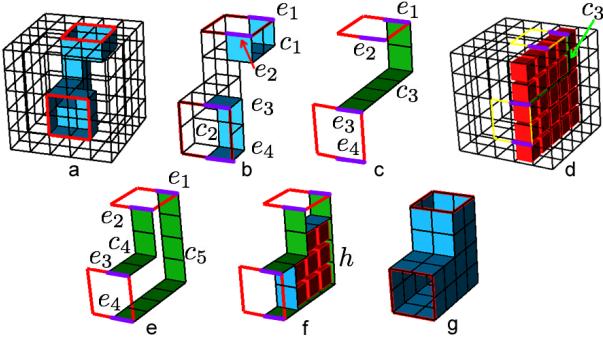


Figure 22: The 2nd example of the quad set optimization: (a) the initial quad set; (b)  $e_1$ - $e_4$  on the same sheet and two chords  $c_1$  and  $c_2$ ; (c) the new chord  $c_3$  connecting  $e_1$  and  $e_3$ ; (d)  $e_2$  and  $e_4$  cannot be connected by a chord without intersecting  $c_3$ ; (e) the two new chords  $c_4$  and  $c_5$ ; (f) the hex set  $h$  encompassed by the four chords; (g)  $Q$  is updated by getting the symmetric difference between  $Q$  and  $Q_h$ .

546 is the quality improvement for the mesh boundary by reducing  
547 high node valences.

#### 548 6.1. Complex Sheets Generation for Mesh Matching

549 Mesh matching is an algorithm to convert non-conforming  
550 interfaces to conforming ones by sheet operations[11, 10]. Our  
551 approach in this paper can be used in mesh matching to lo-  
552 cally generate complex sheets that intersect themselves multiple  
553 times, which cannot be done in these previous works. In Fig.  
554 23, there is an unmatched chord on the interface of hex mesh 2  
555 in Fig. 23(b). A new sheet needs to be created to match it on  
556 hex mesh 1 under the constraints shown in Fig. 23a ( $H$  is the  
557 volumetric constraints).

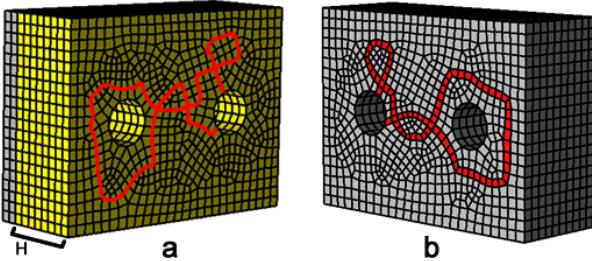


Figure 23: Complex sheet inflation is needed for mesh matching: (a) the constraints for sheet inflation on mesh 1; (b) the unmatched chord(red) on mesh 2.

558 Under the constraints, we construct the boundary loop as  
559 shown in Fig. 24. Since the constraint edges contain three in-  
560 tersecting nodes, a new intersecting node is created as shown in  
561 Fig. 24(b). By running the max-flow-min-cut algorithm, new  
562 edges are found, enabling the boundary loop to do local separa-  
563 tion. The final boundary loop is shown in Fig. 24(e).

564 Intersecting lines are then constructed, as well as the int-4-  
565 hex-sets as shown in Fig. 25(a) and Fig. 25(b). Then we get  
566 the merged hex sets in Fig. 25(c) and the initial quad set in  
567 Fig. 25(d). The variation of irregular degrees of the initial quad  
568 set is 478. By contrast, the final quad set is much better after

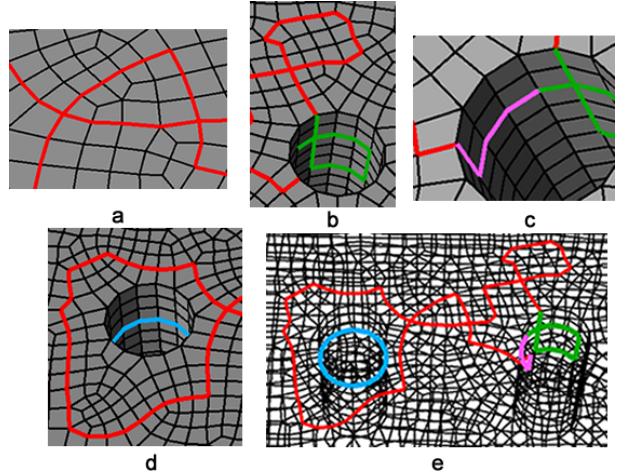


Figure 24: Determination of the boundary loop: (a) the two intersecting nodes are paired; (b) a new intersecting node is created; (c) the boundary loop is closed; (d) new edges are determined by the max-flow-min-cut algorithm; (e) the final boundary loop.

569 optimization as shown in Fig. 25(e) whose variation of irregular  
570 degrees is reduced to 351.

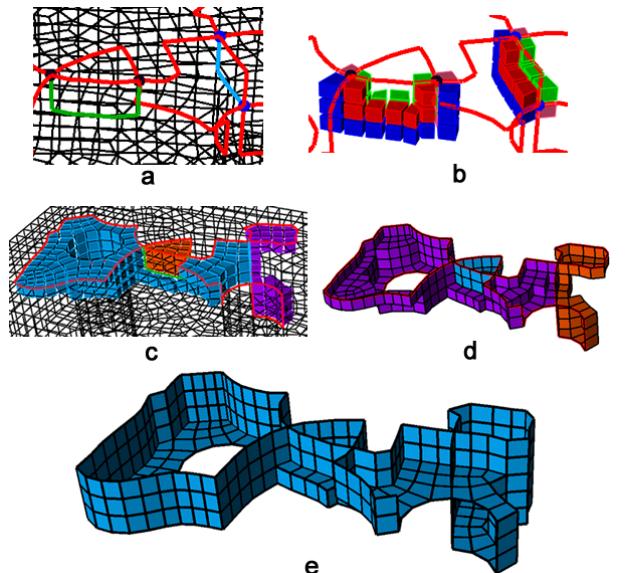


Figure 25: Determination of the quad set: (a) constructing the intersecting lines; (b) the int-4-hex-sets; (c) the merged hex sets; (d) the initial quad set; (e) the optimized quad set.

571 The new sheet is inflated from the quad set as shown in Fig.  
572 26.

#### 573 6.2. High Edge Valences Reduction

574 In hex meshes, the elements in the area near the boundary are  
575 usually very important for finite element analysis[13]. Mean-  
576 while, the node valence and edge valence are critical for the  
577 quality of the quad mesh and hex mesh respectively[14, 11, 15].

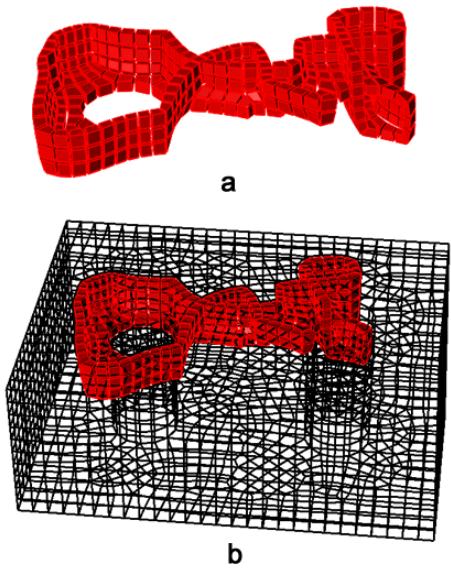


Figure 26: Sheet inflation: (a) the new sheet inflated from the quad set; (b) the new sheet in the hex mesh.

High node valence or edge valence will largely reduce the accuracy and efficiency of the analysis, or even make the mesh unable to be handled by the solver. The fun sheet matching algorithm proposed by Kowalski et al. [13] can help the hex mesh capture the geometric boundary by inserting fundamental sheets. This algorithm, however, can neither improve the node valences on the mesh boundary nor the edge valence near the mesh boundary. In this example, we will show how our approach can be used to reduce the high node valences and edge valences near the mesh boundary.

Figure 27(a) shows two nodes  $v_1$  and  $v_2$  with high valences. The valences of  $v_1$  and  $v_2$  are 6 and 7 respectively. Consequently, the edges inside the hex mesh that are adjacent to these two nodes also have high valences which are 6 and 7 respectively as shown in Fig. 27(b). It is usually unacceptable for node or edge valences to be larger than 5 near the boundary. Therefore, these high valences need to be reduced.

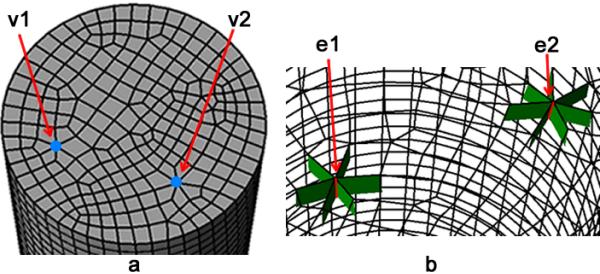


Figure 27: Nodes and edges with high valences: (a)  $v_1$ 's valence is 6 and  $v_2$ 's valence is 7; (b)  $v_1$ 's adjacent edge  $e_1$  has the valence of 6 and  $v_2$ 's adjacent edge  $e_2$  has the valence of 7.

To reduce the high valences by our approach, the user firstly select several edges on the mesh boundary that are adjacent to

the two nodes and the local region as shown in Fig. 28(a). Our approach takes these edges as constraints, and then try to achieve the sheet inflation under the constraints. Figure 28(b) and Fig. 28(c) show the process of determination of the boundary loop, and the final boundary loop is shown in Fig. 28(d).

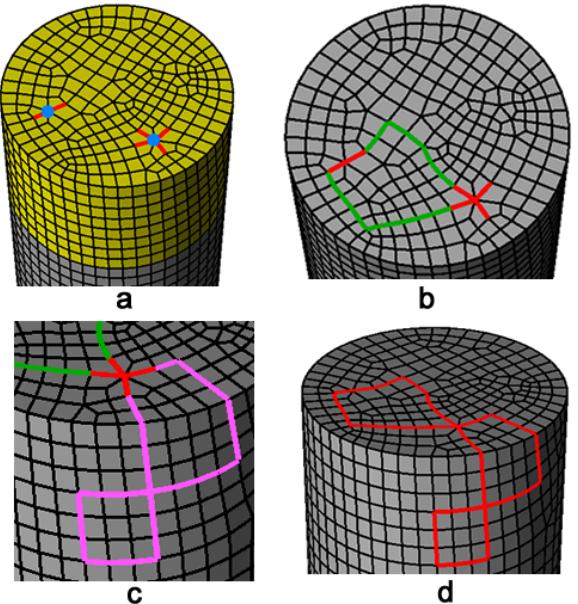


Figure 28: User-specified constraints and the determination of the boundary loop: (a) several boundary edges adjacent to  $v_1$  and  $v_2$  are selected as boundary constraints(red) and a set of hexahedra are selected as local region (yellow); (b) the constraint edges are firstly connected; (c) a new intersecting node is created; (d) the final boundary loop.

The quad set is then constructed through the procedures shown in Fig. 29. The variation of irregular degrees of the initial quad set is 184. The final optimized quad set is shown in Fig. 29(e) and its variation of irregular degrees is reduced to 159.

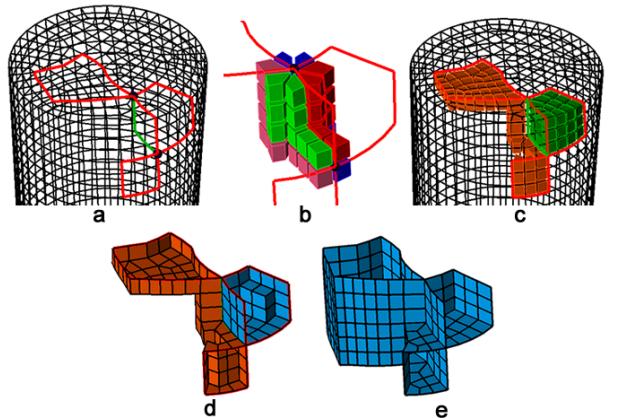


Figure 29: Process of the determination of the quad set: (a) the intersecting line is determined; (b) the int-4-hex-sets are constructed; (c) the merged hex sets; (d) the initial quad set; (e) the optimized quad set.

The sheet inflation is conducted after getting the quad set.

608 The new sheet is shown in Fig 30(a) and Fig. 30(b). The two  
 609 nodes  $v_1$  and  $v_2$  are actually splitted into two nodes and four  
 610 nodes respectively as shown in Fig. 30(c). The new edges split-  
 611 ted from  $e_1$  and  $e_2$  are shown in Fig. 30(b). There has no longer  
 612 any nodes or edges with valences larger than 5.

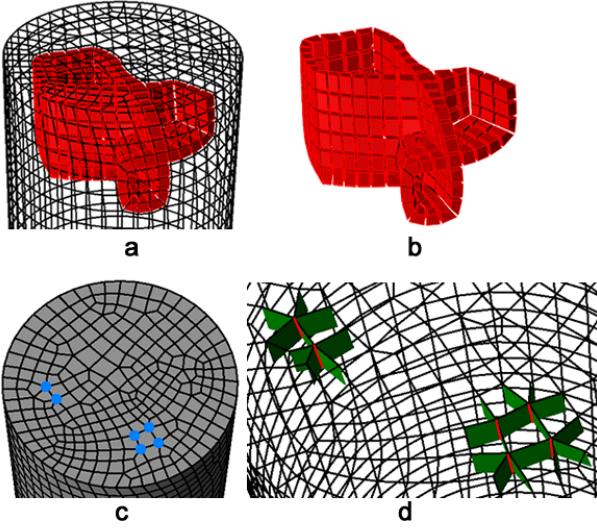


Figure 30: Results of high valences reduction: (a) the new sheet inflated from the quad set in the hex mesh; (b) the new sheet; (c) the new nodes with valences no larger than 5; (d) new edges with valences no larger than 5.

## 613 7. Conclusions and Future Work

614 In this paper, a new approach is proposed to achieving opti-  
 615 mized complex sheet inflation under various constraints. Main  
 616 features of our approach are summarized:

- 617 1. By using A\* and max-flow-min-cut algorithms, the valid  
 618 boundary loop of the quad set for sheet inflation is ef-  
 619 fectively determined, which satisfies the boundary con-  
 620 straints.
- 621 2. With intersecting nodes on the boundary loop being rea-  
 622 sonably paired by using two templates, intersecting lines  
 623 are validly constructed between each pair of nodes by us-  
 624 ing A\* algorithm, and the int-4-hex-sets are determined  
 625 around the intersecting lines, enabling our approach to ef-  
 626 fectively construct complex quad sets that intersect them-  
 627 selves more than once.
- 628 3. By using max-flow-min-cut algorithm, the valid initial  
 629 quad set is effectively and efficiently determined within the  
 630 local region specified by the user.
- 631 4. A chord-based optimization method is proposed which can  
 632 effectively improve the quality of the quad set while avoid-  
 633 ing the difficulty in optimizing the quad set as a whole.

634 The examples demonstrate that our approach can success-  
 635 fully generate complex sheets within a local region. These two  
 636 examples also present two applications that our approach can be  
 637 used: one is modifying the interfaces in mesh matching and the

638 other is reducing node valences and edge valences to improve  
 639 mesh quality.

640 The shortcomings of our approach and future work are listed  
 641 below:

- 642 1. It is not very convenient to generate self-touching sheets.  
 643 In some rare cases, self-touching sheets need to be gen-  
 644 erated. By combining other dual operations like column  
 645 collapse and sheet extraction, our approach is able to cre-  
 646 ate self-touching sheets. However, currently it is not very  
 647 convenient. We will propose simpler solutions to generate  
 648 self-touching sheets;
- 649 2. Currently the constraints can only be specified by bound-  
 650 ary edges and hexahedra. However, sometimes the user  
 651 needs to specify edges or quads inside the hex mesh to con-  
 652 trol the sheet inflation, e.g. improving the edge valences  
 653 inside the hex mesh. Hence, we plan to adapt our approach  
 654 to accept constraints specified inside the hex mesh.

## 655 References

- 656 [1] Mitchell SA, Tautges TJ. Pillowing doublets: refining a mesh to ensure  
 657 that faces share at most one edge. 4th International Meshing Roundtable  
 658 1995;.
- 659 [2] Shepherd JF. Topologic and geometric constraint-based hexahedral mesh  
 660 generation. Ph.D. thesis; School of Computing, University of Utah; Salt  
 661 Lake City, UT, USA; 2007.
- 662 [3] Shepherd JF, Johnson CR. Hexahedral mesh generation constraints. En-  
 663 gineering with Computers 2008;24(3):195–213.
- 664 [4] Murdoch P, Benzley S, Blacker T, Mitchell SA. The spatial twist con-  
 665 tinuum: A connectivity based method for representing all-hexahedral  
 666 finite element meshes. Finite Elements in Analysis and Design  
 667 1997;28(2):137–49.
- 668 [5] Tautges TJ, Knop SE. Topology Modification of Hexahedral Meshes  
 669 Using Atomic Dual-based Operations. IMR 2003;:415–23.
- 670 [6] Ledoux F, Shepherd J. Topological modifications of hexahedral meshes  
 671 via sheet operations: a theoretical study. Engineering with Computers  
 672 2009;26(4):433–47.
- 673 [7] Ramos JS, Ruiz-Gironés E, Navarro XR. Unstructured and semi-  
 674 structured hexahedral mesh generation methods 2014;.
- 675 [8] Suzuki T, Takahashi S, Shepherd J. An interior surface genera-  
 676 tion method for all-hexahedral meshing. Engineering with Computers  
 677 2010;26(3):303–16.
- 678 [9] Staten ML, Shepherd JF, Ledoux F, Shimada K. Hexahedral Mesh Match-  
 679 ing: Converting non-conforming hexahedral-to-hexahedral interfaces into  
 680 conforming interfaces. International Journal for Numerical Methods in  
 681 Engineering 2009;:1475–509.
- 682 [10] Chen J, Gao S, Zhu H. An improved hexahedral mesh matching algo-  
 683 rithm. Engineering with Computers 2015;doi:10.1007/s00366-015-0414-  
 684 1.
- 685 [11] Staten ML, Shimada K. A close look at valences in hexahedral element  
 686 meshes. International Journal for Numerical Methods in Engineering  
 687 2010;:899914doi:10.1002/nme.2876.
- 688 [12] Mitchell S. A characterization of the quadrilateral meshes of a sur-  
 689 face which admit a compatible hexahedral mesh of the enclosed volume.  
 690 STACS 96 1996;.
- 691 [13] Kowalski N, Ledoux F, Staten ML, Owen SJ. Fun sheet matching: to-  
 692 wards automatic block decomposition for hexahedral meshes. Engineering  
 693 with Computers 2011;doi:10.1007/s00366-010-0207-5.
- 694 [14] Owen S. A survey of unstructured mesh generation technology. 7th In-  
 695 ternational Meshing Roundtable 1998;.
- 696 [15] Tarini M, Pietroni N, Cignoni P, Panozzo D, Puppo E. Practical quad  
 697 mesh simplification. Computer Graphics Forum 2010;29(2):407–18.  
 698 doi:10.1111/j.1467-8659.2009.01610.x.