Advances in Aerodynamics

RESEARCH Open Access

ISpliter: an intelligent and automatic surface mesh generator using neural networks and splitting lines

Zengsheng Liu[1†], Shizhao Chen[1†], Xiang Gao[1,2,3*], Xiang Zhang[1,2,3], Chunye Gong[2,3], Chuanfu Xu[1,2,3] and Jie Liu[2,3]

†Zengsheng Liu and Shizhao Chen share co-first authorship.

*Correspondence: gaoxiang12@nudt.edu.cn

[1] State Key Laboratory of High Performance Computing, National University of Defense Technology, Changsha 410073, China
[2] College of Computer, National University of Defense Technology, Changsha 410073, China
[3] Laboratory of Digitizing Software for Frontier Equipment, National University of Defense Technology, Changsha 410073, China

Abstract
In this paper, we present a novel surface mesh generation approach that splits B-rep geometry models into isotropic triangular meshes based on neural networks and splitting lines. In the first stage, a recursive method is designed to generate plentiful data to train the neural network model offline. In the second stage, the implemented mesh generator, ISpliter, maps each surface patch into the parameter plane, and then the trained neural network model is applied to select the optimal splitting line to divide the patch into subdomains continuously until they are all triangles. In the third stage, ISpliter remaps the 2D mesh back to the physical space and further optimizes it. Several typical cases are evaluated to compare the mesh quality generated by ISpliter and two baselines, Gmsh and NNW-GridStar. The results show that ISpliter can generate isotropic triangular meshes with high average quality, and the generated meshes are comparable to those generated by the other two software under the same configuration.

Keywords: Surface mesh generation, Artificial neural network, Splitting line, Triangular element, Feature extraction

1 Introduction

For modern numerical simulations like computational fluid dynamics and computational structural mechanics, surface mesh serves as a key bridge connecting Computer-Aided Design (CAD) models and numerical algorithms. The surface meshing process aims to generate a discrete mesh that uses triangular/quadrilateral elements to approximate the given CAD model. Subsequently, the volumetric mesh generation and numerical algorithms are performed based on it. The quality and efficiency of surface mesh generation are important factors that affect subsequent volumetric mesh generation and the results of numerical simulations. Therefore, it is of great practical value to develop fully automatic and high-quality surface meshing technology to promote the development of scientific engineering computing.

Although various mesh generation algorithms have developed in recent 30 years, automatically generating correct and satisfying meshes for complex models is still a big challenge [1, 2]. Among all types of surface meshes, triangular meshing attracts the most attention because of its simplicity and flexibility. Its mainstream approaches can be divided into two categories: algorithms based on Delaunay Triangulation (DT) [3], and algorithms based on Advancing Front Technique (AFT) [4]. The Delaunay method has excellent mathematical properties. It maximizes the minimum angle of the triangulation for a given point set. For this reason, Delaunay triangulation is often called the MaxMin triangulation. It also ensures that no other points are included in the circumcircle of any triangular element [5]. However, the Delaunay-based method encounters difficulties in recovering surface boundaries [6]. In contrast, the AFT method starts from the boundary of the surface and gradually generates elements inward. This method can better maintain the integrity of the boundaries and the mesh quality near boundaries is guaranteed. However, the AFT-based method generates only one element at each step, which is inefficient, and colliding fronts are easy to appear [7]. From another perspective, surface meshing can also be divided into direct methods [8] and mapping methods [9]. Direct methods generate mesh directly on the 3D surface in the physical space, while mapping methods first perform 2D meshing approaches in the parametric space, and then project the mesh back to the physical space. In order to synthesize the advantages of these two methods, a lot hybrid approaches have been proposed in recent studies [10, 11]. Nevertheless, generating engineering-practical meshes still requires a lot of manual labor. The level of automation and intelligence in this field needs to be improved.

In recent years, the vigorous development of Artificial Intelligence (AI) technologies such as deep learning has brought new impetus to the mesh generation research. For example, to predict the local mesh density throughout the domain before meshing, Z. Zhang et al. [12] presented an artificial neural network named MeshingNet to guide a standard mesh generator, and further extended it to 3D tetrahedral mesh generation [13]. L. Zhang et al. [14] used a neural network model as the point selection strategy for the AFT method to replace complex calculations. The model can obtain the target point by inputting the coordinates of the reference front and template points. Subsequently, they further developed an anisotropic hybrid meshing technique for viscous flow simulations by training two AI models, which were used to predict the advancing direction and control the element size, respectively [15]. J. Liu et al. [16, 17] proposed an automatic framework based on convolutional neural networks to evaluate the overall mesh quality for 2D/3D structured meshes, which helps the generator to identify good meshes. X. Liu et al. [18] developed a mesh optimization method embedded a machine learning regression model into the variational mesh adaptation, which can automatically move the mesh points to the domains where the flow field varies drastically. To make the tetrahedral meshing easier, S. Owen et al. [19] used machine learning approaches to defeature CAD models by predicting mesh quality for geometric features before meshing. In the field of 3D reconstruction, there are also many studies using deep learning to generate meshes [20, 21].

In this paper, we focus on generating isotropic triangular surface meshes from B-rep CAD models, and develop a mesh generator named ISpliter based on artificial neural networks and the idea of splitting line [22, 23]. Specifically, the main contributions of this paper are summarized as follows:

- Extending the 2D splitting line method to surface mesh generation, and a machine learning framework is proposed to select the best splitting line. Furthermore, the algorithm can be easily generalized to quadrilateral surface meshing.
- A novel and promising surface mesh generator that could handle industrial CAD models is developed. With proper detection of the discrete surface loops, it can remove unnecessary small geometrical features automatically.
- In terms of skewness and minimum angle to evaluate the mesh quality, the presented mesh generator is compared with two well-known software through several typical cases, and satisfactory results have been achieved.

2 Overview

The input of the proposed algorithm is a standard CAD model, which is represented by a collection of trimmed parametric surface patches, $G = \{P_i\}_{i=1}^{m}$. Each patch P_i is a B-spline surface represented by an analytically defined mapping, denoted as $S(u, v) = (x(u, v), y(u, v), z(u, v))$, from a bounded two-dimensional domain called parametric space, into the three-dimensional physical space. m is the number of patches. On each patch, there are one or several loops, each of which is defined by a closed and ordered set of curves. Moreover, if a patch has more than one loop, it means there are holes on the surface patch. Our approach can only deal with clean geometries at present, which means the common boundary of adjacent patches is the same curve stored in the geometry data structure. For dirty CAD models with gaps, overlaps, or topological loss, another topic about CAD fixing before mesh generation needs to be studied. In our approach, we first generate meshes for all boundary curves, and then the surface meshing is processed patch by patch. Therefore, new mesh points and elements are directly added to the global mesh data structure, and finally the complete object surface can be formed.

The output of the algorithm is the generated surface mesh, $M = \{t_i\}_{i=1}^{n}$. Where t_i is each triangular element, and n is the number of mesh elements. The proposed algorithm requires to specify only one parameter, L_{max}, as the maximum edge length of the target mesh. In addition, specifying the desired number of mesh points on the curves is supported. As presented in Fig. 1, the workflow of our algorithm is divided into the following five steps:

(a) Geometry reading. At first, the B-rep type geometry G is read in. Some preprocessing such as face normal/area calculation and validity check is done with third-party CAD engines.

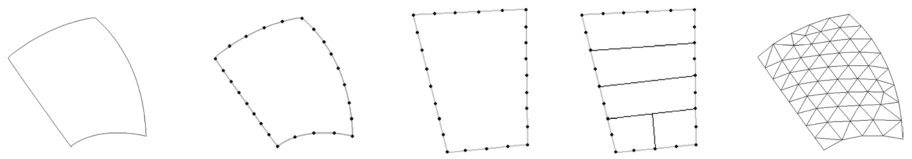
(a) Input geometry (b) Loop sampling (c) Point mapping (d) Polygon splitting (e) Mapping back

Fig. 1 The pipeline of our algorithm based on the idea of splitting line

(b) Loop sampling. The loops of each patch are retrieved and the curves of each loop are sorted by traversing common vertexes. In addition, we separate loops that form outer boundaries and holes. Then loop sampling is performed by dividing each curve into straight edges according to L_{max}.

(c) Point mapping. Based on the expression of each B-spline surface, the sampled points of the loops are mapped to the corresponding parametric plane. Therefore, each patch is converted to a polygon on the plane.

(d) Polygon splitting. Guided by a well-trained artificial neural network, two non-adjacent points of the polygon are connected at each step for splitting, and then the polygon is split into two subdomains. Further, mesh points are added to discretize this splitting line. The above process will be repeated until all polygons are decomposed into triangles.

(e) Mapping back. After the mesh for each patch in the parametric plane is generated, we map it back to the 3D surface and further optimize the mesh by applying Laplacian smoothing and surface projection.

With appropriate modification, the proposed algorithm can also be applied to stereo-lithography-type geometric models. Implementation details of each step are described in the following sections.

3 Methodology

3.1 Loop sorting and sampling

In our implementation of the proposed algorithm, we use Open Cascade [24] to get the geometry information. After getting the loops, curves and vertexes, we merge overlapping vertexes and remove degenerate curves and empty loops. Then, the curves of each surface patch are sorted, as presented in Algorithm 1. The algorithm starts from the first curve of the patch, and then looks for the next curve with a common vertex. If the end vertex of the current curve is the same as the first vertex of the loop, it means that the current curve has formed a sub-loop, and then the nearest new curve is found as the start of a new sub-loop.

```
Input: the read-in curves for patch $P_i$;
Output: ordered curve list $L_i = \{c_j\}_{j=1}^{n}$ forming the loop, location of $w$ holes
  $H_i = \{h_j\}_{j=1}^{w}$, direction flag $D_j$ for each curve;

$n \leftarrow$ number of curves of patch $P_i$
set all curves of patch $P_i$ as unvisited
number of holes $w \leftarrow 0$
unset new sub-loop flag
for $j \leftarrow 1$ to $n$ do
    if ($j$ equals 1) then
        current candidate curve $c \leftarrow$ first curve in patch $P_i$
        direction of the $j$-th curve $D_j \leftarrow$ positive
        first vertex of the sub-loop $fv \leftarrow$ begin vertex of $c$
    else
        $lc \leftarrow$ last curve in loop list $L_i$
        last vertex $lv \leftarrow$ ($D_{j-1}$ is positive) ? end or begin vertex of $lc$
        $c \leftarrow$ unvisited curve with vertex nearest to $lv$
        $D_j \leftarrow$ (distance from $lv$ to begin vertex of $c <$ to end vertex of $c$) ?
        positive or negative
    endif
    mark curve $c$ as visited
    append $c$ to loop list $L_i$
    current vertex $cv \leftarrow$ ($D_j$ is positive) ? end or begin vertex of $c$
    if (new sub-loop flag is set) then
        $w \leftarrow w + 1$
        append $j$ to hole location list $H_i$
        $fv \leftarrow$ ($D_j$ is positive) ? begin or end vertex of $c$
        unset new sub-loop flag
    endif
    if ($cv$ is the same vertex as $fv$) then
        set new sub-loop flag
    endif
endfor
```

Algorithm 1 Sorting the loop by traversing directed curves

After sorting the loops for each patch, we need to place points on each curve of the CAD model. First, curves that have specified the desired number of mesh points are uniformly discretized according to the number and the curve length. The remaining curves distribute the points according to their length and the maximum edge length L_{max}. On each mesh point, we define an element size value to guide interior element generation.

For mesh points on geometric curves, its value is calculated from the average length of the edges on both sides. Advanced density control strategies such as density sources or sizing functions [25] can also be applied in our method. To represent the loops discretely, Algorithm 2 is performed by splicing the corresponding discrete curves. Now the patch boundary is represented by several continuous, closed sub-loops of simply connected points instead of curves.

```
Input: curve list $L_i$ and hole location list $H_i$ for patch $P_i$, direction flag $D_j$ for
  each curve, which are all obtained in Algorithm 1, and middle points placed on
  each curve;
Output: point list $Q_i = \{q_j\}_{j=1}^{m}$ sampled on the loop, hole start list
  $S_i = \{s_j\}_{j=1}^{w}$;

$n \leftarrow$ number of curves in $L_i$
for $j \leftarrow 1$ to $n$ do
    if ($D_j$ is positive) then
        if ($j$ equals 1 or $j$ in $H_i$) then
            append begin vertex of curve $j$ to point list $Q_i$
        endif
        if ($j$ in $H_i$) then
            append current length of $Q_i$ to hole start list $S_i$
        endif
        append middle points on curve $j$ to $Q_i$ from begin to end
        if ($j+1$ not in $H_i$) then
            append end vertex of curve $j$ to $Q_i$
        endif
    else
        if ($j$ in $H_i$) then
            append end vertex of curve $j$ to $Q_i$
            append current length of $Q_i$ to $S_i$
        endif
        append middle points on curve $j$ to $Q_i$ from end to begin
        if ($j+1$ not in $H_i$) then
            append begin vertex of curve $j$ to $Q_i$
        endif
    endif
endfor
```

Algorithm 2 Loop discretization

For CAD models containing tiny features or irrelevant details that will have little effect on simulation results, we can remove these features at this step, making mesh generation and numerical simulation more efficient. For discrete polygons of the surfaces, tiny holes with an area smaller than the threshold can be directly deleted, and small patch loops could be removed by merging near points.

3.2 Surface patch mapping

While after the loops of each patch are discretized, we convert the 3D coordinates of these discrete points into 2D coordinates in the parametric space. Moreover, in order to keep the same proportional shape after mapping and improve numerical accuracy during mesh generation, we scale the parametric coordinates to the same size as in the physical space. Specifically, we multiply the u, v coordinates by the scaling coefficients $scale_u, scale_v$, respectively, as shown in Eq. (1).

$$\begin{aligned} S(u,v) =&(x(u,v),y(u,v),z(u,v)) \rightarrow (u,v), \\ U =&scale_u \cdot u, \\ V =&scale_v \cdot v, \end{aligned} \tag{1}$$

where $scale_u$ and $scale_v$ are equal to the lengths of the patch in two directions in the physical space divided by their lengths in the parametric space, respectively. Because the lengths of the patch in the physical space are not directly available, they can be calculated as shown in Eq. (2).

$$\begin{aligned} L_u =&\sum_{i=1}^{n} \|S(u_i,v_{mid}) - S(u_{i-1},v_{mid})\|, u_i = u_0 + \frac{i}{n} l_u, v_{mid} = v_0 + \frac{1}{2} l_v, \\ L_v =&\sum_{i=1}^{n} \|S(u_{mid},v_i) - S(u_{mid},v_{i-1})\|, v_i = v_0 + \frac{i}{n} l_v, u_{mid} = u_0 + \frac{1}{2} l_u, \end{aligned} \tag{2}$$

where n is the number of sampling points, which can be set to the number of mesh points on that loop, or a constant. $\|\cdot\|$ means the Euclidean distance of two points. l_u, l_v are the lengths of the patch in the parametric space, respectively. It should be noted that the element size value on each point is computed after patch mapping.

For very distorted and complex surfaces, the mesh generated by the mapping method may result in poor quality after being remapped back to the original space. Further work could be studied to find optimal splitting lines in the parametric space, but split the surface patch in the physical space for meshing.

3.3 Intelligent optimal splitting line selection

In the parametric plane of each patch, the proposed algorithm splits the domain into two parts by connecting two mesh points each time, then puts points on the splitting line, and further splits these subdomains until they are all triangles. For patches with holes, it needs to connect each sub-loop first to form a single connected domain, as shown in Fig. 2. For the selection of splitting line at each step, our strategy is to find all feasible splitting lines first, and then use a machine learning method to select the optimal one.

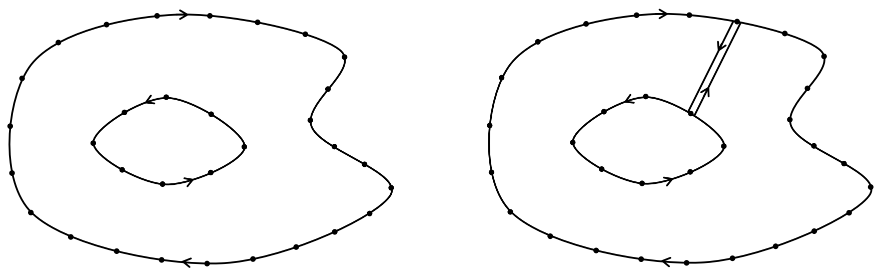
(a) Loop with a hole (b) One continuous loop formed by connecting a line between two sub-loops

Fig. 2 The schematic diagram of $n-1$ splitting lines connecting n sub-loops. Each of these lines can be seen as two lines in opposite directions, and the direction of the holes should be opposite to the direction of the outer sub-loop

To connect sub-loops as one continuous loop, the intuitive method is to connect points on different sub-loops, and determine whether they intersect with other parts. If there is no intersection, then it is a feasible splitting line. When all sub-loops are concatenated into a ring, it forms a simple polygonal area. In this case, we have more efficient ways to find feasible splitting lines as presented in [23, 26] based on the concept of visibleness.

Point q is visible from point p whenever the line connecting p and q does not pass outside the boundary. For convex loops, all non-adjacent points are visible to each other. For a given point p on the concave loop, the visible points are determined by two scans around the loop in two directions. Let the two segments connected to point p be reference line 1 and 2, respectively. The angles between reference line 1 and a line formed by connecting every other point on the loop to point p are computed. This angle θ keeps changing as the scan progresses. The first scan only records points with increasing angles, and the second scan in the reverse order only records points with decreasing angles. The points recorded in both scans are the visible points of point p. That is, the point whose angle changes monotonically in both directions is the visible point. Although this method may miss some extreme points than necessary, it is very efficient because of its linear complexity. A more detailed discussion can be found in [23].

After finding all feasible splitting lines for the current subdomain, the next task is to choose the best one among them, and this is the core algorithm developed in ISpliter. Previous studies used a weighting function to represent the quality of the splitting line, and considered the influence of angle, length, area, and point distribution according to different weights. However, the weight coefficients are obtained empirically, and it is difficult to find a configuration that works well in all situations. Therefore, we thought of training a machine learning model to score the splitting lines.

Since the quantity and characteristics of the raw data of the loops are protean, it is difficult to represent it as an end-to-end model. We extract a fixed number of features from the splitting line on the loop as the input of the model, and an artificial neural network (or called multi-layer perception [27]) with two hidden layers is proposed as shown in

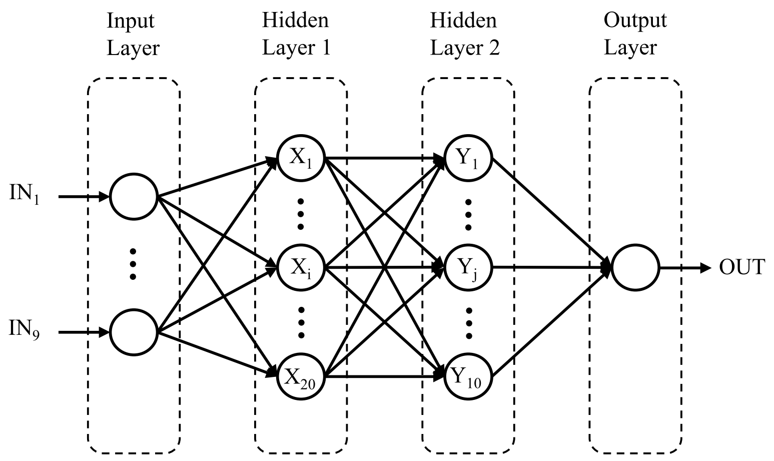

Fig. 3 The structure of the proposed artificial neural network in ISpliter

Fig. 3. The input layer has 9 neurons, the two hidden layers have 20 and 10 neurons, respectively, depending on the representation capability and computational efficiency, and all adjacent layers are fully connected. The output layer gives the score for that splitting line, and thus the splitting line with the highest score is the final choice.

The nine input features of the splitting line are presented as follows. The first four features relate to the angles α_i $(i = 1, .., 4)$ between the splitting line and the loop, as shown in Fig. 4. In order to make the generated elements close to equilateral triangles, the interior angle after split should be a multiple of 60° as close as possible. Hence, these four features are calculated as Eq. (3) shows.

$$f_i = \min \frac{\left|\alpha_i - \frac{\pi}{3} \cdot j\right|}{\pi}, i = 1, .., 4, j = 1, .., 5. \tag{3}$$

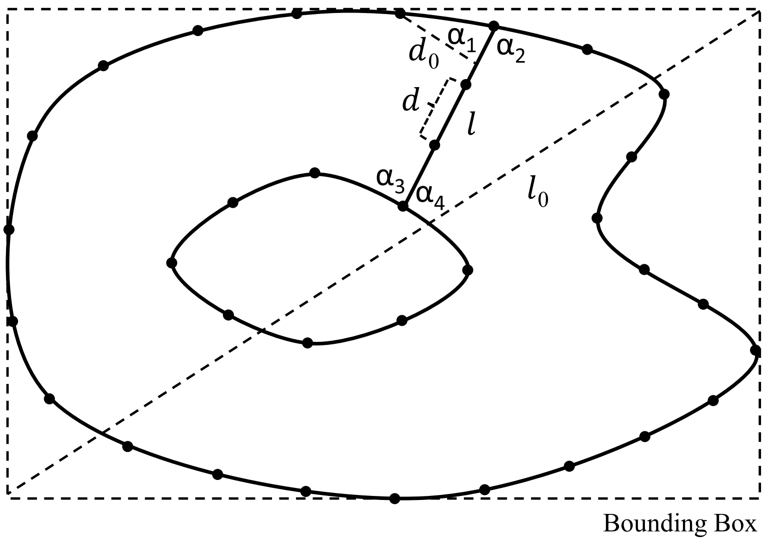

Fig. 4 The schematic diagram of features of the splitting line

The fifth feature describes the relative length of the splitting line, $f_5 = l/l_0$, where l is the length of the splitting line, and l_0 is the diagonal length of the bounding box of the loop. The sixth feature represents the distance from the splitting line to the boundary, which can be expressed as $f_6 = d/d_0$, where d denotes the length of the shortest line segment after the splitting line is discretized. This can be measured as the shortest distance between one of the two endpoints of the splitting line and its neighbors on the loop. d_0 is the shortest distance from the points on the loop to the splitting line, as shown in Fig. 4. The seventh feature represents the error between the actual number of elements m and the ideal number of elements m_0 on the splitting line, $f_7 = |m_0 - m|/m_0$. The specific calculation methods of these two variables are presented in Section 3.4. The eighth feature gives the ratio of the number of elements on the splitting line to the number of elements on one side of the loop, $f_8 = m/n_s$. n_s is the minimum number of points on one of the two loop sides. The last feature computes the symmetry of the two loop sides after splitting, $f_9 = |2 \cdot n_s - n|/n$, where n denotes the number of elements on the loop. In general, for all features, a smaller value indicates a better feature.

Our neural network is implemented in the Pytorch framework [28], and the model is exported after being trained. Then we read in the model by Libtorch (Pytorch C++ version) that has been integrated in ISpliter and do inference scoring for candidate splitting lines. It should be noted that our model only needs to be trained once, and will be reused for every selection. For more details about the model configuration and training data, see Section 4.1.

3.4 Point placement on splitting line

After connecting the selected splitting line, the next step is to place points on the line to discretize it. Unlike the linear spacing used in literature [23, 26], we apply equal spacing and geometric spacing methods according to the difference in element size values at two endpoints of the splitting line. When the gradient of the element size value along a splitting line is high, the linear spacing method tends to make large elements to monopolize the spacing, and this will make the transition region uneven.

As shown in Fig. 5, the element size value at endpoints of a splitting line is determined by the average length of the line segments on both sides of the point on the boundary. Set

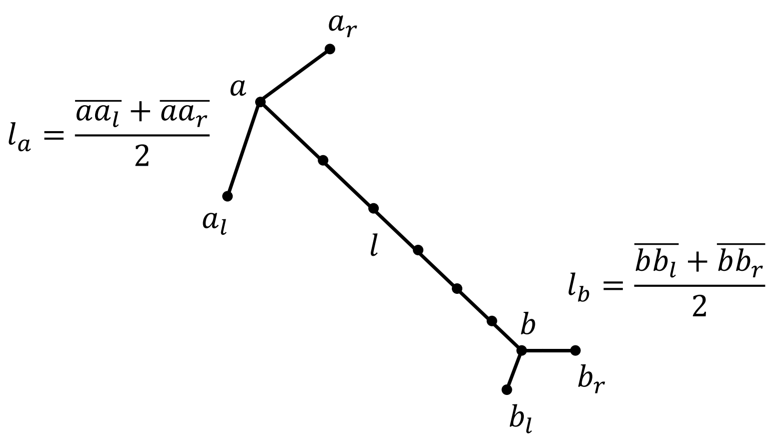
a_r
a
$l_a = \frac{\overline{aa_l} + \overline{aa_r}}{2}$
a_l
l
$l_b = \frac{\overline{bb_l} + \overline{bb_r}}{2}$
b
b_r
b_l

Fig. 5 The schematic diagram of point placement on the splitting line

it as l_a, l_b respectively. When l_a, l_b are about the same, say $1/(1+h) < l_b/l_a < (1+h)$, we adopt the equal spacing method, otherwise the geometric spacing method is used. In this work we set $h = 0.01$. For equal spacing method, the ideal number of elements (newly created points) on the splitting line is calculated as shown in Eq. (4). The actual number of elements is the integer closest to the ideal value, i.e. $m = \lfloor m_0 + 1/2 \rfloor$.

$$m_0 = \frac{2 \cdot l}{l_a + l_b}. \tag{4}$$

Therefore, the relative coefficients of the created equidistant points are shown in Eq. (5), and the coordinates of these points can be calculated from Eq. (6), where a and b denote the coordinates of the two endpoints, respectively.

$$l_i = \frac{i}{m+1}, i = 1, .., m. \tag{5}$$

$$p_i = (1 - l_i) \cdot a + l_i \cdot b, i = 1, .., m, 0 < l_i < 1. \tag{6}$$

For the geometric spacing method, the ideal number of elements m_0 on the splitting line can be calculated as shown in Eq. (7), and the relative coefficients of the created points are shown in Eq. (8).

$$m_0 = \frac{\ln\left(l_b/l_a\right)}{\ln\left(\frac{l_b - l_a}{l + l_a} + 1\right)} - 1. \tag{7}$$

$$l_i = \frac{\left(l_b/l_a\right)^{\frac{i+1}{m+2}} - \left(l_b/l_a\right)^{\frac{1}{m+2}}}{\left(l_b/l_a\right) - \left(l_b/l_a\right)^{\frac{1}{m+2}}}, i = 1, .., m. \tag{8}$$

After placing points on the splitting line, it splits the loop into two complete sub-loops for simple loops. For loops with a hole, it connects the loop into one simple loop. Further, the algorithm keeps splitting these sub-loops until they are all triangles. We handle this process in a queue data structure.

3.5 Post processing

Applying the approach presented above, each surface patch generates a triangular mesh in the parameter plane, and then it needs to be inversely mapped back to the physical space. Specifically, the coordinates of each mesh point need to be divided by the scaling coefficients and then converted to 3D coordinates, as shown in Eq. (9).

$$\begin{aligned} u =& U/scale_u, \\ v =& V/scale_v, \\ (u, v) \rightarrow & (x(u, v), y(u, v), z(u, v)). \end{aligned} \tag{9}$$

Furthermore, smoothing technique is applied to fine-tune the generated mesh and equalize the element size. The smoothing method used here is the Laplacian method [29]. The core idea of Laplacian method is to move any given internal point to the centroid of its neighbors, as presented in Eq. (10).

$$x_i = \frac{1}{N_i}\sum_{j=1}^{N_i} x_j, \quad y_i = \frac{1}{N_i}\sum_{j=1}^{N_i} y_j, \quad z_i = \frac{1}{N_i}\sum_{j=1}^{N_i} z_j, \quad j \in neighbor(i). \tag{10}$$

After smoothing each internal point, it needs to be reprojected onto the surface. Smoothing can be performed multiple times until the number of iterations reaches the upper limit, or the maximum distance between the points before and after smoothing is less than the threshold. Finally, the Cuthill-McKee renumbering algorithm [30] is used to reduce the bandwidth of the generated mesh.

In addition, the algorithm proposed in this paper can be easily accelerated by parallelization, and the related discussion of parallelization can refer to literature [26]. In further work, we will apply multi-level parallelism between patches, subdomains, and splitting lines.

4 Results

In this section, we first present the generation of training data for the proposed neural network model, and then the detailed configuration and training method of the model is given. Subsequently, the results of ISpliter and two well-known mesh generators are compared through several typical cases. One of the two generators is the open source software Gmsh [31] maintained by Professor C. Geuzaine and J. Remacle, and the other is the unstructured version of NNW-GridStar [32] launched by China Aerodynamics Research and Development Center (CARDC). Their version numbers are 4.10.5 and V3.0.0 respectively.

4.1 Model training

To generate the sample data for training, the key is how to score the splitting line. First, we define the quality of a triangular element by skewness, which is equal to the actual area divided by the optimal area, as shown in Eq. (11), and higher value means better quality.

$$skewness = \frac{S_a}{S_o}, \quad 0 \le skewness \le 1, \tag{11}$$

where the optimal area S_o is the area of an equilateral triangle having the same circumcircle. Then, the quality of a subdomain mesh is represented by the minimum skewness of the elements. The quality of a sub-loop is represented by the best meshing results of that subdomain among all possible meshes using the splitting line method. Therefore, the score of a splitting line can be represented by the lower quality of the sub-loops on either side of it. This recursive process is shown in Algorithm 3.

```
Input: loop $L$;
Output: features and scores of all possible splitting lines for that loop and
  further sub-loops;

function splitone($L$)
    if ($L$ is a triangle) then
        return $skewness(L)$
    else
        best ← 0
        for each possible splitting line of $L$ do
            compute 9 features of this splitting line
            place points on this splitting line
            split $L$ into $L_1$, $L_2$
            current ← min(splitone($L_1$), splitone($L_2$))
            best ← max(best, current)
        endfor
        return best
    endif
endfunction
```

Algorithm 3 Compute features and scores of splitting lines for training

According to Algorithm 3, a simple loop will generate many different types of splitting line samples. Hence we use a triangular loop with 9 points and a square loop with 8 points to generate the training data, and a circular loop with 8 points to generate the test data. There are 230,037 samples in total. Figure 6 shows the three sample loops, and the first best splitting line selected by Algorithm 3 for each loop is the red dotted line. It should be noted that the number of possible splitting lines in Algorithm 3 increases exponentially. Therefore, it is impossible to directly use it to find the optimal line in the real mesh generation process.

In the hidden layers, we use the conventional rectified linear unit (ReLU) as the activation function to enhance the nonlinearity of the neural network model, and apply dropout to prevent the neural network from overfitting [33]. The mean-squared error (MSE)

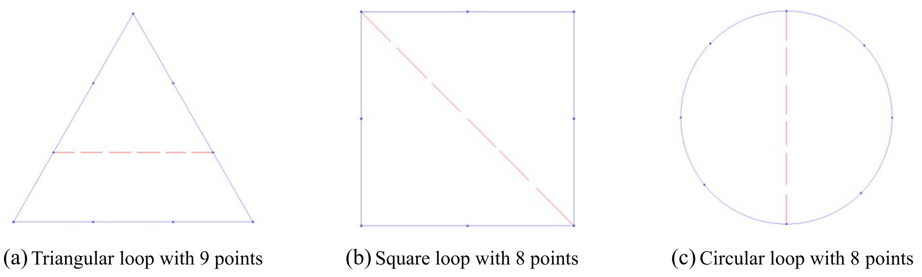
(a) Triangular loop with 9 points (b) Square loop with 8 points (c) Circular loop with 8 points

Fig. 6 Three loops for training data and test data generation. The red line is the first best splitting line selected for that loop

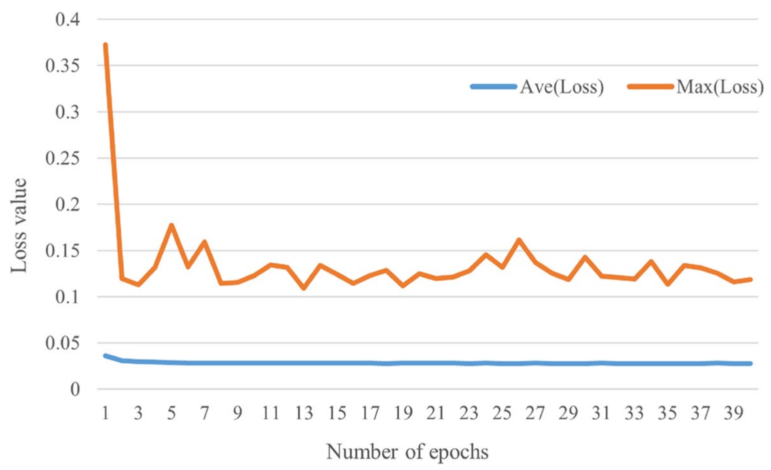

Fig. 7 The average and maximum loss in the training process of the proposed artificial neural network

Table 1 The number of points and elements of the meshes generated by the baselines and ISpliter. L is the characteristic length of the geometry. L_{max} is the maximum edge length of the mesh

Cases	Triangle $L=10$ $L_{max}=1$	Square $L=10$ $L_{max}=1$	Circle $L=2R=10$ $L_{max}=1$	Hole $L=1$ $L_{max}=0.02$
Gmsh	66/100	145/248	122/210	3054/5750
GridStar	72/112	174/278	105/177	2785/5213
ISpliter	66/100	135/228	108/183	2674/4997
Cases	Capsule $L=25$ $L_{max}=1$	X43 $L=3.7$ $L_{max}=0.05$	Missile $L=25397$ $L_{max}=200$	M6 $L=1376$ $L_{max}=33$
Gmsh	2052/4100	5659/10712	18289/35806	47489/92396
GridStar	4891/9778	100941/201792	14949/29894	7289/14574
ISpliter	2006/4008	4895/9694	20776/41548	44258/88512

loss is used to train the regression model. We train the model with the adaptive moment estimation (Adam) optimizer for 40 epochs. The batch size is set as 10, and the learning rate is set as 0.001. The average and maximum loss after each epoch in the training process is shown in Fig. 7. It can be seen that the training converges quickly. Furthermore, the average and maximum loss of the test dataset for the trained model is 0.026 and 0.163, respectively.

4.2 Comparisons

To demonstrate the performance of ISpliter, based on 8 typical geometries, we compare the surface meshes generated by ISpliter, Gmsh and GridStar from multiple perspectives. For each case, we specify the same maximum edge length L_{max} in the three generators. GridStar has to specify another parameter, curvature adaptive angle, and we use its default value 18° for all cases. The Delaunay-Frontal algorithm that is recommended to generate high quality elements is selected in Gmsh. Table 1 presents the number of points and elements of the generated meshes. We can see that, except for the X43 and M6 cases, the meshes generated by the three software are comparable in size. Since the geometric curvature of the X43 case varies greatly, GridStar automatically performs

Table 2 The quality of the meshes generated by the baselines and ISpliter. They are the average skewness, the minimum skewness, the average minimum angle, and the minimum minimum angle of all elements, respectively. The best values are bolded

Cases	Triangle	Square	Circle	Hole	Capsule	X43	Missile	M6
Gmsh	0.98	**0.97**	**0.96**	0.97	0.91	0.90	0.97	0.97
	0.59	**0.83**	0.77	0.68	0.17	**0.01**	0.00	0.00
	57.75	**55.23**	**53.76**	55.41	51.13	50.22	55.60	55.84
	37.74	**43.72**	**43.42**	36.21	21.54	0.54	0.00	0.00
GridStar	0.97	0.91	0.95	0.97	0.89	0.87	**0.98**	0.87
	0.64	0.56	**0.81**	0.76	0.09	**0.01**	**0.07**	**0.08**
	55.82	49.22	51.61	**55.63**	45.54	44.83	**57.17**	45.18
	34.00	27.82	40.93	**40.45**	13.71	0.09	**4.54**	**3.00**
ISpliter	**1.00**	**0.97**	**0.96**	**0.98**	**0.96**	**0.93**	0.88	**0.99**
	1.00	0.77	0.62	**0.77**	**0.55**	**0.01**	0.01	0.03
	60.00	55.11	52.20	53.78	**54.02**	**52.20**	44.55	**57.39**
	60.00	41.45	31.49	37.13	**26.35**	**0.97**	1.42	2.82

adaptive refinement on it, making its mesh size larger. For the M6 case, we set to generate 300 points on the curves in the spanwise direction, but GridStar does not support specifying the number of points on the specific curve, so the number of generated elements is less.

In this paper, we use the skewness ([0,1], as shown in Eq. (11)) and minimum interior angle ([0°,60°]) to quantify the quality of a triangle element. While the closer the skewness value is to 1 or the closer the minimum angle is to 60 degree, the closer the element is to an equilateral triangle, which means that the better its isotropy is. Therefore, the quality and anisotropy of a triangular mesh can be represented by the average, minimum and distribution of these two values for its all elements. Table 2 summarizes the overall qualities of the meshes generated by the baselines and ISpliter for all cases. It can be seen that Gmsh generally generates the best meshes for simple plane geometries, but it may easily generate degenerate element for complex surfaces. In contrast, the quality of the worst element generated by GridStar is the highest for complex models. This may relate to its curvature adaptation ability. For ISpliter, it generates meshes with the highest average quality, and the quality of its worst elements is not too far from the best results. Specific resulting meshes and quality distributions are presented in the following figures.

Figure 8 shows the generated meshes of a triangle geometry, and gives the skewness distributions and histograms of the minimum angle. Three software generate high quality results, but it is particularly noteworthy that ISpliter generates the ideal mesh with all equilateral triangles. This preliminarily demonstrates the effectiveness of the splitting line method scored by neural networks. In addition, because the number of points distributed on the loop is different, the triangle and square geometries used to generate splitting line samples in the training stage are not exactly the same as those in the experiments.

Similarly, Fig. 9 presents the meshes generated for the square geometry. It can be seen that the elements near the boundaries generated by Gmsh and ISpliter are better

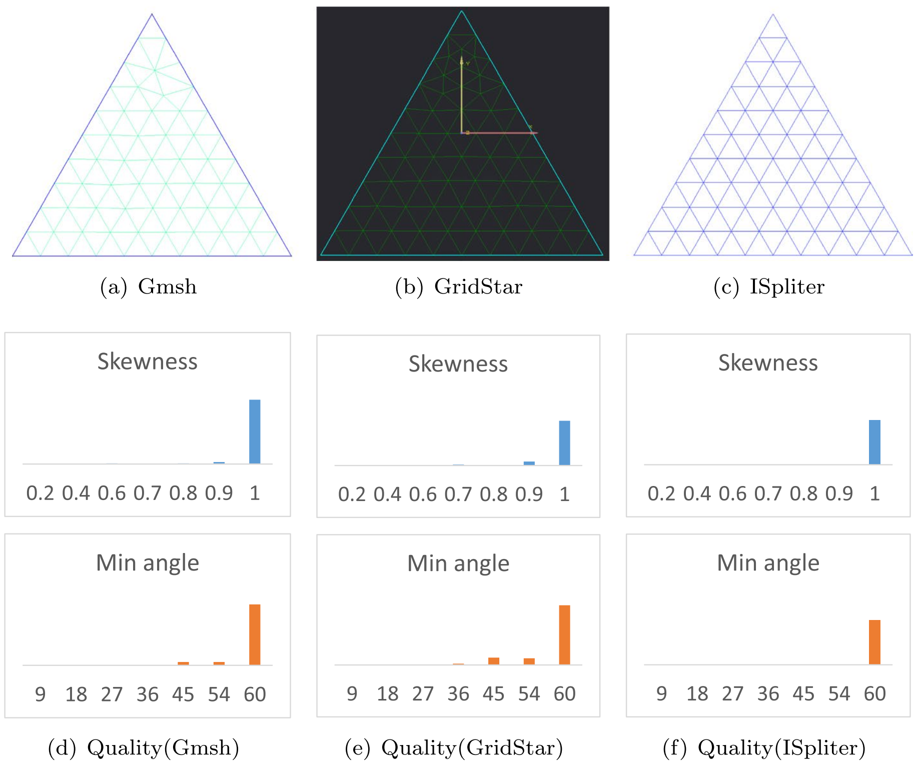

Fig. 8 The meshes of the triangle generated by the baselines and ISpliter

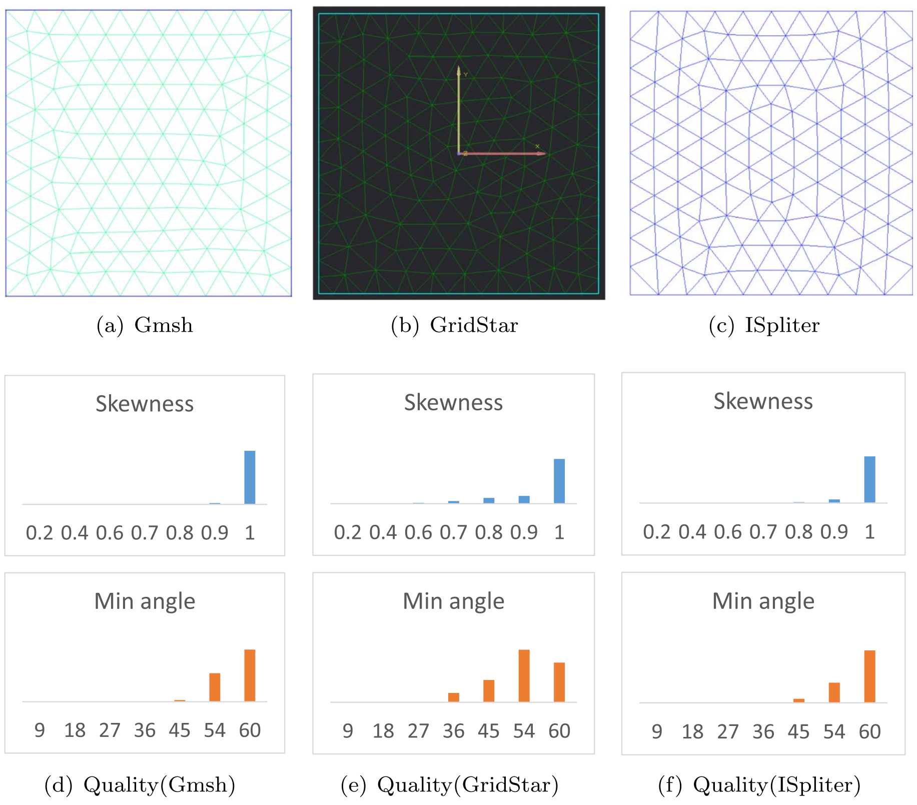

Fig. 9 The meshes of the square generated by the baselines and ISpliter

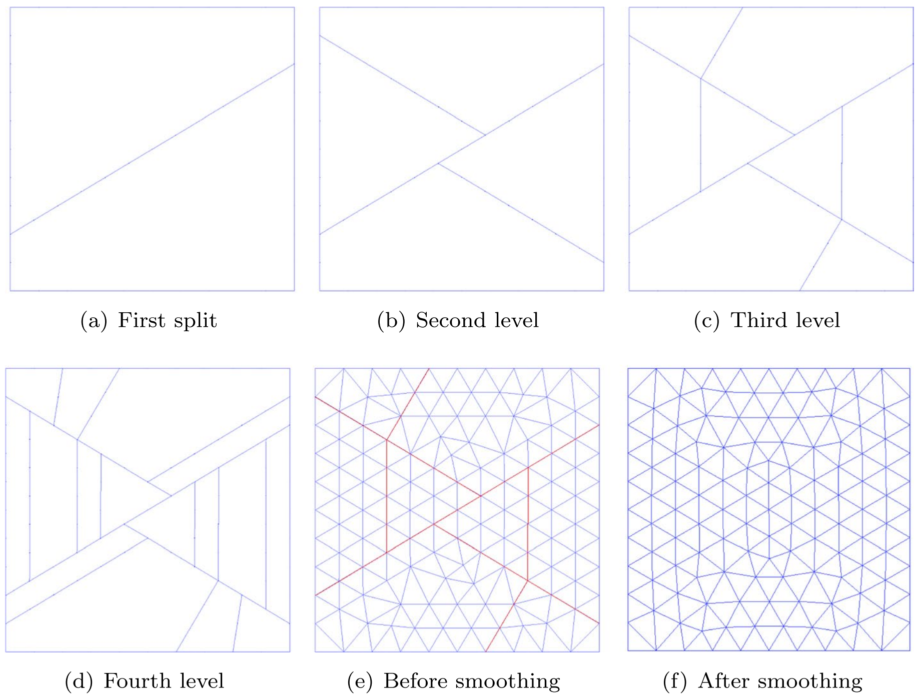

Fig. 10 The mesh of the square generated by ISpliter in different stages

Table 3 The statistics of the mesh quality distribution of the square generated by ISpliter before and after smoothing. The first part is the number of elements with corresponding quality. The last two columns are the average and minimum values

Skewness	0-0.2	0.2-0.4	0.4-0.6	0.6-0.7	0.7-0.8	0.8-0.9	0.9-1	Ave	Min
Before	0	0	4	2	14	18	190	0.95	0.51
After	0	0	0	0	4	16	208	0.97	0.77
Min angle	0-9	9-18	18-27	27-36	36-45	45-54	54-60	Ave	Min
Before	0	0	0	6	20	44	158	54.66	33.82
After	0	0	0	0	12	60	156	55.11	41.45

than GridStar, although GridStar places more points on the boundaries. All the quality indicators of Gmsh are the highest, and the results of ISpliter are also very close to it. Their smallest angles are all greater than 40°.

To investigate the behavior of each interior step of the proposed approach, the mesh of the square geometry generated by ISpliter in different stages is presented in Fig. 10. It first splits the geometry into two domains with equal area, and four interior angles of the splitting are all close to 60° or 120°. The second level splitting even forms two nearly equilateral triangles, and the selection of splitting lines in further levels is the same for domains with the same shape after rotation. This shows the effectiveness of feature extraction of the proposed approach. The generated initial mesh before smoothing is shown in Fig. 10(e). It can be seen that the difference before and after optimization is not significant. The statistics of the mesh quality are summarized in Table 3.

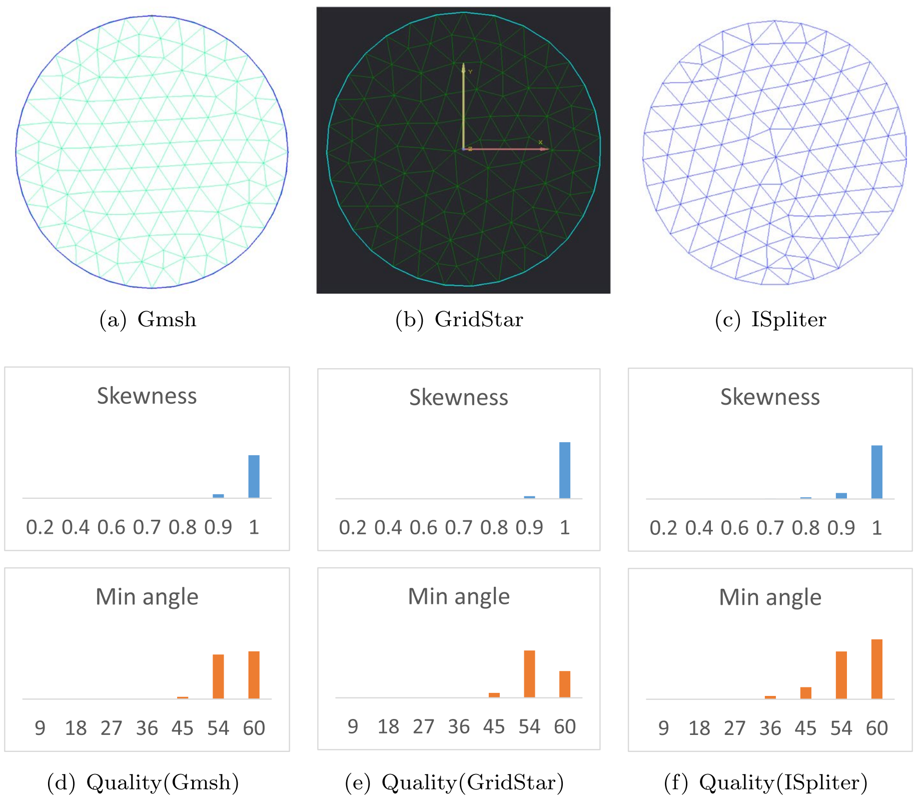

Fig. 11 The meshes of the circle generated by the baselines and ISpliter

The meshes and their quality distributions of the circle geometry generated by the three software are presented in Fig. 11. Again, the best mesh is still generated by Gmsh, especially its middle region is almost all uniform equilateral triangles. Although the transition between the middle and bottom regions of the mesh generated by ISpliter is not very smooth, its average skewness and minimum angle are pretty high. The first three simple cases examine the mesh generation capabilities of the three software for basic planar shapes, which is the basis for the surface mesh generation of real CAD models. The results show that ISpliter is comparable to the baselines in planar mesh generation.

Figure 12 shows the mesh results of a curved surface with complex topology structures. From the perspective of skewness and minimum angle, the mesh generated by Gmsh is the worst. It can also be seen from Table 2 that the quality of all meshes of the subsequent geometries generated by Gmsh is inferior to GridStar and ISpliter. Combined with the results of previous plane cases, it shows that Gmsh has slightly poor processing ability for 3D surfaces. For this surface with 5 holes, the mesh generated by ISpliter has the best skewness values, and GridStar has the best minimum angles. Moreover, it can be inferred from the elements around the inner holes that GridStar adopts the AFT method.

The latter four cases are the CAD models with multiple trimmed surfaces. The capsule model has 60 curves and 32 surfaces, and the generated meshes are shown in Fig. 13. It

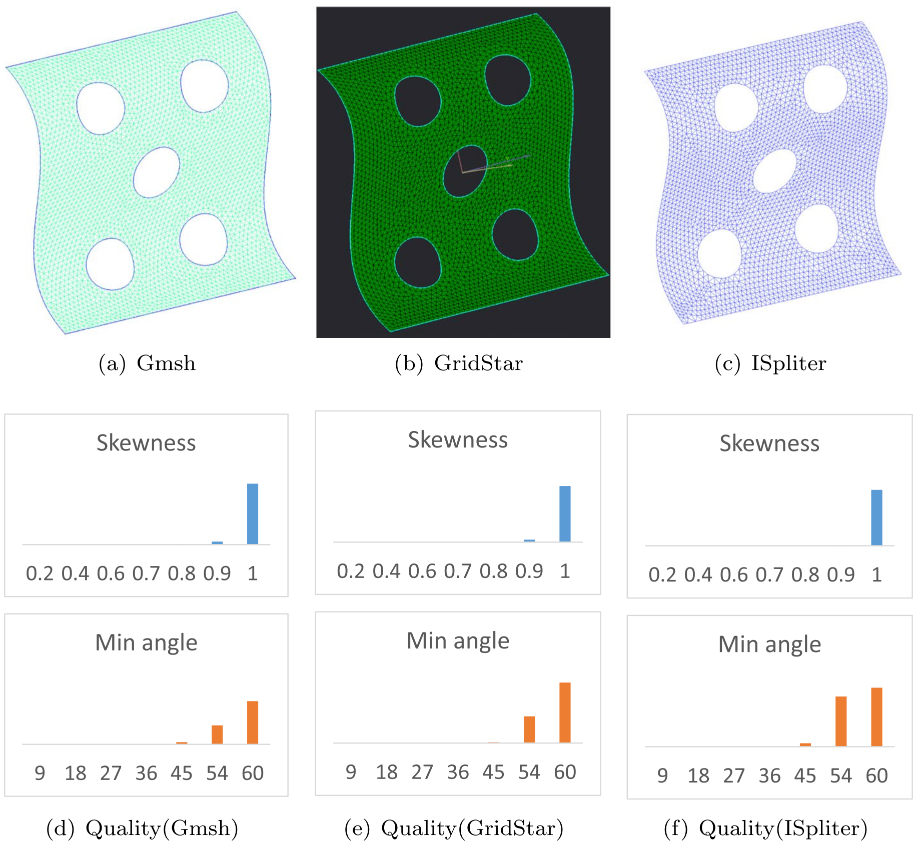

Fig. 12 The meshes of the curved surface with holes generated by the baselines and ISpliter

can be found that, as the mesh density increases, ISpliter generates better meshes for the circular plane compared with Fig. 11. This is because it has larger space to choose more suitable splitting lines. From the distributions of skewness and minimum angle we can easily see that the mesh generated by ISpliter is of the best quality. Moreover, because the size and shape of patches in different regions of the geometry may vary greatly, if the size of the patch is too small in certain dimensions compared with L_{max}, the nonuniform distribution of elements will inevitably occur. For example, for the long and thin band at the edge of the capsule in Fig. 13, the three software generate elements smaller than L_{max} in that area. GridStar even generates denser elements based on the curvature factor. In such cases, one can specify smaller mesh size parameters if a uniform mesh is wanted.

Figure 14 shows the meshes of the X43 aircraft generated by the baselines and ISpliter. The X43 aircraft model has 211 curves and 90 surfaces, and there are very thin edges at the wings. Since the relative large value of L_{max} is set, the minimum angle of the generated elements near the wings is small. The minimum skewness of these three meshes all equals 0.01 in Table 2, but under the same conditions, ISpliter still generates a mesh with better quality.

Figure 15 gives the results of a missile model. This missile model has 125 curves and 32 surfaces. From the distributions of the skewness and minimum angle, it can be seen

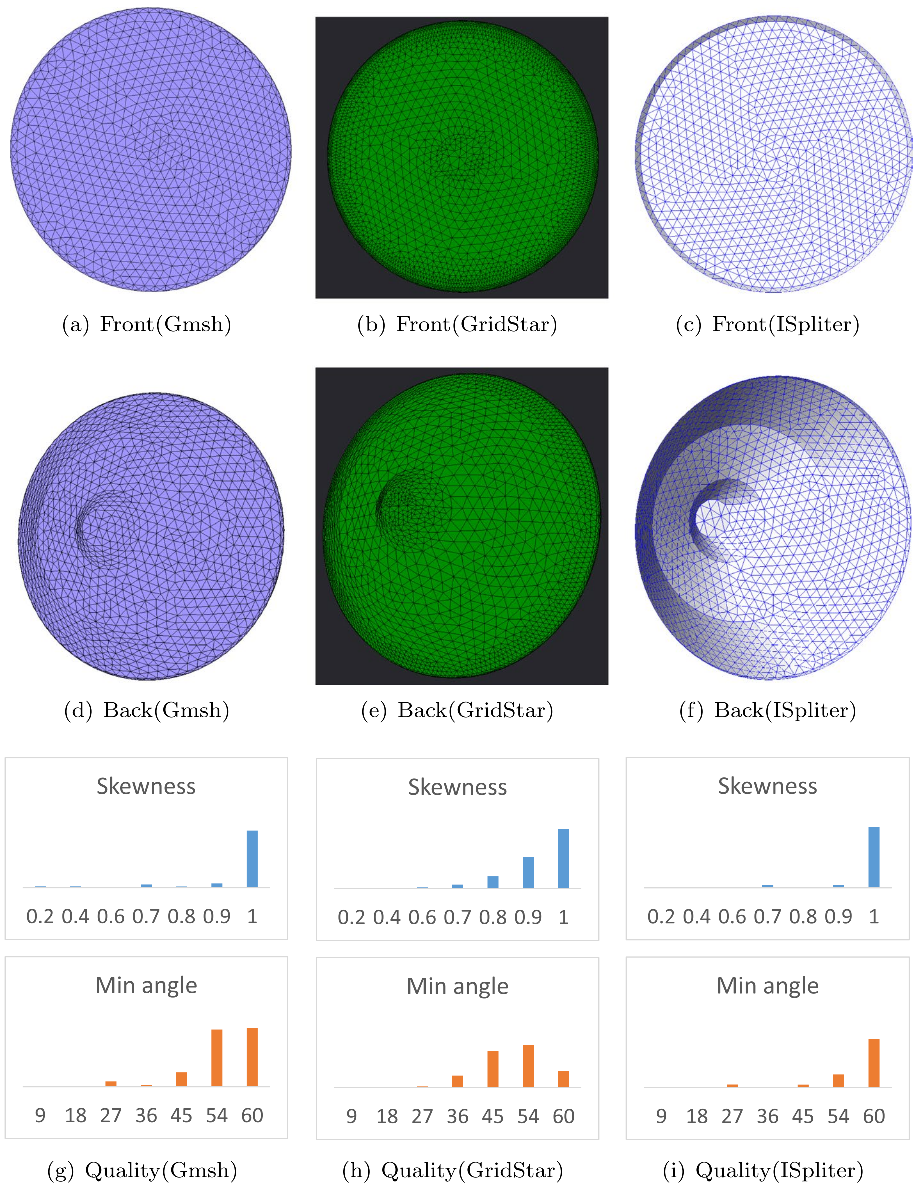

Fig. 13 The meshes of the capsule generated by the baselines and ISpliter

that Gmsh and GridStar generate elements with higher quality. However, it can be found from Table 2 that Gmsh generates several degenerate triangles, which have interior angles with zero degree. The mesh generated by ISpliter has a minimum angle of 1.4° and its average quality is not bad, and the 6 worst elements are shown in Fig. 15. In this case, the mesh generated by GridStar has the best quality.

The M6 wing model has 12 curves and 6 surfaces, and its size span from thickness to length is close to 3 orders of magnitude. Therefore, we set 4 curves in the spanwise direction to place 300 points on each of them to generate fine meshes (in ISpliter and

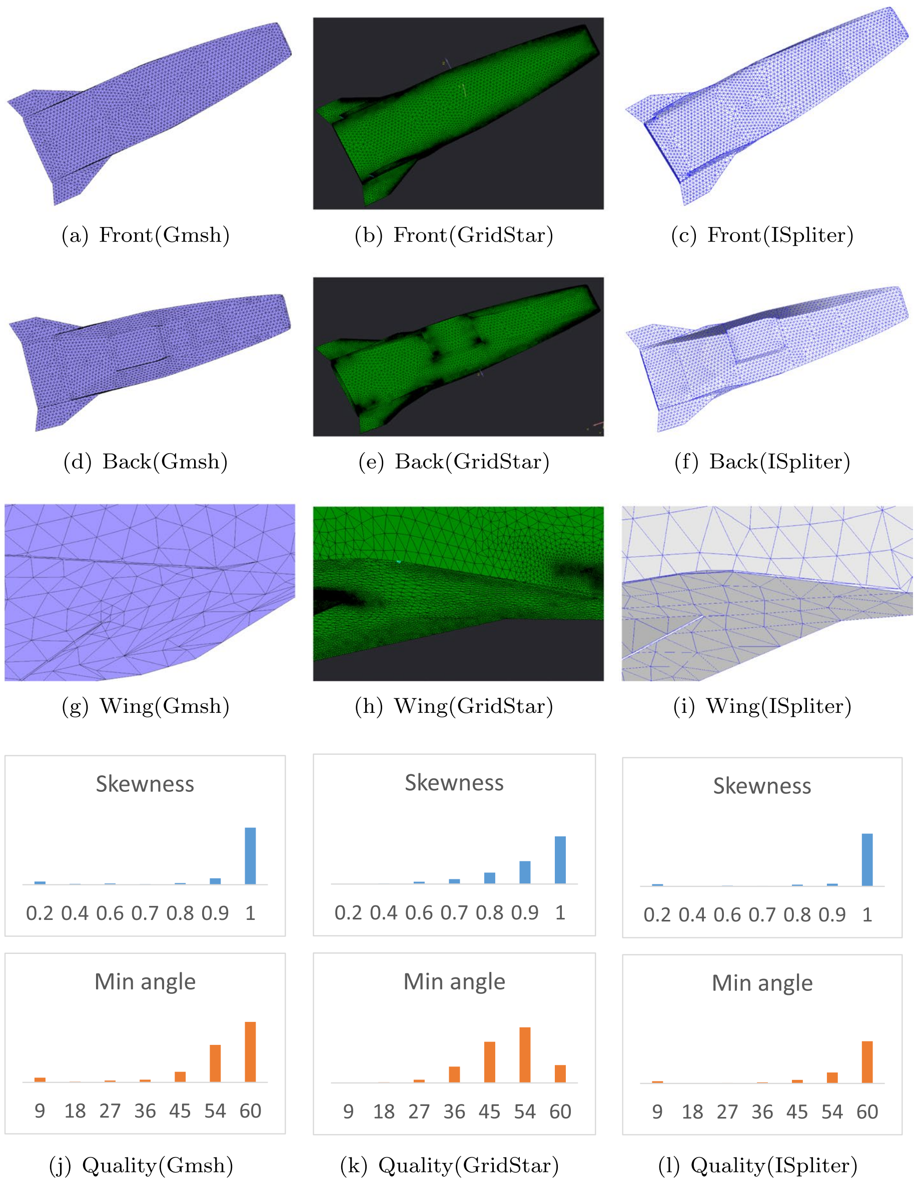

Fig. 14 The meshes of the X43 aircraft generated by the baselines and ISpliter

Gmsh). This allows for a comparative analysis with the results of the X43 aircraft case. The generated meshes of the three software are shown in Fig. 16. We can see that GridStar does not generate denser elements on the trailing edge of the wing by its curvature adaptive refinement. ISpliter and Gmsh generate similar meshes with smooth density transitions.

For meshes generated by ISpliter and Gmsh, the banding phenomenon is formed at the middle part of the wing meshes. In our method, the number of points on the splitting line is determined by the size value of its two endpoints, as shown in Eqs. (4) and

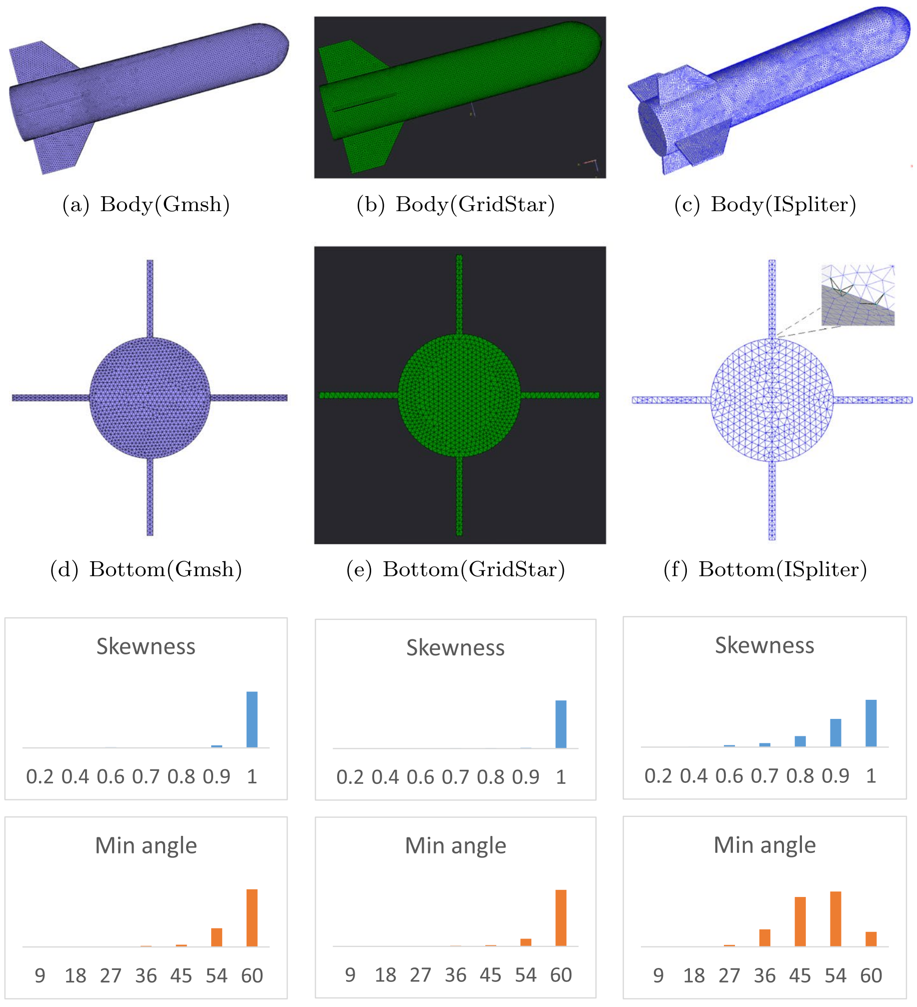

Fig. 15 The meshes of the missile generated by the baselines and ISpliter

(7), and it tends to choose shorter splitting lines due to the fifth feature of the neural network. Therefore, the mesh near the splitting lines connecting the leading and trailing edges of the wing is denser. However, it can be seen from Table 2 that Gmsh once again generates several degenerate elements. This indicates that Gmsh is less robust on complex surfaces. Overall, ISpliter generates the mesh that meets expectations and is of the best quality.

5 Conclusions

In this paper, a novel surface mesh generator ISpliter is developed based on the splitting line method using neural networks. Nine features of a splitting line are extracted to represent it in the neural network, and the best splitting line selected by the neural network is then applied to split the surface into triangles. In addition, a recursive approach is

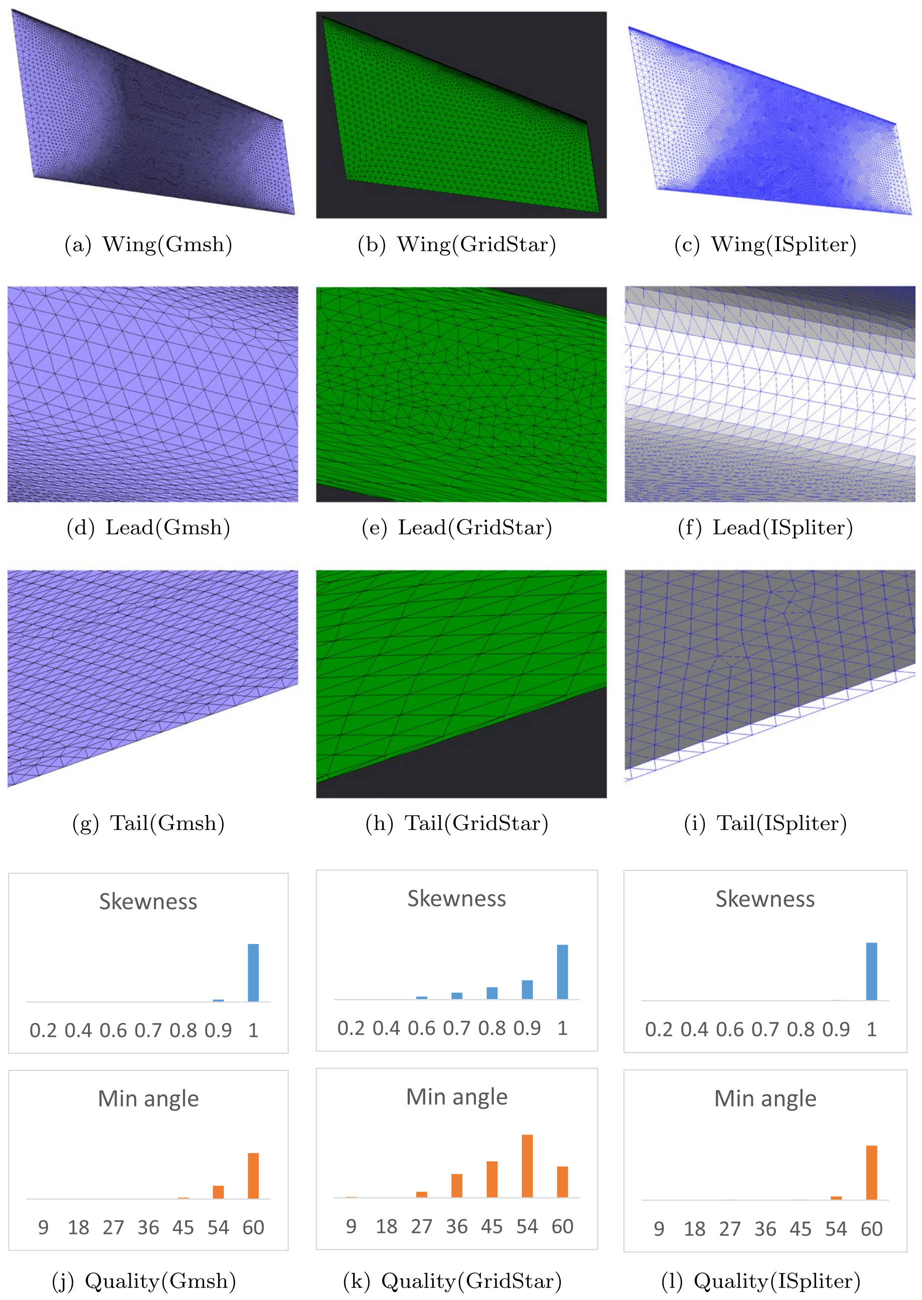

Fig. 16 The meshes of the M6 wing generated by the baselines and ISpliter

proposed that can easily generate abundant data to train the model. The detailed experimental results show that ISpliter can generate high-quality isotropic triangular surface meshes for CAD models with boundary representation. In future work, we will make minor modifications to extend ISpliter to generate quadrilateral meshes, and apply multi-level parallelism to improve its performance. In the meanwhile, more test cases are expected to further verify the codes.

Acknowledgements
Not applicable.

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

Funding
This work is supported by the National Key Research and Development Program of China (No. 2021YFB0300101), the National Natural Science Foundation of China (Nos. 12102467 and 12102468), the Foundation of National University of Defense Technology (No. ZK21-02), and the Foundation of State Key Laboratory of High Performance Computing of China (Nos. 202101-01 and 202101-19).

Availability of data and materials
The data sets 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.

Received: 2 January 2023 Accepted: 17 April 2023
Published online: 01 June 2023

References
1. Baker TJ (2005) Mesh generation: Art or science? Prog Aerosp Sci 41(1):29–63
2. Shimada K (2011) Current issues and trends in meshing and geometric processing for computational engineering analyses. J Comput Inf Sci Eng 11(2):021008
3. Schroeder WJ, Shephard MS (1988) Geometry-based fully automatic mesh generation and the Delaunay triangulation. Int J Numer Methods Eng 26(11):2503–2515
4. George PL, Seveno E (1994) The advancing-front mesh generation method revisited. Int J Numer Methods Eng 37(21):3605–3619
5. Rivara MC, Diaz J (2020) Terminal triangles centroid algorithms for quality Delaunay triangulation. Comput Aided Des 125:102870
6. Liu Y, Lo SH, Guan ZQ et al (2014) Boundary recovery for 3D Delaunay triangulation. Finite Elem Anal Des 84:32–43
7. Adamoudis LD, Koini G, Nikolos IK (2012) Heuristic repairing operators for 3D tetrahedral mesh generation using the advancing-front technique. Adv Eng Softw 54:49–62
8. Nakahashi K, Sharov D (1995) Direct surface triangulation using the advancing front method. In: Proceedings of the 12th Computational Fluid Dynamics Conference. AIAA, Reston, pp 442–451
9. Borouchaki H, Laug P, George PL (2000) Parametric surface meshing using a combined advancing-front generalized Delaunay approach. Int J Numer Methods Eng 49(1–2):233–259
10. Marchandise E, Remacle JF, Geuzaine C (2012) Quality surface meshing using discrete parametrizations. In: Quadros WR (ed) Proceedings of the 20th international meshing roundtable. Springer, Berlin, Heidelberg, pp 21–39
11. Guo J, Ding F, Jia X et al (2019) Automatic and high-quality surface mesh generation for CAD models. Comput Aided Des 109:49–59
12. Zhang Z, Wang Y, Jimack PK et al (2020) MeshingNet: a new mesh generation method based on deep learning. In: Krzhizhanovskaya VV, Závodszky G, Lees MH et al (eds) Computational science - ICCS 2020. Lecture notes in computer science, vol 12139. Springer, Cham, pp 186–198
13. Zhang Z, Jimack PK, Wang H (2021) MeshingNet3D: Efficient generation of adapted tetrahedral meshes for computational mechanics. Adv Eng Softw 157–158:103021
14. Wang N, Lu P, Chang X et al (2021) Preliminary investigation on unstructured mesh generation technique based on advancing front method and machine learning methods. Chin J Theor Appl Mech 53(3):740–751 (in Chinese)
15. Lu P, Wang N, Chang X et al (2022) An automatic isotropic/anisotropic hybrid grid generation technique for viscous flow simulations based on an artificial neural network. Chin J Aeronaut 35(4):102–117
16. Chen X, Liu J, Pang Y et al (2020) Developing a new mesh quality evaluation method based on convolutional neural network. Eng Appl Comput Fluid Mech 14(1):391–400
17. Chen X, Liu J, Gong C et al (2021) MVE-Net: An automatic 3-D structured mesh validity evaluation framework using deep neural networks. Comput Aided Des 141:103104
18. Wu T, Liu X, An W et al (2022) A mesh optimization method using machine learning technique and variational mesh adaptation. Chin J Aeronaut 35(3):27–41
19. Owen SJ, Shead TM, Martin S (2020) CAD defeaturing using machine learning. In: Proceedings of the 28th International Meshing Roundtable. CERN Data Centre, Geneva, pp 348–365
20. Wen C, Zhang Y, Li Z et al (2019) Pixel2Mesh++: Multi-view 3D mesh generation via deformation. In: Proceedings of the 2019 IEEE/CVF International Conference on Computer Vision (ICCV). IEEE, Piscataway
21. Wei X, Chen Z, Fu Y et al (2021) Deep hybrid self-prior for full 3D mesh generation. In: Proceedings of the 2021 IEEE/CVF International Conference on Computer Vision (ICCV). IEEE, Piscataway
22. Schoofs AJG, Van Beukering LHThM, Sluiter MLC (1979) A general purpose two-dimensional mesh generator. Adv Eng Softw 1(3):131–136

23. Talbert JA, Parkinson AR (1990) Development of an automatic, two-dimensional finite element mesh generator using quadrilateral elements and Bezier curve boundary definition. Int J Numer Methods Eng 29(7):1551–1567
24. OCCT (2022) http://www.opencascade.com. Accessed 15 Aug 2022
25. Chen J, Xiao Z, Zheng Y et al (2017) Automatic sizing functions for unstructured surface mesh generation. Int J Numer Methods Eng 109(4):577–608
26. Chen JJ (2006) Unstructured mesh generation and its parallelization. Dissertation, Zhejiang University (in Chinese)
27. Hagiwara K (2003) A review of research for multi-layer perception. IEICE Tech Rep Neurocomputing 103:7–12
28. Paszke A, Gross S, Massa F et al (2019) PyTorch: an imperative style, high-performance deep learning library. In: Proceedings of the 33rd International Conference on Neural Information Processing Systems (NeurIPS 2019). Neural Information Processing Systems Foundation, Inc., Vancouver
29. Mittal K, Fischer P (2019) Mesh smoothing for the spectral element method. J Sci Comput 78:1152–1173
30. Liu WH, Sherman AH (1976) Comparative analysis of the Cuthill–McKee and the reverse Cuthill–McKee ordering algorithms for sparse matrices. SIAM J Numer Anal 13(2):198–213
31. Geuzaine C, Remacle JF (2009) Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. Int J Numer Methods Eng 79(11):1309–1331
32. Lu F, Qi L, Jiang X et al (2020) NNW-GridStar: Interactive structured mesh generation software for aircrafts. Adv Eng Softw 145:102803
33. Srivastava N, Hinton G, Krizhevsky A et al (2014) Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 15(1):1929–1958

Publisher's Note