

An approach to achieving optimized complex sheet inflation under constraints

Abstract

Sheet inflation is an enhanced and more general version of the classic pillowining procedure[1] used to modify hexahedral 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, hexahedral 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]. Figure 1 shows an example that even though we want to make small local modification as adding one new quad to the boundary of the hex mesh(Fig. 1b), a circle of quads on the boundary(Fig. 1c) as well as a set of hex have to be generated in order to keep the hex mesh valid(Fig. 1d). The dual structures, which contribute to the global connectivity of hex meshes, are called sheets. The set of hex in Fig. 1d is actually a sheet.

More specifically, starting from one mesh edge, we recursively get all its topologically parallel mesh edges. All of these mesh edges along with the adjacent hex define a sheet. In addition to its primal representation as a composition of vertices, edges, faces and hexahedra, a hex mesh can also be seen as a set of intersecting sheets, known as Spatial Twisted Continuum[4].

Therefore, as a set of operations that directly and effectively deal with the sheets, sheet operations attract more and more attention in recent years. The most common sheet operations include pillowining[1], dicing[8], column collapse[9], sheet extraction[? 9] and sheet inflation[10, 11, 9].

Figure 2 shows the process of these dual operations. Pillowining takes a hex set, which is called shrinking set, as input, and then shrinks the hex set and fills the gap with new hex between the shrinking set and the rest of the mesh. Dicing duplicates a sheet by splitting its topologically parallel mesh edges that defines the sheet. Column collapse can change the sheets' shape

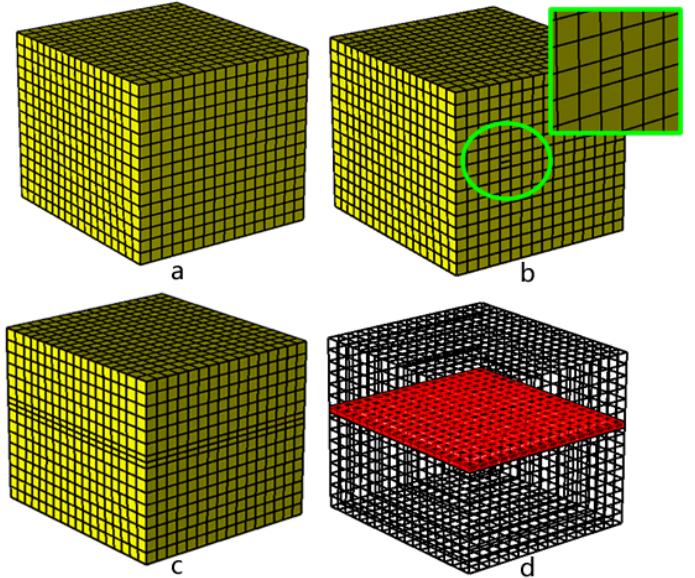


Figure 1: An example of local modification resulting in global change: (a) the original hex mesh; (b) local modification by adding a quad on the mesh boundary; (c) a circle of quads on the mesh boundary have to be added; (d) a set of hexahedra have to be added.

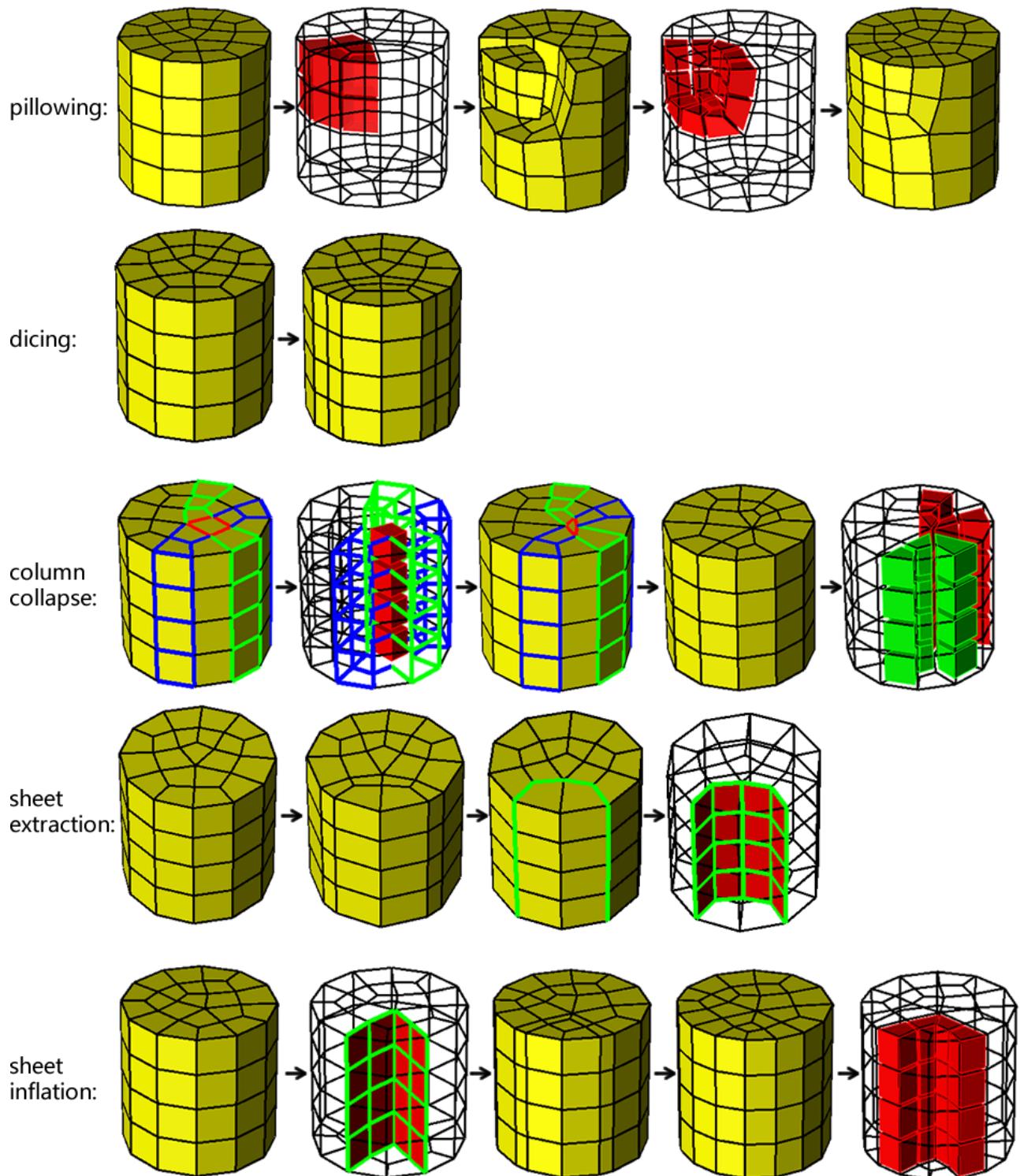


Figure 2: Common dual operations: pillowing, dicing, column collapse, sheet extraction and sheet inflation.

36 or topologies by collapsing the column, which is the hex set
 37 where one sheet intersects with itself or another sheet. Sheet
 38 extraction extracts the sheet by merging the two mesh nodes on
 39 each of the mesh edges that define the sheet, resulting a contin-
 40 uous set of quads.

41 Sheet inflation can be seen as the reverse of sheet extraction,
 42 which "inflates" a continuous quad set into a new sheet. The
 43 quad set is critical for sheet inflation because it determines the
 44 position, shape and topology of the sheet to be generated. Since
 45 topologically any sheet can be extracted (though sometimes ex-
 46 traction may not be possible due to geometrical restrictions),
 47 sheet inflation has the potential ability to create any kind of
 48 sheets, provided that a suitable quad set are determined.

49 Due to the versatility to create various sheets, it is very
 50 promising to utilize sheet inflation in many mesh modification
 51 scenarios. One common scenario is to use sheet inflation to lo-
 52 cally change the mesh topology, especially the boundary topol-
 53 ogy. To meet the modification requirements, new sheets need
 54 to be created at the specified position within a delimited region.
 55 As the quad set plays a key role for sheet inflation, the main
 56 difficulty to achieve this is how to construct a qualified quad set
 57 under the user's constraints. In practice, these constraints are
 58 usually specified by a set of boundary edges and a set of hexa-
 59 hedra. The former determines the positions where the new sheet
 60 should appear on the mesh boundary, and the latter delimits the
 61 region where the new sheet can propagate.

62 Figure 3 shows an example when mesh modification on the
 63 boundary is required. Node valences (the number of adjacent
 64 edges of a node) and edge valences (the number of adjacent
 65 faces of an edge) are important quality metrics for quad meshes
 66 and hex meshes respectively, and high valences usually lead
 67 to poor mesh quality [12, 13, 14, 15]. Although valences of
 68 5 are relatively acceptable, valences equal to or higher than 6
 69 are usually considered to be undesirable. The hex mesh (Fig.
 70 3a) contains one boundary node whose valence is 6. This high
 71 valence can be reduced by splitting the node into two nodes
 72 with lower valences as shown in Fig. 3b.

73 In recognizing the importance of sheet inflation, many re-
 74 searchers have been investigating algorithms for decades and
 75 have achieved great progress. Mitchell et al. proposed the Pil-
 76 lowing algorithm to deal with the doublet problem[1]. This
 77 algorithm involves inflating a new sheet guided by a hex set
 78 called shrinking set. Its simplicity and effectiveness makes Pil-
 79 lowing prevalent in mesh modification, especially in mesh qual-
 80 ity improvement. Although it can be adapted to satisfy simple
 81 constraints, the Pillowing lacks the ability to generate complex
 82 sheets such as self-intersecting sheets.

83 Suzuki et al. introduced a method to construct interior sheet
 84 surfaces when the boundary dual cycles are given[16]. They
 85 used this method to firstly construct dual structures and then
 86 deduce the primal hex meshes from these dual structures. While
 87 in the paper they illustrated the topological structures of sheet
 88 surfaces in the dual space, particularly self-intersecting sheet
 89 surfaces, they didn't mention how to determine the quad set
 90 and generate the corresponding sheets on the primal hex mesh.

91 Staten et al. proposed the General Sheet Inflation method
 92 which explained in detail how to do sheet inflation on hex

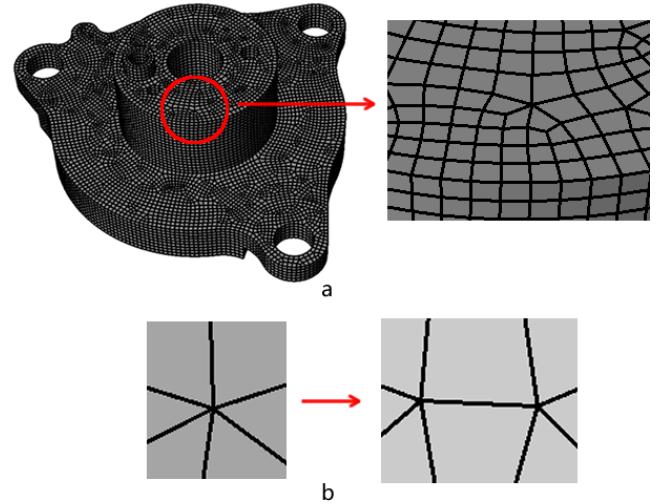


Figure 3: An example when boundary modification is required: (a) the hex mesh with a high-valence node; (b) the high valence is reduced by splitting the node into two nodes.

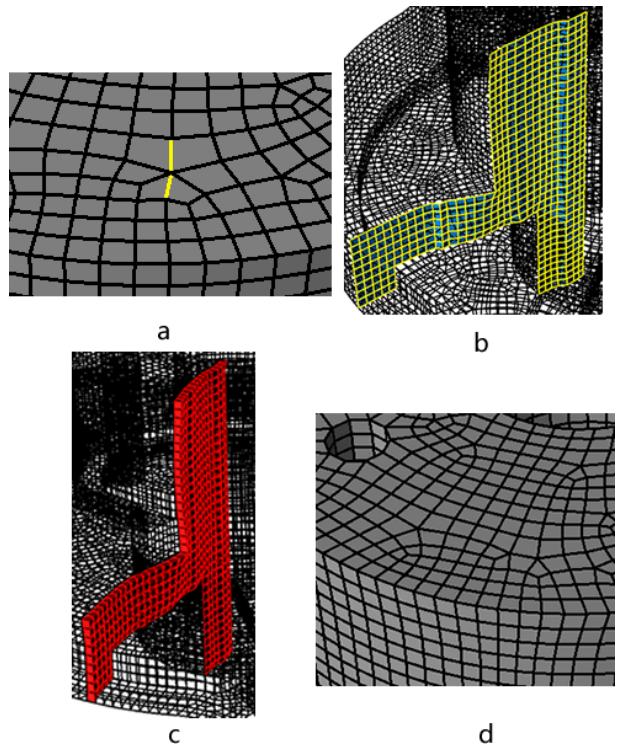


Figure 4: Reducing high valence by non-localized sheet inflation : (a) selected mesh edges where new sheet needs inflating; (b) the quad set for sheet inflation without local consideration; (c) the new sheet; (d) the high valence is improved.

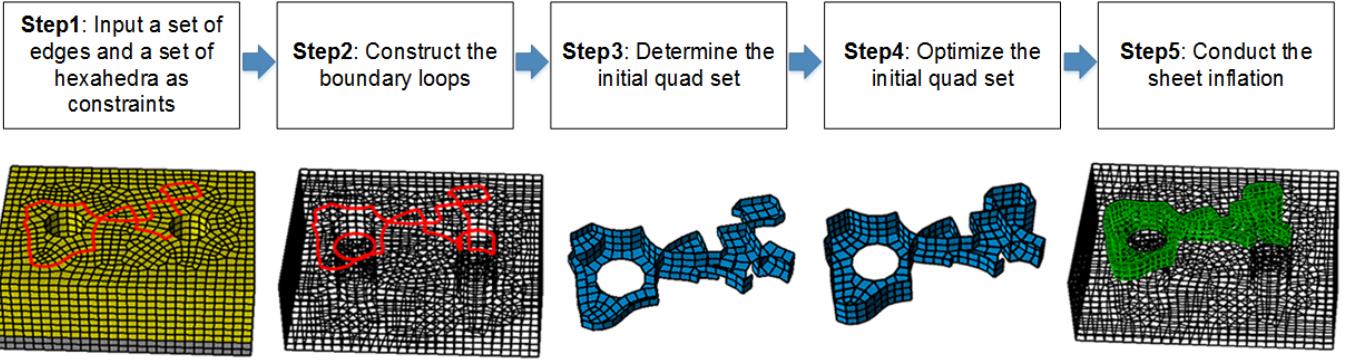


Figure 5: Overview of the approach.

meshes when boundary mesh edges are specified[9]. This method can create normal, self-touching and self-intersecting sheets. However, determining the quad sets for sheet inflation in this method seems to rely on the sweeping structures of the mesh. Meanwhile it is not clear whether the method can create self-intersecting sheets within a local region.

Chen et al. proposed a new approach to inflate sheets when conducting mesh matching on complex interfaces[17]. This method firstly constructs the boundary loop of the quad set, and then determines the interior quad set. Although it can locally generate self-intersecting sheets under constraints, it allows the sheets to intersect themselves once at most. Meanwhile, the optimization method it applies on quad set cannot guarantee the quality improvement.

Despite these achievements, to inflate complex sheets under various constraints, especially to generate self-intersecting sheets within a local region, is still very difficult. Furthermore, how to assure the mesh quality for complex sheet inflation is also a challenging problem. In this paper we propose a new approach to achieving complex sheet inflation. By firstly determining the boundary loop and then utilizing the max-flow-min-cut algorithm, we construct an initial quad set that fulfills all the constraints. By applying a chord-based optimization method, we improve the quality of the quad set and thus the mesh quality after sheet inflation is guaranteed.

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

2. Overview of the Approach

In this paper, an approach is proposed to achieving complex sheet inflation under various constraints while assuring the mesh quality. It takes a set of boundary mesh edges and a set of hexahedra as input. The edges and hexahedra specify the boundary position and local region of the sheet. Under these constraints, it constructs a qualified quad set and then creates

the new sheet as output. To avoid the difficulty of determining a qualified inflatable quad set directly, our approach involves three major steps: (a) determining the boundary loops for the quad set that satisfy the boundary constraints; (b) constructing the initial quad set based on the boundary loops; (c) optimizing the initial quad set. Figure 5 shows the overview of our approach.

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

To solve these two problems, our basic ideas are:

1. We use path searching and max-flow-min-cut algorithms to construct the valid boundary loop and initial quad set that meet the various boundary constraints. In accordance with the inherent characteristics of valid boundary loops and quad sets, loops can be formed by the path searching algorithm if the boundary constraints are not loops, and intersecting lines on the quad set can also be determined by the path searching algorithm. The valid boundary loop and initial quad set can be effectively determined by the max-flow-min-cut algorithm due to the fact that the determination of these structures on the mesh is analogous to finding a cut set on a graph;
2. We optimize the initial quad set by optimizing its chords. The chords, the dual structures of the quad set, are more global than individual edges while less constrained than the whole quad set. By optimizing the chords, we can effectively improve the quality of the quad set while avoiding the difficulty in optimizing the quad set as a whole.

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

3. Determination of the Boundary Loops

It is quite difficult to find a qualified quad set satisfying all the constraints directly. The quad set's boundary loop is ac-

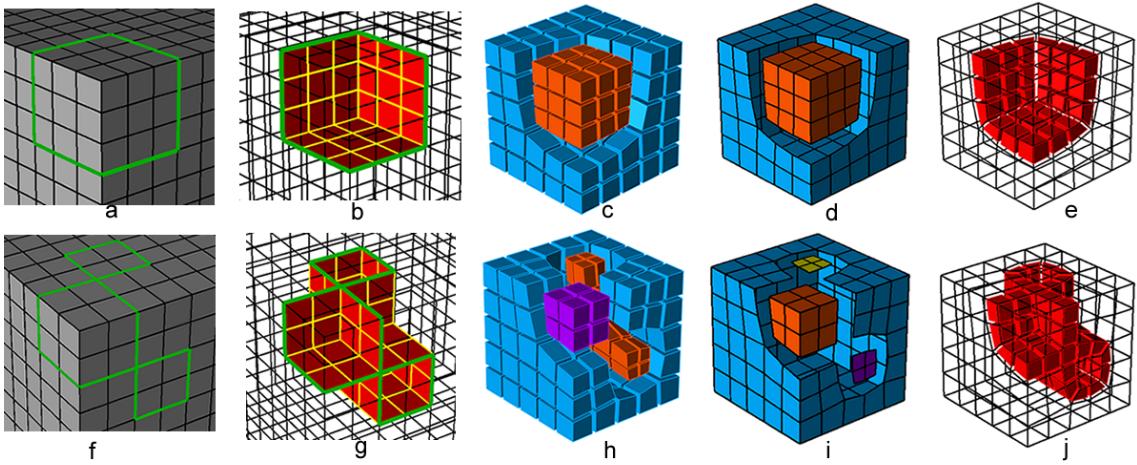


Figure 6: Characteristics of valid boundary loops: (a) a non-self-intersecting boundary loop; (b) the non-self-intersecting quad set; (c) two hex sets separated by the quad set; (d) two boundary quad sets separated by the boundary loop; (e) the new non-self-intersecting sheet; (f) a self-intersecting 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.

cordingly constructed at first. This section firstly discusses the characteristics of valid boundary loops and quad sets, and then presents the method to construct a valid boundary loop under the boundary constraints.

By observation of the structures of valid quad sets and the process of sheet inflation, an inflatable quad set and its boundary loop have two characteristics:

1. An inflatable quad set separates its local hex set into $2 + N$ subsets, where N stands for how many times the quad set intersects itself. For example, the non-self-intersecting quad set in Fig. 6(b) separates the local hex set into two parts (Fig. 6(c)). The self-intersecting quad set in Fig. 6(g) separates the local hex set into 3 subsets (Fig. 6(h)). Meanwhile, the boundary loop of the quad set separates the boundary of the local hex set into $2 + n$ subsets, where n is the number of self-intersecting nodes on the boundary loop. The non-self-intersecting boundary loop in Fig. 6(a) separates the boundary of the local hex set into two subsets (Fig. 6(d)), and the self-intersecting boundary loop in Fig. 6(f) separates the boundary of the local hex set into 4 subsets (Fig. 6(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. 6(c), Fig. 6(e), Fig. 6(h) and Fig. 6(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

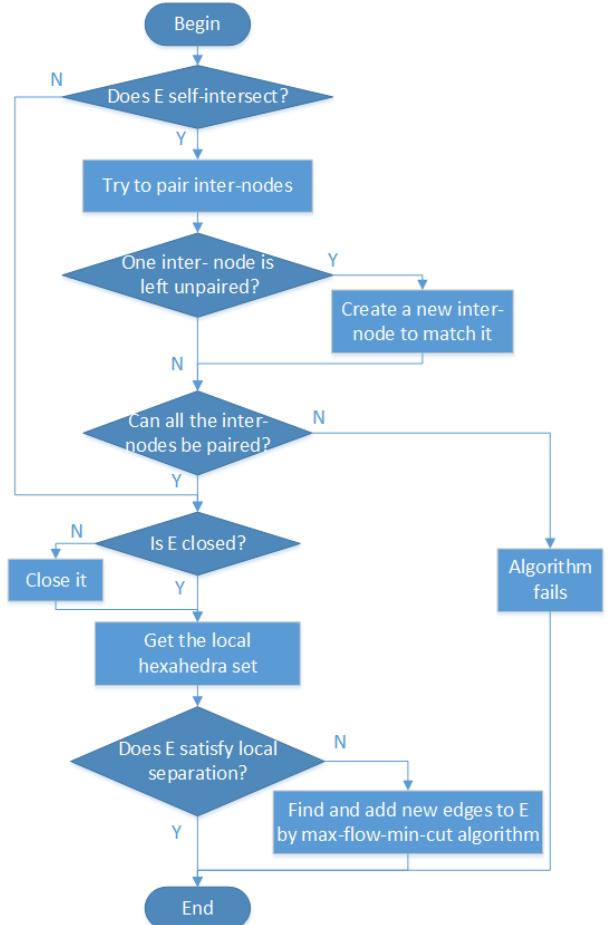


Figure 7: Flowchart of the determination of boundary loops.

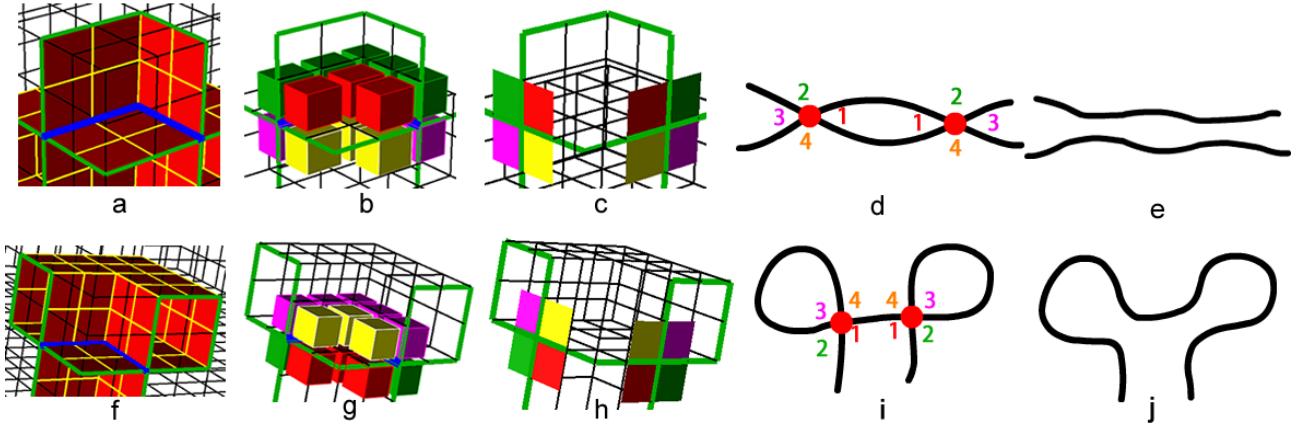


Figure 8: Two local templates for pairing intersecting nodes: (a) the 1st local intersecting structure of the quad set; (b) int-4-hex-sets around the 1st local intersecting structure of the quad set; (c) the associated local structure of the boundary loop; (d) the 1st template; (e) intersecting nodes 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.

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. 7. The rest of this section will provide detailed description about the method of the determination of boundary loops.

3.1. Pairing Intersecting Nodes

For self-intersecting quad sets, the intersecting lines play a very important role in defining the structures of the quad sets. The intersecting nodes on the boundary lines need to be paired in order to construct the intersecting line between each pair of intersecting nodes. It needs decision which two can be paired when there are more than two intersecting nodes. Improper pairing of intersecting nodes will result in non-orientable surface with very poor quality or even no surface at all[16]. Based on the observation of the structures of quadrilateral sets, we introduce 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. 8(a) and Fig. 8(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. 8(d) and Fig. 8(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 [16]. To recursively find next pair of intersecting nodes, the local topological structures are modified accordingly as shown in Fig. 8(e) and Fig. 8(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. 8(b) and Fig. 8(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. 8(c) and Fig. 8(h). The quads of same color mean they are adjacent to the same int-4-hex-set. The numbers in different colors on the two templates in Fig. 8(d) and Fig. 8(i) indicate the corresponding int-4-hex-sets.

Two examples in Fig. 9 are presented to explain how to pair intersecting nodes by these two templates.

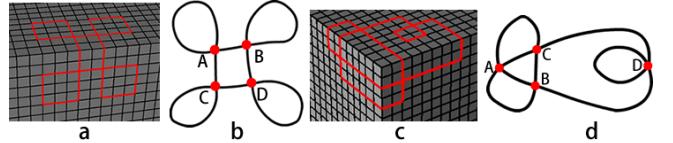


Figure 9: Two examples of pairing intersecting nodes: (a) the boundary loop of the 1st example; (b) the topological structure of the boundary loop of the 1st example; (c) the boundary loop of the 2nd example; (d) the topological structure of the boundary loop of the 2nd example.

The pairing process for the 1st example is shown in Fig. 10. 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.

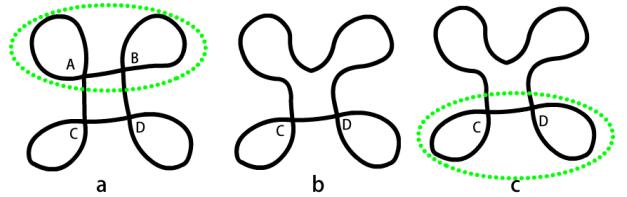


Figure 10: The pairing process of the 1st example: (a) node A and B are paired; (b) node A and B are resolved; (c) node C and D are paired.

The pairing process for the 2nd example is shown in Fig. 11. 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.

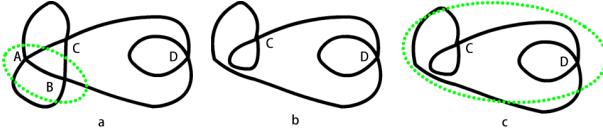


Figure 11: The pairing process of the 2nd example: (a) node A and B are paired; (b) node A and B are resolved; (c) point C and D are paired.

After recursively applying these two templates, if there is still one intersecting node left unpaired, we use the templates to create a new intersecting node at an appropriate position to be paired with this intersecting node. For example, in Fig. 12(a), the intersecting node A is unpaired. We use the 2nd template to create a new intersecting node B to be paired with A as shown in Fig. 12(b).

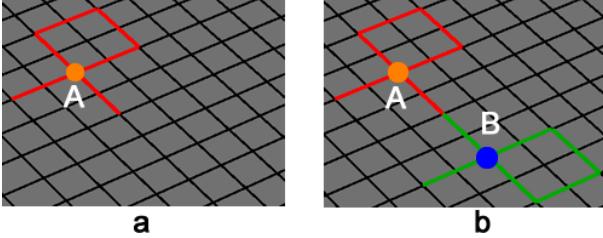


Figure 12: Handling of the situation when single intersecting node is unpaired: (a) intersecting node A is unpaired; (b) intersecting node B is created to be paired with A .

Our algorithm would fail if some intersecting nodes are unable to be paired by the two templates. These boundary loops usually determine non-orientable quad sets[16] which will largely degenerate the mesh quality after sheet inflation. Therefore we do not handle such kind of boundary loops in this paper.

3.2. Forming Closed Boundary Loops

The ability to do local separation requires the constraint edges to form closed loops. Sometimes the constraint edges, which are specified by the user according to specific mesh modification requirements, may not be closed loops. When this happens, we need to close the constraint edges by adding new edges on the mesh boundary. This is done by connecting the dangling edges on the constraint edges using the A* path searching algorithm.

A* path searching is a one-source-one-target path searching algorithm which runs very efficiently. When applying A* algorithm, in addition to considering the count of mesh edges on the path, we also take the angles between adjacent edges on the path into consideration in order to get a smooth path. Figure 13(b) shows the result when the angles between adjacent edges are not considered, and Fig. 13(c) shows a better result when the angles are taken into consideration.

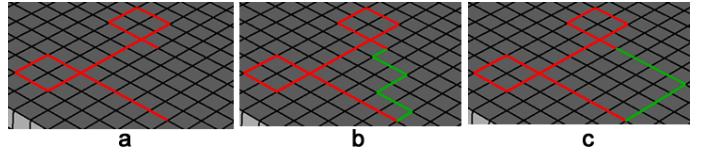


Figure 13: Closing the boundary loop: (a) the unclosed boundary loop; (b) new edges determined by A* algorithm without considering the turning angles; (c) new edges determined by A* algorithm considering the turning angles.

3.3. Making the Boundary Loop Able to Do Local Separation

After previous sections, the constraint edges specified by the user become a closed boundary loop with intersecting nodes being paired. According to the discussion in the beginning of Section 3, a valid boundary loop should be able to do local separation. Sometimes the closed boundary loop may fail to do local separation due to the existence of through holes on the hex mesh. To fix that, we use the max-flow-min-cut algorithm to add necessary edges to the boundary loop to make it able to do local separation. Suppose the boundary loop is E , the algorithm flowchart is illustrated in Fig. 14.

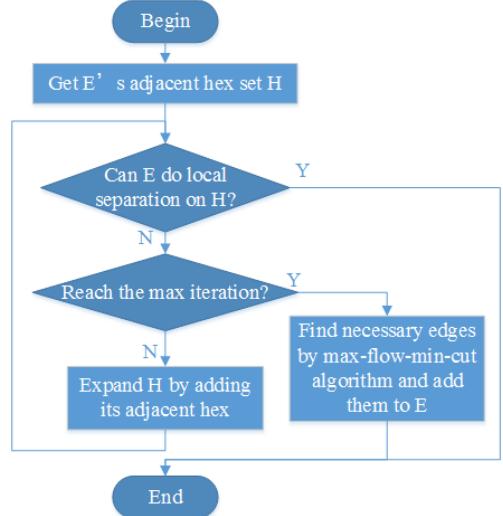


Figure 14: Flowchart of the process of making the boundary loop able to do local separation.

The following example illustrates the process. Figure 15(a) shows the hex mesh containing a through hole and the boundary loop E (red). To check whether E is able to do local separation, we firstly get E 's adjacent hex set H (Fig. 15(b)) and iteratively add H 's adjacent hexahedra to H and test whether the new boundary of H can be separated by E into two subsets until reaching the maximum iteration times (Fig. 15(c)). However, E is still unable to do local separation on H since there are still only one quad set (blue in Fig. 15(d)) on H 's boundary. Hence, we need to use the max-flow-min-cut algorithm to add other necessary mesh edges to E to make it able to do local separation.

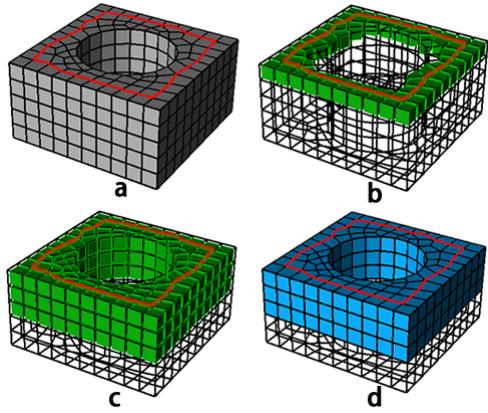


Figure 15: An example of boundary loops that are unable to do local separation: (a) the boundary loop E ; (b) E 's adjacent hexahedra H (green); (c) expand H by iteratively adding its adjacent hexahedra; (d) E still cannot separate H' 's boundary into two subsets when reaching maximum expanding iteration times.

311 Max-flow-min-cut algorithm is an effective and efficient tool
312 to find the minimal cut set in a directed graph[18].

313 As shown in Fig. 16(a), to use the max-flow-min-cut algo-
314 rithm, a directed graph should be constructed with an s node
315 and a t node, and the weights of the edges should also be set
316 before calculation. The algorithm will find a minimal cut set
317 that determines the max flow from s to t .

318 If we take the boundary of H as a graph where the nodes
319 stand for a quad or a quad set and the edges stand for the ad-
320 jacency relationship between the quads or quad sets, then to be
321 able to do local separation is similar to find a cut set on this
322 graph. Hence, the max-flow-min-cut algorithm can be used to
323 help us to find necessary edges to make the boundary loop sat-
324 isfy the local separation.

325 To use the algorithm, we need to construct the directed graph
326 at first, including constructing the s and t nodes and deciding
327 the edges' weights. As the current boundary loop E must be
328 part of the final boundary loop, the two quad sets on its two
329 sides just can be the s and t nodes as shown in Fig. 16(b). Since
330 all of the edges on the final boundary loop should be on the
331 boundary of the hex mesh, the quads shared by H and the rest
332 of the hex mesh, which is shown as Q_{in} in Fig. 16(b), should be
333 accordingly grouped into a single node in the graph to guarantee
334 that Q_{in} won't be separated by the min cut. Similarly, in order to
335 forbid the mesh edges on sharp CAD model edges to be added
336 in to the boundary loop, which usually results in poor mesh
337 quality, we group the quads sharing these mesh edges into a
338 single node, e.g. q_2 and q_3 in Fig. 16(c).

339 After constructing the nodes, we assign weights to the graph
340 edges according to how many mesh edges are shared by the
341 two nodes. For example, q_1 in Fig. 16(c) shares two edges with
342 node t , so the weight of the directed edges in the graph between
343 q_1 and t is 2. All of the edges in the graph are bi-directional
344 except the edges adjacent to s and t .

345 The final directed graph is shown in Fig. 16(d). After cal-
346 culation, new edges are found as shown in Fig. 16(e), and now
347 E is able to do local separation after adding these new edges as

348 shown in Fig. 16(f).

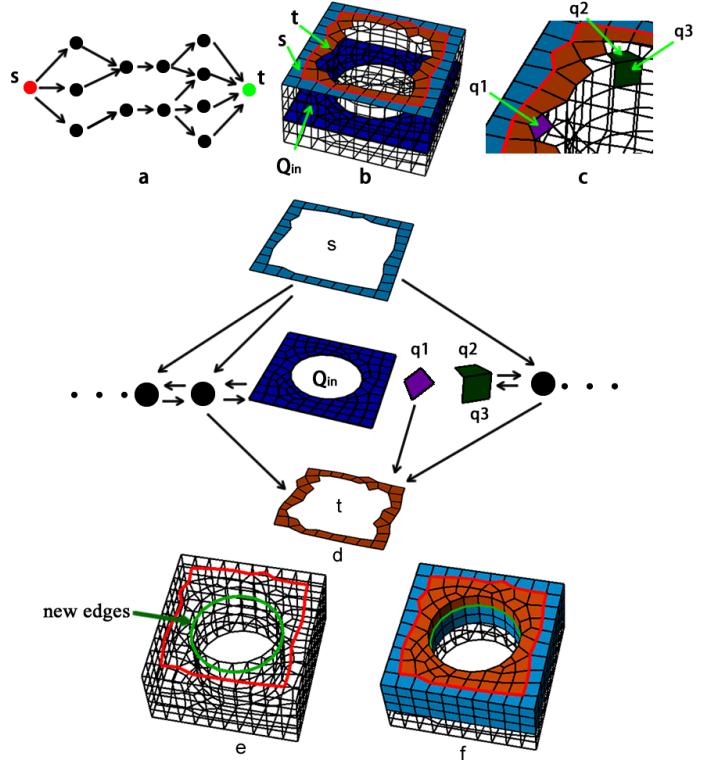


Figure 16: Using max-flow-min-cut algorithm to find necessary edges for local separation: (a) directed graph used for ordinary max-flow-min-cut problem; (b) the quad sets on E 's two sides can be s and t nodes; (c) q_1 shares two edges with t , q_2 and q_3 share an edge on the geometric hard edge; (d) 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.

349 When a self-intersecting boundary loop is unable to do local
350 separation, we also need to apply max-flow-min-cut algorithm
351 to find necessary new edges to make it able to do local separa-
352 tion. The procedures are quite similar to non-self-intersecting
353 boundary loop discussed previously. We treat a self-intersecting
354 boundary loop as several non-self-intersecting boundary loops
355 and handle each of them individually. The self-intersecting
356 boundary loop in Fig. 17(a) consists of three sub loops $loop_1$,
357 $loop_2$ and $loop_3$. The local hex set is shown in Fig. 17(b). Due
358 to the existence of three through holes on the hex mesh, the
359 boundary loop is unable to do local separation. After applying
360 max-flow-min-cut algorithm to $loop_1$ and $loop_3$, new edges are
361 found as shown in Fig. 17(c).

362 4. Determination of the Initial Quad Set

363 This section provides specific details of the determination of
364 the initial quad set based on the boundary loop. The intersect-
365 ing lines are firstly constructed according to the pairing of the
366 intersecting nodes. The initial quad set is then constructed us-
367 ing the max-flow-min-cut algorithm. Since the boundary loops
368 satisfy the boundary constraints and the max-flow-min-cut al-
369 gorithm is conducted within the local region specified by the
370 user, the initial quad set fulfills all the constraints.

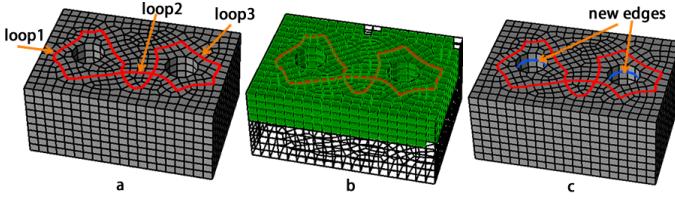


Figure 17: Making self-intersecting boundary loop able to do local separation: (a) the self-intersecting boundary loop with three sub loops; (b) the local hex set; (c) new edges (blue) are found by using max-flow-min-cut algorithm.

371 4.1. Determination of Intersecting Lines

372 Intersecting lines are very important for defining the structures of self-intersecting quad sets. Before determining the initial quad set, we need to construct the intersecting lines first. In 373 Section 3.1 the two templates not only make two intersecting nodes be paired but also setup the corresponding relationships 374 between the int-4-quad-sets of the two intersecting nodes. In 375 this section, we use this information to construct the intersecting lines and the int-4-hex-sets of these intersecting lines.

380 The following example shows the process of constructing the 381 intersecting line and its int-4-hex-sets. In Fig. 18(a), the blue 382 nodes are two intersecting nodes paired by the 1st template (in 383 this example they can also be paired by the 2nd template), and 384 the numbers and different colors denote the correspondence be- 385 tween the int-4-quad-sets of the two points. We use A* algo- 386 rithm to determine the intersecting line between the two inter- 387 secting nodes. Similar to Section 3.2, in order to get a smooth 388 path, we take the turning angles between adjacent edges in the 389 path into consideration while applying the A* searching. The 390 intersecting line is shown in Fig. 18(b). Next, we get all the 391 hexahedra adjacent to the intersecting line, and use the corre- 392 spondence between the int-4-quad-sets to construct the int-4- 393 hex-sets as shown in Fig. 18(c).

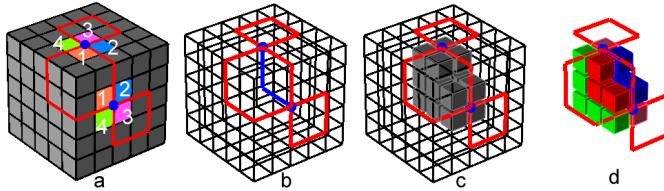


Figure 18: Process of constructing the intersecting line and its int-4-hex-sets: (a) two intersecting nodes paired by the 1st template; (b) the intersecting line determined by A* algorithm; (c) the hexahedra adjacent to the intersecting line; (d) the int-4-hex-sets of the intersecting line.

394 4.2. Determination of Initial Quad Set by Max-flow-min-cut Al- 395 gorithm

396 After constructing the intersecting lines and their int-4-hex- 397 sets, we now construct the initial quad set. The idea is similar to 398 that mentioned in Section 3.3, we convert the problem from be- 399 ing able to do local separation to acquiring min cut on a graph. 400 If we see the hexahedra as nodes and the adjacency between the 401 hexahedra as edges of the graph, the quad set is actually a cut set

402 of the graph. Hence, max-flow-min-cut algorithm can be used 403 to efficiently get a quad set that is able to do local separation. 404 This section provides specific details about this process.

405 As the boundary loop separates the boundary of the local hex 406 set into $2+n$ subsets (n is the number of intersecting nodes), we 407 can get $2+n$ hex sets that are adjacent to these $2+n$ quad sets. 408 From these $2+n$ hex sets and the int-4-hex-sets of the inter- 409 secting lines, we merge hex sets that share common hexahedra. 410 This will result in $2+N$ hex sets (N is the number of intersecting 411 lines). We then apply the max-flow-min-cut algorithm multiple 412 times to get the initial quad set. The pseudo-codes are provided 413 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 ad- jacent to $Q_i, i = 1, 2, \dots, 2+n$;
- 5: $H_{int} = \{H_{ini1}^1, H_{ini1}^2, H_{ini1}^3, H_{ini1}^4, \dots, H_{inN}^1, H_{inN}^2, H_{inN}^3, H_{inN}^4\}$ where $\{H_{ini1}^1, H_{ini1}^2, H_{ini1}^3, H_{ini1}^4\}$ 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**

414 Two examples are provided to illustrate the procedures of Al- 415 gorithm 1. The first example is a non-self-intersecting boundary 416 loop as shown in Fig. 19. n and N are both 0 for this example. 417 The two quad subsets on the boundary of the local hex set H 418 are Q_{b1} and Q_{b2} as shown in Fig. 19(c). We get the two hex 419 sets adjacent to Q_{b1} and Q_{b2} as H_{b1} and H_{b2} (Fig. 19(d)). We 420 construct the directed graph by taking H_{b1} as s node and H_{b2} as 421 t node, taking other hexahedra as normal nodes, and assigning 422 weights to the edges according to the number of quads shared 423 by two hex sets. After running the max-flow-min-cut algorithm, 424 we get the initial quad set Q_{init} as shown in Fig. 19(e).

425 The second example is a self-intersecting boundary loop as 426 shown in Fig. 20. The boundary loop (red in Fig. 20(a)) 427 intersects itself once, hence $n = 2, N = 1$. Firstly we 428 get the int-4-hex-sets $i4h_1-i4h_4$ (Fig. 20(b)). The bound- 429 ary of the local hex set H is separated into $Q_{b1}-Q_{b4}$ (Fig. 430 20(c)), and the hex sets adjacent to these quad subsets are 431 $H_{b1}-H_{b4}$ (Fig. 20(d) and Fig. 20(e)). Therefore $H_{merge} = 432 H_{int} \cup H_{bound} = \{i4h_1, \dots, i4h_4, H_{b1}, \dots, H_{b4}\}$. After merging,

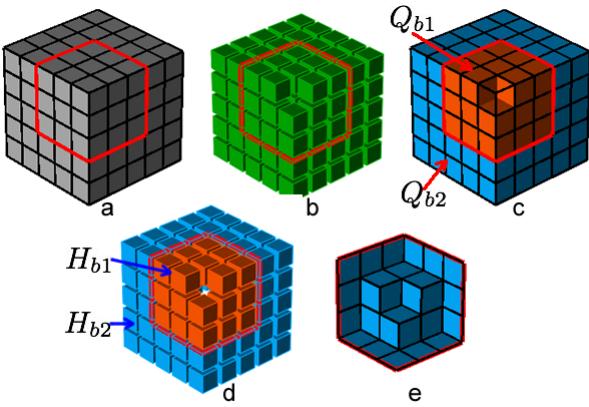


Figure 19: Process of the determination of the initial quad set from a non-intersecting boundary loop: (a) the non-intersecting boundary loop; (b) the local hex set; (c) the two boundary quad subsets Q_{b1} (orange) and Q_{b2} (blue); (d) two hex sets H_{b1} (orange) and H_{b2} (blue); (e) the initial quad set Q_{init} .

⁴³³ $H_{merge} = \{H_{m1}, H_{m2}, H_{m3}\}$ (Fig. 20(f) and Fig. 20(g)). We then
⁴³⁴ conduct the max-flow-min-cut algorithm twice. For the first
⁴³⁵ time, we take H_{m1} as s node and $H_{m2} \cup H_{m3}$ as t node (Fig.
⁴³⁶ 20(h)), and get the quad set Q_1 (Fig. 20(i)). For the second
⁴³⁷ time, we take H_{m2} as s node and $H_{m1} \cup H_{m3}$ as t node (Fig.
⁴³⁸ 20(j)), and get the quad set Q_2 (Fig. 20(k)). Finally the initial
⁴³⁹ quad set is $Q_{init} = Q_1 \cup Q_2$ (Fig. 20(l)).

5. Optimization of the Initial Quad Set

⁴⁴¹ Although the initial quad set determined by previous section
⁴⁴² satisfies all the constraints specified by the user, the quality of
⁴⁴³ the mesh after sheet inflation is usually not good enough. In
⁴⁴⁴ this paper we propose a new optimization method for the quad
⁴⁴⁵ set based on the quad set's dual structures. In this section we
⁴⁴⁶ discuss the details of this optimization method.

5.1. Quality Evaluation of the Quad Set

⁴⁴⁸ In a hex mesh, the valence of a mesh edge stands for the count
⁴⁴⁹ of its adjacent quads. A mesh edge is called a regular edge if
⁴⁵⁰ it is inside the mesh or on the mesh boundary and its valence
⁴⁵¹ equals 4 or 3 respectively. Otherwise the edge is called an ir-
⁴⁵² regular edge. The irregular degree is the difference of valences
⁴⁵³ between one edge and its corresponding regular edge. Suppose
⁴⁵⁴ e stands for an edge and v_e stands for its valence, the irregular
⁴⁵⁵ degree of e is computed as Equation 1.

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

⁴⁵⁶ Staten et al. showed that the edge valence has direct impact
⁴⁵⁷ on the mesh quality in [15]. Therefore in this paper we use the
⁴⁵⁸ edges' valences and irregular degree to evaluate the quality of
⁴⁵⁹ the quad set.

⁴⁶⁰ When creating new sheets, different quad sets have different
⁴⁶¹ impact on the quality of the mesh. In Fig. 21(a), before sheet

⁴⁶² inflation, $ID(e_1) = 0$; after sheet inflation, e_1 is splitted into
⁴⁶³ two edges e_2 and e_3 and $ID(e_2) + ID(e_3) = 0$ as shown in Fig.
⁴⁶⁴ 21(c). However, if Q_2 (Fig. 21(d)) is inflated, although e_4 is a
⁴⁶⁵ regular edge, the two edges splitted from e_4 are irregular edges
⁴⁶⁶ as shown in Fig. 21(f), and $ID(e_5) + ID(e_6) = 2$. The inherent
⁴⁶⁷ reason for the difference of irregular degrees is that Q_1 and Q_2
⁴⁶⁸ separate the adjacent hex sets into different configurations as
⁴⁶⁹ shown in Fig. 21(b) and Fig. 21(e).

⁴⁷⁰ Suppose e is an edge on quad set Q , and Q separates e 's ad-
⁴⁷¹ jacent hexahedra into two subsets H_1 and H_2 , then the variation
⁴⁷² between the irregular degrees of e and two edges splitted from
⁴⁷³ e can be computed by Equation 2. $\|H_1\|$ and $\|H_2\|$ represent the
⁴⁷⁴ numbers of hexahedra in H_1 and H_2 respectively. A negative
⁴⁷⁵ ΔV_e means the mesh quality near e has been improved by the
⁴⁷⁶ inflation, otherwise the mesh quality becomes worse.

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

⁴⁷⁷ If Q is non-self-intersecting, $E = \{e_1, e_2, \dots, e_n\}$ is the set of
⁴⁷⁸ edges on Q where n is the count of the edges, Q 's variation of
⁴⁷⁹ irregular degrees ΔV_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

⁴⁸⁰ To improve the quality of the quad set, it needs to adjust the
⁴⁸¹ structure of the quad set to make it contain as less edges with
⁴⁸² large variation of irregular degrees as possible. It is, however,
⁴⁸³ not a trivial task since any changes applied on one edge will
⁴⁸⁴ inevitably impact its adjacent edges. Theoretically, if we eval-
⁴⁸⁵ uate all the possible quad sets then we can find the optimal quad
⁴⁸⁶ set with least variation of irregular degrees. Nevertheless it is
⁴⁸⁷ almost impossible due to the large searching space. Chen et al.
⁴⁸⁸ proposed an optimization method which locally handles con-
⁴⁸⁹ cave or convex edges one by one[17]. This method avoids the
⁴⁹⁰ difficulty of global optimization by handling the irregular edges
⁴⁹¹ on the quad set in a greedy-based manner. However, it cannot
⁴⁹² guarantee the mesh quality due to the interaction between edges
⁴⁹³ on the quad set, and sometimes it may be even not convergent.
⁴⁹⁴ In this paper, we propose an optimization method based on the
⁴⁹⁵ dual structure of the quad set. Our method not only avoids the
⁴⁹⁶ difficulty in global optimization, but also guarantees the mesh
⁴⁹⁷ quality after sheet inflation.

⁴⁹⁸ On a quad mesh, starting from an edge, we can recursively
⁴⁹⁹ get all the edges that are topologically parallel to this edge.
⁵⁰⁰ These edges and the adjacent quads form a chord, the dual
⁵⁰¹ structure of the quad set[4, 19]. Similar to the quad mesh, the
⁵⁰² quad set used for sheet inflation also consists of chords. All
⁵⁰³ of the mesh edges on the boundary loop pairwise belong to one
⁵⁰⁴ chord. For example, in Fig. 22(b), e_1 and e_2 belong to the chord
⁵⁰⁵ c_1 , and e_3 and e_4 belong to the chord c_2 . The difference between
⁵⁰⁶ a quad mesh and a quad set for sheet inflation is that there is a
⁵⁰⁷ sheet associated with each chord on the quad set. For example,
⁵⁰⁸ in Fig. 22(c), e_3, e_4 and their chords are contained in the sheet s

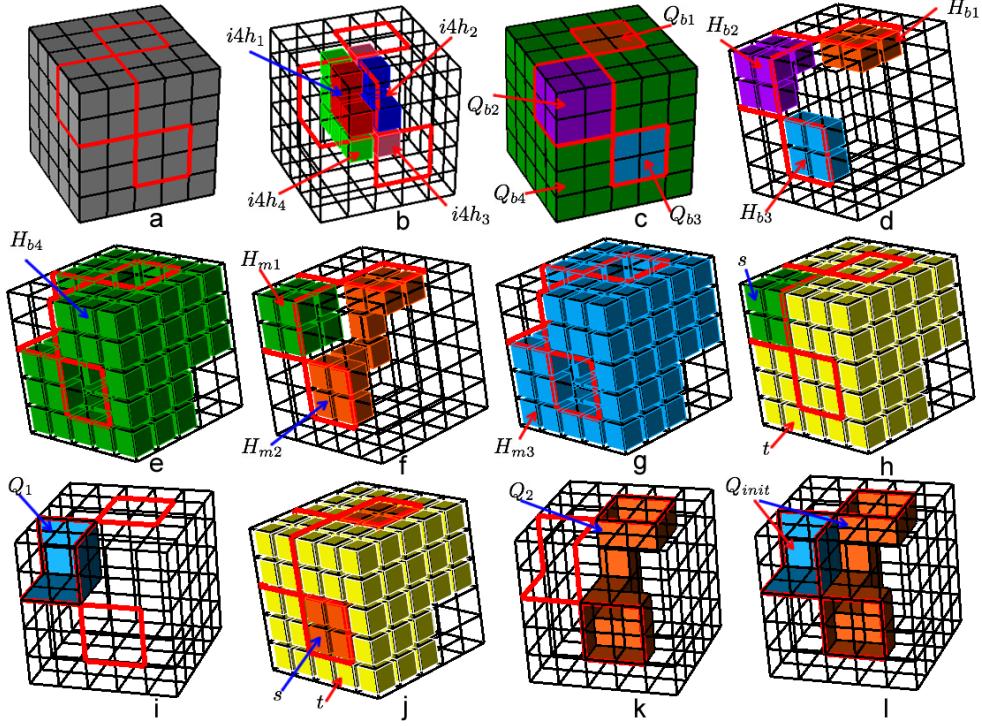


Figure 20: Process of the determination of the initial quad set from a self-intersecting boundary loop: (a) the self-intersecting boundary loop; (b) the int-4-hex-sets $i4h_1$ - $i4h_4$ of the intersecting line; (c) the four quad subsets Q_{b1} - Q_{b4} on the boundary of the local hex set separated 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 $Q_{init} = Q_1 \cup Q_2$.

510 (red). This relationship reveals that if two edges on the bound-
511 ary loop belong to a same chord, then we can find other chords
512 connecting these two edges within the corresponding sheet.

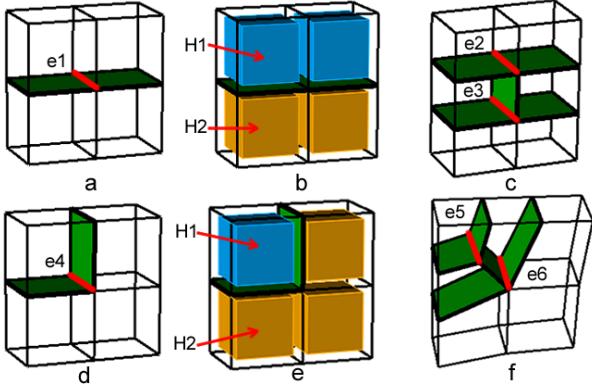


Figure 21: Different impacts on the mesh quality of different quad sets: (a) quad set Q_1 (green) and one of its mesh edges e_1 (red); (b) the hex set adjacent to e_1 is separated into two subsets H_1 and H_2 by Q_1 ; (c) e_1 is splitted into two edges e_2 and e_3 after inflation; (d) quad set Q_2 and 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.

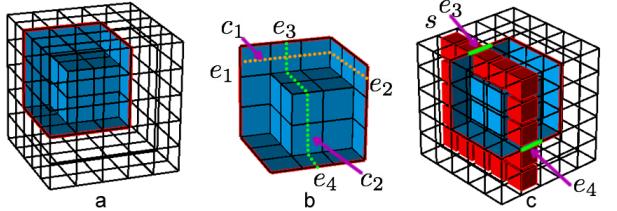


Figure 22: The chords of the quad set: (a) the quad set; (b) two chords c_1 and c_2 on the quad set; (c) the sheet s that contains e_3 and e_4 .

513 We improve the quality of the quad set by iteratively im-
514 proving the quality of the chords that contain the edges on the
515 boundary loop. The chords are neither too constrained as the
516 whole quad set nor too local as one single concave or convex
517 edge. By optimizing the chords, we can effectively improve the
518 quality of the quad set while avoiding the difficulty of optimiz-
519 ing the quad set as a whole. Suppose the quad set is Q and
520 its edge set is E , the details of the optimization procedures are
521 explained:

- 522 1. Divide the edges on the boundary loop into groups accord-
523 ing to whether they are in the same sheet. For example, e_3

and e_4 are grouped in Fig. 23(a), and $e_1 - e_4$ are grouped in Fig. 24(a);

- 524 2. Select a group of edges that have not been processed. Get
525 all of the edges in the corresponding sheet and the quads
526 adjacent to these edges. These edges and quads com-
527 pose the searching space for new chords that connect these
528 edges. Suppose the edge group is $g = \{e_3, e_4\}$, Fig. 23(b)
529 shows the edges that are in the same sheet of g and the
530 adjacent quads;
531 3. Since the edges in the group pairwise connect to one
532 chord, we use A* algorithm to search for the new chords.
533 While searching, we take the edges' variation of irregular
534 degrees into consideration. For example, in Fig. 23(c),
535 there are three quads adjacent to q_1 . If we select q_2 or q_4
536 as the next forward step, the variation of irregular degrees
537 of e_5 will be 2; if we select q_3 as the next forward step,
538 the variation of irregular degrees e_5 becomes 0. Hence we
539 select q_3 . The new chord connecting e_3 and e_4 is shown in
540 Fig. 23(d). If there are more than two edges in the group,
541 it needs deciding which two edges should be paired. For
542 example, four edges $e_1 - e_4$ in Fig. 24(b) are in the same
543 group. If we connect e_1 and e_3 and get the new chord c_3
544 (blue in Fig. 24(c)), then e_2 and e_4 cannot be connected
545 without intersecting with c_3 which is not allowed for keep-
546 ing the quad set valid as shown in Fig. 24(d). So we search
547 new chords between e_1 and e_4 , e_2 and e_3 . The two new
548 chords are shown in Fig. 24(e);
549 4. The new chords and the original chords encompass a hex
550 set on the sheet. For example, the hex set h in Fig. 23(e)
551 is encompassed by the two chords c_2 and c_4 , and the hex set
552 h in Fig. 24(f) is encompassed by the four chords c_1, c_2, c_4
553 and c_5 . We get the boundary quad set Q_h of h , and let Q
554 be the symmetric difference between Q and Q_h , i.e. $Q =$
555 $Q \cup Q_h - Q \cap Q_h$. The updated Q is shown in Fig. 23(f)
556 and Fig. 23(g);
557 5. Repeat the process of Step 2-4 until all of the edge groups
558 are handled. The final quad sets for the two examples are
559 shown in Fig. 23(g) and Fig. 24(g).

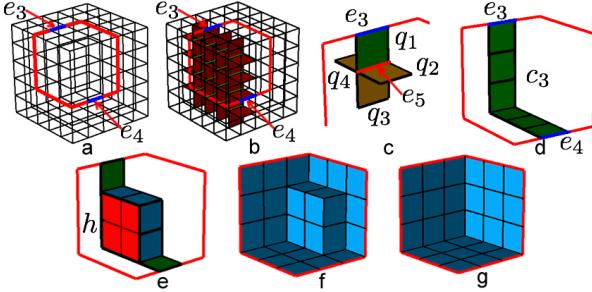


Figure 23: The 1st example of the quad set optimization: (a) two edges e_3 and e_4 on the boundary loop; (b) the edges on the sheet (green) and adjacent quads (red); (c) q_1 's three adjacent quads q_2, q_3 and q_4 ; (d) the new chord connecting e_3 and e_4 ; (e) the hex set h encompassed by the two chords; (f) Q is updated by getting the symmetric difference between Q and Q_h ; (g) the final quad set.

The variation of irregular degrees ΔV_Q of the initial quad set in Fig. 22(a) is 42. After optimization, the variation of irregular

564 degrees of the quad set in Fig. 23(g) $\Delta V'_Q$ becomes 18, which is
565 24 smaller than ΔV_Q .

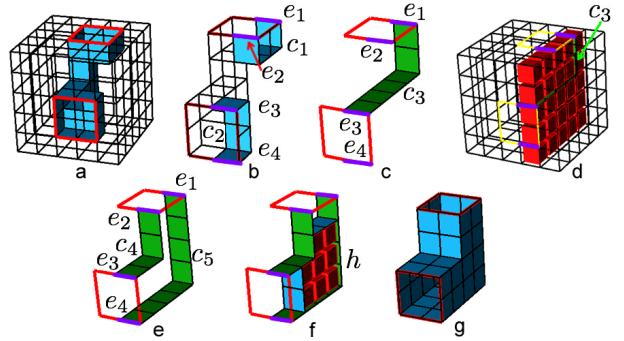


Figure 24: The 2nd example of the quad set optimization: (a) the initial quad set; (b) $e_1 - e_4$ on the same sheet and two chords c_1 and c_2 ; (c) the new chord c_3 connecting e_1 and e_3 ; (d) e_2 and e_4 cannot be connected by a chord without intersecting c_3 ; (e) the two new chords c_4 and c_5 ; (f) the hex set h encompassed by the four chords; (g) Q is updated by getting the symmetric difference between Q and Q_h .

566 The variation of irregular degrees ΔV_Q of the initial quad set
567 in Fig. 24(a) is 76, and it becomes 56 after optimization which
568 is 20 smaller.

569 6. Examples

570 We implement our approach in C++ on a 32-bit Windows
571 7 platform, using Visual Studio 2010. The approach has been
572 tested on different meshes under various constraints. This sec-
573 tion presents two practical applications of the approach: one is
574 complex sheets generation for mesh matching and the other one
575 is the quality improvement for the mesh boundary by reducing
576 high node valences.

577 6.1. Complex Sheets Generation for Mesh Matching

578 Mesh matching is an algorithm to convert non-conforming
579 interfaces to conforming ones by sheet operations[17, 15]. Our
580 approach in this paper can be used in mesh matching to lo-
581 cally generate complex sheets that intersect themselves multiple
582 times, which cannot be done in these previous works. In Fig.
583 25, there is an unmatched chord on the interface of hex mesh 2
584 in Fig. 25(b). A new sheet needs to be created to match it on
585 hex mesh 1 under the constraints shown in Fig. 25a (H is the
586 volumetric constraints).

587 Under the constraints, we construct the boundary loop as
588 shown in Fig. 26. Since the constraint edges contain three in-
589 tersecting nodes, a new intersecting node is created as shown in
590 Fig. 26(b). By running the max-flow-min-cut algorithm, new
591 edges are found, enabling the boundary loop to do local separa-
592 tion. The final boundary loop is shown in Fig. 26(e).

593 Intersecting lines are then constructed, as well as the int-4-
594 hex-sets as shown in Fig. 27(a) and Fig. 27(b). Then we get
595 the merged hex sets in Fig. 27(c) and the initial quad set in
596 Fig. 27(d). The variation of irregular degrees of the initial quad
597 set is 478. By contrast, the final quad set is much better after

598 optimization as shown in Fig. 27(e) whose variation of irregular
599 degrees is reduced to 351.

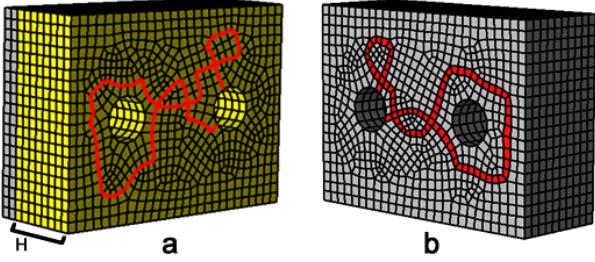


Figure 25: Complex sheet inflation is needed for mesh matching: (a) the constraints for sheet inflation on mesh 1; (b) the unmatched chord(red) on mesh 2.

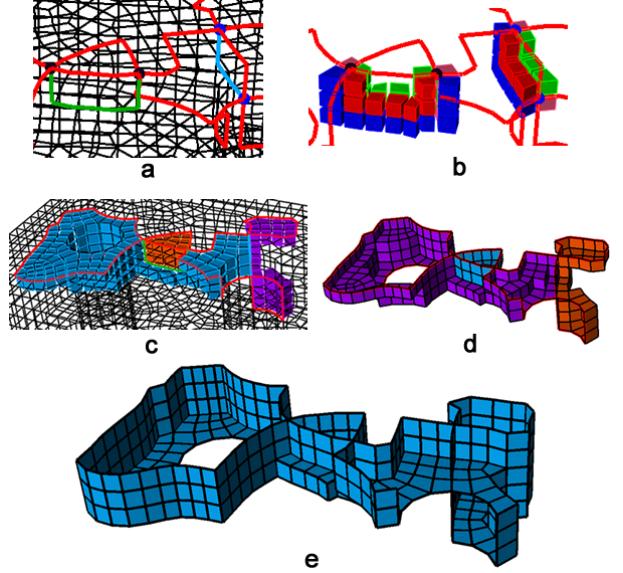


Figure 27: Determination of the quad set: (a) constructing the intersecting lines; (b) the int-4-hex-sets; (c) the merged hex sets; (d) the initial quad set; (e) the optimized quad set.

600 The new sheet is inflated from the quad set as shown in Fig.
601 28.

602 6.2. High Edge Valences Reduction

603 In hex meshes, the elements in the area near the boundary are
604 usually very important for finite element analysis[20]. Mean-
605 while, the node valence and edge valence are critical for the
606 quality of the quad mesh and hex mesh respectively[21, 15, 22].
607 High node valence or edge valence will largely reduce the ac-
608 curacy and efficiency of the analysis, or even make the mesh
609 unable to be handled by the solver. The fun sheet matching
610 algorithm proposed by Kowalski et al. [20] can help the hex
611 mesh capture the geometric boundary by inserting fundamental
612 sheets. This algorithm, however, can neither improve the node
613 valences on the mesh boundary nor the edge valence near the
614 mesh boundary. In this example, we will show how our ap-
615 proach can be used to reduce the high node valences and edge
616 valences near the mesh boundary.

617 Figure 29(a) shows two nodes v_1 and v_2 with high valences.
618 The valences of v_1 and v_2 are 6 and 7 respectively. Conse-
619 quently, the edges inside the hex mesh that are adjacent to these
620 two nodes also have high valences which are 6 and 7 respec-
621 tively as shown in Fig. 29(b). It is usually unacceptable for
622 node or edge valences to be larger than 5 near the boundary.
623 Therefore, these high valences need to be reduced.

624 To reduce the high valences by our approach, the user firstly
625 select several edges on the mesh boundary that are adjacent to
626 the two nodes and the local region as shown in Fig. 30(a).
627 Our approach takes these edges as constraints, and then try to
628 achieve the sheet inflation under the constraints. Figure 30(b)

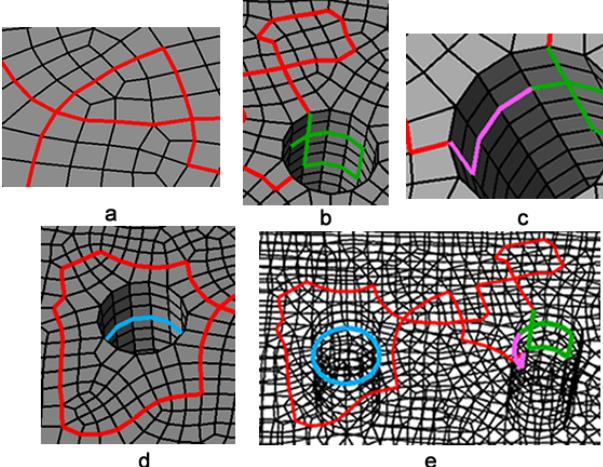


Figure 26: Determination of the boundary loop: (a) the two intersecting nodes are paired; (b) a new intersecting node is created; (c) the boundary loop is closed; (d) new edges are determined by the max-flow-min-cut algorithm; (e) the final boundary loop.

629 and Fig. 30(c) show the process of determination of the bound-
630 ary loop, and the final boundary loop is shown in Fig. 30(d).

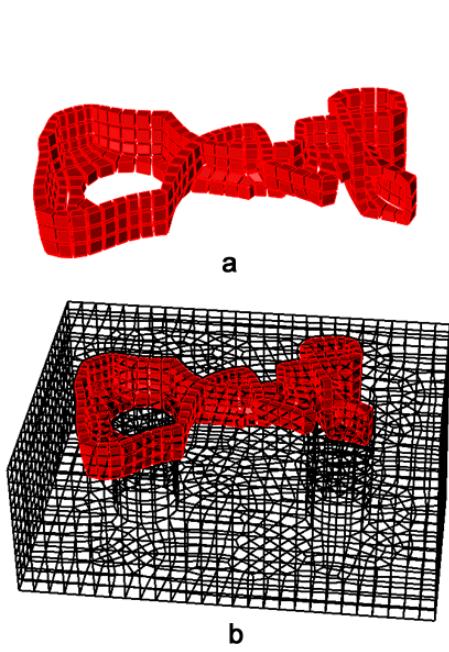


Figure 28: Sheet inflation: (a) the new sheet inflated from the quad set; (b) the new sheet in the hex mesh.

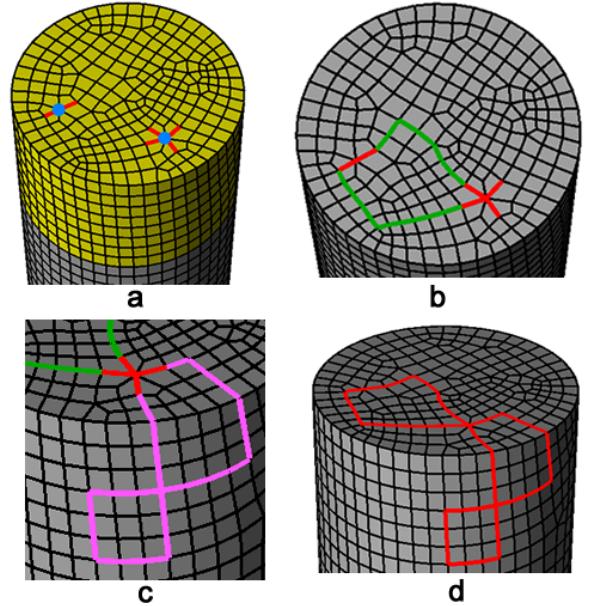


Figure 30: User-specified constraints and the determination of the boundary loop: (a) several boundary edges adjacent to v_1 and v_2 are selected as boundary constraints(red) and a set of hexahedra are selected as local region (yellow); (b) the constraint edges are firstly connected; (c) a new intersecting node is created; (d) the final boundary loop.

631 The quad set is then constructed through the procedures
632 shown in Fig. 31. The variation of irregular degrees of the
633 initial quad set is 184. The final optimized quad set is shown
634 in Fig. 31(e) and its variation of irregular degrees is reduced to
635 159.

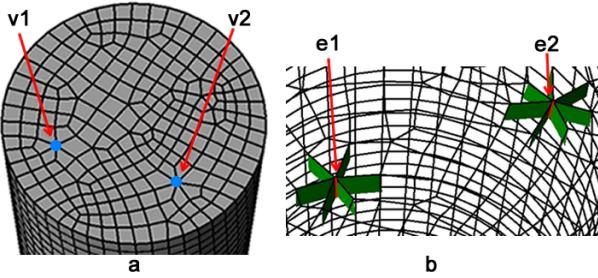


Figure 29: Nodes and edges with high valences: (a) v_1 's valence is 6 and v_2 's valence is 7; (b) v_1 's adjacent edge e_1 has the valence of 6 and v_2 's adjacent edge e_2 has the valence of 7.

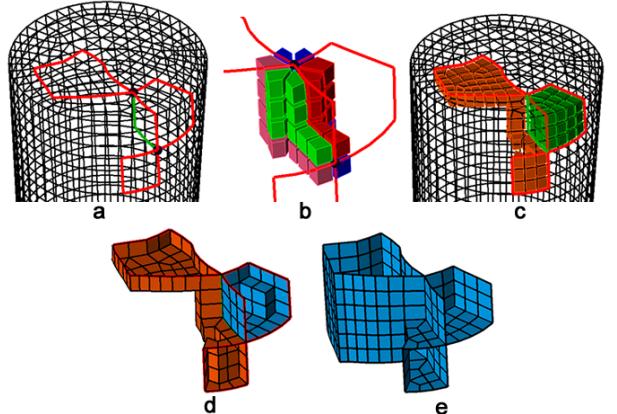


Figure 31: Process of the determination of the quad set: (a) the intersecting line is determined; (b) the int-4-hex-sets are constructed; (c) the merged hex sets; (d) the initial quad set; (e) the optimized quad set.

636 The sheet inflation is conducted after getting the quad set.
637 The new sheet is shown in Fig 32(a) and Fig. 32(b). The two
638 nodes v_1 and v_2 are actually splitted into two nodes and four
639 nodes respectively as shown in Fig. 32(c). The new edges split-

ted from e_1 and e_2 are shown in Fig. 32(b). There has no longer any nodes or edges with valences larger than 5.

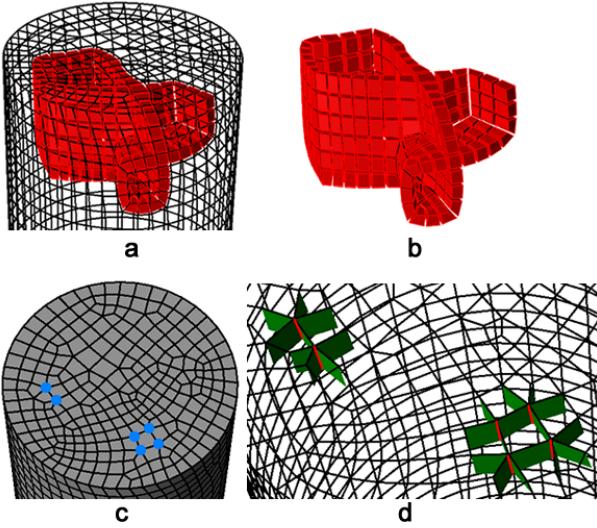


Figure 32: Results of high valences reduction: (a) the new sheet inflated from the quad set in the hex mesh; (b) the new sheet; (c) the new nodes with valences no larger than 5; (d) new edges with valences no larger than 5.

7. Conclusions and Future Work

In this paper, a new approach is proposed to achieving optimized complex sheet inflation under various constraints. Main features of our approach are summarized:

1. By using A* and max-flow-min-cut algorithms, the valid boundary loop of the quad set for sheet inflation is effectively determined, which satisfies the boundary constraints.
2. With intersecting nodes on the boundary loop being reasonably paired by using two templates, intersecting lines are validly constructed between each pair of nodes by using A* algorithm, and the int-4-hex-sets are determined around the intersecting lines, enabling our approach to effectively construct complex quad sets that intersect themselves more than once.
3. By using max-flow-min-cut algorithm, the valid initial quad set is effectively and efficiently determined within the local region specified by the user.
4. A chord-based optimization method is proposed which can effectively improve the quality of the quad set while avoiding the difficulty in optimizing the quad set as a whole.

The examples demonstrate that our approach can successfully generate complex sheets within a local region. These two examples also present two applications that our approach can be used: one is modifying the interfaces in mesh matching and the other is reducing node valences and edge valences to improve mesh quality.

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

1. It is not very convenient to generate self-touching sheets. In some rare cases, self-touching sheets need to be generated. By combining other dual operations like column collapse and sheet extraction, our approach is able to create self-touching sheets. However, currently it is not very convenient. We will propose simpler solutions to generate self-touching sheets;
2. Currently the constraints can only be specified by boundary edges and hexahedra. However, sometimes the user needs to specify edges or quads inside the hex mesh to control the sheet inflation, e.g. improving the edge valences inside the hex mesh. Hence, we plan to adapt our approach to accept constraints specified inside the hex mesh.

References

- [1] Mitchell SA, Tautges TJ. Pillowing doublets: refining a mesh to ensure that faces share at most one edge. 4th International Meshing Roundtable 1995;.
- [2] Shepherd JF. Topologic and geometric constraint-based hexahedral mesh generation. Ph.D. thesis; School of Computing, University of Utah; Salt Lake City, UT, USA; 2007.
- [3] Shepherd JF, Johnson CR. Hexahedral mesh generation constraints. Engineering with Computers 2008;24(3):195–213.
- [4] Murdoch P, Benzley S, Blacker T, Mitchell SA. The spatial twist continuum: A connectivity based method for representing all-hexahedral finite element meshes. Finite Elements in Analysis and Design 1997;28(2):137–49.
- [5] Tautges TJ, Knoop SE. Topology Modification of Hexahedral Meshes Using Atomic Dual-based Operations. IMR 2003;:415–23.
- [6] Ledoux F, Shepherd J. Topological modifications of hexahedral meshes via sheet operations: a theoretical study. Engineering with Computers 2009;26(4):433–47.
- [7] Ramos JS, Ruiz-Gironés E, Navarro XR. Unstructured and semi-structured hexahedral mesh generation methods 2014;.
- [8] Melander DJ, Benzley S, Tautges T. Generation of multi-million element meshes for solid model-based geometries: The dicer algorithm. Tech. Rep.; Sandia National Labs., Albuquerque, NM (United States); 1997.
- [9] Staten ML, Shepherd JF, Ledoux F, Shimada K. Hexahedral Mesh Matching: Converting non-conforming hexahedral-to-hexahedral interfaces into conforming interfaces. International Journal for Numerical Methods in Engineering 2009;:1475–509.
- [10] Ledoux F, Shepherd J. Topological and geometrical properties of hexahedral meshes. Engineering with Computers 2009;26(4):419–32.
- [11] Staten ML. Sheet-based generation and modification of unstructured conforming all-hexahedral finite element meshes. Ph.D. thesis; CARNEGIE MELLON UNIVERSITY; 2010.
- [12] Anderson BD, Benzley SE, Owen SJ. Automatic all quadrilateral mesh adaption through refinement and coarsening. In: Proceedings of the 18th International Meshing Roundtable. Springer; 2009, p. 557–74.
- [13] Canann SA, Muthukrishnan S, Phillips R. Topological improvement procedures for quadrilateral finite element meshes. Engineering with Computers 1998;14(2):168–77.
- [14] Miller STI, Benzley SE, Owen SJ, Staten ML. Using edge valence prediction to drive localized all-hex coarsening. 20th International Meshing Roundtable ????:23–6.
- [15] Staten ML, Shimada K. A close look at valences in hexahedral element meshes. International Journal for Numerical Methods in Engineering 2010;899–914doi:10.1002/nme.2876.
- [16] Suzuki T, Takahashi S, Shepherd J. An interior surface generation method for all-hexahedral meshing. Engineering with Computers 2010;26(3):303–16.
- [17] Chen J, Gao S, Zhu H. An improved hexahedral mesh matching algorithm. Engineering with Computers 2015;doi:10.1007/s00366-015-0414-1.
- [18] Lawler E. 4.5. combinatorial implications of max-flow min-cut theorem, 4.6. linear programming interpretation of max-flow min-cut theo-

- 736 rem. Combinatorial Optimization: Networks and Matroids 2001;:117–
737 20.
738 [19] Mitchell S. A characterization of the quadrilateral meshes of a sur-
739 face which admit a compatible hexahedral mesh of the enclosed volume.
740 STACS 96 1996.;
741 [20] Kowalski N, Ledoux F, Staten ML, Owen SJ. Fun sheet matching: to-
742 wards automatic block decomposition for hexahedral meshes. Engineering
743 with Computers 2011;doi:10.1007/s00366-010-0207-5.
744 [21] Owen S. A survey of unstructured mesh generation technology. 7th In-
745 ternational Meshing Roundtable 1998;
746 [22] Tarini M, Pietroni N, Cignoni P, Panozzo D, Puppo E. Practical quad
747 mesh simplification. Computer Graphics Forum 2010;29(2):407–18.
748 doi:10.1111/j.1467-8659.2009.01610.x.