Neural Networks 136 (2021) 152–179

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

How to teach neural networks to mesh: Application on 2-D simplicial contours

Alexis Papagiannopoulos [a,*], Pascal Clausen [b], François Avellan [a]

[a] EPFL, Departement of Mechanical Engineering, Laboratory of Hydraulic Machines (LMH), Avenue de Cour 33 Bis, 1007 Lausanne, Switzerland
[b] Osylum SA, Route Cantonale 7, 1844 Villeneuve, Switzerland

ARTICLE INFO

Article history:
Received 16 December 2019
Received in revised form 19 October 2020
Accepted 17 December 2020
Available online 2 January 2021

Keywords:
Mesh generation
Simplicial mesh
Neural networks
Machine learning

ABSTRACT

A machine learning meshing scheme for the generation of 2-D simplicial meshes is proposed based on the predictions of neural networks. The data extracted from meshed contours are utilized to train neural networks which are used to approximate the number of vertices to be inserted inside the contour cavity, their location, and connectivity. The accuracy of the scheme is evaluated by comparing the quality of the mesh generated by the neural networks with that generated by a reference mesher. Based on an element quality metric, after conducting tests on contours for a various number of edges, the results show a maximum average deviation of 15.2% on the mean quality and 27.3% on the minimum quality between the elements of the meshes generated by the scheme and the ones generated from the reference mesher; the scheme is able to produce good quality meshes that are suitable for meshing purposes. The meshing scheme is also applied to generate larger scale meshes with a recursive implementation. The findings encourage the adaption of the scheme for 3-D mesh generation.

1. Introduction

Meshes are applied in multiple scientific areas such as computer graphics (Portaneri et al., 2019; Sumner & Popović, 2004), Computer Aided Design (CAD) (Beniere et al., 2013; Lavoué et al., 2005), structural analysis (Bennet & Botkin, 1985), and fluid dynamics (Baker, 1997; Lohner, 1995). Problems encountered in these areas require the construction of simplicial elements (triangular in 2-D, tetrahedral in 3-D) that fill the interior domain of an object (cavity) (Fig. 1) and satisfy criteria in terms of shape and size (quality). Mesh generation algorithms have to guarantee robustness, be adaptive to the complexities of a geometry, and fit the element shape requirements to satisfy solution accuracy of the underlying physics (Owen, 1998). To ensure efficiency, post mesh improvement heuristics that move the vertices of the elements and re-adjust their connectivity might be required. Altogether, the requirements imposed on mesh generation and improvement lead to algorithms that suffer from high complexity and speed limitations. Preparing a mesh for hydraulic simulation can take a large part of the simulation time as well; both global and local remeshing during simulation steps is currently barely practical in terms of speed and robustness. There is therefore a demand for a computationally efficient meshing framework that is able to comply to the needs and exceptions of a geometry, satisfy mesh requirements, and avoids as much as possible explicit treatment. Machine learning algorithms use computational methods that are able to solve complex problems based on data observation and pattern recognition without relying on predetermined equations or explicit algorithm implementation. Machine learning has the potential to be a useful tool for mesh generation. Existing meshing algorithms are able to provide datasets to train machine learning models for mesh generation without the need of the underlying meshing technique involved. This data driven approach to mesh generation has the potential to provide an automated meshing framework that bypasses the complexities and computational hurdles of existing meshing algorithms. It is, therefore, within the aim of the present paper to simplify the process of meshing and to adapt mesh generation to an automatic data driven framework using machine learning tools that could help overcome the aforementioned issues.

1.1. Mesh generation algorithms and challenges

There are various approaches to generate simplicial meshes in a geometry. Constrained Delaunay mesh generation algorithms (Paul Chew, 1989; Shewchuk, 2002) start with bounding simplicial elements (triangles in 2-D, tetrahedra in 3-D) containing the geometry. The domain that includes the points of the initial elements along with the boundary points of the geometry is tessellated according to the Delaunay criterion that dictates that no vertex should be included in the circumscribed hyper-sphere

* Corresponding author.
E-mail address: alexis.papagiannopoulos@epfl.ch (A. Papagiannopoulos).

https://doi.org/10.1016/j.neunet.2020.12.019
0893-6080/

Nomenclature

V	Interior domain (cavity)
∂V	Boundary of domain (contour)
N_C	Number of contour edges
P_C	Contour vertices coordinates
l_s	Target edge length
N_I	Number of inner vertices
P_I	Inner vertices coordinates
G	Grid
S_G	Scores of grid points
A	Connection table
q_{worst}	Worst quality value of element in mesh
q_{mean}	Mean quality value of elements in mesh
e_{worst}	Relative difference of q_{worst} between predicted mesh and reference mesh
e_{mean}	Relative difference of q_{mean} between predicted mesh and reference mesh

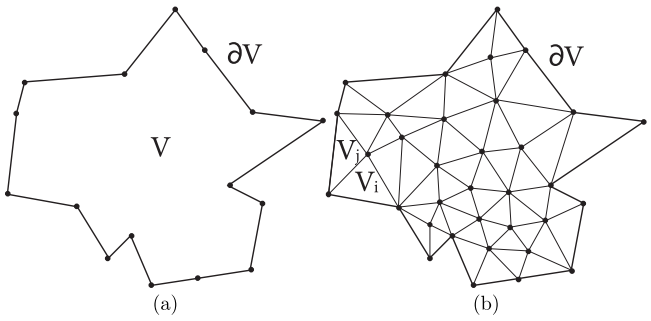


Fig. 1. (a) A set of points and edges defines a closed boundary ∂V (contour) with an interior continuous domain V (cavity). (b) To form a good quality mesh consisting of simplicial (triangular) elements whose edges respect a specific length, vertices are inserted in strategic locations of the interior domain V and are connected resulting in the discretization of V into $V_i, i = 1, 2, \ldots, N_{el}$ triangular elements, where N_{el} is the number of elements such that $\cup_{i=1}^{N_{el}} V_i = V$. The intersection of subdomains $V_i \cap V_j$ is at most an element edge; the tessellated domain does not contain intersections between the elements of the mesh.

of an element. All elements that are located in the exterior of the geometry are then deleted. As a next step, to avoid the appearance of thin elements (sliver), constrained Delaunay algorithms may strategically insert new points (Steiner points) to create new elements (Delaunay refinement). Advancing front methods (Mavriplis, 1995; Schöberl, 1997) generate a mesh by progressively adding mesh elements starting from the boundaries of the geometry. At each of the iterations, new elements are created by inserting a point inside the boundary and connecting it with a boundary face. After the extraction of the elements, the boundary is updated. The location of the inserted point has to meet a number of criteria that require complex analysis of the surrounding region. This iterative process stops when no boundaries exist. Other methods involve the use of block grids with adaptive size (quad-trees in 2-D, oct-trees in 3-D) that contain the geometry to be meshed (Fujita et al., 2016; Pajarola et al., 2002). The grid cells located in the interior of the domain are trivially meshed. The boundary cells need special handling such as moving them to the interior of the geometry according to physics-based simulations (Neil Molino & Fedkiw, 2003) or snapping the vertices of the cell to the input surface and then cutting the crossing elements (Labelle & Shewchuk, 2007). Mesh generation algorithms are not always able to produce a good quality mesh. As a result, to improve the quality of a generated mesh, improvement heuristics must follow. Sliver exudation (Cheng et al., 2000) deletes sliver tetrahedral elements based on an extension of the Delaunay criterion and a weight assignment to the points of elements that satisfy a ratio property. Variational methods (Alliez et al., 2005) improve the quality of a mesh through an iterative process where all the vertices are moved (global smoothing) and the connectivity is updated at each step to minimize a global energy over the mesh domain. Local re-meshing algorithms improve the mesh quality by relocating the position of a vertex to improve the quality of the adjoint elements (local smoothing) and/or by changing the mesh topology locally, i.e. removing elements to regions around the badly-shaped element to replace them with other elements occupying the same space (topological operations) (Klingner & Shewchuk, 2008; Wicke et al., 2010). Topological operations include: (1) edge removal that changes the connections of elements which share the same face along with its inverse operation of multiface removal, (2) vertex insertion that inserts a vertex, deletes elements around it to create an unmeshed subdomain, and then meshes this subdomain by connecting its faces with the vertex, and (3) edge contraction that removes an edge from an element and replaces the two endpoints with a single vertex. Different combinations among these operations are also possible, leading to greater improvement at the cost of more complex algorithms and speed limitations.

Meshing algorithms often account for codes of extensive size and are based on heuristic techniques. For example, in dynamic mesh-based simulations (Misztal et al., 2012), where the vertices move according to the motion of the material points, the quality of the mesh has to be improved at each step of the simulation to ensure a converging solution. Mesh improvement is achieved through local re-meshing algorithms that apply re-meshing operations heuristically (Clausen et al., 2013). This re-meshing process involves large computational times and is not always robust. For the tessellation of complex geometries, adaptivity is an essential mesh property. Physical objects are usually simple and smooth in some parts, complex and contorted in others. For accuracy in engineering solutions in fields such as hydraulic simulations to study the flow around turbines (Gupta & Prasad, 2012), structural analysis of complex geometries (dam, oil platform) (Camata & Coutinho, 2013; Lemos et al., 2008), and aerodynamic analysis of aircraft wings (Hassan et al., 1996), the elements in a region of complex geometry must be sufficiently small. Moreover, the shape of the elements must be elongated in certain directions (anisotropic elements) according to the needs of the physical problem under study. The creation of such optimal adaptive meshes is a complicated task and demands many computational resources. Meshing algorithms are difficult to be transported to acceleration platforms such as GPU architectures limiting the choices of computational frameworks.

The analysis to generate a compatible mesh that respects the geometric features and the accuracy requirements of numerical solutions takes up to 80% of the whole meshing procedure on account of automation absence (Hughes et al., 2005). The automation of the meshing procedure is still considered a critical bottle neck. In recent years, several strategies have been proposed to lessen user intervention to generate good quality meshes. In real world applications, the meshing of input surfaces can be proven a complicated problem as they may contain small gaps, self-intersections etc. There is no robust and automatic way to cleanup a surface. Users have to perform a cleanup manually to assure a well-defined input which can be proven to be a computational laboring work. If the input is not well defined, a raw implementation of a meshing algorithm could either not be robust or generate a mesh lacking features of the original geometry. To this extend, an automation aspect of the mesh generation procedure includes techniques to robustly generate high

quality meshes of the input geometry and preserve its features as much as possible even if the input is not well defined. In Hu et al. (2018), a pipeline method is presented to generate tetrahedral meshes taking as input triangle surfaces that represent the boundary of a geometry without connectivity information, also known as triangle soups. Initially, a bounding box that encloses the vertices of the input triangle surfaces is constructed and a Delaunay tetrahedralization is performed. To conform to the input boundary and resolve potential self-intersections of the input geometry, a binary space partitioning is performed to detect convex polygons formed by the intersection between the elements of the initial mesh and the triangle faces of the input geometry. The vertices of the polygons are connected with their barycenter to form an initial volumetric mesh within a bounding box larger than the triangle soup. Mesh improvement operations are followed to improve the quality of the mesh. Finally, the winding number filtering algorithm is applied to remove all the elements outside the boundary of the input geometry. In a similar automation context, Guo et al. (2019) propose an automatic triangular surface mesh generation pipeline for CAD models comprised of surface patches. An initial simple coarse triangulation is performed which tessellates the CAD model by preserving the geometric fidelity. This coarse triangulation may include inverted or self-intersecting elements caused by the intersection between the inner and outer boundaries or when a point of the patch is too close to a curve of the geometry. Therefore, a procedure is followed to delete invalid edges and to insert points to avoid an invalid topology. Next, based on the initial triangulation, each patch is meshed in a parametric 2D domain using Constrained Delaunay Triangulation and, then, is mapped back to the 3D object space. Finally, a mesh improvement routine is applied to improve the quality of the mesh. Automation can also refer to either the inclusion of additional features or the modification to the body of an established mesh generation algorithm. For example, an automated meshing process is presented in Ma and Sun (2019) based on the advancing front method for the creation of quadratic elements. To address the issue of manual domain decomposition for mesh generation with internal feature constraints, the algorithm is enriched with an automatic subregion decomposition that is defined by constraint lines and points in the interior of a geometry. After performing the decomposition, each subregion is then meshed separately. The extension of the quad-tree algorithm for the creation of quadrilateral elements is addressed in Pochet et al. (2016), where for a better mesh adaption, points of cells near a target curve geometry are attracted to the curve. Next, for a better approximation of the geometry, the cells near the curve are subdivided using six refinement cells instead of four. Other automatic procedures focus on generating meshes that are suitable computational domains to apply numerical methods. For example, starting from a geometry represented by a stereolithographic (STL) model, Liu et al. (2017) propose an automatic routine that generates polyhedral meshes suitable for the application of scaled boundary finite method to perform stress analysis. An octree grid is created to enclose the geometry and then the mesh is constructed by applying trimming operations that take into account the recovery of sharp features first to the edges, then faces and finally cells. The trimming operations are based on signs assigned to the vertices of the grid that represent whether they are located inside, outside or on the geometry. The aforementioned work names only a few of the latest advancements on the automation of mesh generation. Although these methods are proven to be efficient and are a step forward to counter explicit user handling, they still involve the development of complicated algorithms that may include an extensive computational time trade-off. The approach taken in the proposed meshing scheme, is focused in using machine learning tools to accommodate the creation of a direct meshing framework that avoids the implication of convoluted algorithms and the need for post treatment of the mesh. In particular, neural networks that are proven to be effective for problems that are complex and time consuming are integrated in an automatic mesh generation procedure.

1.2. Related work

In recent years, advancements in NNs have led to outstanding performances when applied for tasks of object classification and semantic segmentation using images (Calisto & Lai-Yuen, 2020; Chen et al., 2016; ElAdel et al., 2017; Sermanet et al., 2014; Simonyan & Zisserman, 2015). Convolution and pooling layers are able to take advantage of the Euclidean regular grid like structure of images to extract local features and offer an invariant framework to variations of an input (Krizhevsky et al., 2012; LeCun, 2012). Recently, there has been an increasing interest to generalize deep learning methods to non-Euclidean structured data such as graphs and manifolds, with a variety of applications from the domains of network analysis, computational social science, or computer graphics. Geometric deep learning (Bronstein et al., 2017) refers to the field of deep learning applied to non-Euclidean data such as graphs or discrete manifolds (i.e. meshes). To generalize convolution in graphs a spectral approach is considered; Observing that the complex exponential corresponds to the eigenfunctions of the Laplacian operator in Euclidean domains, the eigenfunctions of a graph Laplacian operator are considered as a generalized version of the typical Fourier basis. Graph convolution (Bruna et al., 2014; Defferrard et al., 2016; Henaff et al., 2015; Kipf & Welling, 2017) can be achieved by projecting a provided signal to the eigenfunctions of the graph Laplacian operator (graph Fourier transform), multiplying the obtained spectrum with a set of learnable spectral filter coefficients and projecting everything back to the original domain. This intuition has led to good results for signals defined over one graph. However, graph Laplacian eigenfunctions are inconsistent across different domains (basis-dependent). To extend convolution in a consistent way across different domains, spatial approaches suggest the application of filters to local patch operators that are intrinsic to a mesh. Such patch operators include the use of geodesic polar coordinates (Masci et al., 2015), anisotropic heat kernels (Boscaini et al., 2016), and a family of learnable mixture Gaussian kernels (Monti et al., 2017). Other approaches to adapt NN architectures to mesh topological processing suggest the mapping of a mesh to a flat torus topology (Maron et al., 2017); 2D CNN operators are well defined over a grid of the flat torus. In Verma et al. (2018), the authors propose a dynamic approach to convolution since the operator is calculated according to the features of a vertex. Tangent convolution (Tatarchenko et al., 2018) projects the local surface geometry on a tangent plane of the vertices of a mesh yielding a set of tangential images that can be treated as 2D grids upon which the convolution operator can be applied. MeshCNN (Hanocka et al., 2019) is an NN architecture with convolution, pooling and unpooling layers that are adapted to take into account the properties of triangular surfaces. Using as input invariant descriptors of the edges of a mesh, pooling and unpooling operations are defined to collapse edges for the task of mesh simplification.

The advancements of integrating mesh topology to NN architectures are proven to be efficient for tasks such as mesh classification, mesh segmentation and shape correspondence, and open the path for extended applications. In Baqué et al. (2018), given a mesh, Geodesic Convolutional Neural Networks (GCNN) (Monti et al., 2017) are trained to emulate fluid dynamics simulations by regressing the vector and scalar field values (e.g. pressure

and drag) over the discrete domain. The GCCN can also be used in an objective function to optimize the mesh representation of shape using an ADAM algorithm with respect to a desirable physics effect. In Wang et al. (2018) the Pixel2Mesh architecture takes as an input a 2D RGB image to generate a mesh representing the depicted object. The architecture combines the classic CNN networks to extract feature from the image with a Graph Convolutional Network (GCN) (Defferrard et al., 2016) to deform an initial coarse ellipsoid mesh with triangular faces. The vertices of the mesh are adjusted using the GCN while a graph unpooling layer inserts new vertices to the edges of the mesh to refine regions according to the geometrical features of the input image. The architecture is later on extended for generating meshes from multi-view images (Wen et al., 2019). Litany et al. (2018) use the local spatial patch operators of Monti et al. (2017) to introduce a variational autoencoder that performs shape completion. Although these recent advancements show a promising path for mesh related tasks, they assume the existence of an initial mesh that can be given as input.

The integration of NNs to the mesh generation procedure has been previously studied in both an unsupervised and supervised learning setting. Self-organizing maps (SOM) (Fort, 2006; Kohonen, 2013; López-Rubio & Ramos, 2014) are unsupervised learning NNs that map multidimensional data onto lower dimensional subspaces where geometric relationships between linked neurons indicate their similarity. The weights of the neurons are adjusted using a competitive learning algorithm; at each iteration, the neurons compete each other to win an input pattern and only one neuron is activated and declared as a winner (Best Matching Unit). The weights of the neurons that are linked with the winner neuron are updated to better fit the input pattern while the weights of the other neurons remain unchanged. SOMs have been utilized for generating meshes based on input patterns that correspond to a set of points distributed inside a geometry by a density function (Ahn Chang-Hoi et al., 1991; Manevitz et al., 1997; Nechaeva, 2006). The density function dictates which parts of the geometry should be approximated by more elements than elsewhere. In the meshing context, a SOM is a fixed grid (triangular or quadrilateral) of linked neurons that adapt their point coordinates (weights) according to the points of the mesh density function using competitive learning. A main drawback of using SOM is that the size of the used grid is fixed which may cause the appearance of badly shaped elements for irregular mesh density functions. Moreover, strategies have to be adopted to ensure that the generated mesh fits the boundary and that elements do not appear outside the boundary of non-convex geometries. In order for the SOM to fit the boundary of the geometry domain Manevitz et al. (1997) suggest an interweaving algorithm of 1D and 2D competitive learning between boundary neurons and interior neurons. For non convex geometries, elements that are located out of the boundary are simply deleted. Based on the interweaving algorithm, Nechaeva (2006) presents an improved algorithm that is parallelizable and adapts better on the boundary of non convex geometries.

To counter the drawback of the SOM's fixed size grids Let-it-grow (LIG) neural networks (Alfonzetti et al., 1996; Triantafyllidis & Labridis, 2002) are suggested for mesh generation. LIG networks start with an initial coarse grid and additional neurons are added in accordance to a density function. During the first phase of the mesh generation algorithm using LIG networks, using competitive learning like SOMs, a point (input pattern) is provided by the density function, the BMU is located and is moved along with its connected neurons towards the position of the point. Next, a signal counter that is assigned to the BMU is incremented and the mesh is checked for topological validity. This is repeated for a specified number of iterations. During the second phase, a new node is added in the midpoint of the edge that contains the neuron with the maximum signal counter and the furthest neighborhood neuron. During the learning process the Delaunay criterion can be used to adjust the connections between the neurons. The algorithms iterate until a desirable number of nodes is achieved. One main drawback of LIG networks lies in the computational complexity of finding the BMU when the mesh includes a large number of neurons. Triantafyllidis and Labridis (2002) suggest an initial Constrained Delaunay Triangulation applied to the boundary of the geometry to form the initial grid upon which LIG competitive learning is applied. Moreover, algorithms that reduce the computational complexity of finding the BMU are presented for the generation of large meshes.

The Growing Neural Gas (GNG) network model also referred as Topology learning network (Fritzke, 1995; Martinetz & Schulten, 1994) is an unsupervised learning algorithm that uses competitive learning but unlike SOM and LIG networks does not need an initial specification on the number of neurons or prefixed connections; additional neurons are added and the connections are adjusted as long as the algorithm keeps running. The algorithm starts by adding an initial input pattern and generate neurons that are connected by an edge. The BMU is moved closer to the input pattern as well as all the neurons connected to the BMU. Next, the second BMU (SBMU) is determined. If the BPU and SBMU are connected the age of the edge is set to zero otherwise the neurons are connected. The age of edges emanating from the BPU is then incremented. If an edge has an age larger than a maximum age threshold then it is deleted. If the deletion of the edge results in neurons with no edges then they are deleted as well. After a specified number of iterations, a neuron is inserted halfway between the edges of the worst matching unit and the neighbor neurons. The process is iterated until a specified condition is met, such as a maximum number of iterations. The method is proven to be able to create Delaunay triangulations (Martinetz & Schulten, 1994) under a proper distribution of training input patterns. A raw implementation of GNN to a point cloud however may not result in a topologically valid mesh; after the termination of the algorithm, the set of neurons and edges may not result in global triangle coverage. Therefore, to acquire a triangular mesh, post processing steps are required like removing invalid edges that are not adjacent to two triangular elements, face reconstruction, face reorientation, and fill potential cavities with elements (Holdstein & Fischer, 2008; Melato et al., 2007).

Using supervised learning, in Yao et al. (2005), a NN is utilized to accommodate the meshing process with 2-D quadrilateral elements using the advancing front method. The NN is used to bypass heuristic 'if-then' rules of the element extraction process that define a good quality element. The coordinates of some boundary points are the input of the NN and the parameters that are used to create good quality quadrilateral elements are the output. The construction of training samples relies on manually determining the patterns of good quality quadrilateral elements. The NN is able to extract good quality elements, however, it is restricted to be trained with a limited number of boundary points as the complexity to find training patterns increases with the increase in the number of boundary points that are included as the input to the NN.

In Vinyals et al. (2015), pointer networks are introduced as a new neural architecture. The networks consist of an encoding and a decoding Recursive NN (RNN). At each step of the decoding process, a pointer selects a member of the input sequence as the output. Unlike sequence-to-sequence models (Sutskever et al., 2014) and Neural Turing Machines (Graves et al., 2014), the size of the output does not need to be fixed a priori. Pointer networks are applied to explore the application of NNs for combinatorial problems where the size of the output is variable and depends

on the size of the input. The problem of Delaunay triangulation is examined on a set of points using this architecture. The inputs are the coordinates of the points and the output is a set that contains triplets of integers that correspond to the order of the input and represent the vertices forming the triangles. As the network is not directly addressed for mesh generation, the resulting meshes may have intersecting connections and partial triangle coverage.

A recent addition for mesh generation with NNs using supervised learning is the MeshingNet NN (Zhang et al., 2020) guides standard mesh software to generate meshes with an element distribution that provides accurate solutions when solving Partial Differential Equations (PDE) with FEM. The traditional approach to generate such a mesh involves an a posteriori error estimation. An initial solution is calculated on a relatively coarse mesh and then auxiliary problems are calculated on an element or a patch of elements to approximate local errors. The local errors can be used in conjunction with the initial solution to approximate a global error that can indicate which regions of the mesh should either contain more elements (refinement) or potentially contain less elements while not harming the accuracy of the solution. MeshingNet takes as input the coordinates of the geometry's boundary (polygon), parameters of the PDE and the mean value coordinates of a point (parametric coordinates relative to the vertices of a polygon) inside the geometry to output the local area upper bound around that point that dictates the local element distribution. To train the network, the local area upper bound is calculated based on the error acquired by solving the PDE on a low density uniform element mesh and a high density uniform element mesh. After the training, a low density uniform mesh is generated for the given geometry using an external mesher and the vertices of the mesh are given as input to the network. Based on the local area upper bound of the inner vertices, the external mesher is called to refine accordingly.

Despite the variety of approaches in the use of machine learning frameworks for mesh generation, none of these methods can currently replace the standard mesh generation algorithms. The integration of machine learning in the field of mesh generation remains an area open exploration with great potential. The presented meshing scheme proposes a novel methodology that incorporates and adapts the topological properties of mesh generation to accommodate a machine learning framework using NNs.

1.3. Problem statement and proposed meshing scheme

In this paper, the problem of simplicial mesh generation for a target element size is studied in bounded domains. The boundary of the domains ∂V, i.e. contour, is composed of piecewise linear segments forming the edges of the contour (Fig. 1). The continuous interior domain V, i.e. cavity, is then tessellated into $V_i, i = 1, 2, \ldots, N_{el}$ triangular elements. To achieve a good quality mesh composed of a target element edge size (target edge length), inner vertices are strategically placed in the cavity of the contour. The final mesh is topologically valid if each element edge is incident to only one or two elements and there are no element entanglements (manifold condition).

A data driven meshing framework using NNs is proposed. The presented meshing scheme uses NNs to generate triangular meshes of good quality on 2D contours for a target element size. Each of the NNs is assigned to predict a step towards mesh generation. Three NNs are used to predict the number of inner vertices that must be inserted inside the cavity of the contour, their location, and how to connect vertices to form elements. The predictions of a NN assigned to predict a step of the meshing scheme are pipelined to proceed to the next step. The meshing scheme generates meshes, based on datasets of meshed contours that are generated using a reference 2D meshing algorithm. Here, the results using the Constrained Delaunay Triangulation algorithm as a reference mesher are presented. The strategies applied to acquire the final mesh are transparent to 3D and have the potential to be carried out for tetrahedral mesh generation.

The meshing scheme is trained using meshed contour datasets and generates meshes without any post treatment by providing information about a contour and a target element size. The resulting mesh is guaranteed not to contain any intersecting elements using an intermediate meshing algorithm that is based on predictions of a NN. Since the intermediate meshing algorithm used in the meshing scheme is not coupled with the underlying reference mesher, this offers the possibility to adapt the scheme to the behavior of any 2D simplicial meshing algorithm. The meshing scheme offers also a computational advantage over previous works using machine learning techniques, since the number and location of inner vertices that are inserted in the cavity of the contour to satisfy element size criteria is predicted a priori the connection phase. This offers a more direct pathway to acquire a mesh that satisfies element size criteria; it avoids the incremental creation of a mesh by inserting inner vertices or elements one by one and skips intermediate topological consistency (manifold) checks. A trade-off of automating the meshing process as much as possible lies in a more evident impact of the approximation error of the NNs to the overall quality of the resulted mesh. The main scope of this paper is therefore to explore the integration of NNs and study their accuracy when used as a main component for robust mesh generation.

2. Meshing scheme

2.1. Algorithm overview

The machine learning methods that are used for data with a grid-like underlying structure (e.g. image processing) cannot be applied to a meshing framework. Unlike the grid structure of pixel-based data, the underlying Euclidean space of possible contour inputs does not have such an organized structure. From our experience, a naive brute force method of providing as input the unprocessed point coordinates of a random contour does not result in sufficiently robust and accurate pattern recognition from the NNs. The proposed scheme is instead intentionally designed to ensure a high level of consistency among the data provided to the NNs such as to ease NNs learning abilities while minimizing the amount of learning data. To this end, the first stage of the presented algorithm consists of a pre-processing step that applies a feature transformation to best fit a reference contour circumscribed in a unitary circle. This transformation provides a scale and rotation invariant shape representation of the contours. Furthermore, the contour vertex indexing has to follow a specific ordering rule, e.g. clockwise or anti-clockwise, to underline patterns of vertex connections that form the edges of the contour.

To achieve a target element size, the target edge length is part of the inputs of the NNs of the scheme. The accuracy of an approximation of the inner vertices from a reference mesher is very crucial to the overall mesh output of the presented meshing scheme. The use of a grid covering the contour is opted to estimate the location of the inner vertices. The cells of a grid are associated with a distance function. The location of the inner vertices is stored in the form of a distance function to these vertices. A local interpolation is performed around grid points to maximize the accuracy of the approximation.

To avoid intersection between elements and opt for connections that lead to good quality elements, a triangulation algorithm is used. The triangulation algorithm is based on the output values of a NN. The output values of the NN represent the probability of

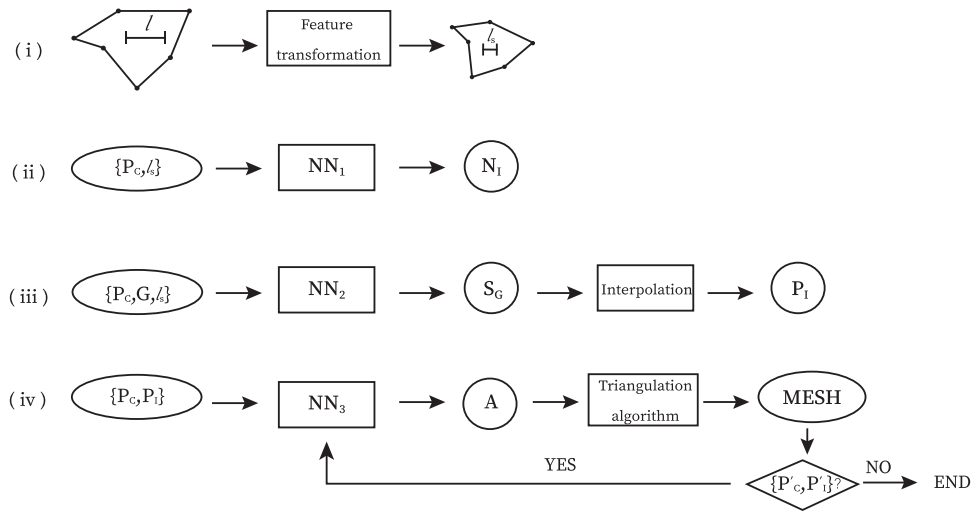


Fig. 2. The meshing scheme consists of four steps: (i) The initial contour with N_C edges is scaled and rotated with respect to a regular polygon with N_C edges inscribed in a unitary radius circle. (ii) NN_1 is used to approximate the number of inner points N_I, providing as input the contour vertex coordinates P_C and the requested target edge length l_s. (iii) NN_2 takes as input the vertex coordinates P_C, patches of grid points from G, and the target edge length l_s. It outputs the scores S_G for each grid point. Based on S_G, N_I grid points are selected and interpolation is applied to a region around them. Next, to approximate the inner vertices P_I, the local minimum of the interpolated surface is found. (iv) NN_3 takes as input the contour vertices P_C and the inner point vertices P_I and outputs the entries of a connection table A. The contour is meshed with a triangulation algorithm that meshes the cavity of the contour based on A. After the termination of the algorithm, if a sub-contour with P'_C contour coordinates is created containing N'_I inner vertices with P'_I coordinates, NN_3 is called recursively to mesh.

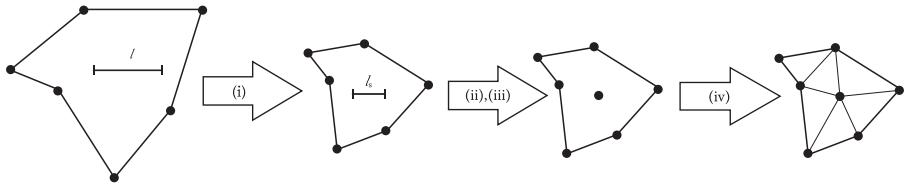


Fig. 3. From left to right: Steps followed to acquire the mesh of a contour. Step (i) of the algorithm consists of a feature transformation applied to the contour that causes the scaling of the target elements size of edge length l to l_s. Next, following steps (ii) and (iii) of the proposed meshing scheme, based on the prediction of NN_1, one vertex is inserted in the interior of the contour (cavity), and its location is predicted using NN_2. The final step (iv) of the meshing scheme uses NN_3 and a triangulation algorithm to connect the edges of the contour with inner vertices or contour vertices to create the mesh.

connecting the face of an element (edge) to another vertex of the mesh. The values are stored in a connection table which is used by the triangulation algorithm to assess the connections and avoid element entanglement.

The proposed meshing algorithm consists of the following steps (Figs. 2, 3):

(i) To mesh a contour of N_C edges for a user target element size of edge length l, first a feature transformation is applied to it. The contour is scaled and rotated with respect to a regular polygon of N_C edges that is inscribed in a circle of unit radius. This transformation also changes l to $l_s = Sl$, where S is the scaling factor of the transformation and l_s is the scaled target edge length.

(ii) The first neural network NN_1 takes as input the transformed contour vertex coordinates $P_C = \{p_i = (x_i, y_i), i = 1, 2, \ldots, N_C\}$ of the transformed contour and target edge length l_s. The output of NN_1 is the number of vertices N_I that should be inserted inside the cavity of the contour, i.e. the interior domain of the contour, to achieve the target edge length l_s.

(iii) The approximation of the coordinates of the inner vertices $P_I = \{p_{I,i}, i = 1, 2, \ldots, N_I\}$ is done with the help of a square grid G over the contour. The second neural network NN_2 takes as input the coordinates of the contour P_C, the target edge length l_s, and patches of G. A surface is defined over the grid whose local minima determine the most probable locations of the inner vertices. NN_2 outputs the values of the surface for the grid points contained in the input patch. Based on these values, N_I grid points are selected and a regional interpolation follows. Finally, the local minima of the interpolated surface reveal the approximated locations of P_I.

(iv) The third neural network NN_3 takes as input the vertex coordinates P_C of the transformed contour and the coordinates of the inner vertices P_I and outputs the entries of a connection table A. The connection table contains values representing the probability of connecting the face of an element (edge) to one of the inner vertices $p_{I,i}$ or with another contour vertex p_i. The mesh is created using a triangulation algorithm that is based on the values of the connection table. Because the triangulation algorithm connects contour edges with vertices, this may lead to the formation of inner sub-contours that are not meshed. In this case, the triangulation algorithm is called recursively to mesh the sub-contours.

Gmsh© is used as the reference mesher. *Gmsh*© is a wrapper implementing a Delaunay algorithm, written in *C++*, as presented in Lambrechts et al. (2008). The NNs are implemented and trained using *PyTorch* (Paszke et al., 2017). The triangulation algorithm used to predict the connection of the mesh is implemented in *Python*.

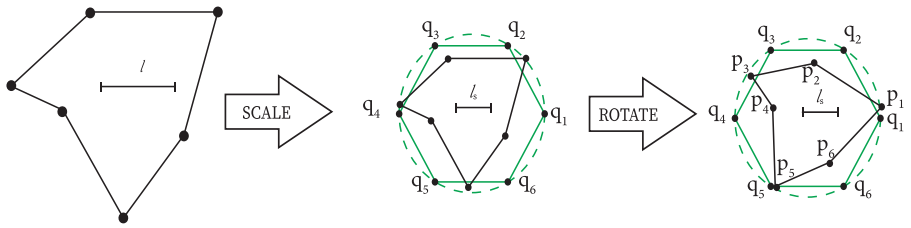


Fig. 4. Procrustes superimposition on contours with 6 edges with requested element size l. The reference contour is a regular hexagon inscribed in a unit circle with coordinates $Q = \{q_i, i = 1, 2, \ldots, 6\}$. The contour is scaled by a scale factor S, changing the target edge length from l to $l_s = Sl$, and rotated to best fit the point of the reference polygon to acquire the points of the transformed contour $P_C = \{p_i, i = 1, 2, \ldots, 6\}$.

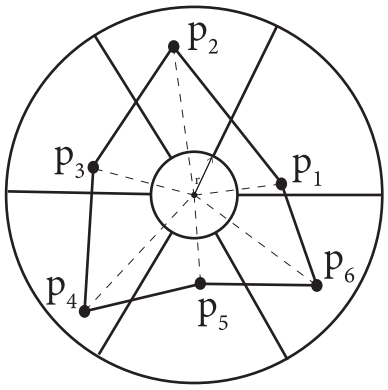


Fig. 5. Example of creation of contour with 6 edges. A unit circle is divided into 6 sectors. From each sector a point $p_i, i = \{1, \ldots, 6\}$ is selected. To avoid the creation of contours with very short edges, no points are selected from the inner region of a circle with small radius r.

2.2. Feature transformation and training data acquisition

The connectivity of a mesh is not changed if a contour is rotated or scaled. The contour to be meshed undergoes a feature transformation that assists pattern recognition by the NNs involved in the scheme. The required scaling and rotation invariance is achieved by applying the Procrustes superimposition on a reference contour (Gower, 1975). For a contour with N_C edges and P_C^* contour coordinates, a regular polygon is used with N_C edges inscribed in a unit circle as a reference. Procrustes superimposition imposes a linear transformation to the contour points $P_C^* = \{p_i^*, i = 1, 2, \ldots, N_C\}$ so that they best conform to the points of the reference contour $Q_C = \{q_i, i = 1, 2, \ldots, N_C\}$ (Fig. 4). This pre-processing step is essential to ensure consistency among the data provided to the NN. The centered Euclidean norms $\|P_C^*\| = \sum_{i=1}^{N_C} (p_i^* - \overline{p^*})^2$ and $\|Q_C\| = \sum_{i=1}^{N_C} (q_i - \overline{q})^2$, where $\overline{p^*} = \sum_{i=1}^{N_C} p_i^*/N_C$ and $\overline{q} = \sum_{i=1}^{N_C} q_i/N_C$, scale the coordinates of P_C^* and Q_C to the same unit norm by applying the transformation $P_{\|C\|}^* = P_C^*/\|P_C^*\|$ and $Q_{\|C\|} = Q_C/\|Q_C\|$. Next, Singular Value Decomposition (SVD) is applied to $A = Q_{\|C\|}^{\intercal} P_{\|C\|}^*$. SVD decomposes A to $A = UCV$ which yields the optimal rotation matrix $R = UV^{\intercal}$ and the scaling factor $S = \|Q_C\| tr(C)$. The coordinates of the transformed contour are given by $P_C = SP_{\|C\|}^* R + \overline{q}$.

The training of the NNs is based on sets of contours with N_C edges. To generate a mesh contour of N_C edges (Fig. 5), random points are chosen from a disk of unit radius that is divided into N_C sectors. A point is randomly selected from each sector and these points are connected subsequently to form a random contour. To avoid the creation of a contour with very short edges, the selection of points is excluded from the inner region of a circle with small radius r.

After applying the Procrustes transformation to the contour sets, the contours are meshed using Constrained Delaunay Triangulation (**CDT**) (Paul Chew, 1989) for multiple-scaled target edge lengths l_s. CDT generates a triangular mesh with respect to the boundaries of a contour and inserts inner vertices (Steiner points) to comply with the target edge length and satisfy quality criteria. Constraints are added to avoid vertex insertion along the edges of the contour. The number of inner vertices N_I inserted and their coordinates P_I are then used to train NN_1 and NN_2 respectively. The information on the inner vertices also allows to compute the connection table A of the contour that is used to train NN_3 (Fig. 6).

2.3. Approximation of inner vertices number

To predict the number N_I of inner vertices NN_1 is used. The N_I inner vertices that are inserted by CDT inside a contour is used to train NN_1 which outputs an approximation $\hat{N}_I$. Based upon the approximated number of inner vertices $\hat{N}_I$, the meshing scheme proceeds to approximate their location.

NN_1 is a feedforward NN with multilayer perceptrons. For a contour with N_C edges, NN_1 takes as input the contour vertex coordinates P_C and target edge length l_s (Fig. 7). The network is trained to minimize the loss function $L(N_I, \hat{N}_I) = |N_I - \hat{N}_I|$, where N_I is the number of vertices that are inserted using CDT and $\hat{N}_I$ is the number of vertices that are predicted by NN_1 (Alg. 1).

Algorithm 1: Training algorithm of NN_1 for the prediction of number of inner vertices.

P_C: Contour vertices coordinates
l_s: Target edge length
N_I: Number of inner vertices
$\hat{N}_I$: Estimated number of inner vertices
N_{train}: Number of training data population
Initialise weights W_K of NN_1
while *required number of iterations is not reached* **do**
  foreach *training example in* $D = \{(P_C^{(n)}, l_s^{(n)}, N_I^{(n)})\}_{n=1}^{N_{train}}$ **do**
    Compute $\hat{N}_I^{(n)}$ using current parameters
    Calculate loss function $L(N_I^{(n)}, \hat{N}_I^{(n)}) = |N_I^{(n)} - \hat{N}_I^{(n)}|$ and $\frac{\partial L}{\partial W_K}$
    Update W_K using Adam learning rate optimization
  end
end

2.4. Inner vertices positioning

To approximate the location of the inner vertices inside the cavity, a grid G of resolution $N_G = n_G \times n_G$ is first defined inside a boundary box that includes all the vertices P_C of the contour. After acquiring the coordinates of the inner vertices P_I from meshing a contour with CDT, the distance between each grid point and inner vertex in the contour is calculated. Each grid point is assigned a score, which is defined as the distance to the closest vertex. The scores of the grid points are used to choose those that are closest to the inner vertices (Fig. 8a). First, the grid point with the

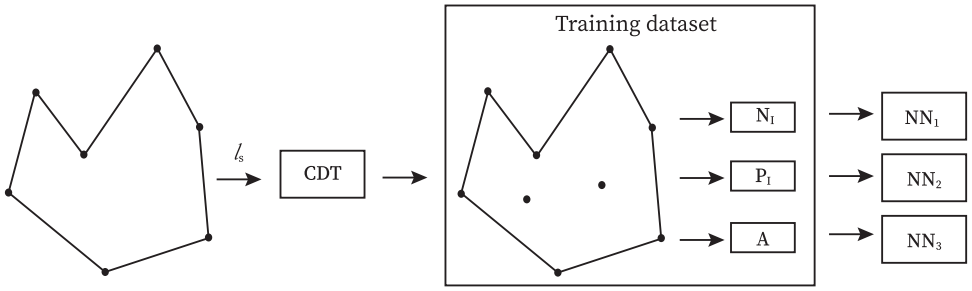


Fig. 6. The generated contours are meshed with CDT for various target edge lengths l_s. The number of inner vertices N_I and their coordinates P_I are used to train the NN_1 and NN_2. By knowing the location P_I of the inner vertices, the connection table A of the contour is calculated to be included in the training dataset of NN_3.

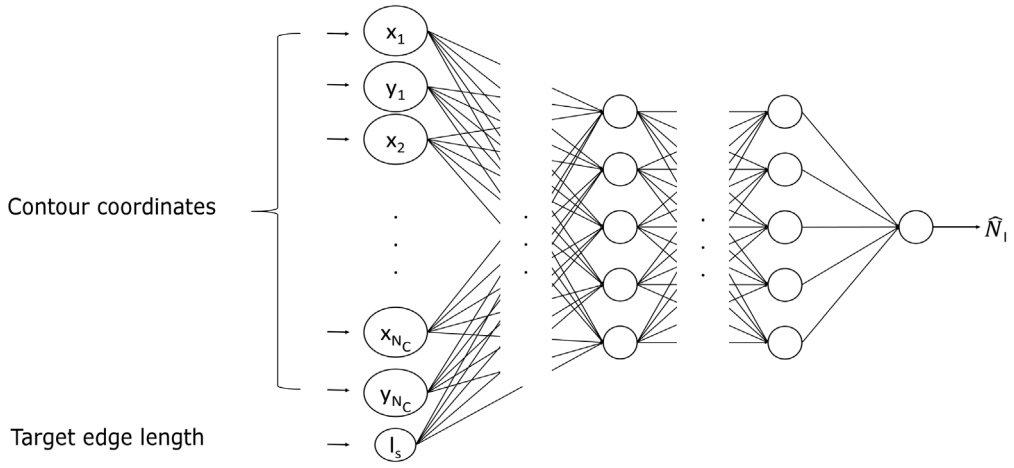


Fig. 7. NN_1 architecture for the prediction of number of vertices. It takes as input the contour coordinates $P_C = \{p_i = (x_i, y_i), i = 1, 2, \ldots, N_C\}$ and the scaled target edge length l_s. NN_1 outputs the approximation $\hat{N}_I$ of the number of inner vertices that should be inserted inside the cavity of the contour to achieve the target edge length l_s.

lowest score is selected. Next, interpolation is applied to find the scores on a local domain of the selected grid point to determine the local minimum (Fig. 8b). The minimum is the approximation of an inner vertex. The number of grid points included in the local domain depends on the target edge length l_s. Finally, the grid points around the selected grid point are excluded. The same procedure is applied for the next grid point with the lowest score to acquire the next approximation of an inner vertex (Fig. 8c).

NN_2 is used to predict the scores of the grid points. The use of the grid allows the adaption of the score of the grid points so that the predictions of NN_2 lead to valid triangulations and adhere to the target edge length. To avoid the predictions of vertices that are located outside the contour, the score of the grid points that are located near or outside the boundary of the contour is penalized. After choosing a grid point with minimum score, the grid points around it are restricted from being selected for the approximation of the subsequent inner vertex, to ensure that the predictions complied with the desired target edge length. This procedure avoids predictions of inner vertices that are too close due to inaccuracy of the scores by NN_2.

NN_2 is a feedforward NN with multilayer perceptrons that takes as input the contour vertex coordinates P_C, the coordinates of the grid points P_{G_k} contained in each patch, and the target edge length l_s (Fig. 9). It outputs the scores $s_{i,g}$ for the points contained in the patch. The objective loss function is the mean squared error $L(s_{i,j}, \hat{s}_{i,j}) = \sum_{i,j}(s_{i,j} - \hat{s}_{i,j})^2 / N_{G_k}$, where $s_{i,j}$ is the calculated score of the grid point $p_{G_{i,j}}$, $\hat{s}_{i,j}$ is the score that the NN outputs, and N_{G_k} is the number of grid points included in the patch (Alg. 2).

Algorithm 2: Training algorithm of NN_2 for the prediction of location of inner vertices.

P_C: Contour vertices coordinates
l_s: Target edge length
P_{G_k}: Coordinates of grid points contained in the patch
S_{G_k}: Scores of grid points contained in the patch
$\hat{S}_{G_k}$: Estimated scores of grid points contained in the patch
N_{train}: Number of training data population
Initialize weights W_K of NN_2
while *required number of iterations is not reached* **do**
  foreach *training example in* $D = \{(P_C^{(n)}, l_s^{(n)}, P_{G_k}^{(n)}, S_{G_k}^{(n)})\}_{n=1}^{N_{train}}$ **do**
    Compute $\hat{S}_{G_k}^{(n)}$ using current parameters
    Calculate loss function $L(S_{G_k}^{(n)}, \hat{S}_{G_k}^{(n)}) = ||S_{G_k}^{(n)} - \hat{S}_{G_k}^{(n)}||_2^2 / N_{G_k} = \sum_{i,j}(s_{i,j}^{(n)} - \hat{s}_{i,j}^{(n)})^2 / N_{G_k}$ and $\frac{\partial L}{\partial W_K}$
    Update W_K using Adam learning rate optimization
  end
end

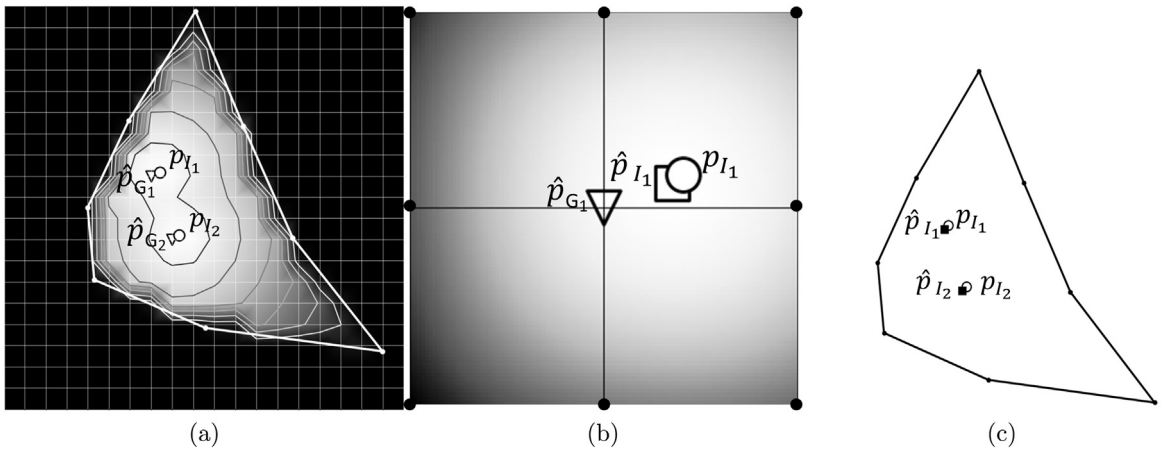


Fig. 8. Example of the NN_2 approximation of two inner vertices $p_{I,1}$ and $p_{I,2}$ ($N_I = 2$) for a contour with 8 edges. (a) Based on the scores S_G, the grid points $\hat{p}_{G_1}$ and $\hat{p}_{G_2}$ are selected as the first two grid points with the minimum score. (b) Then, interpolation is applied to a local region around them. Here, interpolation is applied to find the scores on a region around $\hat{p}_{G_1}$. This region includes the grid points around $\hat{p}_{G_1}$. The number of grid points included in the region depends on the target edge length l_s. By locating the local minimum of the interpolated surface, the approximation $\hat{p}_{I,1}$ of $p_{I,1}$ is acquired. (c) The interpolation procedure is also applied to $\hat{p}_{G_2}$ to obtain $\hat{p}_{I,2}$.

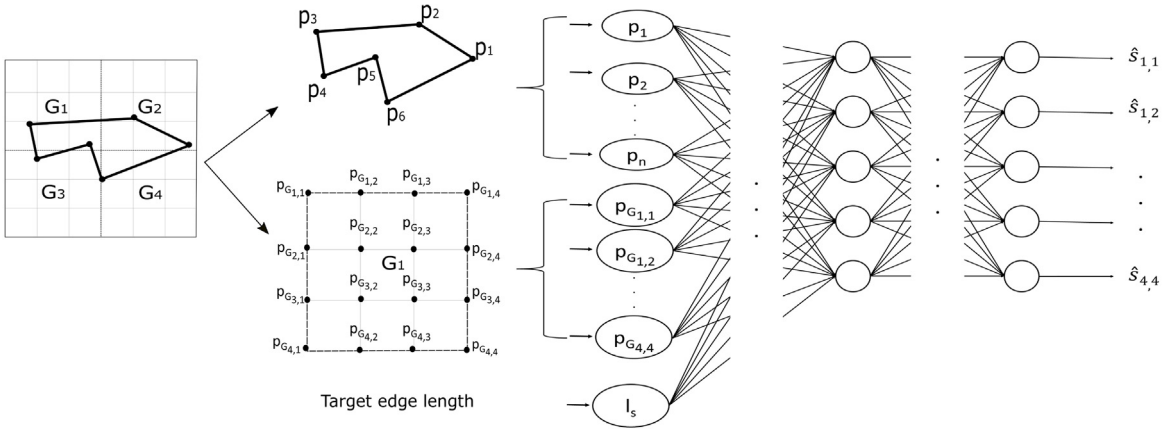


Fig. 9. The grid G defined over the contour is divided into N_p patches G_k, $k = \{1, 2..N_p\}$ (here $N_p = 4$). NN_2 takes as input the contour coordinates $P_C = \{p_i, i = 1, 2, \ldots, N_C\}$, the coordinates of the N_{G_k} grid points $P_{G_k} = \{p_{G_{(i,j)}} : (i, j) = \{1, 2, \ldots, a\} \times \{1, 2, \ldots, a\}, a^2 = N_{G_k}\}$ that are included inside a patch, and the target edge length l_s. It outputs the scores $\hat{s}_{i,j}$ that correspond to each grid point inside the patch.

2.5. Connectivity scheme

2.5.1. Triangulation algorithm

The final step of the meshing algorithm is to find the most probable connections between a face (edge) of the contour and a vertex, that would lead to a good quality mesh. The connection table A is a tool that lists the different probabilities for each contour edge to be connected with either a contour vertex or an inner vertex. Based on the values of A, a triangulation algorithm is used to mesh the contour.

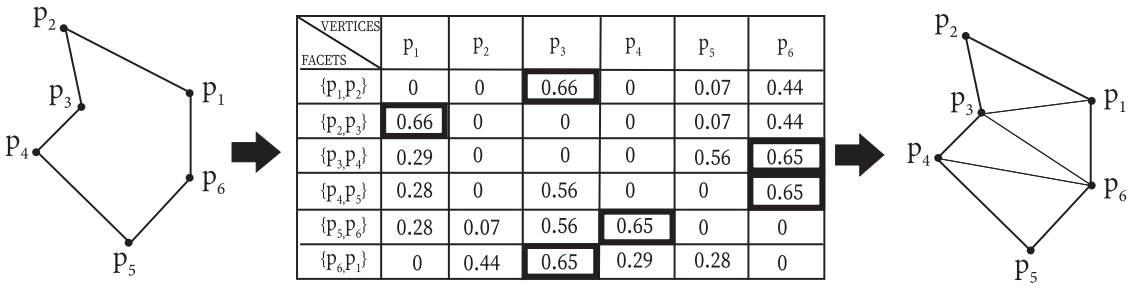
Given a facet F_i (edge in 2-D) of a contour, the probability $P(F_i, v_j)$ of connecting this facet to a vertex v_j must be determined. For each entry $P(F_i, v_j)$, starting with the contour, the facet F_i is connected to a vertex v_j and then the remaining region is meshed using CDT. The element quality of the resulted mesh is measured using the following metric:

$$q_{el} = \frac{4A_{el}}{\sqrt{3}l_{rms}^2}$$

where $0 < q_{el} \leq 1$, A_{el} is the area of the triangular element and $l_{rms} = \sqrt{\frac{1}{3}\sum_{i=1}^{3} l_i^2}$, where l_i, $i = 1, 2, 3$ are the edge lengths of the triangular element. The lowest and mean element quality, q_{worst} and q_{mean}, respectively, are calculated among the mesh elements, and stored as the probability $P(F_i, v_j) = (q_{worst}, q_{mean})$. q_{mean} is used to differentiate between two vertices of the same lowest quality q_{worst}.

The connection table A contains the entries $a_{i,j} = P(F_i, v_j)$, where $a_{i,j}$ is the probability that the F_i (ith row) facet of the contour connects with the v_j (jth column) vertex; v_j being either a contour vertex or an inner vertex. The connection table is then ordered by increasing quality; first the facets are ordered (rows) and then the vertices (columns). Given an ordered connectivity table, a region is meshed by the following procedure (Fig. 10): the entry with the highest probability is first chosen. If two entries having the highest probability had the same value of q_{worst}, the one with the highest q_{mean} is favored. This entry indicates which facet is to be connected to which vertex. The row containing the entries for the former facet is then eliminated.

After a facet has been connected, a element is formed that contains vertices and facets which may not be available for further connection. In such a case, the vertices and facets are considered to be locked and are included in a set of locked vertices V_{locked} and a set of locked facets F_{locked} (Fig. 11) (Alg. 3, Lines 23–26). The connection of facets in F_{locked} is bypassed by removing the row from the connection table (Alg. 3, Line 25). The connections



VERTICES / FACETS	p_1	p_2	p_3	p_4	p_5	p_6
$\{p_1,p_2\}$	0	0	**0.66**	0	0.07	0.44
$\{p_2,p_3\}$	**0.66**	0	0	0	0.07	0.44
$\{p_3,p_4\}$	0.29	0	0	0	0.56	**0.65**
$\{p_4,p_5\}$	0.28	0	0.56	0	0	**0.65**
$\{p_5,p_6\}$	0.28	0.07	0.56	**0.65**	0	0
$\{p_6,p_1\}$	0	0.44	**0.65**	0.29	0.28	0

Fig. 10. The cavity is meshed according to the entries of the connection table. Here, the connection table contains the values of q_{worst}. Each facet (row) is connected with the vertex (column) that has the maximum value of q_{worst}. First, each row is ordered by increasing quality. Subsequently, the columns are ordered with the same criteria. Once the connection table is sorted, the meshing algorithm is called. All invalid connections are set to zero. In the depicted example, the triangulation algorithm starts by connecting the facet $\{p_1, p_2\}$ with the vertex p_3 to create the element $\{p_1, p_2, p_3\}$. This connection is done with accordance to the higher value of the row of the connection table (i.e. 0.66). The algorithm proceeds to the next facet with the vertex that contains the highest entry. The facet $\{p_2, p_3\}$ has a highest entry for the vertex p_1. Since the element $\{p_1, p_2, p_3\}$ is already created, the algorithm will proceed to connect the next facet. In a similar fashion, all the facets of the contour are connected with the vertex that contains the highest entry to form the elements of the mesh.

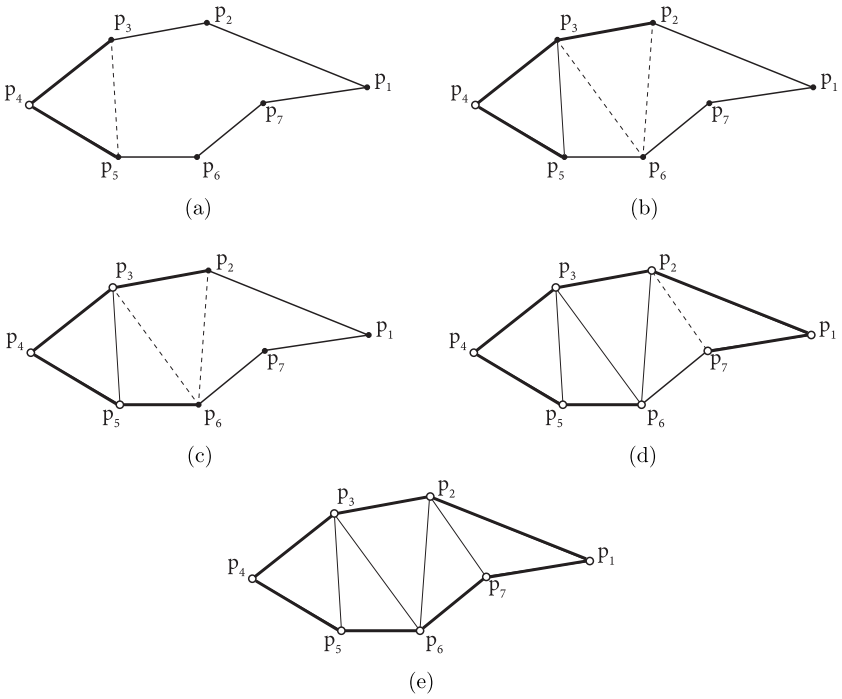


Fig. 11. (a)–(e) Example of meshing a 2-D cavity while the sets V_{locked} and F_{locked} are updated. (a) First, the facet (edge) $\{p_3, p_4\}$ is connected to vertex p_5. The creation of the element $\{p_3, p_4, p_5\}$ locks vertex p_4, as it can no longer be connected with another facet. The facet $\{p_4, p_5\}$ is also locked as it can no longer connect with another vertex. V_{locked} now contains vertex p_4 and F_{locked} contains the facet $\{p_4, p_5\}$. (b) Facet $\{p_2, p_3\}$ connects with vertex p_6 creating the element $\{p_2, p_3, p_6\}$. (c) The creation of element $\{p_2, p_3, p_6\}$ causes the apparition of element $\{p_3, p_5, p_6\}$. V_{locked} will be updated with vertices p_3 and p_5 and F_{locked} with facet $\{5, 6\}$. (d) By connecting the facet $\{p_1, p_2\}$ with vertex p_7 the element $\{p_1, p_2, p_7\}$ locks vertex p_1 and facet $\{p_1, p_7\}$. (e) The creation of element $\{p_1, p_2, p_7\}$ causes the apparition of element $\{p_2, p_6, p_7\}$ that locks vertices p_2 and p_6.

of facets with vertices included in V_{locked} is also bypassed (Alg. 3, Lines 13–15). Connections that crossed existing mesh elements are naturally avoided by assigning a group id to separate sub-cavities (Alg. 3, Lines 18–21) and enforcing connections among facets and vertices having the same group id (Alg. 3, Lines 10–12). This procedure is repeated for the next highest entries.

At the end of the procedure vertices that remain open for connection may cause the appearance of sub-cavities. If the sub-cavity contains only a single element (Fig. 12a), this element is merely added to the mesh. Otherwise, the sub-cavity forms a sub-contour that may also contain inner vertices (Fig. 12b). In such a case, the algorithm is called recursively (Alg. 3, Lines 28–33).

The connection tables A of the training contour datasets are calculated and then used to train NN_3.

NN_3 outputs the entries of the connection table A of dimension $d_A = N_C \times (N_C + N_I)$, where N_C is the number of edges of the contour and N_I is the number of inner vertices. NN_3 takes as input the coordinates of the contour P_C and the coordinates of the inner vertices P_I. It first applies convolution and pooling to the coordinates of the contour. The flattened results of pooling along with the coordinates of the inner vertices are then connected with multilayer perceptrons (Fig. 13). The network is trained to minimize the loss function $L(a_{i,j}, \hat{a}_{i,j}) = \sum_{i,j}(a_{i,j} - \hat{a}_{i,j})^2 / N_{d_A}$, where $a_{i,j}$ is the real value of the entry to the connection table A, $\hat{a}_{i,j}$ is the entry that NN_3 predicts, and $N_{d_A} = N_C \cdot (N_C + N_I)$ (Alg. 4).

2.5.2. Inner vertices grid sampling based data augmentation

NN_3 takes as input the contour vertices coordinates P_C and the N_I inner vertices coordinates P_I. For each contour, the N_I inner vertices that are generated by CDT are used along with sets of N_I vertices that are sampled randomly inside the contour. This process of data augmentation is mandatory for the

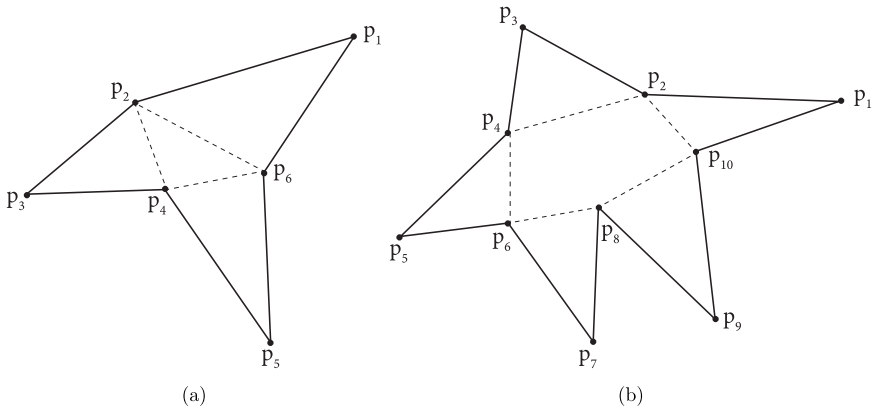


Fig. 12. (a) Example of the appearance of a sub-cavity with a single element. After a first iteration of the triangulation algorithm the facets of the contour are connected with the vertex that corresponds to the highest entry of the connection table. This also causes the appearance of the sub-cavity containing the element $\{p_2, p_4, p_6\}$. The element is added to the list of elements to terminate the triangulation process. (b) Example of sub-cavity that forms a contour. After a first iteration of the triangulation algorithm, once all facets of the contour are connected to the vertex that corresponds to the highest entry, the contour $\{p_2, p_4, p_6, p_8, p_{10}\}$ is formed. In this case, the triangulation algorithm is called recursively to mesh the new contour.

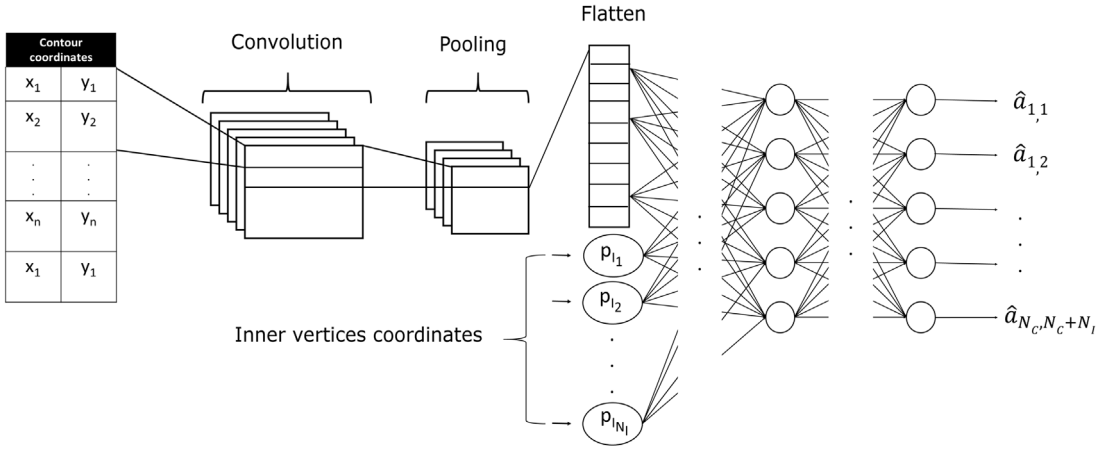


Fig. 13. NN_3 starts by applying 2-D convolution to the coordinates P_C that are ordered in a circular way. It proceeds by applying a pool function to the convoluted result. The flattened outcome of pooling along with the coordinates of the inner vertices P_I are then connected with multilayer perceptrons. It outputs the entries $a_{i,j}$ of the connection table.

meshing scheme as the vertices provided as input to NN_3 are an approximation of the inner vertices inserted by CDT. Thus, to increase the efficacy of learning, NN_3 must be trained for multiple N_I inner vertices and not only the ones inserted by CDT. The output of NN_3 depends on the order in which the N_I vertices are inserted. The structure of the connection table is column-wise dependent on the order of the input inner vertices of NN_3. As a result, small perturbations of the inner vertices may have a great impact on the mesh connectivity. After considering various ordering methods (angular, coordinates), it is found that none of them are invariant to perturbations of the inner vertices coordinates. Therefore, a sampling process is chosen that selects N_I vertices randomly from the interior of a contour cavity with a target edge length rule (Fig. 14). The sampled inner vertices along with the coordinates of the contour are included in the training set of NN_3 without applying any ordering rule to them. This choice improves consistency which facilitates the learning process of NN_3, as the approximation of the entries of the connection table is now based on the actual coordinates of the inner vertices, and do not rely on a specific ordering rule.

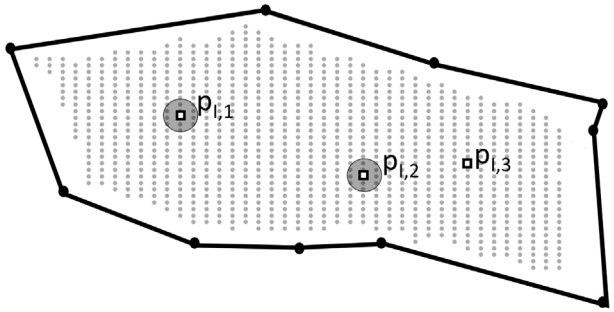


Fig. 14. Example of sampling inner vertices for the training of NN_3 for $N_I = 3$. From a grid of inner vertices, $p_{I,1}$ is randomly chosen and all the vertices contained at a distance of $0.1l_s$ from it are excluded from being selected later on. $p_{I,2}$ is then selected, imposing the same exclusion zone, and finally $p_{I,3}$. All the vertices from the grid are at a distance of $0.1l_s$ from the edges of the contour.

2.5.3. Adaptive sampling strategy

Although the process of random sampling facilitates the learning of NN_3, it causes the accumulation of large training populations. In an attempt to reduce the training populations and maintain or improve the accuracy of the connectivity predictions, an alternative method of sampling is also examined. A NN, NN_3^*, is trained that takes as input the coordinates P_C of the contour, the coordinates of an inner vertices $p_{I,j}$, where $j = 1, 2...N_I$, and an edge index i, where $i = \{1, 2, 3...N_C\}$. NN_3^* outputs the entries of the connection table of the ith row, i.e. the quality values of

Algorithm 3: Triangulation algorithm used for connecting facets of the contour with inner vertices. The algorithm is coded in *Python*.

```
A_ordered: ordered connection table row-wise and then column-wise
F: set of contour edges
V: set of contour vertices and inner vertices
V_open : set of vertices open for connections
V_locked: set of locked vertices
Mesh (F,V,A_ordered):
    foreach facet F_i (row) in A_ordered do
        do
            Connect F_i with v_j (column) to form element e_i,j
            if every vertex v in e_i,j doesn't have the same group id then
                proceed to connect F_i with next v_j
            end
            if there is a vertex v in e_i,j that belongs to V_locked then
                proceed to connect with next v_j
            end
            validate e_i,j
        while e_i,j is not validated;
        if e_i,j divides cavity to sub-cavities then
            foreach subcavity do
                assign same group id to vertices of sub-cavity
            end
        end
        if a vertex v of e_i,j is locked then
            insert v in V_locked
            remove facet F_k (k_th row) from A_ordered
        end
    end
    V_open=V \ V_locked
    if V_open is not empty then
        CheckForSubContours(V_open)
        foreach Sub-contour with edges F' and set of vertices V' do
            Mesh(F', V', A'_ordered)
        end
    end
```

Algorithm 4: Training algorithm of NN_3 for the prediction of connectivity.

```
P_C: Contour vertices coordinates
P_I: Inner vertices coordinates
A: Connection table
Â: Estimated connection table
N_train: Number of training data population
N_dA: Dimension of flattened connection table
Initialize weights W_K of NN_3
while required number of iterations is not reached do
    foreach training example in
    D = {(P_C^(n), P_I^(n), A^(n))}_i, i = 1, .., N_train do
        Compute Â^(n) using current parameters
        Calculate loss function L(A^(n), Â^(n)) =
        ||A^(n) − Â^(n)||_2^2 / N_dA = Σ_i,j (a_i,j^(n) − â_i,j^(n))^2 / N_dA and ∂L/∂W_K
        Update W_K using Adam learning rate optimization
    end
end
```

the mesh that correspond to connections with the ith edge of the contour. Therefore, the predictions of the full connection table are obtained by calling NN_3^* for $i = 1, 2, \ldots, N_C$. On a grid G, for every vertex $g_{i,j} \in G$ that is located inside a contour, the worst quality of the mesh is calculated, given that the vertex is connected with an edge (Fig. 15a). In the process, a quality surface S_i is defined, where $i = \{1, 2, \ldots, N_C\}$, for every edge of a contour (Fig. 15b). Each quality surface S_i is smoothed to avoid the appearance of sudden peaks while maintaining the maximum values (Fig. 15c). Vertices are sampled from S_i according to the curvature loss criteria, i.e. more vertices are sampled from regions where the rate of change of curvature is high (Fig. 15d). The process of sampling of vertices from the smoothed quality surface using the curvature loss criteria is more efficient than that from the unsmoothed surface. This is because peaks with a sudden change in curvature lead to a sampling set of vertices with higher reconstruction loss as compared to that from the smoothed surface. Thus, the smoothing process assists in the training of NN_3^* by providing a set of sampled vertices that provides more representative information on the quality surface within a contour.

Similarly, for multiple inner vertices, given the position of $N_I - 1$ inner vertices, a quality surface is defined by calculating the minimum quality of each grid vertex from G. Each surface $S_{n,i}$, where $n = 1, 2..., N_I$ and $i = 1, 2, 3...N_C$, is smoothed and vertices are sampled by using the curvature loss criteria to form sets of inner vertices $V_{n,i}$ (Fig. 16). From each set $V_{n,i}$, an inner vertex is sampled. An additional criterion is imposed so that the sampled vertices must be at a distance of at least $0.3l_s$ from one another. The sampled vertices are used to train NN_3^*.

3. Experiments and results

3.1. Experimental conditions

3.1.1. Error metrics

Experiments are conducted in testing populations of random contours. The predictive approximation on the number of vertices using NN_1 is evaluated by calculating the mean absolute error between the predictive value and the real value. For a contour, the predictive approximation of the location of the inner points (NN_2) and the connectivity (NN_3, NN_3^*) are evaluated in terms of quality of the generated mesh using the output values of the respective NN. Two different mesh qualities are calculated, the worst quality $\hat{q}_{worst} = \min_{i\in\{1,\ldots,N_{el}\}} q_{el}^{(i)}$ and the mean quality $\hat{q}_{mean} = \sum_i^{N_{el}} q_{el}^{(i)}/N_{el}$, where N_{el} is the number of total elements and q_{el} is the quality metric as defined in Section 2.5.1. The contour is then meshed using the parameters acquired by applying the reference algorithm of CDT to calculate the values q_{worst} and q_{mean}. The relative differences $e_{worst} = (q_{worst} - \hat{q}_{worst})/q_{worst}$ and $e_{mean} = (q_{mean} - \hat{q}_{mean})/q_{mean}$ define the worst and mean triangulation error, respectively, for a contour. e_{worst} and e_{mean} are used as error metrics when approximating with NN_2, NN_3 and NN_3^*. For a population of contours with N_C edges and N_I inner vertices the expected values $E(e_{worst})$ and $E(e_{mean})$ are estimated by calculating the averages $\bar{e}_{worst} = \sum^{n_{N_{C,I}}} e_{worst}/n_{N_{C,I}}$ and $\bar{e}_{mean} = \sum^{n_{N_{C,I}}} e_{mean}/n_{N_{C,I}}$, where $n_{N_{C,I}}$ is the number of meshed contours participating in the test population.

3.1.2. Training dataset populations

Contour training datasets are populations of contours for $N_c = \{4, 6, 8, 10, 12, 14, 16\}$ edges. The population of contours that are generated, are increased at a nearly exponential rate with the number of edges (Fig. 17); it is observed that this increase is necessary to retain a level of accuracy from the NNs involved in the meshing scheme so that it provides good quality meshes.

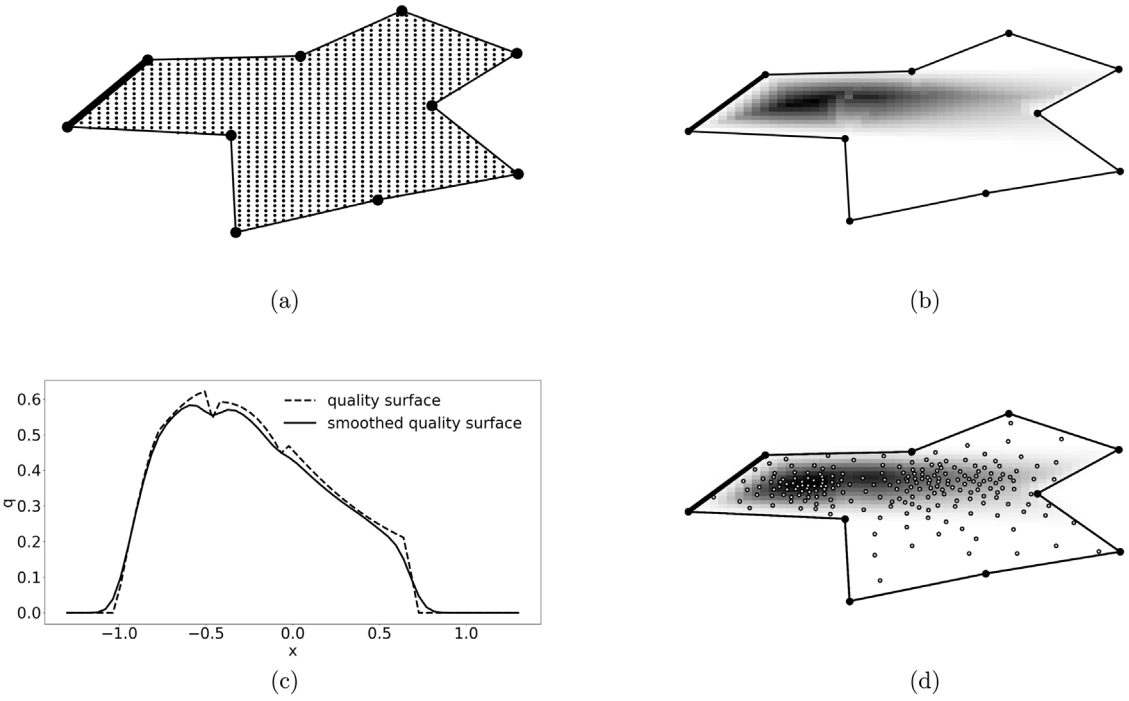


Fig. 15. (a) For the highlighted edge, the worst mesh quality is calculated by connecting this edge with each point of the grid. (b) By doing so, a quality surface is defined for this edge. (c) Curves depicting the quality values q along the original quality surface and the smoothed quality surface for a fixed y-value. By smoothing, sudden peaks are eradicated. (d) Vertices are sampled from the smoothed surface according to curvature criteria; the higher the curvature the more vertices are sampled.

The populations of contours used for the training of NN_1 (Fig. 17) are divided into groups with respect to the number of inner vertices inserted by CDT for target edge lengths ranging from 0.2 to 1 (Fig. 18). Each of these groups is then used to train NN_2 and NN_3.

The training data were generated in a machine with 64 GB memory and 2 CPUs Intel© Xeon© E5-2660v2 running at 2.2 GHz and 10 cores and a machine with 128 GB memory with the same 2 CPUs.

3.1.3. NN_i hyperparameters

NN_1 is trained for 3000 epochs with a learning rate of $lr = 10^{-4}$ and a weight decay of $w = 10^{-1}$ with a batch size of $n_{batch} = 512$. There are 3 layers with batch normalization and the ReLU activation function for the first 2 layers with $4 \cdot N_C$ hidden nodes.

For the training of NN_2, the grid G used for the approximation of the inner vertices, is divided into N_p patches G_k, $k = \{1, 2..N_p\}$. NN_2 is trained for 5000 epochs with a learning rate of $lr = 10^{-4}$ and a weight decay of $w = 10^{-2}$ with a batch size of $n_{batch} = 512$. There are 3 layers with batch normalization and the ReLU activation function for the first 2 layers. The first two layers contain $2 \cdot N_C + N_{G_k}$ hidden nodes and the output layer contains N_{G_k} nodes.

The populations of contours used for training NN_3 are the same as the ones used for the training of NN_2 (Section 3.1.2) (Fig. 18). NN_3 is trained for 5000 epochs with a learning rate of $lr = 10^{-4}$ and a weight decay of $w = 10^{-1}$ with a batch size of $n_{batch} = 512$. The convoluted layer contains N_C number of filters. The convolution is done using a stride of 2 and filter size 2×2. Max pooling is then applied with stride 2 with 2×1 filters to the convoluted result. There are 3 fully connected layers with batch normalization and the tanh activation function for the first 2 layers. The number of hidden nodes for the first two layers is $2 \cdot N_C \cdot (N_C + N_I)$. The output layer contains $N_C \cdot (N_C + N_I)$ nodes which is the flattened dimension of the connection table.

3.2. Results

3.2.1. Predictions for the number of inner vertices

The prediction of the number of inner vertices from NN_1 is evaluated on groups of random contours with N_C edges. The confidence level and confidence interval of each test population is 95% and 5%, respectively, of each training population as presented in Section 3.1.2 (Fig. 17). The random contours are meshed by applying CDT for target edge lengths ranging from 0.2 to 1. The mean absolute error $\bar{e}$ is examined. The mean absolute error is defined as $\bar{e} = \overline{|N_I - \hat{N}_I|} = \sum_{i=1}^{n_{N_C}} (N_I^{(i)} - \hat{N}_I^{(i)})/n_{N_C}$, where $\hat{N}_I = (\hat{N}_I^{(1)}, \ldots, \hat{N}_I^{(n_{N_C})})$ are the approximations of NN_1 and n_{N_C} is the number of contours with N_C edges in the test population. By examining $\bar{e}$ it is observed that the error increases with the number of edges (Fig. 19). For example, for a target edge length of $l_s = 0.2$, the mean absolute error $\bar{e}$ for the contour with 16 edges is approximately 2.3 times higher than that for the contour with 4 edges. This is attributed to the fact that there is a much larger variation in the number of vertices N_I being inserted by CDT for the contour population of 16 edges than the contour population with 4 edges. The standard deviation of N_I points being inserted is $std_{16} = 5.2$ and $std_4 = 1.4$ for the contour populations with 16 and 4 edges, respectively.

Moreover, in every population of contours with N_C edges, the error is higher for smaller target edge lengths l_s. As explained before, this increase in the error is due to the larger variation of N_I points being inserted by CDT inside a contour population with N_C edge length, as the target edge length gets smaller; e.g. for the contour population of 14 edges the mean absolute error $\bar{e}$ for $l_s = 0.2$ and $l_s = 1$ decreases from 1.3 to 0.2 and the standard

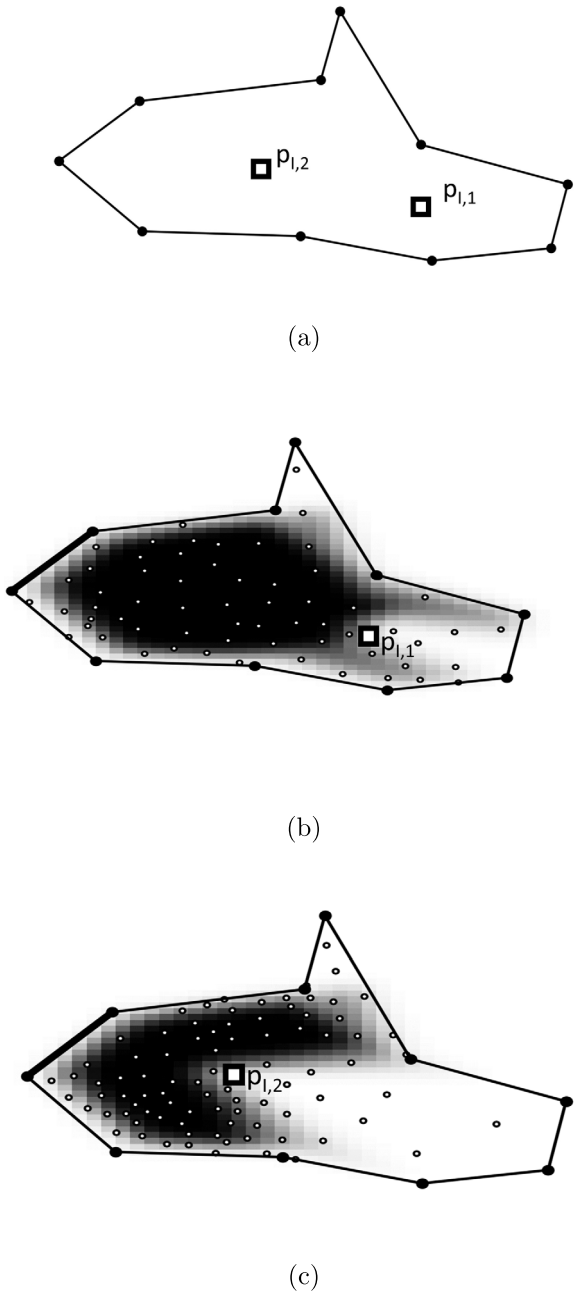


Fig. 16. (a) Example of adaptive sampling for i_{th} edge (highlighted) of a contour with 10 edges and two inner vertices. (b) Surface $S_{1,i}$ is defined by computing the minimum quality of each grid vertex taking into account $p_{l,1}$ as a second point. By smoothing $S_{1,i}$ and implementing the curvature loss criteria, vertices are sampled from the surface to form the set of inner vertices $V_{1,i}$. (c) In a similar fashion the surface $S_{2,i}$ is defined and vertices from it are sampled to form the set $V_{2,i}$. To train the NN, a pair of vertices with one vertex belonging to $V_{1,i}$ and another belonging to $V_{2,i}$ is sampled. The collected vertices must be at a distance of at least $0.3l_s$ from each other.

deviation of N_I for these target edge lengths decreases by 12%. Despite the increase in the mean absolute error $\bar{e}$ with increase in the number of contour edges and decrease in the target edge length, the predictions of NN_1 are considered accurate enough to be used for meshing purposes (Fig. 20); e.g. for a random contour of 16 edges the number of points inserted by CDT and the number of points predicted by NN_1 differ by one for target edge lengths $l_s = \{0.2, 0.3, 0.4\}$ and is the same as the target edge length increased (Fig. 20b).

3.2.2. Prediction of the inner vertices locations

The grid used for the approximation of the inner vertices P_I has a bounding box $[-1.2, 1.2] \times [-1.2, 1.2]$ which includes all the vertices P_C of the contours. The resolution of the grid affects the accuracy of the predictions. To find an appropriate resolution, the accuracy of NN_2 for the prediction of one inner vertex $p_{I,1}$ on $n = 100$ random contours is examined. The mean squared error of the euclidean distance $\bar{e}_{dist}$ is examined for grid resolutions of 10×10, 20×20, and 40×40. The mean squared error is defined as $\bar{e}_{dist} = \overline{\|p_{I,1} - \hat{p}_{I,1}\|} = \sum_i^n (p_{I,1}^{(i)} - \hat{p}_{I,1}^{(i)})^2/n$, where $\hat{p}_{I,1} = (\hat{p}_{I,1}^{(1)}, \ldots, \hat{p}_{I,1}^{(n)})$ are the approximations of the inner vertex for n contours with N_C edges. $\bar{e}_{dist}$ is calculated for the aforementioned grid resolutions . By examining the convergence rate of $\bar{e}_{dist}$, it is concluded that for the case studies a cell size of 20×20 is adequate to get accurate results (Fig. 21). This resolution also ensures that the distance between successive grid points is 33% smaller than the smallest target edge length $l_s = 0.2$. Once a grid point with minimum score is selected, all the grid points contained at a distance of $0.2l_s$ participate in the interpolation process. The scores of all the grid points that are contained within a distance of $0.1l_s$ from the edges of the contour are penalized. Finally, all the grid points contained closer than a distance of $0.1l_s$ from a chosen grid point with minimum score are excluded as possible candidates for approximation of the next inner vertex. The grid is divided into 100 patches. Each of the patches contains 4 grid points.

To measure the triangulation error caused by the approximations of NN_2 the averages errors $\bar{e}_{worst}$ and $\bar{e}_{mean}$ are measured between the meshes generated using the approximated inner vertices $\hat{P}_{I,i}$, where $i = \{1, 2..., N_I\}$, and the meshes generated using the inner vertices $P_{I,i}$ that are inserted by CDT (Fig. 22). All the tests are conducted on random contours with a sample size of 95% confidence level and 5% confidence interval of each training group presented in Section 3.1.2 (Fig. 18).

The triangulation error in the prediction of inner vertices is due to two factors: (1) The resulting mesh with the predicted inner vertices may have the same connectivities as the mesh with the inner vertices inserted by CDT. In this case, the approximated inner vertices form triangular elements that are of worse quality due to their displacement (Fig. 23). (2) Applying CDT with the approximated inner vertices may result in a mesh with different connectivities altogether, where worse quality elements appear. The average error of $\bar{e}_{worst}$ increases with the number of inner vertices N_I, reaching a maximum of 23.41% for the case of contours with 12 edges and 14 inner vertices (Fig. 22e). This increase demonstrates how the complexity of a mesh affects the approximations of NN_2; the displacement error accumulated by the prediction of the inner vertices increases with the number of inner vertices, which in turn causes the increase in $\bar{e}_{worst}$. This effect is mostly noticeable in the case of contours with 6 edges with one inner vertex and 2 inner vertices (Fig. 22b). Merely by the addition of one inner vertex, $\bar{e}_{worst}$ is increased from 1.6% to 10.5%. A few mild fluctuations in $\bar{e}_{worst}$ within the populations of N_C edges are explained by the choice of random contours that belong to the test dataset (e.g. for the case of contours with 4 edges with 3 and 4 inner vertices) (Fig. 22a) or because there is an increase in the training population as the number of inner vertices is increased (e.g. for the case of contours with 14 edges containing 8 and 10 inner vertices) (Fig. 22f). In addition, it is observed that $\bar{e}_{mean}$, which has a maximum of 19.7% in the case of a contour with 12 edges and 14 inner vertices (Fig. 22e), does not necessarily follow the pattern of $\bar{e}_{worst}$ (Fig. 22d). It is expected for $\bar{e}_{mean}$ to be lower than $\bar{e}_{worst}$, as the mean quality q_{mean} takes into account the qualities of all the elements in the mesh and not just that of the worst element.

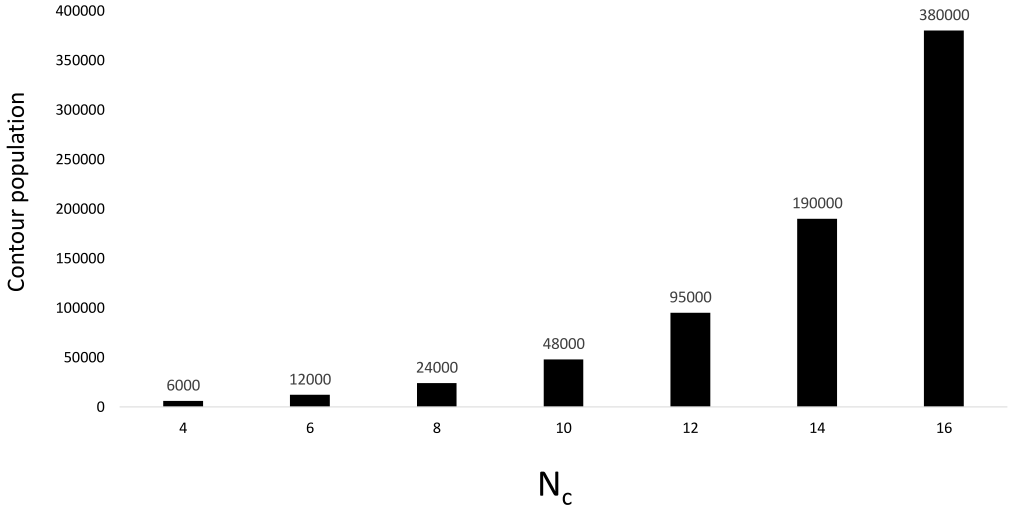


Fig. 17. Histogram of contour populations with N_C edges that are used for training. The population of 6000 contours with 4 edges is found to be an adequate training set for acquiring satisfactory accuracy from the *NNs* involved in the meshing procedure. To retain or acquire a level accuracy needed for good quality meshing, for contours with 6 and 8 edges a training population of approximately 12 000 and 24 000 contours, respectively, is required; this leads to the choice of generating contour populations used for training that increase exponentially with the number of edges.

3.2.3. Prediction of the connectivity

To estimate the triangulation error caused by the triangulation algorithm, the average errors, $\overline{e}_{worst}$ and $\overline{e}_{mean}$, are measured between the meshes generated with the calculated connection table A and those generated by CDT on the same random contours as those used in Section 3.2.2. A maximum triangulation error $\overline{e}_{worst} = 1.9\%$ is reached in the case of contours with 14 edges and 10 inner vertices (Fig. 24f) and a maximum of $\overline{e}_{mean} = 1.5\%$ for contours with 16 edges and 16 inner vertices (Fig. 24g). In some cases, there is no triangulation error (Fig. 24a, b). The fluctuation in the errors may be caused by the random choice of contours participating in the test dataset. The level of triangulation error in $\overline{e}_{worst}$ and $\overline{e}_{mean}$ demonstrates that the algorithm does not account for a significant error propagation in the connection scheme.

The number of sampled vertices chosen for a contour increases according to the number of vertices N_I that are intended to be inside the cavity. 50 N_I groups of sample vertices are chosen for $2 \leq N_I \leq 4$, 100 for $5 \leq N_I \leq 8$, and 200 for $N_I \geq 9$.

The triangulation error caused by the approximations of NN_3 is tested for the random contours used in Section 3.2.2 which contained the N_I inner vertices that are inserted by CDT. The average triangulation errors, $\overline{e}_{worst}$ and $\overline{e}_{mean}$, are calculated between the meshes that are generated with entries of the connection table that are predicted using NN_3 and those generated by using CDT. A maximum triangulation error $\overline{e}_{worst} = 24.04\%$ is reached for the contour with 8 edges and 12 inner vertices. It is observed that there are more fluctuations as compared to the triangulation errors caused by the prediction of NN_2. The following observations are made upon examining the average triangulation error $\overline{e}_{worst}$:

- For contours with N_C edges and N_I inner vertices, the accuracy of the connectivity depends on the number of sampled vertices N_I used to train NN_3. This is quite evident in the case of contours with 6 edges for $N_I = \{4, 6\}$ (Fig. 25b). Although the training contour population for $N_I = 6$ inner vertices is 37% lower than that for $N_I = 4$, there is a 23% reduction in $\overline{e}_{worst}$ as the number of sampled inner vertices for $N_I = 6$ is twice as big as for $N_I = 4$. The same is observed in the case of contours with 10 edges for $N_I = \{8, 10\}$ (Fig. 25c). Even though the complexity of the triangulations is increased (i.e. the number of inner vertices) and the training contour population for $N_I = 10$ is 38% lower than that for $N_I = 8$, $\overline{e}_{worst}$ decreases by 29%.
- For a fixed number of N_I sampled inner vertices, $\overline{e}_{worst}$ increases with increase in the number of vertices or decrease in the training data. For example, for the contour with 12 edges with $N_I = \{10, 12\}$ (Fig. 25e), $\overline{e}_{worst}$ increases from 3.6% to 13% whereas the population of the training contours decreases by 41%. However, for the same contour population for $N_I = \{6, 8\}$, even though the complexity of triangulation is increased, $\overline{e}_{worst}$ decreases from 10.7% to 8% whereas the

Table 1
Sampling sizes for the mean populations of the contours inner vertices with random and adaptive sampling for different confidence levels.

N_I	Sampling method	Mean population of inner vertices	Confidence level	Sample size
1	Random sampling	689	50	206
			68	332
			90	495
			96	594
	Adaptive sampling	450	50	178
			68	265
			90	359
			96	382
2	Random sampling	1378	50	242
			68	437
			90	772
			96	889
	Adaptive sampling	2×450	50	252
			68	374
			90	595
			96	662
4	Random sampling	2756	50	265
			68	519
			90	1071
			96	1312
	Adaptive sampling	4×450	50	252
			68	472
			90	888
			96	1047

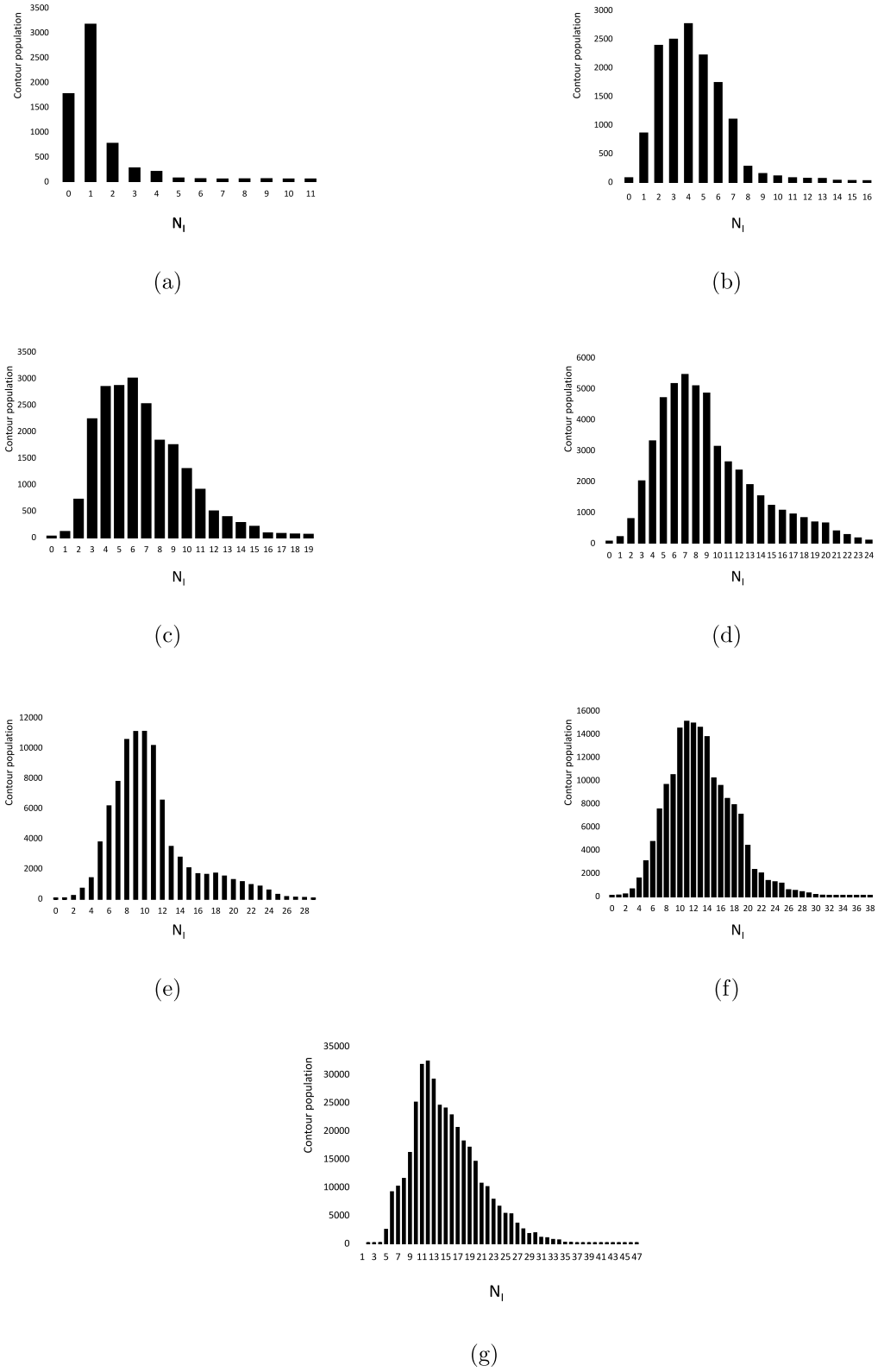


Fig. 18. (a)–(g) In alphabetical order, the histogram of the contour populations for $N_C = (4, 6, 8, 10, 12, 14, 16)$ number of edges (Fig. 17) divided into groups according to the number of vertices N_I that are inserted by CDT. Each of these groups is used to train NN_2 and NN_3.

training contour population is increased by 42%. The accuracy of NN_3 appears to be more dependent than NN_2 on the size of the training contour populations (Fig. 18).

3.2.4. Efficiency of adaptive sampling

To examine the efficacy of the adaptive sampling, the predictions of NN_3^* are studied for the case of contours with 10

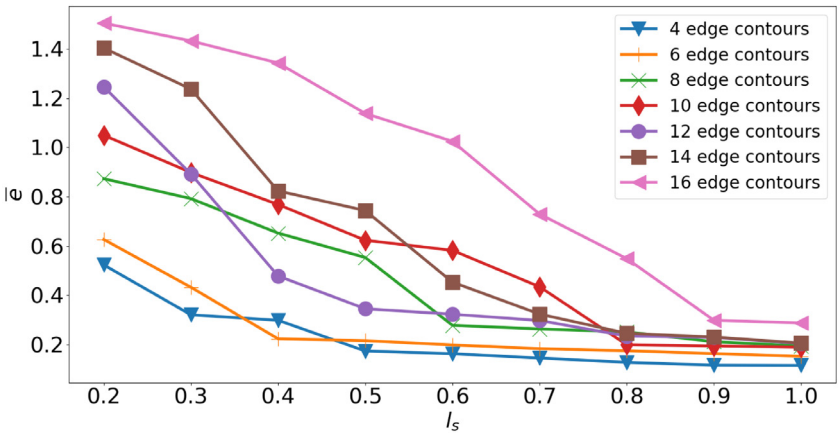


Fig. 19. The mean absolute error $\bar{e}$ as a function of target edge lengths l_s ranging from 0.2 to 1. The mean absolute error is defined as $\bar{e} = \overline{|N_I - \hat{N}_I|} = \sum_{i=1}^{n}(N_I^{(i)} - \hat{N}_I^{(i)})/n$, where $\hat{N}_I = (\hat{N}_I^{(1)}, \ldots, \hat{N}_I^{(n_{N_C})})$ are the number of vertices predicted by the NN, $N_I = (N_I^{(1)}, \ldots, N_I^{(n_{N_C})})$ are the number of vertices calculated by CDT and n_{N_C} the number of contours with N_C edges in the test population. The mean absolute error $\bar{e}$ increases with increase in the number of contour edges N_C. For example, for a target edge length $l_s = 0.2$, $\bar{e}$ for the population of contours with 16 edges is approximately 2.3 times higher than the mean error for the population of contours with 4 edges due to the larger variation of number of inner vertices N_I that CDT inserts for the populations of 16 edges. $\bar{e}$ also increases with the decrease of the target edge length l_s which is also due to the fact that the variation on the number of inner vertices N_I inserted by CDT is larger for smaller target edge lengths. For instance, for the population of contours with 14 edges and the target edge lengths $l_s = 0.2$ and $l_s = 1$, $\bar{e}$ decreases from 1.3 for 0.2 respectively, while the standard deviation of inner vertices N_I for these target edge lengths decreases by 12%.

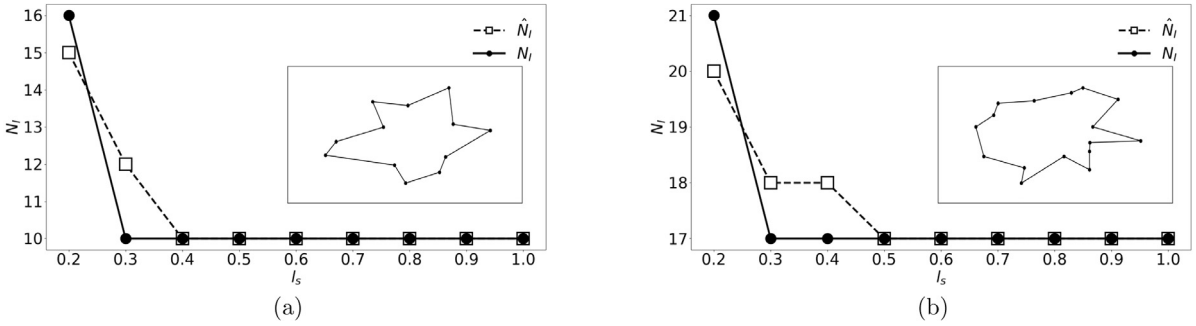


Fig. 20. The real number of inner vertices N_I and the predicted number of inner vertices $\hat{N}_I$ as a function of the target edge length for a random contour with 12 edges (a) and a random contour of with 16 edges (b). For the contour with 12 edges N_I and $\hat{N}_I$ differ by one point for $l_s = 0.2$, two points for $l_s = 0.3$ and are the same for the rest of the target edge lengths. For the contour with 16 edges, N_I and $\hat{N}_I$ differ by one point for the target edge lengths $l_s = \{0.2, 0.3, 0.4\}$ and are same for the rest of the values l_s. It can be concluded that NN_1 is appropriate to use for meshing purposes.

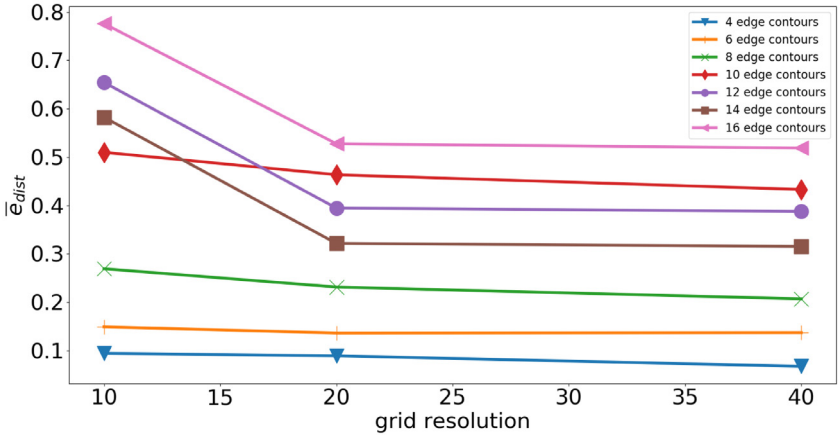


Fig. 21. The mean squared error of euclidean distance $\bar{e}_{dist}$ as a function of grid resolutions of 10 × 10, 20 × 20, and 40 × 40. The mean square error is defined as $\bar{e}_{dist} = \overline{\|p_{I,1} - \hat{p}_{I,1}\|} = \sum_i^n (p_{I,1}^{(i)} - \hat{p}_{I,1}^{(i)})^2/n$, where $p_{I,1} = (p_{I,1}^{(1)}, \ldots, p_{I,1}^{(n)})$ are the real locations of the inner vertex inserted by CDT and $\hat{p}_{I,1} = (\hat{p}_{I,1}^{(1)}, \ldots, \hat{p}_{I,1}^{(n)})$ are the predictions of the inner vertex for $n = 100$ contours with N_C edges. For every contour population, the error reaches convergence by using a grid with resolution 20 × 20.

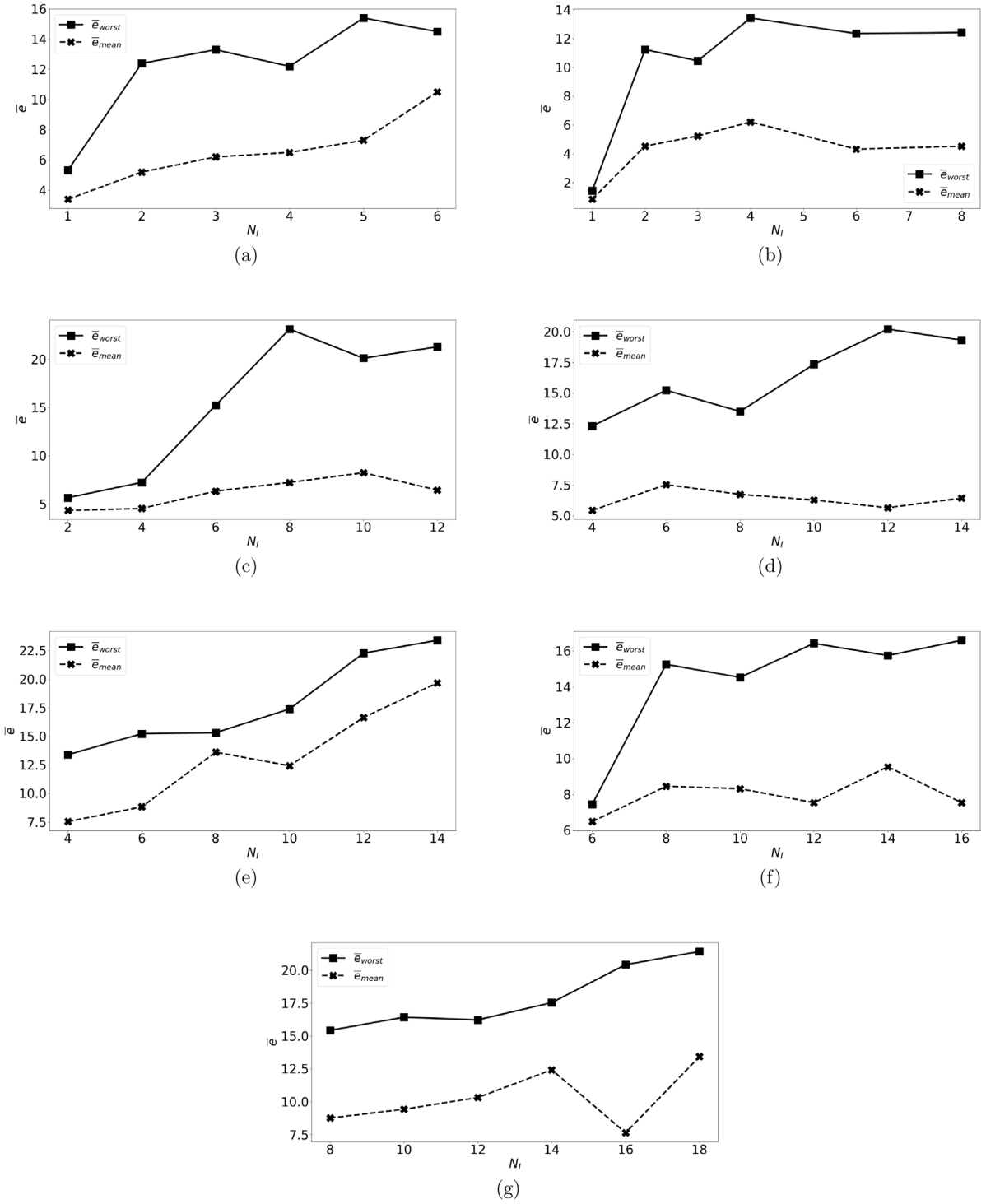


Fig. 22. (a)–(g) In alphabetical order, the average errors $\bar{e}_{worst} = \sum^{n_{N_{C,I}}} e_{worst}/n_{N_{C,I}}$ (%) and $\bar{e}_{mean} = \sum^{n_{N_{C,I}}} e_{mean}/n_{N_{C,I}}$ (%) for a number of $n_{N_{C,I}}$ random contours with $N_C = \{4, 6, 8, 10, 12, 14, 16\}$ edges as a function of inner vertices N_I. The range of inner points covers at least 68% of each contour population with N_C edges. Maximum $\bar{e}_{worst}$ of 23.41% occurs for the case of contours with 12 edges and 14 inner vertices (e). In most cases, both $\bar{e}_{worst}$ and $\bar{e}_{mean}$ tend to increase with the increase in the number of inner vertices; the displacement error from the predicted vertices of NN_2 increases with the number of vertices N_I, which, in turn, increases the triangulation errors e_{worst} and e_{mean}.

edges with one inner vertex for a training population of 2000 contours. On a grid G of resolution 50×50, the worst quality of the mesh is calculated, given that the vertex is connected with an edge (Fig. 15a). First, NN_3^* is trained with inner vertices that are sampled randomly from the grid for different sample sizes of the whole population of inner vertices (Table 1). Next, NN_3^* is trained by providing every ith edge index with vertices that are sampled adaptively from S_i by using the curvature criteria. For each edge, 45 inner vertices are sampled adaptively i.e. the whole population of inner vertices for a contour is 450. NN_3^* is trained for different sample sizes of this population. The accuracy of NN_3^* is tested on random contours that accounted for a 95% confidence level

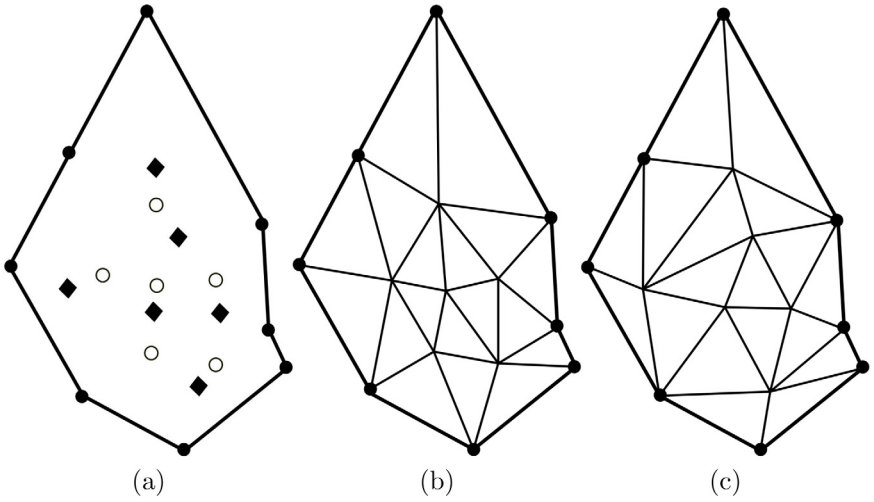


Fig. 23. (a) A contour with 8 edges with inner vertices inserted by CDT (circular points) and inner vertices approximated by the scores of NN_2 (diamond points). (b) The mesh with inner vertices from CDT. (c) The resulting mesh with approximated vertices. In this case, the triangulation errors account for $e_{worst} = 22.3\%$ and $e_{mean} = 9.8\%$.

and 5% for each training population. Using adaptive sampling, the accuracy is improved by 27% while requiring 35% less samples with respect to the use of random sampling. (Fig. 26a).

Next, adaptive sampling is applied for the case of contours with 10 edges with $N_I = \{2, 4\}$ inner vertices and a training population of 2000 to determine the accuracy of NN_3^*. Given the position of $N_I - 1$ inner vertices, a quality surface is defined by calculating the minimum quality of each grid vertex from G. For each edge 45 vertices are sampled from each surface $S_{n,i}$. Thus, for each contour, the whole population is $45 \times N_I \times 10$ inner vertices. The accuracy of NN_3^* is tested on random contours that accounted for 95% confidence level and 5% for each training population. Once again, using adaptive sampling leads to 18% and 27% better accuracy for $N_I = 2$ and $N_I = 4$ respectively, with populations that are 22% and 17% lower than the ones of random sampling (Fig. 26b, c).

3.2.5. Overall prediction of the meshing scheme

The meshing scheme is tested for populations of contours with $N_C = (4, 6, 10, 12, 14, 16)$ edges for target edge lengths varying from 0.2 to 1.0 (Fig. 27) with a sample size of 95% confidence level and 5% confidence interval of each training group (Fig. 18). The number of inner vertices N_I, their coordinates, and entries of the connection table are predicted using NN_1, NN_2, and NN_3, respectively. By using all of the NNs involved in the meshing scheme, an increase in the triangulation error compared to our previous tests for every contour population is observed, e.g. for the case of contours with 10 edges and 8 inner vertices (Fig. 27g), the average worst triangulation error is $\bar{e}_{worst} = 19.4\%$ whereas the average worst triangulation errors caused by NN_2 and NN_3 are 17.3% and 10%, respectively. The increase in the error is not only due to the approximation of the connectivity using NN_3 but also because of the triangulation error caused by using NN_2 for the approximation of inner vertices. A maximum triangulation error $\bar{e}_{worst} = 27.3\%$ is obtained for the contour population of 16 edges with 18 inner vertices (Fig. 27g).

4. Meshing larger meshes

The proposed neural network meshing algorithm is used to mesh larger meshes as follows: Given a high resolution contour S, vertices are sampled to form an initial contour (Fig. 28a)

Table 2
The worst quality q_{worst} and mean quality q_{mean} for the circle (Fig. 28), airfoil (Fig. 29) and contour with circular hole (Fig. 30) examples (the closer a quality value is to 1 the better).

Example	q_{worst}	q_{mean}
Circle	0.88	0.97
Airfoil	0.69	0.90
Contour with hole	0.66	0.85

(Alg. 5, Lines 7–8). The number of vertices sampled to form the initial contour is $4 < N_C < 16$ in accordance with the trained NNs of the meshing scheme. Then, NN_3 is called to assemble the connections and output an initial mesh (Fig. 28b) (Alg. 5, Line 9). Assuming a target edge length l_s, for an edge e with edge length $l_e > l_s$, based on the integer ratio $K = [l_e/l_s]$, $n_K = K - 1$ equidistant vertices are inserted to e (Fig. 28c) (Alg. 5, Lines 13–21). If the vertices of an edge belong to the high resolution contour, the inserted vertices are projected to it (Fig. 28d) (Alg. 5, Lines 22–24). Similarly, to be in accordance with the trained NNs, the maximum number of inserted vertices per edges is $n_K = 4$. The insertion of the vertices in the elements of the initial mesh forms sub contours that are meshed using the proposed meshing scheme for the target edge length that corresponds to the average length of the edges of a sub-contour $l_s^* = \sum_{k=1}^{N} l_{e^{(k)}}/N$ (Fig. 28e) (Alg. 5, Line 29). Subsequently, NN_2 is used for the computation of mesh coordinates to improve the quality (Alg. 5, Line 32). The aforementioned process is repeated until a target edge length close to l_s is reached and no further vertices are inserted (Fig. 28f) (Lines 27, 35).

Adaptive mesh generation is achieved by defining a sizing function $h(\Omega_S)$ (Figs. 29a, 30a) over the inner domain of the high resolution contour Ω_S. The values of the sizing function correspond to a target edge length which may vary over the domain Ω_S; the generated mesh using a sizing function contains smaller scale element to approximate better characteristics of the geometry (e.g. curvature) and larger scale elements elsewhere. In the context of the present meshing scheme, an adaptive strategy is applied by inserting vertices to the edges of an initial mesh that is generate using the connectivity network NN_3 (Figs. 29c, 30c) according to the prescribed values of the sizing function. During the refinement process, equidistant vertices $p = (p_1, \ldots, p_{n_K})$, $n_K \leq$

4, are inserted to an edge e incrementally while $\left|l_{e^{(j)}} - h(p_j)\right| < \epsilon$ for $j = 1, 2, \ldots, n_K$, where $l_{e^{(j)}}$ is the length of the edge $e^{(j)}$ formed by the vertices (p_j, p_{j+1}) (Alg. 6, Lines 12–26); the segment lengths of the subdivided edge should be close to the value of the sizing function of each inserted vertex. The strategy to mesh larger scale meshes with an adaptive or uniform element size inherits the automation of the meshing scheme as it is based exclusively on the use of NNs. Table 2 lists the qualities of examples used to create uniform (Fig. 28) and adaptive (Figs. 29, 30) large meshes.

Algorithm 5: Uniform scale element mesh generation using NN_3

```
1  S: high resolution contour
2  h(Ω_S): sizing function defined over the domain Ω_S of the
   high resolution contour
3  l_s: target edge length
4  l_e: length of edge e
5  (p_o, p_K): endpoint vertices of edge e
6  P: list of inserted vertices
7  Sample N_C vertices from S
8  Connect the N_C vertices with a line to form a contour with
   N_C edges
9  Form an initial mesh using NN_3
10 Refine=True
11 while Refine do
12     foreach edge e = (e^(1), ..., e^(i), .., e^(n)) in current mesh M
       do
13         if |l_e(i) − l_s| < ε then
14             K = [l_e(i)/l_s]
15             if K > 5 then
16                 K = 5
17             end
18             for j = 1,2,..K-1 do
19                 Insert p_j^(i) = p_0^(i) + (j/K)(p_0^(i) − p_K^(i))
20                 P ← p_j^(i)
21             end
22             if endpoint vertices (p_0^(i), p_K^(i)) of edge e^(i) belong to
               the high resolution contour S then
23                 project inserted vertices p = (p_1^(i), .., p_K−1^(i)) to
                   the high resolution contour S
24             end
25         end
26     end
27     if P is not empty then
28         foreach subcontour formed by the vertices of an
           existing element of M with the inserted points P do
29             Mesh subcontour using the meshing scheme for
               a target edge length equal to the average edge
               length of the sub-contour
30         end
31         Update mesh M
32         Use NN_2 to calculate new mesh coordinates to
           improve quality of M
33         empty P
34     else
35         Refine=False
36     end
37 end
```

Algorithm 6: Adaptive scale element mesh generation using NN_3

```
1  S: high resolution contour
2  h(Ω_S): sizing function defined over the domain Ω_S of the
   high resolution contour
3  l_s: target edge length
4  l_e: length of edge e
5  (p_o, p_K): endpoint vertices of edge e
6  P: list of inserted vertices
7  Sample N_C vertices from S
8  Connect the N_C vertices with a line to form a contour with
   N_C edges
9  Form an initial mesh using NN_3
10 Refine=True
11 while Refine do
12     foreach edge e = (e^(1), ..., e^(i), .., e^(n)) in current mesh M
       do
13         if P is not empty then
14             empty P
15         end
16         K = 1
17         do
18             K += 1
19             for j = 1,2,..K-1 do
20                 Insert p_j^(i) = p_0^(i) + (j/K)(p_0^(i) − p_K^(i))
21                 P ← p_j^(i)
22             end
23         while |l_e(j) − h(p_j^(i))| < ε, where j = 1, 2, ..K − 1 and
           e^(j) is the edge with vertices (p_j^(i), p_j+1^(i)), or K! = 5;
24         if endpoint vertices (p_0^(i), p_K^(i)) of edge e^(i) belong to the
           high resolution contour S then
25             project inserted vertices p = (p_1^(i), .., p_K−1^(i)) to the
               high resolution contour S
26         end
27     end
28     if P is not empty then
29         foreach subcontour formed by the vertices of an
           existing element of M with the inserted points P do
30             Mesh subcontour using the meshing scheme for
               a target edge length equal to the average edge
               length of the sub-contour
31         end
32         Update mesh M
33         Use NN_2 to calculate new mesh coordinates to
           improve quality of M
34         empty P
35     else
36         Refine=False
37     end
38 end
```

5. Conclusions

A novel machine learning simplicial mesh generation scheme was presented for meshing 2D simplicial contours with target edge length. The meshing scheme uses three NNs that predicted the number of inner vertices, their location, and the entries of a connection table. Based on the entries of the connection table, a triangulation algorithm is applied to mesh the contour. The proposed meshing scheme generates topologically valid meshes with no element intersections (manifold mesh) and can be trained using any classic mesh generator given that no additional vertices are inserted on the edges of the contour. The meshing scheme avoids the incremental creation of a mesh because the location of the inserted inner vertices is predicted and known prior to the connection phase. Heavy post improvements are avoided

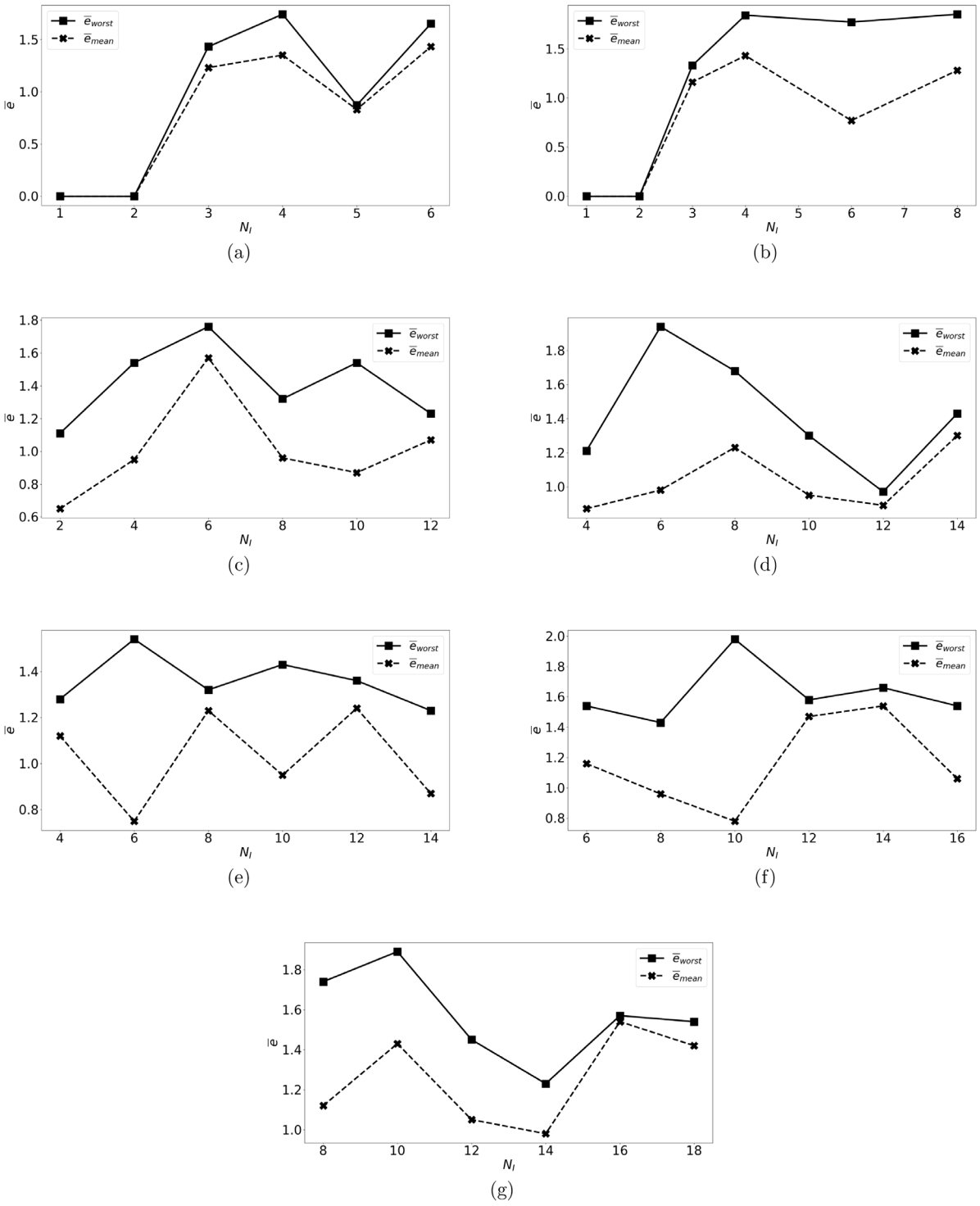


Fig. 24. (a)–(g) The average triangulation errors, $\bar{e}_{worst}$ and $\bar{e}_{mean}$, between meshes that are generated by using the triangulation algorithm with calculated connection table and meshes that are generated using CDT for random contours with $N_C = \{4, 6, 8, 10, 12, 14, 16\}$ edges as a function of number of inner vertices N_I. The levels of triangulation errors indicate that the triangulation algorithm causes little to no significant error propagation in the connection scheme.

because inner vertices location and connectivities are trained so as to maximize mesh quality. The algorithm is compact and easily transportable. Furthermore, the meshing scheme features the ability to be extended to generate larger scale meshes with an adaptive or uniform element size using its trained NNs.

A training set that includes $N_C = \{4, 6, 8, 10, 12, 14, 16\}$ edges is generated and CDT is applied for target edge lengths ranging between 0.2 and 1. The number of the inner vertices inserted by CDT is used to train NN_1; their location is used to extract scores of grid points that are used to train NN_2. The training of the

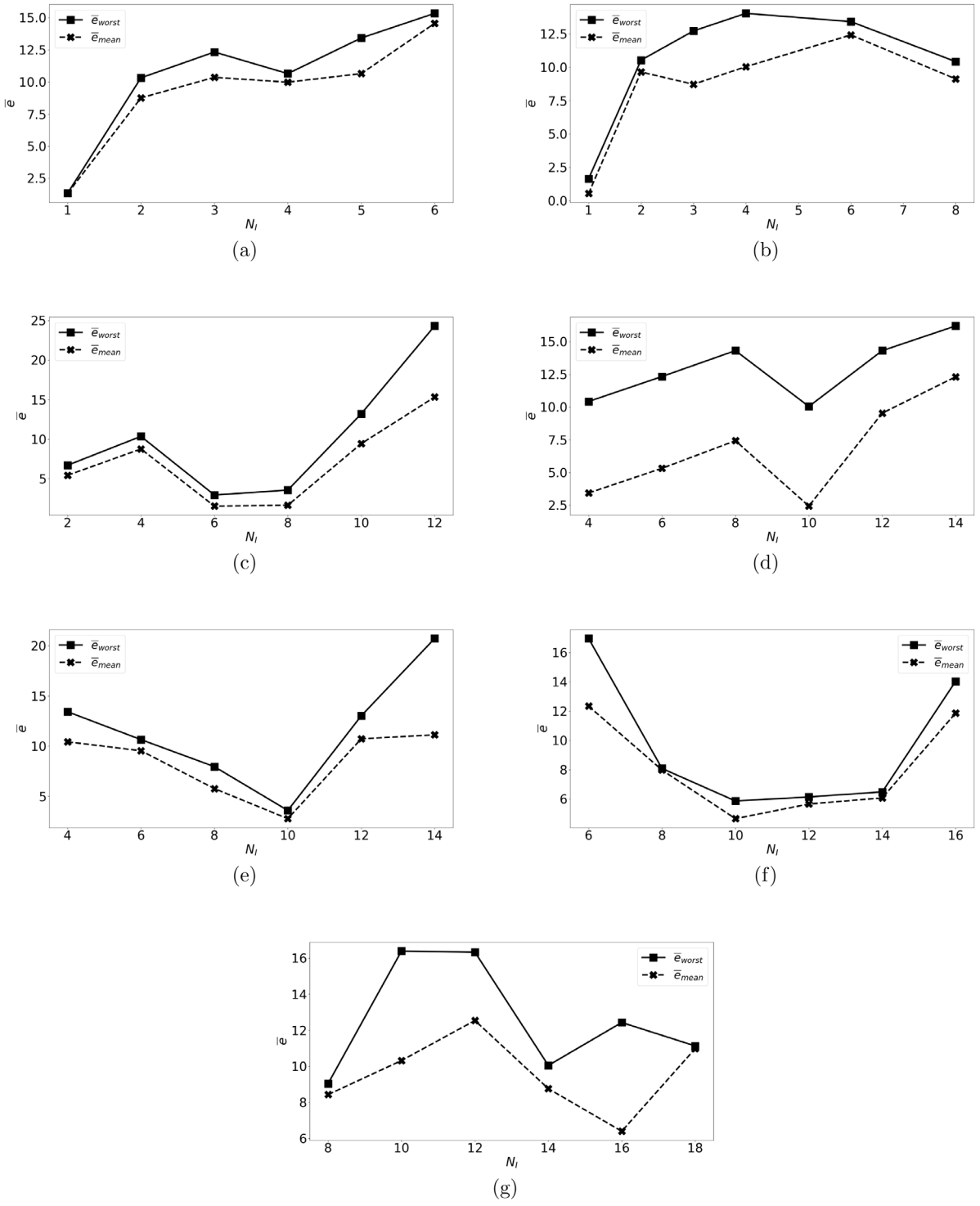


Fig. 25. (a)–(g) The average triangulation errors, $\bar{e}_{worst}$ and $\bar{e}_{mean}$, between contours meshed by using the triangulation algorithm (predicted connection table by NN_3) and meshes that are generated by using CDT for random contours with $N_C = \{4, 6, 8, 10, 12, 14, 16\}$ edges as a function of the number of inner vertices N_I. The inner vertices are those inserted in the cavity by applying CDT. The accuracy of NN_3 is dependent on the number of N_I sampled inner vertices of a contour during the data augmentation process. For example, in the case of contours with 6 edges for $N_I = 6$ (b) the training contour population is 37% lower than that for $N_I = 4$ but there is 23% reduction in $\bar{e}_{worst}$; this is due to the fact that the training of NN_3 for $N_I = 6$ relies on sampling 100 groups of six inner vertices for each contour, whereas for $N_I = 4$, 50 groups of four inner vertices are sampled.

connectivity network NN_3 is based on an augmentation scheme; for a contour, multiple N_I pairs of inner vertices are sampled to calculate the connection tables that are used to train NN_3. The sampling of the inner vertices is based on a random selection of grid points located inside the contour with a target edge length rule. To reduce the training data for the connectivity network, an adaptive sampling strategy of inner vertices is also studied. The meshing scheme starts by applying a feature transformation to a

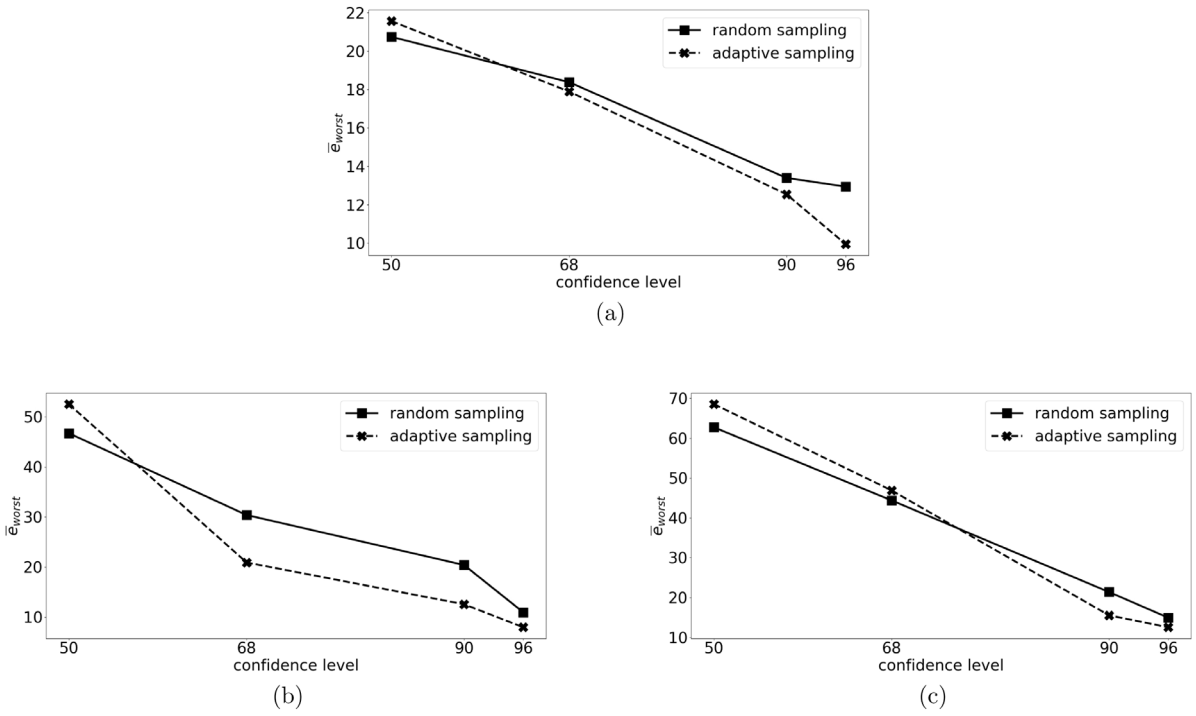


Fig. 26. Average worst triangulation error $\bar{e}_{worst}$ when NN_3^* is trained for different confidence levels of inner vertices populations (Table 1) with random and adaptive sampling. (a) For the prediction of the connectivity with one inner vertex at 96% confidence level, the sample population of inner vertices with adaptive sampling is 35% lower than that with random sampling. Even though NN_3^* is trained with lesser number of samples, 27% better accuracy is achieved. (b) For the prediction of the connectivity with two inner vertices at 90% confidence level, 18% better accuracy is achieved for a 22% lower sample population by applying adaptive sampling. (c) The accuracy for prediction of the connectivity with four inner vertices at 90% confidence level is 27% higher for a 17% lower sample population with adaptive sampling as compared to random sampling.

contour and uses the predictions of the NNs in a pipeline fashion to mesh it (Fig. 31).

The accuracy of every NN involved in the meshing scheme and that of the overall meshing scheme is determined based on the reference CDT mesher. For the prediction of the number of inner points N_I by NN_1, it is observed that the mean absolute error $\bar{e}$ on the number of points increases with number of contour edges N_C and the decrease in the target edge length l_s; this can be attributed to the increase of the standard deviation on the number of points that are inserted by CDT. A maximum mean absolute error of $\bar{e} = 1.5$ occurs for contours with $N_C = 16$ edges and a target edge length $l_s = 0.2$. For predicting the locations of the inner vertices using NN_2, the use of a grid with resolution 20×20 is proven to be adequate to obtain an accurate approximation on the location of the inner vertices. It is also observed that the worst triangulation error e_{worst} is increased with the increase in the number of inner vertices; the accumulative displacement error from the prediction of inner vertices increases the triangulation errors e_{worst} and e_{mean}. A maximum $\bar{e}_{worst} = 23.41\%$ occurs in the case of contours with $N_C = 12$ edges and $N_I = 14$. When assembling the connections using the triangulation algorithm of the meshing scheme, it is validated that compared to CDT it does not account for a significant error propagation in the connection scheme. For the prediction of the connectivity of the mesh using NN_3, it is observed that the accuracy improves with the increase in the number of N_I-pair of vertices that are sampled for each contour. A maximum $\bar{e}_{worst} = 24.04\%$ occurs for contours with $N_C = 8$ edges and $N_I = 12$ inner vertices. When applying the meshing scheme by using all the neural networks, it is demonstrated that the maximum average worst triangulation error, $\bar{e}_{worst}$, is 27.3% for the studied contour population; in all cases, the triangulation error is higher than the triangulation error when NN_2 and NN_3 are used to separately. To give a clear insight into the accuracy of the meshing scheme, given the quality metric q_{el} used (Section 2.5.1) for an ideal mesh composed of regular triangular elements with 60° angles, the aforementioned level of error corresponds to variation in angles between 28° and 106° in the worst case.

These levels of error confirm the reasonably good quality of meshes generated by the scheme. Growth in mesh complexity (i.e. increase in the number of contour edges and number of inner vertices) requires additional training data to attain the demonstrated levels of accuracy from the *NNs* involved in the meshing algorithm. The training contour population is increased exponentially with the number of N_C edges. Furthermore, the pairs of N_I inner vertices per contour used for the training of connectivity predictions, increase with the number N_I, reaching a maximum of 200 groups of N_I inner vertices for $N_I > 9$; although these pairs are chosen with a target edge length criterion, their selection is random. Adaptive sampling allows to increase the accuracy for the prediction of the connectivity while using significantly less data; it is shown that using the adaptive sampling strategy for a group of contours with $N_C = 10$ edges with $N_I = 4$ vertices, a 27% higher accuracy can be achieved with a 17% less sample population compared to the random sampling of inner vertices. To further optimize the accuracy of the overall scheme and reduce the accumulation of large training sets, it is worth investigating more strategic approaches for the selection of contour populations and pairs of inner vertices used for the training procedure. The proposed procedure might still require a large dataset for contour with a high number of contour vertices. However, up to a certain amount of vertices on a contour, it

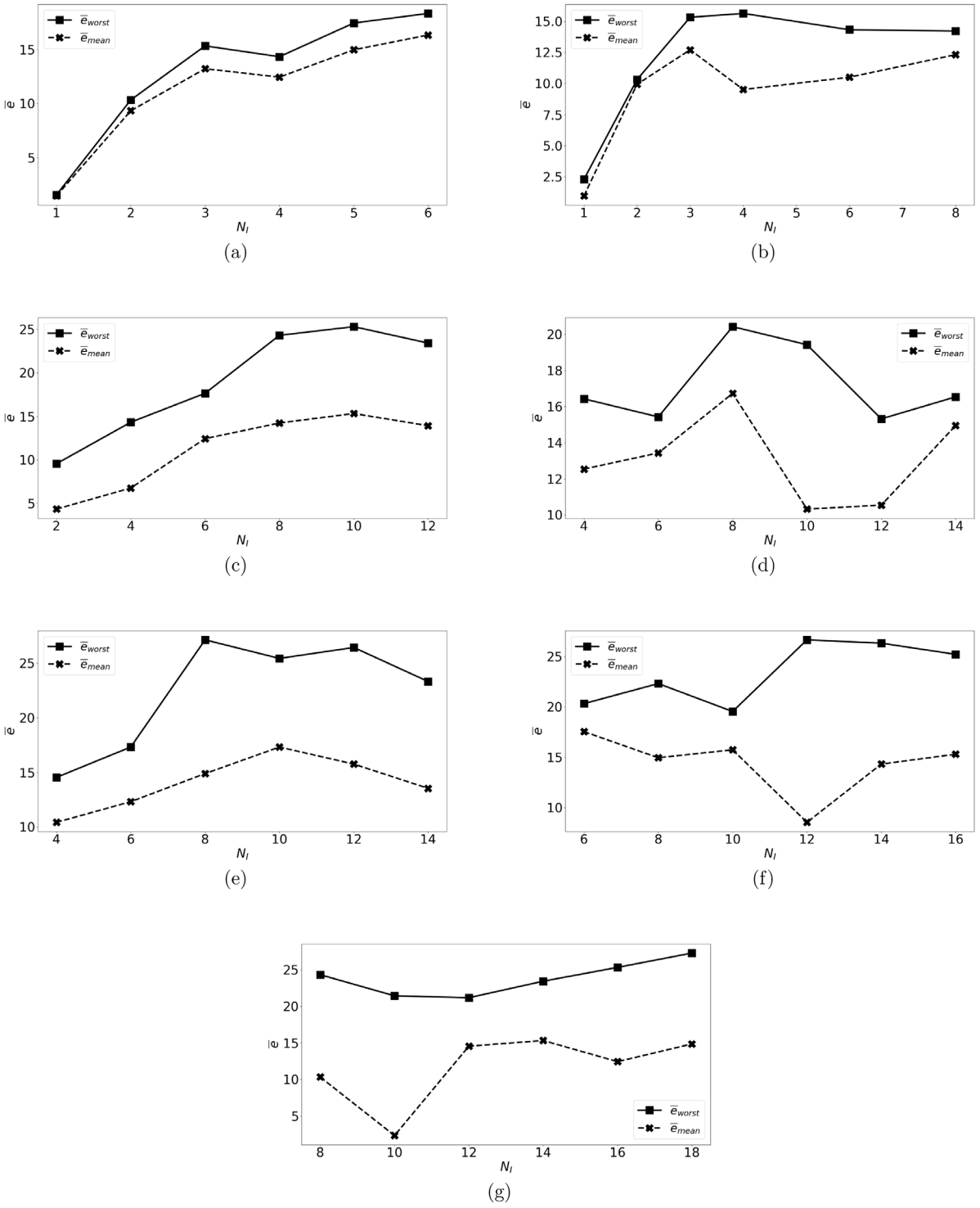


Fig. 27. (a)–(g): The average triangulation errors, $\bar{e}_{worst}$ and $\bar{e}_{mean}$, between meshes generated with the meshing scheme and those by CDT for random contours with $N_C = \{4, 6, 8, 10, 12, 14, 16\}$ edges as a function of number of inner vertices N_I. The inner vertices are those predicted by NN_2. In all cases, there is an increase of $\bar{e}_{worst}$ and $\bar{e}_{mean}$ compared to the previous tests where NN_2 and NN_3 are used separately. A maximum $\bar{e}_{worst}$ of 27.3% occurs for the case of contours with 16 edges and 18 inner vertices

is meaningful to mesh regions of a domain separately, as the meshes generated in two mesh regions located far apart are expected to be weakly correlated.

The meshing scheme overcomes some limitations of previous approaches using NNs with supervised learning for mesh generation (Vinyals et al., 2015; Yao et al., 2005; Zhang et al., 2020) and is more automatic as it is able to: (i) generate topologically valid meshes and provide full triangle coverage of a domain (ii) generate meshes without the use of an external meshing algorithm (iii) be trained using an automatic procedure without the need of manual exploration for training patterns. Furthermore, for the generation of large scale meshes, the extended scheme (Section 4) offers a more direct approach for mesh generation when compared to unsupervised learning NNs; the use of SOM,

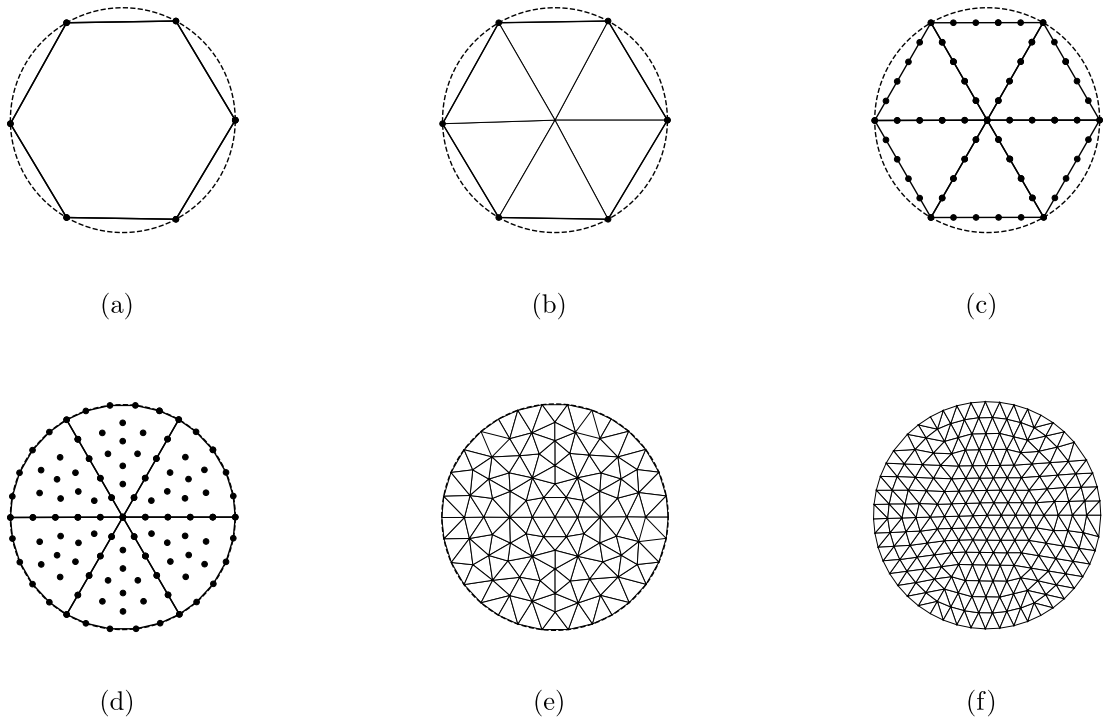


Fig. 28. Example of the refinement process using NN_3 to create a mesh with a target edge length l_s for a high resolution contour S (150 edges) forming a circle. (a) Points are sampled from S to form an initial contour. (b) NN_3 is called to mesh the initial contour. (c) If an edge of the initial mesh has a length l_e that is bigger than l_s, then $n_K = K - 1$ vertices are inserted to the edge, where $K = [l_e/l_s]$. (d) If the vertices of an edge belong to the high resolution contour S, the inserted vertices are projected to S. NN_1 and NN_2 are used to predict the number and location of inner vertices for each sub-contour with a target edge length equal to the average edge length of each sub-contour. (e) Each sub-contour with its inner vertices is meshed using NN_3. As a post-treatment, NN_2 is called to compute mesh coordinates and improve the quality. (f) The process is repeated until the edge lengths are close to l_s.

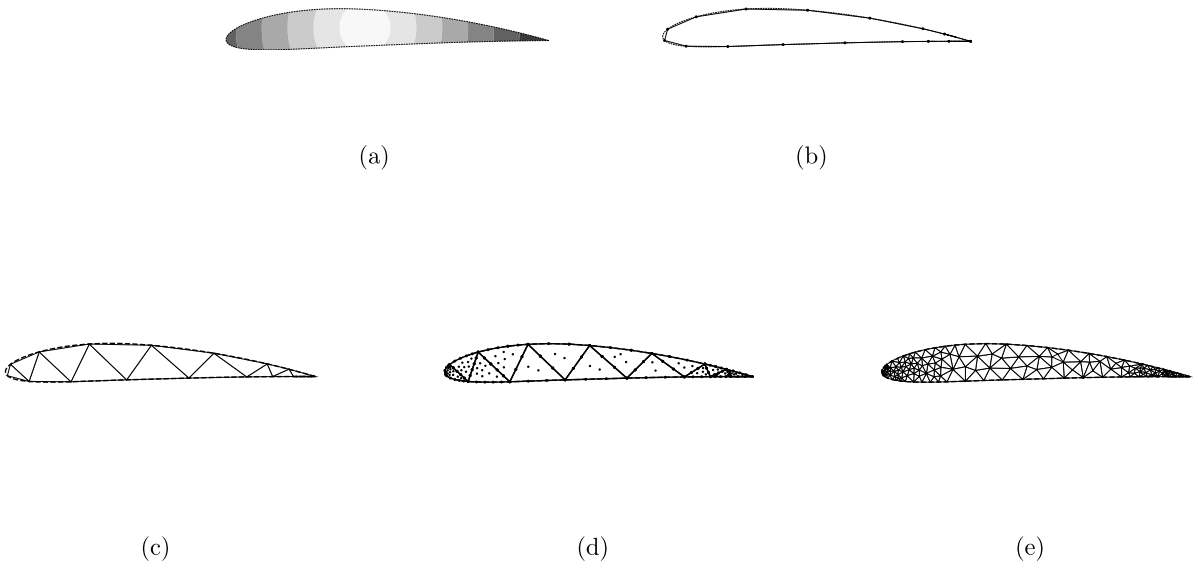


Fig. 29. Example of adaptive meshing for a high resolution contour S forming an airfoil. (a) A sizing function h is defined over the inner domain Ω_S. The values of the sizing function represent the local target edge lengths that will dictate the elements sizes. The darker areas represent smaller values of the sizing function, i.e. regions where smaller elements should be created to better approximate the geometry of the airfoil. (b) Points are sampled from S to form an initial contour that is meshed using NN_3 (c). Points are inserted incrementally on each edge until all lengths of the segments that are created after the subdivision are close to the assigned size function value for each inserted vertex. The number of interior points and their location are predicted using NN_1 and NN_2. (e) Finally, after meshing each sub-contour with its inner vertices using NN_3, NN_2 is called to calculate new mesh coordinates and improve the quality.

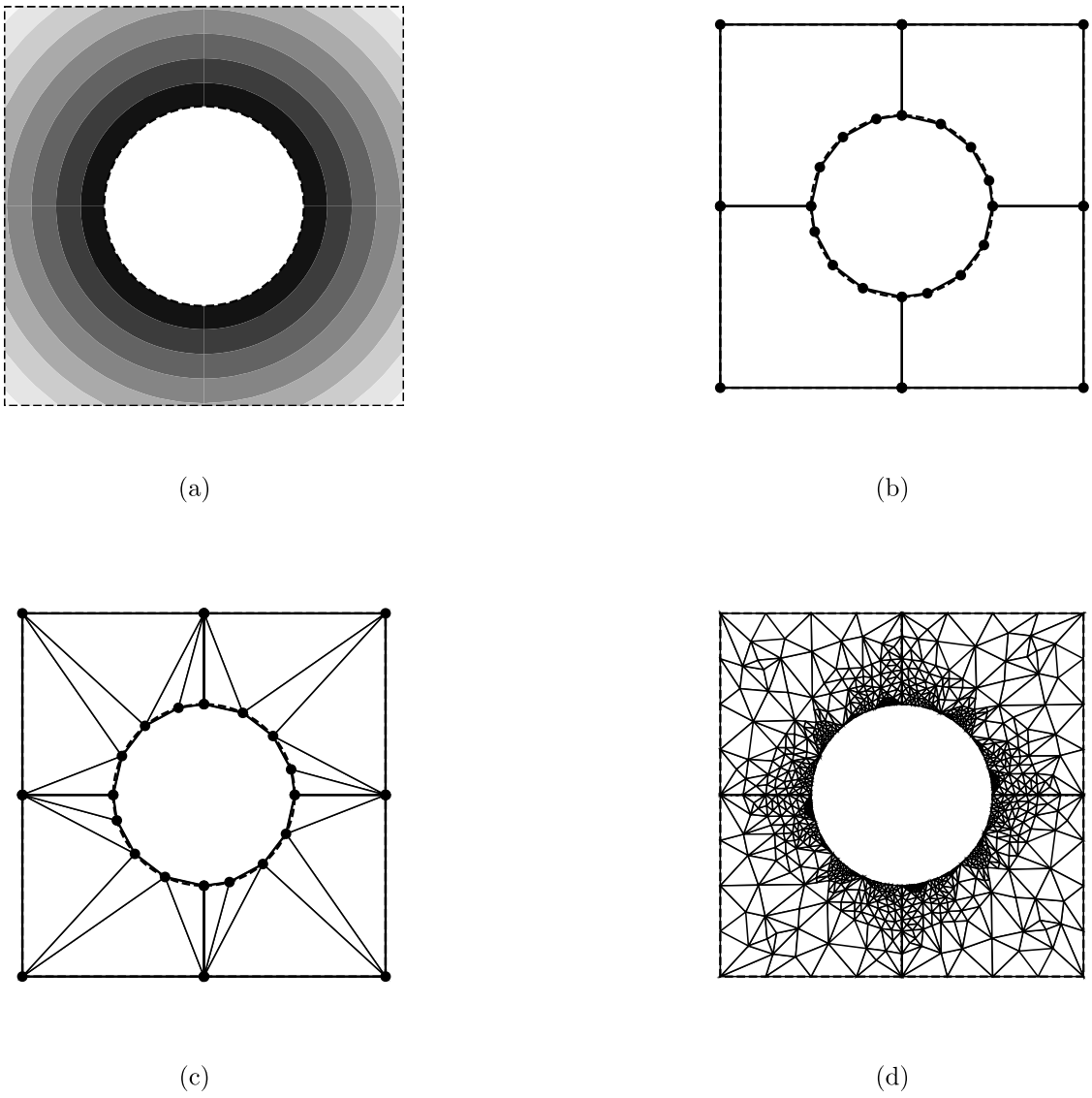


Fig. 30. Example of adaptive meshing for a high resolution contour with a circular hole in the middle. (a) The sizing function is defined such that elements of smaller size are created near the circular hole. (b) Since NN_3 is able to only mesh contours that are watertight (i.e. no holes), the high resolution contour is divided into four sub-regions. Points are sampled from the sub-regions to form four contours. (c) Each of the four contours, is meshed using NN_3. (d) Based on the initial meshes, the adaptive meshing process is applied and iterated until the edge lengths of elements are close to the values of the assigned sizing function.

LIG and GNG entail a computational complexity (the search for BMU) and many iterations to converge to an acceptable mesh. The worst case example of the contour with a circular hole (Fig. 30) has a minimum and mean quality $(q_{worst}, q_{mean}) = (0.66, 0.85)$ (Table 2). In Alfonzetti et al. (1996) and Triantafyllidis and Labridis (2002) that use LIG networks for simplicial (triangular) mesh generation, worst case examples are presented with $(q_{worst}, q_{mean}) = (0.38, 0.92)$ and $(q_{worst}, q_{mean}) = (0.29, 0.90)$ respectively. Although the examples address mesh generation for different geometries, the results show that the extended scheme is able to generate large meshes with a quality that is in line with previous approaches.

Performance tests indicate that the proposed meshing scheme is approximately four times slower than the reference mesher. The meshing scheme is coded in *Python* while the reference mesher is written in C++; assuming a speed factor of 5 to 20 between *Python* and *C++*, and that the current implementation of the algorithm is not optimized for performance, the aforementioned difference in speed validates that the scheme attains reasonably good performance. There is also a large potential to increase the speed of the meshing algorithm as it is transportable in terms of code and memory to acceleration platforms such as GPU and FPGA architectures. As future work, since the scheme is transparent to 3D, it is intended to be applied for the creation of tetrahedral meshes.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

We would like to thank the Swiss National Science Foundation (SNSF) (Project Grant No. PZENP2_166865) for the financial support.

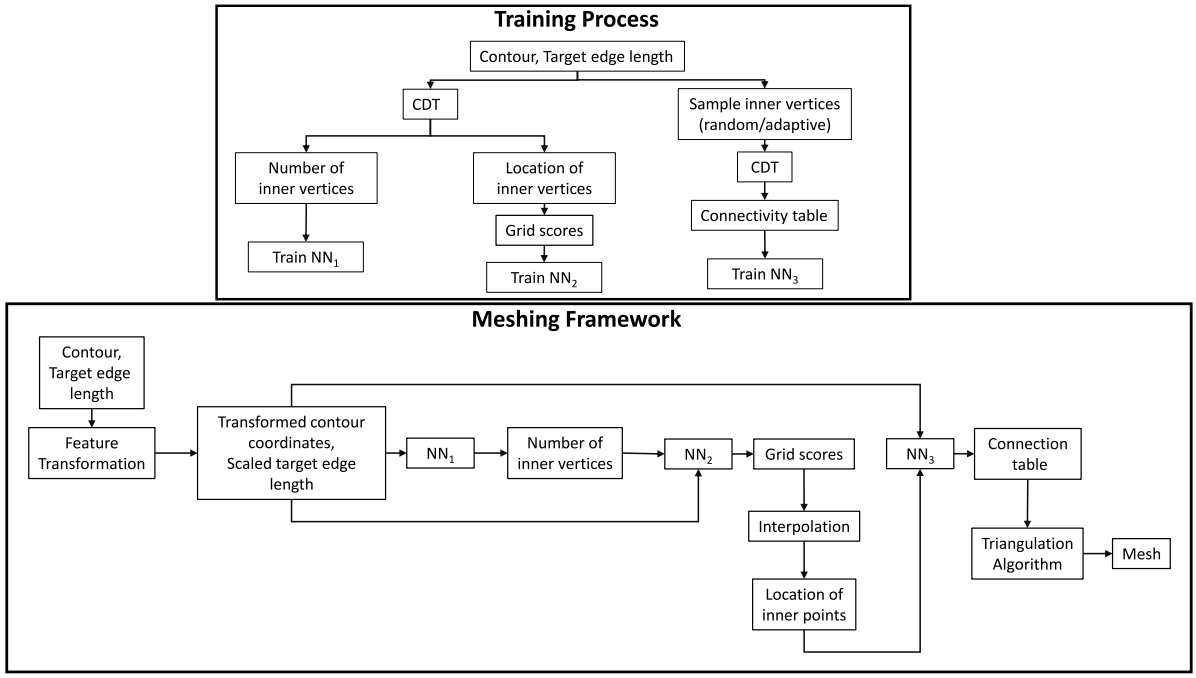


Fig. 31. Flowchart representing the training process and the meshing scheme.

References

Ahn Chang-Hoi, Sang-Soo, Lee, Hyuek-Jae, Lee, & Soo-Young, Lee (1991). A self-organizing neural network approach for automatic mesh generation. *IEEE Transactions on Magnetics*, *27*(5), 4201–4204.

Alfonzetti, S., Coco, S., Cavalieri, S., & Malgeri, M. (1996). Automatic mesh generation by the let-it-grow neural network. *IEEE Transactions on Magnetics*, *32*(3), 1349–1352.

Alliez, P., Cohen-Steiner, D., Yvinec, M., & Desbrun, M. (2005). Variational tetrahedral meshing. *ACM Transactions on Graphics*, *24*(3), 617–625.

Baker, T. J. (1997). Mesh adaptation strategies for problems in fluid dynamics. *Finite Elements in Analysis and Design*, *25*(3), 243–273, Adaptive Meshing, Part 2.

Baqué, P., Remelli, E., Fleuret, F., & Fua, P. (2018). Geodesic convolutional shape optimization. CoRR, abs/1802.04016.

Beniere, R., Subsol, G., Gesquière, G., Breton, F. L., & Puech, W. (2013). A comprehensive process of reverse engineering from 3D meshes to CAD models. *Computer-Aided Design*, *45*(11), 1382–1393.

Bennet, J. A., & Botkin, M. E. (1985). Structural shape optimization with geometric description and adaptive mesh refinement. *AIAA Journal*, *23*(3), 458–464.

Boscaini, D., Masci, J., Rodolà, E., & Bronstein, M. (2016). Learning shape correspondence with anisotropic convolutional neural networks. In *Advances in neural information processing systems* (pp. 3189–3197).

Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., & Vandergheynst, P. (2017). Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*, *34*(4), 18–42.

Bruna, J., Zaremba, W., Szlam, A., & Lecun, Y. (2014). Spectral networks and locally connected networks on graphs. In *International conference on learning representations (ICLR2014), CBLS, 2014.*

Calisto, M. B., & Lai-Yuen, S. K. (2020). Adaen-net: An ensemble of adaptive 2d-3D fully convolutional networks for medical image segmentation. *Neural Networks*, *126*, 76–94.

Camata, J. J., & Coutinho, A. L. G. A. (2013). Parallel implementation and performance analysis of a linear octree finite element mesh generation scheme. *Concurrency Computations: Practice and Experience*, *25*(6), 826–842.

Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., & Yuille, A. (2016). Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *PP*.

Cheng, S.-W., Dey, T. K., Edelsbrunner, H., Facello, M. A., & Teng, S.-H. (2000). Silver exudation. *Journal of the ACM*, *47*(5), 883–904.

Clausen, P., Wicke, M., Shewchuk, J. R., & O'Brien, J. F. (2013). Simulating liquids and solid-liquid interactions with Lagrangian meshes. *ACM Transactions on Graphics*, *32*(2), 17:1–17:15.

Defferrard, M., Bresson, X., & Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems* (pp. 3844–3852).

ElAdel, A., Zaied, M., & Amar, C. B. (2017). Fast DCNN based on FWT, intelligent dropout and layer skipping for image retrieval. *Neural Networks*, *95*, 10–18.

Fort, J. (2006). SOM's mathematics. *Neural Networks*, *19*(6), 812–816, Advances in Self Organising Maps - WSOM'05.

Fritzke, B. (1995). A growing neural gas network learns topologies. In *Advances in neural information processing systems* (pp. 625–632).

Fujita, K., Katsushima, K., Ichimura, T., Hori, M., & Maddegedara, L. (2016). Octree-based multiple-material parallel unstructured mesh generation method for seismic response analysis of soil-structure systems. *Procedia Computer Science*, *80*, 1624–1634, International Conference on Computational Science 2016, ICCS 2016, 6-8 2016, San Diego, California, USA.

Gower, J. C. (1975). Generalized procrustes analysis. *Psychometrika*, *40*(1), 33–51.

Graves, A., Wayne, G., & Danihelka, I. (2014). Neural turing machines.

Guo, J., Ding, F., Jia, X., & Yan, D.-M. (2019). Automatic and high-quality surface mesh generation for CAD models. *Computer-Aided Design*, *109*, 49–59.

Gupta, V., & Prasad, V. (2012). Numerical investigations for jet flow characteristics on pelton turbine bucket. *International Journal of Emerging Technology and Advance Engineering*, *2*, 364–370.

Hanocka, R., Hertz, A., Fish, N., Giryes, R., Fleishman, S., & Cohen-Or, D. (2019). MeshCNN: a network with an edge. *ACM Transactions on Graphics*, *38*, 1–12.

Hassan, O., Morgan, K., Probert, E., & Peraire, J. (1996). Unstructured tetrahedral mesh generation for three-dimensional viscous flows. *International Journal for Numerical Methods in Engineering*, *39*(4), 549–567.

Henaff, M., Bruna, J., & LeCun, Y. (2015). Deep convolutional networks on graph-structured data. CoRR, abs/1506.05163.

Holdstein, Y., & Fischer, A. (2008). Three-dimensional surface reconstruction using meshing growing neural gas (mgng). *The Visual Computer*, *24*(4), 295–302.

Hu, Y., Zhou, Q., Gao, X., Jacobson, A., Zorin, D., & Panozzo, D. (2018). Tetrahedral meshing in the wild. *ACM Transactions on Graphics*, *37*(4), 60:1–60:14.

Hughes, T., Cottrell, J., & Bazilevs, Y. (2005). Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Computer Methods in Applied Mechanics and Engineering*, *194*(39), 4135–4195.

Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *5th international conference on learning representations, ICLR 2017, Toulon, France, April (2017) 24-26, conference track proceedings:* . OpenReview.net.

Klingner, B. M., & Shewchuk, J. R. (2008). Aggressive tetrahedral mesh improvement. In *Proceedings of the 16th international meshing roundtable* (pp. 3–23).

Kohonen, T. (2013). Essentials of the self-organizing map. *Neural Networks*, *37*, 52–65, Twenty-fifth Anniversay Commemorative Issue.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems, Vol. 25* (pp. 1097–1105). Curran Associates, Inc.

Labelle, F., & Shewchuk, J. R. (2007). Isosurface stuffing: Fast tetrahedral meshes with good dihedral angles. *ACM Transactions on Graphics*, *26*, 57.

Lambrechts, J., Comblen, R., Legat, V., Geuzaine, C., & Remacle, J.-F. (2008). Multiscale mesh generation on the sphere. *Ocean Dynamics*, *58*(5), 461–473.
Lavoué, G., Dupont, F., & Baskurt, A. (2005). A new CAD mesh segmentation method, based on curvature tensor analysis. *Computer-Aided Design*, *37*(10), 975–987.
LeCun, Y. (2012). Learning invariant feature hierarchies. In *European conference on computer vision* (pp. 496–505).
Lemos, J., Oliveira, S., & Mendes, P. (2008). Analysis of the dynamic behaviour of cabril dam considering the influence of contraction joints. In *7th European conference on structural dynamics (EURODYN 2008)*. Univ. de Southampton.
Litany, O., Bronstein, A., Bronstein, M., & Makadia, A. (2018). Deformable shape completion with graph convolutional autoencoders. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1886–1895).
Liu, Y., Saputra, A. A., Wang, J., Tin-Loi, F., & Song, C. (2017). Automatic polyhedral mesh generation and scaled boundary finite element analysis of STL models. *Computer Methods in Applied Mechanics and Engineering*, *313*, 106–132.
Lohner, R. (1995). Mesh adaptation in fluid mechanics. *Engineering Fracture Mechanics*, *50*(5), 819–847.
López-Rubio, E., & Ramos, A. D. (2014). Grid topologies for the self-organizing map. *Neural Networks*, *56*, 35–48.
Ma, X., & Sun, L. (2019). An automatic approach to constrained quadrilateral mesh generation. *Engineering Computations*, *37*, 929–951.
Manevitz, L., Yousef, M., & Givoli, D. (1997). Finite element mesh generation using self organizing neural networks. *Computer-Aided Civil and Infrastructure Engineering*, *12*(4), 233–250.
Maron, H., Galun, M., Aigerman, N., Trope, M., Dym, N., Yumer, E., Kim, V. G., & Lipman, Y. (2017). Convolutional neural networks on surfaces via seamless toric covers. *ACM Transactions on Graphics*, *36*(4).
Martinetz, T., & Schulten, K. (1994). Topology representing networks. *Neural Networks*, *7*(3), 507–522.
Masci, J., Boscaini, D., Bronstein, M., & Vandergheynst (2015). Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings of the IEEE international conference on computer vision workshops* (pp. 37–45).
Mavriplis, D. J. (1995). An advancing front delaunay triangulation algorithm designed for robustness. *Journal of Computational Physics*, *117*(1), 90–101.
Melato, M., Hammer, B., & Hormann, K. (2007). *Neural Gas for Surface Reconstruction*: *Institut für informatik-ifi technical report series*.
Misztal, M. K., Erleben, K., Bargteil, A., Fursund, J., Christensen, B. B., Bærentzen, J. A., & Bridson, R. (2012). Multiphase flow of immiscible fluids on unstructured moving meshes. In *Proceedings of the ACM SIGGRAPH/eurographics symposium on computer animation, SCA '12* (pp. 97–106). Goslar Germany, Germany: Eurographics Association.
Monti, F., Boscaini, D., Masci, J., Rodolà, E., Svoboda, J., & Bronstein (2017). Geometric deep learning on graphs and manifolds using mixture model CNNs. In *2017 IEEE conference on computer vision and pattern recognition (CVPR)* (pp. 5425–5434).
Nechaeva, O. (2006). Composite algorithm for adaptive mesh construction based on self-organizing maps. In S. D. Kollias, A. Stafylopatis, W. Duch, & E. Oja (Eds.), *Artificial neural networks – ICANN 2006* (pp. 445–454). Berlin, Heidelberg: Springer Berlin Heidelberg.
Neil Molino, R. B., & Fedkiw, R. 2003. Tetrahedral mesh generation for deformable bodies. In *In Proc. symposium on computer animation*.
Owen, S. J. (1998). A survey of unstructured mesh generation technology. In *International meshing roundtable* (pp. 239–267).
Pajarola, R., Antonijuan, M., & Lario (2002). QuadTIN: Quadtree based triangulated irregular networks. In *Proceedings of the conference on visualization '02, VIS '02* (pp. 395–402). Washington, DC, USA: IEEE Computer Society.
Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., & Lerer, A. (2017). Automatic differentiation in pytorch. In *NIPS-W*.
Paul Chew, L. (1989). Constrained delaunay triangulations. *Algorithmica*, *4*(1), 97–108.
Pochet, A., Filho, W. C., Lopes, H., & Gattass, M. (2016). A new quadtree-based approach for automatic quadrilateral mesh generation. *Engineering with Computers*, *33*, 275–292.
Portaneri, C., Alliez, P., Hemmer, M., Birklein, L., & Schoemer, E. (2019). Cost-driven framework for progressive compression of textured meshes. In *Proceedings of the 10th ACM multimedia systems conference, MMSys '19* (pp. 175–188). New York, NY, USA: ACM.
Schöberl, J. (1997). NETGEN an advancing front 2D/3D-mesh generator based on abstract rules. *Computing and Visualization in Science*, *1*(1), 41–52.
Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., & LeCun, Y. (2014). Overfeat: Integrated recognition, localization and detection using convolutional networks. CoRR, abs/1312.6229.
Shewchuk, J. R. (2002). Constrained delaunay tetrahedralizations and provably good boundary recovery. In *In eleventh international meshing roundtable* (pp. 193–204).
Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. CoRR, abs/1409.1556.
Sumner, R. W., & Popović, J. (2004). Deformation transfer for triangle meshes. *ACM Transactions on Graphics*, *23*(3), 399–405.
Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems* (pp. 3104–3112).
Tatarchenko, M., Park, J., Koltun, V., & Zhou, Q. (2018). Tangent convolutions for dense prediction in 3D. In *2018 IEEE/CVF conference on computer vision and pattern recognition* (pp. 3887–3896).
Triantafyllidis, D. G., & Labridis, D. P. (2002). A finite-element mesh generator based on growing neural networks. *IEEE Transactions on Neural Networks*, *13*(6), 1482–1496.
Verma, N., Boyer, E., & Verbeek, J. (2018). Feastnet: Feature-steered graph convolutions for 3D shape analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2598–2606).
Vinyals, O., Fortunato, M., & Jaitly, N. (2015). Pointer networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in neural information processing systems, Vol. 28* (pp. 2692–2700). Curran Associates, Inc.
Wang, N., Zhang, Y., Li, Z., Fu, Y., Liu, W., & Jiang, Y.-G. (2018). Pixel2mesh: Generating 3D mesh models from single RGB images. In *Proceedings of the European conference on computer vision (ECCV)* (pp. 52–67).
Wen, C., Zhang, Y., Li, Z., & Fu, Y. (2019). Pixel2mesh++: Multi-view 3D mesh generation via deformation. In *Proceedings of the IEEE international conference on computer vision* (pp. 1042–1051).
Wicke, M., Ritchie, D., Klingner, B. M., Burke, S., Shewchuk, J. R., & O'Brien, J. F. (2010). Dynamic local remeshing for elastoplastic simulation. *ACM Transactions on Graphics (TOG)*, *29*(4), 1–11.
Yao, S., Yan, B., Chen, B., & Zeng, Y. (2005). An ANN-based element extraction method for automatic mesh generation. *Expert Systems Applications*, *29*, 193–206.
Zhang, Z., Wang, Y., Jimack, P. K., & Wang, H. (2020). Meshingnet: A new mesh generation method based on deep learning. In V. V. Krzhizhanovskaya, G. Závodszky, M. H. Lees, J. J. Dongarra, P. M. A. Sloot, S. Brissos, & J. Teixeira (Eds.), *Computational science – ICCS 2020* (pp. 186–198). Cham: Springer International Publishing.