


RESEARCH

Open Access



# An ultralight geometry processing library for parallel mesh refinement

Bohan Wang<sup>1,2</sup>, Bo Chen<sup>3\*</sup>, Kaixin Yu<sup>1,2</sup>, Lijun Xie<sup>1,2\*</sup>  and Jianjun Chen<sup>1,2,4</sup>

\*Correspondence:  
chenbo01010401@163.com;  
LijunXie@zju.edu.cn

<sup>2</sup> School of Aeronautics and Astronautics, Zhejiang University, Hangzhou 310027, China<sup>3</sup> China Aerodynamics Research and Development Center, Mianyang 621000, China  
Full list of author information is available at the end of the article

## Abstract

In applications such as parallel mesh refinement, it remains a challenging issue to ensure the refined surface respects the original Computer-Aided Design (CAD) model accurately. In this paper, an ultralight geometry processing library is developed to resolve this issue effectively and efficiently. Here, we say the kernel is *ultralight* because it has a very small set of data-structures and algorithms by comparison with industrial-level geometry kernels. Within the library, a simplified surface boundary representation (B-rep) and a radial edge structure are developed respectively to depict the geometry model and the surface mesh, plus hash tables that record the connections between the geometry model and the surface mesh. Based on these data structures, a set of efficient algorithms are developed, which initializes the connection tables, projects a point back to the original geometry, etc. With these data-structure and algorithmic infrastructures set up, the callings of eight well-designed Application Programming Interfaces (APIs) are powerful enough to enable the parallel mesh refinement algorithm outputs a mesh respecting the input CAD model accurately. Numerical experiments will be finally presented to evaluate the performance of the overall parallel mesh refinement algorithm and the algorithms in relation with the developed library.

**Keywords:** Mesh generation, Mesh refinement, Boundary representation (B-rep), Mesh deformation, Computer-aided design (CAD)

## 1 Introduction

Mesh generation has attracted much attention since it is the major performance bottleneck of applying numerical methods to solve partial differential equations (PDEs). In fields such as computational fluid dynamics (CFD) and computational electro magnetics (CEM), large-scale meshes containing hundreds of millions of elements or more are now required to simulate some challenging problems [1–3]. Sequential approaches are usually inefficient, if not incompetent, to generate so big a mesh due to the bottlenecks in terms of memory usage and computing time. Parallel approaches have thus been developed to overcome these obstacles [1, 4–8].

Presently, the prevailing parallel approaches of mesh generation are based on domain decomposition [5, 9] or by exploiting fine-grained concurrencies within sequential mesh generation algorithms [10]. The former approaches are usually implemented in a distributed-parallel style and thus capable of avoiding memory bottlenecks in case of

large-scale mesh generation. By comparison, the later approaches are suitable for shared memory architecture and its capability is limited by the size of available memories.

Parallel refinement is another simple but powerful approach to create large-scale meshes. With a coarse mesh as input, parallel refinement subdivides each element according to templates in parallel. Commonly, large-scale meshes contain different types of elements, for instance, the hybrid prism-tetrahedra mesh for viscous simulations, in which layered prismatic elements are aligned with domain boundaries, tetrahedral elements are filled in far fields, and a few pyramids are used in the transition region. It is challenging to create a hybrid prism-tetrahedra mesh having billions of elements by domain decomposition approaches. However, much larger meshes have been demonstrated by using parallel refinement techniques [11].

One drawback of refinement-based approaches is that the new generated surface meshes will deviate from the original geometry. Reconstructing a high-order representation locally is a solution [12]; however, its accuracy depends on how accurately the initial surface mesh approximates the original geometry. Another choice is to respect the original geometry. The core issue here is how to represent the geometry and implement computations such as mapping and projection efficiently on that geometry. Zhao et al. [13] employ OpenCascade (OCC) [14], an open-source computer-aided design (CAD) kernel to represent the original geometry and reuse the algorithms provided by the kernel to accomplish the projection procedure. However, using industrial-level CAD kernels like OCC in this context is too heavy, not only because their library size is too big and their functions are too redundancy, but also because their learning curve is very high. Meanwhile, these CAD kernels are designed for general applications and its efficiency is not acceptable in critical parallel applications [13, 15].

In this study, we suggest the development of an ultralight geometry processing library (also called ultralight geometry kernel) for parallel mesh refinement. Here, we say the geometry library is *ultralight* because it has a very small set of data-structures and algorithms by comparison with industrial-level geometry kernels:

1. *Data structures.* A simplified surface boundary representation (B-rep) is used to record the topology objects. Ferguson curves and Coons surfaces, for their simplicity, are presently employed to define the curve and surface geometry [16–20]. The connections between B-rep objects (points/curves/faces) and surface mesh objects (vertices/edges/facets) are maintained to support the implementation of accurate and efficient projection algorithms [21].
2. *Algorithms.* Two key algorithms are developed: one algorithm initializes the connection between surface B-rep objects and surface mesh objects, the other one projects a point back to the original geometry. Techniques to improve the robustness and efficiency of both algorithms will be depicted.

An essential set of programming interfaces (APIs) is provided to perform the data query and key geometry algorithms. Examples show that a few callings of these APIs are powerful enough to enable the refined mesh to respect the original geometry. We will demonstrate this desirable feature by applying the kernel in a refinement-based parallel mesh generator. It is worth noting that the similar approach can be naturally

extended for applications such as high-order mesh generation [22, 23] and surface mesh adaptation [24].

The following discussion is organized as below. In Section 2, the layered structure of the developed geometry kernel is introduced. In Sections 3 and 4, the focuses are on the implementation of data structures and key algorithms, respectively. In Section 5, the set of APIs for external callings is listed and how they are used in parallel mesh refinement is demonstrated. Section 6 presents numerical studies on the proposed approach. Section 7 summarizes the article and presents a few suggestions on future work.

## 2 Layered structure of the kernel

As seen in Fig. 1, the ultralight kernel is organized into three layers:

1. *The data structures layer*, in which the geometry model, the surface mesh and their connections are represented by a simplified surface B-rep, a radial-edge structure and some hash tables.
2. *The algorithms layer*, which consists of the algorithms projecting a point to a curve and a surface, the algorithm setting up the connections between the geometry model and the surface mesh, the input/out algorithms and the basic query algorithms of the fundamental data structures.
3. *The APIs layer*, which, for the users' convenience, consists of a set of user functions that are implemented by warping or combining the algorithms implemented in the algorithms layer.

## 3 Implementations of the data structures layer

### 3.1 The surface B-rep

The surface B-rep is introduced in [21], which refers to a subset of the solid B-rep and it includes three basic topology entities: face, curve and point, as illustrated in Fig. 2.

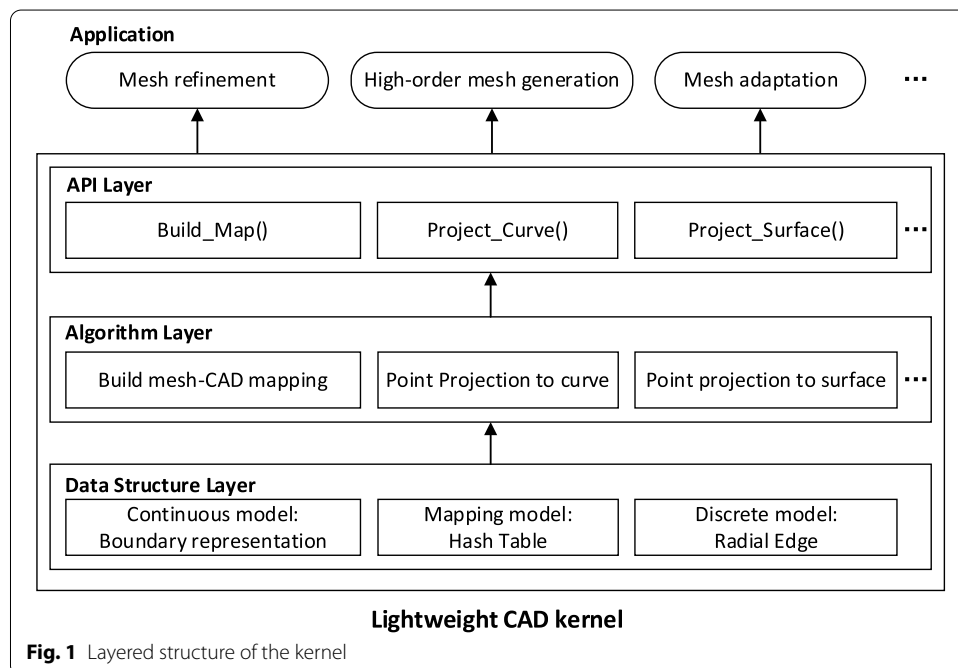
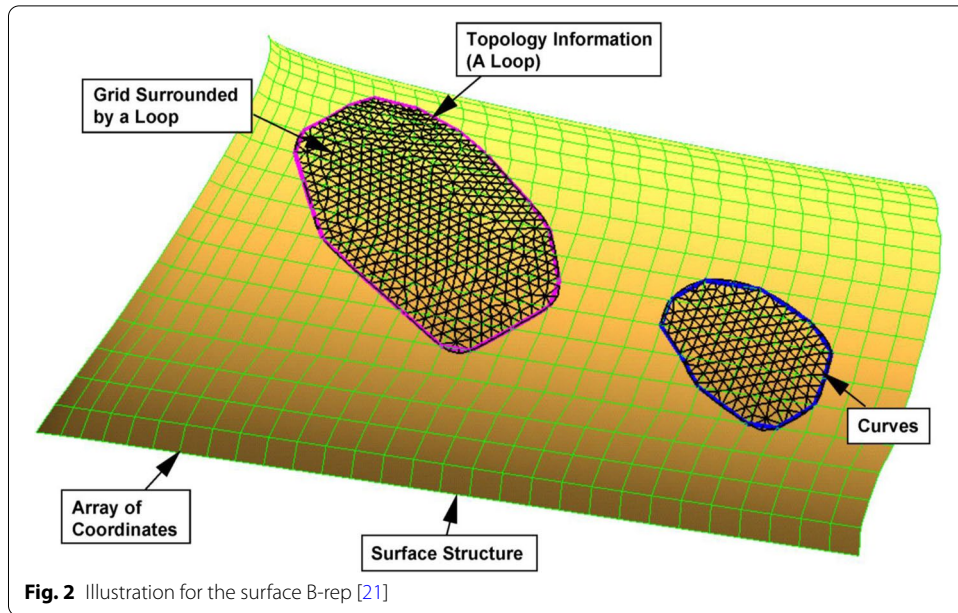


Fig. 1 Layered structure of the kernel



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 from other topology entities.

With respect to the geometric description of curves and surfaces, Ferguson curves and Coons surfaces [25, 26] are selected for their simplicity and powerful capability for geometry representation. For completeness, their analytic definitions are presented as below.

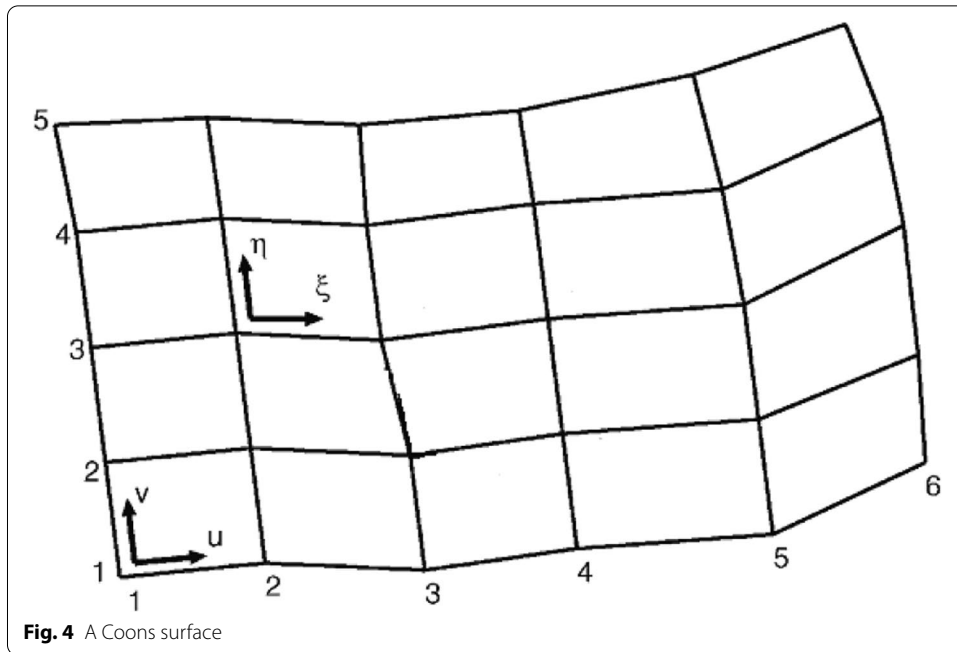
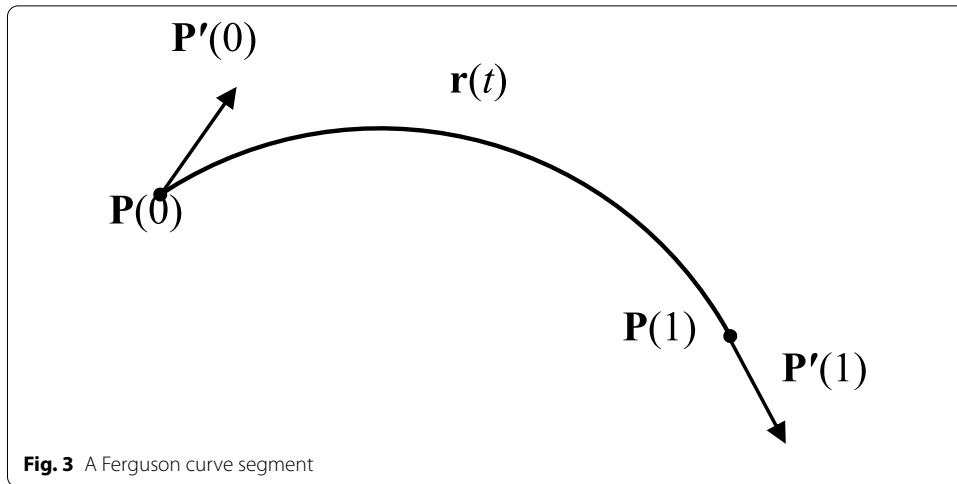
A Ferguson curve is composed of many end-to-end connected curve segments. Each segment is analytically defined as below (see Fig. 3).

$$\begin{aligned}
 \mathbf{r}(t) &= \mathbf{a}_3 t^3 + \mathbf{a}_2 t^2 + \mathbf{a}_1 t + \mathbf{a}_0 \\
 &= (2(\mathbf{P}(0) - \mathbf{P}(1)) + \mathbf{P}'(0) + \mathbf{P}'(1))t^3 \\
 &\quad + (3(\mathbf{P}(1) - \mathbf{P}(0)) - 2\mathbf{P}'(0) - \mathbf{P}'(1))t^2 \\
 &\quad + \mathbf{P}'(0)t + \mathbf{P}(0) \\
 &= \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{P}(0) \\ \mathbf{P}(1) \\ \mathbf{P}'(0) \\ \mathbf{P}'(1) \end{bmatrix},
 \end{aligned}$$

where  $\mathbf{P}(0)$  and  $\mathbf{P}(1)$  refer to starting and ending points of the segment, respectively, and  $\mathbf{P}'(0)$  and  $\mathbf{P}'(1)$  refer to tangent vectors at respective points. Given a set of interpolation points, the tangent vectors at these points could be computed by introducing two-order continuous conditions and two boundary conditions at the starting and ending points of the entire curve.

A Coons surface is composed of a matrix of surface patches, and each patch is analytically defined as below (see Fig. 4).

$$\begin{aligned}
 \mathbf{r}(u, v) &= \mathbf{F}(u)\mathbf{A}\mathbf{F}(v) \\
 &= \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \mathbf{M} \mathbf{A} \mathbf{M}^T \begin{bmatrix} v^3 & v^2 & v & 1 \end{bmatrix}^T \quad (0 \leq u \leq 1, 0 \leq v \leq 1)
 \end{aligned}$$



where

$$\mathbf{M} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \mathbf{A} = \begin{bmatrix} \mathbf{r}(0,0) & \mathbf{r}(0,1) & \mathbf{r}_\eta(0,0) & \mathbf{r}_\eta(0,1) \\ \mathbf{r}(1,0) & \mathbf{r}(1,1) & \mathbf{r}_\eta(1,0) & \mathbf{r}_\eta(1,1) \\ \mathbf{r}_\xi(0,0) & \mathbf{r}_\xi(0,1) & \mathbf{r}_{\xi\eta}(0,0) & \mathbf{r}_{\xi\eta}(0,1) \\ \mathbf{r}_\xi(1,0) & \mathbf{r}_\xi(1,1) & \mathbf{r}_{\xi\eta}(1,0) & \mathbf{r}_{\xi\eta}(1,1) \end{bmatrix},$$

and  $\mathbf{F}(u) = (F_1(u), F_2(u), F_3(u), F_4(u))$  is a vector of values computed by four Hermite interpolation functions as below,

$$\begin{cases} F_1(u) = (1 + 2u)(u - 1)^2 \\ F_2(u) = u^2(3 - 2u) \\ F_3(u) = u(1 - u)^2 \\ F_4(u) = u^2(u - 1) \end{cases}.$$

Given a matrix of interpolation points, the partial differentials defined in the matrix **A** could be computed by introducing two-order continuous conditions and four boundary conditions at the corner points of the entire surface. Interested readers are referred to [25, 26] for more details.

For simplicity, only one type of curve and one type of surface are supported in our geometry kernel, far fewer than that number supported in industry geometry kernels. For instance, in OpenCascade 9 types of curves and 11 types of surfaces are respectively supported. To use this library, it is necessary to convert other types of curves and surfaces to their Ferguson counterparts at first. The deviation error between the Coons representation and the original one can be controlled by the resolutions of sample points. Table 1 shows the deviation error between the Coons representation of a sphere with radius 50 and the analytic representation of the sphere. The error is evaluated by the distance between the sample points on Coons surface and the analytic surface. About 2 million points are uniformly sampled on the surface. Given the fact that the error is usually smaller by a few factors than the resolution of the required mesh, we say it is a reasonable compromise to use Coons surface instead of the more general but complex NURBS representation.

### 3.2 The data structure for surface mesh

Here, the data structure introduced in [27] is reused, in which the surface mesh is represented by a list of edges and a list of facets. The following codes present the data structures used to define surface edges and facets:

```
typedef struct SurfEdge {
    int nodes[2], faceH;
} SurfEdge;

typedef struct SurfFace {
    int edges[3], neigs[3];
} SurfFace;
```

To consider non-manifold cases, the number of facets adjacent to one edge is not fixed. Figure 5 explains how to link the adjacent faces of an edge cyclically, where *e.faceH* is the head of the link. For the case shown in Fig. 5, *e* is supposed to be the first edge of its adjacent faces; therefore, the first neighboring index of each face is used to point to the next face.

A frequently employed routine is the search of a facet with its three corner nodes as inputs. Its brute-force implementations need to traverse the facet list. Therefore, a hash table is created to improve the performance of this routine, where the smallest index of the corner nodes is the key value of a facet, and the facets having an identical key value form a backup list. Likely, this technique is used to speed up the routine that searches a surface edge by its corner nodes.

**Table 1** Statistics of interpolation errors

Model	#u knot	#v knot	Max sample error	Relative error
Sphere radius = 50	16	24	[-0.0747, 0.0177]	0.001494
	32	48	[-0.0188, 0.0048]	0.000376
	64	96	[-0.0047, 0.0012]	0.000094
	128	192	[-0.0011, 0.0003]	0.000022

**3.3 Classifications and reverse classifications**

Three basic mappings between their topology entities are defined as follows to connect the B-rep and the surface mesh in [21]:

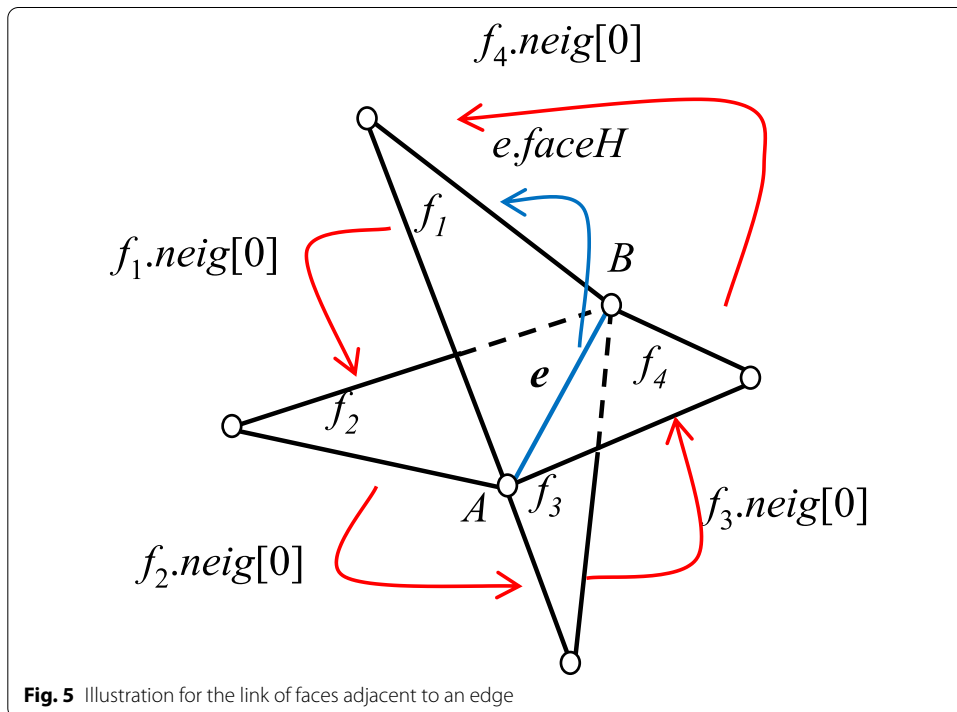
1. *The face-facet mapping.* A face corresponds to a set of facets.
2. *The curve-edge mapping.* A curve corresponds to a set of edges.
3. *The point-vertex mapping.* A point corresponds to a vertex.

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.

Two definitions are introduced below to describe the above mappings [21, 28]:

**Definition 3.1 (Classification).** Given a  $d_i$ -dimensional topology entity ( $d_i=0 \sim 2$ )  $M^{d_i}$  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  $M^{d_i} \sqsubseteq G^{d_j}$ .

**Definition 3.2 (Reverse Classification Set, RCS).** Given a  $d$ -dimensional topology entity ( $d=0 \sim 2$ )  $G^d$  of the B-rep, the  $d$ -dimensional topology entities of the discrete model classified on  $G^d$  form a reverse classification set, denoted as  $RCS(G^d) = \{M^d | M^d \sqsubseteq G^d\}$ .



**Fig. 5** Illustration for the link of faces adjacent to an edge

## 4 Key algorithms

### 4.1 Setting up connections between CAD model and surface mesh

This algorithm is employed to set up three basic mappings between a CAD model and a surface mesh. A bottom-up workflow is thus developed, which sets up the point-vertex mapping first, then the curve-edge mapping, and finally the face-facet mapping (see Fig. 6). Here we depict the respective procedures setting up these three mappings in details as below.

#### 4.1.1 Setting up point-vertex mapping

This procedure traverses all the CAD points and attempts to classify a surface mesh vertex on each point. Here, we say a vertex is *classified* on a point when their distance is smaller than a user-specified tolerance. The timing performance of a brute-force implementation is evidently unacceptable. An octree is used to speed up the computation presently.

#### 4.1.2 Setting up curve-edge mapping

This procedure traverses all the CAD curves and attempts to classify a set of surface edges on each curve. Here, we say an edge is *classified* on a curve when the distances between the ending points of the edge and the curve are all smaller than a user-specified tolerance. The timing performance of a brute-force implementation by computing the distances of each pair of a curve and a mesh vertex is unacceptable. Algorithm 1 presents the improved version of this procedure.

---

#### Algorithm 1. Search a set of edges classified on a curve

---

$c, p_s, p_e$ : a CAD curve and its starting and ending points

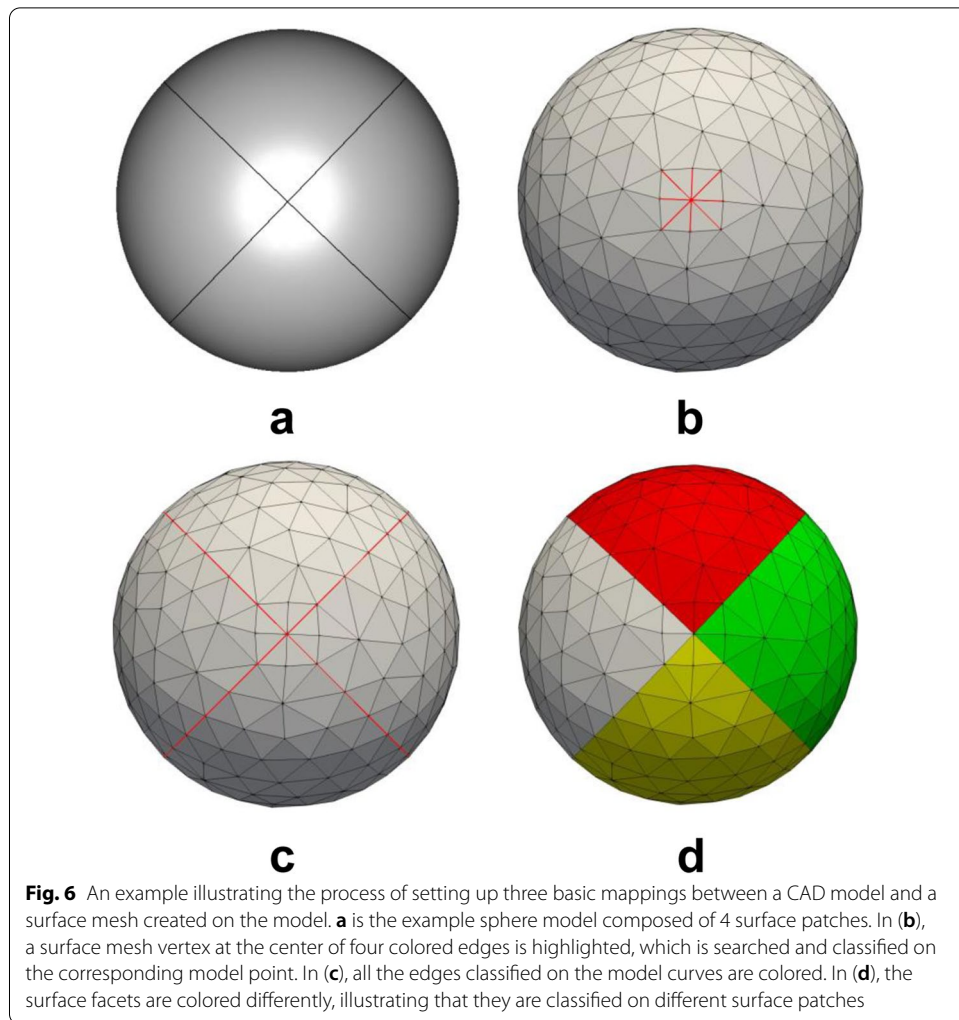
$\varepsilon$ : a user-specified tolerance

1. Find the surface vertex classified on  $p_s$  and  $p_e$ , denoted by  $v_s$  and  $v_e$
  2. Denote the active vertex by  $v_a$ , and initialize  $v_a$  to be  $v_s$
  3. **Do**
  4.   Query the set of surface edges adjacent to  $v_a$ , denoted by  $E_a$
  5.   **For** each edge  $e$  belonging to  $E_a$
  6.     Get the ending vertex of  $e$  other than  $v_a$ , denoted by  $v_o$
  7.     Compute the distance between  $c$  and  $v_o$ , denoted by  $l$
  8.     **If**  $l < \varepsilon$
  9.       Classify  $e$  on  $c$ , and set  $v_a$  to be  $v_o$
  10.    **Break**
  11.    **End if**
  12.    **End for**
  13. **While** ( $v_a \neq v_e$ )
- 

#### 4.1.3 Setting up face-facet mapping

This procedure traverses all the CAD faces and attempts to classify a set of surface facets on each face. Here, we say a facet is *classified* on a face when all the corner vertices of the





facet are classified on the face. Since the surface is trimmed by its boundary loop (see Fig. 2), we say a point is classified on a face if and only if:

1. The distance between the vertex and the supporting surface of the face is smaller than a user-specified tolerance; and
2. The vertex is located within the valid trimmed region of the face.

To investigate whether the second condition is met, a winding number algorithm is presently implemented in the parametric space of the face [29].

The timing performance of a brute-force implementation by computing the distances of each pair of a face and a mesh vertex is unacceptable. We implemented an improved version, which identifies one facet classified on the face first and then employs the coloring algorithm to search all the other facets classified on the face. Algorithms 2 and 3 present the procedure identifying the first facet and the coloring procedure, respectively.

**Algorithm 2.** Identify the first facet classified on a face

---

$f, s, C$ : a CAD face, its supporting surface and the set of boundary curves  
 $\varepsilon$ : a user-specified tolerance

1. Query all the surface edges classified on the boundary curves, denoted by  $E_b$
2. **For** each edge  $e$  belonging to  $E_b$
3.   Query the facets meeting at  $e$ , denoted by  $F_b$
4.   **For** a facet  $t$  belonging to  $F_b$
5.     Get the corner vertex of  $t$  other than two ending points of  $e$ , denoted by  $v_o$
6.     Compute the distance between  $s$  and  $v_o$ , denoted by  $l$
7.     **If**  $l < \varepsilon$  and  $v_o$  is located within the valid trimmed region of  $f$
8.       Classify  $t$  on  $f$ , and return  $t$
9.     **End if**
10.   **End for**
11. **End for**

---

**Algorithm 3.** The coloring procedure that searches all facets classified on a face

---

$f, s, C$ : a CAD face, its supporting surface and the set of boundary curves  
 $t$ : a facet classified on  $f$

1. Push  $t$  to a facet stack, and set on its test flag
2. **While** the stack is not empty
3.   Pop the top element of the stack as the base element, denoted by  $t_b$
4.   **For** each of the three neighbouring facets of  $t_b$ , denoted by  $t_n$
5.     **If**  $t_n$  is not tested and the edge shared by  $t_b$  and  $t_n$  is not classified as a boundary
6.       Classify  $t_n$  onto  $f$
7.       Push  $t_n$  to the stack and set on its test flag
8.     **End if**
9.   **End for**
10. **End while**

---

**4.2 Projecting point to curve**

Given a parametric curve  $\mathbf{r} = \mathbf{r}(u)$ , the expression of the problem becomes:

$$\min | \mathbf{r}^* - \mathbf{r}(u) |, \quad (1)$$

subjecting to

$$| \mathbf{r}^* - \mathbf{r}(u) | < \varepsilon. \quad (2)$$

Here,  $\mathbf{r}^*$  is the physical coordinates of the point,  $u$  is the parametric coordinate of  $\mathbf{r}^*$ , and  $\varepsilon$  is a distance tolerance for duplicate points. Presently, the Brent's algorithm [30] is employed to solve the above nonlinear programming (NLP) problem.

**4.3 Projecting point to surface**

Given a parametric surface  $\mathbf{r} = \mathbf{r}(u, v)$ , the expression of the problem becomes:

$$\min | \mathbf{r}^* - \mathbf{r}(u, v) |, \quad (3)$$

subjecting to

$$|\mathbf{r}^* - \mathbf{r}(u, v)| < \varepsilon. \quad (4)$$

Here,  $\mathbf{r}^*$  is the physical coordinates of the point,  $(u, v)$  are the parametric coordinates of  $\mathbf{r}^*$ , and  $\varepsilon$  is a distance tolerance for duplicate points.

Solving the above NLP problem takes the following steps [31]:

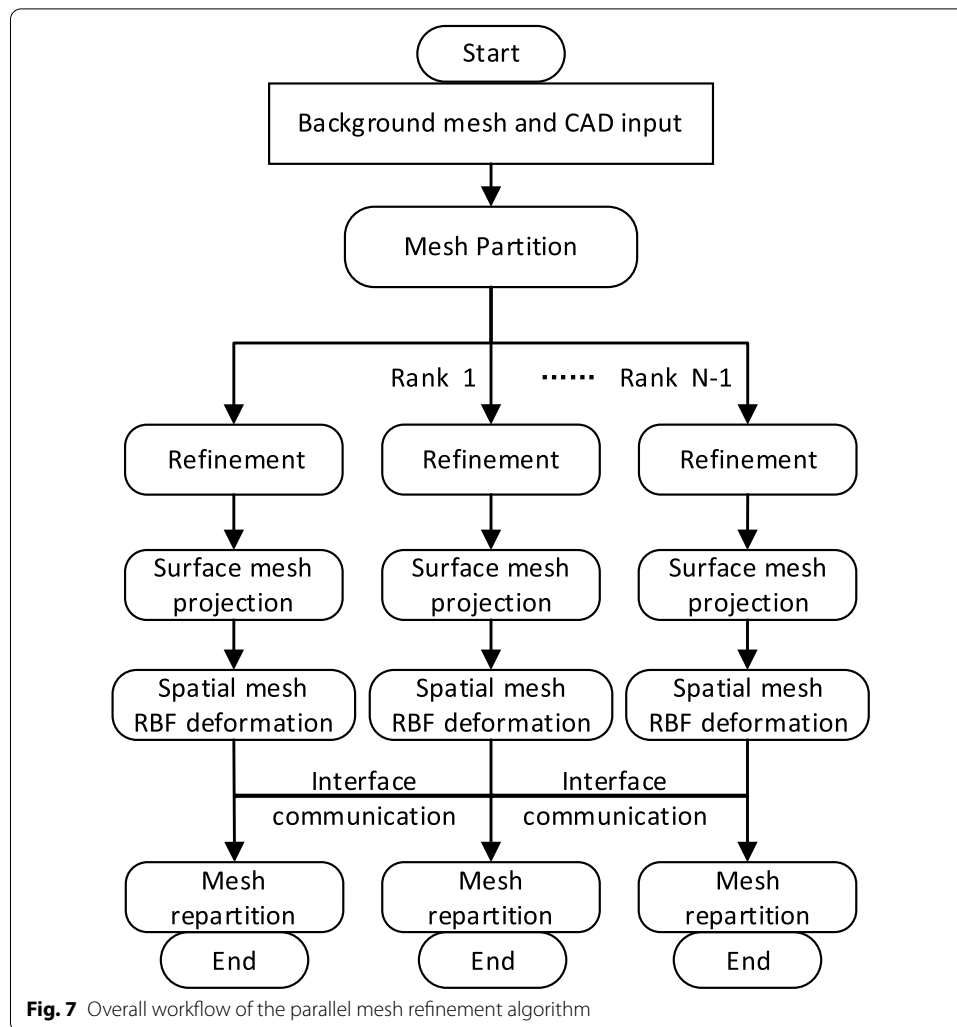
1. Check if any singular point of the surface meets Inequality (4). If no singular point meets Inequality (4), seek the solution of NLP (1) and (2) in the parametric spaces of the boundary curves as mentioned in Section 4.2.
2. If no solution is targeted in Step 1, seek the solution in the interior of the parametric space of the surface by using the GSA (Geometric Strategy Algorithm) for orthogonal projection [32].
3. If Step 2 still fails to present a solution, a direct searching procedure is executed, which is more robust but less efficient. This procedure creates a quadtree to cover the parametric space by recursive refinement: amongst existing quadrants, the one that is the closest to  $\mathbf{r}^*$  (e.g., evaluated by the distance of the centroid of the quadrant and  $\mathbf{r}^*$ ) is subdivided into four child quadrants. This refinement procedure is repeated until the centroid of a quadrant meets Inequality (4) or the size of the quadrant is smaller than the floating-point precision.

## 5 APIs and parallel mesh refinement

### 5.1 List of APIs for parallel mesh refinement

For the convenience of external callings, a few APIs are implemented in the kernel to wrap the query of data structures and the key algorithms presented in Section 4. Listed as below are 8 APIs that will be called by the parallel mesh refinement algorithm. As discussed in Section 5.2, a few callings of these APIs are powerful enough to enable the refined mesh to respect the original geometry.

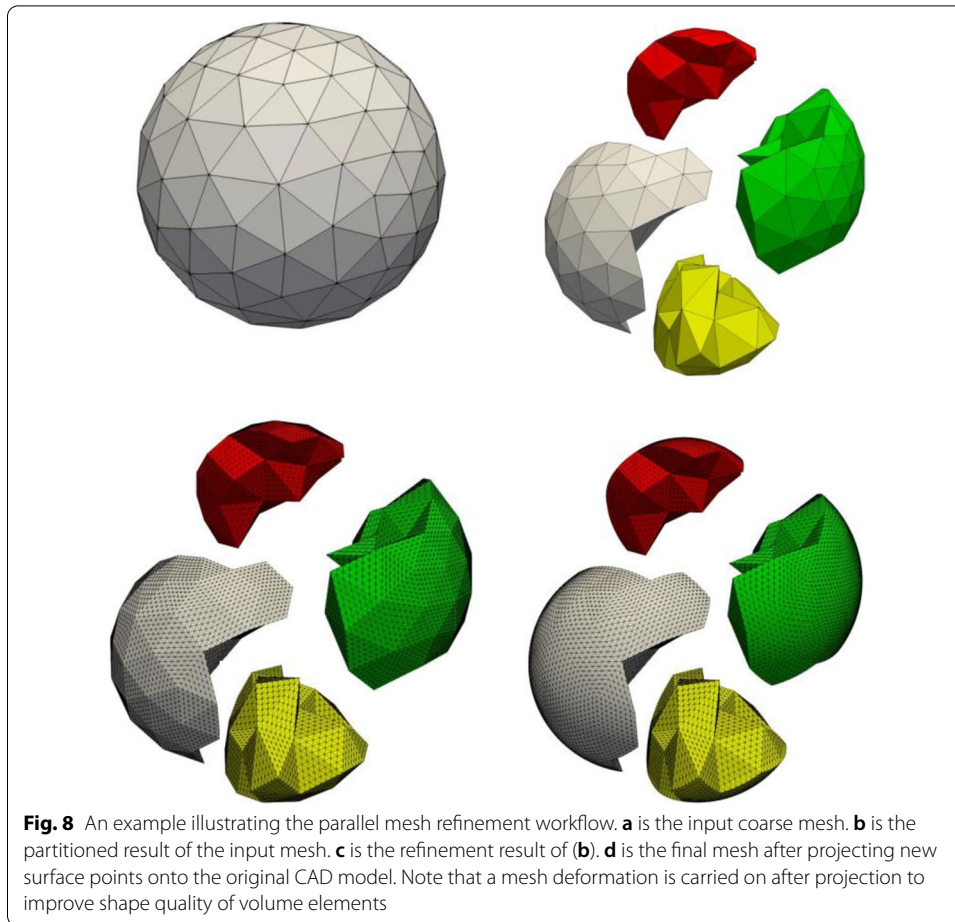
1. **Build\_Map(...)**, which sets up the connections between a CAD model and a surface mesh by employing the algorithm introduced in Section 4.1.
2. **Project\_Curve(...)**, which computes the projection of a point at a curve by employing the algorithm introduced in Section 4.2.
3. **Project\_Surface(...)**, which computes the projection of a point on a surface by employing the algorithm introduced in Section 4.3.
4. **Get\_PatchID(...)**, which returns the face that a facet is classified on.
5. **Get\_PatchCurves(...)**, which returns the curves that bound a face.
6. **Get\_FacetIDs(...)**, which returns a set of facets meeting at an edge.
7. **Attach\_Face(...)**, which classifies a facet onto a face.
8. **Detach\_Face(...)**, which removes a facet-on-face classification.



## 5.2 Parallel mesh refinement

Figure 7 presents the overall workflow of the parallel mesh refinement algorithm, which adopts a manager/worker structure. The manager inputs a CAD model and a coarse hybrid prism-tetrahedra mesh, then extracts the surface of the volume mesh and sets up the connections between the CAD model and the surface mesh. After that, the manager employs a graph partition tool named Metis [33, 34] to subdivide the volume mesh into sub-meshes and distribute each sub-mesh onto the workers. Note that the manager will get one sub-mesh for itself such that it can do the same work as the workers. As will be demonstrated in Section 6, the memory consumption of lightweight kernel is neglected in real applications. Therefore, each worker also gets all the data stored in the ultralight geometry kernel such that the subsequent projection procedure could be executed in parallel. See Fig. 8 for an example illustrating the above workflow.

After the sequential mesh partitioning, all the following steps are conducted in parallel. Each worker refines its sub-mesh by using the subdivision templates shown in Fig. 9,

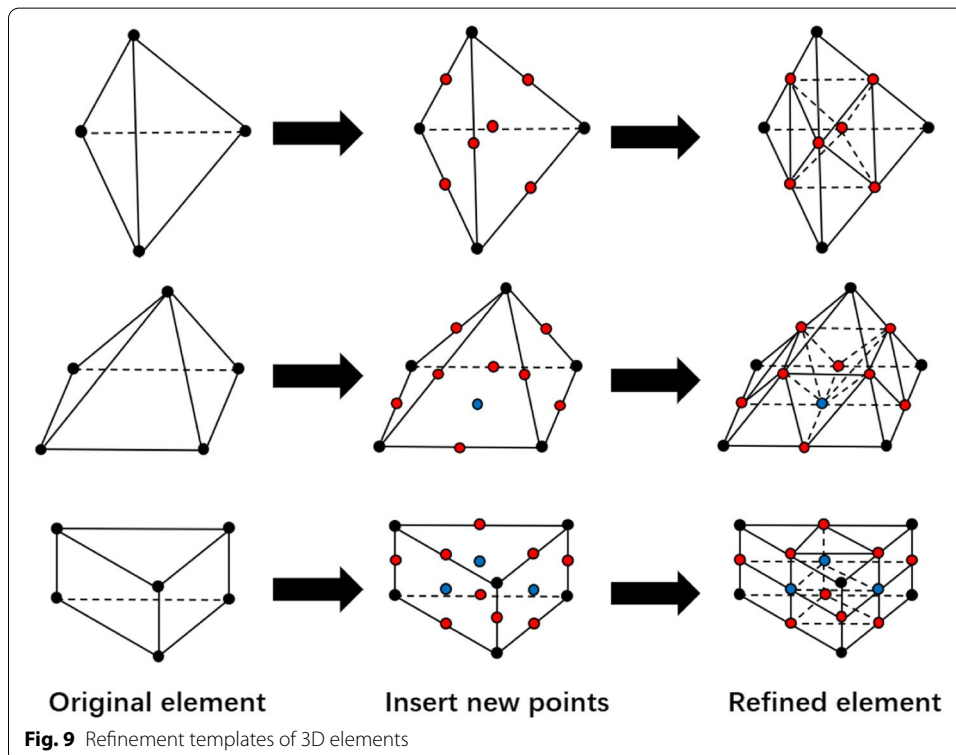


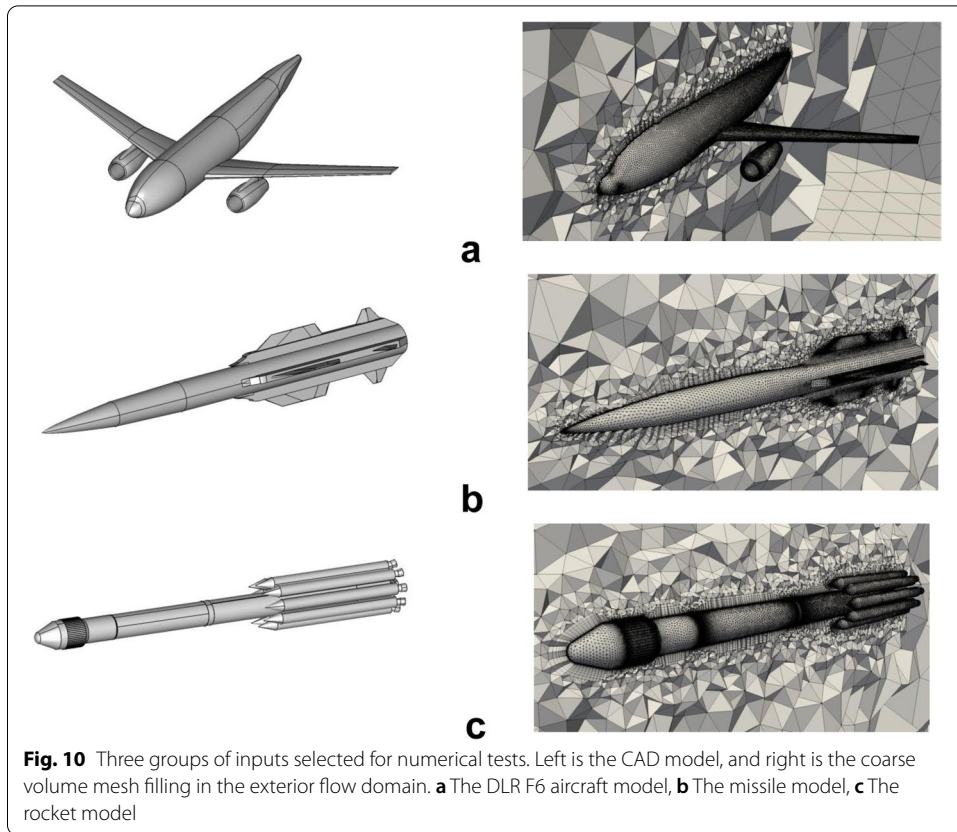
and then projects newly inserted surface points onto the input CAD model. To improve the mesh quality after projection, a moving mesh technique by using radial basis functions (RBF) is employed [35–38]. If the available memory on the manager process is large enough, the distributed refined mesh could be sent back to the manager for combination. However, since the algorithm is developed to deal with large-scale meshes, the output is a distributed mesh in default, which is produced by repartitioning the refined mesh to achieve a better tradeoff between loading balance and communications. The parallel version of Metis, namely ParMetis, is presently employed to achieve this goal.

Three procedures of the parallel mesh refinement algorithm require the callings of APIs listed in Section 5.1.

1. In the initialization procedure, the manager calls **Initialize\_map(...)** to set up the connections between the CAD model.
2. In the refinement procedure, the workers call **Attach\_face(...)** to classify newly formed surface facets onto CAD faces and **Detach\_face(...)** to remove the connections between split facets and CAD faces.

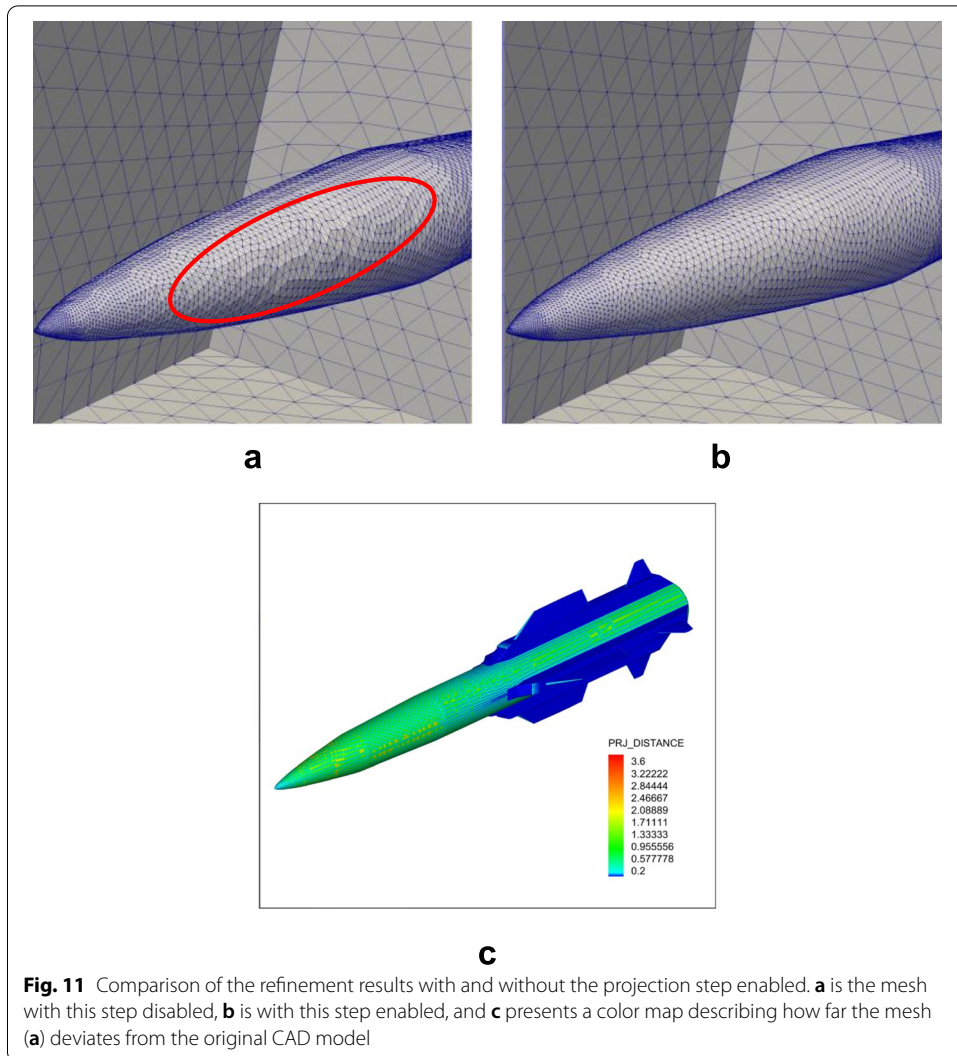
3. In the project procedure, the workers project newly inserted points back onto the original CAD model. Points are dealt with individually. Algorithm 4 presents how to deal with one point by classifying different cases.





**Table 2** Statistics of the selected inputs

Model	#Curves	#Surfs.	#Total elems	#Tetra. elems	#Pyramid elems	#Prism elems	#Vol. nodes	#Surf. elems
F6	238	98	5.13 M	1.57 M	0.04 M	3.52 M	2.057 M	0.174 M
Missile	343	127	2.30 M	0.54 M	0.03 M	1.73 M	0.977 M	0.072 M
Rocket	1098	439	5.73 M	0.52 M	0.04 M	5.17 M	2.712 M	0.147 M



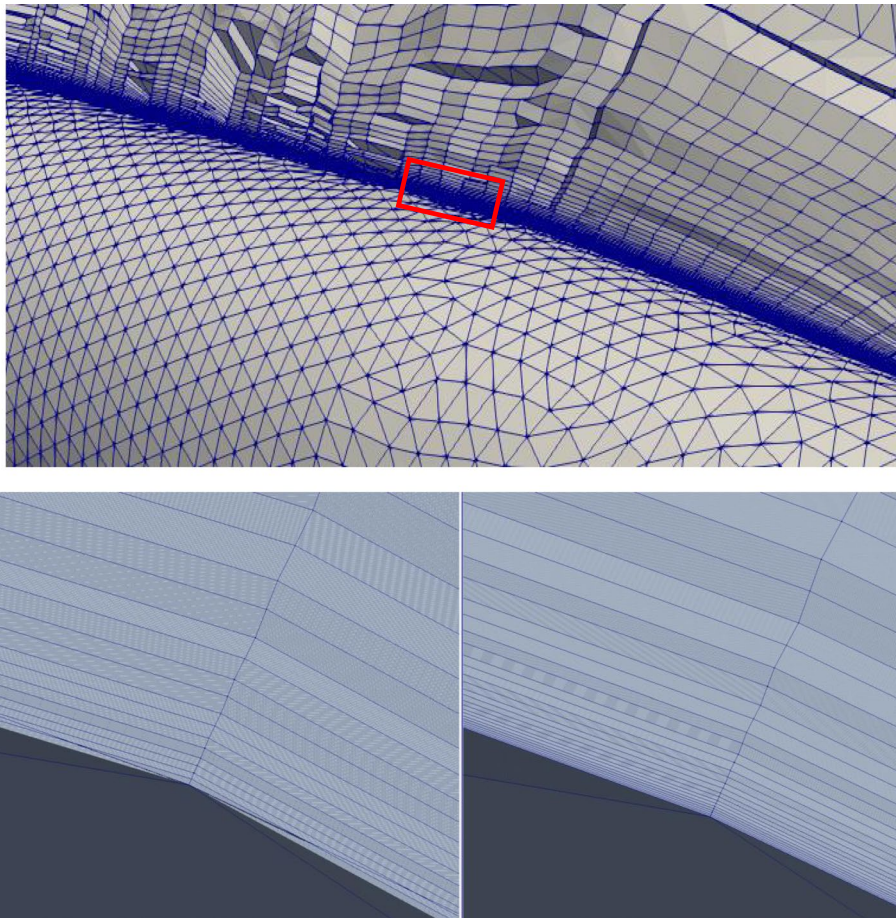


**Table 3** Total timing data and breakdown of the overall refinement algorithm

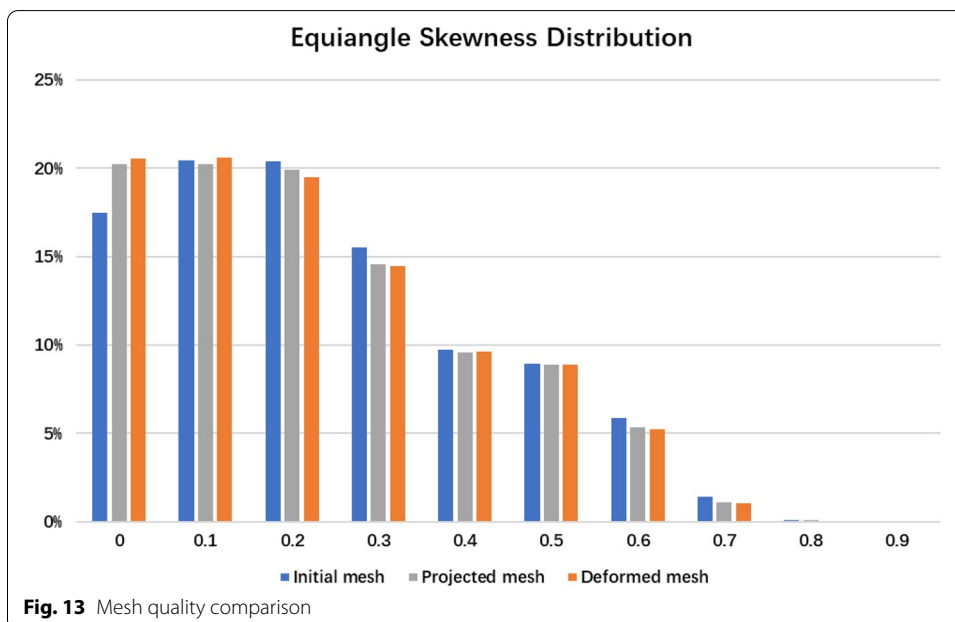
Model	#Output elems.	#Cores	Time consumed by individual steps (s)					Total Time (s)
			Map	Partition	Refine	Project	Deform	
F6	0.33B	256	2.7	164.5	2.3	25.3	1901.8	2108.6
		512	2.7	164.0	1.2	13.8	489.5	687.1
		1024	2.8	193.4	0.7	8.7	225.4	451.4
Missile	0.15B	256	1.2	68.4	1.1	16.9	146.1	228.2
		512	1.1	59.6	0.6	9.0	82.5	154.3
		1024	1.1	69.3	0.3	5.1	38.8	121.1
Rocket	0.37B	256	3.3	155.7	5.0	13.5	1450.1	1644.4
		512	3.2	185.6	2.4	7.9	746.2	969.8
		1024	3.1	179.5	0.9	5.3	469.8	679.8

**Table 4** Comparing of projection time

Model		EGADSIite	Our method
sphere_example	#Surf. elems	265,216	269,690
	#Surf. nodes	132,610	134,847
	Time of projection / s	11.1	5.8
Nozzle_example	#Surf. elems	228,040	231,278
	#Surf. nodes	114,022	115,641
	Time of projection / s	9.1	6.3
Complex (medium size)	#Surf. elems	68,156	69,844
	#Surf. nodes	34,038	34,882
	Time of projection / s	13.2	8.6
Complex (finer size)	#Surf. elems	279,068	281,040
	#Surf. nodes	139,494	140,480
	Time of projection / s	31.6	18.6



**Fig. 12** Comparison of elements before and after mesh deformation. **a** The prism elements near the surface. **b** inverted elements. **c** corrected elements after RBF deformation



**Algorithm 4.** Project a point back onto the original CAD model

---

$v$ : the surface mesh vertex to be projected  
 $e$ : provided only when  $v$  is splitting an edge (see blue points in Figure 9)  
 $f$ : the facet split by  $v$ . When more than one facet is split by  $v$ , provide any one of them  
 $\varepsilon$ : a user-specified tolerance

1. **If**  $v$  is a point splitting an edge  $e$  (see blue vertices in Figure 9)
2. Call `Get_facetIDs( $e$ , & $T$ )` to get a set of facets meeting at  $e$ , denoted by  $T=\{t_1, \dots, t_n\}$
3. **For**  $i=1, \dots, n$
4. Call `Get_patchID( $t_i$ , & $f_i$ )` to get the face that  $t_i$  is classified on, denoted by  $f_i$
5. Call `Query_curveIDs( $f_i$ ,  $C_i$ )` to get a set of curves bounding  $f_i$ , denoted by  $C_i$
6. **End for**
7. **If** the intersection set of  $C_1, \dots$ , and  $C_n$  (denoted by  $C$ ) is not empty
8. **For** each curve  $c$  belonging to  $C$
9. Call `Project_curve( $v$ ,  $c$ , & $p$ )` to get the possible projection of  $v$ , denoted by  $p$
10. **If** the distance between  $v$  and  $p$  is smaller than  $\varepsilon$
11. **Return**  $p$
12. **End if**
13. **End for**
14. **End if**
15. **End if**
16. Call `Get_patchID( $t$ , & $f$ )` to get the face that  $t$  is classified on, denoted by  $f$
17. Call `Project_surface( $v$ ,  $f$ , & $p$ )` to get the projection of  $v$ , denoted by  $p$
18. **Return**  $p$

---

**6 Numerical experiments**

To prove the effectiveness and efficiency of the proposed method and library, numerical tests were carried out on TianHe Exascale Prototype System, in which each computer node contains a Matrix-2000+ CPU configured with 32 computer cores and 16 GB memory. Three typical aerodynamics inputs (see Fig. 10) are selected and their info is listed in Table 2.

The mapping and projection are accomplished by the ultralight geometry kernel. The mapping step is a serial process and only depends on the complexity of the input geometry and the size of initial surface mesh and it is not a compute-intensive task.

The projection process is a critical step in mesh refinement. Figure 11 shows how the projection process improves the quality of a mesh on the missile model. Figure 11a is the one-level refinement result before projection and 11b is its counterpart after projection. The mesh in Fig. 11b is visually smoother than the mesh in Fig. 11a because the mesh after projection respects the original geometry more accurately. To quantify the movement distance resulted by the projection, Fig. 11c presents a color map of that distance, in which each node is assigned a value representing the distance between the origin position of a mesh point and the original geometry.

Table 3 lists the timing data of the overall refinement algorithm. The timing cost of the projection step is relatively small in all tests. By comparison, if a general-purpose CAD kernel such as OpenCascade is employed, this cost can be much larger, as reported in [13]. To evaluate the efficiency of our projection algorithm, we compared our method with another open-source lightweight geometry kernel named EGADSLite, which is reported much faster than OpenCascade in such a kind of computation [15].

The geometry models provided in the EGADSlite project are used as inputs. Testing meshes are generated separately by the two libraries with similar element number and the projection of EGADSlite is accomplished by the `EGLite_invEvaluate()` function. Table 4 shows the time cost of the two projection algorithms on a personal computer with an Intel I5-6500 CPU and 16 GB memory.

Table 4 shows that our projection algorithm is faster than EGADSlite in all cases.

The last step in the parallel refinement is mesh deformation, which takes the largest portion of the total time in all the tests listed in Table 3. The primary reason is that we presently employ an RBF algorithm without incorporating any data reduction techniques [38], which ensures a better mesh deformation effect at a rather huge cost of computing time. Nevertheless, the percentage of this portion decreases evidently when more computer cores are invested owing to the efficient parallelization of this step.

The projection step directly replaces newly inserted surface mesh nodes with their projections on the original CAD model. This process may degrade or even invert the elements connected to these nodes, in particular when the elements are stretched prisms with very small lateral edges. Figure 12b presents a case in which some prisms near the wall of the missile model are inverted due to the projection. After performing the RBF-based mesh deformation, the quality of these elements is remarkably improved, as seen in Fig. 12c.

Finally, we analyze the mesh quality statistics with the metric Equiangular skewness [39] which is used to evaluate the shape quality of elements. This value varies between 0 and 1: 0 stands for an element with the best quality and 1 for a degenerate element. In real simulations, it is required that the skewness values of the majority of elements are below 0.8 and the elements with their skewness values above 0.9 should be removed as many as possible. For the F6 case, Fig. 13 presents the prism element quality of the initial mesh, the projected mesh and the final mesh (after deformation). In general, the quality of the refined elements highly depends on the input ones. However, projection and deformation can make certain improvement of mesh quality. In this case, the projection and deformation procedure increases the percentage of elements with satisfactory skewness values (i.e., below 0.2) and decreases unsatisfactory skewness values (i.e., above 0.8) remarkably.

## 7 Conclusions

In this study, it is shown that an ultralight geometry processing library is powerful enough to ensure the refined surface respects the original CAD model accurately. A very small set of data-structures and algorithms is included in the kernel. Techniques are developed to improve the robustness and efficiency of these data-structures and algorithms. Meanwhile, with the aid of this kernel, a parallel mesh refinement algorithm is enhanced with the ability to respect the original CAD model at a small computing cost. Numerical experiments configured with geometry with industry-level complexity are presented to certify the performance of the developed algorithms.

### Acknowledgements

N/A

### Authors' contributions

The research output comes from a joint effort. All authors read and approved the final manuscript.

### Funding

This research is funded by the National Numerical Wind Tunnel Project of China.

**Availability of data and materials**

The datasets used and/or analyzed during the current study are available from the corresponding author upon reasonable requests.

**Declarations****Competing interests**

The authors declare that they have no competing interests.

**Author details**

<sup>1</sup>Center for Engineering and Scientific Computation, Zhejiang University, Hangzhou 310027, China. <sup>2</sup>School of Aeronautics and Astronautics, Zhejiang University, Hangzhou 310027, China. <sup>3</sup>China Aerodynamics Research and Development Center, Mianyang 621000, China. <sup>4</sup>State Key Lab of CAD&CG, Zhejiang University, Hangzhou 310027, China.

Received: 16 September 2021 Accepted: 23 September 2021

Published online: 29 March 2022

**References**

- Weatherill NP, Hassan O, Morgan K, Jones JW, Larwood BG, Sorenson K (2002) Aerospace simulations on parallel computers using unstructured grids. *40*(1–2):171–187
- De Cougny HL, Shephard MS (1999) Parallel unstructured grid generation. In: Thompson JF, Soni BK, Weatherill NP (eds) *CRC Handbook of Grid Generation*. CRC Press, Boca Raton, pp. 24.1–24.18
- Chrisochoides N (2006) Parallel mesh generation. In: Bruaset AM, Tveito A (eds) *Numerical solution of partial differential equations on parallel computers*. Springer, Heidelberg, pp 237–264
- Löhner R (2014) Recent advances in parallel advancing front grid generation. *Arch Comput Meth Eng* 21(2):127–140
- Chen J, Zhao D, Zheng Y, Xu Y, Li C, Zheng J (2017) Domain decomposition approach for parallel improvement of tetrahedral meshes. *J Parallel Distribut Comput* 107:101–113
- Laug P, Guibault F, Borouchaki H (2017) Parallel meshing of surfaces represented by collections of connected regions. *Adv Eng Softw* 103:13–20
- Yilmaz Y, Ozturan C (2015) Using sequential NETGEN as a component for a parallel mesh generator. *Adv Eng Softw* 84:3–12
- Freitas MO, Wawrzynek PA, Cavalcante-Neto JB, Vidal CA, Martha LF, Ingraffea AR (2013) A distributed-memory parallel technique for two-dimensional mesh generation for arbitrary domains. *Adv Eng Softw* 59:38–52
- Chen J, Zhao D, Huang Z, Zheng Y, Wang D (2012) Improvements in the reliability and element quality of parallel tetrahedral mesh generation. *Int J Numer Methods Eng* 92(8):671–693
- Zhao D, Chen J, Zheng Y, Huang Z, Zheng J (2015) Fine-grained parallel algorithm for unstructured surface mesh generation. *Comput Struct* 154:177–191
- Lintermann A, Schlimpert S, Grimm JH, Günther C, Meinke M, Schröder W (2014) Massively parallel grid generation on HPC systems. *Comput Methods Appl Mech Eng* 277:131–153
- Jiao X, Wang D (2012) Reconstructing high-order surfaces for meshing. *Eng Comput* 28(4):361–373
- Zhao Z, Zhang Y, He L, Chang X, Zhang L (2020) A large-scale parallel hybrid grid generation technique for realistic complex geometry. *Int J Numer Methods Fluids* 92(10):1235–1255
- Open Cascade (2016) Open Cascade Technology 7.2.0. <http://www.opencascade.com/>
- Haines R, Dannenhoffer J (2018) EGADSLite: a lightweight geometry kernel for HPC. Paper presented at the 2018 AIAA aerospace sciences meeting, AIAA 2018-1401. Kissimmee, Florida, 8–12 January 2018
- Sheffer A, Bercovier M, Blacker TED, Clements JAN (2000) Virtual topology operators for meshing. *Int J Comput Geom Appl* 10(03):309–331
- Inoue K, Itoh T, Yamada A, Furuhashi T, Shimada K (2001) Face clustering of a large-scale CAD model for surface mesh generation. *Comput Aided Des* 33(3):251–261
- Sheffer A (2001) Model simplification for meshing using face clustering. *Comput Aided Des* 33(13):925–934
- Dannenhoffer J, Haines R (2003) Quilts: a technique for improving boundary representations for CFD. Paper presented at the 16th AIAA computational fluid dynamics conference, AIAA 2003-4131. Orlando, Florida, 23–26 June 2003
- Foucault G, Cuillière J-C, François V, Léon J-C, Maranzana R (2008) Adaptation of CAD model topology for finite element analysis. *Comput Aided Des* 40(2):176–196
- Chen J, Cao B, Zheng Y, Xie L, Li C, Xiao Z (2015) Automatic surface repairing, defeaturing and meshing algorithms based on an extended B-rep. *Adv Eng Softw* 86:55–69
- Turner M, Peiró J, Moxey D (2016) A variational framework for high-order mesh generation. *Proc Eng* 163:340–352
- Zhao Z, Li M, He L, Shao S, Zhang L (2019) High-order curvilinear mesh generation technique based on an improved radius basic function approach. *Int J Numer Methods Fluids* 91(3):97–111
- Tang J, Cui P, Li B, Zhang Y, Si H (2020) Parallel hybrid mesh adaptation by refinement and coarsening. *Graph Model* 111:101084
- Zhu X (2000) Free-form curve and surface modeling technology. Science Press, Beijing. (in Chinese)
- Ferguson J (1964) Multivariable curve interpolation. *J ACM* 11(2):221–228
- Xiao Z, Chen J, Zheng Y, Zheng Z, Wang D (2016) Booleans of triangulated solids by a boundary conforming tetrahedral mesh generation approach. *Comput Graph* 59:13–27
- Dey S, Shephard MS, Flaherty JE (1997) Geometry representation issues associated with p-version finite element computations. *Comput Methods Appl Mech Eng* 150(1–4):39–55

29. Hormann K, Agathos A (2001) The point in polygon problem for arbitrary polygons. *Comput Geom* 20(3):131–144
30. Brent RP (2013) Algorithms for minimization without derivatives. Courier Corporation, North Chelmsford
31. Peirò J (1999) Surface grid generation. CRC Press, New York
32. Li X, Wu Z, Pan F, Liang J, Zhang J, Hou L (2019) A geometric strategy algorithm for orthogonal projection onto a parametric surface. *J Comput Sci Technol* 34(6):1279–1293
33. George Karypis (2013) METIS - Serial graph partitioning and fill-reducing matrix ordering. <http://www.glaros.dtc.umn.edu/gkhome/metis/metis/overview>.
34. Karypis G, Kumar V, Comput S (1998) A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J Sci Comput* 20(1):359–392
35. de Boer A, van der Schoot MS, Bijl H (2007) Mesh deformation based on radial basis function interpolation. *Comput Struct* 85(11–14):784–795
36. Rendall T, Allen C (2008) Fluid-structure interpolation and mesh motion using radial basis functions. *Int J Numer Methods Eng* 74:1519–1559
37. Rendall TCS, Allen CB (2009) Efficient mesh motion using radial basis functions with data reduction algorithms. *J Comput Phys* 228(17):6231–6249
38. Rendall TCS, Allen CB (2010) Reduced surface point selection options for efficient mesh deformation using radial basis functions. *J Comput Phys* 229(8):2810–2820
39. Cadence Design Systems (2021) POINTWISE user manual. <http://www.pointwise.com/doc/user-manual/examine/functions/equiangle-skewness.html>. Accessed 09 Jan 2021

### Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

**Submit your manuscript to a SpringerOpen<sup>®</sup> journal and benefit from:**

- ▶ Convenient online submission
- ▶ Rigorous peer review
- ▶ Open access: articles freely available online
- ▶ High visibility within the field
- ▶ Retaining the copyright to your article

---

Submit your next manuscript at ▶ [springeropen.com](https://www.springeropen.com)

---