An approach to achieving optimized sheet inflation under constraints

Jinming Chen[a], Shuming Gao[a], Rui Wang[a], Haiyan Wu[a]

[a] *CAD&CG State Key Laboratory, Zhejiang University, Hangzhou, China*

Abstract

As a type of dual operation, sheet inflation is irreplaceable for hex mesh generation, modification and topological optimization. However, current algorithms cannot effectively construct sheet inflation under user-specified constraints with guaranteed mesh quality, especially when self-intersecting sheets need to be generated. In this paper we propose an approach to achieving optimized sheet inflation under constraints. As it is very difficult to directly construct the qualified quad set which is used to inflate the sheet, we firstly construct the boundary loops under the boundary constraints, then determine and optimize the interior quad sets under the volumetric constraints, and at last inflate the new sheets. We successfully apply this approach in 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[1, 2].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[3, 4, 5, 6].

In recent years, as a kind of tools that can effectively deal with these global structures, sheet operations attract more and more attention and have been applied in different situations. 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.

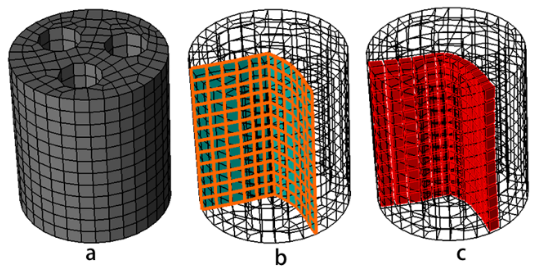

Figure 1: Sheet inflation procedures: (a) the original hex mesh; (b) the quad set; (c) the new sheet inflated from the quad set.

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 while applying in order to meet various mesh modification demand. The controllability usually includes the ability to control the positions where the sheet intersects with the mesh boundary and the region in the mesh volume where the sheet can propagate. In practice, one convenient and natural way for the users to gain the controllability is to setup some constraints on the mesh to guide the inflation process. The constraints are usually specified by a set of boundary edges and a set of hex. The boundary edges determine the positions where the new sheet should appear on the mesh boundary, and the set of hex stands for the region where the new sheet can be generated. As shown in Fig. 2b, the user specifies a set of boundary edges(green) and a set of hex(yellow) to control the shape and position of the new sheet. One satisfactory sheet is shown in Fig. 2d.

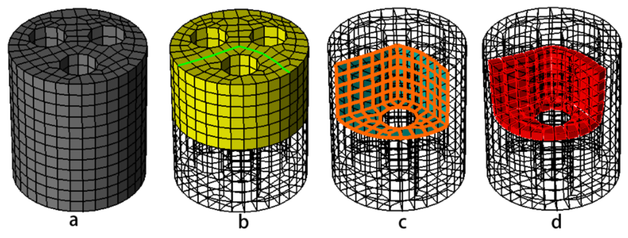

Figure 2: Sheet inflation under user-specified constraints: (a) the original hex mesh; (b) the user-specified boundary edges (green) and hex set (yellow); (c) the internal quad set; (d) the new sheet.

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 construct 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. 2b, 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. 2c.

Recognizing the importance of sheet inflation, many researchers have been investigating algorithms for decades and have achieved great progress. Mitchell et al. proposed the Pillowing algorithm to deal with the doublet problem[7]. This algorithm involves inflating a new sheet guided by a hex set called shrinking set. The simplicity and effectiveness makes Pillowing prevailing in mesh modification, especially in mesh quality improvement. Although it can be adapted to satisfy simple constraints, it lacks the ability to generate complex sheets such as self-intersecting sheets.

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

Staten et al. proposed the General Sheet Inflation method which explained in detail how to do sheet inflation on hex meshes when boundary mesh edges are specified[9]. This method can create normal sheets, self-touching sheets and self-intersecting sheets. However, determining the quad sets for sheet inflation in this method seems to rely on the sweeping structures of mesh, hence the method may fail when the meshes are not sweeping structures. Meanwhile it is not clear whether the method can control the region when creating self-intersecting sheets.

Chen et al. proposed a new approach to inflate sheets when conducting mesh matching on complex interfaces[10]. 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 requires the sheets intersecting themselves once at most. Meanwhile, the optimization method it applies on quad set cannot guarantee the quality improvement.

Despite these achievements, there are still some problems to inflate qualified sheets to meet complex constraints with guaranteed mesh quality, especially when self-intersecting sheets needs to be generated within a local region. In practice, as the magnitude of hex meshes grows and the complexity of modification requirements raises, the need to generate complex sheets under constraints with guaranteed mesh quality becomes not rare. In order to handle this problem, in this paper we propose a new approach to achieving optimized sheet inflation under user-specified constraints. To determine the qualified quad set, our approach tries to firstly construct the boundary loops, and then determine and optimize the interior quad set. At last we create the new sheets by inflating the quad set.

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 determining the boundary loops of quad sets and the interior quad sets using the boundary loops in the third and the fourth chapters respectively, and then give some results in the fifth chapter and list conclusions and future work in the last chapter.

2. Overview of the Approach

In this paper, we propose an approach to constructing optimized sheet inflation under constraints specified by the users. In this section we provide the overview of our approach including the main idea, the input and output and the major steps.

When constraints are specified on the hex mesh, it is very difficult to directly determine a qualified quad set that satisfy all the constraints while guaranteeing the mesh quality after sheet inflation. To handle this problem, our main idea is to firstly construct an initial quad set that satisfies the constraints but may suffer from poor quality, and then use optimization algorithm to improve its quality.

Nevertheless, constructing the initial quad set is also not a trivial task due to the large searching space in the hex mesh which may contain tens of thousands of quads. To do this, we firstly determine the quad set's boundary loop which is the set of edges where the quad set intersects with the mesh boundary, and then determine the interior quad set based on the boundary loop. As seen in Fig. 3, the topology of the boundary loop and its corresponding quad are tightly associated, i.e. if the quad set is non-self-intersecting (Fig. 3a), its boundary loop is also non-self-intersecting (Fig. 3b), and vice versa (Fig. 3e). Compared with directly determining the quad set, constructing the boundary loop is easier since the mesh boundary is a 2D manifold with much smaller searching space.

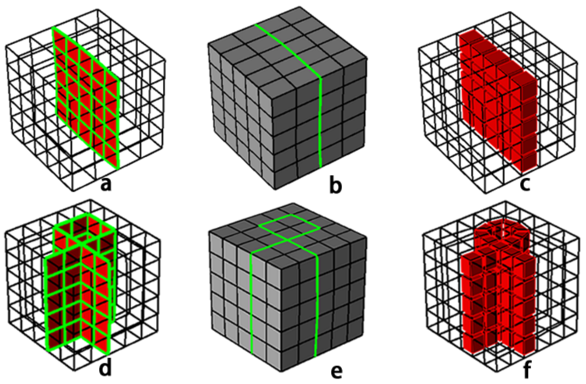

Figure 3: Two quad sets and their boundary loops: (a)the 1st quad set; (b)the boundary loop of the 1st quad set; (c)the sheet inflated from 1st quad set; (d)the 2nd quad set; (e)the boundary loop of the 2nd quad set; (f)the sheet inflated from 2nd quad set.

As discussed in the previous section, the boundary constraints and volumetric constraints are specified by a set of boundary mesh edges and a set of hex respectively. Therefore, our approach takes a set of boundary mesh edges and a set of hex as input which are either from users direct input or derived from the user's specification. The output of our approach is the resultant hex mesh after sheet inflation.

Our approach can generate non-self-intersecting and self-intersecting sheets, and allow the self-intersecting sheets intersect itself more than once. The main procedures of our approach are illustrated in Fig. 4, and key steps are explained in detail in the following sections.

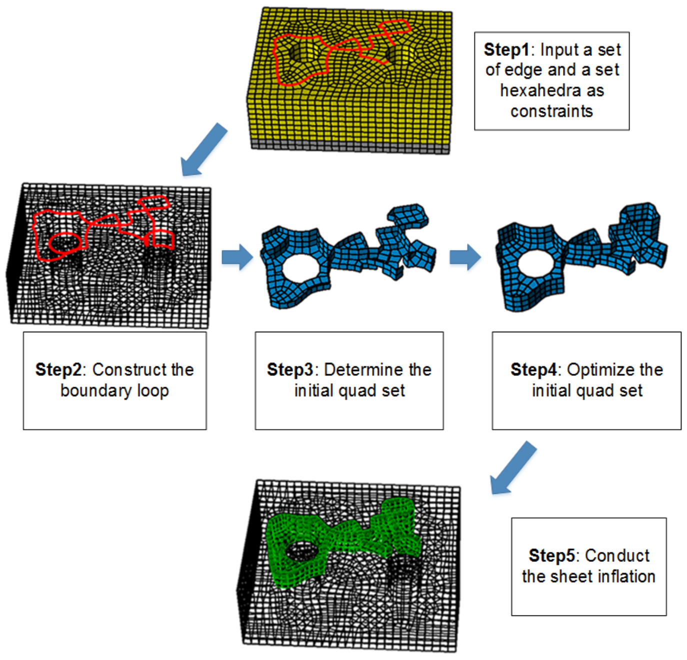
Step1: Input a set of edge and a set hexahedra as constraints
Step2: Construct the boundary loop
Step3: Determine the initial quad set
Step4: Optimize the initial quad set
Step5: Conduct the sheet inflation

Figure 4: Overview of the approach.

3. Determination of Boundary Loops

As previously discussed, it is quite difficult to directly find a qualified quad set satisfying all the constraints. Hence we firstly determine the boundary loop, and then construct the initial quad set based on the boundary loop. In this section, firstly we will discuss the characteristics of a valid boundary loop. And secondly we try to construct a valid boundary loop containing the specified constraint edges by the path searching algorithm and max-flow-min-cut algorithm.

According to 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 needs to be able to separate 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. 5b separates the local hex set into two parts as shown in Fig. 5c. The self-intersecting quad set in Fig. 5g separates the local hex set into 3 subsets as shown in Fig. 5h. Accordingly, the boundary loop of the quad set should be able to separate the boundary of the local hex set into 2 + n subsets, where n is the number of self-intersecting points on the boundary loop. For example, the non-self-intersecting boundary loop in Fig. 5a separates the boundary of the local hex set into two subsets as shown in Fig. 5d, and the self-intersecting boundary loop in Fig. 5f separates the boundary of the local hex set into 4 subsets as shown in Fig. 5i.
2. The number of intersecting points on the boundary loop should be even because each intersecting line on the quad set has two intersecting points on the boundary loop.

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 hex, as shown in Fig. 5c, Fig. 5e, Fig. 5h and Fig. 5j. 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 point on the boundary loop needs another intersecting points to be matched 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 determination of boundary loop is shown in Fig. 6. The rest of this section will provide detailed description about the method of determination of boundary loop.

3.1. Intersecting Points Pairing

For self-intersecting quad sets, the intersecting lines play a very important role for defining the structures of the quad sets. Each intersecting line's two end points are two intersecting points on the boundary loop. Therefore the intersecting points need to be paired so that we can use the path searching algorithm to construct the intersecting line between each pair of intersecting points. When there are more than two intersecting pointsit needs decision about which two intersecting points are suitable to be paired. In this section we provide two local templates to find suitable intersecting points pairs.

Suzuki et al. used the node types which are calculated from the winding numbers to decide whether two intersecting points can be paired[8]. They also provided examples showing that inappropriate pairing of intersecting points will result in non-orientable surface with very poor quality or even no surface existing at all. However they did not provide clear instructions on how to find suitable intersecting points pairs, and the winding numbers may not be calculated because the boundary constraint edges specified by the user may not be closed loops.

In this paper, based on the observation of the structures of quadrilateral sets, we bring out two local templates for pairing intersecting points as shown in Fig. 7. Intersecting points on complex self-intersecting boundary loops can be paired by recursively using these two local templates until all of them are paired with each other. Fig. 7a and Fig. 7f show two local self-intersecting structures of quad sets. The hexahedra adjacent to the intersecting lines are separated in to four subsets by the local structures of quad sets, as shown in Fig. 7b and Fig. 7g. These hex subsets are called int-4-hex-sets of the intersecting line. The associated local structures of boundary loops are

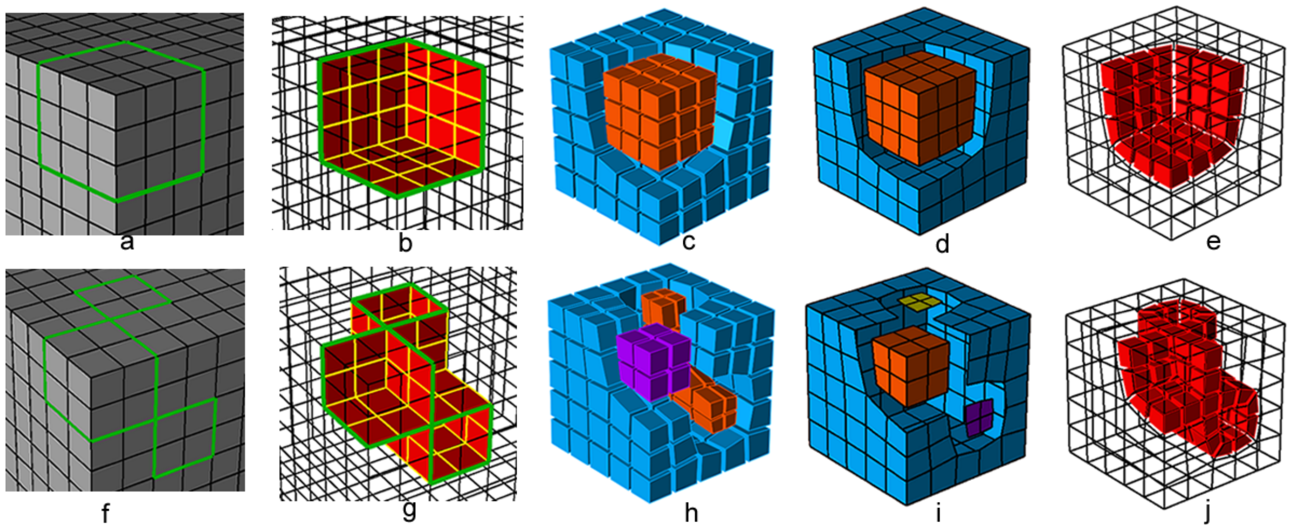

Figure 5: Characteristics of valid boundary loops: (a)a non-self-intersecting boundary loop; (b)the non-self-intersecting quad set; (c)two hex sets separated by the quad set; (d)two boundary quad sets separated by the boundary loop; (e)the new non-self-intersecting sheet; (f)a self-intersecting boundary loop; (g)the self-intersecting quad set; (h)three hex sets separated by the quad set; (i)four boundary quad sets separate by the boundary loop; (j)the new self-intersecting sheet.

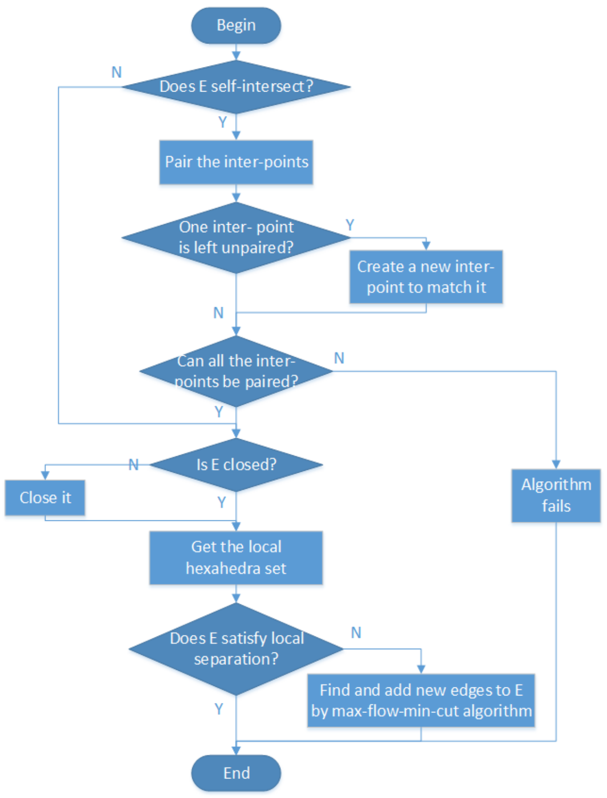
Begin
Does E self-intersect?
N
Y
Pair the inter-points
One inter- point is left unpaired?
Y
Create a new inter-point to match it
N
Can all the inter-points be paired?
N
Y
Is E closed?
N
Close it
Y
Algorithm fails
Get the local hexahedra set
Does E satisfy local separation?
N
Find and add new edges to E by max-flow-min-cut algorithm
Y
End

Figure 6: Overview of the approach.

shown in Fig. 7c and Fig. 7h. The quads in same color mean they are adjacent to the same int-4-hex-set. The two local templates for pairing intersecting points are shown in Fig. 7d and Fig. 7i respectively, where the numbers in different colors indicate the adjacent int-4-hex-sets. After two intersecting points are paired, in order to recursively find next pair of intersecting points, we remove the intersecting points the way shown in Fig. 7e and Fig. 7j.

Next, we use examples to show how to use these two templates to pair intersecting points. The first example is shown in Fig. 8.

The pairing process for the 1st example is shown in Fig. 9. Firstly the intersecting point A and B can be paired using the 1st template. After removing the two intersecting points, we find point C and D can also be paired using the 1st template.

The pairing process for the 2nd example is shown in Fig. 10. Firstly the intersecting point A and B can be paired using the 2nd template. After removing the two intersecting points, point C and D can be paired using the 1st template.

If there are two or more intersecting points on the boundary loop that cannot be paired by the two templates, our algorithm would fail. According to the description in [8], these boundary loops usually can only determine non-orientable quad sets which will bring poor quality to the mesh after sheet inflation. Hence in this paper we do not handle this kind of boundary loops.

After recursively applying these two templates, if there is still one intersecting point left unpaired, we use the templates to create a new intersecting point at a appropriate position to be paired with this intersecting point. For example in Fig. 11a, the intersecting point A is unpaired. We use the 2nd template to create a new intersecting point B to be paired with A as shown in Fig. 11b.

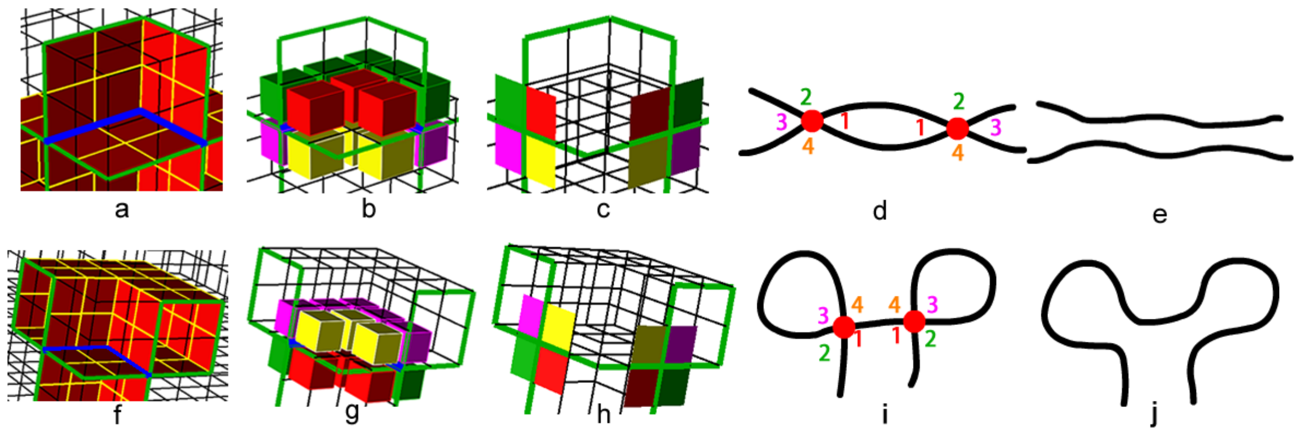

Figure 7: Two local templates for pairing intersecting points: (a)the 1st local intersecting structure of quad set; (b)int-4-hex-sets around the 1st local intersecting structure of quad set; (c)the associated local structure of boundary loop; (d)the 1st template; (e)intersecting points are resolved after pairing; (f)the 2nd local intersecting structure of quad set; (g)int-4-hex-sets around the 2nd local intersecting structure of quad set; (h)the associated local structure of boundary loop; (i)the 2nd template; (j)intersecting points are resolved after pairing.

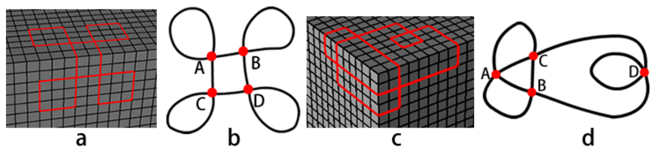

Figure 8: Two examples of pairing intersecting points: (a)the boundary loop of the 1st example; (b)the topologic structure of the boundary loop of the 1st example; (c)the boundary loop of the 2nd example; (d)the topologic structure of the boundary loop of the 2nd example.

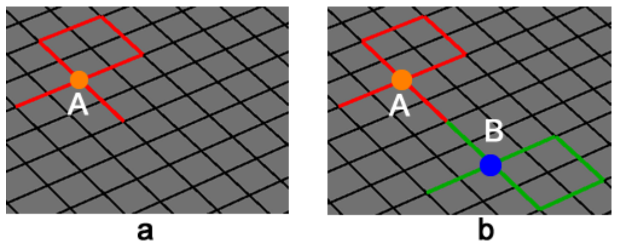

Figure 11: Handling of the situation when single intersecting point is unpaired: (a)intersecting point A is unpaired; (b)intersecting point B is created to be paired with A.

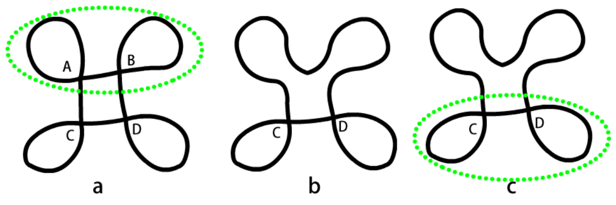

Figure 9: The pairing process of the 1st example: (a)point A and B are paired; (b) point A and B are removed; (c)point C and D are paired.

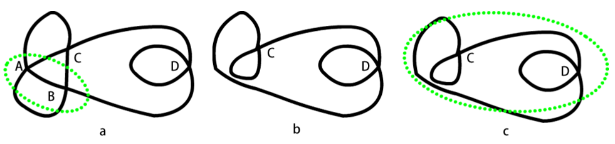

Figure 10: The pairing process of the 2nd example: (a)point A and B are paired; (b) point A and B are removed; (c)point C and D are paired.

3.2. Closing the Boundary Loop

As discussed previously, the boundary loops should be closed. However, the constraint edges specified by the user may not form closed loops. Therefore, 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 numbers 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. Fig. 12b shows the result when we do not consider the angles between adjacent edges, and Fig. 12c shows a better result when the angles are taken into consideration.

3.3. Making the Boundary Loop Able to Do Local Separation

After being handled by previous sections, the constraint edges specified by the user become a closed boundary loop with intersecting points being paired. According to the discussion in the beginning of Section 3, a valid boundary loop should be able to separate the boundary of a local hex set into $2 + n$ subsets, where n is the number of intersecting points, a characteristic called local separation. Sometimes due to the existence

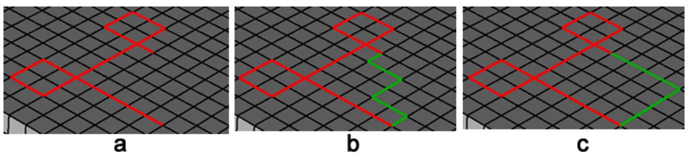

Figure 12: Closing the boundary loop: (a)the unclosed boundary loop; (b)new edges determined by A* algorithm without considering the turning angles; (c)new edges determined by A* algorithm considering the turning angles.

of through holes on the hexahedral mesh, the closed boundary loop may fail to satisfy local separation. In this section, we explain how to make the boundary loop satisfy local separation. Our main idea is: we try to iteratively determine a local hex set whose boundary can be separated by the boundary loop into $2 + n$ subsets; if no such hex set is found within maximum iteration times, we try to determine and add necessary edges by max-flow-min-cut algorithm 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. 13.

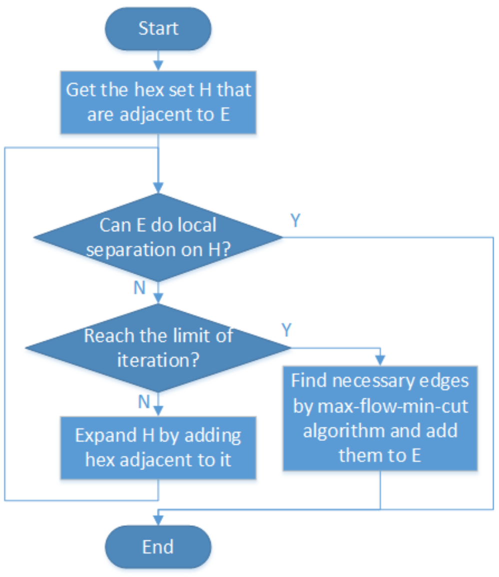
Start
Get the hex set H that are adjacent to E
Can E do local separation on H?
Y
N
Reach the limit of iteration?
Y
N
Expand H by adding hex adjacent to it
Find necessary edges by max-flow-min-cut algorithm and add them to E
End

Figure 13: Flowchart of making the boundary loop able to do local separation.

We use an example to explain the process. Figure 14a 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. 14b) 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. 14c). However, E is still unable to do local separation on H since there are still only one quad set (blue in Fig. 14d) 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 E able to do local separation.

Max-flow-min-cut algorithm is an effective and efficient tool to find the minimal cut set in a directed graph. As shown in Fig. 15a, to use the max-flow-min-cut algorithm, a directed

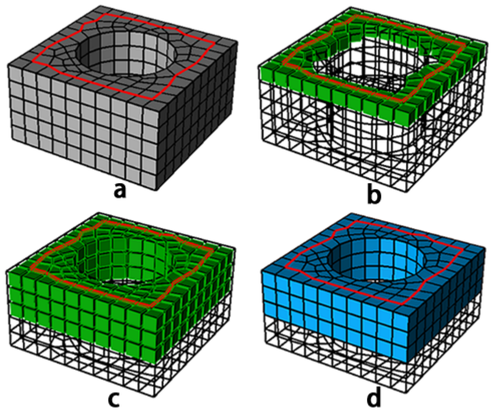

Figure 14: An example of boundary loop that is unable to do local separation: (a)the boundary loop E; (b)E's adjacent hexahedra H(green); (c)expand H by iteratively adding its adjacent hexahedra; (d)E still cannot separate H's boundary into two subsets when reaching maximum expanding iteration times.

graph should be constructed with an s node and a t node, and the weights of the edges should also be set before calculation. The algorithm will find a minimal cut set that determines the max flow from s to t.

If we take the boundary of H as a graph where the nodes stand for a quad or a quad set and the edges stand for the adjacency relationship between the quads or quad sets, then to be able to do local separation is similar to find a cut set on this graph. Hence, the max-flow-min-cut algorithm can be used to help us to find necessary edges to make the boundary loop satisfy the local separation.

To use the algorithm, we need to construct the directed graph at first, including constructing the s and t nodes and deciding the edges' weights. As the current boundary loop E must be part of the final boundary loop, the two quad sets on its two sides just can be the s and t nodes as shown in Fig. 15b. Since all the edges on the final boundary loop should be on the boundary of the hexahedral mesh, so the quads shared by H and the rest of the hexahedral mesh, which is shown as Q_{in} in Fig. 15b, should be grouped into a single node in the graph to guarantee that Q_{in} won't be separated by the min cut. Similarly, in order to forbid the mesh edges on sharp CAD model edges to be added in to the boundary loop, which usually results in poor mesh quality, we group the quads sharing these mesh edges into single nodes, e.g. q_2 and q_3 in Fig. 15c.

After constructing the nodes, we assign weights to the graph edges according to how many mesh edges shared by the two nodes. For example, q_1 in Fig. 15c shares two edges with node t, so the weight of the directed edges in the graph between q_1 and t is 2. All the edges in the graph are bi-directional except the edges adjacent to s and t.

The final directed graph is shown in Fig. 15d. After calculation, new edges are found as shown in Fig. 15e, and now E is able to do local separation after adding these new edges as shown in Fig. 15f.

When a self-intersecting boundary loop is unable to do local separation, we also need to apply max-flow-min-cut algorithm

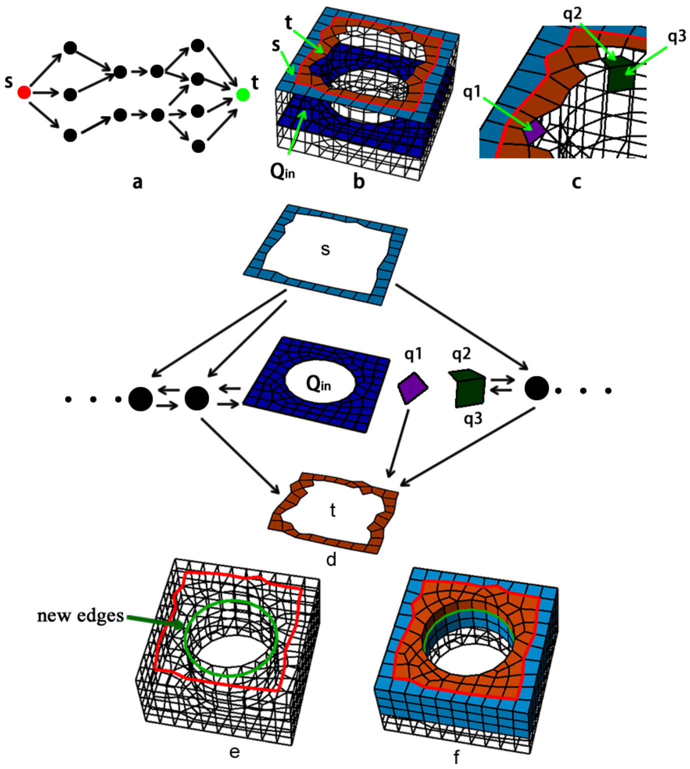

Figure 15: Using max-flow-min-cut algorithm to find necessary edges for local separation: (a)directed graph used for ordinary max-flow-min-cut problem; (b)the quad sets on E's two sides can be s and t nodes; (c)q_1 shares two edges with t, q_2 and q_3 shares an edge on the geometric hard edge; (d)the 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.

to find necessary new edges to make it able to do local separation. The procedures are quite similar to non-self-intersecting boundary loop discussed previously. We treat a self-intersecting boundary loop as several non-self-intersecting boundary loops and handle each of them individually. The self-intersecting boundary loop in Fig. 16a consists of three sub loops $loop_1$, $loop_2$ and $loop_3$. The local hex set is shown in Fig. 16b. Due to existence of three through holes on the hex mesh, the boundary loop is unable to do local separation. After applying max-flow-min-cut algorithm to $loop_1$ and $loop_3$, new edges are found as shown in Fig. 16c.

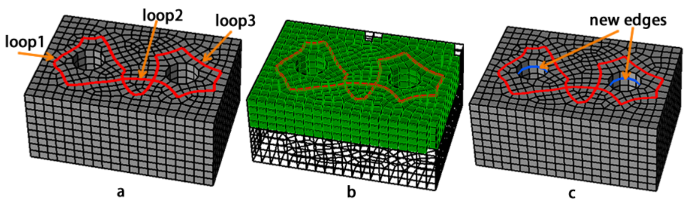

Figure 16: Making self-intersecting boundary loop able to do local separation: (a)the self-intersecting boundary loop with three sub loops; (b)the local hex set; (c)new edges (blue) are found by using max-flow-min-cut algorithm.

4. Determination of Interior Quad Set

After determining the boundary loop, we now use the boundary loop to construct the interior quad set. However, it is still difficult to directly construct a quad set from the boundary loop with guaranteed mesh quality. Our main idea for determination of the interior quad set is: based on the boundary loop, we firstly construct an initial inflatable quad set that satisfies all the constraints, and then we apply optimization techniques to improve its quality. In this section, we provide details of determination of interior quad set including determination of intersecting lines, constructing initial quad set by max-flow-min-cut algorithm and the optimization method for the quad set.

4.1. Determination of Intersecting Lines

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 Section 3.1 the two templates not only make two intersecting points be paired but also setup the correspondence relationship between the int-4-quad-sets of the two intersecting points. In this section, we use this information to construct the intersecting lines and the int-4-hex-sets of the intersecting line.

We use an example to explain the process of constructing the intersecting line and its int-4-hex-sets. In Fig. 17a, the blue nodes are two intersecting points paired by the 1st template (in this example they can also be paired by the 2nd template), and the numbers and different colors denote the correspondence between the int-4-quad-sets of the two points. We use A* algorithm to determine the intersecting line between the two intersecting points. Similar to Section 3.2, in order to get a smooth path, we take the turning angles between adjacent edges in the path into consideration while doing the A* searching. The intersecting line is shown in Fig. 17b. Next, we get all the hexahedra adjacent to the intersecting line, and use the correspondence between the int-4-quad-sets to construct the int-4-hex-sets as shown in Fig. 17c.

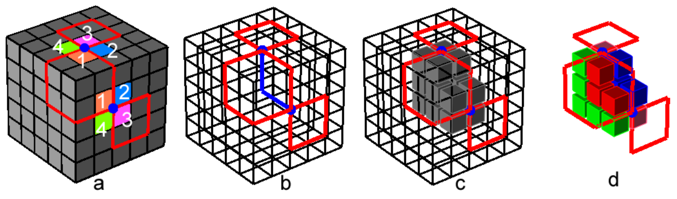

Figure 17: Process of constructing the intersecting line and its int-4-hex-sets: (a)two intersecting points paired by the 1st template; (b)the intersecting line determined by A* algorithm; (c)the hexahedra adjacent to the intersecting line; (d)the int-4-hex-sets of the intersecting line.

4.2. Determination of Initial Quad Set by Max-flow-min-cut Algorithm

After constructing the intersecting lines and their int-4-hex-sets, we continue to construct the initial quad set. The idea is similar to Section 3.3, we convert the problem of being able to do local separation to the problem of acquiring min cut on a graph. If we see the hexahedra as nodes and the adjacency between the hexahedra as edges in the graph, the quad set is actually a cut set in the graph. Hence, max-flow-min-cut algorithm can be used to efficiently get a quad set that is able to do local separation. In this section we provide details about this process.

As the boundary loop separates the boundary of the local hex set into $2+n$ subsets (n is the number of intersecting points), we can get $2+n$ hex sets that are adjacent to these $2+n$ quad sets. From these $2+n$ hex sets and the int-4-hex-sets of the intersecting lines, we merge hex sets that share common hexahedra. This will result in $2+N$ hex sets (N is the number of intersecting lines). Then we apply the max-flow-min-cut algorithm multiple times to get the initial quad set. The pseudo-codes are provided in Algorithm 1.

We provide two examples to explain the procedures of Algorithm 1. The first example is a non-self-intersecting boundary loop as shown in Fig. 18. n and N are both 0 for this example. The two quad subsets on the boundary of the local hex set H are Q_{b1} and Q_{b2} as shown in Fig. 18c. We get the two hex sets adjacent to Q_{b1} and Q_{b2} as H_{b1} and H_{b2} (Fig. 18d). We construct the directed graph by taking H_{b1} as s node and H_{b2} as t node, taking other hexahedra as normal nodes, and assigning weights to the edges according the numbers of quads shared by two hex sets. After running the max-flow-min-cut algorithm, we get the initial quad set Q_{init} as shown in Fig. 18.

The second example is a self-intersecting boundary loop as shown in Fig. 19. The boundary loop (red in Fig. 19a) intersects itself once, so $n = 2, N = 1$. Firstly we get the int-4-hex-sets $i4h_1$ $i4h_4$(Fig. 19b). The boundary of local hex set H are separated into Q_{b1} Q_{b4}(Fig. 19c), and the hex sets adjacent to these quad subsets are H_{b1} H_{b4} (Fig. 19d, and Fig. 19e). So

Algorithm 1 Determination of Initial Quad Set

```
Input: The boundary loop $L$ and the localhex set $H_{local}$;
Output: The initial quad set $Q_{init}$;
 1: $n$ = the count of self-intersecting points on $L$;
 2: $N$ = the count of intersecting lines;
 3: Get the quad subsets on the boundary of $H_{local}$ as $Q_{bound} = \{Q_{b1}, Q_{b2}, \cdots, Q_{b2+n}\}$;
 4: $H_{bound} = \{H_{b1}, H_{b2}, \cdots, H_{b2+n}\}$ where $H_i$ is the hex set adjacent to $Q_i$,$i = 1, 2, \cdots, 2+n$;
 5: $H_{int} = \{H^1_{in1}, H^2_{in1}, H^3_{in1}, H^4_{in1}, \cdots, H^1_{inN}, H^2_{inN}, H^3_{inN}, H^4_{inN}\}$ where $\{H^1_{ini}, H^2_{ini}, H^3_{ini}, H^4_{ini}\}$ 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
```

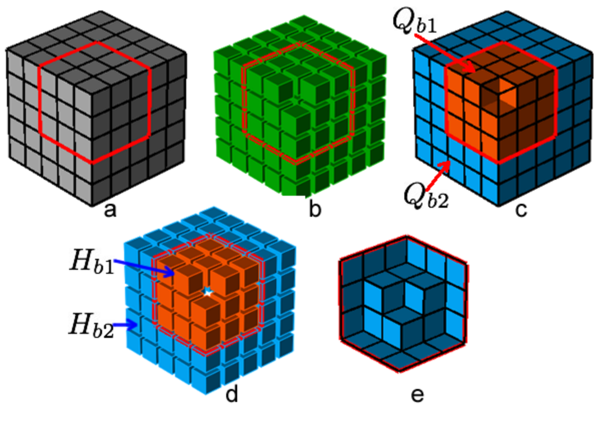

Figure 18: Process of determination of initial quad set from a non-intersecting boundary loop: (a)the non-intersecting boundary loop; (b)the local hex set; (c)the two boundary quad subsets Q_{b1}(orange) and Q_{b2}(blue); (d)two hex sets H_{b1}(orange) and H_{b2}(blue); (e)the initial quad set Q_{init}.

$H_{merge} = H_{int} \cup H_{bound} = \{i4h_1, \cdots, i4h_4, H_{b1}, \cdots, H_{b4}\}$. After merging, $H_{merge} = \{H_{m1}, H_{m2}, H_{m3}\}$ (Fig. 19f and Fig. 19g). Then we conduct the max-flow-min-cut algorithm twice. For the first time, we take H_{m1} as s node and $H_{m2} \cup H_{m3}$ as t node (Fig. 19h), and get the quad set Q_1 (Fig. 19i). For the second time, we take H_{m2} as s node and $H_{m1} \cup H_{m3}$ as t node (Fig. 19j), and get the quad set Q_2 (Fig. 19k). Finally the initial quad set is $Q_{init} = Q_1 \cup Q_2$ (Fig. 19l).

4.3. Optimization of the Initial Quad Set

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

4.3.1. Quality Evaluation of Quad Set

In a hex mesh, the valence of a mesh edge stands for the count of its adjacent quads. A mesh edge is called a regular edge if it is inside the mesh and its valence equals 4 or it is on the mesh boundary and its valence equals 3. Otherwise the edge is called an irregular edge. The difference between an edge's valence and the corresponding regular edge' valence is called irregular degree. Suppose the edge is e and $v(e)$ stands for e's valence, e's irregular degree is computed as Equation. 1.

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

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

When inflating new sheets, different quad sets have different impact on the quality of the mesh. In Fig. 20a, before sheet inflation, $ID(e_1) = 0$; after sheet inflation, e_1 is splitted into two edges e_2 and e_3 and $ID(e_2) + ID(e_3) = 0$ as shown in Fig. 20c. However, if Q_2 (Fig. 20d) is inflated, although e_4 is a regular edge, the two edges splitted from e_4 are irregular edges as shown in Fig. 20f, and $ID(e_5) + ID(e_6) = 2$. The inherent reason for the differences of irregular degree is that Q_1 and Q_2 separate the adjacent hex sets into different configurations as shown in Fig. 20b and Fig. 20e.

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

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

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

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

4.3.2. Optimization of Quad Set Based on Dual Structure

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. In fact, however, it is 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 degree. 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 on the quad set, and sometimes it may be even not convergent. In this paper, we propose an optimization method based on the dual structure of the quad set. Our method not only avoids the global optimization difficulty, but also guarantees the mesh quality after sheet inflation.

On a quad mesh, starting from an edge, we can recursively get all the edges that are topologically parallel to this edge. These edges and the adjacent quads form a chord, the dual structure of the quad set[3, 12]. Similar to the quad mesh, the quad set used for sheet inflation also consists of chords, and all the mesh edges on the boundary loop pairwise belong to one chord. For example, in Fig. 21b, 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 a quad mesh and a quad set for sheet inflation is that there is a sheet associated with each chord on the quad set. For example, in Fig. 21c, e_3, e_4 and their chord are contained in the sheet s (red). This relationship reveals that if two edges on the boundary loop belong to a same chord, then we can find other chords connecting these two edges within the corresponding sheet.

We improve the quality of the quad set by iteratively improving the quality of the chords that contain the edges on the boundary loop. The chords are neither too constrained as the whole quad set nor too local as one single concave or convex edge. By optimizing the chords, we find a tradeoff between globality and locality. Suppose the quad set is Q and its edge set is E, the details of the optimization procedures are explained:

1. Divide the edges on the boundary loop into groups according to whether they are in the same sheet. For example, e_3 and e_4 are grouped in Fig. 22a, and e_1 e_4 are grouped in Fig. 23a;
2. Select a group of edges that has not been processed. Get all the edges in the corresponding sheet and the quads adjacent to these edges. These edges and quads compose the searching space for new chords that connect these edges. Suppose the edge group is $g = \{e_3, e_4\}$, Fig. 22b shows the edges that are in the same sheet of g and the adjacent quads;

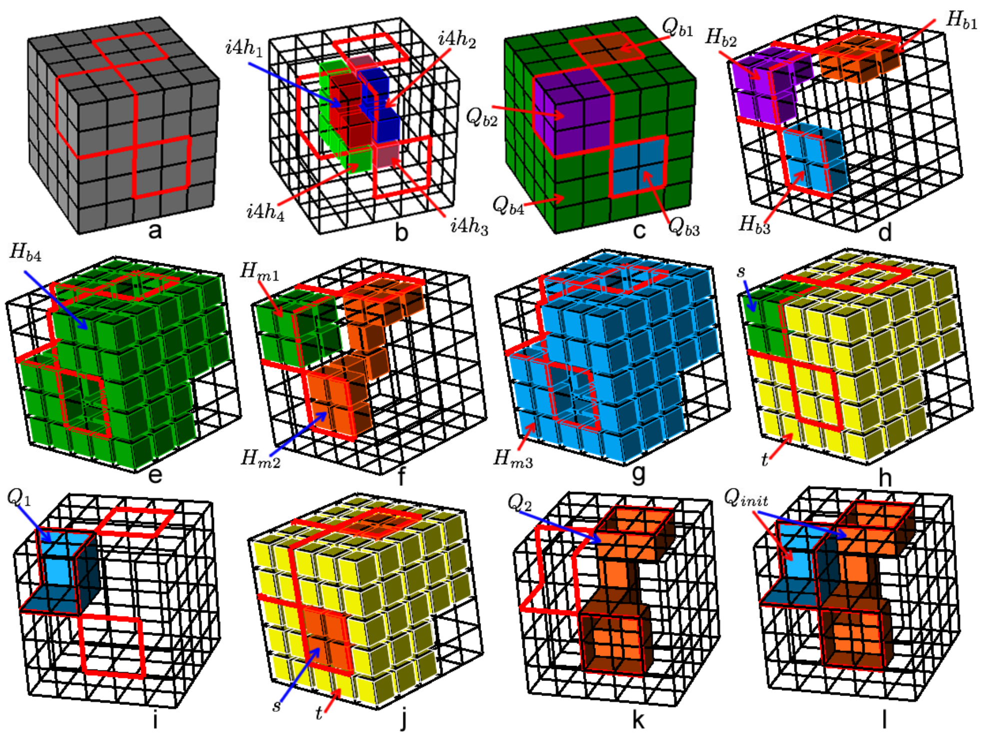

Figure 19: Process of determination of initial quad set from a non-intersecting boundary loop: (a)the non-intersecting boundary loop; (b)the local hex set; (c)the two boundary quad subsets Q_{b1}(orange) and Q_{b2}(blue); (d)two hex sets H_{b1}(orange) and H_{b2}(blue); (e)the initial quad set Q_{init}.

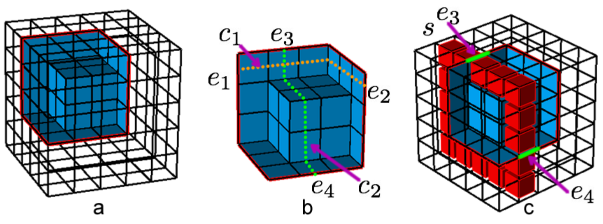

Figure 21: Dual structures of quad set: (a)the quad set; (b)two chords c_1 and c_2 on the quad set; (c)the sheet s that contains e_3 and e_4.

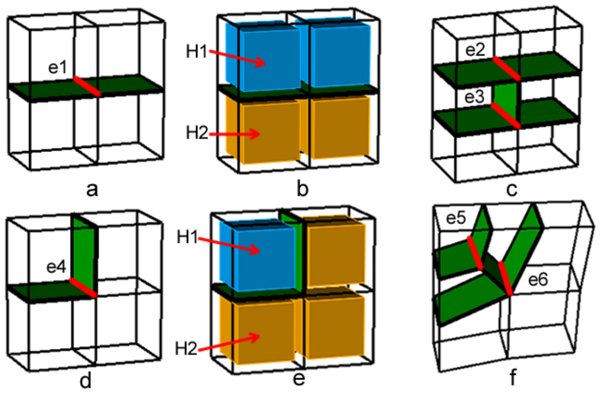

Figure 20: Different impact on the mesh quality of different quad sets: (a)quad set Q_1(green) and one of its mesh edges e_1(red); (b)the hex set adjacent to e_1 is separated into two subsets H_1 and H_2 by Q_1; (c)e_1 is splitted into two edges e_2 and e_3 after inflation; (d)quad set Q_2 and one of 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.

3. Since the edges in the group pairwise connect to one chord, we use A* algorithm to search for the new chords. While searching, we take the edges' variation of irregular degree into consideration. For example in Fig. 22c, there are three quads adjacent to q_1. If we select q_2 or q_4 as the next forward step, the variation of irregular degree of e_5 will be 2; if we select q_3 as the next forward step, the variation of irregular degree e_5 becomes 0. Hence we select q_3. The new chord connecting e_3 and e_4 is shown in Fig. 22d. If there are more than two edges in the group, it needs deciding which two edges should be paired. For example, four edges e_1 e_4 in Fig. 23b are in the same group. If we connect e_1 and e_3 and get the new chord c_3 (blue in Fig. 23c), then e_2 and e_4 cannot be connected without intersecting with c_3 which is not allowed for keeping the quad set valid as shown in Fig. 23d. So we search new chords between e_1 and e_4, e_2 and e_3. The two new chords are shown

in Fig. 23e;

4. The new chords and the original chords encompass a hex set on the sheet. For example, the hex set h in Fig. 22e is encompassed by the two chords c_2 an c_4, and the hex set h in Fig. 23f is encompassed by the four chords c_1,c_2,c_4 and c_5. We get the boundary quad set Q_h of h, and let Q 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. 22f and Fig. 22g;
5. Repeat the process of Step 2 4 until all the edge groups are handled.

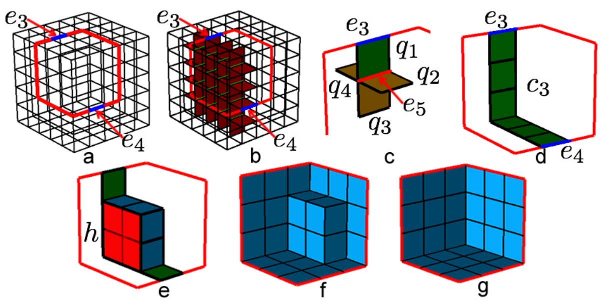

Figure 22: Dual structures of quad set: (a)the quad set; (b)two chords c_1 and c_2 on the quad set; (c)the sheet s that contains e_3 and e_4.

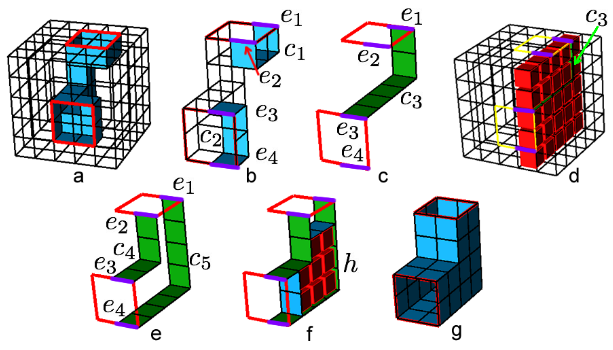

Figure 23: Dual structures of quad set: (a)the quad set; (b)two chords c_1 and c_2 on the quad set; (c)the sheet s that contains e_3 and e_4.

References

[1] 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.

[2] Shepherd JF, Johnson CR. Hexahedral mesh generation constraints. Engineering with Computers 2008;24(3):195–213.

[3] 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.

[4] Tautges TJ, Knoop SE. Topology Modification of Hexahedral Meshes Using Atomic Dual-based Operations. IMR 2003;:415–23.

[5] Ledoux F, Shepherd J. Topological modifications of hexahedral meshes via sheet operations: a theoretical study. Engineering with Computers 2009;26(4):433–47.

[6] Ramos JS, Ruiz-Gironés E, Navarro XR. Unstructured and semi-structured hexahedral mesh generation methods 2014;.

[7] Mitchell SA, Tautges TJ. Pillowing doublets: refining a mesh to ensure that faces share at most one edge. 4th International Meshing Roundtable 1995;.

[8] Suzuki T, Takahashi S, Shepherd J. An interior surface generation method for all-hexahedral meshing. Engineering with Computers 2010;26(3):303–16.

[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] Chen J, Gao S, Zhu H. An improved hexahedral mesh matching algorithm. Engineering with Computers 2015;doi:10.1007/s00366-015-0414-1.

[11] Staten ML, Shimada K. A close look at valences in hexahedral element meshes. International Journal for Numerical Methods in Engineering 2010;:899914doi:10.1002/nme.2876.

[12] Mitchell S. A characterization of the quadrilateral meshes of a surface which admit a compatible hexahedral mesh of the enclosed volume. STACS 96 1996;.