Fine-Grained Parallel Algorithm for Unstructured Surface Mesh Generation

Jianjun Chen[1,2,*], Dawei Zhao[1], Yao Zheng[1], Zhengge Huang[1], and Jianjing Zheng[1]

[1] Center for Engineering and Scientific Computation, and School of Aeronautics and Astronautics, Zhejiang University, Hangzhou 310027, China
chenjj@zju.edu.cn
[2] Civil and Computational Engineering Centre, School of Engineering, Swansea University, Swansea SA2 8PP, Wales, U.K.

Abstract. Surface mesh generation is one time-consuming step in preparing an unstructured mesh model. However, its parallelisation attracts little attention. In this study, a fine-grained parallel surface meshing algorithm is proposed by taking advantage of the parallelism within the meshing process of a single face. Compared with the scheme which meshes the faces individually in parallel, the proposed algorithm behaves better in terms of parallel efficiency and scalability. One integral part of the proposed algorithm is a novel domain decomposition approach, which decomposes the simplified version of an input background mesh rather than the input mesh itself. The simplification procedure is based on a set of well-designed operations on the dual graphs of the mesh. No undesirable features are formed in subdomain boundaries; hence, no postprocessing steps are required to improve the quality of elements adjacent to subdomain boundaries.

Keywords: Mesh generation, Parallel algorithm, Surface mesh, Domain decomposition, Graph partitioning.

1 Introduction

Thanks to the rapid advance of High Performance Computing (HPC) technologies, parallel machines are more and more cost-effective. In both the academic and industry communities, various CFD codes have been parallelised for many years to exploit the huge computing power of parallel machines efficiently. It was reported that some academic parallel CFD codes were able to exploit hundreds of thousands of computer cores to efficiently solve a problem containing billions of elements [1]. In the aerodynamics industry, CFD simulations usually involve thousands of computer cores, and the simulation time ranges from hours to days.

* Corresponding author.

J. Sarrate & M. Staten (eds.), *Proceedings of the 22nd International Meshing Roundtable*,
DOI: 10.1007/978-3-319-02335-9_31,

However, the wall-clock time to finish a complex simulation is far more than that consumed by a parallel simulation code, of which the time for mesh generation accounts for a large portion. For instance, it usually takes weeks or more to prepare a block-structured mesh in the exterior flow simulation of a complete aircraft model, even when the engineer is an expert user of a state-of-the-art commercial or in-house meshing tool [2]. Unstructured mesh generation does not require a painful process to decompose a complex domain into blocks, hence it is more automatic. However, the process of generating an unstructured mesh composed of tens of millions mesh nodes may consume hours of CPU time if executed sequentially [3]. In practice, when the input geometry is very complex, or a high standard is set for mesh quality, mesh generation is a trial-and-error process, and may need to be repeated many times. Therefore, the wall-clock time for preparing a large-scale unstructured mesh is usually comparable with the time for conducting parallel simulations.

It is well known that parallel algorithms can speed up the meshing process. Parallel mesh generation has been studied since the 1990s, and the principal motivation was to overcome the memory bottleneck to generate a large-scale mesh in the early stage of parallel meshing study [4, 5]. Nowadays, 64 bit computers have become mainstream and a desktop computer may be configured with a large amount of memory at an affordable price. Hence, the memory issue is no longer prominent, although to break it is still beneficial. However, the other motivation of developing parallel meshers, i.e., to overcome the time bottleneck of generating a large-scale mesh, is still meaningful. To fully exploit the computing power of ever emerging parallel computers, a completely parallelised simulation environment is now highly demanded, where both the simulation code and its pre and post processing codes are required to be parallelised [6].

For a typical 3D simulation, surface mesh generation, volume mesh generation, and volume mesh improvement are three major steps involved in the preparation of an unstructured mesh model. We have presented parallel schemes for volume mesh generation and improvement recently [7]. In this study, we attempt to parallelise the final sequential step, i.e. parallel surface meshing, in order to develop a complete parallel pipeline for the generation of large-scale unstructured meshes.

Parallel volume meshing study abounds in the literature [4, 5], whereas its surface counterpart has been discussed rarely so far. In [8], a very simple scheme was proposed for CAD models composed of many faces. The original sequential process meshes each face individually. So, considering the process of meshing each face as a task, a task pool can be initialised. A manager/worker model is then started to parallelise the surface meshing process: the manager monitors the task pool and sends the tasks to hungry workers; the workers demand the unmeshed faces from the manager and then mesh them. This scheme is essentially not scalable, and in practice, its efficiency highly depends on geometric natures of the surface model. If the number of faces (M) is far more than the number of computer cores (N), and the meshing time of each face (referred to as the load of a face hereafter) is roughly equal, a high efficiency is possible; otherwise, the efficiency may be very low.

In the parallel mesher developed in this study, the above parallel scheme is revisited for its simplicity. Meanwhile, an enhanced scheme is proposed to improve its scalability and efficiency. In the enhanced scheme, a face with large loads will be split into many smaller subdomains, and the meshing problem of each subdomain is individually conducted. Therefore, the proposed scheme is called a *fine-grained* algorithm because it takes advantage of the parallelism within the meshing process of a face. The number of tasks is scalable with N rather than being fixed as M. Correspondingly, the load difference between different tasks no longer depends on the face loads, but is controlled under a limit to avoid the case where the time consumed by one single task dominates. Moreover, each subdomain will have a similar representation like the original face, and can be meshed without modifying the sequential mesher. This full reuse capability of sequential codes is very desirable for a parallel meshing algorithm because it minimises the cost of paralleling a sequential mesh code that is improved constantly.

The decomposition of a surface model is another essential part of the proposed parallel scheme, where a subdomain boundary without small angles is desired. The domain decomposition tools that prevail in parallel solutions of Partial Differential Equations (PDEs) mainly focus on reducing load imbalances and interface communications, and are incompetent to avoid the generation of poorly shaped subdomain boundaries that are harmful in the context of parallel mesh generation:

(1) Some mesh generation schemes require that the boundary angles are within certain bounds in order to guarantee the termination and to achieve a provably good element quality. When these schemes are employed on subdomain mesh generation, the artificial features such as small angles are prohibitive [9-12].
(2) Poorly shaped boundaries are still troublesome when the employed mesh generation scheme has no constraints on boundary shapes because low-quality elements are usually formed in the neighbourhoods of these features. A time-consuming post-processing step to improve these elements is then indispensable, e.g., assimilating the submeshes (if the memory allows) and then improving the assimilated mesh sequentially [7, 13, 14], or improving the distributed mesh concurrently at the cost of inter-processor operations [15].

In this study, a novel approach for domain decomposition is examined, featuring its ability to produce subdomains having good geometric properties in their boundaries, except to meet the fundamental requirements of loads and communications. The domain to be decomposed is interiorly filled up with a non-overlapping mesh, referred to as a background mesh hereafter. Instead of directly sending this mesh to a graph partitioner [16, 17], which usually produces badly shaped subdomain boundaries, an intermediate procedure is proposed to simplify the mesh based on the well-defined operations related to the dual graphs of the mesh. Partitioning the simplified mesh rather than the initial mesh will produce a distributed mesh that not only fulfils the dual goals of load balancing and minimisation of communications, but also contains desirable geometric properties in the inter-domain boundary.

2 Domain Decomposition Approach

2.1 Basic Terms and Definitions

Definition 1 (Non-overlapping Mesh): A non-overlapping *d*-dimensional mesh is a group of mesh entities whose dimensions range from 0 to *d*, i.e.

$$M = \{E^i \mid i = 0,1,...,d\},$$

where

$$E^i = \{e_k^i \mid k = 0,1,...,n_i\} \quad (i = 0,1,...,d)$$

refers to the set of *i*-dimensional mesh entities, and

(1) $\forall e_k^i, e_l^i \in E^i (0 \le i \le d)$, e_k^i does not intersect e_l^i, or intersect e_l^i at a mesh entity whose dimension is less than i;

(2) $\forall e_k^{i-1} \in E^i (0 < i \le d)$, it must be a boundary entity of one or more *i*-dimensional entities.

In a non-overlapping mesh, elements and sides refer to mesh entities with $i = d$ and $i = d - 1$, respectively. If an (*i*-1)-dimensional entity e_l^{i-1} is the boundary entity of an *i*-dimensional entity e_k^i, then name e_k^i the domain entity of e_l^{i-1}. Each mesh side may have one or two domain entities. If the side has one domain entity, it is called a boundary side; otherwise, it is called an interior side. Two mesh entities e_k^i and e_l^i are neighbours if both of them are the domain entities of an (*i*-1)-dimensional entity e_m^{i-1}.

More strictly, the sides of a 3D mesh must be defined by co-planar vertices in order to define the angles adjacent to neighbouring sides accurately. If two boundary sides of an element are neighbours, the angle they form interior of the element is called the interior angle of the element.

Definition 2 (*i*-Dimensional Dual Graph): Given a *d*-dimensional non-overlapping mesh

$$M = \{E^i \mid i = 0,1,...,d\},$$

an *i*-dimensional dual graph ($1 \le i \le d$) of M is denoted as $G = \{V, A\}$, where V is the set of graph nodes, and A is the set of graph edges. Each graph node $v_k \in V$ corresponds to one mesh entity e_k^i in M, i.e.

$$v_k = \phi_v(e_k^i), \text{ and } e_k^i = \phi_v^{-1}(v_k).$$

If two mesh entities of M, e_k^i and e_l^i, are neighbours across a shared boundary entity e_m^{i-1}, an graph edge $a_{kl} \in A$ exists, and

$$a_{kl} = < v_k = \phi_v(e_k^i), v_l = \phi_v(e_l^i) >.$$

Particularly, a_{kl} is called a graph edge classified on the mesh entity e_m^{i-1}.

The nodes and edges of a dual graph may be weighted for some application-specific purposes. Assuming that $w_v(v)(v \in V)$ and $w_a(a)(a \in A)$ are the node and edge weight functions, respectively, the weighted dual graph is a tetrad, i.e.

$$G = \{V, A, w_v(v), w_a(a)\}.$$

According to Definition 2, the **S**ide **D**ual **G**raph (**SDG**) and **E**lement **D**ual **G**raph (**EDG**) are two specific cases of the dual graphs of a mesh, i.e. the (*d*-1)-dimensional and *d*-dimensional dual graphs of the mesh, respectively.

Definition 3 (Node Deletion): Node deletion defined on a graph $G = \{V, A\}$ is a local modification operation with respect to a node v_k of G. After deletion, a new graph $G' = \{V', A'\}$ is generated. The set of new graph nodes is defined as $V' = V \setminus \{v_k\}$. The set of new graph edges is defined by removing all of the edges in G that are adjacent to v_k.

Definition 4 (Edge Contraction): Edge contraction is a local modification operation defined on a weighted graph $G = \{V, A, w_v(v), w_a(a)\}$ with respect to an edge $a_{kl} =< v_k, v_l >$ of G. After contraction, a new graph $G' = \{V', A', w_v'(v'), w_a'(a')\}$ is generated. The set of new graph nodes is defined as $V' = V \setminus \{v_k, v_l\} \cup \{v_m'\}$, where v_m' is a new graph node, and $w_v'(v_m') = w_v(v_k) + w_v(v_l)$. The set of new graph edges is defined as follows: the edge a_{kl} is removed; and each edge $a_{kn} =< v_k, v_n >$ (or $a_{ln} =< v_l, v_n >$) that is adjacent to v_k (or v_l) and another graph node v_n is removed, instead, an edge $a_{mn}' =< v_m', v_n >$ is inserted. In the case that both v_k and v_l are adjacent to v_n,

$$w_a'(a_{mn}') = w_a(a_{kn}) + w_a(a_{ln});$$

otherwise

$$w_a'(a_{mn}') = w_a(a_{kn}) \text{ (or } w_a'(a_{mn}') = w_a(a_{ln})).$$

Fig. 1 presents a simple example to explain the above definitions. The left part of Fig. 1a is a 2D non-overlapping mesh, which consists of one triangular and two quadrilateral elements. The left parts of Figs. 1b and 1c correspond to the EDG and SDG of this mesh, respectively. Here, only the EDG is weighted, and the weights of its nodes and edges are initially set to be 1.0. The right part of Fig. 1a shows the simplified mesh obtained by deleting the side *de*, where the adjacent small angles *def* and *aed* vanish as well. Consequently, the elements labelled as 1 and 3 are merged to a new pentagonal element, labelled as 13. Correspondingly, an edge-contraction operation and a node-deletion operation are required on the EDG and SDG to obtain the respective duals of the simplified mesh, as illustrated in Fig. 1b and 1c.

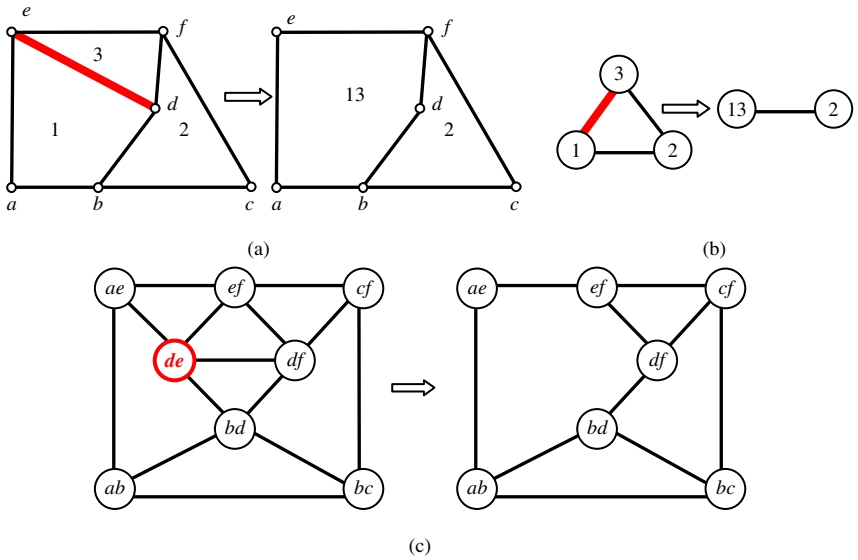


Fig. 1 Meshes and their dual graphs. (a) A side-deletion operation simplifies the mesh into a new one that has fewer small angles. (b) An edge-contraction operation transforms the initial EDG into a dual of the simplified mesh. (c) A node-deletion operation transforms the initial SDG into a dual of the simplified mesh.

2.2 The Domain Decomposition Flowchart

Fig. 2 presents the basic flowchart of the proposed domain decomposition approach, where the frames, solid arrows and dotted arrows represent the data, algorithms and inputs of the algorithms, respectively. The input is a non-overlapping mesh composed of any type of polyhedral or polygonal elements (triangular mesh in this study). The output is the partitioned result of the input mesh, where no subdomains (submeshes) contain artificial small angles, and optionally, poorly shaped sides in their boundaries. If the input mesh is appropriate, the dual goals of load balancing and minimisation of communications are obtained as well. The intermediate steps that connect the input and output are as follows.

(1) Build the SDG and EDG of the input mesh.
(2) Identify some mesh sides with undesirable shapes as removable, and simplify the SDG by deleting the graph nodes that correspond to removable sides. In this study, this option is disabled by default.
(3) Simplify the SDG by deleting some graph nodes, and identify the mesh sides they correspond to as removable, to ensure that the mesh without these sides contains no small interior angles less than a predefined threshold.
(4) Simplify the EDG by contracting all of the edges that are identified as removable in Steps 2 and 3.

(5) Decompose the simplified EDG into a predefined number of partitions using a graph-partitioning tool, aimed at the dual goals of load balancing and minimisation of communications.
(6) Decompose the input mesh into subdomains according to the partitioning result of the simplified EDG.

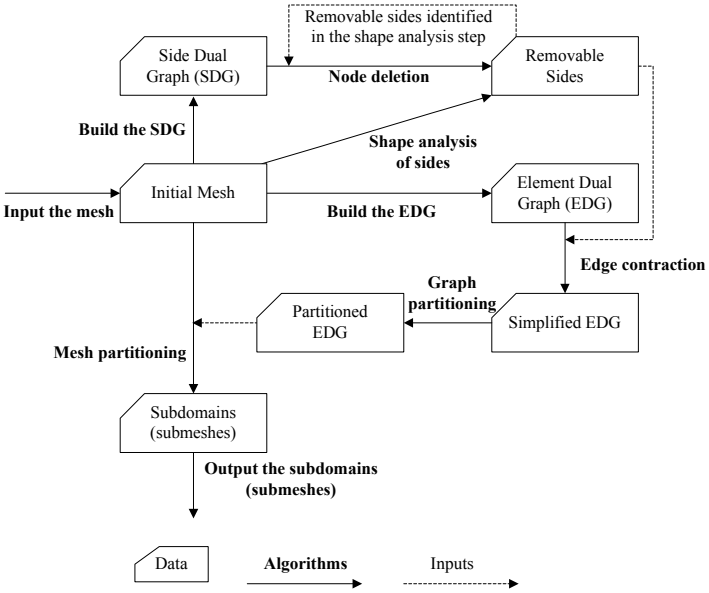


Fig. 2 The flowchart of the proposed domain decomposition approach

2.3 Node Deletion of the SDG

Step 3 of the proposed domain decomposition approach can ensure that no small interior angles (except those adjacent to two boundary sides) will survive in the simplified mesh with a minimal number of mesh sides removed. This is achieved by the node-deletion operation defined on a weighted SDG. To clarify it, two definitions are presented as follows:

Definition 5 (Weighted SDG): Given a d-dimensional non-overlapping mesh M and a predefined angle threshold β_{th}, the weighted SDG is the (d-1)-dimensional dual graph of M, with its nodes and edges weighted as follows.

(1) For each graph edge, its weight equals the value of the interior angle bounded by two sides that the end nodes of the graph edge represent. In the case that there are two such angles (for example, in the right mesh of Fig. 1a, both angles bounded by *bd* and *df* are interior of the domain), the smaller value will be weighted to the graph edge.

(2) For each graph node, if it corresponds to a boundary side, its weight equals zero; otherwise, its weight equals the number of its adjacent edges whose weights are less than the predefined threshold β_{th}.

Here, a graph node with a non-zero weight is called an *unqualified node*, and an SDG containing unqualified nodes is called an *unqualified SDG*, which corresponds to a mesh having small interior angles.

Because Definition 3 only applies for the unweighted graph, a new definition is presented for the node-deletion operation of a weighted SDG.

Definition 6 (Node deletion of a weighted SDG): Node deletion defined on a weighted SDG is a local modification operation related to a node v_k of this graph. The graph topologies are updated as in Definition 3. The weights of the remaining nodes and edges are kept unchanged except for the following particular cases:

(1) For each unqualified node adjacent to v_k, its weight decreases by 1.

(2) For each graph edge ended with the nodes adjacent to v_k, if it is the only one graph edge classified on one ($d-2$)-dimensional mesh entity (d is the mesh dimension), reset its weight value as the smaller one of w_e and $2\pi - w_e$, where w_e is the original edge weight.

Fig. 3 illustrates the above definitions with the mesh examples shown in Fig. 1a. It is supposed that three angles are less than the predefined threshold in the unsimplified mesh, i.e. $\angle dfc$, $\angle fde$ and $\angle def$. The interior sides that bound these angles are *de* and *df*, and each of them is adjacent to two of the three small angles. Therefore, their corresponding graph nodes are weighted to be 2, and the other nodes are all weighted to be 0. After the graph node *de* is removed, its adjacent edges are removed as well, and the weight of the node *df* decreases from 2 to 1. In the meantime, the graph edge ending with *bd* and *df* becomes the unique edge classified on the mesh vertex *d*; therefore, the weight of this edge is set to be $\angle fdb$, which is smaller than the original edge weight $\angle bdf$.

The input of the node-deletion step is an unqualified SDG, and the output is a subgraph of the input that contains no unqualified nodes. The simplest solution is to delete all of the unqualified nodes in the input SDG. However, the preferred solution shall attempt to delete the graph nodes as few as possible:

(1) For the interior sides that form less-than-threshold angles with boundary sides, remove their corresponding unqualified nodes in the SDG by node-deletion operations.

(2) Initialise a priority queue for the remaining unqualified graph nodes in a descending order of the node weights, and repeat the following steps until the queue is empty:
   (a) Remove the head node of the queue by a node-deletion operation; consequently, the weights of the adjacent unqualified nodes will decrease by 1.

(b) If the adjacent nodes become qualified ones, remove them from the priority queue; otherwise, insert them in the right positions of the priority queue.

(3) Label the interior mesh sides that correspond to the deleted graph nodes as removable.

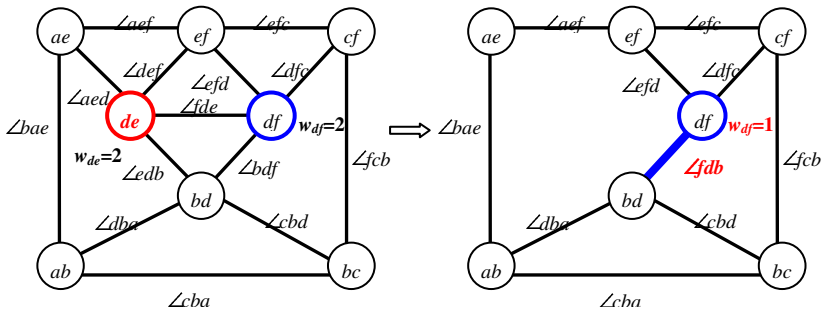


Fig. 3 Illustration for the weighted SDG and the node-deletion operation defined on it. The corresponding meshes are shown in Fig. 1a.

2.4 *Edge Contraction of the EDG*

Step 4 of the proposed domain decomposition approach inputs the EDG of the input mesh, and outputs the EDG of the simplified mesh. The EDG is also a weighted graph, and the weighting strategy depends on the application purposes. For parallel mesh generation, the graph nodes and edges represent the elements and sides of the background mesh, respectively. The domains restricted by the background elements and sides will be refined in the following steps according to the predefined mesh sizing information. Therefore, the nodes and edges of the EDG will be weighted with the predicated numbers of the finer elements and sides to be generated on their corresponding mesh entities, respectively. The accuracy of the predication is vital to the quality of the subsequent domain decomposition results in terms of load balancing and minimisation of communication.

Instead of sending the EDG that corresponds to the input mesh for partitioning, a variant of this EDG is needed here to prevent removable sides recognised in Steps 2 and 3 from appearing in the inter-domain meshes. Here, two types of variant EDGs are considered.

(1) The first one is generated by *penalising* the graph edges that correspond to removable sides with very large weights [7]. Because the graph partitioner [16, 17] intends to minimise the size of the cutting edge weights, it may prevent these penalised edges from becoming cutting edges.

(2) The second one is generated by contracting all of the graph edges that represent *removable* sides. For the initial EDG, each graph edge only represents

one mesh side. The graph edge is identified as *contractable* if the corresponding mesh side is removable. However, in the intermediate process of edge contraction, one graph edge may represent more than one mesh side. The graph edge is identified as contractable if any of these sides are removable. For instance, the graph edge between the nodes 13 and 2 shown in Fig. 1b represents the mesh sides *bd* and *df*, and if either of them is removable, this graph edge will be required to be contracted later.

It is evident that the process based on the first type of EDG is much simpler. Nevertheless, the following consideration justifies why the second one is selected as the default. The graph partitioner does intend to minimise the weights of cutting edges, but the performance highly relies on the input. When the percentage of edges with very large weights is very small, the partitioner will work as expected; otherwise, the partitioner may occasionally fail to prevent these edges from becoming cutting edges. When the partitioner works abnormally, the goal of minimisation of cutting edge weights may lead to an undesirable result because the punitive weights may stray a lot from the actual meaning of the edge weights, e.g. the number of the final mesh pieces to be generated in the side represented by this graph edge.

3 Background Meshes

When the proposed domain decomposition approach is applied for parallel mesh generation, an issue remains regarding the generation of the input background mesh. In the previous studies, three types of background meshes have been examined, as illustrated by a 2D example in Fig. 4:

(1) *Type-I*. The domain boundary is discretised into pieces that conform to the prescribed size map. Then the boundary points are incrementally inserted to form a Delaunay triangulation. After constrained boundary recovery, a coarse mesh is defined interior of the domain. This mesh contains no interior Steiner points except those occasionally generated in the boundary recovery procedure for 3D domains. Said *et al.* [14] have even applied this type of mesh for parallel tetrahedral mesh generation.
(2) *Type-II*. The type-I mesh is refined by inserting interior field points. A map with enlarged interior element sizes controls the termination of this refinement procedure in order to generate a very coarse background mesh. Löhner [6] has recently investigated the application of this type of mesh in a parallel advancing front mesher.
(3) *Type-III*. The prescribed size map is scaled, and both the domain boundary and interior are discretised with the scaled map. Given that the number of elements decreases with the power of the element size, a background mesh with elements whose edge lengths are n times as large as the desired one will only contain n^{-2} (n^{-3} for the volume mesh) elements as the desired mesh. Ito *et al.* [13] have examined a parallel advancing front mesher with this type of background mesh as input.

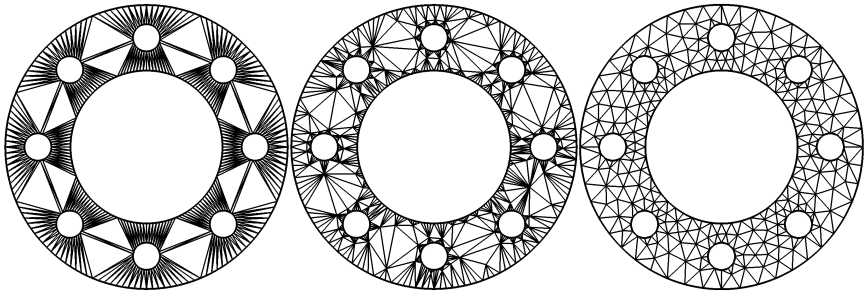


Fig. 4 Three types of background meshes for the domain decomposition process of parallel mesh generation. (a) Type-I: boundary conforming mesh having no field points. (b) Type-II: boundary conforming mesh with a small number of field points. (c) Type-III: a coarse mesh adapt to a scaled size map for both the domain boundary and interior.

The type-I and type-II background meshes are of similar type in terms that their boundary surface element sizes conform to the final requirement while the interior meshes are far coarser. Therefore, when meshing the subdomains composed of this kind of background element, only the inter-domain boundary is refined, and the original surface boundary will be kept unchanged. However, for the type-III background mesh, both the surface and volume meshes are far coarser than the final ones, and are required to be refined in the parallel mesh generation process. The type-III background mesh is adopted in this study because it has some advantages over the former two background meshes.

(1) A key issue of domain decomposition is regarding how to weight the background elements and sides accurately. These weights refer to the numbers of mesh pieces that will be finally generated in these background entities. If the type-I and type-II meshes are input, the weights are usually estimated by the volumes, areas or lengths of the background entities and certainly inaccurate when the interior element sizes are highly graded. However, if the type-III mesh is input, a uniform weighting strategy can be adopted initially because it is generated under a coarsened size map that roughly scales to the final one in both the domain boundary and interior.
(2) The performance of the proposed domain decomposition approach depends on how intensive the dual graph is simplified. If the type-I or type-II mesh is input, the simplification will be rather intensive and result in a dual graph with many nodes and edges over-weighted. Consequently, the decomposition of this over-simplified graph will yield a highly unbalanced load distribution. Linardakis *et al.* [18] proposed to solve this issue by employing a recursive decomposition scheme, which involves more computing costs and it remains a challenge to extend this scheme for three-dimensional problems. However, if a type-III background mesh is input, the intensity of simplification usually maintains in a very low lever so that the decomposition of the simplified graph can yield hundreds of partitions with a satisfactory load distribution. More importantly, this scheme is extendible for three-dimensional problems.

Nevertheless, it needs emphasis that the type-I and type-II background meshes are also candidate inputs for the proposed domain decomposition approach when surface problems are considered. A more detailed comparison between the domain decomposition approaches based on these types of background meshes will be interesting, but is beyond the scope of this study.

4 Parallel Surface Mesh Generation

In general, the mesher processes the surface model in a bottom-up manner, i.e., the boundary curves are meshed first, and then the faces are meshed individually with their boundary meshes as inputs. The curve discretisation usually runs very fast, and the face discretisation dominates the whole process in terms of CPU time.

Tremel *et al.* [8] suggested a simple parallel surface meshing scheme that meshes the faces individually in different processes. Its parallel efficiency and scalability may be very low when the model contains a small number of faces and/or the meshing time for one or several faces dominates. Based on the proposed domain decomposition procedure, a fine-grained parallel algorithm is suggested to overcome these drawbacks in this study, which takes the following steps:

(1) Input the CAD model and element-sizing definition.
(2) Generate a type-III background mesh using the coarsened sizing definition.
(3) Decompose the faces with large loads into subdomains (the faces with small loads are considered as one subdomain).
(4) Mesh each subdomain individually on available computer cores using the dynamic load balancing strategy.
(5) Merge the submeshes into one unified mesh.

4.1 Background Mesh Generation

The background mesh is generated using a coarsened element-sizing map. In the CFD simulations, the typical element-sizing field is defined by a background mesh and many grid sources [19]. With a scale factor input by the user, this size field can be roughly coarsened by scaling up the size values defined on background nodes and grid sources. In the region where geometrical proximity features are prominent [20], the scaled size values may be too large so that the resulting type-III background mesh contains badly shaped elements in this region. A more reasonable coarsening scheme allows the smaller scale factor defined in this region. Anyway, because the proximity features only impact a small fraction of elements, the simpler scheme is adopted in this study.

With the size map coarsened, the coarse-grained parallel scheme which meshes each face individually is adopted to generate the background mesh.

4.2 Subdomain Formation

Because a dynamic load balancing strategy is adopted in the subsequent parallel subdomain meshing step **[7, 21]**, the total number of subdomains (M) ought to be several times that of the number of computer cores (N). Given N and a multiplying factor F_1 (default value is 8), the desirable number of subdomains is $N \times F_1$. Assuming that the total number of background elements is E, then the expected number of background elements for each subdomain is

$$E_s = E/(N \times F_1).$$

For a face with E_f background elements, the expected number of subdomains is

$$M_s = (\mathbf{int})[E_f/(E_s \times F_2) + 0.5],$$

where F_2 is an amplification factor (default value is 1.3).

For the face whose M_s value is greater than 1, the proposed domain decomposition approach is adopted to split the background mesh to submeshes. In order to fully reuse the sequential mesher whose input is a group of trimmed surfaces, an *artificial* trimmed surface is reconstructed for each subdomain. As we know, a support surface and several loops of boundary curves that limit the valid geometric region are two ingredients necessary to represent a trimmed surface [22, 23]. Here, a subdomain can inherit the support surface from the original face, and its boundary curves are the boundary lines of the submesh. However, although the end points of these boundary lines are located on the support surface, their interiors may deviate a lot from the surface. To get boundary curves that are close enough to the support surface, these lines need be *projected* onto the support surface.

The boundary lines of a submesh are grouped into two different categories: the lines defined on the original boundary curves and the interior mesh lines. The projection procedure for the first type of lines is rather simple. If the corresponding boundary curves cross over several subdomains, they are trimmed (or split when the support mesher cannot represent the trimmed curve); otherwise, the curves are fully reused. For the interior mesh lines, the projection procedure is conducted individually. For each line, it is refined to a required number of segments first, and all the mesh points are then projected back to the surface. Finally, the projection points are employed to interpolate a curve, which can be considered as an *approximate* geodesic limited by the end points of the mesh line.

The procedure that calculates the projection of a point onto a surface is time consuming. Its parallelisation is necessary to maintain the overall parallel efficiency when many computer cores are employed. Here, the projection procedure of each interior mesh line is considered as a task, and a simple parallel scheme is implemented by distributing all the tasks on the available computer cores.

4.3 Parallel Subdomain Meshing

Similar to the sequential meshing process, the parallel process also follows a bottom-up procedure, i.e., the boundary curves are discretised first, and then the subdomain interiors. This bottom-up process can ensure that the curves shared by more than one faces are discretised only once. Therefore, no difficulty will be introduced to maintain the mesh conformity on these curves in the final step of merging the subdomain meshes.

It is obvious that the boundaries and interiors of different subdomains can be meshed in parallel. Because both the number of boundary curves and that of subdomains are far more than the number of computer cores, a dynamic load balancing strategy shown in Fig. 5 is demonstrated to be very efficient to parallelise the procedures of meshing subdomain boundaries and interiors, where each single curve or subdomain is distributed by the manager process to the worker processes for meshing, respectively. Communications only exist between the manager and the workers, and no interactions are required among the workers. Therefore, the efficiency of this parallel procedure is satisfactory.

```
Manager:
while (!noTask())
   sendATask(whoIsHungry(), selectATask());
tellWorkersNoTasks();

Workers:
do {
  tellMngIAmHungry();
} while (recvAndDoTask() == 1)
```

Fig. 5 Dynamic load balancing strategy for parallel subdomain meshing

5 Results

The numerical experiments presented here were conducted on the TH-1A system managed by the National Supercomputer Center in Tianjin, China. Each node contains 2 six-core CPUs whose frequency is 2.93GHz, and the local memory for each node is 24GB. The selected geometry is the outflow computational model of the DLR-F6 wing-body-nacelle-pylon aircraft (referred to as the F6 model hereafter) that is composed of 36 faces and 119 curves [24]. The sequential mesher adopts the mapping-based advancing front algorithm, and has been integrated in our in-house preprocessing system HEDP/Pre [25]. Grid sources are configured to refine the local meshes where the smaller elements are required for a better resolution of geometrical and physical details. Fig. 6 presents the geometry, surface grid and grid sources for the F6 model in the graphical user interface of HEDP/Pre.

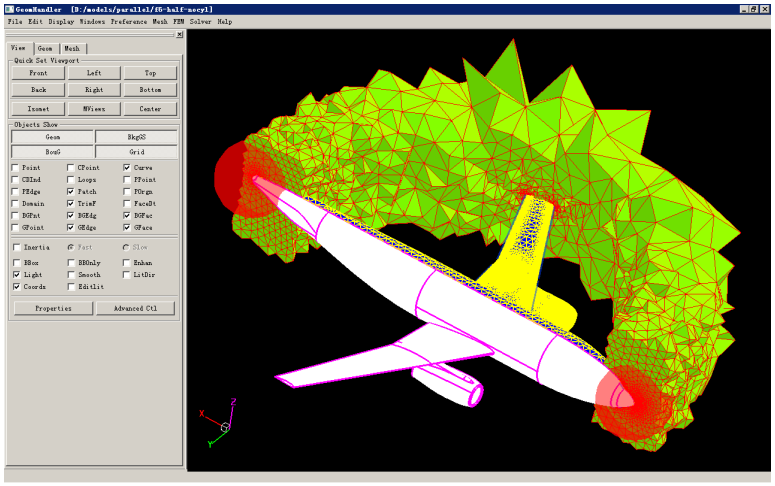


Fig. 6 Visual representation of the geometry, the coarsest surface and volume grid in the experiments for the F6 aircraft model

Fig. 7 illustrates the flowchart of parallel surface meshing and visually compares the meshes generated by the sequential and parallel meshers. Fig. 7a is the background mesh that corresponds to the coarsened size map, and Fig. 7b is the simplified background mesh and its domain decomposition output. An element of the simplified mesh may be a combination of some original triangular elements and therefore be a polygon bounded with any numbers of edges. The interior angles of any element of the simplified mesh, apart from those formed by two boundary edges, are ensured to be larger than a predefined value (60 degrees in this test). Therefore, the decomposition of the simplified mesh would not produce angles less than 60 degrees artificially on subdomain boundaries. This enables the parallel mesher to generate a mesh (Fig. 7c) whose element quality is comparable with that generated by the sequential mesher (Fig. 7d) under the constraint of respecting all of the subdomain boundary edges. The parallel subdomain meshing step would be very efficient because no inter-processor communications are required to maintain the conformity of neighboring submeshes and improve the quality of elements adjacent to subdomain boundaries. Meanwhile, the implementation of the parallel mesher suffers few coding efforts because the sequential code can be fully reused for meshing all of the subdomains.

To evaluate the performance of the proposed algorithm, surface meshes with various sizes are generated by scaling the spacing values of sources. In this test, a background mesh is generated under a coarsened size map first, which contains 13,078 elements and 6,539 nodes. Next, under a similarly refined size map, different surface meshes are generated by changing the number of computer cores. The surface mesh generated sequentially contains 694,160 elements and 347,080

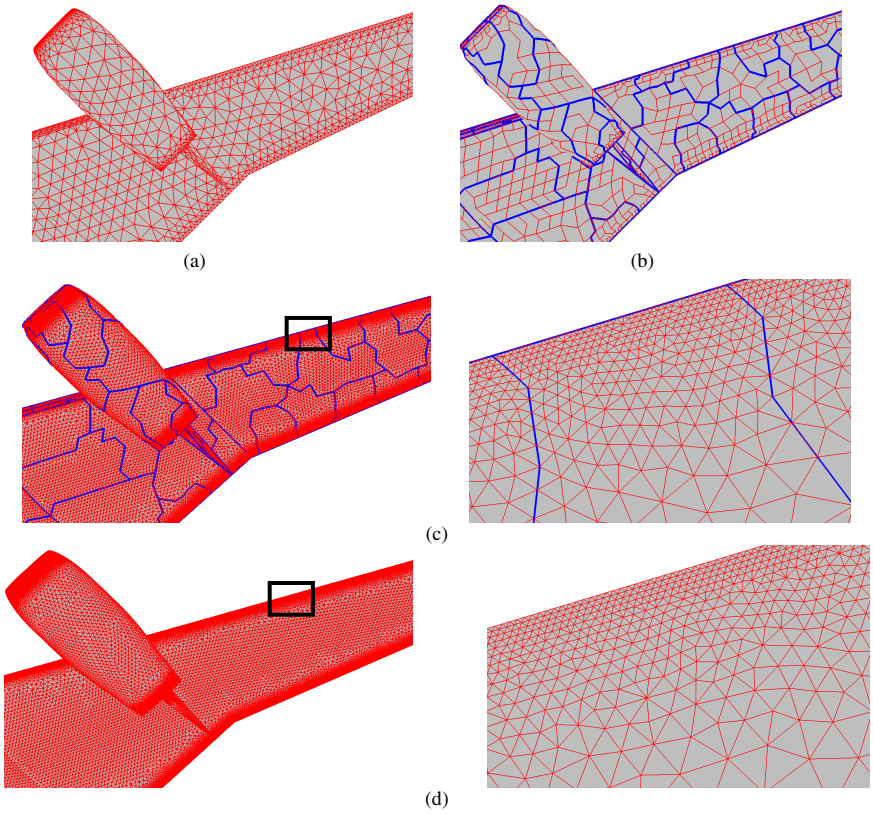


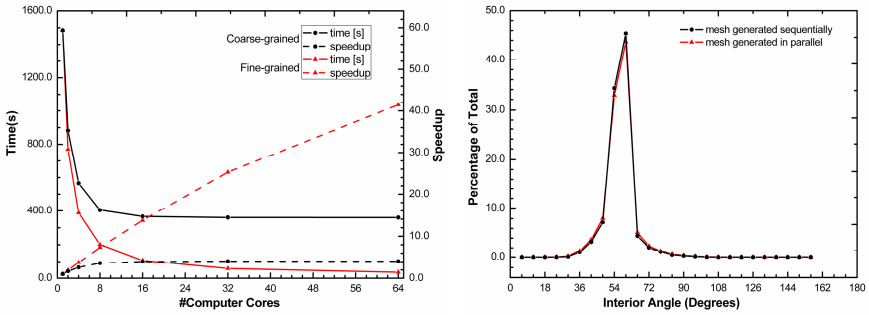
Fig. 7 The domain decomposition flowchart and the meshes generated by the sequential and parallel meshers. (a) The original background mesh. (b) The simplified background mesh and its decomposition result. (c) The mesh generated by the parallel mesher. (d) The mesh generated by the sequential mesher.

nodes, and the meshing time is 1,484 seconds. The size of the meshes generated by the proposed parallel mesher vary slightly, but the meshing time is reduced at a speed proportional to the number of computer cores employed, as shown in Fig. 8a. For instance, the time is only 36 seconds (including I/O time) when 64 cores are employed, and a speedup of 41 is achieved. If only the parallel subdomain meshing step is considered, the time is 28.3 seconds, and a speedup of 53 is achieved. As a comparison, if the conventional coarse-grained parallel mesher which only considers the inter-face parallelism instead of the intra-face parallelism is employed [8], the maximal speedup is only about 4. A speedup close to this value is achieved when 8 computer cores are employed, and it increases slightly or stays unchanged when more cores are employed. A simple analysis reveals that the limited parallel performance of the conventional mesher can be attributed to two geometric factors of the surface model. Firstly, the number of faces of this model is 36; therefore, no extra speedup will be achieved when the number of

computer cores exceeds 36. In addition, the meshing procedures of some faces of this model consume the majority of computing time, of which one procedure consumes about 357 seconds. This time cost is almost a quarter of the total time consumed by the sequential mesher, which explains why the maximal speedup of the conventional mesher is about 4.

Apart from the timing performance, the stability is another key indicator of the parallel mesher, which evaluates whether the quality data of the meshes generated sequentially and in parallel are comparable [5]. Fig. 8b compares the distribution of interior angles of elements generated by the sequential and parallel scheme that employs 64 computer cores. In general, the quality data of these two meshes are on a similar level, and only a minor difference is observed. Because the focus is on the worst elements in numerical simulation, the distribution of the minimum interior angles (α) in the range of 0 to 24 degrees is also analysed. A triangle is classified as a low-quality element if its α value is smaller than 24 degrees or as a bad element if its α value is smaller than 12 degrees. It is observed that no α value in both meshes is less than 6 degrees (the minimal values are 6.26 and 6.47, respectively). The numbers of bad elements and low-quality elements are 14 and 84 for the mesh generated sequentially, respectively. These first number changes slightly for the mesh generated in parallel, becoming 9; the second number remains unchanged, at 84.

To evaluate the entire parallel preprocessing pipeline of modeling this exterior flow problem, the surface mesh generated by employing 64 computer cores is input to the parallel volume mesher and improver we proposed in [7]. 64 cores are employed in this process as well. The resulting volume mesh is composed of 49,293,607 tetrahedral elements and 9,221,028 nodes. The time costs for the parallel volume mesher and improver are 92.17 and 64.34 seconds, respectively. As we estimate, if sequential codes are executed, the time cost for generating and improving the volume mesh of this magnitude will be about 1.5 hours. This time cost will become larger if the issue remains unresolved about how to enhance a sequential mesher or improver to obtain a linear timing performance when the mesh size increases to the magnitude of this experiment or larger. Here, the volume mesher adopted is based on the Delaunay triangulation algorithm [26], and the improver adopted is the open-source *Grummp* [27, 28] enhanced with the small polyhedron reconnection routine to improve the worst elements [29]. The time period consumed by the sequential improver is almost several times of that consumed by the sequential Delaunay mesher. Here, their parallel counterparts perform at a comparable level because the parallel meshing time includes the cost for a time-consuming domain decomposition step. To our knowledge, in the open-source community, Grummp is one of the most cost-effective tetrahedral improvers. Alternatively, Klingner [30] declared that his open-source improver *Stellar* [31] could aggressively optimize the worst tetrahedra, and thus get a final tetrahedral mesh with better quality than Grummp. However, it was also reported that the better mesh quality was achieved at a considerably higher time cost. Therefore, it requires more attention to speed up the mesh improvement procedure in the future.



(a) (b)

Fig. 8 Timing performance and element quality data of the proposed method. (a) Timing performance comparison for the fine-grained and coarse-grained algorithms. (b) Comparison for element quality of the meshes generated sequentially and in parallel.

The benefit of a complete parallel pipeline of generating an unstructured mesh of this magnitude is evident. Roughly speaking, the total time is reduced from 2 hours to 3 minutes when a moderate parallel machine (64 cores) is adopted in our experiment. This parallel pipeline is expected to be still scalable when hundreds of cores are employed **[7]**. In that case, it is hoped that the time cost of generating an unstructured mesh of hundreds of millions of elements can be reduced from 10 hours to less than 10 minutes.

6 Conclusions

Surface mesh generation is a time-consuming step when complex CAD models are input and large-scale simulations are considered. Parallelisation is an essential methodology to speed up this step. The previously proposed parallel algorithm does not consider the decomposition of a single face; therefore, its parallel performance is limited. In this paper, a fine-grained parallel algorithm that allows the decomposition of a single face is proposed. Satisfactory parallel performance is achieved irrespective of the nature of the input CAD model and the number of computer cores employed. Moreover, the proposed domain decomposition approach introduces no artificial features in subdomain boundaries. The stability of the parallel algorithm is ensured without a time-consuming postprocessing step that is usually required by many parallel meshers to improve the quality of elements adjacent to poorly shaped subdomain boundaries.

Geometry idealization and element-sizing assignment are two labor-intensive preprocessing stages of surface meshing. In the next step, we will focus on both issues to get a more automatic preprocessing pipeline. In addition, the parallel meshing issues in the unsteady CFD simulations with moved boundaries are being investigated in our group.

Acknowledgments. The authors appreciate the joint support for this project by the National Natural Science Foundation of China (Grant No. 11172267, 10872182 and 61100160), Zhejiang Provincial Natural Science Foundation (Grant No. Y1110038). The first author acknowledges Dr. Hongtao Wang in Zhejiang University for his help in accessing a parallel computer, and the joint support from Zhejiang University and China Scholarship Council for his visiting research at Swansea University, UK.

References

1. Zhou, M.: Petascale adaptive computational fluid dynamics. PhD Thesis, Rensselaer Polytechnic Institute (2009)
2. Baker, T.J.: Mesh generation: art or science? Progress in Aerospace Sciences 41, 29–63 (2005)
3. Weatherill, N.P., Hassan, O., Morgan, K., Jones, J.W., Larwood, B.G., Sorenson, K.: Aerospace simulations on parallel computers using unstructured grids. International Journal for Numerical Methods in Fluids 40, 171–187 (2002)
4. de Cougny, H. L., Shephard, M.S.: Parallel unstructured grid generation. In: Thompson, J.F., Soni, B.K., Weatherill, N.P. (eds.) CRC Handbook of Grid Generation, ch. 24. CRC Press, Inc., Boca Raton (1999)
5. Chrisochoides, N.: Parallel mesh generation. In: Bruaset, A.M., Tveito, A. (eds.) Numerical Solution of Partial Differential Equations on Parallel Computers, ch. 7, pp. 237–266. Springer (2006)
6. Löhner, R.: A 2nd generation parallel advancing front grid generator. In: Jiao, X., Weill, J.-C. (eds.) Proceedings of the 21st International Meshing Roundtable, vol. 123, pp. 457–474. Springer, Heidelberg (2013)
7. Chen, J., Zhao, D., Huang, Z., Zheng, Y., Wang, D.: Improvements in the reliability and element quality of parallel tetrahedral mesh generation. International Journal for Numerical Methods in Engineering 92, 671–693 (2012)
8. Tremel, U., Deister, F., Hassan, O., Weatherill, N.P.: Parallel generation of unstructured surface grids. Engineering with Computers 21, 36–46 (2005)
9. Chew, L.P.: Guaranteed-quality mesh generation for curved surfaces. In: Proceedings of the Ninth Annual Symposium on Computational Geometry, San Diego, CA, USA, pp. 274–280 (1993)
10. Ruppert, J.: A Delaunay refinement algorithm for quality 2-dimensional mesh generation. Journal of Algorithms 18, 548–585 (1995)
11. Shewchuk, J.R.: Delaunay refinement mesh generation. PhD Thesis, Carneigie Mellon University (1997)
12. Si, H.: An analysis of Shewchuk's Delaunay refinement algorithm. In: Proceedings of the 18th International Meshing Roundtable, Salt Lake City, UT, USA, pp. 499–518 (2009)
13. Ito, Y., Shih, A.M., Erukala, A.K., Soni, B.K., Chernikov, A., Chrisochoides, N., Nakahashi, K.: Parallel unstructured mesh generation by an advancing front method. Mathematics and Computers in Simulation 75, 200–209 (2007)
14. Said, R., Weatherill, N.P., Morgan, K., Verhoeven, N.A.: Distributed parallel Delaunay mesh generation. Computer Methods in Applied. Mechanic Engineering 177, 109–125 (1999)

15. Freitag, L., Jones, M., Plassmann, P.: A Parallel Algorithm for Mesh Smoothing. SIAM Journal on Scientific Computing 20, 2023–2040 (1999)
16. METIS - Serial graph partitioning and fill-reducing matrix ordering (June 08, 2013), http://glaros.dtc.umn.edu/gkhome/metis/metis/overview
17. Karypis, G., Kumar, V.: Multilevel algorithms for multi-constraint graph partitioning. In: Proceedings of the 1998 ACM/IEEE Conference on Supercomputing, San Jose, CA, USA, pp. 1–13 (1998)
18. Linardakis, L., Chrisochoides, N.: A static geometric medial axis domain decomposition in 2D Euclidean space. ACM Transactions on Mathematical Software 34, 1–28 (2008)
19. Zheng, Y., Weatherill, N.P., Edward, A.T.S.: Interactive geometry utility environment for multi-disciplinary computational engineering. International Journal for Numerical Methods in Engineering 53, 1277–1299 (2002)
20. Xie, L., Chen, J., Liang, Y., Zheng, Y.: Geometry-based adaptive mesh generation for continuous and discrete parametric surfaces. Journal of Information & Computational Science 9, 2327–2344 (2012)
21. Larwood, B.G., Weatherill, N.P., Hassan, O., Morgan, K.: Domain decomposition approach for parallel unstructured mesh generation. International Journal for Numerical Methods in Engineering 58, 177–188 (2003)
22. Peirò, J.: Surface grid generation. In: Thompson, J.F., Soni, B.K., Weatherill, N.P. (eds.) CRC Handbook of Grid Generation, ch. 19. CRC Press, Inc., Boca Raton (1999)
23. Tremel, U., Deister, F., Hassan, O., Weatherill, N.P.: Automatic unstructured surface mesh generation for complex configuration. International Journal for Numerical Methods in Fluids 45, 341–364 (2004)
24. NASA DLR-F6 Geometry (June 08, 2013), http://aaac.larc.nasa.gov/tsab/cfdlarc/aiaa-dpw/Workshop2/DLR-F6-geom.html
25. Xie, L., Zheng, Y., Chen, J., Zou, J.: Enabling technologies in the problem solving environment HEDP. Communications in Computational Physics 4, 1170–1193 (2008)
26. Chen, J., Zhao, D., Huang, Z., Zheng, Y., Gao, S.: Three-dimensional constrained boundary recovery with an enhanced Steiner point suppression procedure. Computers and Structures 89, 455–466 (2011)
27. Ollivier-Gooch, C.: GRUMMP (June 8, 2013), http://tetra.mech.ubc.ca/GRUMMP/
28. Freitag, L.A., Ollivier-Gooch, C.: Tetrahedral mesh improvement using swapping and smoothing. International Journal for Numerical Methods in Engineering 40, 3979–4002 (1997)
29. Liu, J., Chen, B., Sun, S.: Small polyhedron reconnection for mesh improvement and its implementation based on advancing front technique. International Journal for Numerical Methods in Engineering 79, 1004–1018 (2009)
30. Klingner, B.: Improving Tetrahedral Meshes. PhD Thesis, University of California (2008)
31. Klingner, B.M., Shewchuk, J.R.: Stellar: A tetrahedral mesh improvement program (June 8, 2013), http://www.cs.berkeley.edu/~jrs/stellar