CCF Transactions on High Performance Computing (2020) 2:149–163
https://doi.org/10.1007/s42514-020-00024-x

REGULAR PAPER

Parallel and automatic isotropic tetrahedral mesh generation of misaligned assemblies

Peng Zheng[1,2] · Yang Yang[2] · Zhiwei Liu[3] · Quan Xu[2] · Junji Wang[3] · Juelin Leng[2] · Tiantian Liu[2] · Zhaoxu Zhu[3] · Jianjun Chen[3]

Received: 29 November 2019 / Accepted: 14 February 2020 / Published online: 27 February 2020

Abstract
Mesh generation is a challenge for high-performance numerical simulation, one reason is the complex geometry representing solution domain makes pre-processing difficult, especially for those assembly model containing hundreds and thousands of components involving misaligned interfaces between neighboring parts, and no state-of-art meshing tools could provide automatic functions for processing such complex model, another reason is hundreds of millions or even billions meshes should be generated quickly, which also exceeds the capabilities of available tools. In this paper, a novel parallel and automatic mesh generation method is proposed. Firstly, a surface imprinting algorithm based on the hybrid representation of discrete and continuous surfaces is proposed to process misaligned assembly model automatically. Then, the repaired assembly model is used as an input for a carefully designed mesh generation pipeline which connects the procedures of mesh sizing control, and three-level parallel tetrahedral mesh generation in order. This proposed method could produce hundreds of millions consistent mesh qualified for high-performance numerical simulation based on thousands of geometry components. Numerical experiments on a giant dam model and an integrated circuit board model demonstrates the effectiveness of this method.

Keywords Mesh generation · Assembly model · Surface imprint · Hybrid representation · Parallel mesh generation

1 Introduction

To facilitate a numerical simulation, one needs first to prepare a geometry model and then discretize the model into a mesh. For a simulation involving complex geometry configurations and/or complex physics, mesh generation is usually the major performance bottleneck. With the rapid development of high-performance computing (HPC) technologies, various simulation codes have been parallelized to exploit their potentials of running efficiently on increasingly powerful parallel computing machines. However, the clock time to finish a complex simulation is much longer than that expected by the end-user. A significant percentage of this

✉ Peng Zheng
eliza_zheng@126.com

✉ Jianjun Chen
chenjj@zju.edu.cn

Yang Yang
yang_yang_rj@iapcm.ac.cn

Zhiwei Liu
zhvliu@zju.edu.cn

Quan Xu
jluxuquan@126.com

Junji Wang
frankking@zju.edu.cn

Juelin Leng
leng_juelin@iapcm.ac.cn

Tiantian Liu
liu_tiantian@iapcm.ac.cn

Zhaoxu Zhu
zhuzx@zju.edu.cn

[1] Institute of Computer Application, CAEP, Mianyang 621900, Sichuan, China

[2] CAEP Software Center for High Performance Numerical Simulation, Beijing 100088, China

[3] Center for Engineering and Scientific Computation and School of Aeronautics and Astronautics, Zhejiang University, Hangzhou 310027, Zhejiang, China

time is consumed by the mesh generation step owing to the following two facts.

Firstly, mesh generation is still a labor-intensive and time-consuming step. For instance, it takes weeks or more to prepare a block-structured mesh for an external flow simulation over a complete aircraft model, even by an engineer with expertise in applying state-of-the-art meshing tools (Baker 2005). Unstructured mesh generation does not require a painful block-decomposition process, but preparing a suitable input for the unstructured meshing pipeline remains a major performance bottleneck in many cases (Shimada 2011; Chen et al. 2015, 2017), in general, this input contains a geometry that defines the meshing domain and a sizing control that defines the distribution of element scales over the meshing domain (Deister et al. 2004; Kania and Pirzadeh 2005; Pirzadeh 2010; Quadros et al. 2010). It remains a very active research topic to automate the preparation of this input.

Secondly, the sequential run of a mesh generator is time-consuming when large-scale meshes are considered. It was reported that the process of generating an unstructured mesh in excess of hundreds of millions elements could consume about 1.5 days if being executed sequentially (Löhner 2014). In practice, when the input geometry is becoming very complicated, or the resultant mesh quality is set at a high standard, mesh generation is often becoming a trial-and-error process. Its execution possibly needs to be repeated several times before an ideal mesh is obtained. Therefore, the clock time for preparing a large-scale unstructured mesh is usually comparable with or more than the time consumed for conducting parallel simulations.

As a result, mesh generation has become a prominent factor that determines the efficiency and scalability of parallel simulations (Weatherill et al. 2002; Freitas et al. 2013; Löhner 2013, 2014; Yilmaz and Ozturan 2015; Laug et al. 2017; Chen et al. 2017). Automating and parallelizing the mesh generation pipeline becomes a long-standing goal pursued by both the mesh generation and HPC communities (Zagaris et al. 2009; Loseille et al. 2015). At present, the implementation of this goal for any geometry input and any type of mesh outputs is not realistic due to various difficulties. However, it is likely to implement this goal for specific geometry inputs and specific types of mesh outputs, for instance, creating a qualified unstructured mesh in parallel for misaligned assemblies. Here, an assembly refers to a representation of products containing many components. In the past years, a strong interest of numerical simulation studies has been switched from simulations of single component towards simulations of products at assembly level (Chrisochoides 2016). However, by using existing mesh generation tools, the setting of a qualified mesh model on complex assemblies cannot be usually handled within the prescribed time frame. In this study, our focus is on how to overcome this bottleneck by developing a set of automatic and parallel algorithms to set up an efficiently parallel and fully automatic meshing pipeline.

The first algorithm regards how to remove multiple representations on shared interfaces of neighboring components. A solution is to merge these multiple representations into a single one by surface imprinting. Depending on how the surfaces are depicted, existing surface imprinting algorithms can be categorized into two groups: those applied directly to the continuous model (Sheffer 2000; Sheffer et al. 2000; Inoue et al. 2001; Foucault et al. 2008; Dannenhoffer and Haimes 2003) and those to the discrete model (Patel et al. 2006; Patel and Marcum 2008; Quadros and Owen 2009; Smith et al. 2010). One contribution of this study is the development of a novel surface imprinting algorithm based on a new data structure namely the hybrid surface B-rep. In the hybrid surface B-rep, each topology entity has a dual geometry representation in default. In addition, the mappings are maintained between the topology entities of the B-rep and their counterparts on the discrete model. Compared with the algorithms solely based on the discrete or the continuous models, the developed surface imprinting algorithm could keep the strengths of those solely based on continuous or discrete surfaces but avoid their weaknesses at the same time. On the one hand, most geometry computations are performed on the discrete model, avoiding the tedious, expensive and unreliable calculations on the continuous model. On the other hand, the input continuous model is not modified, and the output mesh is loyal to the input model rather than a discrete one or a modified one. This feature is often much desirable in simulations that require meshes with high geometry fidelity. The second algorithm is developed for parallel mesh generation. In general, the existing parallel mesh generation approaches could be classified into the algorithm-parallel ones and the problem-parallel ones (Cougny and Shephard 1999; Chrisochoides 2006; Foteinos and Chrisochoides 2011; Chen et al. 2017, 2018). Presently, the problem-parallel approaches are preferred in many studies because of their better capability of reusing state-of-the-art sequential mesh generation codes (Chrisochoides and Nave 2003; Chen et al. 2012). Domain decomposition is an essential step of the problem-parallel approaches. It subdivides a problem domain into many subdomains such that the following meshing procedure could be conducted on these subdomains in parallel. The performance of a parallel mesh generation algorithm is thus highly dependent on its domain decomposition approach. The existing domain decomposition techniques are usually dependent on complex geometric computations to split the domain into subdomains artificially (Larwood et al. 2005). An overuse of these techniques to achieve parallelism can introduce issues such as mesh quality, and the reliability, scalability and efficiency of the entire algorithm. In this study, we attempt to develop a three-level

parallel mesh generation algorithm to relieve the issues on existing algorithms. The first level of parallelism sources from the B-rep of assemblies, in which the meshing task of each surface and each volume is considered as an independent meshing task. The second level of parallelism depends on existing domain decomposition techniques, which split the big volume into smaller ones so that no single meshing task can dominate the parallel meshing procedure in terms of computing time. The third level of parallelism is from the employed Delaunay tetrahedralization algorithm. A multi-threaded version of the algorithm is developed by using OpenMP and incorporated into the original MPI version of the parallel algorithm. Evidently, this MPI/OpenMP hybrid implementation could better adapt to the architecture of present parallel machines, in which each computer node contains many computing cores connected by shared memory while computer nodes themselves are connected by high-speed networks.

The above algorithms have enabled us to build up a parallel and automatic mesh generation pipeline for misaligned assemblies. Numerical experiments will be presented to demonstrate that the developed approach is robust and applicable to geometry models of a complicated level experienced in industry.

Our paper is organized as follows: Sect. 2 introduces the overview of the algorithms; Sect. 3 describes the hybrid surface B-rep for geometry representation; Sect. 4 proposes surface imprinting algorithm for misaligned assemblies; Sect. 5 discusses the automatic sizing control for mesh generation; Sect. 6 describes three-level parallel mesh generation; Sect. 7 presents our experimental results; Sect. 8 concludes this work.

2 Overview of the algorithm

Figure 1 shows example of some misaligned components, and inconsistent mesh generated from it.

Figure 2 presents the workflow of the proposed algorithm. It inputs an assembly geometry model with misaligned interfaces and outputs a volume meshes that fills in the domain bounded by the model surfaces. Basically, the following steps are involved in the Algorithm.

Step 1: Initializing the hybrid surface B-rep In this step, the input CAD geometry is tessellated and a hybrid surface B-rep data structure is created to record both the continuous and discrete representations of the geometry and the connections between both representations.

Step 2: Imprinting misaligned surfaces At first, the misaligned discrete model is imprinted by introducing a set of intersection and subdivision operations of discrete entities. After that, the B-rep of the continuous model is repaired at the aid of the connections between the continuous and discrete models.

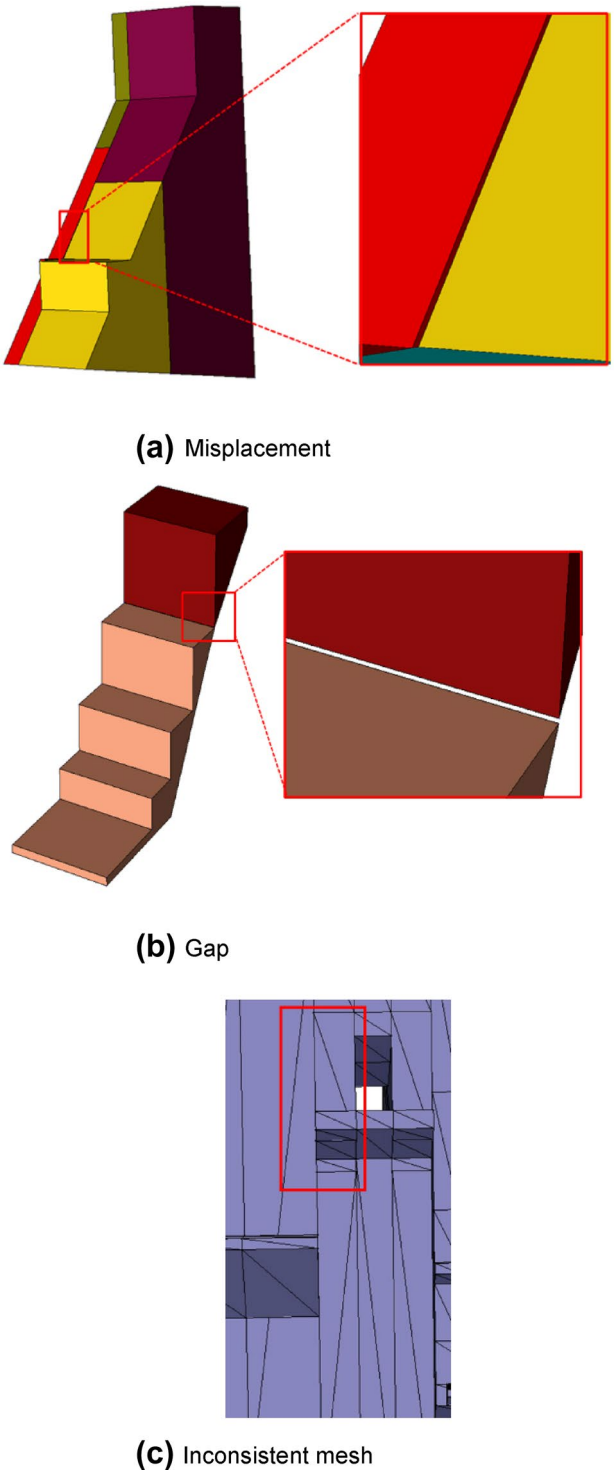


Fig. 1 Misaligned assemblies and inconsistent mesh generated. **a** Misplacement, **b** gap, **c** inconsistent mesh

Step 3: Computing the sizing function A background mesh enclosing the problem domain is created and the sizing

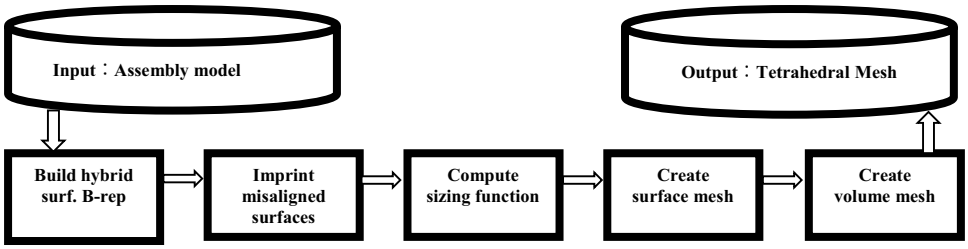


Fig. 2 The workflow of the proposed algorithm

values at background mesh nodes are initialized by considering both geometry factors and user parameters. To ensure a well-graded sizing function, the initial sizing function is smoothed by solving a non-linear programming problem.

Step 4: Creating the mesh A surface mesh is first created on the repaired geometry under the guidance of the smoothed sizing function. The employed surface mesher is based on the advancing front technique (AFT). After that, an unstructured volume mesh is created by employing a Delaunay-triangulation (DT) based tetrahedral mesher. To ensure the entire mesh generation workflow can process the task of creating a mesh containing hundreds of millions of elements or more, this workflow has been efficiently parallelized.

3 Hybrid surface B-rep

3.1 Definition of the new data structure

3.1.1 Surface B-rep

The surface B-rep is a subset of the solid B-rep and it includes three basic topology entities: face, curve and point, as illustrated in Fig. 3. Meanwhile, a specific topology entity named loop is used to limit the valid region of a face. Internally, a loop refers to a set of boundary curves and is a group entity that distinguishes with other topology entities.

3.1.2 The Hybrid surface B-rep

The hybrid surface B-rep is an enhanced B-rep data structure, in which each topology entity has two types of geometric representations in default. One representation corresponds to geometry entities defined on the continuous model such as *face*, *curve* and *point*, while the other representation describes the discrete duals of these continuous entities, such as the triangular *facets* filling in a *face*, the linear *edges* connected end-to-end within a *curve* and a mesh *vertex* exactly located at a geometry point. Therefore, the standard surface B-rep is enhanced by adding two new data structures. One data structure represents the discrete model, for instance by the half-edge structure; while the other data structure represents the connections between the continuous and discrete models. Conceptually, the connections can be reduced to three basic mappings as below.

The face-facet mapping A face corresponds to a set of facets.

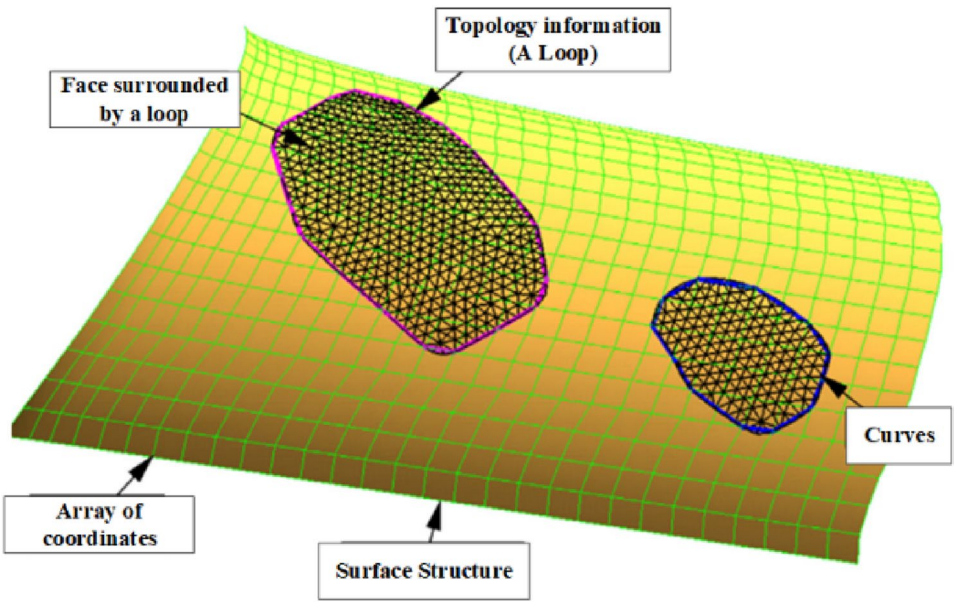


Fig. 3 Illustration for the surface B-rep

The curve-edge mapping A curve corresponds to a set of edges.

The point-vertex mapping A point corresponds to a vertex.

Two definitions are introduced below to describe the above mappings:

Definition 1 (Classification) Given a d_i-dimensional topology entity (d_i=0~2) M^{d_i} of the discrete model, M^{d_i} is *classified* on a d_j-dimensional topology entity ($d_i \leq d_j \leq 2$) G^{d_j} of the B-rep if M^{d_i} lies on G^{d_j}, denoted as $M^{d_i} \subseteq G^{d_j}$.

Definition 2 (Reverse Classification Set, RCS) Given a d-dimensional topology entity (d=0~2) G^d of the B-rep, the d-dimensional topology entities of the discrete model classified on G^d form a *reverse classification set*, denoted as

$$RCS(G^d) = \{M^d | M^d \subseteq G^d\} \tag{1}$$

Other mappings can be defined as well, e.g., between a curve and all vertices that lie on the curve, or between a face and all edges that bound the face. As these additional mappings can be derived from the basic mappings, they are not explicitly represented in the extended B-rep.

3.1.3 Initialization of the hybrid surface B-rep

Depending on the input model, the dual representation can be initialized in two ways. If the input geometry is a continuous model, its curves and faces are meshed into linear segments and triangles, respectively. At this stage, the triangle shapes do not matter, but the tessellated model must be geometrically close to the continuous model. If the input geometry is a discrete model, it is first subdivided into many patches, after which a continuous representation is built by parameterizing each patch. The first approach is employed in this study because a continuous input is considered.

4 Surface imprinting

4.1 Imprinting of discrete surfaces

Geometry errors in the input misaligned assembly are inherited by the discrete model after tessellation. The first step of surface imprinting attempts to detect all the local intersections between misaligned interfaces and then resolve these intersections by inserting points at the intersection positions and using these new points to subdivide the intersected entities accordingly. Table 1 summarizes the possible cases of intersections that need to be treated. Note that the facet–facet intersection is not included because this intersection could be resolved as a result of the resolution of the intersections between boundary edges of a facet and the other facet. Usually, ε_r is define as a global tolerance, which is the minimum value of 1/15 of the diagonal of the bounding box of all geometric entities.

Apart from the vertex–vertex intersections, the treatments of other four types of intersections need to consider their respective degenerate cases, for instance, the vertex coincides with the ending point of an edge or a face, the edge intersects another edge or a face at one of its ending vertex, and so on. To relieve the tedious coding efforts of treating these degenerate cases, we suggest resolving different types of intersections in the order listed in Table 1. As a result, it is ensured that the degenerate

Table 1 Resolving different types of intersections

Cases	Criteria of detection	Schemes used to remove the intersections
Vertex–vertex intersection	The distance between the vertices v_1 and v_2 is smaller than ε_r.	Merge v_1 and v_2
Vertex–edge intersection	The distance between the vertex v and the edge e is smaller than ε_r.	1. Compute the projection of v on e, namely v^* 2. Split e by v^* 3. Merge v and v^*
Vertex–facet intersection	The distance between the vertex v and the facet f is smaller than ε_r.	1. Compute the projection of v on f, namely v^* 2. Split f by v^* 3. Merge v and v^*
edge–edge intersection	The distance between the edges e_1 and e_2 is smaller than ε_r.	1. Compute the point (namely v_1) at c that is nearest to e_2 and the projection of v_1 (namely v_2) at e_2 2. Split e_1 and e_2 by v_1 and v_2, respectively 3. Merge v_1 and v_2
edge-facet intersection	The distance between the edge e and the facet f is smaller than ε_r.	1. Compute the point (namely v_1) at e that is nearest to f and the projection of v_1 (namely v_2) on f 2. Split e and f by v_1 and v_2, respectively 3. Merge v_1 and v_2

cases of one type of intersections could be resolved in the previous steps. For instance, the degenerate case of the vertex–edge intersection occurs when the vertex coincides with the ending point of an edge. Nevertheless, this case should already be treated during the step of processing vertex–vertex intersections. As a result, we could skip over this degenerate case during the step of processing vertex–edge intersections and only consider the regular case of the vertex being located at the middle of the edge.

Algorithm 1 The scheme used to resolve all possible intersections between discrete interface entities

```
#Step 1. Ensure no self-intersection for all single parts
  For each part of the assembly
    Compute the intersections between two facets: f_i and f_j
    If any intersections are found
      add <f_i , f_j> to Set P
#Step 2. Process the vertex-vertex intersections
  For all in Set P
    Collect all vertex pairs that intersect each other within tolerance,
  Sorting the pairs in the ascending order of distances
  Processing the intersected vertex pairs in order
#Step 3. Process the vertex-edge intersections
  For all in Set P
    Collect all vertex-edge pairs that intersect each other within tolerance
  Sorting the pairs in the ascending order of distances
  Processing the intersected vertex-edge pairs in order.
#Step 4. Process the vertex-facet intersections
  For all in Set P
    Collect all vertex-facet pairs that intersect each other within tolerance
  Sorting the pairs in the ascending order of distances
  Processing the intersected vertex-facet pairs in order.
#Step 5. Process the edge-edge intersections
  For all in Set P
    Collect all edge pairs that intersect each other within tolerance
  Sorting the pairs in the ascending order of distances
  Processing the intersected edges pairs in order
#Step 6. Process the edge-facet intersections
  For all in Set P
    Collect all edge-face pairs that intersect each other within tolerance
  Sorting the pairs in the ascending order of distances
  Processing the intersected edge-face pairs in order
#Step7. Process the facet-facet intersections
  For all in Set P
    Collect all modified facet and new facets generated by split
  Finding all duplicated facets by comparing their boundary edges
  Delete the duplicated one
```

Following the idea above, Algorithm 1 presents the scheme used to resolve the intersections between discrete entities. Since we only consider the intersections between misaligned interface entities at present, no edge-facet intersection cases need to be treated at all. The process of treating existing intersections occasionally introduces a few new intersections. A simple strategy to resolve this issue is to repeat the execution of Algorithm 1 for several times until all intersections are treated appropriately.

The discrete model will be repaired via three mesh operations embedded in Algorithm 1, i.e., merging two points, splitting an edge and splitting a facet. Note all these operations may introduce duplicate edges and facets and these duplicate entities should be cleared once detected. For instance, as seen in Fig. 4, the edge P_1P_3 becomes duplicate to the edge P_2P_3 once P_1 and P_2 are merged. To remove this duplication, both edges should be removed and replaced by the new edge PP_3. Consequently, the small gap defined by the angle $P_1P_3P_2$ is seamed correctly.

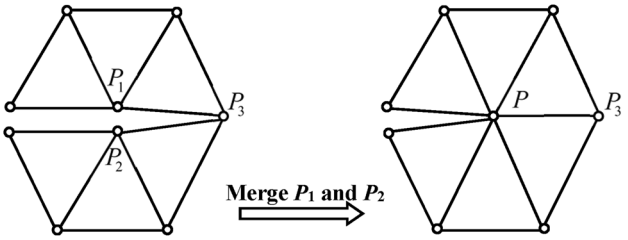


Fig. 4 Merging two points

4.2 Imprinting of continuous surfaces

In this step, we attempt to build a correct B-rep for the continuous surface model. To achieve this, we need to update the mappings between the continuous model and its discrete counterpart at the same time of executing three basic mesh operations, i.e., merging two points, splitting an edge and splitting a facet. Here, we illustrate this update scheme by using the example shown in Fig. 5, in which the new vertex P must inherit the classifications of P_1 and P_2 after executing the merging operation: if $P_1 \subseteq G_i^0$, then $P \subseteq G_i^0$; and if $P_2 \subseteq G_j^0$, then $P \subseteq G_j^0$. Meanwhile, the new edge PP_3 must inherit the classifications of P_1P_3 and P_2P_3: if $P_1P_3 \subseteq G_i^1$, then $PP_3 \subseteq G_i^1$; and if $P_2P_3 \subseteq G_j^1$, then $PP_3 \subseteq G_j^1$. Note that an edge or facet can have more than one classification at this stage.

Now we can create the missing topology of the continuous model by employing a top-down procedure, i.e., targeting faces first, followed by curves and points. The process of creating missing face topology is very similar to its counterpart of creating missing curve topology. Both processes need to first subdivide existing topology entities and then merge the resulting duplicate entities. Here, the process of creating curve topology is presented in Algorithm 2, in which a curve splitting procedure is introduced to glue adjacent faces in the B-rep. The process of creating surface topology is very similar, and no more discussions on it are presented because of the limitation of the length of the paper.

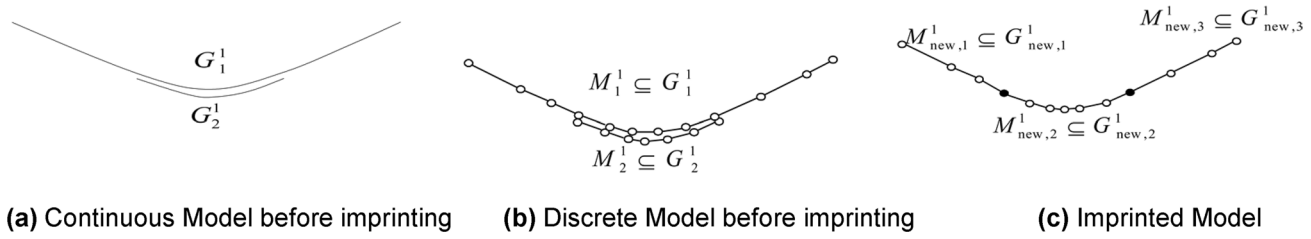


Fig. 5 Repairing the Brep. **a** Initial B-rep, **b** initial discrete model, **c** the repaired model

Algorithm 2 The curve splitting procedure that glues adjacent faces in the B-rep

1. G^1 :all curves in the B-rep $G^1 = \{G_1^1, \quad, G_n^1\}$
2. Initialize the masks of all curves to be 0
3. **while** the set of unmasked curves is not empty
4. G_i^1 : an unmasked curve in G^1
5. $M_i^1 : M_i^1 = \{M_{ij}^1 \mid M_{ij}^1 \subseteq G_i^1, j = 1, \quad, m\}$
6. **if** $m \le 1$
7. Mask G_i^1
8. **continue**
9. **end if**
10. $M_{ij,0}^0, M_{ij,1}^0$: the start and end vertices of M_{ij}^1, where $M_{ij,1}^0 = M_{ij+1,0}^0$
11. $G_{ij}^1 : G_{ij}^1 = \{G_{ijk}^1 \mid M_{ij}^1 \subseteq G_{ijk}^1, k = 1, \quad, n_{ij}\} (j = 1, \quad, m)$
12. **for** $j = 2$ **up to** m
13. **if** $G_{ij}^1 \neq G_{i1}^1$
14. **break**
15. **If** $j > m$
16. Mask G_i^1
17. **Continue**
18. **end if**
19. Create a new curve G_{new}^1: $RCS(G_{\text{new}}^1) = M_{\text{new}}^1 = \{M_{ik}^1 \mid M_{ik}^1 \subseteq G_i^1, k = 1, \quad, j-1\}$
20. Mask G_{new}^1
21. **end for**
22. **for** $k = 1$ **up to** n_{i1}
23. $M_{i1k}^1 = RCS(G_{i1k}^1)$
24. **if** $M_{i1k}^1 = M_{\text{new}}^1$
25. Replace G_{i1k}^1 with G_{new}^1 in the B-rep
26. **else if** M_{i1k}^1 is a superset of M_{new}^1
27. Split G_{i1k}^1 into 2~3 curves (one of them is G_{new}^1)
28. Replace G_{i1k}^1 with its splitting result
29. Insert new curves into G^1 with their masks initialized to be 0
30. **end if**
31. **end for**
32. **end while**

It is worth to give more explanations for Lines 24 and 27 in Algorithm 2. In Fig. 5, the curve G_2^1 is replaced by a new curve $G_{new,2}^1$ because they have identical discrete representations. However, the curve G_1^1 is split by $G_{new,2}^1$ into three curves (including $G_{new,2}^1$) because the discrete dual of $G_{new,2}^1$, i.e., $M_{new,2}^1$ overlaps with the intermediate part of the dual of G_1^1, i.e., M_1^1. Instead, if either the end vertex of $M_{new,2}^1$ is identical to an end vertex of M_1^1, G_1^1 should be split into two curves. In Line 27, the splitting results of G_{i1k}^1 may be new curves or existing curves in G^1 (G_{new}^1 is always a new curve). Here, a curve is new if and only if its discrete representation is unique in the B-rep. After creating the missing topology of surfaces and curves, the topological points of the B-rep are updated to ensure all ending points of curves are included in the B-rep.

5 Automatic sizing control

In general, the input of a mesh generator contains a geometry that defines the meshing domain and a sizing function that defines the distribution of element scales over the meshing domain. A valid geometry is now prepared after surface imprinting, and the remaining issue is how to prepare an appropriate sizing. In principle, a good sizing should define smaller element scales in the region where geometrical and physical characteristics exist and larger scales elsewhere. Moreover, the gradient of element scales must be limited so that the quality of elements in gradation regions is ensured. We use an automatic and fast mesh size specification algorithm (Deister et al. 2004) to calculate local mesh sizes, which are stored and smoothed in a Cartesian background mesh. The following steps are involved in the algorithm.

Step 1 A maximal size gradient α, minimal mesh size $s_{\min}$ and maximal mesh size $s_{\max}$ are specified by user. α should be larger than 1.0.

Step 2 Curvature and proximity features of the input geometry are calculated, and source points with respect to those features are generated. Here, curvature is computed on boundary sample points, and an approximated Euclidean distance metric is used to calculate the proximity between geometric entities, such as surfaces, curves, and vertices. Each source point p_i has local size s_i (Deister et al. 2004).

Step 3 An adaptive Cartesian background mesh is generated according to source points. On the basis of the initial uniform background mesh, a cube is divided so that it's side length is smaller than local size of all source points in it. In addition, a cube needs to be further subdivided so that the depth difference between two adjacent cubes in is less than or equal to 1.

Step 4 The size at every lattice node σ is calculated by interpolating the local feature size of the nearby source points. The mesh sizing function $F(\sigma)$ is defined as:

$$F(\sigma) = \max\left\{ \min_i \{f_i(\sigma)\}, s_{\min} \right\}$$

$$f_i(x) = \begin{cases} s_i & \|x - p_i\| \le s_i, \\ s_{\max} & else. \end{cases} \tag{2}$$

Step 5 Once the background mesh has been constructed, the mesh sizing function is smoothed to limit the gradients, so as to prevent unaccepted large gradation between mesh regions with different element size values. Supposed σ_1 and σ_2 are any adjacent lattice nodes, the smoothing function is denoted as:

$$\left|F(\sigma_1) - F(\sigma_2)\right| \leq \ln a * \left\|\sigma_1 - \sigma_2\right\| \tag{3}$$

this step needs to traverse all the lattice nodes to adjust its mesh size to meet the above smoothing condition.

During the surface mesh generation, the local size of a given point is determined by the mesh sizes stored in the adaptive Cartesian background mesh. Since an Octree-based data structure is applied, the Cartesian cube which encloses the given point can be identified efficiently. Then, the local size at the given point is interpolated linearly using the mesh sizes stored at the cube lattice nodes.

In Fig. 6, a fully automatic surface meshing pipeline is developed by incorporating the sizing function and tested by using a simple mechanical example.

6 Parallel mesh generation

6.1 Overview of the parallel algorithm

Figure 7 show three-level parallel mesh generation paradigm. It inputs an imprinted assembly geometry model generated previously and outputs hundreds of millions meshes. The following steps are involved in the algorithm.

Step 1: Domain decomposition by geometry model In this step, the input geometry is decomposed into sub-domains based on adjacency topology of different components and their estimated mesh size. Each sub-domain containing components and is assigned to a computing process group.

Step 2: Surface mesh generation A surface mesh is created in each computing process group based on the method of advancing front technique (AFT). The interfaces of surface mesh belonging to adjacent components, although be generated in different process, must be consistent.

Step 3: Domain decomposition by surface mesh Decompose the surface mesh into smaller sub-domains and assign them to the other process groups.

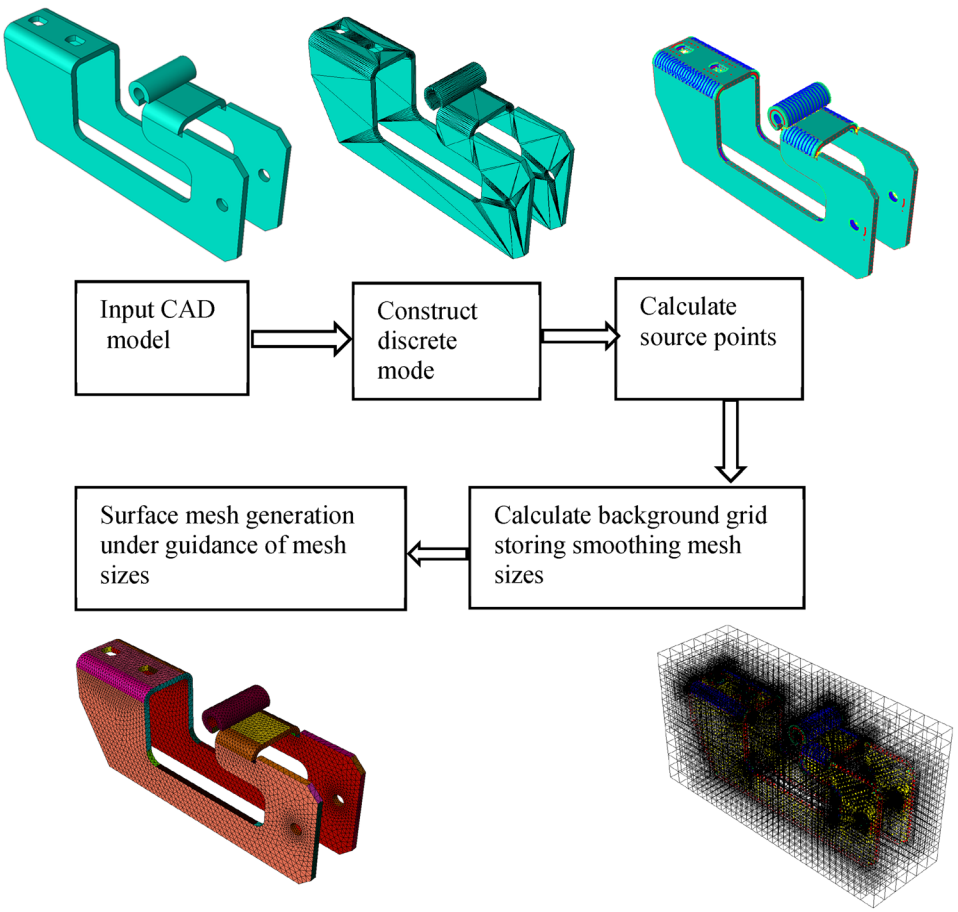


Fig. 6 Workflow of creating the proposed sizing function and surface meshing

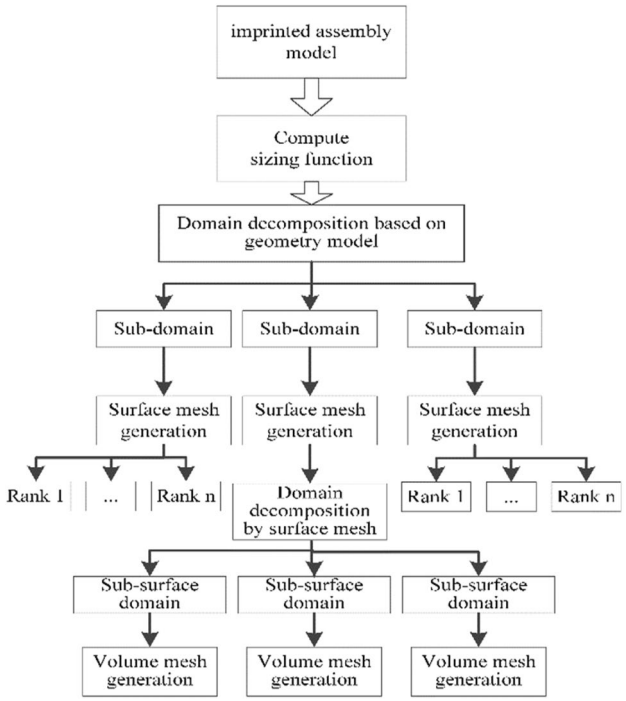


Fig. 7 Three-level parallel mesh generation paradigm

Step 4: volume mesh generation A volume mesh is generated by multi-threaded Delaunay-triangulation (DT) method in each process.

6.2 Domain decomposition based on geometry model

There are many components in assembly model. In first level of the proposed parallel method, a novel mesh scale prediction algorithm based on mesh size are proposed together with ordinary graph partition algorithm (Karypis and Kumar 1998) in domain decomposition, which ensures meshing tasks of components are distributed to different process groups as evenly as possible. When constructing the undirected graph of geometry model, the nodes of the graph are on behalf of components while the edge between nodes represent the adjacency relationship between them, the estimated mesh scale for each component is added as weighted value to node. Through this way, neighboring components are allocated to same computing node, which reduces interfaces between different domains. The process of predicating mesh scale is presented in Algorithm 3. As we know, the Cartesian background mesh are based on an Octree-data structure which storing local sizes in its nodes, these data structures are used for the spatial search of the cell containing the 3D point, which local size R_P is obtained by tri-linear interpolation of values storing in related nodes. Given the geometry model M is composed of 3D entity m_i, the predicated mesh scale of m_i is S_i, the predicated mesh scale of M is S, the computation can be defined as:

$$S = \sum_{i=1}^{n} S_i \tag{4}$$

Algorithm 3 The mesh scale of m_i predicating procedure

$S_i=0$
α :the minimum mesh size of m_i
$bounding(m_i)=(x_{low}, x_{up}, y_{low}, y_{up}, z_{low}, z_{up})$: the bounding box of m_i
$nx : nx=\left[(x_{up}-x_{low})/\alpha\right]+1$
$ny : ny=\left[(y_{up}-y_{low})/\alpha\right]+1$
$nz : nz=\left[(z_{up}-z_{low})/\alpha\right]+1$
for i = 0 up to nx
  for j = 0 up to ny
    $P_0 : (x_{low}+i*\alpha, y_{low}+j*\alpha, z_{low})$: point P_0
    $P_1 : (x_{low}+i*\alpha, y_{low}+j*\alpha, z_{up})$: point P_1
    $line : (P_0, P_1)$: construct a ray line
    P^l :all the intersection points between ray line and volume m_i. $P^l=\{P_1^l \cdots, P_n^l\}$
    if $P^l=\phi$
      Continue
    else
    for k=0 up to n-1
      for z = $(P_k^l)_z$ up to $(P_{k+1}^l)_z$
        $P(x_{low}+i*\alpha, y_{low}+j*\alpha, z)$: point P
        if P is in volume m_i
          $\beta=R_p*R_p*R_p$
          $\varphi=\alpha*\alpha*\alpha$
          $S_i=S_i+\left[\beta/\varphi\right]$
        end if
      end for
    end for
  end for
end for

6.3 Surface mesh generation

Sequential AFT surface mesher works in this way, firstly, points are generated according to vertexes of a geometry model, then, line segments on the boundary edges are generated according to points, after that, surface meshes on the boundary are generated based on the line meshes.

Parallel AFT surface mesher follows the paradigm as shown in Fig. 8. Firstly, in each sub-domain, vertexes in inner part are taken to generate inner points, on the interfaces between this sub-domain and others, only vertexes in process with small rank ID number are used to generate interface points, then the interface points are sent to other sub-domains sharing the same interface by MPI, so that each sub-domain can get a complete set of points. Secondly, in each sub-domain, points in inner part are used to generate inner line segments, only points on interfaces in the process with small rank ID number are used to generate line meshes, which are sent to other regions and then. Lastly, surface meshes in sub-domains are generated in same way mentioned above.

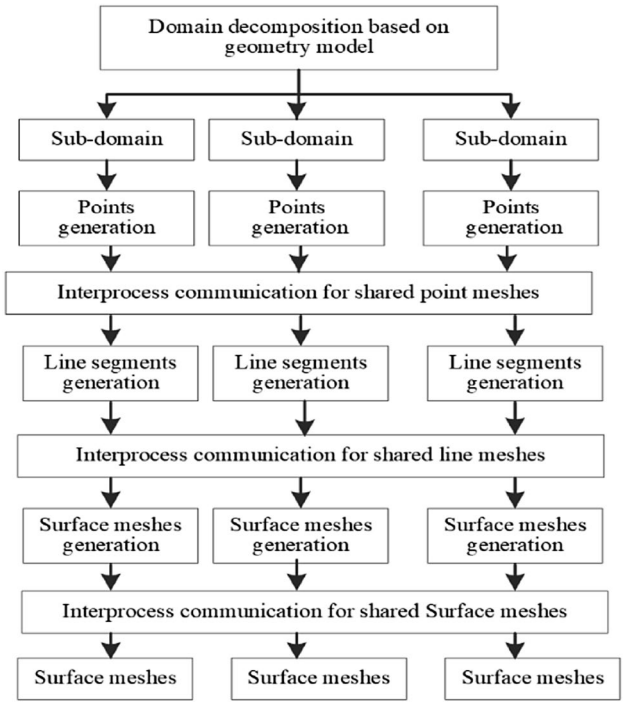


Fig. 8 Workflow of parallel surface mesh generation

6.4 Domain decomposition based on surface mesh

We further decompose the surface mesh into sub-surface domains and assign them to the other process groups to generate the volume mesh. The surface mesh decomposition is recursive (Chen et al. 2012), As shown in Fig. 9, each time the region is decomposed, an interface mesh is inserted into the region to be decomposed, and the region is divided into two sub-regions. The recursive decomposition ends only when the mesh scale of all sub-surface domains is less than the given threshold.

6.5 Volume mesh generation

We proposed a multi-threaded parallel Delaunay tetrahedral meshing algorithm based on OpenMP to generate volume mesh, and parallelism is available by changing point sets insertion style from incremental one to simultaneous one. Firstly, the algorithm performs Hilbert sort on the point sets, then it divides point sets into groups according to sorting results, and implements insertions simultaneously in each thread, during insertion, only if the Delaunay cavities of the two inserted points do not overlap, can the points be inserted, otherwise, one of the threads will give up this insertion task and places it to the tail of the queue. The details of algorithm can be referenced in related paper (Wang et al. 2018).

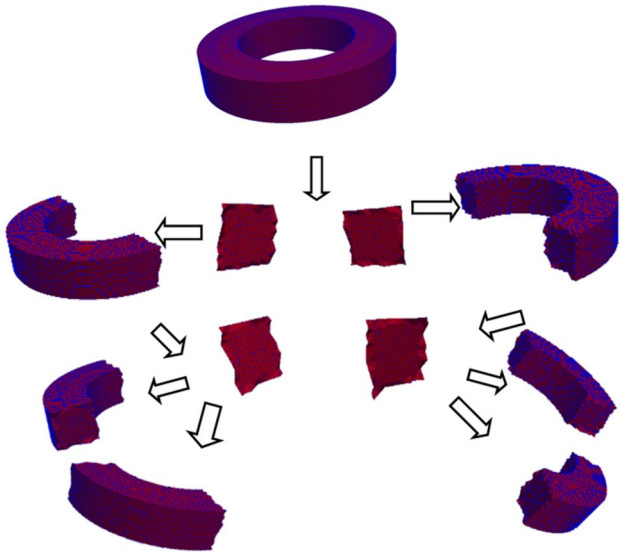


Fig. 9 Surface mesh decomposition

7 Numerical example

We uses a giant dam model to test the effectiveness of the proposed algorithm. The Sugon Pluto2018 cluster system is used as the test environment, which is equipped with Intel Xeon Gold 6132 processors, 96 GB physical memory, and each node has 28 cores. It took 6.4 min to generate 437 million mesh elements using 224 CPU cores.

As shown in Fig. 10, the giant dam model consists of 1876 geometric components, which include the left-plant part, the flood-discharge, the right-plant part, the non-overflow part, the ship lift, the ship lock, and connecting part. The model is 2309 meters long and 181 meters high.

Figure 11 shows inconsistent facet meshes between different components. The proposed imprinting algorithm based on hybrid B-rep outputs consistent facet meshes.

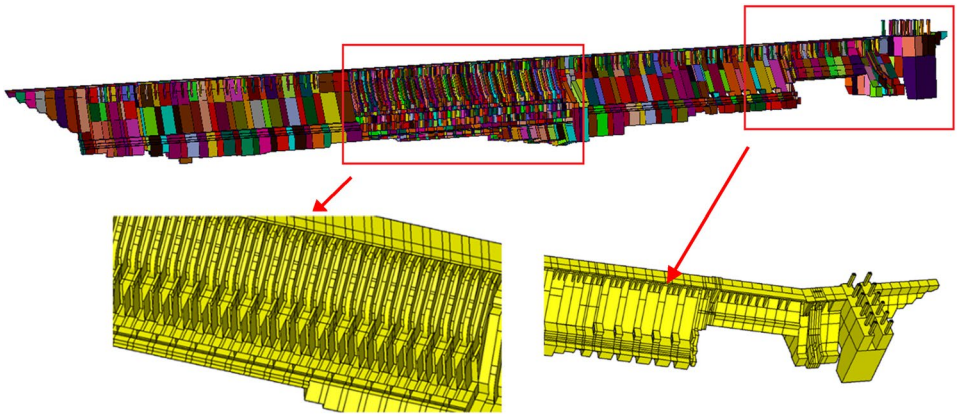
As shown in Fig. 12, a Cartesian background mesh containing mesh sizes of the model, the largest mesh size is 50.0 m and the smallest size is 2.5 m, the depth of Octree is 4, source feature points of the model count 322,977 in total.

Figure 13 shows the comparison between results of domain decomposition based solely on neighboring graph and that with weighted neighboring graph according to predicated mesh scale. It shows that when both factors are considered, the load balancing can be improved.

We also tested parallel speedup of different number of processors. Table 2 shows the result, and Fig. 14 gives a comparison with the linear one, the result shows the good scalability of this parallel mesh generation paradigm

We counted the time breakdown of all steps for parallel and automatic mesh generation, as shown in Table 3,

Fig. 10 Giant dam model



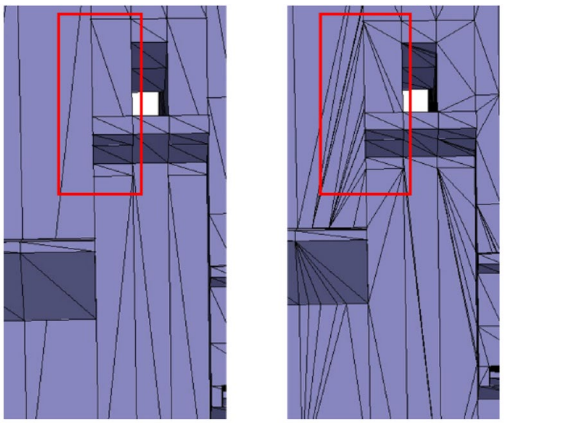


Fig. 11 Imprinted giant dam model based on hybrid B-rep, **a** before imprinting, **b** after imprinting

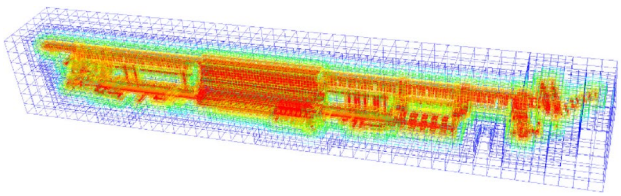


Fig. 12 Mesh sizes stored in Cartesian background mesh

the overall time was nearly 0.5 h, which was much shorter than the average period of several days for such complicated models. Since there doesn't exists any open source toolsets for such a comprehensive work in the international community or within china, it's hard to make comparison using representative toolsets, but we tried to make comparison in parallel mesh generation aspect. According to the published article (Löhner 2014), on SGI ITL supercomputer, it took 5 min for 256 cores to generate 100 million grid, our method took 6.4 min to generate 437 million mesh elements using 224 CPU cores, under the same scale, it is better than the reference data provided in the report. Someone may argue why we choose not to do tetrahedral mesh generation for different components firstly and then merge different parts together. There are two reasons. First, in order to keep mesh between adjacent parts in consistent, mesh imprinting must be done, which is more expensive in implementation. The scale of the mesh is much larger than that of discrete surface, and mesh smoothing process is inevitable because imprinting destroys smoothness of the generated mesh. Second, due to the different complexity among parts, good load balancing of processors couldn't be achieved when doing mesh imprinting in parallel. However, from the result, we can find that there is still room for improvement. First, although surface imprinting is implemented automatically, it still occupies most of the time. Second, when three-level parallel mesh generation running above 1000 processors, the speedup ratio decreases. When the number of processors increases, the time spending in the decomposition based on surface mesh is increased. There will be a main processor to decompose the surface mesh, and distribute sub-surfaces to more processors at the same time (Chen et al. 2012), which increases the proportion of sequential time and decreases the parallel speedup ratio.

The generated meshes is qualified for static analysis, and Fig. 15 shows the visualization of the simulation result.

We also uses an integrated circuit board model to test the proposed algorithm. As shown in Fig. 16, there are a lot of subtle structures in the model, such as through silicon via (TSV), chips and transmission lines, which consists of 4653 components. 102 million tetrahedral meshes are generated by the proposed algorithm, Fig. 17 shows detail view. Figure 18 shows the visualizations of simulation results based on the meshes.

8 Conclusion

We have shown a novel algorithm for parallel and automatic mesh generation of misaligned assemblies with good scalability by MPI/OpenMP hybrid implementation, in

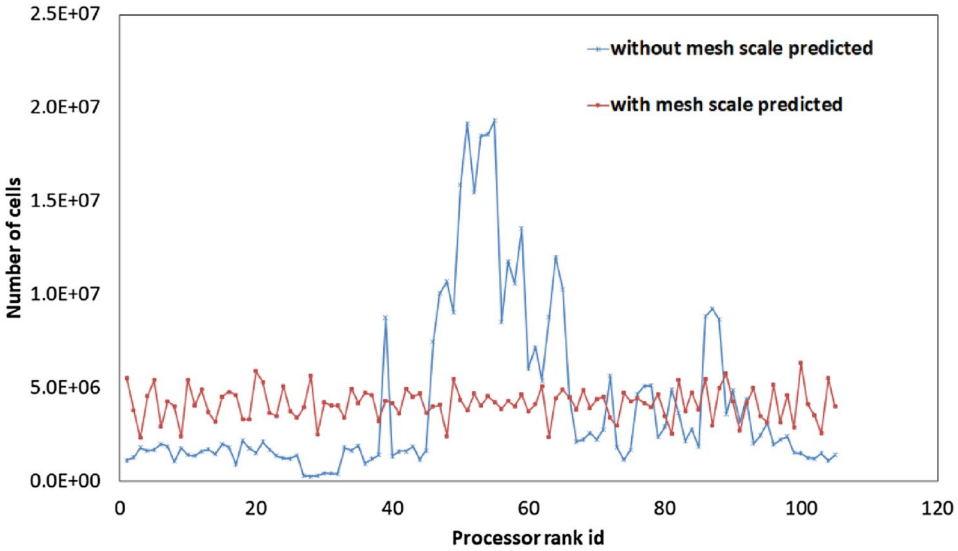


Fig. 13 Load balancing comparison between with and without mesh scale predicated

Table 2 Parallel speed up

nproc	224	448	896	1792
Speedup	224	449.89	861.72	1336.49

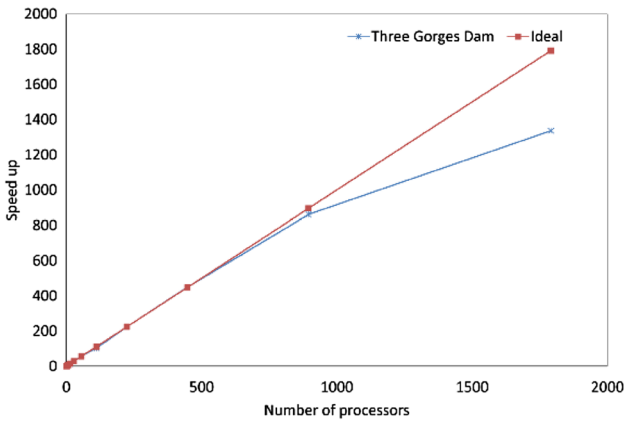


Fig. 14 Speed up of three-level parallel mesh generation

Table 3 Time breakdown of all steps in seconds

Initialize hybrid B-rep	Surface imprint	Mesh sizing	Parallel mesh generation (1792 processors)
27 s	1337 s	154 s	64.7 s
Sum			1582.7 s

which serial AFT and DT algorithm were transplanted and reused. Since future computing systems will feature increasingly more heterogeneity and hierarchy in computing units and memory systems, we believe that MPI + X(OpenMP, OpenACC, OpenCL, and CUDA) will still be the mainstream programming paradigm for future parallel and automatic mesh generation, the large legacy code in mesh generation would evolve to adapt to hardware and application changes. In the near future, in order to get higher parallel efficiency, we plan to optimize parallel surface mesh generation strategy and implement a parallelization of surface imprinting.

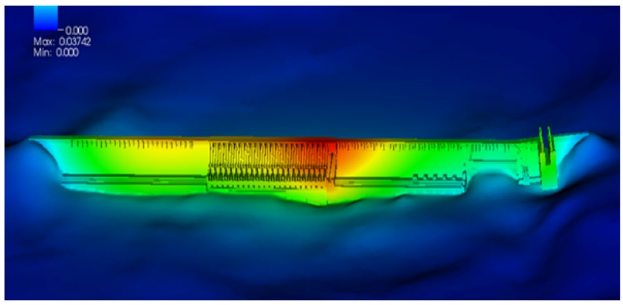


Fig. 15 Static analysis for giant dam

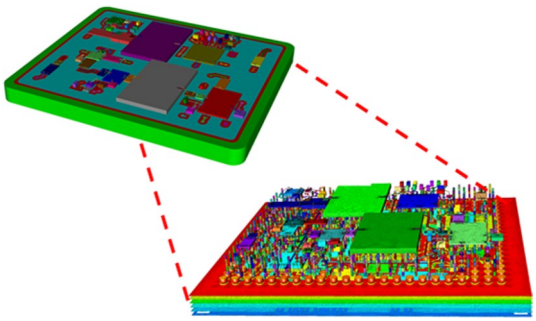


Fig. 16 Integrated circuit board model

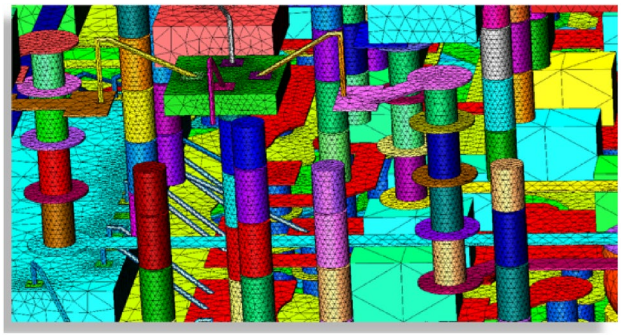


Fig. 17 Detail view of meshes

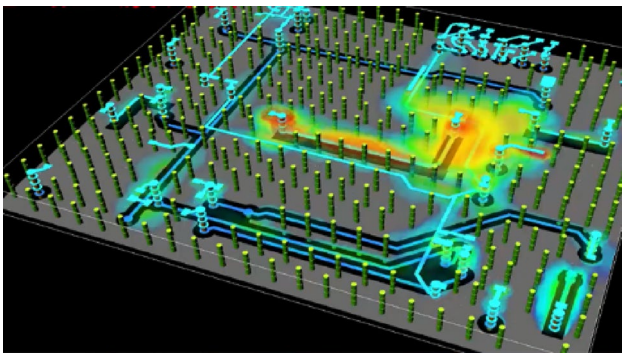


Fig. 18 Full-wave electromagnetic analysis

Acknowledgements This research was partially supported by Science Challenge Project of China (no. TZ2016002), National key R & D program of the Ministry of science and technology of China (no. 2017YFB0202203), National Natural Science Foundation of China (no. 11801037).

References

Baker, T.J.: Mesh generation: art or science? Prog. Aerosp. Sci. **41**(1), 29–63 (2005)

Chen, J., Cao, B., Zheng, Y., Xie, L., Li, C., Xiao, Z.: Automatic surface repairing, defeaturing and meshing algorithms based on an extended B-rep. Adv. Eng. Softw. **86**, 55–69 (2015)

Chen, J., Zhao, D., Huang, D., Zheng, Y., Wang, D.: Improvements in the reliability and element quality of parallel tetrahedral mesh generation. Int. J. Numer. Methods Eng. **92**, 671–693 (2012)

Chen, J., Zhao, D., Zheng, Y., Xu, Y., Li, C., Zheng, J.: Domain decomposition approach for parallel improvement of tetrahedral meshes. J. Parallel Distrib. Comput. **107**, 101–113 (2017)

Chen, J., Xiao, Z., Zheng, Y., et al.: Scalable generation of large-scale unstructured meshes by a novel domain decomposition approach. Adv. Eng. Softw. **121**, 131–146 (2018)

Chrisochoides, N.: Parallel mesh generation. In: Bruaset, A.M., Tveito, A. (eds.) Numerical Solution of Partial Differential Equations on Parallel Computers, Lecture Notes in Computational Science and Engineering, vol. 51, pp. 237–264. Springer, Heidelberg (2006)

Chrisochoides, N.: Telescopic approach for extreme-scale parallel mesh generation for CFD applications. AIAA (2016). https://doi.org/10.2514/6.2016-3181

Chrisochoides, N., Nave, D.: Parallel Delaunay mesh generation kernel. Int. J. Numer. Methods Eng. **58**, 161–176 (2003)

Cougny, H.L., Shephard, M.S.: Parallel refinement and coarsening of tetrahedral meshes. Int. J. Numer. Methods Eng. **46**, 1101–1125 (1999)

Dannenhoffer, J., Haimes, R.: Quilts: a technique for improving boundary representations for CFD. AIAA (2003). https://doi.org/10.2514/6.2003-4131

Deister, F., Udo, T., Oubay, H., Nigel, P.W.: Fully automatic and fast mesh size specification for unstructured mesh generation. Eng. Comput. **20**(3), 237–248 (2004)

Foteinos, P., Chrisochoides, N.: Dynamic parallel 3D Delaunay triangulation. In: Quadros, W.R. (eds.) Proceedings of the 20th International Meshing Roundtable. Springer, Heidelberg (2011)

Foucault, G., Cuillière, J., François, V., Léon, J., Maranzana, R.: Adaptation of CAD model topology for finite element analysis. Comput. Aided Des. **40**, 176–196 (2008)

Freitas, M., Wawrzynek, P., Cavalcante-Neto, J., Vidal, C., Martha, L., Ingraffea, A.: A distributed-memory parallel technique for two-dimensional mesh generation for arbitrary domains. Adv. Eng. Softw. **59**, 38–52 (2013)

Inoue, K., Itoh, T., Yamada, A., Furuhata, T., Shimada, K.: Face clustering of a large-scale CAD model for surface mesh generation. Comput. Aided Des. **33**(3), 251–261 (2001)

Kania, L., Pirzadeh, S.: Geometrically-derived background function for automated unstructured mesh generation. AIAA (2005). https://doi.org/10.2514/6.2005-5240

Karypis, G., Kumar, V.: Multilevel k-way partitioning scheme for irregular graphs. J. Parallel Distrib. Comput. **48**(1), 96–129 (1998)

Larwood, B.G., Weatherill, N.P., Hassan, O., Morganx, K.: Domain decomposition approach for parallel unstructured mesh generation. Int. J. Numer. Methods Eng. **58**(2), 177–188 (2005)

Laug, P., Guibault, F., Borouchaki, H.: Parallel meshing of surfaces represented by collections of connected regions. Adv. Eng. Softw. **103**, 13–20 (2017)

Löhner, R.: A 2nd generation parallel advancing front grid generator. In: Jiao, X., Weill, J.C. (eds.) Proceedings of the 21st International Meshing Roundtable. Springer, Heidelberg (2013)

Löhner, R.: Recent advances in parallel advancing front grid generation. Arch. Comput. Methods Eng. **21**(2), 127–140 (2014)

Loseille, A., Menier, V., Alauzet, F.: Parallel generation of large-size adapted meshes. Procedia Eng. **124**, 57–69 (2015)

Patel, P.S., Marcum, D.L., Remotigue, M.G.: Automatic CAD model topology generation. Int. J. Numer. Methods Fluids. **52**(8), 823–841 (2006)

Patel, P.S., Marcum, D.L.: Robust and efficient CAD topology generation using adaptive tolerances. Int. J. Numer. Methods Eng. **75**, 355–378 (2008)

Pirzadeh, S.Z.: Advanced unstructured grid generation for complex aerodynamic applications. AIAA J. **48**(5), 904–915 (2010)

Quadros, W.R., Owen, S.J.: Defeaturing CAD models using a geometry-based size field and facet-based reduction operators. Eng. Comput. **28**, 301–318 (2009)

Quadros, W.R., Vyas, V., Brewer, M., Owen, S.J., Shimada, K.: A computational framework for automating generation of sizing function in assembly meshing via disconnected skeletons. Eng. Comput. **26**(3), 231–247 (2010)

Sheffer, A.: Model simplification for meshing using face clustering. Comput. Aided Des. **33**(13), 925–934 (2000)

Sheffer, A., Bercovier, M., Blacker, T., Clements, J.: Virtual topology operators for meshing. Int. J. Comput. Geom. Appl. **10**(03), 309–331 (2000)

Shimada, K.: Current issues and trends in meshing and geometric processing for computational engineering analyses. J. Comput. Inf. Sci. Eng. **11**, 1530–9827 (2011)

Smith, B.M., Tautges, T.J., Wilson, P.P.H.: Sealing faceted surfaces to achieve watertight CAD models. In: Shontz,

S. (eds.) Proceedings of the 19th International Meshing Roundtable. Springer, Heidelberg (2010)
Wang, J., Zhu, C., Chen, J., Zheng, P., Xu, Q.: A multithreaded parallel Delaunay triangulation algorithm based on lock-free atomic operations. Comput. Eng. Sci. **40**(05), 773–779 (2018)
Weatherill, N.P., Hassan, O., Morgan, K., Jones, J.W., Larwood, B.G., Sorenson, K.: Aerospace simulations on parallel computers using unstructured grids. Int. J. Numer. Methods Fluids **40**(1–2), 171–187 (2002)
Yilmaz, Y., Ozturan, C.: Using sequential NETGEN as a component for a parallel mesh generator. Adv. Eng. Softw. **84**, 3–12 (2015)
Zagaris, G., Pirzadeh, S.Z., Chrisochoides, N.: A framework for parallel unstructured grid generation for practical aerodynamic simulations. In: Proceedings of the 47th AIAA Aerospace Sciences Meeting. AIAA-American Institute of Aeronautics and Astronautics (2009). https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20090007630.pdf

Peng Zheng Researcher, she received her Ph.D. in Beijing University of science and technology in 2006. She was chief expert of high performance numerical simulation software center (Beijing) of China Academy of Physics, and was selected into the "two hundred talents" project of China Academy of Engineering Physics. Her research interests include high-performance numerical simulation pre-processing and software development, scientific and engineering calculation visualization, virtual simulation, etc.

Yang Yang Associate Research Fellow. He received the Ph.D. degree from Peking University in 2015. He is a Development Engineer at Software Center for High Performance Numerical Simulation of China Academy of Engineering Physics. He participated in a number of projects funded by NSFC. His research interests include Geometry Processing, Computational Geometry, etc.

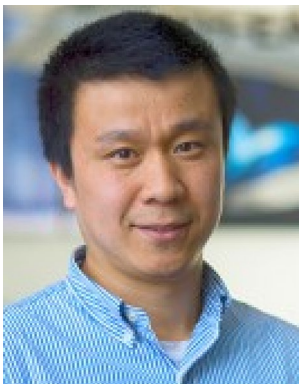
Zhiwei Liu received the B.Eng. degree in computer science and engineering from Southeast University, China. He has been studying as a Ph.D. candidate in aerospace information technology at Zhejiang University since 2015. His main research interest includes geometry repairing and mesh generation.

Quan Xu Ph.D. candidate. He received a master's degree in Computer software and theory from Graduate school of China Academy of Engineering and Physics in 2013. He is currently a Research assistant at Software Center for High Performance Numerical Simulation, CAEP and a Ph.D. candidate supervised by Professor Zeyao Mo in CAEP. His research interests include high performance computing and parallel mesh generation.

Junji Wang received the B.Eng. degree in flight vehicle design and engineering from Zhejiang University, China. He has been studying as a M.Eng. candidate in aerospace information technology at Zhejiang University since 2017. His main research interest includes parallel mesh generation.

Juelin Leng Associate Research Fellow. She received the Ph.D. degree in Computational Mathematics from University of Chinese Academy of Sciences in 2014. She is a research associate at Institute of Applied Physics and Computational Mathematics, Software Center for High Performance Numerical Simulation of China Academy of Engineering Physics. Her research interests include Mesh Generation, Computational Geometry, etc.

Tiantian Liu Assistant Research Fellow. She received the Ph.D degree in Computational Mathematics from University of Chinese Academy of Sciences in 2017. She is a Development Engineer at Software Center for High Performance Numerical Simulation of China Academy of Engineering Physics. She participated in a number of projects funded by NSFC. Her research interests include mesh generation, geometry processing, parallel computing, etc.



Zaoxu Zhu received his B. Eng. degree in mechanical engineering from China Agricultural University and M. Eng. degree in mechatronic engineering from Beihang University at Beijing, China. He received his Ph.D. degree in aircraft design from Delft University of Technology at Delft, the Netherlands. Currently he works at School of Aeronautics and Astronautics, Zhejiang University as a Postdoctoral researcher. His research domains include Aircraft Design, Parametric Modelling, Aircraft EWIS (Electrical Wiring Interconnection System) design, Multidisciplinary Design Optimization and System Engineering.

Jianjun Chen received the B.Eng. degree in civil engineering from Nanjing Institute of Civil Engineering and Architecture, China, the M.Eng. degree in computational mechanics and Ph.D. degree in computer science from Zhejiang University, China. He has been working at Zhejiang University since 2006 and is currently a full professor at School of Aeronautics and Astronautics. From 2012 to 2014, he conducted a visiting research in Swansea University, UK. His main research interests include high performance computing, mesh generation and computational engineering and science.