Palestine Journal of Mathematics

Vol 13(Special Issue I)(2024) , 189–208

GENERATION OF 2-D UNSTRUCTURED MESH USING A NOVEL ALGORITHM FOR NODE PLACEMENT

Mohamed Mrini, Amal Bergam, Anouar El Harrak and Hatim Tayeq

MSC 2010 Classifications: Primary 65D18, 65N50; Secondary 68U05, 68Q25, 65M50, 65M60.

Keywords and phrases: Mesh Generation, Unstructured Mesh, Node Placement, Algorithm, Distmesh.

Abstract In this work, we present improvements to the meshing algorithm, Distmesh, specifically regarding creating a nonuniform triangular mesh. We introduce a novel strategy for determining the placement of nodes in the domain targeted to be meshed, to generate an initial distribution based on a user-defined mesh size function. In our algorithm, the creation of well-shaped triangles, similar to the Distmesh algorithm, can be achieved by connecting the nodes using Delaunay triangulation and applying in the smoothing process a new internode force with an attractive effect. Finally, through a series of tests, we validate the benefits of our proposed enhancements with regard to computational efficiency and robustness while globally preserving mesh quality.

1 Introduction

Mesh generation plays a vital role in a variety of scientific and engineering fields. Generating efficient and high-quality meshes is crucial for obtaining accurate and reliable results in these domains. Over the years, numerous methods for creating mesh structures have been developed to address the challenges associated with creating high-quality meshes for numerical simulations. Some of the most common methods include structured and unstructured mesh generation approaches. Structured methods, such as finite difference and finite-volume techniques, utilize regularly spaced grid points and are commonly employed in problems with simple geometries. However, they may not be suitable for complex domains or problems with irregular boundaries. Unstructured mesh generation techniques, on the other hand, are more versatile and can adapt to complex geometries and boundary conditions. Popular unstructured methods include Delaunay triangulation [7], advancing front techniques [9, 10], and the Distmesh algorithm [13]. Such techniques have found extensive use in various applications due to their flexibility and adaptability [18, 3, 11].

Among various mesh generation techniques, the Distmesh algorithm [13], introduced by Persson and Strang, has become popular for its simplicity and versatility. This algorithm provides a straightforward approach to generating unstructured triangular meshes in two-dimensional domains using signed distance functions. Although the original Distmesh algorithm has demonstrated its effectiveness in many applications, there is still room for improvement regarding computational efficiency and robustness. The primary objective of this study is to propose and investigate a series of enhancements to the Distmesh algorithm, in order to address these computational limitations and further improve its performance in 2D mesh generation.

The Distmesh algorithm is an adaptable solution for creating unstructured triangular meshes in both 2D and 3D settings. Although it offers several advantages, the algorithm also presents some challenges. The probabilistic rejection method used can result in an excessively large number of nodes, causing complications. Additionally, node placement obtained via the rejection method may not conform to the intended distribution specified by the desired edge length function (the mesh size function), leading to inconsistencies in the final mesh. Furthermore, the mesh smoothing process requires a large number of iterations, which reduces the efficiency of the algorithm. The increased frequency of Delaunay re-triangulation also adds complexity to the mesh generation process in terms of computational cost. Finally, as the number of nodes grows, so does the demand for computational resources such as time and memory, making the algorithm less efficient when handling larger meshes.

In order to address these challenges, we have designed an algorithm specifically aimed at generating nodes within the initial distribution based on a user-defined size function. This process begins by placing nodes on the domain boundary with respect to the mesh size function, followed by injecting nodes within the domain using the same function. By employing this strategy, our mesh generator is able to produce an optimal node count while simultaneously reducing the number of iterations needed during our new mesh smoothing processes.

In this paper, we present a detailed description of the proposed improvements to the Distmesh algorithm, focusing on their foundations and practical implementation. We then conduct a comprehensive test to assess the effectiveness of the enhanced algorithm, comparing it with the original Distmesh approach. Through a series of tests, we demonstrate the benefits of our proposed enhancements in terms of computational efficiency and robustness while globally preserving mesh quality.

The structure of this paper proceeds in the following manner: Section 2 offers an overview of the Distmesh algorithm and some existing mesh generation techniques. Section 3 introduces the proposed enhancements to the Distmesh algorithm, details our new node placement strategy, and discusses the mesh smoothing forces adopted. Section 4 discusses the implementation of the new algorithm and the performance metrics used to evaluate the improved algorithm. In Section 5, we present the validation results and performance comparisons. In conclusion, Section 6 presents a summary and outlines potential avenues for future research.

2 Node Placement & Problem Statement

In this section, we present a concise overview of node placement strategies in mesh generation techniques and examine the most pertinent literature in the domain, with particular emphasis on the Distmesh algorithm.

2.1 Overview on Node Placement

Node placement is an essential factor in all meshing algorithms, as it requires the generation of nodes at varying positions throughout the region requiring meshing. This process is vital to accurately representing the geometry and providing a solid foundation for subsequent analysis. Various methods have been devised to address node placement, each with its own advantages and disadvantages. Numerous alternative meshing methods have been suggested in the literature, with the aim of overcoming the limitations of various techniques relative to each other. The choice of method can greatly impact the quality, efficiency, and adaptability of the final mesh, making it a critical consideration for researchers and engineers when designing and implementing mesh generation algorithms. Ultimately, selecting the most appropriate node placement strategy depends on the specific application, desired level of detail, and available computational resources. Here, we provide a brief background on node placement on mesh generation techniques and review the most relevant literature in the field.

Delaunay triangulation methods have been widely used for their ability to generate high-quality meshes with a well-defined mathematical foundation [7]. However, Delaunay triangulation can be computationally expensive and may struggle with complex geometries. Delaunay meshing methodologies typically involve the gradual addition of nodes into a pre-existing mesh [2]. Throughout this procedure, the Delaunay criterion is preserved through the interconnection of nodes. However, relying solely on this approach can lead to a suboptimal quality mesh. To address this issue, the researchers in [4] and [14] proposed a technique to enhance the quality of triangles within a mesh by adding new nodes at the central points of the circumcircles or circumspheres of the element. This method ensures a lower bound on any angle present in the mesh, consequently boosting the quality of the triangle. However, the primary drawback of this approach is that the number of elements can grow substantially. This increase may lead to higher computational complexity and memory requirements, posing challenges for real-world applications. Balancing the trade-offs between mesh quality and resource efficiency is, therefore, an important consideration when employing this strategy for mesh generation.

Advancing front techniques represent another popular approach for mesh generation, offering the ability to generate high quality meshes by iteratively advancing the mesh boundary [12].

While these methods can produce excellent results, they can also be sensitive to the choice of parameters and may require significant user input. In advancing front triangular meshing techniques, as referenced in [9, 12], the nodes are also incrementally inserted. This method enables adaptive refinement of the mesh as it progresses, allowing greater control over local element density and size. However, generating a mesh comprised of thousands of elements using the advancing front method can be computationally demanding due to the substantial number of search operations required. These searches are necessary to identify suitable edges or triangles from the front and to ensure that the newly formed elements satisfy the mesh quality criteria. This can lead to increased processing time and computational resource consumption, particularly in cases with complex geometries or when high-quality meshes are desired.

In contrast to methods that involve sequentially placing nodes into a primal mesh, some meshing techniques focus on initially positioning all the nodes and subsequently connecting them to form the mesh. Physically-based mesh generators exemplify this approach, including bubble meshing method [15], Monte Carlo method [19], Distmesh [13], Molecular dynamics-based method [20], and the more recent FlowMesher method [17]. These techniques often rely on principles derived from physics, such as force-based interactions or energy minimization, to guide node placement and optimize mesh quality. For instance, nodes might be distributed in a way that simulates the behavior of repelling particles or bubbles, striving to achieve an even spacing throughout the domain. Once the nodes are optimally placed, they are connected to form the mesh elements, resulting in a well-distributed and high-quality mesh.

Physically-based meshing methods offer several advantages, in particular the ability to generate meshes with a more uniform distribution of elements and improved adaptability to complex geometries. However, they may also present certain challenges, including increased computational complexity and potential convergence issues. The choice of a meshing technique ultimately depends on the specific application, with factors such as desired mesh quality, computational resources, and processing time constraints, which plays a crucial role in the decision-making process.

The bubble meshing method involves populating the entire domain with spheres or bubbles and then creating well-shaped mesh elements by connecting their centers through constrained Delaunay triangulation or tetrahedrization after mesh relaxation. However, this approach necessitates a suitable initial bubble configuration to optimize convergence during relaxation and can be time consuming due to the packing process.

In the Monte Carlo method, the nodes are randomly inserted into the domain and treated as interacting particles. Employing a Monte Carlo simulation to reduce the potential energy of the system, these particles are placed for an almost optimal distribution. When the nodes are linked using constrained Delaunay triangulation, high-quality triangles can be created. However, due to the random insertion of points and their movement using the statistical sampling-based Monte Carlo approach to determine a configuration close to equilibrium, the method may result in more nodes than desired. This excess depends on the user-defined mesh element size function, and many of these nodes could be located near boundary edges, potentially leading to the formation of low-quality triangles close to the boundary of the domain.

In the Molecular dynamics-based method, the nodes are treated as interacting particles. After nodes are placed using molecular dynamics simulations, well-shaped triangles are formed by connecting nodes through Delaunay triangulation or tetrahedrization. Nevertheless, this method can be slow as a result of the inherently computationally intensive and time-consuming nature of molecular dynamics simulations, making it unsuitable for real-time remeshing requirements.

2.2 Distmesh Algorithm

Persson and Strang introduced the Distmesh algorithm as a simple and efficient technique to generate unstructured triangular meshes [13]. In the Distmesh algorithm, the domain's geometry targeted for meshing is depicted through the utilization of signed distance functions. This method enables the rapid identification of a specific point's position, whether it resides within or beyond the designated domain boundaries.

Consider a subset Ω of $\mathbb{R}^2$. The corresponding signed distance function related to Ω is established as follows:

$$d_\Omega(p) = \begin{cases} -\min\{d(p,B);\ B \in \partial\Omega\} & \text{if } p \in \Omega \\ +\min\{d(p,B);\ B \in \partial\Omega\} & \text{Otherwise.} \end{cases} \tag{2.1}$$

In this context, $\partial\Omega$ denotes the boundary of Ω, and $\min\{d(p,B);\ B \in \partial\Omega\}$ represents the smallest distance between point p and its nearest point on the boundary. Here, the selected signed distance function, d_Ω, exhibits negative values within the domain, zeros along the boundary, and positive values outside the domain Ω.

Here, the function $h : \Omega \to \mathbb{R}_+^*$ controls the size of the element and governs the density of the mesh. Although the function h does not directly represent the required size, it influences the relative distribution of nodes across the domain. To achieve a uniform node distribution, Distmesh generates a grid of nodes within the bounding box and eliminates all points external to Ω by using the signed distance function. In this scenario, $h(p) = 1$ for every $p \in \Omega$. However, to obtain a nonuniform distribution, a set of points is removed from the uniform distribution with a probability of $1/h(p)^2$.

An initial triangulation is achieved by connecting all nodes via Delaunay triangulation. However, many of the resulting triangles lack desirable quality. To address this issue, Persson and Strang proposed moving the nodes towards their optimal locations by employing a physical concept, the repulsive force. This idea comes from the similarity between a mesh of triangles and a two-dimensional truss configuration, where the repulsive force helps guide the nodes toward better positions. In this comparison, the triangle edges are analogous to bars, while the vertices act as truss joints. Each bar is connected to a spring constant, represented as $f(l_c, l_d)$, which is influenced by the current length of the bar l_c and the desired length l_d. By addressing a balanced force state within the structure, the node positions can be established through solving a corresponding nonlinear system of equations

$$W(p) = 0. \tag{2.2}$$

A synthetic time-related change is implemented:

$$\frac{dp}{dt} = W(p), \quad t \geq 0. \tag{2.3}$$

A stationary solution for the system of ordinary differential equations 2.3 satisfies the equation 2.2. This static solution is derived through the implementation of the forward Euler method:

$$p^{n+1} = p^n + \Delta t . W(p^n), \tag{2.4}$$

where the time discretization is $t_n = n\Delta t$ and $p^n = p(t_n)$.

That is

$$p_i^{n+1} = p_i^n + \Delta t \sum_{j \in \mathcal{N}_i} f(p_i^n, p_j^n) \frac{p_i^n - p_j^n}{(l_c)_{ij}^n} \qquad i = 1, ..., N. \tag{2.5}$$

In this context, N denotes the total count of nodes, $\mathcal{N}_i$ represents the neighboring nodes surrounding node i, and $(l_c)_{ij}^n = \left\| p_i^n - p_j^n \right\|$ signifies the distance between node i and node j at the n-th iteration. In addition, the bar force, denoted as $f(p_i^n, p_j^n)$, relies on both the current length $(l_c)_{ij}^n$ and the desired length $(l_d)_{ij}^n$, that is:

$$f(p_i^n, p_j^n) = \begin{cases} (l_d)_{ij}^n - (l_c)_{ij}^n & \text{if } (l_c)_{ij}^n < (l_d)_{ij}^n, \\ 0 & \text{if } (l_c)_{ij}^n \geq (l_d)_{ij}^n. \end{cases} \tag{2.6}$$

The desired bar length $(l_d)_{ij}^n$ is given by

$$(l_d)_{ij}^n = F_{scale} s^n h_{ij}^n. \tag{2.7}$$

Here, $s^n = (\sum_{ij}((l_c)_{ij}^n)^2 / \sum_{ij}(h_{ij}^n)^2)^{\frac{1}{2}}$ represents the scaling factor, while $h_{ij}^n = h((p_i^n + p_j^n)/2)$ refers to the value of the size function at the midpoint of the edge connecting the nodes p_i^n and p_j^n at the n-th iteration.

The fixed factor F_{scale} is chosen to ensure that most bars generate repulsive forces $f > 0$. Distmesh code utilizes $F_{scale} = 1.2$. Ensuring that most bars produce repulsive force is crucial to facilitate the distribution of points throughout the geometry. If point p_i^n crosses the geometry boundary following an update, it is relocated to the nearest position on the boundary by applying a specific formula:

$$p_i^n = p_i^n - d_\Omega(p_i^n)\nabla d_\Omega(p_i^n). \tag{2.8}$$

The gradient of d_Ω provides the direction towards the nearest boundary point and is calculated numerically employing a finite difference scheme. The formula 2.8 represents a perpendicular reaction force to the boundary.

The main advantage of the Distmesh algorithm is its simplicity and ease of implementation, making it an attractive option for various applications. Despite its popularity, the Distmesh algorithm has certain limitations, including sensitivity to initial conditions, potential for suboptimal mesh quality, and computational inefficiency in some cases. These drawbacks have motivated researchers to explore improvements and alternative methods for mesh generation. For instance, researchers have developed various optimization strategies, including the use of spatial data structures to accelerate search operations and the implementation of parallel processing techniques to distribute the computational load. The selection of an appropriate meshing method and an optimization approach ultimately depends on the specific application requirements, such as the desired mesh quality, computational resources, and processing time constraints.

In summary, the literature on mesh generation techniques, including the Distmesh algorithm and its alternatives, highlights the ongoing challenges in developing efficient and versatile methods for 2D mesh generation. In the following sections, we aim at addressing these challenges by proposing a series of enhancements to the Distmesh algorithm, focusing on improving its mesh quality, computational efficiency, and robustness. In fact, the node positions generated by the aforementioned mesh generation methods for creating unstructured meshes may not always align with the desired positions specified by the user-defined size function. Consequently, the generated mesh may not accurately represent the desired local element density, which leads to a suboptimal mesh disaccording with the user's needs. Such deviations can impact the mesh quality, which affects the subsequent analyses and simulations that rely on the mesh. To address these issues, the mesh generation algorithms can be enhanced or combined with the new node placement to better fit the user-defined size function. This approach can help ensure that the generated mesh meets the required element distribution, size, and density criteria, and increase computational efficiency.

3 The Improved Distmesh Algorithm

The Distmesh algorithm is efficient in creating unstructured triangular meshes in both 2D and 3D applications. However, despite its benefits, it has some limitations that manifest in the following areas:

- The probabilistic rejection method may result in the retention of an excessively large number of nodes, which can lead to complications.
- The node placement derived through the rejection method may not adhere to the intended node distribution as defined by the desired edge length function. This can cause inconsistencies in the generated mesh.
- The process of mesh smoothing, which aims to enhance the initial mesh, demands a significantly high number of iterations, contributing to inefficiency in the algorithm's performance.
- The frequency of Delaunay re-triangulation experiences an increase, further complicating the mesh generation process.
- With an increase in the number of nodes, the computational resources, including the time and memory space required to store the mesh elements, also grow proportionally. This can lead to increased resource consumption and reduced efficiency in handling larger meshes.

Due to these constraints, although Distmesh is capable of generating 3D meshes, its performance and mesh quality in 3D situations may not be as strong as in 2D cases. This leads to a more challenging environment when dealing with 3D scenarios, potentially affecting the algorithm's effectiveness and efficiency. To address these limitations, we have developed an algorithm focused on generating nodes for the initial distribution using a user-specified size function. This method starts by positioning nodes along the domain boundary in line with the mesh size function and continues by incorporating nodes inside the domain based on the same function. This approach enables our mesh generator to achieve an optimal node count and minimize the count of iterations necessary for mesh smoothing, ultimately improving the overall efficiency of the process.

Here, we present a series of enhancements to the Distmesh algorithm aimed at addressing its limitations and improving its performance for 2D mesh generation. We discuss the design of these modifications, which include a New Node Placement Strategy and an Adaptive Force Mechanism, to optimize the algorithm's effectiveness.

3.1 New Node Placement Strategy

The probabilistic rejection method employed in the Distmesh algorithm has the potential to retain a large number of nodes, which leads to complications in the mesh generation process. This issue can arise from the method's inherent randomness, which may not effectively consider the desired edge-length function during node placement. Consequently, the resulting node distribution may deviate from the intended distribution as defined by the mesh size function. This deviation can, in turn, cause inconsistencies within the generated mesh, which affects its quality and potentially impacts its performance in various applications. Here, we present our New Node Placement Strategy in order to replace the probabilistic rejection method in the Distmesh algorithm.

In our approach, we begin by partitioning the domain boundary into an initial collection of segments. The number of these segments is dictated by either the vertices that define the geometry or by an initial set of points specified by the user. Subsequently, we inject points into each of these segments in accordance with the mesh size function. Indeed, given two endpoints, p_i^1 and p_i^2, of a particular segment e_i, we insert a point p_i^{new} into the segment using the mesh size function h as follows:

$$p_i^{\text{new}} = \frac{\frac{p_i^1}{h(p_i^1)} + \frac{p_i^2}{h(p_i^2)}}{\frac{1}{h(p_i^1)} + \frac{1}{h(p_i^2)}}. \tag{3.1}$$

The set of points p_i^{new} injected on all segments forming the geometry will then be projected onto the boundary $\partial\Omega$ using:

$$p_i^{\text{new}} = p_i^{\text{new}} - d_\Omega(p_i^{\text{new}})\nabla d_\Omega(p_i^{\text{new}}). \tag{3.2}$$

After that, we repeat the process for the new set of obtained segment until a predefined stopping criterion is satisfied.

To guarantee that all points are well distributed along the domain boundary based on the mesh size function, we proceed to the 1D balancing phase. For this purpose, we employ a concept that allows each point to freely move within the limits of two barriers, specifically the midpoints of the two adjacent segments connected to the vertex in question. Let p_i, p_j, and p_k be three consecutive points, $M_{i,j}$ and $M_{j,k}$ be the midpoints of the segments $[p_i, p_j]$ and $[p_j, p_k]$, respectively. The point p_j is free to move between the two points $M_{i,j}$ and $M_{j,k}$ ($M_{i,j}$ and $M_{j,k}$ are the barriers) according to the iterative process

$$p_j^{n+1} = \frac{\frac{M_{i,j}^n}{h(M_{i,j}^n)} + \frac{M_{j,k}^n}{h(M_{j,k}^n)}}{\frac{1}{h(M_{i,j}^n)} + \frac{1}{h(M_{j,k}^n)}}. \tag{3.3}$$

Here, p_j^{n+1} represents the updated position of the point p_j^n. The iterative process begins with $p_j^0 = p_j$, and during each iteration, we evaluate all the boundary points p_j, excluding only the points considered fixed. In this strategy, we should prevent points from shifting in a circular manner during the 1D balancing phase by fixing a single point for closed smooth curves, any sharp turns for non-smooth ones, and at least the two endpoints for open curves.

After partitioning the boundary according to the function h, we start the second step; Advancing Node Injection technique. We inject points within the domain by initiating an advancing front injection process from the boundary edges moving inwards into the domain. Initially, we designate the starting boundary edges as the active front. As the method advances and new points are injected, we update the active front by deleting the edges already used for injecting points and adding new points used to create new boundary edges. By using this strategy, whose concept is similar to the Advancing Front Method, we can create a good initial distribution of nodes, which in turn reduces the required number of improvement steps. We employ the principle of Advancing Front Method to inject point without generating triangulation and performing excessive search operations, thereby reducing computational costs.

3.2 Attractive Internode Force for Mesh Smoothing

The mesh smoothing process is an essential component of the Distmesh algorithm. This process focuses on smoothing and balancing the initial distribution to create a more uniform and accurate mesh. However, this step often requires a considerably high number of iterations to achieve the desired level of quality. Furthermore, the smoothing function used in the Distmesh algorithm, which exhibits repulsive behavior, does not yield satisfactory results. Consequently, we have explored alternative functions, such as:

- Laplacien smoothing function [5] that exhibits an attractive behavior,
- Lennard-Jones function [16],

$$f(l_d, l_c) = (\frac{l_c}{l_d})^{-13} - (\frac{l_c}{l_d})^{-7}$$

  which has an attractive-repulsive behavior,
- Bossen-Heckbert smoothing function [1],

$$f(l_d, l_c) = (1 - (\frac{l_c}{l_d})^4) exp(-(\frac{l_c}{l_d})^4)$$

  which has also an attractive-repulsive behavior.

None of the functions that we have tested yielded satisfactory results in terms of the quality of the obtained triangles and the number of iterations in the mesh smoothing steps. As a result, we developed a function that adapts to the approach chosen in our algorithm. This smoothing function describes an attractive behavior and is defined as follows:

$$f(l_d, l_c) = -0.2\frac{l_c}{l_d}. \tag{3.4}$$

Our algorithm may provide better results than the Distmesh algorithm in terms of computational cost, especially when attempting to create a non-uniform triangular mesh with respect to a user-defined function h. Because, we replaced the probabilistic method used by the Distmesh algorithm to obtain an initial distribution with a deterministic approach, which consists of directly injecting points into their positions according to the function h with Advancing Node Injection. This allows us to obtain a good quality initial triangulation, reduce the number of iterations during the smoothing process, decrease the computation time, and satisfy user-defined requirements for the desired mesh.

In the following section, we will detail the implementation of our proposed enhancements to the original Distmesh algorithm. We will present our approach as a sequence of elementary steps that can be easily coded and executed using an appropriate program.

4 Implementation

The originality of our research can be found in the ability of our technique to directly inject points into the entire domain based on the user-defined mesh size function. In this section, we

Algorithm 1 The new algorithm for generating 2-D unstructured mesh.

```
Step 1: Meshing of Boundary: we inject nodes on the domain boundary according to the function h, then we perform a 1D Balancing of the injected points.
Step 2: Advancing Node Injection: we inject nodes inside the domain following the Advancing Node Injection technique.
Step 3: Attractive Smoothing Process: we perform a mesh smoothing process similar to the Distmesh algorithm, but with the application of an attractive force.
```

will detail the approach we followed as a sequence of elementary steps that can be easily coded and executed using an appropriate program. In Algorithm 1, we describe the key steps involved in generating a 2D unstructured mesh.

The merging of the boundaries is based on two crucial phases. The first phase is to partition the domain boundary into a collection of segments whose number and lengths are determined by the mesh size function h specified by the user, and using the iterative Injection-Projection process given by Formulas 3.1 and 3.2. After that, to look for optimal positions for those nodes on the boundary, the second phase comes to perform a 1D balancing technique using the iterative process given by Formula 3.3.

To discretize the boundary into a collection of segments according to the mesh size function h, we first consider a minimal set of initial points on the boundary that accurately represent its shape. This guarantees that the projection of any point within a segment lies on the nearest boundary curve of that segment, providing increased flexibility for our algorithm. Consequently, we can guarantee that, using the iterative injection-projection process, given by Formulas 3.1 and 3.2, the boundary is discretized while preserving the connectivity information between points. Furthermore, we should fix and add in the set of initial points all sharp turns for non-smooth curves, and include at least the two endpoints for open curves. To add more points in the initial set, we can enclose the boundary within a circle containing some points. These points can be numerically projected on the boundary using the function d_Ω and Formula 3.2. After that, we can perform the iterative Injection-Projection process on each segment by applying Algorithm 2 in order to discretize the corresponding part of the boundary.

The second phase consists of balancing all nodes injected on the boundary in the first phase, to look for optimal positions on the boundary. At this phase, the injected points are subjected to moving from their original positions in order to adjust their positions based on the iterative process 3.3. Here, we adopt a strategy based on moving each point, except fixed points, between the midpoints of its two adjacent segments, as shown in Algorithm 3, to preserve the connectivity information between points.

After partitioning the boundary based on the mesh size function h, we start the second step; Advancing Node Injection. We insert nodes inside the domain by initiating an advancing front injection process, starting from the boundary edges given by the Meshing boundary step and progressing inwards. Initially, we designate the starting boundary edges as the active front. As the method advances and new points are added, we update the active front by eliminating the edges that are already used to inject points and adding new points that are used to create new boundary edges. Consequently, we can create a good initial distribution of nodes, which reduces the required number of improvement iterations in the smoothing process.

Note that the condition on added angles in the node injection algorithm 4 within the domain is capable of resolving the issue of corners during the initial triangulation. In our code, we adopted a vectorized version because it is easier to understand, is often shorter, and runs much faster.

Attractive Smoothing Process is an essential component of our algorithm. This process focuses on smoothing and balancing the initial good distribution obtained in the Advancing Node Injection step to create a more uniform and accurate mesh. This step requires considerably less number of iterations to achieve the desired level of quality compared to the Distemsh algorithm. This result is achieved by using a smoothing function 3.4 that describes an attractive behavior, making it suitable for the approach chosen in our algorithm.

In the next section, we will validate the study through a series of tests in which we highlight the benefits of our improvements in terms of computational efficiency while globally preserving mesh quality, and we evaluate the performance of the improved algorithm in comparison with

Algorithm 2 Partitioning the corresponding part of the boundary to an initial segment.

Input:

- p: list of point defining the endpoints of the segment in question.
- e: list containing the indices of the endpoints of segments partitioning the part of the boundary. Initially, e contains the initial segment.
- h: mesh size function.
- d_Ω: distance function of the boundary.

Output:

- p: updated version of the input vector p containing the coordinates of points used in the partition.
- e: updated version of the input vector e that contains all segments that partition the boundary part.

```
while true do
    Evaluate the function $h$ at the endpoints of all segments within $e$.
    In each segment within $e$, inject one point using Formula 3.1. Let $p^{\text{New}}$ be the points injected in all segments.
    Project $p^{\text{New}}$ onto the boundary using Formula 3.2.
    Update the vector $p$ by adding the points $p^{\text{New}}$.
    Update the vector $e$ by substituting each segment with the two segments that join its endpoints with the newly added point.
    Evaluate the stopping condition $SC$.
    if $SC$ is satisfied then
        Break
    end if
end while
```

the original Distmesh algorithm.

5 Tests and Results

In this section, we compare the performance of both algorithms using some selected test cases and performance metrics. This comparison highlights the benefits of our proposed enhancements in terms of mesh quality, computational efficiency, robustness, and user-need requirement.

Mesh quality: numerous metrics exist to evaluate mesh quality in terms of "element quality" [6]. In this work, we use the ratio, i.e.

$$q = \frac{(b+c-a)(c+a-b)(a+b-c)}{abc}. \tag{5.1}$$

Here a, b, c are the three sides of a triangle. q represents the relationship between the radius of the largest inscribed circle and the smallest circumscribed circle.

Computational efficiency: CPU time and iteration count are used to evaluate the efficiency of the improved algorithm compared to the original Distmesh algorithm.

Robustness: the capability of the algorithm to produce high-quality meshes for a wide range of problem domains and geometries. We have selected a diverse set of test cases representing a range of geometries. These test cases include standard problems, such as the L-shaped domain and the circular cavity, as well as more complex.

User-need requirement: the capability of the algorithm to produce meshes that satisfy the mesh size function h defined by the user at the initial stage. For that, we calculate for each edge

Algorithm 3 Balancing the injected points on the boundary of the domain.

Input:

- p: list of the points used in the partition of the boundary.
- e: list containing the indices of the endpoints of segments partitioning the boundary.
- h: mesh size function.
- d_Ω: distance function of the boundary.
- p_{fix}: fixed points.

Output:

- p: updated version of the input vector p after the balancing process.
- e: updated version of the input vector e after the balancing process.

```
while true do
    Calculate the midpoint of each segment within $e$.
    Evaluate the function $h$ at the midpoint of all segments within $e$.
    Update the positions of the points $p$ using Formula 3.3, except for fixed points $p_{fix}$. Project
    the points $p$ back onto the boundary using Formula 3.2.
    Evaluate the stopping condition $SC$.
    if $SC$ is satisfied then
        Break
    end if
end while
```

e in the resulting mesh the difference between the actual edge length l_c and the desired edge length l_d evaluated by the function h at the midpoint of e. This error is given by:

$$Er = \frac{\sum_e |l_c - l_d|}{\text{Edges count}}.$$

In order to test the performance of our algorithm compared to the original Distmesh algorithm, we conducted some experiments that led to the following results. The computation has been carried out on HP EliteBook Folio 1040 computer equipped with Intel(R) Core(TM) i7-6600U CPU and 8GB of RAM.

Unit disc

In the unit disc, we will generate points in the initial distribution using the probabilistic rejection method used by the Distmesh algorithm and the approach followed by our algorithm, which consists of injecting nodes directly into the domain based on the size function h. From Figure 1, we can observe that the nodes generated by our algorithm (Figure 1-b) are very close to the desired positions according to the size function h, compared to the Distmesh algorithm (Figure 1-a).

After that, we connect the nodes obtained in the previous step to each other using the Delaunay algorithm to create the initial triangulation of the unit disc. Figure 2 shows that the initial triangulation obtained by our algorithm (Figure 2-b) requires fewer iterations in the mesh smoothing step than the initial triangulation obtained by the Distmesh algorithm (Figure 2-a).

After applying the mesh smoothing step to the initial triangulation obtained by our algorithm using an attractive force and to the initial triangulation obtained by the Distmesh algorithm using a repulsive force, we obtain the mesh of a unit disc by our algorithm (Figure 3-b) with an optimal number of nodes and a mesh of a unit disc by the Distmesh algorithm (Figure 3-a) with more

Algorithm 4 Injection of points inside the domain.

```
Input:
    • $P_{\text{bnd}}$: list of points used in the partition of the boundary.
    • $E_{\text{bnd}}$: list of segments, oriented in a counterclockwise direction, partitioning the boundary.
    • $h$: mesh size function.
    • $d_\Omega$: distance function of the boundary domain $\Omega$.
Output:
    • $P_{\text{dom}}$: the set of points injected into the entire domain.

  Add the set of points $P_{\text{bnd}}$ to the set $P_{\text{dom}}$.
  Consider the partition of the boundary domain as the initial front, $P_{\text{front}} = P_{\text{bnd}}$ and $E_{\text{front}} = E_{\text{bnd}}$.

  while true do

    Calculate the list of midpoints, $M$, of all edges on the active front $E_{\text{front}}$.
    For each segment $e_i$ within $E_{\text{front}}$, inject $m$ points into the domain following the function $h$ and considering the interior angles between $e_i$ and the previous and the next edges. We denote $p$ this set of inserted points.
    Update the active front by setting $P_{\text{front}} = p$ and $E_{\text{front}} = E$, where $E$ is the set of the oriented segment associated with $p$.
    Evaluate the stopping condition $SC$.

    if $SC$ is satisfied then
      Break
    end if

  end while
```

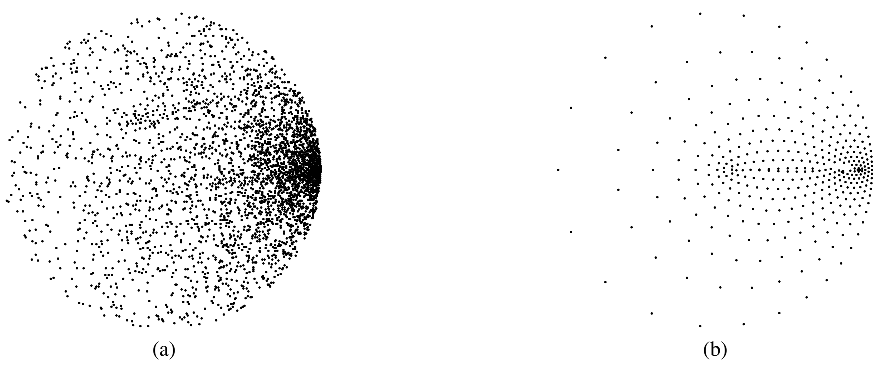
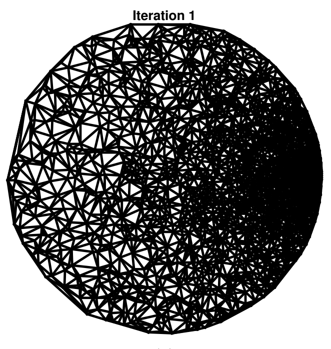


Figure 1: Node placement in the unit disc using: The rejection method, (a) and our algorithm, (b) with $h(x, y) = 0.2\sqrt{(x-1)^2 + y^2} + 0.04$.

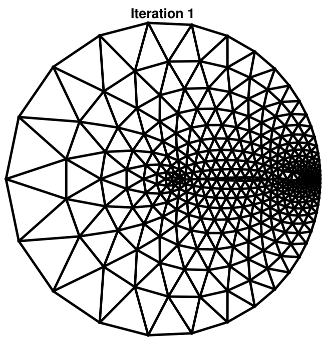
nodes.

Now, we will consider the unit disc domain with the same size function in order to compare the two algorithms in terms of the number of generated nodes.

From Table 1, we notice that our algorithm automatically generates an optimal number of nodes according to the size function h, unlike the Distmesh algorithm which can generate a very large or very small number of nodes depending on the step h_0 defined by the user.

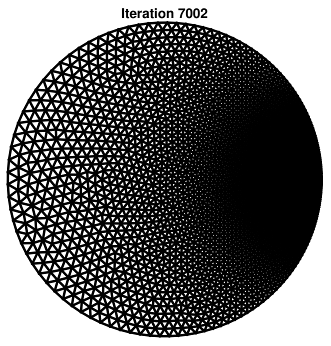


(a)

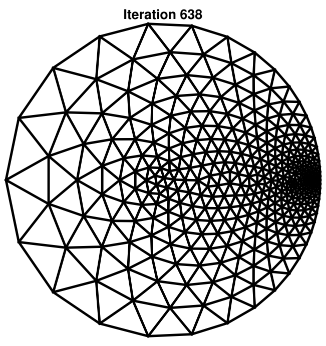


(b)

Figure 2: The initial triangulation of unit disc formed by connecting the nodes generated through both the rejection method, (a), and our algorithm, (b) using Delaunay triangulation.



(a)



(b)

Figure 3: Figures (a) and (b) show the final meshes obtained from the Distmesh algorithm and our algorithm, respectively, after applying mesh smoothing.

Table 1: Number of nodes required to create a unit disc mesh with the desired edge length function $h(x, y) = 0.2\sqrt{(x-1)^2 + y^2} + 0.04$ and different values of h_0 for Distmesh algorithm.

Distmesh algorithm: N [h_0]	8[0.2]	429[0.02]	17222[0.003]
Our algorithm: N	356	356	356

Next, we want to compare the robustness of the two algorithms using the same domain and mesh size function.

Table 2: Robustness of the Distmesh algorithm for the unit disc with $h(x, y) = 0.2\sqrt{(x-1)^2 + y^2} + 0.04$ and $h_0 = 0.01$.

Run	1	2	3	4	5	6
N	1631	1615	1599	1635	1582	1562
iter	4036	6898	9025	5920	3172	3991
q_{min}	0.7991	0.7383	0.8096	0.7631	0.6949	0.7216

According to Table 2, the Distmesh algorithm is not robust for the nonuniform mesh size function, as each execution with the same parameters produces meshes of different qualities and with different numbers of nodes. This quality, measured by q_{min} (i.e. q_{min} represents the minimum quality for all elements of the mesh), can decrease significantly. In addition, the parameter h_0 in Table 2 denotes the distance between points in the initial quasi-uniform distribution used by Distmesh in the first step. In [8], Jonas Koko attempted to modify the Distmesh algorithm to address the non-robustness issue, but the results obtained are not entirely robust. In contrast, our algorithm is robust for nonuniform mesh size function. Each execution with the same parameters produces the same mesh with good quality; see Table 3.

Table 3: Robustness of our algorithm for the unit disc with $h(x,y) = 0.2\sqrt{(x-1)^2+y^2}+0.04$.

Run	1	2	3	4	5	6
N	356	356	356	356	356	356
iter	638	638	638	638	638	638
q_{min}	0.8545	0.8545	0.8545	0.8545	0.8545	0.8545

We now want to compare the two algorithms in terms of the user-need requirement using a unit disc with the same function h.

Table 4: Error indicator of edge length in unit disc mesh with $h(x,y) = 0.2\sqrt{(x-1)^2+y^2}+0.04$ and different values of h_0 for Distmesh algorithm.

Distmesh: Er[h_0]	0.61[0.2]	0.22[0.08]	0.0441[0.03]	0.12[0.01]	0.15[0.003]
Our algorithm: Er	0.07	0.07	0.07	0.07	0.07

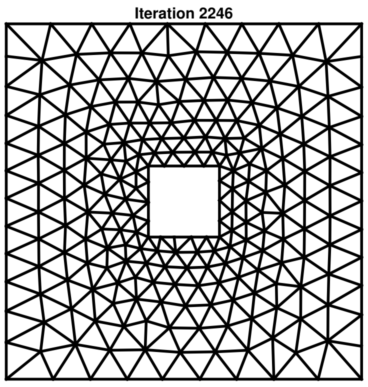
From Table 4, we can observe that the edge lengths obtained from our algorithm are remarkably close to the desired lengths, unlike the edge lengths obtained from the Distmesh algorithm, which depends on the step size h_0.

Now, we present several examples to provide a comprehensive view of our algorithm compared to the original Distmesh algorithm.

Square with hole

The size function using to mesh this domain is

$$h(x,y) = \min(0.2\sqrt{(x-0.4)^2+(y-0.4)^2}+0.05, 0.2\sqrt{(x-0.6)^2+(y-0.4)^2}+0.05,$$
$$0.2\sqrt{(x-0.6)^2+(y-0.6)^2}+0.05, 0.2\sqrt{(x-0.4)^2+(y-0.6)^2}+0.05).$$



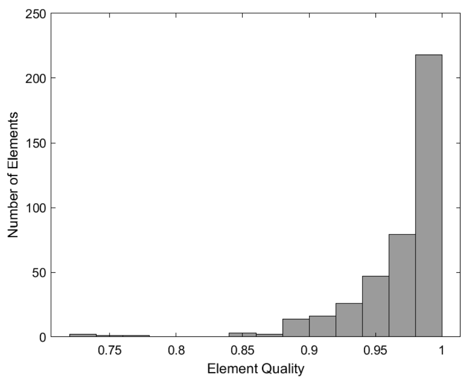
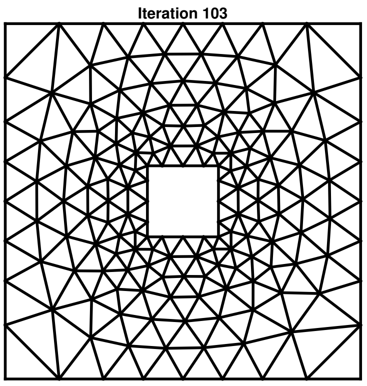


Figure 4: Mesh of square with hole and histogram of elements qualities with $q_{min} = 0.7292$ and h_0=0.04 using Distmesh algorithm.

Circle with hole

To create the mesh for this domain, we use the size function defined as

$$h(x,y) = 0.2\sqrt{x^2+y^2}+0.05.$$



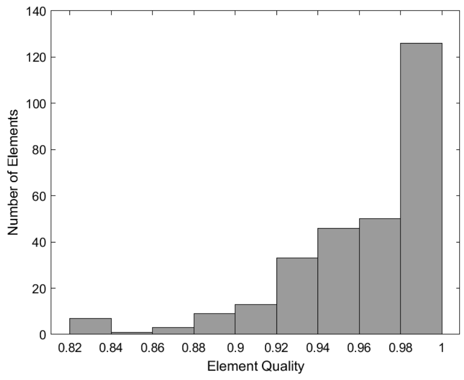
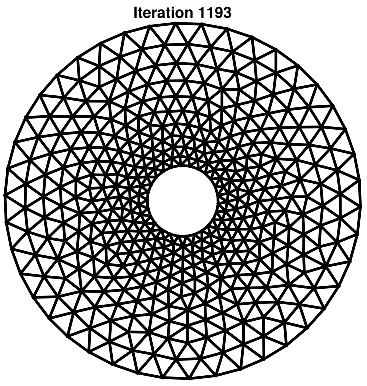


Figure 5: Mesh of square with hole and histogram of elements qualities with $q_{min} = 0,8284$ using our algorithm.

Table 5: Characteristics of a square with hole mesh obtained by two algorithms.

	Distmesh algorithm	Our algorithm
N	234	168
iter	2246	103
q_{min}	0.7292	0.8284
CPU (in sec)	13.0625	7.5938
Er	0.0214	0.0108



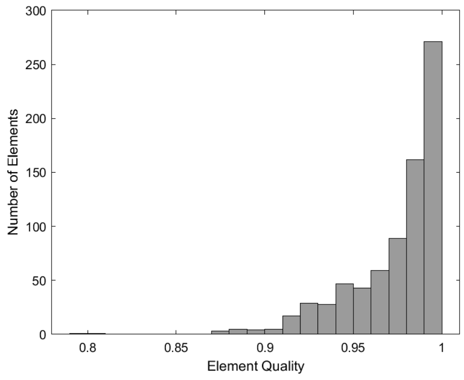


Figure 6: Mesh of circle with hole and histogram of elements qualities with $q_{min} = 0,7925$ and $h_0 = 0.05$ using Distmesh algorithm.

Table 6: Characteristics of a circle with hole mesh obtained by two algorithms.

	Distmesh algorithm	Our algorithm
N	416	205
iter	1193	105
q_{min}	0.7925	0.8223
CPU (in sec)	10.6881	8.0313
Er	0.0884	0.0343

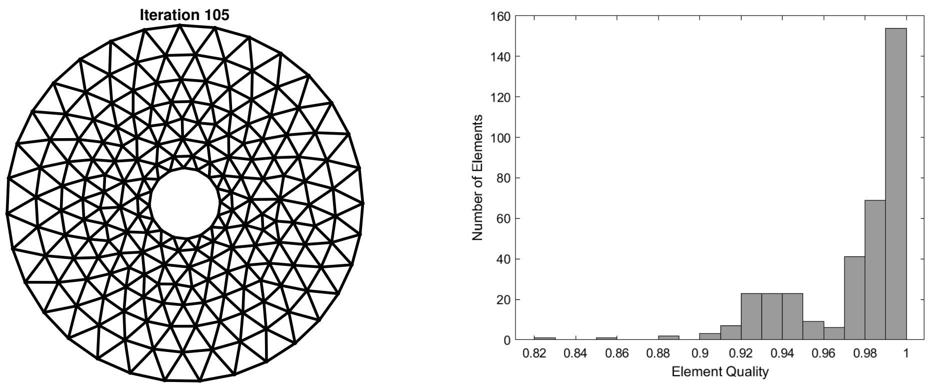


Figure 7: Mesh of circle with hole and histogram of elements qualities with $q_{min} = 0,8223$ using our algorithm.

Square with four holes

The size function used is

$$h(x,y) = \min(0.2\sqrt{(x-0.5)^2+(y-0.5)^2}+0.05, 0.2\sqrt{(x-0.5)^2+(y+0.5)^2}+0.05,$$
$$0.2\sqrt{(x+0.5)^2+(y+0.5)^2}+0.05, 0.2\sqrt{(x+0.5)^2+(y-0.5)^2}+0.05).$$

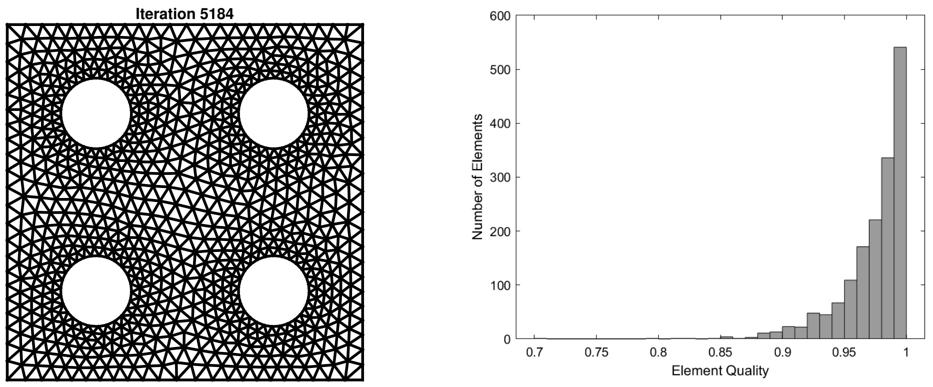
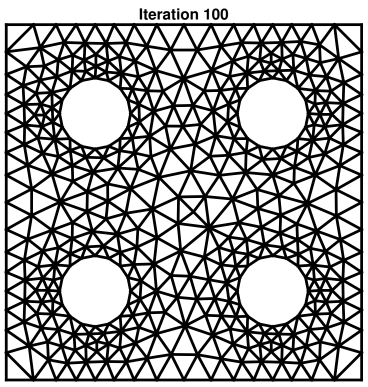


Figure 8: Mesh of square with four holes and histogram of elements qualities with $q_{min} = 0.7066$ and h_0=0.048 using Distmesh algorithm.

Table 7: Characteristics of a square with a four holes mesh obtained by two algorithms.

	Distmesh algorithm	Our algorithm
N	906	420
iter	5184	100
q_{min}	0.7066	0.7335
CPU (in sec)	20.7188	7.2969
Er	0.0601	0.022



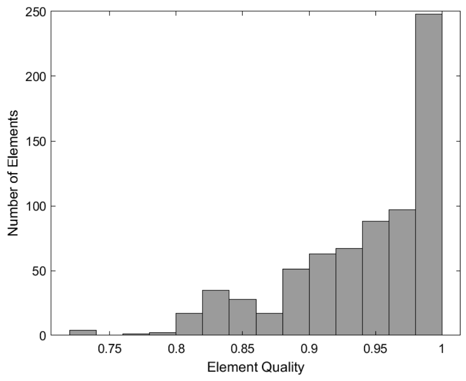
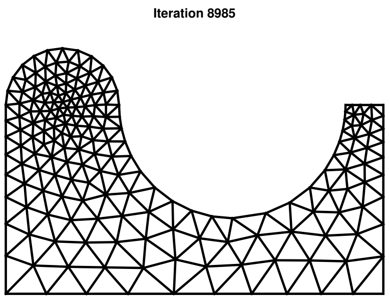


Figure 9: Mesh of square with four holes and histogram of elements qualities with $q_{min} = 0.7335$ using our algorithm.

Domain obtained by two circles and square

The size function used in this domain is

$$h(x, y) = min(0.2\sqrt{x^2 + (y-1)^2} + 0.05, 0.2\sqrt{(x-0.9)^2 + (y-1)^2} + 0.05,$$
$$0.2\sqrt{(x-1.5)^2 + (y-1)^2} + 0.05).$$



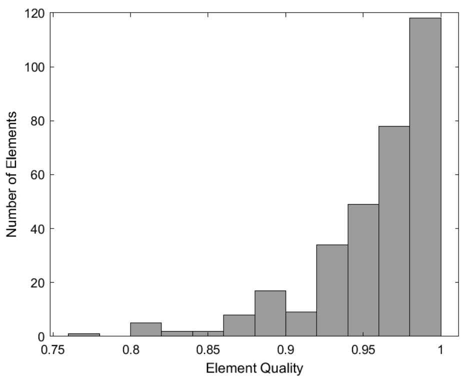


Figure 10: Mesh of domain obtained by two circles and square, and histogram of elements qualities with $q_{min} = 0.7612$ and $h_0 = 0.0086$ using Distmesh algorithm.

Table 8: Characteristics of a domain obtained by two circles and square mesh using the two algorithms.

	Distmesh algorithm	Our algorithm
N	194	180
iter	8985	158
q_{min}	0.7612	0.6466
CPU (in sec)	29.7602	7.9219
Er	0.0224	0.0285

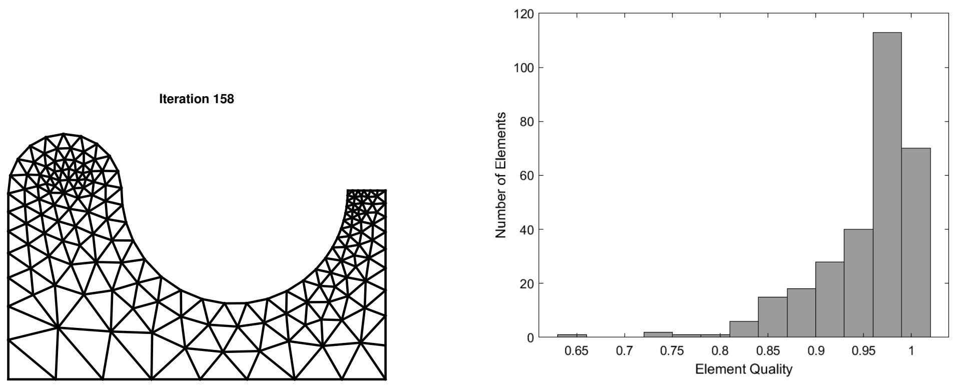


Figure 11: Mesh of domain obtained by two circles and square, and histogram of elements qualities with $q_{min} = 0.6466$ using our algorithm.

L-shaped domain

In order to create the mesh of the L-shaped domain, we use the size function defined as $h(x,y) = \frac{\sqrt{(x-1)^2+y^2}}{7} + 0.03$.

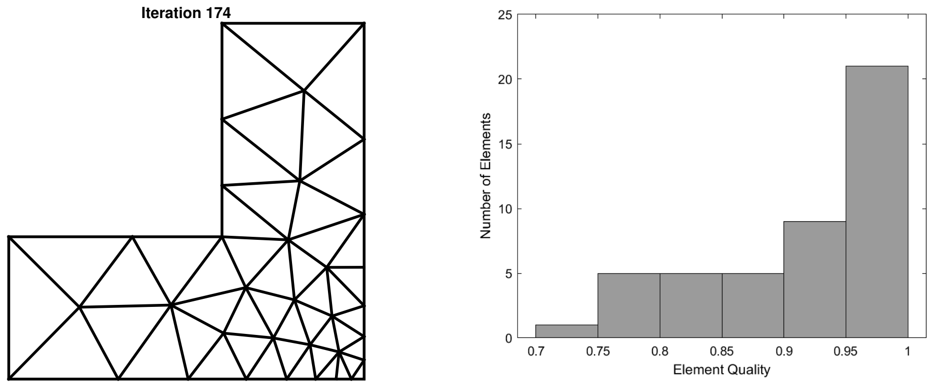


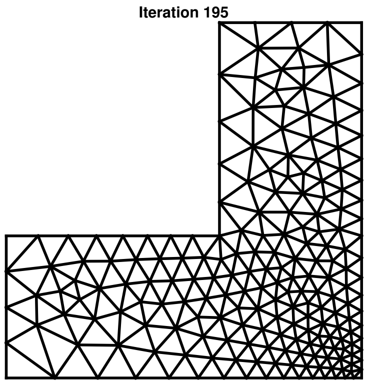
Figure 12: Mesh of an L-shaped domain and histogram of elements qualities with $q_{min} =$ 0.7285, $h(x,y) = \frac{\sqrt{(x-1)^2+y^2}}{7} + 0.03$ and h_0=0.061 using Distmesh algorithm.

Table 9: Characteristics of an L-shaped domain mesh using the two algorithms.

	Distmesh algorithm	Our algorithm
N	35	204
iter	174	195
q_{min}	0.7285	0.707
CPU (in sec)	4.8438	9.7813
Er	0.1536	0.0363

This comparison provides further evidence of the advantages of our proposed enhancements and demonstrates the competitiveness of the improved algorithm in the field of 2D mesh generation. Key findings from this comparison include the following:

Preserving or improving mesh quality: our strategy, based on the deterministic initial distribution of the nodes, globally preserves or improves the mesh quality with well-distributed nodes and well-shaped elements.



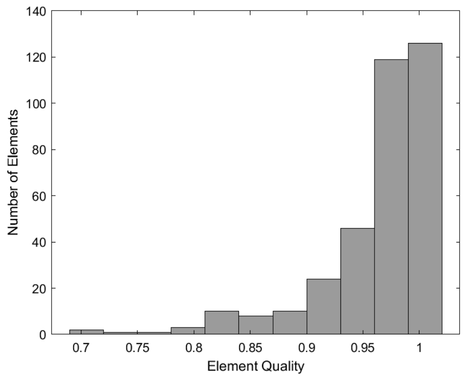


Figure 13: Mesh of an L-shaped domain and histogram of elements qualities with $q_{min} = 0,707$ using our algorithm.

Increased computational efficiency: the initial distribution of nodes in the domain can in the majority cases reduce the count of iterations required in the smoothing process. These techniques result in faster convergence and less CPU time, allowing the algorithm to handle larger and more complex meshes.

Enhanced robustness: the improved algorithm demonstrates a greater ability to generate high-quality meshes for a wide range of geometries.

Statisfying User-need requirement: the capability of the algorithm to produce meshes that satisfy the mesh size function h defined by the user at the initial stage, compared to the original Distmesh algorithm.

The results presented in this section showcase the superior performance of the improved Distmesh algorithm compared to the original algorithm. By addressing the limitations of the original Distmesh algorithm and implementing the proposed enhancements, we have successfully developed a more efficient, robust, and versatile tool for 2D mesh generation. In the next section, we conclude the study and discuss potential directions for future research.

6 Conclusions

In this study, we have presented a series of enhancements to the Distmesh algorithm for the generation of 2D meshes. Our proposed improvements have been shown to significantly enhance computational efficiency, user need requirement in mesh size, and robustness of the algorithm, as demonstrated through various numerical experiments. By comparing the improved Distmesh algorithm with the original version, we have highlighted the benefits of our proposed modifications.

The main contributions of our research include the development of a new strategy to generate the initial distribution of nodes in the domain, called Advancing Node Injection, and also the integration of an attractive force in the smoothing process. These improvements have led to a more efficient and versatile mesh generation tool that has the potential to benefit a wide range of scientific and engineering applications.

Looking ahead, there are several possible directions for future research in the area of mesh generation and improvement of the Distmesh algorithm. One potential avenue is to extend the improved Distmesh algorithm to three-dimensional (3D) mesh generation, which would enable its application to an even broader range of problems.

In conclusion, our proposed enhancements to the Distmesh algorithm represent a significant step forward in the field of 2D mesh generation. Addressing the limitations of the original algorithm and demonstrating the advantages of our improvements.

7 Appendix

In this appendix, we introduce the signed distance functions for various domains used in test cases, as listed in Table 10.

Table 10: Signed distance functions for various domains used in test cases.

Domain	Signed distance function
Unit disc	$d(x,y)=\sqrt{x^2+y^2}$
Square with hole	$d(x,y)=\max(-\min(x,1-x,y,1-y),$ $\min(x-0.4,0.6-x,y-0.4,0.6-y))$
Circle with hole	$d(x,y)=\max(\sqrt{x^2+y^2}-1,-\sqrt{x^2+y^2}+0.2)$
square with four holes	$d(x,y)=\max(-\min(x+1,1-x,y+1,1-y),$ $-\min(\sqrt{(x-0.5)^2+(y-0.5)^2}-0.2,$ $\sqrt{(x+0.5)^2+(y+0.5)^2}-0.2,$ $\sqrt{(x-0.5)^2+(y+0.5)^2}-0.2,$ $\sqrt{(x+0.5)^2+(y-0.5)^2}-0.2)$
Domain obtained by two circles and square	$d(x,y)=\max(\min(-\min(x+1,1-x,$ $y+1,1-y),\sqrt{x^2+(y-1)^2}-0.3),$ $-\sqrt{(x-0.9)^2+(y-1)^2}+0.6)$
L-shaped domain	$d(x,y)=\min(-\min(x,1-x,y,0.4-y),$ $-\min(x-0.6,1-x,y-0.4,1-y))$

References

[1] F. J. Bossen and P. S. Heckbert. A pliant method for anisotropic mesh generation. In *5th Intl. Meshing Roundtable*, **63**. Citeseer (1996).

[2] A. Bowyer. Computing dirichlet tessellations, *Comput. J*, **24(2)**, 162–166 (1981).

[3] J. R. Cebral, P. J. Yim, R. Löhner, O. Soto, and P. L. Choyke. Blood flow modeling in carotid arteries with computational fluid dynamics and mr imaging. *Acad. Radiol*, **9(11)**, 1286–1299 (2002).

[4] L. P. Chew. Guaranteed-quality triangular meshes. Technical report, CORNELL UNIV ITHACA NY DEPT OF COMPUTER SCIENCE (1989).

[5] D. A. Field. Laplacian smoothing and delaunay triangulations. *Commun. Appl. Numer. Methods*, **4(6)**, 709–712 (1988).

[6] D. A. Field. Qualitative measures for initial meshes. *Int. J. Numer. Methods. Eng*, **47(4)**, 887–906 (2000).

[7] P. L. George and H. Borouchaki. *Delaunay Triangulation and Meshing: Application to Finite Elements*. Hermès (1998).

[8] J. Koko. A matlab mesh generator for the two-dimensional finite element method. *Appl. Math. Comput.*, **250**, 650–664 (2015).

[9] S. Lo. A new mesh generation scheme for arbitrary planar domains. *Int. J. Numer. Methods. Eng.*, **21(8)**, 1403–1426 (1985).

[10] R. Löhner and P. Parikh. Generation of three-dimensional unstructured grids by the advancing-front method. *Int. J. Numer. Methods Fluids*, **8(10)**, 1135–1149 (1988).

[11] J. Peraire and P.O. Persson. High-order discontinuous galerkin methods for cfd. In *Adaptive high-order methods in computational fluid dynamics*, 119–152. World Scientific (2011).

[12] J. Peraire, M. Vahdati, K. Morgan, and O. C. Zienkiewicz. Adaptive remeshing for compressible flow computations. *J. Comput. Phys.*, **72(2)**, 449–466 (1987).

[13] P. O. Persson and G. Strang. A simple mesh generator in matlab. *SIAM Rev.*, **46(2)**, 329–345 (2004).

[14] J. Ruppert. A new and simple algorithm for quality 2-dimensional mesh generation. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, **93**, 83–92 (1993).

[15] K. Shimada and D. C. Gossard. Bubble mesh: automated triangular meshing of non-manifold geometry by sphere packing. In *Proceedings of the third ACM symposium on Solid modeling and applications*, 409–419 (1995).

[16] D. J. Tildesley and M. P. Allen. *Computer simulation of liquids*, Clarendon Press, Oxford (1987).

[17] Z. Wang, A. R. Srinivasa, J. Reddy, and A. Dubrowski. Flowmesher: An automatic unstructured mesh generation algorithm with applications from finite element analysis to medical simulations. *arXiv preprint arXiv:2103.05640* (2021).

[18] N. P. Weatherill. Delaunay triangulation in computational fluid dynamics. *Comput. Math. Appl.*, **24(5-6)**, 129–150 (1992).

[19] H. Zhang and A. V. Smirnov. Node placement for triangular mesh generation by monte carlo simulation. *Int. J. Numer. Methods Eng.*, **64(7)**, 973–989 (2005).

[20] A. Zheleznyakova and S. T. Surzhikov. Molecular dynamics-based unstructured grid generation method for aerodynamic applications. *Comput. Phys. Commun.*, **184(12)**, 2711–2727 (2013).

Author information

Mohamed Mrini, SMAD, FPL, Abdelmalek Essaadi University, Tetouan, Morocco.
E-mail: mohamed.mrini3@etu.uae.ac.ma

Amal Bergam, SMAD, FPL, Abdelmalek Essaadi University, Tetouan, Morocco.
E-mail: abergam@uae.ac.ma

Anouar El Harrak, MMA, FPL, Abdelmalek Essaadi University, Tetouan, Morocco.
E-mail: anouarelharrak1@gmail.com

Hatim Tayeq, SMAD, FPL, Abdelmalek Essaadi University, Tetouan, Morocco.
E-mail: tayeq.hatim@gmail.com