Automatic Sizing Functions for Unstructured Surface Mesh Generation

Jianjun Chen [a*], Zhoufang Xiao [a], Yao Zheng [a], Kewei Liang [b], Chenfeng Li [c], Jianjing Zheng [a]

[a] *Center for Engineering and Scientific Computation and School of Aeronautics and Astronautics, Zhejiang University, Hangzhou 310027, China*
[b] *School of Mathematics, Zhejiang University, Hangzhou 310027, China*
[c] *Zienkiewicz Centre for Computational Engineering, and Energy Safety Research Institute, College of Engineering, Swansea University, Swansea SA2 8PP, U.K.*

Abstract. Accurate sizing functions are crucial for efficient generation of high-quality surface meshes, but to define the sizing function is often the bottleneck in complicated mesh generation tasks due to the tedious user interaction involved. We present a novel algorithm to automatically create high-quality sizing functions. First, the tessellation of a CAD model is taken as the background mesh and combined with user-specified parameters to define an initial sizing function. Then, a convex nonlinear programming problem is formulated and solved efficiently to obtain a smoothed sizing function that corresponds to a mesh satisfying necessary gradient constraint conditions with a significantly reduced element number. Finally, this sizing function is applied in an advancing front surface mesher. With the aid of a parametric mesh and a walk-through algorithm, an efficient sizing-value query scheme is developed for surface meshing. Meshing experiments of some very complicated geometry models are presented to demonstrate that the proposed sizing-function approach enables accurate and fully automatic surface mesh generation.

Key words. Mesh generation; Surface mesh; Sizing function; Adaptive; Background mesh

1. Introduction

In the field of computational aerodynamics, a great success of unstructured mesh technologies has been witnessed in the past few decades due to its automatic and adaptive abilities for complex geometry configurations **[1]**. Nowadays, many commercial and in-house codes can generate unstructured meshes in a very reliable and computationally efficient manner **[2-5]**. However, preparing a suitable input for the unstructured meshing pipeline remains a major performance bottleneck in many cases. In general, this input contains a geometry that defines the meshing domain and a sizing function that defines the distribution of element scales over the meshing domain. This study focuses only on the sizing function for unstructured surface mesh generation, while it is noted that preparing high-quality geometry inputs is a very active research topic in the community of mesh generation **[6, 7]**.

A good sizing function 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. In computational aerodynamics, grid sources are popular tools employed by many meshing codes to define sizing functions **[8-12]**. For simple configurations, the time cost of defining the sources may be affordable if a user friendly graphic interface is available. However, for complicated aerodynamics models, e.g. a fully loaded fighter, hundreds of grid sources may be required, and the interactive process that defines these sources is error-prone and time-consuming. By contrast, the subsequent automatic meshing pipeline may only consume minutes of wall-clock time. For these complicated geometry models, without

* Corresponding author: Tel.: +86-571-87951883; Fax: +86-571-87953168.
E-mail: chenjj@zju.edu.cn

considering the geometry preparation step, the performance bottleneck for unstructured mesh generation lies in the phase of defining sizing functions rather than the mesh generation itself.

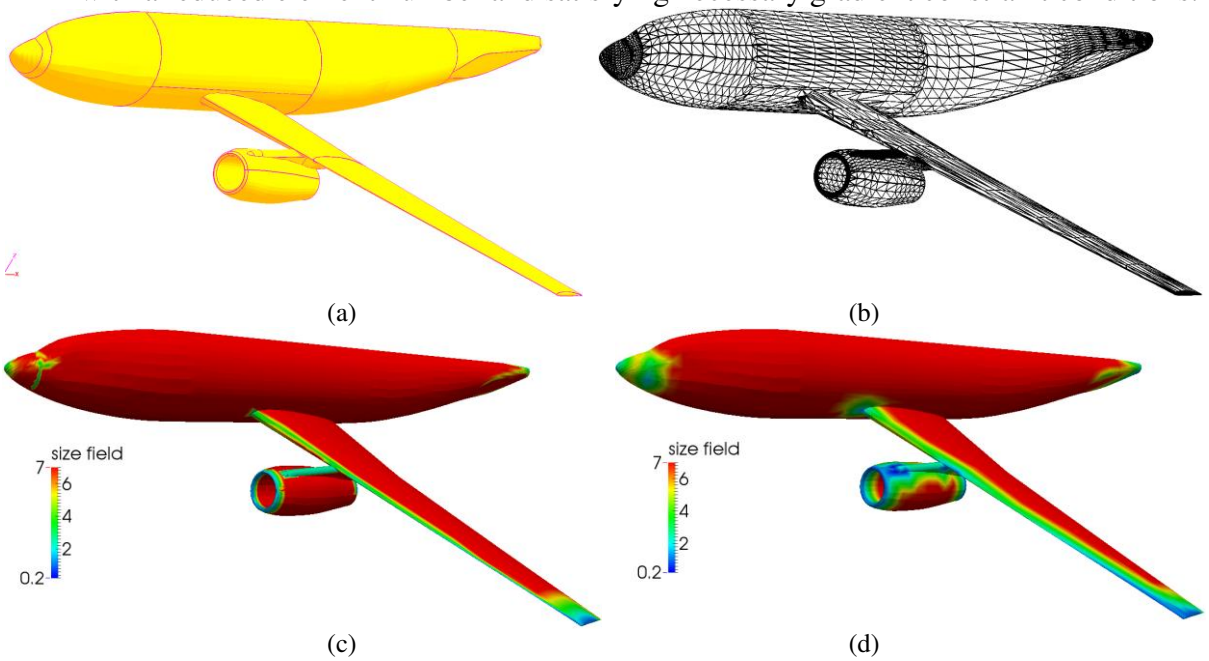
For many years, solution-adaptive techniques have been expected to be able to remove the dependence of analysis accuracy on initial mesh configurations. However, it was reported that an adapted solution might be invalid if it originates from a poor-quality mesh **[10]**. In fact, a good initial mesh can either eliminate the need for adaptive mesh refinement or enhance the performance of many adaptation methods. Therefore, even configured with an adaptive solver, a suitable sizing function is still indispensable for initial mesh generation.

In this study, a fully automatic algorithm that defines sizing functions for surface mesh generation is proposed. Figure 1 illustrates the main steps of this algorithm using the F6 wing-body-nacelle-pylon aircraft model (hereafter referred to as the F6 model) as the example.

(1) *Creating the background mesh*. A triangular mesh is output by tessellating the input CAD model and used as the background mesh of the sizing function.

(2) *Initialising the sizing function*. Firstly, curvature and proximity features of the input model are calculated. Next, element scales adapted to these features are defined at background mesh nodes.

(3) *Smoothing the sizing function*. A convex nonlinear programming (NLP) is formulated and efficiently solved to obtain a smoothed sizing function that corresponds to a mesh with a reduced element number and satisfying necessary gradient constraint conditions.



(a) (b) (c) (d)

Figure 1. An illustration for the proposed algorithm using an F6 aircraft model. (a) The input CAD model. (b) The background mesh of the sizing function, which is output by tessellating the input CAD model. (c) The initial sizing function adapted to local curvature and proximity features. (d) The smoothed sizing function.

Step 3 is the key of the proposed algorithm. To limit the gradients of element scales over a background triangular mesh, a gradient constraint equation is introduced with the nodal sizing values as variables. Based on this equation, a convex NLP is formulated to constrain the gradients of element scales over the entire meshing domain. The solution of this NLP corresponds to a mesh with a reduced magnitude. Theoretical analysis reveals that this NLP is always solvable and any local optimal solution is also its global optimal solution. An efficient numerical scheme is also developed to solve this NLP.

Given a sizing function defined on an unstructured mesh, a challenging issue is how to efficiently query the sizing value of a given point. Two techniques are proposed to resolve

this issue. It is noted that the input CAD model is composed of many parametric curves and surfaces whose meshing procedures are conducted individually. Therefore, the concept of *parametric mesh* is defined to limit sizing-value queries on the parametric space of a particular surface or curve, and the involved elements only refer to those *covering* this surface or curve. In addition, a fast walk-through algorithm is developed to search the background element containing a given point (hereafter referred to as the *base* element) at the shortest path.

The rest of the paper is organized as follows. In Section 2, the contribution of the proposed approach is highlighted in the context of related literature. In Section 3, the convex NLP for sizing-function smoothing is formulated and theoretically analyzed. Section 4 provides the implementation details for the main steps of creating a geometry-based adaptive sizing function with limited gradients of element scales, while Section 5 addresses the issue of efficient query of sizing values. In Section 6, various numerical examples are presented to demonstrate the effectiveness and efficiency of the new approach, after which concluding remarks are summarized in Section 7.

2. The contributions

This contribution of this work mainly includes the background mesh for sizing functions, the gradient-constraint algorithm and the size-query algorithms.

2.1. Using the tessellation of the input CAD model as the background mesh

A fundamental feature that can be used to classify the algorithms of specifying sizing functions is the background mesh adopted **[8-14]**. The most prevailing one is the Cartesian mesh, which is interiorly organized as a quadtree or octree and requires the tree-level difference of neighboring cells to be less than one. Consequently, the refinement of a single cell has to be propagated into its neighboring cells. Therefore, the Cartesian mesh based scheme is expensive in terms of computing time and storage requirement **[8-12]**.

Most CAD systems provide routines to tessellate a CAD model into a triangular mesh. Compared with Cartesian meshes, triangular meshes have far better topological flexibility. Thus, for models with similar geometric complexity, the triangular mesh required to define suitable sizing functions can be coarser than the Cartesian mesh by one order or more **[9-12]**. Besides, unlike the Cartesian mesh where cells are cut through by geometry boundaries, the triangular mesh is *boundary conforming*. This property is very helpful when developing various sizing-function schemes. For instance, an extended B-rep can be setup after tessellating the CAD model to connect the topological entities of the CAD model (*face*, *curve* and *point*) and the mesh entities (*facets*, *edges*, *nodes*) occupying these topological entities. The sizing-value query employed for surface meshing can be improved using this extended B-rep as explained below:

(1) The meshing algorithm manages curves and surfaces individually, and therefore the query employed in the meshing procedure of a single curve or surface only needs to visit the set of background edges or facets classified on the curve or surface to find the base element.

(2) By projecting these edges or facets into the parametric space of the curve or face, the base-element search can be defined in the two-dimensional parametric space instead of the original three-dimensional physical space.

Meanwhile, the mesh magnitude is a key index for evaluating the mesh quality, and can therefore be applied to evaluate the *quality* of a sizing function. The boundary-conforming property of the triangular background mesh makes it possible to predict the magnitude of the mesh adapted to a given sizing function before conducting the meshing task (see Appendix A).

This property is undoubtedly useful for those algorithms that need to evaluate or compare the quality of sizing functions.

2.2. The gradient limiting approach by the solution of an convex NLP

A few approaches have been proposed to limit the gradients of element scales by changing the sizing values at background mesh nodes. Borouchaki *et al.* **[15]** presented a technique named *Hc-Correction* to constrain the gradients of element scales along edges. Later, Pippa and Caligiana **[16]** presented a new technique named *GradH-Correction* to constrain the gradients of element scales over background element interiors. Although the GradH-Correction algorithm ensures a *gradient-limited* sizing function over the entire meshing domain, it does not attempt to minimize the change to the initial sizing function and therefore takes the risk to get a locally over-refined sizing function. To overcome this drawback, Persson *et al.* [**17**] proposed to smooth the sizing function by solving a Hamilton-Jacobi equation. However, it remains an issue to extend this approach to smooth the sizing function defined on a background mesh composed of many highly stretched elements. Based on previous experience, such a mesh is not a qualified input for a PDE solver.

In **[16]**, Pippa and Caligiana proposed an equation that must be met by a gradient-limited sizing function over a triangular domain. Based on this equation, we propose a convex NLP to smooth the sizing function. The solvability and convexity of this NLP is proved rigorously, and an efficient numerical scheme that can balance solution quality and computing time is developed. It is noted that the solution of this NLP is not only subject to the gradation-limiting condition, but also aimed at minimizing the difference of the initial sizing function and the smoothed one. Therefore, unlike the GradH-Correction algorithm, the proposed approach will not result in a locally over-refined sizing-function. Moreover, because a positive correlation exists between the minimization function adopted in the NLP and the goal to minimize the mesh magnitude, the solution of this NLP usually corresponds to a mesh with reduced magnitude.

2.3. The fast scheme of sizing-value query on an unstructured mesh

For a Cartesian background mesh, the time complexity of querying the sizing value of a point is proportional to the depth of the tree. However, if the sizing function is defined on an unstructured mesh, the time complexity of a brute-force implementation of this query is proportional to the magnitude of the background mesh in its worst case. Therefore, although the proposed approach requires less time and memory usage in preparing the sizing function than the Cartesian mesh based approaches, it is necessary to improve sizing-value query on an unstructured mesh in order to ensure a better overall efficiency.

In Section 2.1, it is proposed to limit sizing-value queries to the parametric space of a particular surface or curve that contains the associated elements. However, if not improved further, the computing efficiency of the size-query routine is still unacceptable, in particular for a *big* surface on which thousands of background elements are classified.

Note that the base-element search is the most time consuming step of the sizing-value query scheme. Starting from an initial guess of the base element, the walk-through algorithm **[18]** attempts to find the base element through the shortest path with the aid of element neighboring indices. Nevertheless, the efficiency of the walk-through algorithm highly depends on the distance between the initial guess and the base element. Since the base elements for two geometrically neighboring positions are usually located very closely (or even the same), it is possible to start the walk-through algorithm from a very *good* guess of the base element. For instance, in the advancing front meshing procedure, a new point is needed to connect with the active front edge to form a new triangle. Since the new point is usually in the neighborhood of the active front edge, the base elements enclosing the end

points of the edge can be selected as the starting point of the walk-through algorithm when querying the sizing value at the new point.

The above sizing-value query scheme has been used in our in-house surface mesher. Numerical experiments show that it visits less than ten background elements for each sizing-value query. As a result, the sizing-value query scheme consumes less than 10% of the meshing time. Here, the meshing time does not include the time consumed in preparing the sizing function.

3. The proposed gradient constraint approach

3.1. Preliminary definitions

3.1.1 Unit mesh. Introduced in **[15, 19, 20]**, a unit mesh refers to a mesh adapted to a sizing function over a domain that has all edges of unit length with respect to the Riemannian structure associated with the sizing function. The concept is demonstrated with a one-dimensional mesh in Figure 2. Assuming that the sizing function over the edge is $h(t)$ $(t \in [0,1])$ and the length of the edge is l, the number of mesh segments in this edge is:

$$n = l\int_0^1 \frac{1}{h(t)}dt\,. \tag{1a}$$

For each segment of this edge,

$$1 = l\int_{t_i}^{t_{i+1}} \frac{1}{h(t)}dt \qquad (i = 1,2,\cdots,n)\,. \tag{1b}$$

Given a metric $\mathbf{M} = \left[1/h^2(t)\right]$ defined in this edge, the length of this segment, in the space defined by $\mathbf{M}$, is

$$(\Delta l_i)_{\mathbf{M}} = \int_{t_i}^{t_{i+1}} \sqrt{l \cdot \frac{1}{h^2(t)} \cdot l}\,dt = l\int_{t_i}^{t_{i+1}} \frac{1}{h(t)}dt = 1 \qquad (i = 1,2,\cdots,n)\,.$$

This indicates that an ideal mesh segment in the space defined by **M** has a unit length. Correspondingly, we call this mesh a *unit mesh* **[15, 19, 20]**.

3.1.2 Geometric progressive mesh. For the mesh shown in Figure 1, $\forall \beta \geq 1.0$, if

$$\frac{\Delta l_{i+1}}{\Delta l_i} = \begin{cases} \beta & \Delta l_{i+1} \geq \Delta l_i \\ 1/\beta & \Delta l_{i+1} < \Delta l_i \end{cases} (i = 1,2,\cdots,n-1)\,,$$

this mesh is called a *geometric progressive mesh*, and β the *progressive factor*.

3.1.3 The H-variation of an edge. It is supposed that the mesh shown in Figure 1 satisfies the following assumptions:

(1) It is a unit mesh, i.e., the predefined sizing function is accurately respected;

(2) It is a geometric progressive mesh with the progressive factor β;

(3) The size variation follows a linear function.

Let h_i (i=0,1,...,n) denote the size value at point a_i, without loss of generality, we assume $h_n \geq h_0$ (h_0 and h_n are the sizing values at two end points of the edge) and

$\Delta l_{i+1} / \Delta l_i = \beta \qquad (i = 1,2,\cdots,n-1)$.

Finally, the following relation can be obtained **[15]**:

$$\Delta h / l = (h_n - h_0)/l = \ln \beta\,. \tag{2}$$

In **[15]**, $\Delta h / l$ is called the *H-variation*, denoted by $h_v(a_0a_n)$, where a_0 and a_n are the end points of the edge. When l approaches zero, the H-variation represents the gradient of the sizing function at a point.

It can be derived from Equation 2 that, to limit the length ratio of neighboring mesh segments below β, the H-variation of the edge must satisfy the following relation

$$h_v(a_0a_n) = \Delta h / l \le \ln \beta . \tag{3}$$

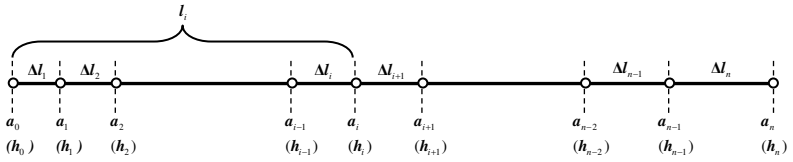


Figure 2. Example of a unit mesh

3.1.4 The gradient of the linear sizing function over a triangular domain. To enable the application to a triangular background mesh, it is necessary to extend the concept of H-variation from an edge to a triangle. Assuming that Ω_T is the domain covered by a triangle T and a linear sizing function is defined over Ω_T, i.e.,

$$h(x, y) = \sum_{i=0}^{2} w_i(x, y)h_i \quad (x, y) \in \Omega_T ,$$

where $h_i (i = 0 \sim 2)$ are the sizing values at three nodes of T, and $w_i(x, y)(i = 0 \sim 2)$ are the area coordinate functions. Given a progressive factor β $(\beta \ge 1.0)$, we require the edge ended with two arbitrary points enclosed by Ω_T to satisfy Equation 3, i.e.,

$$h_v(p_1p_2) = \left|h(p_2) - h(p_1)\right| / \left|p_1p_2\right| \le \ln \beta \quad (\forall p_1, p_2 \in \Omega_T) ,$$

where $h_v(p_1p_2)$ is the H-variation of the edge p_1p_2. When p_2 approaches p_1, it represents the direction derivative of the sizing function $h(x, y)$ along p_1p_2. As p_1p_2 can be an arbitrary direction around p_1, the following relation holds,

$$\left|\nabla h(p_1)\right| = \max_{\forall p_2 \in \Omega_T} \frac{\partial h}{\partial \overrightarrow{p_1p_2}} \le \ln \beta \qquad (\forall p_1 \in \Omega_T) \quad .$$

The gradient of a linear function $h(x, y)$ over Ω_T is constant:

$$\nabla h = const = \frac{1}{2A}\left(\sum_{i=0}^{2} b_i h_i \quad \sum_{i=0}^{2} c_i h_i \right)^T ,$$

where A is the area of T,

$$b_i = y_j - y_m, \quad c_i = x_m - x_j \quad (i = 0 \sim 2; j = (i+1) \bmod 3; m = (j+1) \bmod 3) , \tag{4}$$

and $(x_i, y_i)(i = 0 \sim 2)$ are the coordinates of the ending nodes of T.

Therefore, the sizing function over Ω_T needs to satisfy the following equation.

$$\left|\nabla h\right|^2 = \mathbf{H}^T\mathbf{K}\mathbf{H} \le \ln^2(\beta) , \tag{5}$$

where

$$\mathbf{H} = (h_0, h_1, h_2)^T$$

$$\mathbf{K} = [k_{ij}]_{0 \le i, j \le 2}; \quad k_{ij} = (b_ib_j + c_ic_j) / (4A^2) .$$

Obviously, K is a symmetric matrix, and the principal diagonal elements are positive:

$k_{ii} = (b_i^2 + c_i^2)/(4A^2) = l_i^2/(4A^2) > 0$.

where l_i is the length of the edge opposite to the node i **[16]**.

For a triangle defined in a three-dimensional Cartesian coordinate system, a local coordinate system is defined such that the plane containing the triangle as the XOY plane, and the elements of the matrix **K** are then computed in the XOY space of this transformed coordinate system.

3.2. The constrained optimisation problem

A triangular background mesh composed of *m* triangles and *n* nodes is denoted by

$M_t = \{E = \{e_i \mid i = 1,2,\cdots,m\}, P = \{p_i \mid i = 1,2,\cdots,n\}\}$,

where E and P are the element set and node set, respectively. The sizing value at a node is denoted by $h(p)$, and the gradient of the sizing function over an element is denoted by $\nabla h(e)$.

Given a progressive factor β, to limit the gradient of the sizing function defined on M_t, the following equation must be met:

$|\nabla h(e_i)|^2 \le \ln^2 \beta$, for all $e_i \in E$.

If a sizing function does not satisfy the above condition, it can be corrected by solving the following constrained optimization problem:

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} (h(p_i) - h_0(p_i))^2 \\ s.t. \quad & |\nabla h(e_i)|^2 = \mathbf{H}_i^T \mathbf{K}_i \mathbf{H}_i \le \ln^2(\beta) \quad (i=1,2,...,m). \\ & h_{\min} \le h(p_i) \le h_0(p_i) \qquad (i=1,2,...,n) \end{aligned} \tag{6}$$

Here, $h_0(p_i)$ and $h(p_i)$ are the initial and corrected sizing values at a mesh node p_i. Note that the initial sizing values are usually calculated from certain geometric or physical rules. The corrected sizing value at a mesh node is also required to be less than its initial value because setting larger values may degrade the mesh resolution and eventually affect the simulation accuracy. Meanwhile, to avoid overly small sizing values, a user parameter $h_{\min}$ is set to limit the minimal sizing value. The goal is to minimize the change to the initial sizing values in the least squares sense. As the sizing values are only allowed to be reduced, the optimal sizing values must be as large as possible to achieve this target. Thus, the solution of NLP 6 will result in a sizing function adapted by a mesh containing fewer elements.

3.3. Theoretical analysis of the optimisation model

Fort the sake of generality, a *non-uniform sizing function* is concerned in the following discussions where $\beta > 1.0$. If the preferred sizing function is a uniform one where $\beta = 1.0$, the sizing value at each node should be the same, and the optimal solution of NLP 6 is achieved when

$h(p_i) = \min(h_0(p_1), h_0(p_2), ..., h_0(p_n))$.

Lemma 3.1 The feasible region of NLP 6 is a convex set.

Proof. Let $\mathbf{v} = (h(p_1), h(p_2), ..., h(p_n))$ be the solution of NLP 6, the feasible region of the solution vector is: $\Omega = \Omega_1 \cap \Omega_2$, where

$\Omega_1 = \{\mathbf{v} \mid \mathbf{H}_i^T \mathbf{K}_i \mathbf{H}_i \le \ln^2(\beta)$ for each background element $e_i (i=1,2,...,m)\}$

$\Omega_2 = \{\mathbf{v} \mid h_{\min} \le h(p_i) \le h_0(p_i)$ for each background node $v_i (i=1,2,...,n)\}$.

The constraint defined on a background element is

$F(h_0, h_1, h_2) = \mathbf{H}_i^T \mathbf{K}_i^* \mathbf{H}_i - 4A^2 \ln^2(\beta) \le 0$,

where A is the area of the background element, $h_0 \sim h_2$ are the sizing values at the three nodes of the background element, and

$\mathbf{K}_i^* = 4A^2 \mathbf{K}_i = [k_{jk}^*]_{0 \le j,k \le 2}$; $k_{jk}^* = (b_j b_k + c_j c_k)$.

In the above equation, j and k are indices of the nodes of the background element, and the variables $b_0 \sim b_2$ and $c_0 \sim c_2$ are calculated by Equation 4.

The expression of $F(h_0, h_1, h_2)$ can be rewritten as,

$F(h_0, h_1, h_2) = (h_0, h_1, h_2, 1)\mathbf{M}_c (h_0, h_1, h_2, 1)^T$

where

$$\mathbf{M}_c = [a_{jk}]_{0 \le j,k \le 3} = \begin{pmatrix} b_0^2 + c_0^2 & b_0 b_1 + c_0 c_1 & b_0 b_2 + c_0 c_2 & 0 \\ b_0 b_1 + c_0 c_1 & b_1^2 + c_1^2 & b_1 b_2 + c_1 c_2 & 0 \\ b_0 b_2 + c_0 c_2 & b_1 b_2 + c_1 c_2 & b_2^2 + c_2^2 & 0 \\ 0 & 0 & 0 & -4A^2 \ln^2(\beta) \end{pmatrix}$$

is the coefficient matrix.

Since $F(h_0, h_1, h_2)$ is a quadratic function, the equation $F(h_0, h_1, h_2) = 0$ defines a quadric surface. To determine the type of the quadric surface, the invariants ($I_1 \sim I_4$) and semi-invariants (K_1 and K_2) **[21-23]**† of this quadric surface are computed as follows:

$$I_1 = a_{00} + a_{11} + a_{22} = (b_0^2 + c_0^2) + (b_1^2 + c_1^2) + (b_2^2 + c_2^2) = l_0^2 + l_1^2 + l_2^2 > 0;$$

$$I_2 = \begin{vmatrix} a_{00} & a_{01} \\ a_{01} & a_{11} \end{vmatrix} + \begin{vmatrix} a_{00} & a_{02} \\ a_{02} & a_{22} \end{vmatrix} + \begin{vmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{vmatrix} = (b_0 c_1 - b_1 c_0)^2 + (b_0 c_2 - b_2 c_0)^2 + (b_1 c_2 - b_2 c_1)^2 ;$$

$$= (2A)^2 + (2A)^2 + (2A)^2 = 12A^2 > 0$$

$$I_3 = \begin{vmatrix} a_{00} & a_{01} & a_{02} \\ a_{01} & a_{11} & a_{12} \\ a_{02} & a_{12} & a_{22} \end{vmatrix} = a_{00} \begin{vmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{vmatrix} - a_{01} \begin{vmatrix} a_{01} & a_{12} \\ a_{02} & a_{22} \end{vmatrix} + a_{02} \begin{vmatrix} a_{01} & a_{11} \\ a_{02} & a_{12} \end{vmatrix}$$

$$= (b_0^2 + c_0^2)[(b_1^2 + c_1^2)(b_2^2 + c_2^2) - (b_1 b_2 + c_1 c_2)^2] +$$

$$(b_0 b_1 + c_0 c_1)[(b_0 b_1 + c_0 c_1)(b_2^2 + c_2^2) - (b_0 b_2 + c_0 c_2)(b_1 b_2 + c_1 c_2)] + ;$$

$$(b_0 b_2 + c_0 c_2)[(b_0 b_1 + c_0 c_1)(b_1 b_2 + c_1 c_2) - (b_0 b_2 + c_0 c_2)(b_1^2 + c_1^2)]$$

$$= 0$$

$$I_4 = \begin{vmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{01} & a_{11} & a_{12} & a_{13} \\ a_{02} & a_{12} & a_{22} & a_{23} \\ a_{03} & a_{13} & a_{23} & a_{33} \end{vmatrix} = a_{33} \begin{vmatrix} a_{00} & a_{01} & a_{02} \\ a_{01} & a_{11} & a_{12} \\ a_{02} & a_{12} & a_{22} \end{vmatrix} = -4A^2 \ln^2(\beta) I_3 = 0;$$

$$K_1 = \begin{vmatrix} a_{00} & a_{03} \\ a_{03} & a_{33} \end{vmatrix} + \begin{vmatrix} a_{11} & a_{13} \\ a_{13} & a_{33} \end{vmatrix} + \begin{vmatrix} a_{22} & a_{23} \\ a_{23} & a_{33} \end{vmatrix} = a_{33}(a_{00} + a_{11} + a_{22}) = -4A^2 \ln^2(\beta) I_1 < 0;$$

† Reference **[21]** is a Chinese translation of Reference **[22]**.

$$K_2 = \begin{vmatrix} a_{00} & a_{01} & a_{03} \\ a_{01} & a_{11} & a_{13} \\ a_{03} & a_{13} & a_{33} \end{vmatrix} + \begin{vmatrix} a_{00} & a_{02} & a_{03} \\ a_{02} & a_{22} & a_{23} \\ a_{03} & a_{23} & a_{33} \end{vmatrix} + \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{vmatrix} .$$

$$= a_{33}(\begin{vmatrix} a_{00} & a_{01} \\ a_{01} & a_{11} \end{vmatrix} + \begin{vmatrix} a_{00} & a_{02} \\ a_{02} & a_{22} \end{vmatrix} + \begin{vmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{vmatrix}) = -4A^2 \ln^2(\beta) I_2 < 0$$

Here, $l_0 \sim l_2$ are the lengths of the three edges of the background element.

The values of the above invariants and semi-invariants indicate that the quadric surface defined by the function $F(h_0, h_1, h_2) = 0$ is an elliptical cylinder **[21-23]**, and $F(h_0, h_1, h_2) \le 0$ defines a convex set bounded by this elliptical cylinder. Since Ω_1 is the intersection of a set of such convex sets, it is also a convex set.

Meanwhile, Ω_2 is a convex set because it is composed of a set of box constraints. Therefore, $\Omega = \Omega_1 \cap \Omega_2$ is a closed convex set. □

Theorem 3.1 NLP 6 is a convex programming problem, and any local optimal solution of this problem is also its global optimal solution.

Proof. The objective function of NLP 6

$$f(\mathbf{v}) = \sum_{i=1}^{n} (h(p_i) - h_0(p_i))^2$$

is a least squares function. Its Hessian matrix is a positive definite matrix:

$\mathbf{M}_H = \nabla^2 f(\mathbf{v}) = 2\mathbf{I}$,

where $\mathbf{I}$ is the unit matrix, and

$\mathbf{x}^T \mathbf{M}_H \mathbf{x} > 0, \forall \mathbf{x} \neq \mathbf{0} (\mathbf{x} \in \mathbb{R}^n)$.

Therefore, the objective function is a convex function. Meanwhile, since the feasible region of NLP 6 is a convex set (according to Lemma 3.1), NLP 6 is a convex programming problem.

Note that h_{min} is a user parameter. It can always be set by a value satisfying

$h_{min} < \min(h_0(p_1), h_0(p_2), ..., h_0(p_n))$.

Then, there exists a δ satisfying

$h_{min} < \delta < \min(h_0(p_1), h_0(p_2), ..., h_0(p_n))$,

let $h(p_i) = \delta$, $\forall \beta \ge 1.0$,

$|\nabla h(e_i)|^2 = \mathbf{H}_i^T \mathbf{K}_i \mathbf{H}_i = 0 \le \ln^2(\beta)$.

Hence, the vector $\mathbf{v} = (h(p_1), h(p_2), ..., h(p_n)) = (\delta, \delta, ..., \delta)$ is a feasible solution of NLP 6. If $\beta > 1.0$, this solution strictly satisfies the constraints, which means this solution is an interior point of the feasible region of NLP 6.

Finally, it is known that, for a convex programming problem with at least one interior point in its feasible region, any local optimal solution of this problem is also its global optimal solution **[24]**. □

4. The generation of the sizing function

As illustrated in Figure 1, the process of creating a gradient-constrained sizing function takes three steps. The implementation issues of these steps are detailed in the following subsections.

4.1. Creating the background mesh

Most CAD systems and kernels provide routines to tessellate a CAD model into a triangular mesh. In this study, this mesh is used as the background mesh of the sizing function. After the tessellation step, two additional data structures must be created, namely the *hybrid surface B-rep* **[7]** and the *parametric mesh*.

4.1.1 The hybrid surface B-rep. The surface B-rep is illustrated in Figure 3, where the input surface model includes three basic topology entities, i.e., *face*, *curve* and *point*. A *loop* is a specific topology entity that refers to a set of curves and limits the valid region of a face.

As shown in Figure 1b, the background mesh is a triangular mesh representing the input CAD model. This mesh is structured by three topology entities of different dimensions: *facets*, *edges* and *nodes*.

To connect the input CAD model and the background mesh, three basic mappings between the continuous topology entities and their discrete counterparts can be defined:

(1) *The face-facet mapping*. A face corresponds to a set of mesh facets.

(2) *The curve-edge mapping*. A curve corresponds to a set of mesh edges.

(3) *The point-node mapping*. A point corresponds to a mesh node.

Other mappings can be defined as well, e.g., between a curve and all nodes 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, and are not explicitly represented in the extended B-rep.

Here, we term the extended B-rep as the *hybrid surface B-rep*. In the standard B-rep, each basic topology entity corresponds to a continuous geometry entity. However, in the hybrid surface B-rep, each topology entity has two types of geometric representations in default. One representation corresponds to geometry entities defined on the continuous model, while the other representation describes a discrete model, such as a face defined by a set of triangles and a curve defined by a set of linear segments.

The following definition is introduced to describe the above mappings.

Definition 1 (Classification) [7, 25]. 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 ($d_i \leqslant d_j \leqslant 2$) G^{d_j} of the B-rep if M^{d_i} lies on G^{d_j}, denoted as M^{d_i} ô G^{d_j}. Namely, G^{d_j} is the *host entity* of M^{d_i}.

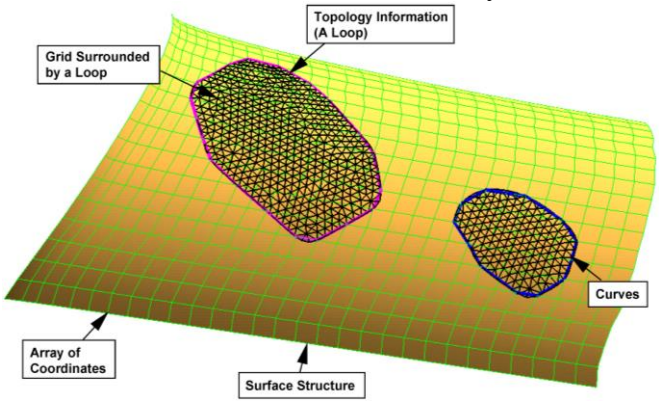


Figure 3. Illustration for the surface B-rep.

4.1.2 The parametric mesh. The projection of the background mesh on the parametric space of a curve or face is namely a *parametric mesh*. For a curve of the B-rep, the computation of its parametric mesh takes the following steps.

(1) Obtain the list of nodes and edges classified on the curve, denoted as N and E. They compose the *physical mesh* of the curve: $M_{\text{crv}} = \{N, E\}$.

(2) Replace physical coordinates of the nodes belonging to N by their counterparts in the parametric space of the curve. Define the new list of nodes as N'.

(3) Compose the *parametric mesh* of the curve as $M'_{\text{crv}} = \{N', E\}$.

Similarly, for a face of the B-rep, its parametric mesh needs to replace the physical coordinates of nodes classified on the face by their counterparts in the parametric space of the face.

The concept of parametric mesh is useful to speed up the meshing procedures for curves and faces by limiting the sizing-value queries employed in these meshing procedures on the parametric mesh of a particular surface or curve.

4.2. Initialising the sizing function

4.2.1 Computing sizing values adapted to curvatures. In highly curved regions, sizing values must be very small to avoid large gaps between the mesh and the geometry.

Figure 4 illustrates the geometric meaning of the sizing value adapted to the local curvature of a point S, denoted by h_c. Here, r_s refers to the curvature radius of S, $\mathbf{r}(t)$ refers to the principal curvature line that S lies in, and O refers to a point along the normal vector of S with $|OS| = r_s$. With O as the center, r_s as the radius and S as the contact point, an inscribed circle of $\mathbf{r}(t)$ is defined, and E is a point on the circle to make a central angle $\varphi_c = \angle SOE$. Then h_c equals the chord length between S and E, given by

$$h_c = |SE| = 2r_s \sin(\varphi_c / 2).$$

For a mesh node classified on more than one face, the principal curvatures of this node on all faces need to be computed. The value of r_s is computed by using the principal curvature having a maximal absolute value.

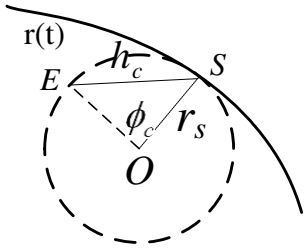


Figure 4. Geometric meaning of the size value adapted to local curvatures.

4.2.2 Computing sizing values adapted to proximities. Elements of small scales are required near narrow regions to avoid generating elements with high aspect ratios. However, computing proximity distances in narrow regions is non-trivial. Cunha *et al.* **[26]** and Zhu *et al.* **[27]** proposed to compute distances between all combinations of sampled entities, which involve a large number of unstable geometry computations. Based on the Cartesian mesh, a few more efficient algorithms were developed later, such as the wave propagation algorithm by Quadros *et al.* **[12]**, the geometry rasterisation algorithm by Deister *et al.* **[11]** and Voronoï cell based algorithm by Luo *et al* **[28]**. However, as unstructured background meshes are preferred in this study, an alternative scheme has to be developed.

In **[29]**, we developed a general algorithm that calculates surface proximity distances using an unstructured background mesh. The theoretical foundation supporting this algorithm is that the lines connecting the circumcenters of neighboring elements of a Delaunay triangulation are the discrete representation of the medial axis. This dual relation has a proof if the boundary edges of the model are all contained in the Delaunay triangulation **[30]**.

However, for those faces where the proximity distances are proximately equal everywhere, a simpler algorithm can be applied to compute a *global* proximity distance for each face:

$d_s = 2.0 \times A_s / l_s$,

where A_s and l_s are the area and circumference of the face, respectively. For background nodes classified on this face, their proximity-adapted sizing values are

$h_{d1} = d_s / \mu_d$,

where the user parameter μ_d defines the expected number of elements within the proximity distance.

Apart from the proximity distance between boundary curves, another proximity feature considered in this study is the curve length, i.e., the distance between the end points of a curve. Because at least one mesh segment is required to be generated in a curve, the sizing values of a background node classified on a curve must be less than the length of the curve. Assuming that h_{d0} is the smallest length of the host curves of a background node, the final proximity-adapted sizing value at this node is

$h_d = \min(h_{d1}, h_{d0})$.

4.2.3 User options. The role of user expertise is irreplaceable in some meshing tasks. The proposed algorithm provides the user with options to influence the element-sizing results via the following parameters:

(1) Global user parameters φ_c, μ_d and β;

(2) A local φ_c value for any curve or face;

(3) Local μ_d and β values for any face;

(4) A predefined sizing value or function for any geometry point, curve or face.

Only the first group of parameters is mandatory and the other parameters are optional. In most meshing tasks, including those to be demonstrated in Section 6, the user only needs to set the mandatory parameters. The initial sizing value at a background node is computed by combining the influence of geometry factors and user parameters:

$h_a = \max(h_{\min}, \min(h_d, h_c, h_u, h_{\max}))$,

where h_c and h_d are the sizing values adapted to local curvature and proximity features, respectively, h_u is the predefined sizing value by the user (if any); $h_{\min}$ and $h_{\max}$ are the user parameters that limit the minimal and maximal sizing values.

4.3. Smoothing the sizing function

The initial sizing function is smoothed by solving NLP 6. Among different numerical schemes for the solution of NLPs, the interior point method (IPM) **[31-33]** is adopted because of its good performance for problems with a large number of inequality constraints.

We first transform NLP 6 into a more general form, where the inequality constraints are rewritten as equality constraints by introducing slack variables ξ_i, η_i and ψ_i:

$$\begin{cases} \mathbf{H}_i^T \mathbf{K}_i \mathbf{H}_i - \ln^2(\beta) + \xi_i = 0, \text{ where } \xi_i \geq 0 \\ h_{\min} - h(p_i) + \eta_i = 0, \text{ where } \eta_i \geq 0 \\ h(p_i) - h_0(p_i) + \psi_i = 0, \text{ where } \psi_i \geq 0 \end{cases} \quad (i=1,2,...,m) \quad (i=1,2,...,n) .$$

Let

$\mathbf{x} = < x_1, x_2, \cdots, x_{3n+m} > = < h_1(p), ..., h_n(p), \xi_1, ..., \xi_m, \eta_1, ..., \eta_n, \psi_1, ..., \psi_n >$

be the extended solution vector, and

$$c_j(\mathbf{x}) = \begin{cases} \mathbf{H}_i^T \mathbf{K}_i \mathbf{H}_i - \ln^2(\beta) + \xi_i & 1 \le j \le m,\ i = j \\ h_{\min} - h(p_i) + \eta_i & m+1 \le j \le m+n,\ i = j-m \\ h(p_i) - h_0(p_i) + \psi_i & m+n \le j \le m+2n,\ i = j-m-n \end{cases},$$

NLP 6 can be reformulated to the new form as below,

$$\begin{aligned} \min \quad & f(\mathbf{x}) \\ s.t. \quad & c_j(\mathbf{x}) = 0 \quad j \in J = \{1, 2, \cdots, 2n+m\} \\ & x_i \ge 0 \qquad i \in I = \{n+1, n+2, \cdots, 3n+m\} \end{aligned}, \tag{7}$$

where

$$f(\mathbf{x}) = \sum_{i=1}^{n} (h(p_i) - h_0(p_i))^2 .$$

The solution of NLP 7 is reduced to a sequence of unconstrained optimization problems with decreasing barrier parameters μ ($\mu \ge 0$):

$$\min \quad \ell(\mathbf{x}, \boldsymbol{\lambda}, \mu) = f(\mathbf{x}) + \boldsymbol{\lambda}\mathbf{c}(\mathbf{x}) - \mu \sum_{i=n+1}^{3n+m} \ln(x_i), \tag{8}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^{2n+m}$ is the vector of Lagrangian multipliers, and $\mathbf{c}(\mathbf{x})$ is the vector of equality constraints $c_j(\mathbf{x}) = 0$ ($j \in J$). For a fixed μ, the solution of NLP 8 can be reduced to the calculation of the following perturbed Karush-Kuhn-Tucker (KKT) conditions **[32, 33]**:

$$\mathbf{F}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\upsilon}) = \begin{pmatrix} \nabla f(\mathbf{x}) + \nabla \mathbf{c}(\mathbf{x})\boldsymbol{\lambda} - \boldsymbol{\upsilon} \\ \mathbf{c}(\mathbf{x}) \\ \mathbf{X}\boldsymbol{\upsilon} - \mu \mathbf{e} \end{pmatrix} = \mathbf{0}, \tag{9}$$

where $\boldsymbol{\upsilon} = < \mu / x_{n+1}, ..., \mu / x_{3n+m} >$ is the dual vector, $\mathbf{e}$ is the vector of ones, and $\mathbf{X}$ is a diagonal matrix with its diagonal elements occupied by the values of x_i ($n+1 \le i \le 3n+m$). Numerically, a tolerance parameter ε is predefined to determine whether a solution satisfies the perturbed KKT conditions:

$$\| \mathbf{F}(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\upsilon}^*) \|_\infty \le \varepsilon . \tag{10}$$

Presently, the open source code IPOPT **[32, 33]** is adopted to accomplish the above numerical solution process. It adopts the Newton's method to compute Equation 9 iteratively, and a linear system needs to be solved in each iteration step. To reduce the computing time, IPOPT provides a parallel option for the solution of this linear system by using OpenMP. More implementation details of IPOPT can be found in **[32, 33]**.

For a general NLP, the above numerical procedure only returns a local optimal solution. However, for the NLP 6, the returned local optimal solution is also the global optimal solution according to Theorem 3.1. To be concise, the final solution that satisfies the perturbed KKT conditions is called *the optimal solution* hereafter. Nevertheless, it is observed that, if Equation 10 is used as the criterion to terminate the solution process of NLP 8, the computing time may beyond the user's expectation. To improve the computing time to an acceptable level, a new criterion is introduced by checking the values of two indices:

(1) For each background triangle, given the sizing values at its three ending nodes, a *real progressive factor* (β_i^{real}) can be computed by solving the following equation:

$$\mathbf{H}_i^T \mathbf{K}_i \mathbf{H}_i - \ln^2(\beta_i^{real}) = 0 . \tag{11}$$

Given a background mesh, the first index is the maximal value of real progressive factors, denoted by $\beta_{\max}^{real}$.

$\beta_{\max}^{real} = \max\{\beta_i^{real} \mid i = 1, 2, \cdots m\}$,

where m is the number of elements contained in the mesh.

(2) The second index is the predicted value of the mesh magnitude E_{num}, which is the sum of the computed number of elements for all background elements (see Appendix A). In each iteration of solving NLP 8, a value of E_{num} is computed, denoted by E_{num}^k, where k is the index numbering the iteration step.

The new criterion is defined in Equation 12:

$$\begin{cases} (\beta_{\max}^{real} - \beta_{\max}^{user}) / \beta_{\max}^{user} < 0.1 \\ | E_{num}^{k+1} - E_{num}^{k} | / E_{num}^{k} < 1.0e-3 \end{cases}. \quad (12)$$

It limits the difference of E_{num} computed in adjacent iteration steps and the difference of $\beta_{\max}^{real}$ and the maximal progress factor predefined by the user ($\beta_{\max}^{user}$). Note that the new criterion no longer strictly limits $\beta_{\max}^{real}$ to be smaller than $\beta_{\max}^{user}$.

5. The application of the sizing function

Once the sizing function is smoothed, it is ready for sizing-value queries by mesh generation algorithms. The efficiency of this query heavily impacts the efficiency of meshing algorithms because the number of such queries in a meshing procedure is usually several times larger than the number of final mesh nodes. In this section, we introduce the application of the proposed sizing-function in an advancing front surface mesher. The key contribution is the development of two efficient routines of sizing-value query.

Like most of prevailing surface meshers, our mesher exploits the parametric representation of input surfaces, and can be classified as mapping based **[34]**. Basically, the meshing algorithm follows a bottom-up procedure, i.e., it first meshes the curves and then meshes the faces individually. Meanwhile, the meshing procedures for both curves and faces are defined in the parametric spaces of curves and faces. This feature makes it possible to speed up the routines of sizing-value query employed in the meshing procedures for both curves and faces by using the concept of parametric mesh introduced in Section 4.1.2.

The first routine inputs the index of a curve and the parametric value (u_p) of a point p at this curve, and returns the sizing value at p by the following steps.

(1) Get the parametric mesh of the curve: $M'_{\text{crv}} = \{N', E\}$, where N' and E are lists of background nodes and edges classified on the curve.

(2) Visit E to find an edge e_i containing p. Assuming the ending nodes of e_i are p_i and p_{i+1} and the parametric values of these two nodes are u_i and u_{i+1} ($u_i < u_{i+1}$), there must be $u_i \le u_p \le u_{i+1}$.

(3) The sizing value at p is a linear interpolation of the sizing values at p_i and p_{i+1}.

Following a similar flowchart, another routine is developed to query the sizing value at a point p located on a face.

(1) Get the parametric mesh of the face: $M'_{\text{fac}} = \{N', E, T\}$, where N', E and T are lists of background nodes, edges and triangles classified on the face.

(2) Visit T to find a triangle t_i containing p.

(3) The sizing value at p is a linear interpolation of the sizing values at the end nodes of t_i.

With the aid of the concept of parametric mesh, the base-element search step (Step 2) of both routines only visits the elements classified on a curve or a face, and moreover, the search occurs in the parametric space of the curve or face instead of the physical space. However, if the second routine is implemented as the above flowchart, it can be very time-consuming because the parametric mesh of a large face may contain thousands of triangles. The timing

performance can be further improved by using the spatial locality, i.e. the fact that the base elements for two geometrically neighboring positions are usually located very closely (or even the same). A much faster base-element search procedure can be developed by employing the walk-through algorithm to find the base element at the shortest path and inputting one more parameter to the walk-through algorithm that refers to a good enough guess for the base element. In the current implementation, a walk-through algorithm suggested by Shan *et al.* is adopted to search the base element **[18]**, which requires no auxiliary structures but the initial guess for the base element and neighboring indices of candidate triangles.

The remaining issue is how to set a good initial guess for the base element. Taking the advancing front surface mesher considered in this study as the example, it meshes a face by repeating the advancing front step, where a front is first selected and a new node is then created. Obviously, when querying the base element of the new node, either of the base elements of the front nodes can be chosen as the initial guess.

6. Numerical results

The tests presented here are conducted on a personal computer (CPU: 3.5GHz; Memory: 24GB). Three CAD models are selected as inputs to analyze the performance of the proposed algorithm. The original geometry files of these models are accessible through the Internet. The London Tower Bridge (referred to as *Bridge* hereafter) **[35]** is the test case geometry from the meshing contest session of the 23rd International Meshing Roundtable. The F6 aircraft model **[36]** is the test case geometry from the 2nd AIAA CFD Drag Prediction Workshop. The F16 aircraft model (referred to as *F16* hereafter) is obtained from GrabCAD (www.grabcad.com). Figures 1, 5 and 6 show the input CAD models, the background meshes, the initial sizing functions and their smoothed counterparts for the F6, F16 and Bridge models, respectively. All of these sizing functions are initialised and smoothed by using the default user parameters, i.e. $\varphi_c = 10^\circ$, $\mu_d = 2$ and $\beta = 1.2$. Meanwhile, the parallel option of the numerical scheme for sizing-function smoothing is enabled in the test cases and 4 computer cores are employed to solve Equation 9. Figures 7~9 render the surface meshes of the three models and some close up views. Table 1 lists the main statistics of these tests.

Table 1. The main performance statistics of the tests conducted in this study.

Index type	Index name	F6	F16	Bridge
Basic indices	#Geometry Curves	97	1,488	9,873
	#Geometry Faces	35	604	3,014
	#Background nodes	4,016	24,358	43,354
	#Background elements	7,853	48,209	88,336
	#Surface mesh nodes	24,107	103,209	695,827
	#Surface mesh elements	47,819	205,066	1,393,282
Timing data (unit: second)	Creating the sizing-function	13.6	78.6	303.0
	Creating the background mesh	4.99	15.94	17.1
	Initialising the sizing function	0.01	0.06	0.1
	Smoothing the sizing function	8.6	62.6	285.8
	Generating the surface mesh	9.1	42.9	233.3
	Querying the sizing values	0.6	2.4	20.0

We used to configure grid sources to define the sizing function under the user interface of our in-house pre-processing system **[37]**. The configurations adopted for the F6 and F16 models include 34 sources and 131 sources, respectively. The manual process of creating grid sources are time consuming: half an hour of interaction time is required for the F6 case, while this timing cost increases up to many hours for the F16 case. By contrast, the proposed

algorithm only consumed 13.6 and 78.6 seconds to generate the two sizing functions in a fully automatic fashion, of which the step of sizing-function smoothing dominates, consuming 8.6 and 62.6 seconds for the F6 and F16 cases, respectively. For the Bridge case, although we never attempt to control element scales by grid sources considering the possible huge amount of interaction time, it is estimated that hundreds of grid sources are required to capture the abundant geometry features of this model and to control the gradation of element scales. By contrast, the proposed algorithm consumed about 5 minutes to create a high-quality sizing function, of which the step of sizing-function smoothing consumes 285.6 seconds. It is evident that the proposed algorithm substantially enhances the automation of element-sizing specification over the method of using grid sources.

Apart from the timing data of sizing-function creation, Table 1 also lists the timing data of surface mesh generation. The advancing front mesher consumed 9.1, 42.9 and 233.3 seconds to handle the F6, F16 and Bridge models, respectively. The velocity values, referring to how many elements the mesher can create per second, are 5,254, 4,780 and 5,972 respectively. It varies within a reasonable range considering the timing performance of a surface mesher always depends on the input CAD models to some extent.

The total timing data consumed by sizing-value queries are presented in Table 1 as well. In the three test cases, these queries at most consume 8.6% of the meshing time, owing to the improved strategies suggested in Section 5. The base-element search is the most time-consuming step for sizing-value query of a surface point, and its timing performance is improved remarkably by the proposed walk-through algorithm. In average, each calling of this procedure visits about 8, 4 and 4 background triangles in the meshing processes of the F6, F16 and Bridge models, respectively.

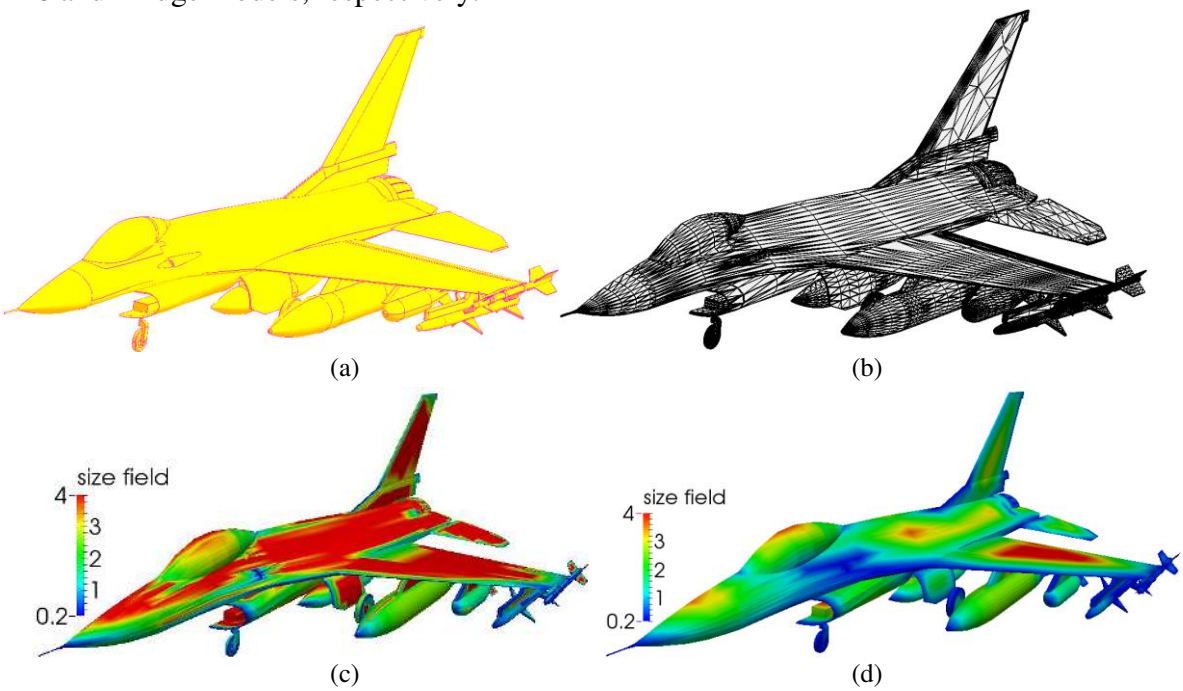
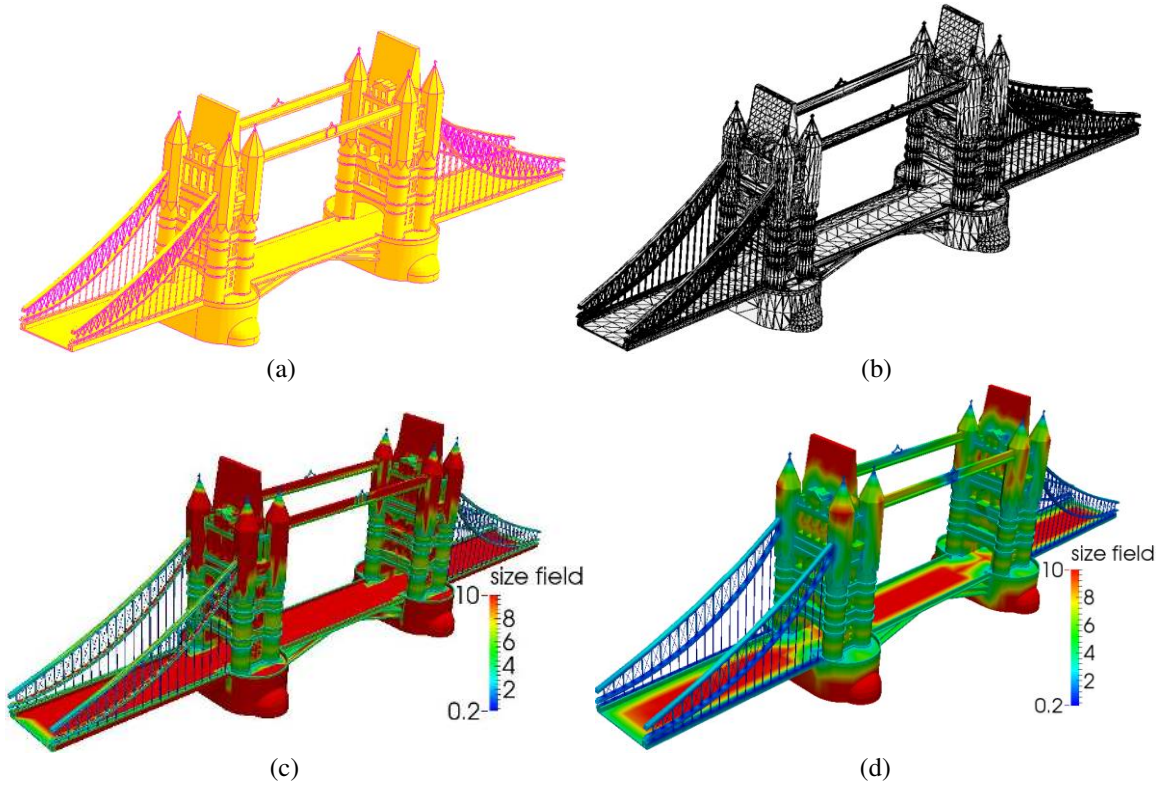
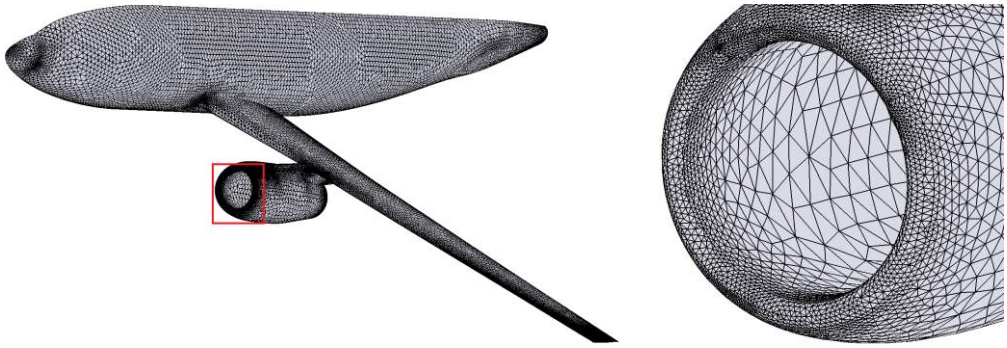


Figure 5. An illustration for the proposed algorithm using an F16 aircraft model. (a) The input CAD model. (b) The background mesh. (c) The initial sizing function. (d) The smoothed sizing function.



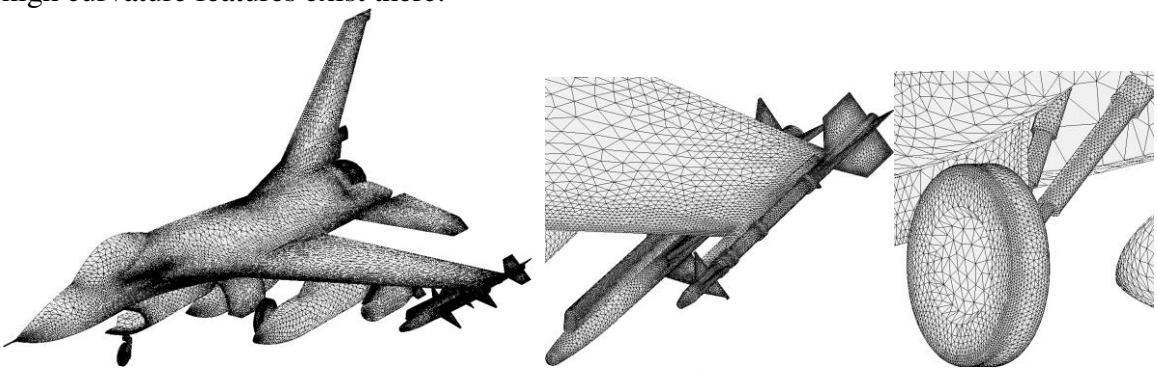
(a) (b) (c) (d)

Figure 6. An illustration for the proposed algorithm using a London Tower bridge model. (a) The input CAD model. (b) The background mesh. (c) The initial sizing function. (d) The smoothed sizing function.



(a) (b)

Figure 7. The surface mesh of the F6 model. (a) The overall mesh. (b) The mesh details of the engine intake lip. A very fine mesh resolution is observed in the engine intake lip because high curvature features exist there.



(a) (b) (c)

Figure 8. The surface mesh of the F16 model. (a) The overall mesh. (b) The mesh details near a missile. (c) The mesh details near the landing gear.

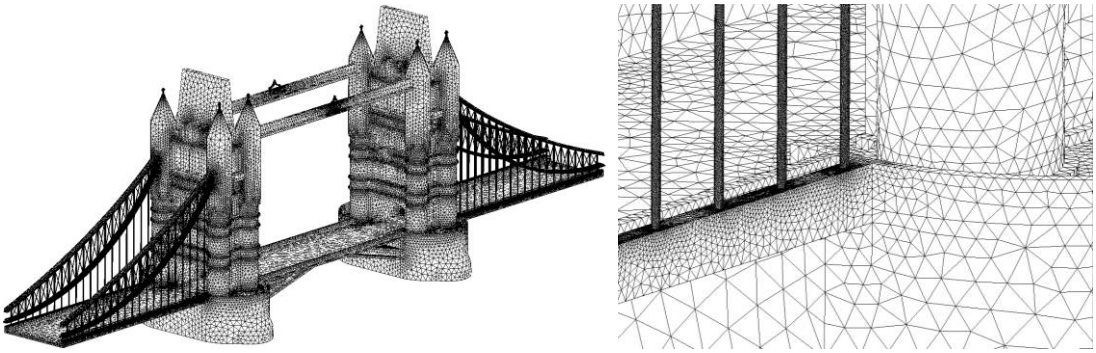
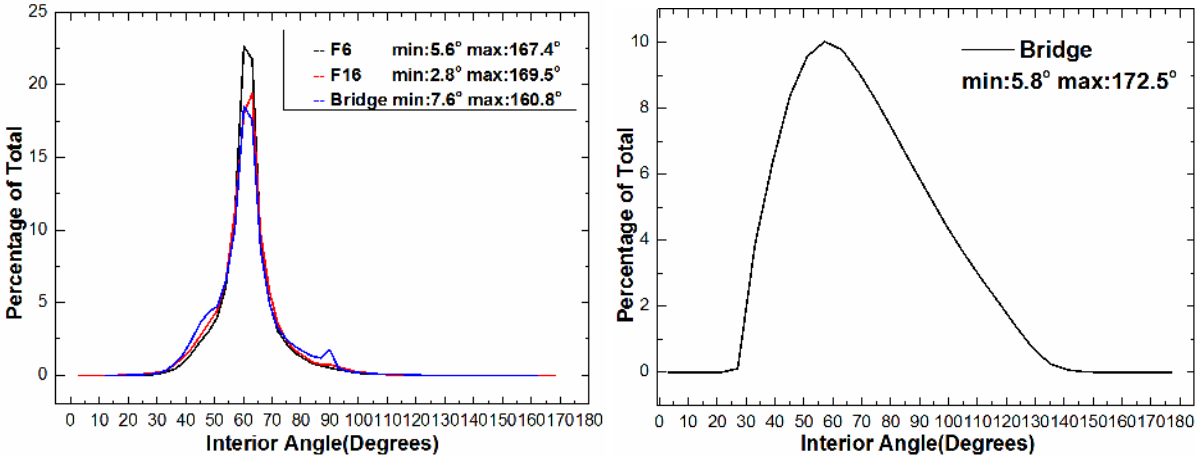


Figure 9. The surface mesh of the Bridge model and its local mesh details.

Another focus of the tests is on the mesh quality. Figure 10a draws the distributions of interior angles of surface elements for the three surface meshes shown in Figures 7~9. Note that the worst elements in a mesh have far more influence than the average elements in numerical simulations. To evaluate those worst elements, a triangle is classified as a *low-quality* element if its minimal angle is smaller than 24 degrees or as a *bad element* if its minimal angle is smaller than 12 degrees. For the surface meshes shown in Figures 7~9, the numbers of bad elements are 11, 41 and 5, respectively, and those of low-quality elements are 82, 1,283 and 3,121 respectively. The percentages of low quality elements are 0.17, 0.63 and 0.22, respectively, and the percentages of bad quality elements are even much smaller.

To verify the quality of the resulting surface mesh further, a tetrahedral mesh is generated from the surface mesh of the Bridge model. Figure 10b draws the distribution of interior angles of tetrahedral elements. The volume mesh contains 33,698,234 elements and is generated in parallel on 32 computer cores **[4, 38]**. Therefore, elements generated on different computer cores are painted in different colours in Figure 11. Owing to the input of a high-quality surface mesh, the tetrahedral mesh is of high quality as well. The minimal and maximal angles are 5.8° and 172.5°, respectively. The percentages of angles below 30° and over 150° are only 0.76 and 0.003, respectively.



(a) (b)

Figure 10. Mesh quality statistics. (a) Distributions of the interior angles of the three surface meshes. (b) Distributions of the interior angles of a volume mesh of the Bridge model.

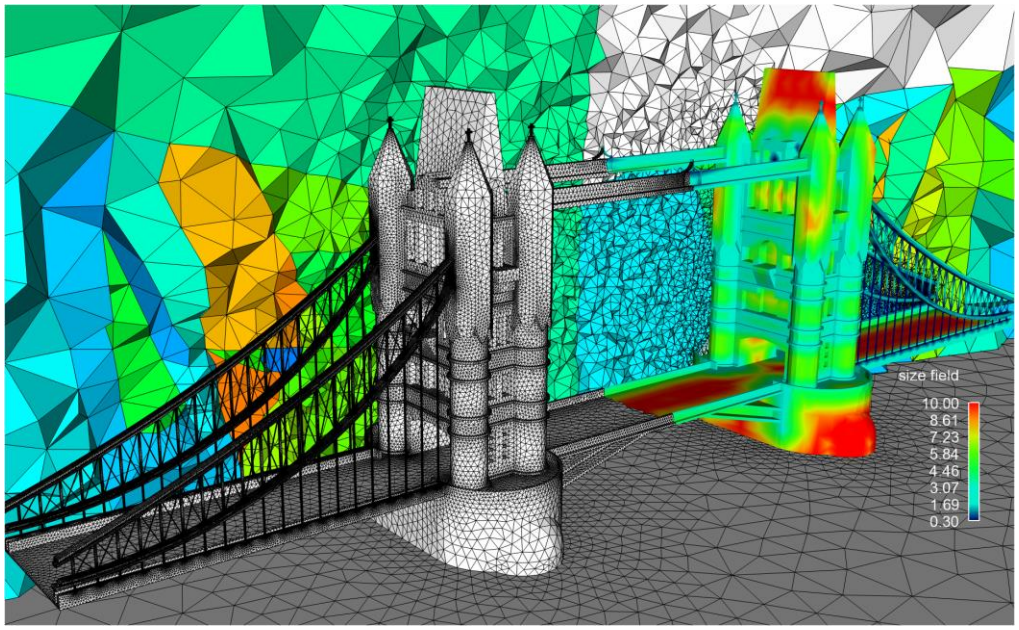


Figure 11. The surface mesh and the volume mesh of the London Tower Bridge model. The volume mesh is generated in parallel on 32 computer cores.

The final focus is on the performance of the proposed numerical scheme for sizing-function smoothing. It is observed that this numerical scheme always converges to a global optimal solution in the three test cases, i.e., a solution that meets Equation 10. However, the timing costs to achieve the optimal solutions are very high. Note that the proposed numerical scheme is composed of two-level loops. In the outer loop, the solution of NLP 7 is reduced to a sequence of unconstrained optimization problems (NLP 8) with decreasing barrier parameters. In the inner loop, NLP 8 is solved by computing Equation 9 iteratively. To evaluate the convergence of the numerical scheme, the iteration steps computing Equation 9 are numbered consecutively. For the F6, F16 and Bridge models, the numerical scheme converges after 2,002, 6,851, 1,7189 iteration steps, respectively, and the computing time is 40.9, 920.6 and 3023.7 seconds, respectively. After introducing the new termination criterion (Equation 12), the numerical scheme stops after 617, 780, 1,759 iteration steps, respectively. Accordingly, the timing costs reduces to 8.6, 62.6 and 285.8 seconds, respectively.

It is emphasized that the quality of the sub-optimal solutions corresponding to the new termination criterion is comparable to the quality of the global optimal solutions. To demonstrate this, the difference of sizing value at a background node is computed by the following equation:

$$s_{diff} = | s_{opt} - s_{sub} | / s_{opt},$$

where s_{opt} and s_{sub} are the optimal and sub-optimal sizing values at this node, respectively. To show more details of the difference of sizing function, the two norms ℓ_2-*norm* and ℓ_∞-*norm* are calculated as below:

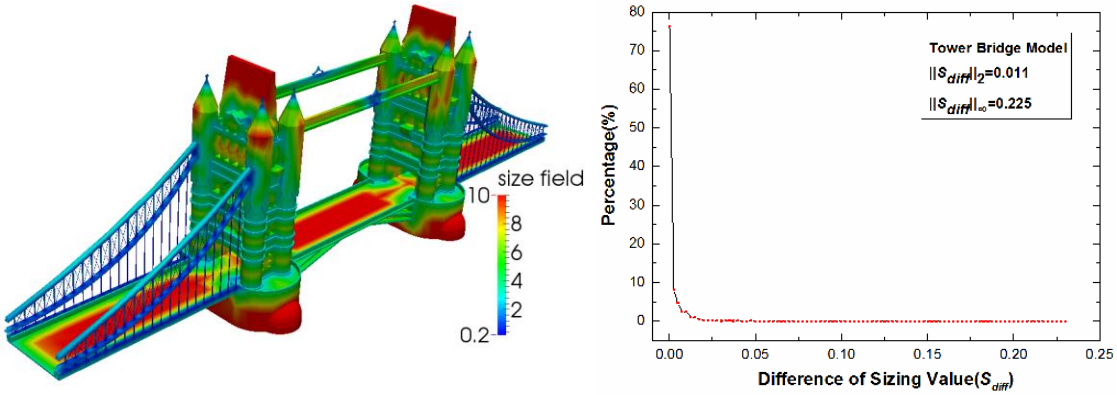
$$\| s_{diff} \|_2 = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(s_{diff}^{i})^2};$$

$$\| s_{diff} \|_\infty = \max(s_{diff}^{i}) \ (i=1,2,...,n).$$

These two norms represent the average difference and the maximum difference, respectively.

Figure 12 presents the optimal sizing function of the Bridge model and the distribution of the difference of this sizing function and the one shown in Figure 6b (the sub-optimal sizing function). For this case, $\| s_{diff} \|_2 = 0.011$, and $\| s_{diff} \|_\infty = 0.225$. Further analysis reveals that the

number of nodes with $s_{diff} \geq 0.1$ is 108, accounting for only 0.23% of the total number of background nodes. Because of the minor difference of the optimal and sub-optimal sizing-functions of the Bridge model, the values of E_{num} adapted to these two sizing functions are also very close: they are 1,383,884 and 1,356,807, respectively. Meanwhile, it is observed that the value of E_{num} adapted to the sub-optimal sizing function is also very close to the number of elements of the finally generated surface mesh shown in Figure 9, which is 1,393,282, as reported in Table 1. It proves that the enhanced surface mesher is capable of respecting the predefined sizing function very accurately.



(a) (b)

Figure 12. Comparison between the optimal and sub-optimal sizing functions for the Bridge model. The sub-optimal sizing function is already rendered in Figure 6d, and (a) is the optimal sizing function. (b) draws the distribution of the differences between the node values that define the two sizing functions.

Meanwhile, Figures 13a~c detail the variations of $\beta_{\max}^{real}$ and E_{num} with the iteration steps (see Section 4.4 for the definitions of the two indices). For all three test cases, a stable decrease of E_{num} is observed. It means a positive correlation exists between the minimization function used in NLP 6 and the goal to minimize the mesh magnitude. Section 3.2 explained this correlation qualitatively. In Figures 13a~c, a rather stable decrease of $\beta_{\max}^{real}$ is also observed, apart from a sharp increase occurred in the solution process of the Bridge case. This explains why the new termination criterion (Equation 12) must also limit the value $\beta_{\max}^{real}$.

Figure 13d presents the distributions of the real progressive factors (β_i^{real}, computed by Equation 11 on each background element) in the finally smoothed sizing functions of the three test cases. The *accumulative ratio function* $r_\beta(x)$ $(x \geq 1.0)$ refers to the percentage of background elements whose β_i^{real} values are below x. It is shown that only a very small fraction of background elements have progress factors above the user parameter β ($\beta = 1.2$ in the tests). If the background elements with $\beta_i^{real} \leq 1.02$ and $\beta_i^{real} \geq 1.1$ are classified as *low-gradation* and *high-gradation* elements, respectively. It is observed that the background meshes for the F6 and F16 models contain the highest percentage of low-gradation and high-gradation elements, respectively. The percentage values are 23.6% and 75.9%, respectively. Since the sizing functions considered in this study are mainly adapted to the geometric features of the input CAD models, this observation reveals that the F6 model has fewer geometry features than the F16 model.

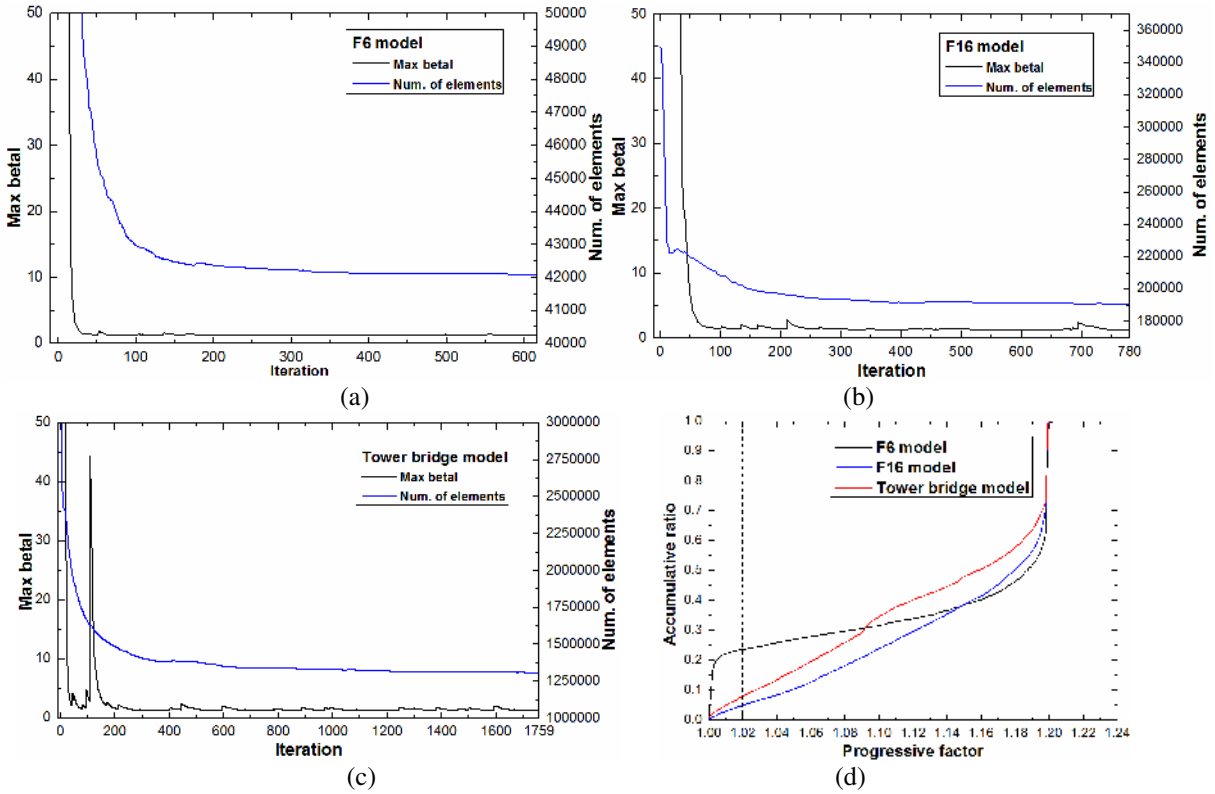


Figure 13. (a)~(c) detail the variations of the indices $\beta_{\max}^{real}$ and E_{num} with the numbers of iteration steps experienced in the sizing-function smoothing procedures. (d) draws the curves of the accumulative ratio function $r_{\beta}(x)$ $(x \geq 1.0)$, which refers to the percentage of background elements with β_i^{real} values (computed by Equation 11) below x.

7. Conclusions

Surface mesh generation is a time-consuming step when complex aerodynamics models are considered. A main performance bottleneck lies in the element-sizing specification procedure. The conventional grid sources based scheme involves intensive manual labors. In this study, an automatic scheme is proposed for the element-sizing specification of unstructured surface mesh generation. Different with existing Cartesian mesh based schemes, the proposed algorithm adopts unstructured triangular mesh as the background mesh. Experiments of complicated aerodynamics configurations show that our algorithm can automatically produce a suitable size map in minutes. By contrast, a conventional grid sources based scheme may need many hours by a tedious manual interaction process.

Due to the topological flexibility of unstructured meshes, the proposed algorithm generates a far coarser background mesh than existing Cartesian mesh based schemes for the problem of unstructured surface mesh generation. Nevertheless, if volume meshing is considered, the surface background mesh is not suitable any more. A possible approach is to use the surface background mesh as inputs to construct a volume mesh. The convex NLP model set up for surface problems (NLP 6) can be naturally extended for volume problems.

Meanwhile, a particular interest of our ongoing research is to extend NLP 6 for anisotropic meshing problem by introducing the Riemannian metric.

Acknowledgments

The authors would like to thank the support from the National Natural Science Foundation of China (Grant Nos. 11172267, 11432013, 10872182 and 11171305), Zhejiang Provincial Natural Science Foundation of China (Grant No. Y1110038). The first author acknowledges the joint support from Zhejiang University and China Scholarship Council and the host of Professor Oubay Hassan and Professor Kenneth Morgan for his visiting research at Swansea University, UK. A preliminary version of this paper was published in the 23th International Meshing Roundtable conference **[39]**. The authors appreciate the valuable comments and constructive suggestions from the anonymous reviewers who served that conference.

Appendix A

Given a triangle F with end nodes $p_i (i = 0 \sim 2)$, the sizing value of a point p inside F is:

$$h(t_0, t_1) = \sum_{i=0}^{2} t_i h_i = t_0 h_0 + t_1 h_1 + (1 - t_0 - t_1) h_2 ,$$

where $h_i (i = 0 \sim 2)$ are the sizing values at nodes $p_i (i = 0 \sim 2)$, $t_i (i = 0 \sim 2)$ are the natural coordinates of p:

$t_i = \text{Area}(F_i) / \text{Area}(F) = A_i / A$.

If quadrilateral elements are generated inside F, the estimated element number is:

$$n_Q = \int_{\Omega_F} \frac{1}{h^2(t_0, t_1)} d\Omega , \tag{A1}$$

where Ω_F is the region bounded by F. Equation (A1) can be written in the natural coordinate system as:

$$n_Q = 2A_F \int_0^1 \int_0^{1-t_1} \frac{1}{h^2(t_0, t_1)} dt_0 dt_1 . \tag{A2}$$

Solving Equation (A2), the final expression of n_Q can be obtained as

$$n_Q = \begin{cases} A / {h_0}^2 & h_0 = h_1 = h_2 \\ \dfrac{2A}{{\Delta_1}^2} (\ln \dfrac{h_1}{h_0} + \dfrac{\Delta_0}{h_1}) & h_0 \neq h_1, \text{ and } h_1 = h_2 \\ \dfrac{2A}{\Delta_0 \Delta_1 \Delta_2} \sum_{i=0}^{2} (\Delta_i \ln h_i) & h_0 \neq h_1 \neq h_2 \end{cases} , \tag{A3}$$

where $\Delta_0 = h_2 - h_1$, $\Delta_1 = h_0 - h_2$ and $\Delta_2 = h_1 - h_0$.

If triangular elements are desired, the estimated element number is:

$$n_\Delta = 2 n_Q . \tag{A4}$$

References

1. Baker TJ. Mesh generation: art or science? *Progress in Aerospace Sciences* 2005; **41**:29-63.
2. Weatherill NP, Hassan O, Morgan K, Jones JW, Larwood BG, Sorenson K. Aerospace simulations on parallel computers using unstructured grids. *International Journal for Numerical Methods in Fluids* 2002; **40**:171–187.
3. Löhner R. Recent advances in parallel advancing front grid generation. *Archives of Computational Methods in Engineering* 2014; **21**:127-140.
4. Chen J, Zhao D, Huang Z, Zheng Y, Wang D. Improvements in the reliability and element quality of parallel tetrahedral mesh generation. *International Journal for Numerical Methods in Engineering* 2012; **92**: 671-693.

5. Zhao D, Chen J, Zheng Y, Huang Z, Zheng J (2014) Fine-grained parallel algorithm for unstructured surface mesh generation. *Computers & Structures*, 2015; **154**:177-191.
6. Shimada K. Current issues and trends in meshing and geometric processing for computational engineering analyses. *Journal of Computing and Information Science in Engineering* 2011; **11**:1 -13.
7. 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. *Advances in Engineering Software*, 2015, **86**:55-69.
8. Pirzadeh, SZ. Structured background grids for generation of unstructured grids by advancing-front method. *AIAA journal* 1993; **31**:257-265.
9. Kania LK, Pirzadeh, SZ. A geometrically-derived background function for automated unstructured mesh generation. *Proceedings of the 17th AIAA Computational Fluid Dynamics Conference*, Toronto, Canada, 2005; AIAA 2005-5240.
10. *Pirzadeh, SZ.* Advanced unstructured grid generation for complex aerodynamic applications. *AIAA Journal* 2010; **48**:904-915.
11. Deister F, Tremel U, Hassan O, Weatherill NP. Fully automatic and fast mesh size specification for unstructured mesh generation. *Engineering with Computers* 2004; **20**:237-248.
12. Quadros W, Vyas V, Brewer M, Owen SJ, Shimada K. A computational framework for automating generation of sizing function in assembly meshing via disconnected skeletons. *Engineering with Computers* 2010; **26**:231-247.
13. Zheng Y, Weatherill NP, Turner-Smith EA. Interactive geometry utility environment for multi-disciplinary computational engineering. *International Journal for Numerical Methods in Engineering* 2002; **53**:1277-1299.
14. Owen SJ, Saigal S. Neighborhood based element sizing control for finite element surface meshing. *Proceedings of 6th International Meshing Roundtable*, 1997:143–154
15. Borouchaki H, Hecht F, Frey PJ. Mesh gradation control. *International Journal for Numerical Methods in Engineering* 1998; **43**:1143-1165.
16. Pippa S, Caligiana G. GradH-Correction: guaranteed sizing gradation in multi-patch parametric surface meshing. *International Journal for Numerical Methods in Engineering* 2005; **62**:495-515.
17. Persson P-O. Mesh size functions for implicit geometries and PDE-based gradient limiting. *Engineering with Computers*, 2006; **22**:95-109.
18. Shan J, Li Y, Guo Y, Guan Z. A robust backward search method based on walk-through for point location on a 3D surface mesh. *International Journal for Numerical Methods in Engineering* 2008; **73**:1061-1076.
19. Hecht F, Mohammadi B. Mesh adaption by metric control for multi-scale phenomena and turbulence. AIAA Paper, 97-0859, 1997.
20. Alauzet F. Size gradation control of anisotropic meshes. *Finite Elements in Analysis and Design*, 2010; 46: 181-202.
21. Qiu G. Analytical geometry (Vol. 2). Beijing: Higher Education Press. 1957. pp. 201-204.
22. Делоне БН, Райков ДН, Аналитическая геометрия II, государственное издательство технико-теоретической дитературы, 1949.
23. Polyanin AD, Chernoutsan AI. A concise handbook of mathematics, physics, and engineering sciences. London: CRC Press. 2010. pp. 99-101.
24. Stephen B, Vandenberghe L. Convex optimization (Seventh printing with corrections). Cambridge: Cambridge University Press. 2009.
25. Dey S, Shephard MS, Flaherty JE. Geometry representation issues associated with p-version finite element computations. *Computer Methods in Applied Mechanics and Engineering* 1997; **150**:39-55.
26. Cunha, A, Canann SA, Saigal S. Automatic boundary sizing for 2D and 3D meshes. P*roceedings of the AMD Trends in Unstructured Mesh Generation, ASME*, Evanston, IL, USA, 1997; 65-72.
27. Zhu J, Blaker T, Smith R. Background overlay grid size functions. *Proceedings of the 11th International Meshing Roundtable*, Ithaca, NY, USA, 2002; 65-74.
28. Luo X, Shephard MS, Yin L, O'Bara RM, Nastasia R, Beall MW. Construction of near optimal meshes for 3D curved domains with thin sections and singularities for p-version method. *Engineering with Computers* 2010; **26**:215-229.
29. Xie L, Chen J, Liang Y, Zheng Y. Geometry-based adaptive mesh generation for continuous and discrete parametric surfaces. *Journal of Information & Computational Science*, 2012; **9**:2327-2344.
30. Frey PJ, George PL. Mesh generation: application to finite elements. Oxford: HERMES Science Publishing. 2000.
31. Nesterov Y. Nemirovskii A, Ye Y. Interior-point polynomial algorithms in convex programming. Philadelphia: Society for Industrial and Applied Mathematics. 1994.
32. Wächter, A. An Interior Point Algorithm for Large-Scale Nonlinear Optimization with Applications in Process Engineering. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, January 2002.
33. Wächter A., Biegler L.T., On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming, *Mathematical Programming*, 2006; **106**:25-57.
34. Peirò J. Surface grid generation. In: Thompson JF, Soni BK, Weatherill NP (eds) Handbook of Grid Generation. CRC Press, Inc., Boca Raton, FL, USA, Chapter 19, 1999.
35. Meshing Contest of the 23rd International Meshing Roundtable. Jun-16-2015. URL: http://imr.sandia.gov/23imr/MeshingContest.html.
36. NASA DLR-F6 Geometry. Jun-16-2015. URL: http://aaac.larc.nasa.gov/tsab/cfdlarc/aiaa-dpw/Workshop2/DLR-F6-geom.html.
37. Xie L, Zheng Y, Chen J, Zou J. Enabling technologies in the problem solving environment HEDP. *Communications in Computational Physics*, 2008; **4**:1170-1193.

38. Chen J, Zhao D, Huang Z, Zheng Y, Gao S. Three-dimensional constrained boundary recovery with an enhanced Steiner point suppression procedure. *Computers and Structures* 2011; **89**: 455-466.
39. Xiao Z, Chen J, Zheng Y, Zeng L, Zheng J. Automatic unstructured element-sizing specification algorithm for surface mesh. *Procedia Engineering*, 2014; 82:240-252.