Received: 30 September 2021 | Revised: 3 April 2022 | Accepted: 7 May 2022
DOI: 10.1002/cnm.3615

APPLIED RESEARCH

WILEY

PIMesh: An automatic point cloud and unstructured mesh generation algorithm for meshless methods and finite element analysis–with applications in surgical simulations

Zhujiang Wang[1,2] | **Arun R. Srinivasa**[3] | **Junuthula N. Reddy**[3] | **Adam Dubrowski**[2]

[1]Mechanical Engineering and Robotics, Guangdong Technion Israel Institute of Technology, Shantou, China

[2]Faculty of Health Sciences, Ontario Tech University, Oshawa, Canada

[3]Department of Mechanical Engineering, Texas A&M University, College Station, Texas, USA

Correspondence
Junuthula N. Reddy, Department of Mechanical Engineering, Texas A&M University, College Station, TX, USA.
Email: jnreddy@tamu.edu

Abstract
We propose a point cloud and mesh generation algorithm, particle injection mesh generator (PIMesh), that can be used to generate optimized high-quality point clouds and unstructured meshes for domains in any shape with minimum (or even no) user intervention. The domains can be scanned images in OBJ format in 2D and 3D or just a line drawing in 2D. Mesh grading can also be easily controlled. The PIMesh is robust and easy to be implemented and is useful for a variety of applications, ranging from generating point clouds for meshless methods, mesh generation for finite element methods, computer graphics applications and surgical simulators. The core idea of the PIMesh is that a mesh domain is considered as an "airtight container" into which particles are "injected" at one or multiple selected interior points. The motion of the particles is controlled by a pseudo-molecular dynamics (PMD) formulation with a pairwise purely repelling "force" moderated by an absolute velocity dependent drag force. The particles repel each other and occupy the whole domain somewhat like blowing up a balloon. When the container is full of particles and the motion is stopped (the particles can be considered as a point cloud), a Delaunay triangulation algorithm is employed to link the particles together to generate an unstructured mesh. The performance of the PIMesh and the comparison with other unstructured mesh generation approaches are demonstrated through generating node distributions and meshes for several 2D and 3D object domains including a scanned image of bones and others.

KEYWORDS
Delaunay triangulation, finite element analysis, mesh generation, meshless methods, node distributions, point cloud, surgical simulations, unstructured mesh

1 | INTRODUCTION

Point cloud and mesh generation is a basic task for all discretization methods for the numerical solutions of physical problems and many computer graphics applications. Various mesh generation strategies have been deployed over the years[1–3] and have been successfully integrated into commercial packages and are in widespread use in the finite element and finite volume community. The meshes applied in numerical simulations are generally classified into two categories: structured and unstructured meshes. Since unstructured meshes provide better conformity to complex geometries than structured meshes,[4] unstructured meshes are more suitable to be used in surgical simulations where the geometries of mesh domains are typically very complex. However, the currently available mesh generators require quite a bit of user intervention in preparing the object for meshing, locating highly distorted meshes and "healing" errors in meshing, and so on. Because of these challenges, mesh generation remains a technically challenging task.

In particular, the challenges in mesh generation are particularly acute in the area of surgical simulations for training students and professionals in surgical interventions. Real-time rendering technology and haptic devices based on high-fidelity surgical simulators have been adopted by the surgical community. However, once surgical simulators are developed, the training scenarios are rarely updated by healthcare educators due to the complexity of re-meshing and computational simulations. Thus, allowing medical educators to set new simulation scenarios directly is extremely valuable, because even for the same type of surgery there can be many different training scenarios since human organs and tissues change with ages and patients can have many different symptoms. For example, the procedures of surgeries to cure stomach cancers can be different according to the geometries as well as the positions of tumors in stomachs and patients' age. However, it is not possible to develop surgical simulators by engineers that cover all the practical scenarios due to the limit of funding resources and an unlimited number of cases in reality.

A prospective technical plan is to develop a surgical simulator that an educator can adjust the existing training scenarios or even create new training scenarios without the need for expertise in numerical simulations. To develop such a simulator, it is essential to integrate an automatic mesh generator to generate meshes for objects on which learners practice. Unstructured meshes are widely adopted in surgical simulation.[5] Furthermore, compared to the autonomous structured meshes generation algorithms, autonomous unstructured meshes algorithms are generally simpler.[6] This work aims to develop a fast simple autonomous unstructured mesh generator that can easily be integrated into surgical simulators, which allows healthcare educators to adjust and create training scenarios.

Autonomous unstructured mesh generation methods is an active research subject in recent decades. Delaunay triangulation-based methods and its variations,[7–9] advancing front methods,[10,11] Octree-based methods,[12,13] and their hybrid methods[14,15] are the most popular mesh generation methods. However, these geometrically based methods, particularly extending the methods from 2D domain to 3D domain, require extensive mathematical descriptions of the objects and so cannot be directly used from scans and OBJ files.

Besides the above geometrically based mesh generation methods, there are some physically based mesh generators, such as the bubble mesh.[16] The bubble mesh is an approach that is based on sphere packing, that is, it considers each node to be a solid sphere and packs them inside the mesh domain one by one using attractive forces to get the bubbles to "stick" together. However, it requires a good initial bubble configuration to reduce the convergence in the relaxation stage. The attractive forces tend to "clump" the mesh points and care has to be taken to eliminate that.

Smoothed-particle hydrodynamics (SPH)[17] based mesh generation algorithms[18,19] are recent methods to generate unstructured meshes. They are based on a level set description of a surface and require a background multi-resolution Cartesian mesh assigning boundary points, seeding it with interior points, and then improving on their locations using the equations for fluid flow. The approach requires assigning mesh nodes on boundary edges or surfaces of mesh domains and using temporal ghost particles at the boundaries, both of which can complicate the implementation of the mesh generation methods or even reduce the efficiency.

Apart from these, there are other mesh generators, such as DistMesh[20] and Gmsh,[21] which are very popular in the finite element methods community. DistMesh is also a physically based mesh generator; however, it requires specification of distance functions for bounding surfaces which are very challenging for complex domains. Additionally, Gmsh uses a local refinement strategy starting from the Delaunay triangulation of the boundary points and then adding new points as required, and requires considerable user interventions.

There has also been a considerable effort focused on the fast triangulation of domains using techniques such as optimal Delaunay triangulation[22] or centroidal Voronoi tessellation.[23] These techniques have seen further impetus due to applications of discrete differential geometry which requires both the original mesh (the triangles or tetrahedral obtained as a result of a Delaunay triangulation for instance) as well as its dual (the Voronoi cell polygons or polyhedral for examples). Approaches such as Hodge-optimized triangulation[24] that preserve the quality of both the mesh and its dual have been developed. In the last decade, approaches that discretize the physical problem using both the primal and dual meshes have increased[25] with geometrical and physical quantities defined on either the primal or dual mesh. Most recently dual mesh approach to finite elements has been pioneered by Reddy and coworkers.[26,27]

In many of these approaches, the mesh generation proceeds in two steps: (1) allocation of points on the boundary and inside a domain and (2) creation of connections or edges between the points in such a way as to give rise to a valid non-intersecting and high-quality mesh. The aforementioned methods all focus on the second task, that is, how to create a mesh given a point cloud[28] and how to adjust the location of the point cloud to improve the mesh quality. Usually, the second step involves a complex geometry-based cost function with multiple minima[29] so that viable solutions depend upon good starting locations for the nodes.

Recently with the increased use of finite element methods to quickly evaluate preliminary designs, there has also considerable interest in generating quick meshes from 2D sketches. There is thus a need for an easy to use mesh generator that

- Does not require an extensive mathematical description of the domain so that it can even generate meshes from a computer sketch;
- Provides a simple and automated way to estimate the number of points required and insert them.
- Provides a high-quality mesh without additional smoothing steps;
- Allows for a graded mesh with higher density where required;
- Allows for particle movement and the injection or removal of points for adaptive mesh generation.

The mesh generation scheme presented here (see Figure 1,2), the PIMesh (Particle Injection Mesh Generator), which is based on a simplified version of molecular dynamics, satisfies all these criteria. The PIMesh mimics a "gas expansion" or "balloon inflation" process (see Figure 3) using a purely repelling "forces" on the particles.

Given a computer sketched region and one or more "injection" points in the interior, the algorithm "injects" particles that repel each other and so occupy the whole domain. The repulsion can be location dependent so that graded meshes are possible. The simulation of the particles' movement is based on Newton's laws with pairwise repulsive forces and velocity-based drag forces constraints that allow particles to dissipate their kinetic energy. Rather than use physically realistic forces, such as van der Waals forces, the approach (a) uses purely repelling simplified forces that allow for rapid computation and (b) uses a velocity limiting scheme so that the particle velocities do not exceed a threshold.

The particles are prevented from crossing any boundary by repulsion from them so that they eventually occupy the region assigned to them with maximally separate distances. If new particles are injected, they will simply readjust based on the repulsive forces. If the boundaries move they will exert repulsive "forces" on the particles which will consequently readjust. With this approach, it is shown that we obtain excellent mesh quality with minimal or NO user intervention.

The major innovations of the PIMesh are

- The repelling forces of the pseudo-molecular dynamic (PMD) drive the particles' motion. Even when the particles' motion is stopped, there are still repelling forces among neighbor particles. Therefore, the particles are always trying to occupy all the space of an object domain and the PIMesh can be used to generate point clouds and meshes for domains with any shape.

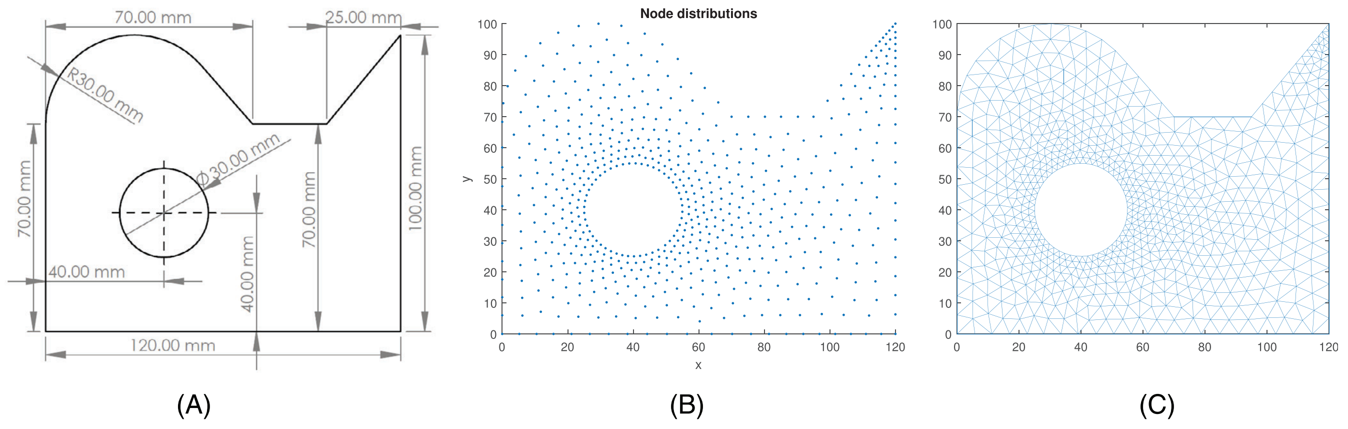

FIGURE 1 Illustration of the capability of the PIMesh to generate graded node distributions for meshless methods and triangular mesh for finite element analysis from CAD drawings. (A) The geometry of a 2D domain. (B) Nonuniform node distributions and (C) a nonuniform mesh for the 2D domain.

- The PIMesh does not require an initial high-quality mesh nodes configuration, which is required in other physically based mesh generation methods, such as the bubble mesh[16] and the SPH based mesh generation methods.[18,19] We can even inject all the nodes from a single point in the domain (or even on the boundary) and the particles will repel each other and occupy the domain.

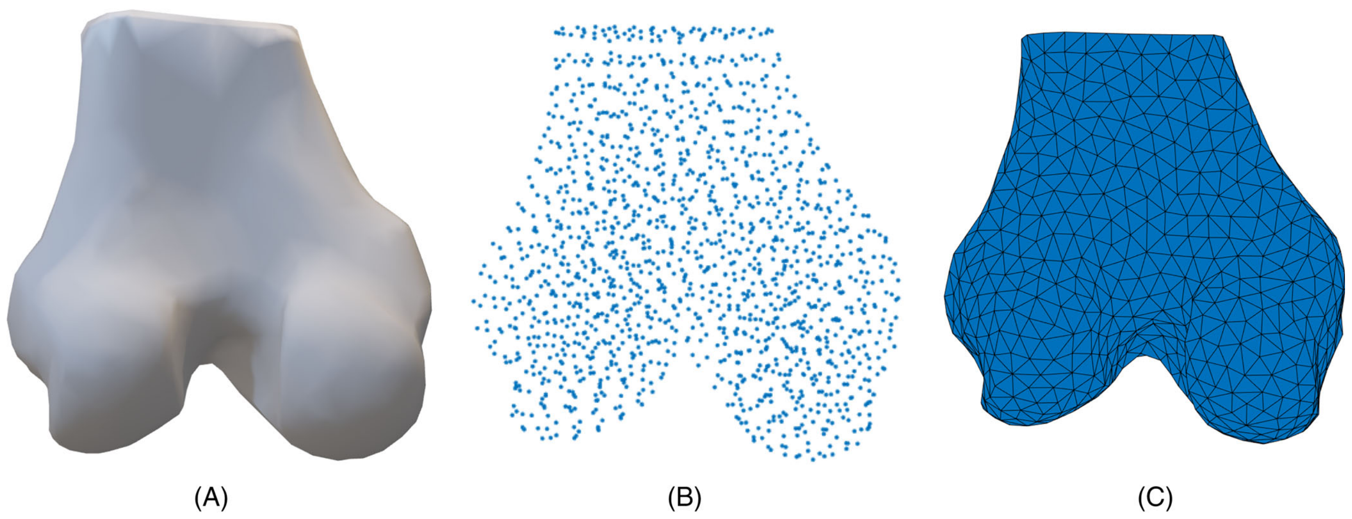

FIGURE 2 Illustration of the efficacy of the PIMesh for surgical simulation applications. (A) A 3D femur model (OBJ format) obtained through CT scan images. (B) Uniform node distributions and (C) a uniform tetrahedral mesh for the 3D femur domain. Note that the node distribution includes nodes inside the mesh domain; the mesh is generated based on the uniform node distribution and the nodes on the surface of the mesh domain are distributed almost uniformly.

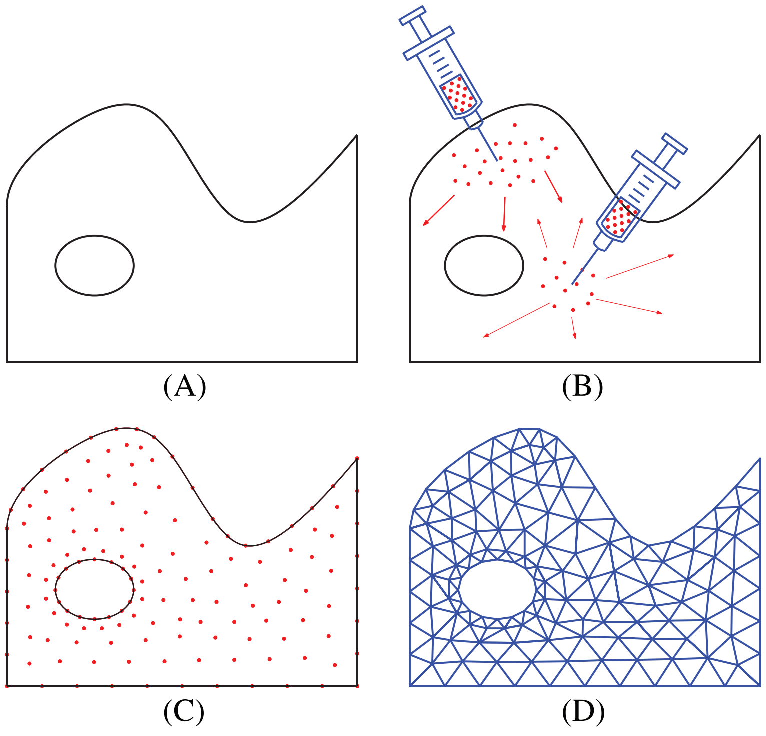

FIGURE 3 (A) The plot of a 2D object represents the mesh domain, which is assumed to be an airtight container. (B) Mesh nodes are assumed to be particles, which are injected at multiple places and can move inside the container. (C) The particles are distributed over the container until the motion is stopped (the speed of the particles is very small). The particles can be considered as a point cloud. (D) Generate a mesh based on the positions of the particles using Delaunay triangulation (or other schemes)

Since the bubble mesh has attractive forces, the initial particle distribution should be distributed to prevent clumping to speed up the convergence (see Section 3.1 for the detailed comparison between the PIMesh and the bubble mesh). The SPH based mesh generation methods require high-quality boundary nodes distributions to guarantee the quality of the resulting meshes. Assigning mesh nodes on boundary edges for 2D mesh domains and boundary surfaces for 3D mesh domains essentially are tasks for generating meshes for 2D curves and 3D surfaces respectively.

- Compared to SPH based mesh generation methods,[18,19] in which the particles' flow is driven by the gradient of pressure, the PIMesh is much easier and at least an order of magnitude faster to implement, because (1) the calculation of repelling and viscous forces in this work is only based on pairwise repellent functions and particle absolute velocities without the need for calculating the density as well as the gradient of pressure and velocities; (2) PIMesh does not need to calculate ghost boundary mesh nodes, which take extra memories and requires extra computation; (3) PIMesh has a dynamic population control algorithm and thus always simulate the motion of a minimum number of particles (see Section 3.2 for the detailed advantages of PIMesh over SPH based mesh generation methods).
- The velocity dependent drag force and the use of a cutoff speed guarantees that the motion simulation always is converged since the viscous forces dissipate the kinetic energy of particles.
- The algorithm can easily handle fixed mesh node constraints or complex interior point constraints;
- The number of particles is automatically updated according to a particle population control algorithm based on the target mesh size (or the target neighbor node distance for point clouds); therefore, the PIMesh does not require a predetermined number of nodes.

As the unstructured mesh generation is based on point clouds, we first discuss the algorithm to obtain admissible point cloud distributions for meshless methods.

2 | ADMISSIBLE NODE DISTRIBUTIONS FOR MESHLESS METHODS

In practice, most node distributions are obtained through meshes by removing the nodal connections. Recently, several works[30–32] have been proposed to generate node distributions for meshless methods. The PIMesh proposed in this work is an alternative method to create high-quality node distributions. We first introduce the algorithm for generating uniform node distributions, and the overview of the algorithm is shown in Table 1. The details are discussed in the following part.

2.1 | Initialization for obtaining uniform node distributions

At the initial step, 2D and 3D domains are prepared in the OBJ file format. The simulation parameters, such as the total simulation time T_{total}, the time step size Δt, and mesh size function over the domain $h(\mathbf{x})$, are set. Since the node distributions is uniform, the target node distance function is set as $h(\mathbf{x}) = h$ (h is a constant).

2.1.1 | Estimate the target number of particles

For a 2D domain represented by a triangular mesh in OBJ file format, we assume that particles are uniform array points distributed over the 2D domain. Thus, the initial target number of particles that will be injected into the domain can be estimated as,

$$N_{total} = \frac{A_{2d}}{h^2}, \tag{1}$$

where A_{2d} is the total area of the domain. Similarly, for a 3D domain represented by a triangular surface mesh in OBJ file format, the initial target number of particles is estimated as,

$$N_{total} = \frac{V_{3d}}{h^3}, \tag{2}$$

TABLE 1 The overview of the PIMesh to obtain uniform node distributions

```
1   Initialization
    1.1  Prepare a domain composed of pure triangles in OBJ format.
    1.2  Set simulation parameters: the target node distance function $h(x) = h$, total simulation time $T_{total}$, time step size $\Delta t$, and
           $N_{status} = False$.
    1.3  Add extra vertices on the domain boundaries to ensure that particles are inside the domain.
    1.4  Calculate the initial target number of particles $N_{total}$.
    1.5  Set injection positions $\mathbf{S}$ where particles will be injected.
    1.6  (Optional) Set fixed particles (nodes).
2   While $t < T_{total}$ (run motion simulation):
    2.1  If $N_p < N_{total}$:
         Generate new particles at the injection positions $\mathbf{S}$.
         Else if $N_p > N_{total}$:
         Remove extra particles.
    2.3  If multiple particles overlap at the same position, keep one and remove extra particles.
    2.4  Update the particles positions according to Equation (3).
    2.5  If a particle move outside of the domain, project the particle onto the boundary of the domain.
    2.6  Calculate the average distance $\Delta d_{avg}$ and that the particles travel and the maximum distance $\Delta d_{max}$ that a particle travel at
           the current time step.
    2.7  If $\Delta d_{avg}$ <0.005 $h$ & $N_p = N_{total}$ & $N_{status} = False$:
         Update $N_{total}$ and set $N_{status} = True$
    2.8  If $\Delta d_{avg}$ >0.006 $h$ or $e_{avg}$ >0.02:
         Set $N_{status} = False$
    2.9  If $\Delta d_{avg}$ <0.005 $h$ & $\Delta d_{max} - \Delta d_{avg}$ <0.02 $h$ & $N_p = N_{total}$:
         Terminate the motion simulation
         If $\Delta d_{avg}$ <0.001 $h$ & $N_p = N_{total}$:
         Terminate the motion simulation
3   The particles distributions are the node distributions (point cloud).
```

where V_{3d} is the volume of the 3D domain. Note that the estimated number of particles is underestimated for both 2D and 3D domains at the initialization step, more particles will be injected during the motion simulation.

2.1.2 | Initialize the particles' injection positions

The essential feature of the approach (and one that makes it different than other approaches in the literature), is the fact that rather than pre-distributing particles (or mesh nodes) which is a complicated task, we just select a few points (it could be as low as one) in the interior to inject particles and let the repulsion between the particles force them to distribute themselves throughout a domain. The injection positions $\mathbf{S}$, where the particles will be injected, can either be manually set by users or be calculated automatically based on the geometry of the domain and mesh size function. An algorithm to obtain the node injection positions automatically is discussed in Appendix A for 2D domains and Appendix B for 3D domains.

2.1.3 | Set fixed mesh nodes

The PIMesh can easily handle the constraints of fixed mesh nodes by specifying the positions of fixed particles. If N_f fixed mesh nodes are set at the positions $\mathbf{x}_f = \{\mathbf{x}_{f1}, \mathbf{x}_{f2}, \cdots, \mathbf{x}_{fN_f}\}$, particles (speeds are set as zeros) are immediately injected at these positions before the simulation of the particles' motion. The method to handle the fixed particles is very simple and straightforward, and is discussed in Section 2.2.1.

Up to now, the initialization is completed and the simulation of the particles' motion is introduced as follows.

2.2 | Simulation of the particles' motion

At the beginning of the particles' motion simulation, we compare the current number of particles N_p and the target number of particles T_{total}.

- If $N_p < T_{total}$: new particles are injected at the injection positions $\mathbf{S}$. For each time step, only one particle is allowed to be injected at one injection position. Therefore, the maximum number of particles that can be injected at each time step is N_s.
- If $N_p > T_{total}$: the recently injected $N_p - T_{total}$ particles are removed at the current time step.

The direction of the initial velocity of a particle is chosen at random so that the particles are distributed throughout the domain with a very low probability of collision. The overlap between the particles is then checked. If multiple particles are overlapped at one location, extra particles are removed and only one particle is kept at this location.

2.2.1 | Pseudo-molecular dynamics

In this work, the simulation of the particles' motion is based on a molecular dynamics-like formulation with a pairwise repelling force moderated by an absolute velocity dependent drag force. Since these do not have to represent any specific physical system, the repelling forces and the drag forces are chosen to be (a) computationally as efficient as possible (b) prevent excessive velocity build up (i.e., the velocity of the particles are capped to a given maximum so that particles do not drift out of control). The PMD formulation is simply based on Newton's second law,

$$m\underset{i}{\ddot{x}}(t) = F_{fi}(x(t)) + F_{vi}\left(\dot{x}\,(t)\right), \tag{3}$$

where $\mathbf{x}_i$ is the position of the i_{th} particle p_i; the mass m of a particle is a constant; $\mathbf{F}_{fi}$ is the repelling force applied on p_i,

$$F_{fi} = k_s \sum W\left(\frac{||x_i - x_j||}{h}\right) \frac{x_i - x_j}{||x_i - x_j||}. \tag{4}$$

$\mathbf{x}_j$ are the positions of the p_i's neighbor particles; the kernel function $W(q)$

$$W(q) = \alpha \begin{cases} (2-q)^3 - 4(1-q)^3 & 0 \le q < 1 \\ (2-q)^3 & 1 \le q < 2, \\ 0 & q \ge 2 \end{cases} \tag{5}$$

is a modification of the kernel function in the work.[33] The kernel width is set as the target mesh size h and therefore $q = \frac{||\mathbf{x}_i - \mathbf{x}_j||}{h}$; α is set as $\alpha = \frac{1}{6}$ for 2D case and $\alpha = \frac{1}{18}$ for 3D case. Once the distance between two particles at $\mathbf{x}_i$ and $\mathbf{x}_j$ is smaller than 2 h, a repelling force is generated between these two particles. The viscous force $\mathbf{F}_{vi}$ is set as

$$F_{vi} = -k_v \frac{m\underset{i}{\dot{x}}}{\Delta t}, \tag{6}$$

to stabilize the motion, where k_v is a constant ($0 < k_v < 1$) and Δt is the size of the time step.

The numerical simulation of the motion is based on a Euler method. During each time step, the distance that a particle has traveled in the recent time step is

$$\Delta d_i = \left\| x_i(t+\Delta t) - x_i(t) \right\|. \tag{7}$$

The average distance that the particles have traveled is,

$$\Delta d_{avg} = \frac{1}{N_p} \sum_{i=0}^{N_p} \Delta d_i. \tag{8}$$

It is worth noting that if there is no repelling force applied on the i_{th} particle ($\mathbf{F}_{fi} = 0$), the viscous force (6) can reduce the velocity of the i_{th} particle from $\underset{i}{\dot{\mathbf{x}}}$ to $(1-k_v)\underset{i}{\dot{\mathbf{x}}}$, which indicates $\underset{i}{\dot{\mathbf{x}}}(t+\Delta t) = (1-k_v)\underset{i}{\dot{\mathbf{x}}}(t)$. The viscous forces always try to dissipate the kinetic energy of the particles as long as the viscous constant $0 < k_v \leq 1$. The bigger k_v is set, the faster the motion is stopped. Therefore, the particle's momentum rate based vicious forces $\mathbf{F}_{vi}$ allow us to adjust the viscous forces dynamically to terminate the simulation quickly. In this work, to get the simulation converged quickly, we update the viscous constant k_v according to the following scheme:

- If $N_p < N_{total}$ or $\Delta d_{avg} > 0.1\ h$, we set the viscous constant as $k_v = 0.5$.
- If $N_p = N_{total}$ and $0.02\ h < \Delta d_{avg} < 0.1\ h$, the viscous constant is set as $k_v = 0.05$ so that the particles can quickly reach all the space of the domain.
- If $N_p = N_{total}$ and $\Delta d_{avg} < 0.02\ h$, we assume that the motion of the particles is almost stopped and the particles are almost uniformly distributed among the domain. To stop the particles' motion simulation quickly, we increase the viscous constant gradually using the following equation,

$$\begin{array}{c} \overline{k}_v = 0.05 + \left(t_i - t_i^{0.02}\right) \cdot 0.001 \\ k_v = \min\left(\overline{k}_v, 0.5\right) \end{array}, \tag{9}$$

where t_i is the current time step number and $t_i^{0.02}$ is the recent time step number when the $\Delta d_{avg} < 0.02\,h$.

In this way, we can stabilize the motion and obtain the particle distributions quickly. The detailed convergence analysis is discussed in Section 2.4. The PMD is the core innovation and the fundamental of the work, which differs from other physically based mesh generation algorithms (see Sections 3.1 and 3.2 for the detailed advantages over physically based mesh generation algorithms, including the SPH based mesh generation methods[18,19]).

2.2.2 | Handle the fixed mesh nodes constraints

To handle the constraints of fixed mesh nodes, we simply set the repelling forces and viscous forces applied on the particles at the positions of

$$x_f = \left\{x_{f1}, x_{f2}, \cdots, x_{fN_f}\right\},$$

as zeros in the governing Equation (3). Then the accelerations of the fixed particles are equal to zero and so the positions of the fixed particles are never updated.

2.2.3 | Project particles that are outside domains

When a particle p_i moves outside the mesh domain (see Appendix C and D for the method to determine the location of a particle), we simply project the particle back to the mesh boundary edge/surface, and the particle's updated position is the projection point p_i', which is the nearest point on the boundary to the particle p_i; the updated velocity is

$$\dot{\mathbf{x}}_{pi}^{new} = \dot{\mathbf{x}}_{pi} - 2\left(\dot{\mathbf{x}}_{pi} \cdot \mathbf{n}_{p_i'}\right)\mathbf{n}_{p_i'}, \tag{10}$$

where $\dot{\mathbf{x}}_{pi}$ is the velocity of the particle p_i before projection; $\mathbf{n}_{p_i'}$ is the normal vector of the domain boundary where the projection point p_i' locates. Note that the particle's updated speed remains the same $\left(\left|\dot{\mathbf{x}}_{pi}^{new}\right| = \left|\dot{\mathbf{x}}_{pi}\right|\right)$, and the direction of the updated particle's velocity $\dot{\mathbf{x}}_{pi}^{new}$ is changed as if the particle is bounced back from a wall.

2.2.4 | Update the target number of particles based on a PID controller

If three criteria (1) $\frac{\Delta d_{avg}}{h} < 5\%$ (the motion is considered to be relatively slow), (2) $N_p = N_{total}$ (the current number of the particles N_p is equal to the target number N_{total}), and (3) $N_{status} = False$ (if $\frac{\Delta d_{avg}}{h} > 6\%$, set $N_{status} = False$) are satisfied, the particles are assumed to move slowly enough for estimating the target number of particles N_{total} and N_{status} is set as $N_{status} = True$.

To update N_{total}, the average node distance error e_{avg} is obtained as

$$e_{avg} = \frac{1}{N_p}\sum_{i=1}^{N_p}\frac{L_i - h}{h}; \quad L_i = \frac{1}{N_{nbr}}\sum \left|\left|x_j - x_i\right|\right|, \tag{11}$$

where L_i the average of the distances from the i_{th} particle to its nearest N_{nbr} particles (for 2D case, $N_{nbr} = 3$; for 3D case, $N_{nbr} = 6$ in this work). If average distance error e_{avg} is greater than 0.02 ($|e_{avg}| > 0.02$), the target number of the particles N_{total} is updated through a digital implementation of a PID controller,

$$\begin{aligned} \overline{u}(t+\Delta t) &= k_P e_{avg} + k_I e_t + k_D e_d \\ e_t &= e_{avg}(t+\Delta t) + e_{avg}(t) \\ e_d &= e_{avg}(t+\Delta t) - e_{avg}(t) \\ u(t+\Delta t) &= \min(\overline{u}(t+\Delta t), 1.0) \\ N_{total}(t+\Delta t) &= \lceil N_{total}(t)[1 + u(t+\Delta t)]\rceil \end{aligned}, \tag{12}$$

where the parameters of the PID controller are set as $k_p = 0.5$, $k_i = 0.05$, $k_d = 0.1$ to avoid overshoot in the number of the particles. To further avoid injecting too many particles at a time, the change in the number of the particles is limited to $N_{total}(t)$.

If $N_p(t) < N_{total}(t + \Delta t)$, more particles will be added; if $N_p(t) > N_{total}(t + \Delta t)$, extra particles will be removed. New particles will be injected at the locations where the particles are sparse. The updated particles injection positions can be obtained thought the algorithm shown in Table 2.

2.2.5 | Terminate the motion simulation

Define the maximum distance Δd_{max} that a particle traveled during the recent time step as,

TABLE 2 The algorithm to calculate the updated particle injection positions for a 2D domain

1	Find each particle's average distance L_i (see Equation (11)).
2	Find the $N_{new} = N_{total}(t + \Delta t) - N_{total}(t)$ largest average distance.
3	For $i = 1, 2, \cdots, N_{new}$: $\mathbf{S}_i = (\mathbf{x}_k + \mathbf{x}_l + \mathbf{x}_m + \mathbf{x}_n)/4$ where $\mathbf{x}_k$ is the i_{th} particle of particles with the N_{new} largest average distances. $\mathbf{x}_l$, $\mathbf{x}_m$, $\mathbf{x}_n$ are the positions of three nearest neighbor particles of $\mathbf{x}_k$.
4	$\mathbf{S} = \{\mathbf{S}_1, \mathbf{S}_2, \cdots, \mathbf{S}_{N_{new}}\}$ are the new injection positions

$$\Delta d_{max} = \max\{\Delta d_0, \Delta d_1, \cdots, \Delta d_{N_p}\}. \tag{13}$$

If one of the two criteria—(1) $\frac{\Delta d_{avg}}{h} < 0.1\%$ and $N_p = N_{total}$ (2) $\frac{\Delta d_{max} - \Delta d_{avg}}{h} < 2.0\%$, $\frac{\Delta d_{avg}}{h} < 0.5\%$, and $N_p = N_{total}$—is satisfied, the motion is assumed to be stopped. Therefore, the motion simulation is terminated. The particle distributions are the node distributions for meshless methods.

2.2.6 | Speed up the motion simulation

A fast collision detection (FCD) technique using uniform cells[34] is employed to speed up the search of a particle's neighbor elements, such as other particles, boundary vertices, boundary edges, and boundary surfaces (for 3D mesh generation). The size of the uniform cell is set as 2 h.

To smooth the motion simulation, the maximum speed of particles is limited to $0.4r/\Delta t$,[35] where r is the width of the kernel function (5) and is set equal to the uniform cell size of the FCD technique ($r = 2\ h$) in this work. This indicates that a particle cannot travel through two uniform cells during each time step. Therefore, whenever a particle moves across the boundaries, there must be at least a boundary element, such as a vertex, an edge, or a triangle surface, occupying the particle's neighbor cells. If we add enough extra boundary vertices on the boundary (see Appendix C for the details of adding extra boundary vertices for 2D and 3D domains), We can detect whether a particle is inside the boundary of the mesh domain by searching for the boundary vertices in the particle's neighbor cells (see Appendix D for the details).

2.3 | Nonuniform node distributions

The nonuniform node distributions generation algorithm is similar to the uniform node distributions algorithm shown in Table 1. The main differences exist in estimating the target number of particles, constructing the node distance function, and updating the target number of particles.

2.3.1 | Estimate the target number of particles

Similar to the uniform case, the initial target number of particles for 2D nonuniform node distributions is estimated as

$$N_{total} = \frac{A_{2d}}{h_{max}^2}, \tag{14}$$

where A_{2d} is the total area of the 2D domain; h_{max} is the maximum target node distance. For 3D case, the initial target number of particles is estimated as

$$N_{total} = \frac{V_{3d}}{h_{max}^3}, \tag{15}$$

where V_{3d} is the volume of the 3D domain.

2.3.2 | Construct nodes distance function

For nonuniform node distributions, the node distance function $h(\mathbf{x})$ can be explicitly defined as $h(\mathbf{x}) = h_c(\mathbf{x})$ (see equation Equation [20] as an example) or constructed through a discrete node distance function as $h(\mathbf{x}) = h_d(\mathbf{x})$ (see the details in Appendix E). With the node distance function, the repelling force applied on the i_{th} particle is updated as

$$F_{fi} = k_s \sum_{j=1}^{N_i} W\left(\frac{\left\|x_i - x_j\right\|}{h(x_i, x_j)}\right) \frac{x_i - x_j}{\left\|x_i - x_j\right\|}, \tag{16}$$

where

$$h(x_i, x_j) = \frac{h(x_i) + h(x_j)}{2}. \tag{17}$$

2.3.3 | Update the target number of particles

Similar to the uniform case, the updated target number of particles is calculated according to Equation (12), and the average node distance error of a nonuniform node distributions is obtained as,

$$\begin{aligned} e_{avg} &= \frac{1}{N_p} \sum_{i=1}^{N_p} e_i \\ e_i &= \frac{1}{N_{nbr}} \sum \frac{\left\|x_j - x_i\right\| - h(x_i, x_j)}{h(x_i, x_j)} \end{aligned}, \tag{18}$$

where $\mathbf{x}_j$ are the N_{nbr} closest particles of $\mathbf{x}_i$ (for 2D case $N_{nbr} = 3$ and 3D case $N_{nbr} = 6$); $h(\mathbf{x}_i, \mathbf{x}_j)$ is target node distance (see Equation (17)). Inserting e_{avg} into Equation (12), we can obtain the updated target number of particles $N_{total}(t + \Delta t)$.

2.4 | Results

2.4.1 | Convergence analysis

To analyze the performance of the PMD and the effects of particles injection process as well as the boundary nodes handling method in this work, we conduct the convergence analysis through obtaining node distributions for a square domain with dimensions $100 \times 100\ \text{mm}^2$ (see Figure 4) in three different cases, given the total number of particles. In the motion simulation, the time step size is set as $\Delta t = 0.5$ s.

Analyze the performance of the PMD

To focus on the performance of the PMD itself, we assign the boundary nodes, the particle injection positions, and the total number of particles during the initialization stage. The injection positions are uniformly distributed inside the mesh domain and set as $\mathbf{S}_{ij} = [x_i, y_i]$, where

$$\begin{aligned} x_i &= -45 + 11.25i \\ y_i &= -45 + 11.25j \end{aligned}, \tag{19}$$

where $i = 0, 1, \cdots, 8$ and $j = 0, 1, \cdots, 8$. The 81 injection positions are uniformly distributed inside the dashed square (see Figure 4) with dimensions of $90 \times 90\ \text{mm}^2$. We distributed N_b nodes uniformly (N_b is the integer closest to $100/h$) on each boundary edge of the domain as the boundary edge. Several simulations were conducted and the results are summarized in Table 3. $N_{converge}$ is the total number of simulation time steps; N_{inject} is the total number of injection time steps; $N_{net} = N_{converge} - N_{inject}$ is the number of simulation time steps after finishing injecting particles. As we can see

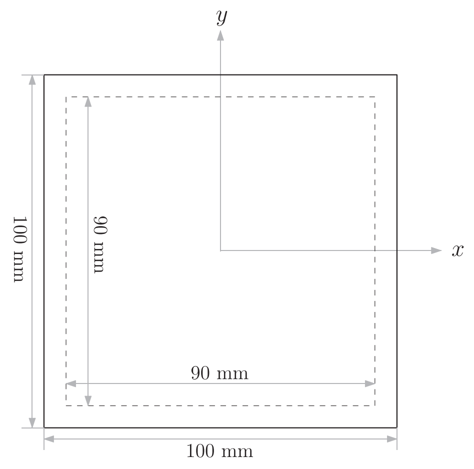

FIGURE 4 2D square domain with dimensions 100 × 100 mm^2. The solid line segments represent the boundary edges of the square domain

TABLE 3 The results of convergence tests given the total number of particles, particle injection positions, and the boundary node distributions. $N_{converge}$ is the total number of simulation time steps; N_{inject} is the total number of injection time steps; $N_{net} = N_{converge} - N_{inject}$ is the number of simulation time steps after finishing injecting particles

N_{total}	200	500	1000	2000	5000	10,000	20,000
h (mm)	7.75	4.80	3.30	2.35	1.45	1.05	0.73
$N_{converge}$	87	98	101	112	148	211	324
N_{inject}	2	6	11	23	59	119	241
N_{net}	85	92	90	89	89	92	83

TABLE 4 The results of convergence tests given different particle injection positions. The target node distance is set as $h = 2.35$ mm and the total number of particles is set as $N_{total} = 2000$. N_S is the total number of injection positions

N_S	81	400	900	1600	900 (Left half)
$N_{converge}$	112	90	90	97	220
N_{inject}	23	5	3	2	3
N_{net}	89	85	87	95	217

from the table, N_{net} for all the seven tests with from 200 to 20,000 particles are around 88 steps. The convergence steps are almost the same no matter how many particles are involved in the simulation in the tests.

The effect of particles' injection positions

To study the influence of particle injection positions on the convergence, the total number of particles is set as $N_{total} = 2000$; the target node distance is set as $h = 2.35$ mm, and the boundary nodes are set according to the method discussed in the above. We have run five different tests and the results are shown in Table 4 For the first four cases with $N_S = 81$, 400, 900, 1600, the injection positions are uniformly distributed inside the dashed square (see Figure 4) with dimensions of 90 × 90 mm^2. In this setting, the particle injection positions roughly distributed uniformly over the domain, and the net convergence steps N_{net} are all around 88 steps. In the last test, the injection positions are set to be distributed uniformly on the left half of the dashed square shown in Figure 4. The net convergence steps increase dramatically and the $N_{net} = 217$, as the particles have to move to the right side of the domain. Therefore, it is better to distribute the injection positions uniformly over the domain, and the simulation converge quickly in this case, because the particles are only required to travel a short distance. Also, the total number of the particle injection positions almost has no effect on the net convergence steps N_{net} as long as the injection positions are distributed uniformly over the domain.

The effect of projecting particles onto the domain boundary

In this group of tests, we will assign the total number of particles, the particle injection positions, and the target node distance to analyze the convergence when handling the boundary nodes. The results of the tests are shown in Table 5. In the tests, the total number of particles are set as $N_{total} = 200$, 500, 1000, 2000, 5000, 10,000, 20,000, and the target node distances are set as $h = 7.75$, 4.80, 3.30, 2.35, 1.45, 1.04, 0.73 mm respectively. The injection positions are uniformly distributed inside the dashed square shown in Figure 4. The total number of the injection positions are set as $N_S = 81$ for the first five tests and $N_S = 400$ for the last two tests to speed up the injection process.

In the first two groups of tests shown in Tables 3 and 4, we observe that the net convergence time steps are almost independent of the total number of particles and the number of injection positions if the injection positions are uniformly distributed over the domain and the boundary nodes are assigned. In the third group of tests (see Table 5), the boundary node distributions are no longer predefined in the initialization stage; when the number of the particles are smaller than 5000, the simulation gets converged around 92 steps, which are slightly more than that in the first two groups of tests, because the extra simulation should be taken to handle the boundary node distributions. Compared to the test with 200 particles, the test with 20,000 particles have more than 100 times the number of particles, while the simulation time steps for the two tests have the same order of magnitudes. Therefore, for the simulation with millions of particles, it is expected that the convergence time steps are in the same order of magnitude as that of the tests with a few thousand particles, if the particle injection positions are uniformly distributed over the domain.

In the following section, the performance of the PIMesh is demonstrated through generating node distributions for various 2D and 3D domains.

2.4.2 | 2D node distributions

Figure 5A shows uniform node distributions for the domain shown in Figure 1A. The target node distance between neighbor particles are set as $h = 2$ mm and two fixed particles are set at $\mathbf{x}_f = [70.0, 70.0; 95.0, 70.0]$. Figure 1B shows the plot of $\Delta d_{avg}/h$ and the total number of particles with respect to the simulation time step number. The initial target number of particles is estimated as 2388, and it takes six simulation time steps to inject the particles inside the domain. At the time step 69, the target number of particles is updated using Equation (12) and 50 new particles are injected inside the domain at sparse areas obtained through the algorithm shown in Table 2. At the time step 78, the target number of particles is updated again and 30 new particles are injected inside the domain at updated injection positions. After 54 time steps, the simulation is converged. It takes total 132 time steps to obtain the resulting node distributions, which have 2468 particles. Figure 1C shows the histogram of L_i (see Equation (11)), which can be considered as a measurement of the regularity of the node distribution. As we can see from Figure 1C, more than 98% of L_i fall in the range of (1.8, 2.2).

Figure 6A shows nonuniform node distributions for the domain shown in Figure 1A. Two fixed points are set at $\mathbf{x}_f = [70.0, 70.0; 95.0, 70.0]$. The target node distance is set as $h = 2$ mm at the inside circular boundary edges and top right sharp corner, and $h = 6$ mm at other boundary edges. Figure 6-b shows the plot of $\Delta d_{avg}/h$ and the total number of particles with respect to the simulation time step number. To obtain the nonuniform node distributions, the simulation gets converged with 333 time steps which is more than (but still in the same order of magnitude as) that for the uniform base, because the initial target number of particles for nonuniform node distributions is estimated based on the maximum target node distance (see Equation (14)) and thus is seriously underestimated. Only 265 particles are injected inside the domain initially and the target number of particles T_{total} is updated 7 times. Figure 6C shows the

TABLE 5 The results of convergence tests given different particle injection positions, the total number of particles, and the target node distance

N_{total}	200	500	1000	2000	5000	10,000	20,000
h (mm)	7.75	4.80	3.30	2.35	1.45	1.04	0.73
N_S	81	81	81	81	81	400	400
$N_{converge}$	97	90	119	113	153	162	210
N_{inject}	3	7	13	25	62	25	50
N_{net}	94	83	106	88	91	137	160

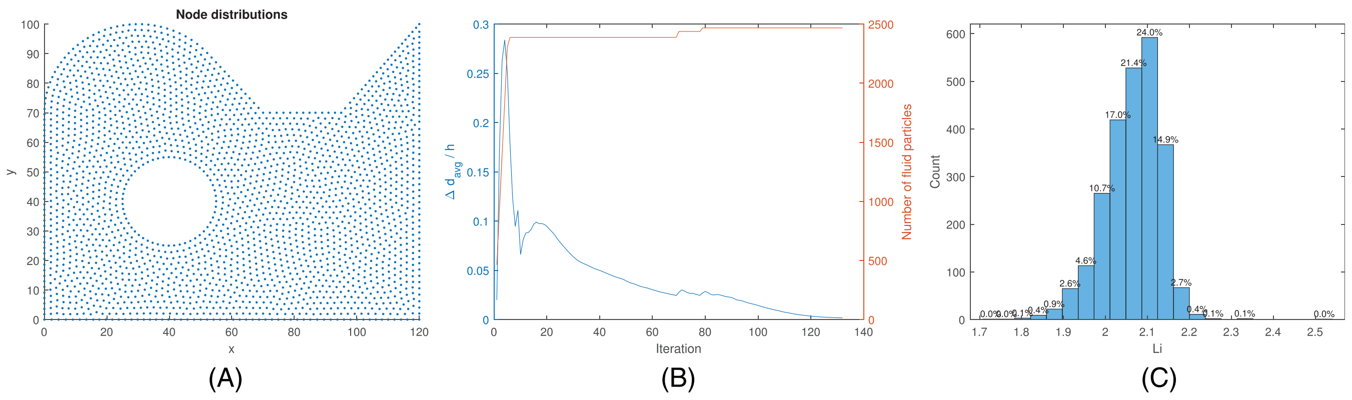

FIGURE 5 (A) Uniform node distributions for the domain shown in Figure 1A. Two fixed points are set at $\mathbf{x}_f = [70.0, 70.0;95.0, 70.0]$ and the target node distance is set as $h = 2$ mm. The node distributions have 2468 particles. (B) The plot of convergence information. The blue line represents the $\Delta d_{avg}/h$ with respect to the simulation time step number and the orange line represents the total number of particles during the simulation (color figure can be viewed in the online issue). (C) Histogram of L_i shown in Equation (11)

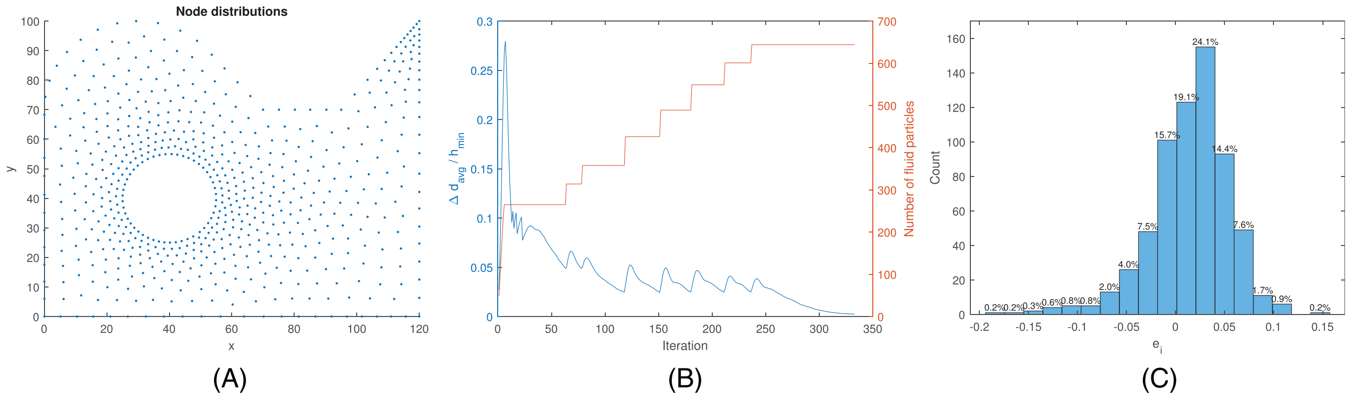

FIGURE 6 (A) Nonuniform node distributions for the domain shown in Figure 1A. Two fixed points are set at $\mathbf{x}_f = [70.0, 70.0;95.0, 70.0]$. The target node distance is set as $h = 2$ mm at the inside circular boundary edges and top right sharp corner, and $h = 6$ *mm* at other boundary edges. The node distributions have 644 particles. (B) The plot of convergence information. The blue line represents the $\Delta d_{avg}/h_{min}$ with respect to the simulation time step number and the orange line represents the total number of particles during the simulation (color figure can be viewed in the online issue). (C) Histogram of the target node distance error e_i (see Equation (18)) for each particle

histogram of e_i (see Equation (18)), which is the error of the average of the distances from a particle to its three nearest neighbor particles. As we can see from Figure 6C, the errors of more than 97% nodes are smaller than 0.1.

Figure 7A shows nonuniform distributions for a 100 × 100 mm^2 domain that is often used in crack analysis. Twenty fixed points are set at $\mathbf{F}_i = [-20.0 + 2i, 0.0]$ where $i = 0, 1, 2, \cdots, 20$. The target node distance is set as $h = 2$ mm near the locations of the fixed nodes, and $h = 5$ *mm* at the boundary edges. The target number of particles T_{total} is updated seven times and it takes 420 time steps to get the node distributions that have 792 particles. Figure 7B shows the histogram of e_i (see Equation (18)). As we can see from Figure 1B, the errors of more than 99% nodes are smaller than 0.1.

2.4.3 | 3D node distributions

In this section, the node distributions for 3D domains are demonstrated. Figure 8A shows uniform distributions for the femur domain shown in Figure 2A. The target node distance is set as $h = 5$ mm and the resulting node distributions have 1513 particles. The target number of particles T_{total} is updated four times and it takes 113 time steps to get the simulation converged. Figure 8B shows the histogram of L_i (see Equation (11)), which is the average of the distances from a particle to its six nearest neighbor particles. The average node distance is 5.17 mm and the standard deviation of node distance is 0.11 mm.

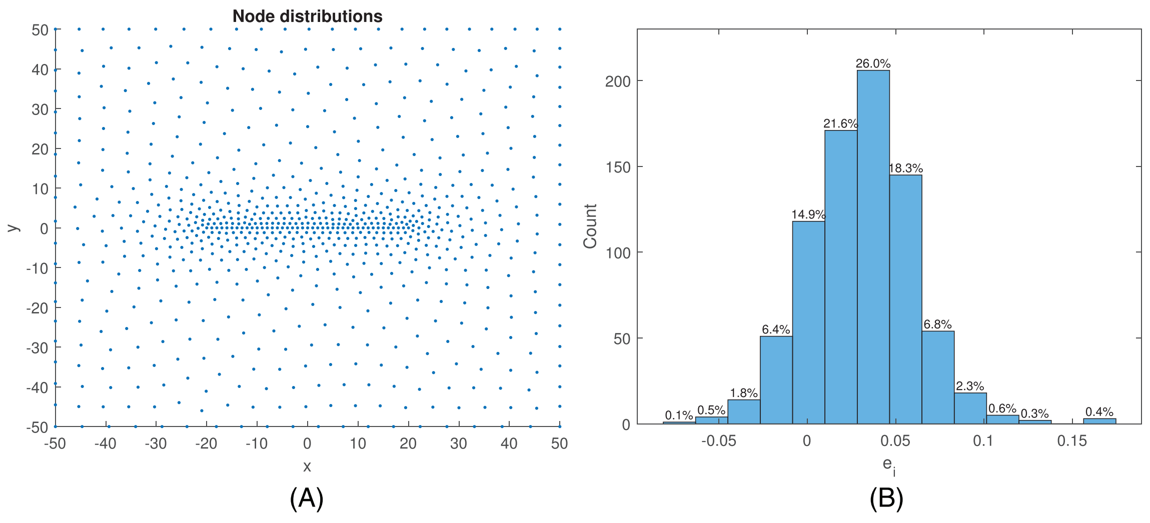

FIGURE 7 (A) Nonuniform node distributions for the domain shown in Figure 4. Twenty fixed points are set at $\mathbf{x}_f^i = [-20.0 + 2i, 0.0]$ where $i = 0, 1, 2, \cdots, 20$. The target node distance is set as $h = 2\,\text{mm}$ near the locations of the fixed nodes; and $h = 5\ mm$ at the outside boundary edges. The node distributions have 792 particles. The simulation converged at 420 time steps. (B) Histogram of the target node distance error e_i (see Equation (18)) for each particle

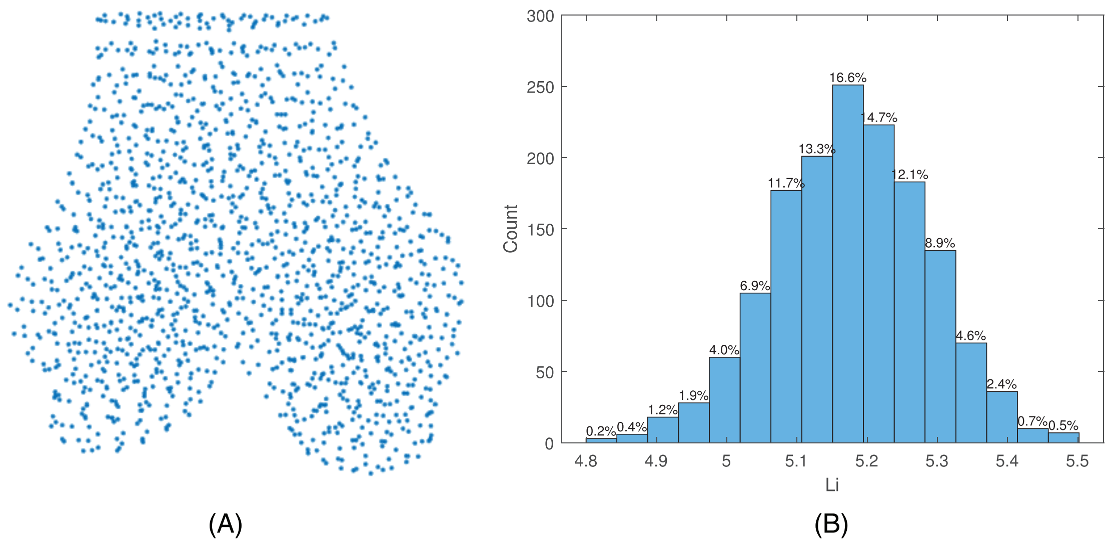

FIGURE 8 (A) Uniform node distributions for the domain shown in Figure 2A. The target node distance is set as $h = 5$ mm and the resulting node distributions have 1513 particles. The simulation get converged at 113 time steps. (B) Histogram of L_i (see Equation (11)), which is the average of the distances from a particle to its six nearest neighbor particles

Figure 9A represents the domain for a piece of hipbone, and Figure 9B shows uniform distributions for the hipbone domain. The target node distance is set as $h = 5$ mm and the resulting node distributions have 3667 particles. The target number of particles T_{total} is updated 1 time and it takes 118 time steps to get the simulation converged. Figure 9C shows the histogram of L_i (see Equation (11)), which is the average of the distances from a particle to its six nearest neighbor particles. The average node distance is 5.25 mm and the standard deviation of the node distance is 0.13.

Figure 10A represents the domain for a 3D liver model, and Figure 10B shows uniform distributions for the liver. The target node distance is set as $h = 5$ mm and the resulting node distributions have 6223 particles. The target number of particles T_{total} is updated three times and it takes 123 time steps to get the simulation converged. Figure 10C shows the histogram of L_i (see Equation (11)), which is the average of the distances from a particle to its six nearest neighbor particles. The average node distance is 5.19 mm and the standard deviation of the node distance is 0.13.

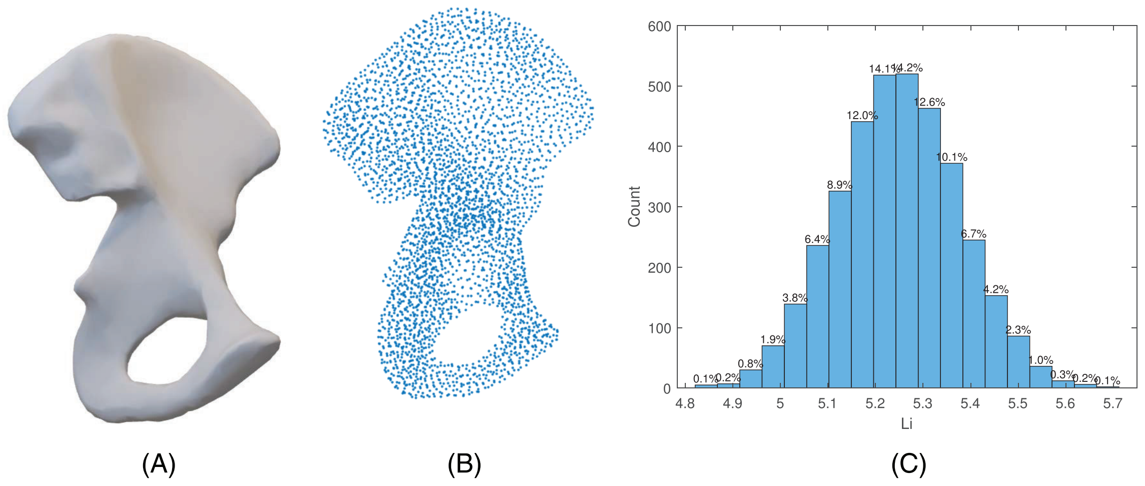

FIGURE 9 (A) The geometry of a hipbone. (B) Uniform node distributions for the hipbone domain. The target node distance is set as $h = 5$ mm and the resulting node distributions have 3667 particles. The simulation converged at 118 time steps. (C) Histogram of L_i (see Equation (11)), which is the average of the distances from a particle to its six nearest neighbor particles

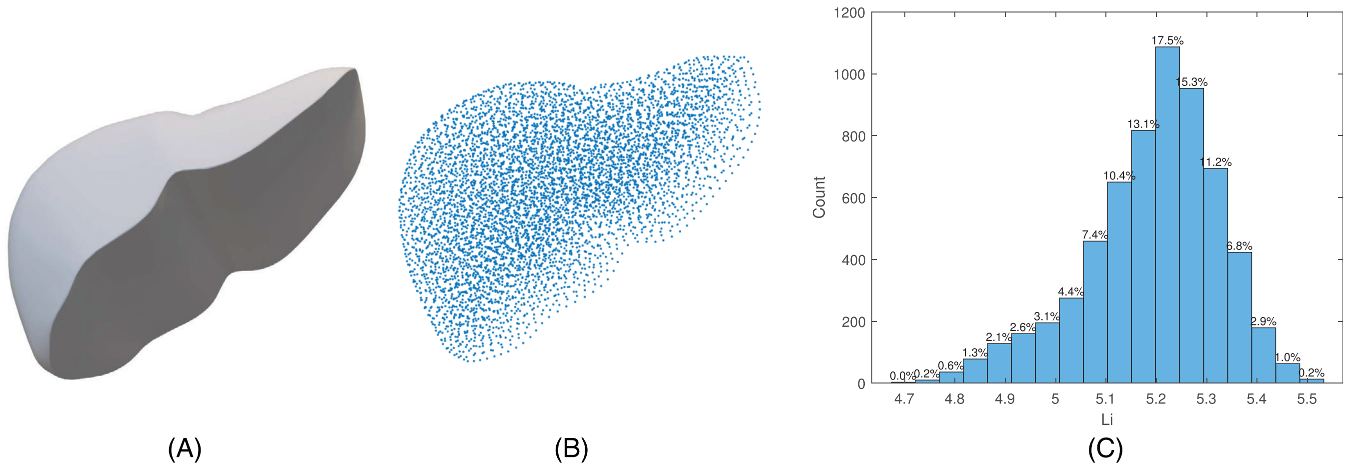

FIGURE 10 (A) The geometry of a 3D live model. (B) Uniform node distributions for the 3D live domain. The target node distance is set as $h = 5$ mm and the resulting node distributions have 6223 particles. The simulation get converged at 123 time steps. (C) Histogram of L_i (see Equation (11)), which is the average of the distances from a particle to its six nearest neighbor particles

Figure 11A shows uniform distributions for a cube domain with dimensions of $150 \times 50 \times 50$ mm^3. The target node distance is defined explicitly as

$$h_c(x) = h_{min} + \frac{x+75}{150}(h_{max} - h_{min}), \tag{20}$$

where the origin of the coordinator locates at the center of the cuboid; $\mathbf{x} = [x, y, z]$ (x axis is along the length of the cuboid); $h_{min} = 3$ mm; and $h_{min} = 10$ mm. The resulting node distributions have 3729 particles. The target number of particles T_{total} is updated nine times and it takes 412 time steps to get the simulation converged. Figure 11B is the histogram of e_i (see Equation (18)), which is the error of the average of the distances from a particle to its six nearest neighbor particles. The average node distance error is $e_{avg} = 0.02$.

Figure 12A is the nonuniform node distributions for the hipbone domain shown in Figure 9A. The target node distance is set as 3 mm at the bottom part and 6 mm at the upper part. The node distributions have $T_{total} = 8299$ particles. The target number of particles T_{total} is updated eight times and it takes 579 time steps to get the simulation converged.

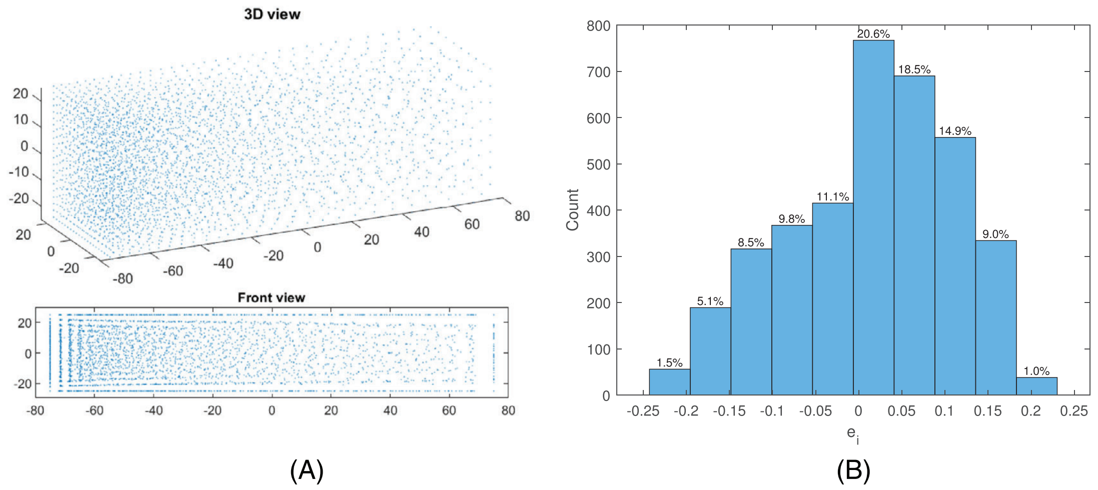

FIGURE 11 (A) Nonuniform node distributions for a cuboid domain with dimensions of 150 × 50 × 50 mm^3. The target node distance is explicitly defined as Equation (20). The node distributions have 3729 particles. The simulation get converged at 412 time steps. (B) Histogram of the target node distance error e_i (see Equation (18)) for each particle

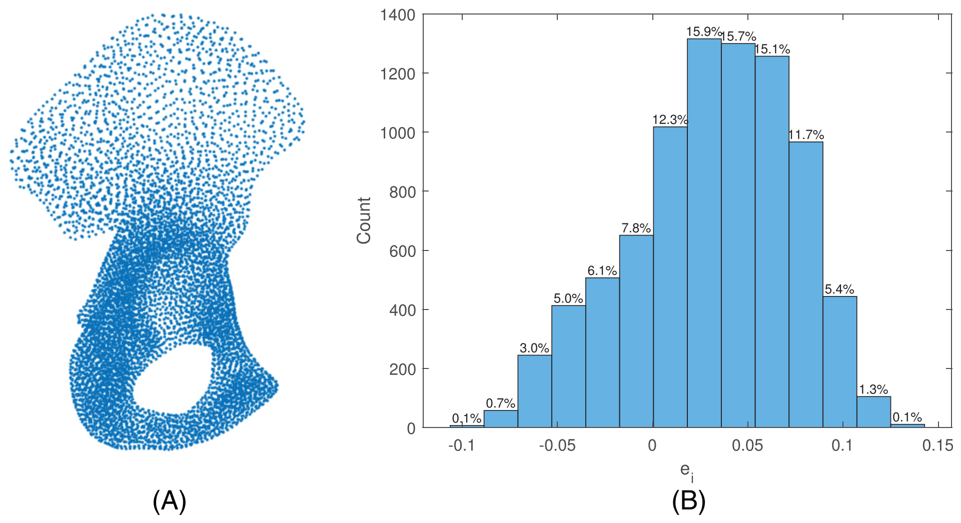

FIGURE 12 (A) Nonuniform node distributions for the hipbone domain shown in Figure 9A. The target node distance is set as 3 mm at the bottom part and 6 mm at the upper part. The node distributions have 8299 particles. The simulation is converged at 579 time steps. (B) Histogram of the target node distance error e_i (see Equation (18)) for each particle

Figure 12B is the histogram of e_i (see Equation (18)), which is the error of the average of the distances from a particle to its six nearest neighbor particles. The average node distances error is $e_{avg} = 0.03$.

3 | MESH GENERATION

Generating unstructured meshes includes two steps: (1) obtaining mesh node distributions and (2) linking the nodes using Delaunay triangulation (or other schemes) to generate resulting meshes. The algorithm to obtain mesh node distributions is similar to the algorithm shown in Table 1, and the differences are (1) the calculation of the average edge distance error (corresponding to the average edge length error e_{avg} for node distributions), and (2) the algorithm to update the particle injection positions.

To update the total number of particles N_{total}, we first draw a mesh based on the existing particles. Then we select all the high-quality triangles with angles greater than 30° and smaller than 110° to create a sample mesh. The average edge length error for uniform meshes is obtained as,

$$e_{avg} = \frac{1}{N_{edges}} \sum_{i=1}^{N_{edges}} \frac{E_i - h}{h}, \tag{21}$$

where N_{edges} is the total number of sample mesh edges; E_i is the length of the i_{th} edge. The average edge length error for nonuniform meshes is obtained as, the average edge length error of the mesh is then obtained as,

$$e_{avg} = \frac{1}{N_{edges}} \sum \frac{\left|\left|x_i - x_j\right|\right| - h\left(x_i, x_j\right)}{h\left(x_i, x_j\right)}, \tag{22}$$

where N_{edges} is the total number of edges of the sample mesh; $\mathbf{x}_i$ and $\mathbf{x}_j$ are the two ends of a mesh edge; $h(\mathbf{x}_i, \mathbf{x}_j)$ is target edge length (see Equation (17)). Then we can insert e_{avg} into Equation (12) to obtain the updated target number of particles $N_{total}(t + \Delta t)$.

To update the particle injection positions, similar to the algorithm shown in Table 2, we (1) obtain the length of each edge of the sample mesh; (2) find the $N_{new} = N_{total}(t + \Delta t) - N_{total}(t)$ longest edges; and (3) the new injections positions are on the center of the N_{new} edges.

3.1 | Comparison to the bubble mesh

The bubble mesh[16] has a similar formulation to the PMD, as both methods are based on Newton's second law, and the forces driving the motion of particles are based on their neighbor particles. Therefore, we make a detailed comparison between the PMD and the bubble mesh in this section.

In the bubble mesh[16] and its modified versions, such as the work of,[36] the forces driving the motion of bubbles is similar to Van der Waals force. However, unlike the method presented here, the aim of bubble mesh is to develop a "zero force positions" of the nodes where forces among bubbles are zero. Thus, when two neighbor bubbles are too close to each other, repelling forces are generated to push bubbles away from each other; when the distance between two bubbles is equal to the target node distance, the forces generated between these two bubbles become zero; when the distance between two neighbor bubbles is larger than the target node distance, attractive forces will be generated to bring these two bubbles closer. In the PMD, the forces driving particles' motion keep pushing neighbor particles away from each other and so there is always "pressure" in the system. The differences in the forces driving the particles' motion bring essential differences and much improved performance of PIMesh compared to bubble mesh.

- The repelling particles in PIMesh are trying to take all the space of a mesh domain, while the positions of the bubbles in the bubble mesh are only adjusted locally. Therefore, the bubble mesh requires a high-quality initial bubble configuration to ensure fast convergence.
- The population control algorithm is different. In the PIMesh, the total number of particles is estimated based on errors between the edge lengths and the target edge lengths, and the calculation of the errors can be parallelized. While for the bubble mesh, the population is controlled based on local space between neighbor particles. If the gap between two bubbles is too big, a new bubble will be inserted; if several bubbles too close to each other, extra bubbles will be removed. The bubble population control algorithm cannot be parallelized. Therefore, the bubble mesh can suffer if (1) the quality of the initial bubble configurations is low or (2) a great many bubbles are required for a large mesh domain.
- The viscous forces to stabilize the motion are also different. In the PIMesh, the viscous force for a particle is based on the particle's momentum rate, and the viscous constant k_v can be any value between $0 < k_v < 1$; therefore, the viscous constant can be optimized to speed up the convergence (see Section 2.2.1 for the details). While for the bubble mesh, the viscous force is based on a bubble's velocity. The viscous constant should be carefully selected to ensure that the viscous force can stabilize the motion simulation.

3.2 | Comparison to the SPH based mesh generation methods

As both SPH based mesh generation methods[18,19] and PMD based mesh generation methods simulate the motion of particles over the domain, we make a direct comparison between the PIMesh and SPH based mesh generation methods[18,19] through generating a mesh for the Zalesak's disk. As the target edge length settings in the work by Fu et al.[18] is not given explicitly, we set the target edge length as $h = 0.37$ mm (uniform mesh). Figure 13A is the plot of the mesh generated for the Zalesak's disk using the PIMesh. The mesh has 5041 particles and the total number of particles is updated four times. The convergence information is shown in Figure 13B. Figure 13C is the histogram of the angles of the mesh, with maximum angle $\theta_{max} = 101.58^{\circ}$ and minimum angle $\theta_{min} = 36.78^{\circ}$. The quality of the resulting mesh is similar to the mesh generated by the SPH based mesh generation method.[18]

Table 6 shows the comparison between the PIMesh and SPH[18] and improved SPH[19] based mesh generation methods. Note that even though the mesh in Figure 13 is uniform, it is still reasonable to make comparisons between the PIMesh and other SPH based methods, because the convergence of the uniform and nonuniform node distributions are in the same order of magnitude (typically the nonuniform node distributions take about four times time steps to get converged). The core difference between the PIMesh and SPH based mesh generators is in forces driving the motion of the particles. For the PIMesh, the forces are the repelling forces calculated through PMD; for the SPH based mesh generation methods, the forces are calculated based on the gradient of pressure among the particles. This difference in the forces results in the advantages of the PIMesh over the SPH based mesh generation methods:

- Compared to the SPH based mesh generation methods, the PMD for the PIMesh avoids the complex calculation of density, the gradient of pressure, and the gradient of velocities that are required in the SPH based mesh generation methods; therefore, the PIMesh is much more concise than the SPH based mesh generation method.
- The PIMesh does not require any initial node distributions and the population of the particles is updated dynamically using the PID controller-based particles population control algorithm. The PID controller can ensure the resulting mesh size is very close to the target size, while the mechanism in the SPH based mesh generation methods to ensure the mesh size is not clear in the work.[18,19]

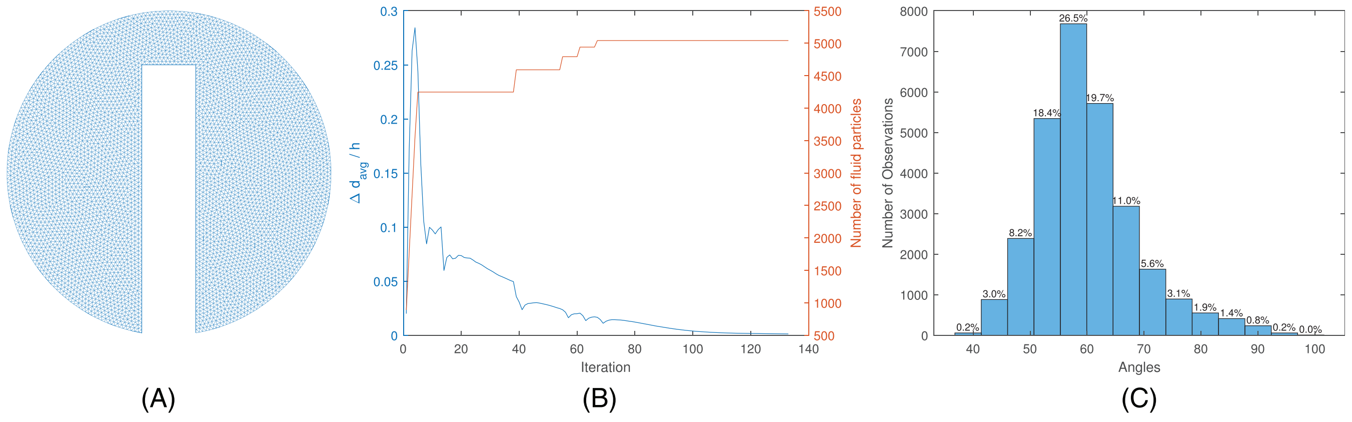

FIGURE 13 (A) Uniform mesh with 5041 particles with two fixed particles at the two concave corners. The average edge length error is 1.95%. (B) The mesh is converged at 133 steps and T_{total} is updated 4 times. (C) The histogram of the angles of the mesh, with $\theta_{max} = 101.58^{\circ}$ and $\theta_{min} = 36.78^{\circ}$

TABLE 6 The comparison between the PIMesh and SPH[18] and improved SPH[19] based mesh generation methods through generating meshes for Zalesak's disk

	Dynamics	Initial nodes	Nodes outside	Convergence steps	N_{total}
PIMesh	PMD	No	No	113	5041
Improved SPH	SPH	Yes	Yes	4200	≈5047
SPH	SPH	Yes	Yes	≈45,000	5047

Abbreviations: PIMesh, particle injection mesh generator; PMD, pseudo-molecular dynamics; SPH, smoothed-particle hydrodynamics.

- The PIMesh does not require any initial boundary node distributions, while the SPH based methods require high-quality boundary node distributions at the initial stage. For the 3D cases, obtaining boundary node distributions is essentially a high-quality 3D surface mesh generation task. This further complicates the SPH based methods.
- The SPH based mesh generation methods employ ghost particles (that require extra computation) rather than the projection method in the PIMesh to handle particles outside the mesh domain. A possible reason is that if the projection method rather than ghost particles is employed, excessive particles will move onto the boundaries of the mesh domain and the resulting meshes could have many mesh elements with small angles near the boundary. This could be caused by two reasons. (1) The particles are driven by the gradient of pressure, and without ghost particles outside the mesh domain, the pressure near the boundaries is always lower than that inside the domain; therefore, excessive particles will be pushed to the boundaries. (2) Unlike the PIMesh which employs the PID controller to control the population of the particles, the SPH based methods allow extra particles to move outside the mesh domain and thus adjust the population of particles. If the projection-based method is employed, the particles which are supposed to move outside the mesh domain are projected back onto the boundaries. Thus, excessive particles will be on the boundaries.
- In the tasks of generating meshes for the Zalesak's disk, the PIMesh is converged at 113 steps, which is an order of magnitude faster than the improved SPH based mesh generation method, and two orders of magnitude faster than the SPH based mesh generation method.

It is worth noting that most of the parallel computing techniques for the SPH can also be used for the PMD. Injecting particles can also be parallelized but it may not be necessary, because the task is very simple, and only random velocities are assigned to particles at given positions. Therefore, it is expected that the PIMesh can be used to generate meshes with millions of nodes efficiently when parallel computing techniques are employed. In the following section, the performance of the PIMesh is demonstrated through generating 2D and 3D meshes.

3.3 | 2D Results

3.3.1 | Uniform mesh

Figure 14 shows a uniform mesh for the mesh domain shown in Figure 1A. In Figure 14A, the target length is set as 2 mm. The resulting uniform mesh has 2845 nodes and the average edge length is 2.04 mm (the average edge length error is 2.0%). Figure 14B shows the histogram of the angles of the mesh triangles. The maximum angle is $\theta_{max} = 103.65^{\circ}$ and the minimum angle is $\theta_{min} = 37.42^{\circ}$.

3.3.2 | Nonuniform meshes

Figure 15 shows a nonuniform mesh for the mesh domain shown in Figure 1A. Two fixed points are set at $\mathbf{x}_f = [70.0, 70.0; 95.0, 70.0]$. The target edge length is set as $h = 2$ mm at the inside circular boundary edges and top right sharp corner; and $h = 6\ mm$ at other boundary edges. The resulting mesh has 765 nodes and the average edge length error $e_{avg} = 0.26\%$. The simulation is converged at 363 steps. Figure 15B is the histogram of the angles of the mesh triangles The maximum angle is $\theta_{max} = 101.2426$ and the minimum angle is $\theta_{min} = 35.6300$, and the mesh quality is similar to the uniform mesh shown in Figure 14A.

Figure 16 shows a nonuniform mesh for the mesh domain with dimensions 100×100 mm^2. Twenty fixed points are set at $\mathbf{x}_{fi} = [-20.0 + 2i, 0.0]$ where $i = 0, 1, 2, \cdots, 20$. The target edge length is set as $h = 2$ mm near the locations of the fixed nodes, and $h = 5\ mm$ at the boundary edges. The resulting mesh has 948 nodes and the average edge length error $e_{avg} = 1.43\%$. The simulation is converged at 478 steps. Figure 16B is the histogram of the angles of the mesh triangles. The maximum angle is $\theta_{max} = 104.49^{\circ}$ and the minimum angle is $\theta_{min} = 32.39^{\circ}$.

In the above 2D uniform and nonuniform triangular meshes, all the angles of the meshes are fall in the range of $(30^{\circ}, 105^{\circ})$. Therefore, there are no bad triangles in the meshes. Even without post-processing algorithms, the 2D triangular meshes are of high-quality.

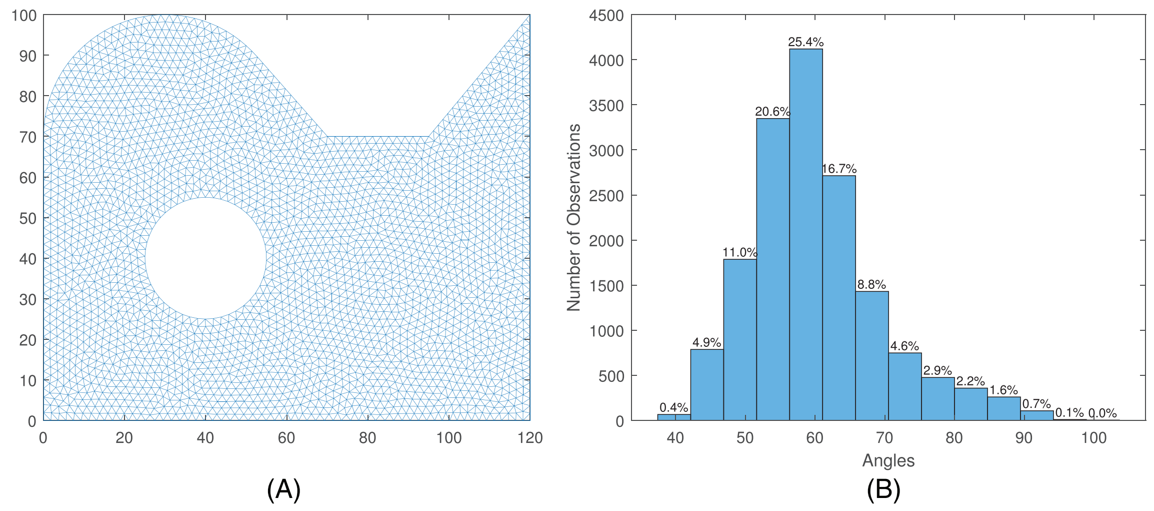

FIGURE 14 (A) A uniform mesh for the mesh domain shown in Figure 1A. The mesh has 2845 particles, with two fixed points at $\mathbf{x}_f = [70.0, 70.0; 95.0, 70.0]$. The target edge length is $h = 2$ mm and the average edge length of the resulting mesh is 2.04 mm. The simulation is converged at 107 steps. (B) The histogram of the angles of the mesh triangles ($\theta_{max} = 103.65^\circ$ and $\theta_{min} = 37.42^\circ$)

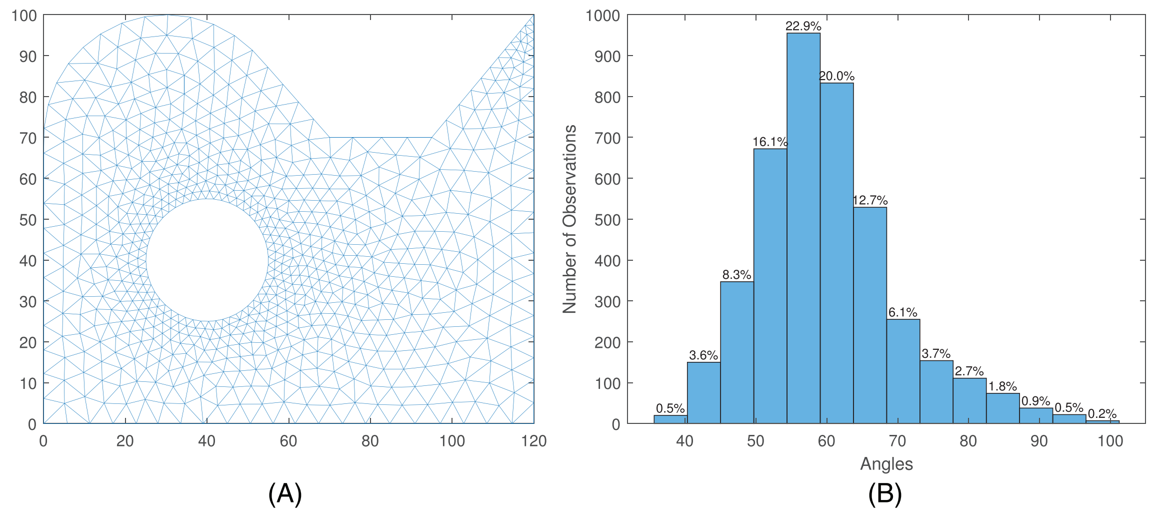

FIGURE 15 (A) A nonuniform mesh for the mesh domain shown in Figure 1A. Two fixed points are set at $\mathbf{x}_f = [70.0, 70.0; 95.0, 70.0]$. The target edge length is set as $h = 2$ mm at the inside circular boundary edges and top right sharp corner; and $h = 6\ mm$ at other boundary edges. The mesh has 765 nodes with the average edge length error $e_{avg} = 0.26\%$. The simulation is converged at 363 steps. (B) The histogram of the angles of the mesh triangles ($\theta_{max} = 101.24^\circ$ and $\theta_{min} = 35.63^\circ$)

3.3.3 | 3D meshes

For 3D meshes, even though the particles are distributed sparsely, the Delaunay triangulation can create a few tetrahedrons that with small dihedral angles. Therefore, we employ the optimization algorithm base on face swapping method[37] to remove the tetrahedrons with small dihedral angles.

Figure 17 shows a uniform mesh for the mesh domain shown in Figure 2A. The target length is set as $h = 5$ mm. The resulting uniform mesh has 2217 nodes and the average edge length error is $e_{avg} = 1.91\%$. The simulation is converged at 192 time steps. Figure 17B shows the histogram of the dihedral angles of the mesh. The maximum angle is $\theta_{max} = 149.93^\circ$ and the minimum angle is $\theta_{min} = 20.06^\circ$.

Figure 18 shows a uniform mesh for the mesh domain shown in Figure 10A. The target length is set as $h = 8$ mm. The resulting uniform mesh has 2720 nodes and the average edge length error is $e_{avg} = -1.79\%$. The simulation is converged at 206 time steps. Figure 18B shows the histogram of the dihedral angles of the mesh. The maximum angle is $\theta_{max} = 150.02^\circ$ and the minimum angle is $\theta_{min} = 20.06^\circ$.

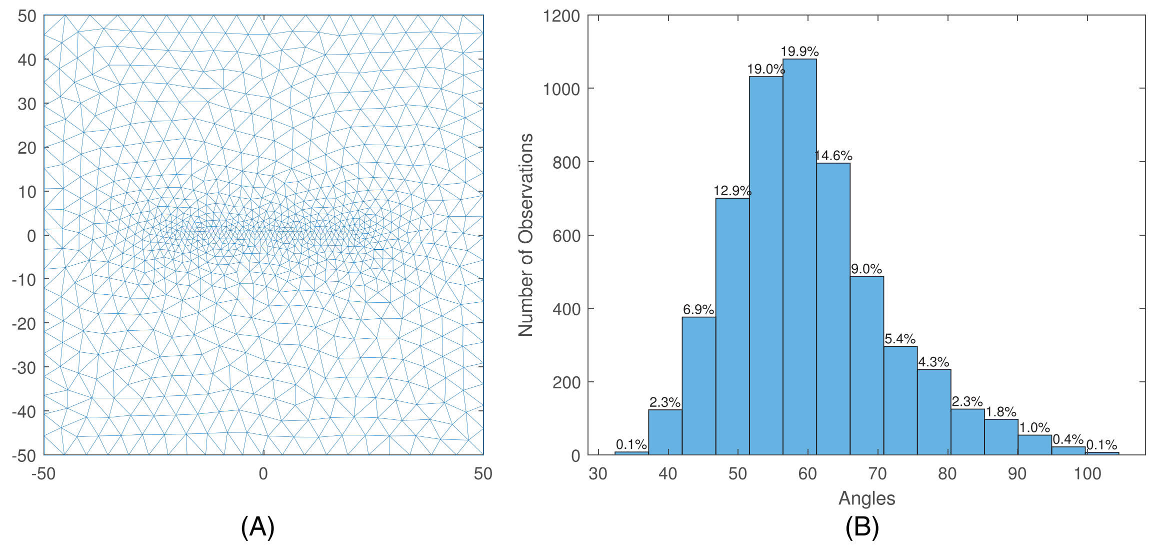

FIGURE 16 (A) A nonuniform mesh for the mesh domain with dimensions 100 × 100 mm^2. Twenty fixed points are set at $\mathbf{x}_f^i = [-20.0 + 2i, 0.0]$ where $i = 0, 1, 2, \cdots, 20$. The target edge length is set as $h = 2\,\text{mm}$ near the locations of the fixed nodes, and $h = 5\ mm$ at the boundary edges. The resulting mesh has 948 nodes and the average edge length error $e_{avg} = 1.43\%$. The simulation is converged at 478 steps. (B) The histogram of the angles of the mesh triangles. The maximum angle is $\theta_{max} = 104.49^\circ$ and the minimum angle is $\theta_{min} = 32.39^\circ$

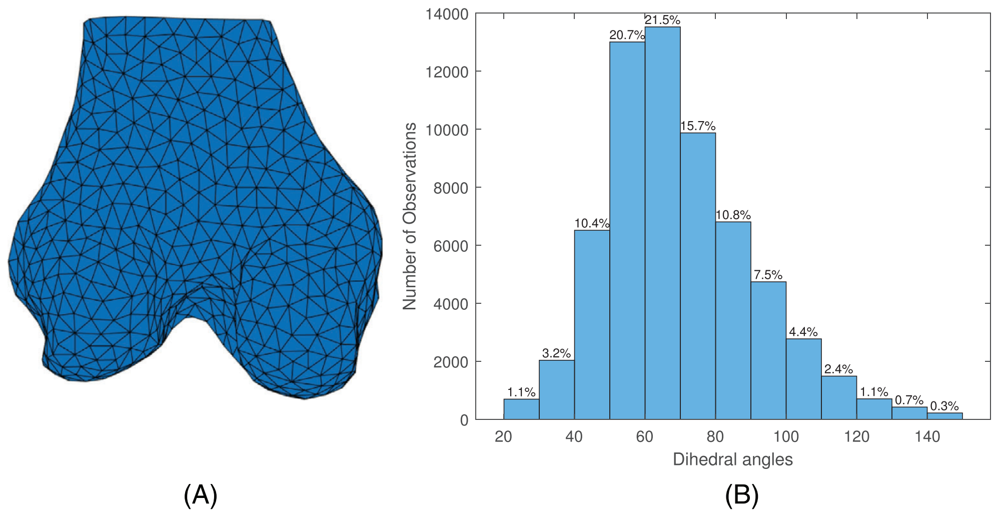

FIGURE 17 (A) A uniform mesh for the mesh domain shown in Figure 2A. The mesh has 2217 particles. The target edge length is $h = 5$ mm and the average edge length error is $e_{avg} = 1.91\%$. The simulation is converged at 192 steps. (B) histogram of the dihedral angles of the mesh. The maximum angle is $\theta_{max} = 149.93^\circ$ and the minimum angle is $\theta_{min} = 20.06^\circ$

Figure 19 shows a uniform mesh for the mesh domain shown in Figure 9A. The target length is set as $h = 5$ mm. The resulting uniform mesh has 5536 nodes and the average edge length error is $e_{avg} = 1.77\%$. The simulation is converged at 204 time steps. Figure 19B shows the histogram of the dihedral angles of the mesh. The maximum angle is $\theta_{max} = 155.66^\circ$ and the minimum angle is $\theta_{min} = 16.51^\circ$.

Figure 20A is a nonuniform mesh for the mesh domain shown in Figure 9A. The target edge length is set as 5 mm at the bottom part and 10 mm at the upper part. The resulting uniform mesh has 2925 nodes and the average edge length error is $e_{avg} = 2.56\%$. The simulation is converged at 350 time steps. Figure 20B shows the histogram of the dihedral angles of the mesh. The maximum angle is $\theta_{max} = 149.42^\circ$ and the minimum angle is $\theta_{min} = 16.94^\circ$.

For all the 3D meshes generated in the above, the average edge length errors are all smaller than 3.0%. Meanwhile, more than 97% dihedral angles are in the range of (30, 150) degrees. There are only a few dihedral angles smaller than 30° but greater than 15°, and less than 0.5% dihedral angles that are in the range of (150, 155) degrees. Therefore, the average edge length is accurately controlled and there are no flatten tetrahedrons in the meshes generated in this work.

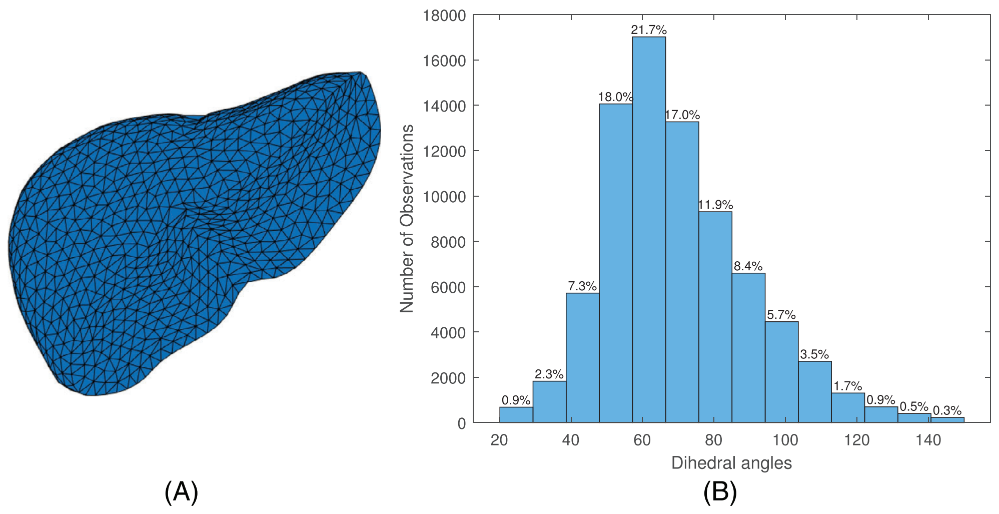

FIGURE 18 (A) A uniform mesh for the mesh domain shown in Figure 10A. The target length is set as $h = 8$ mm. The resulting uniform mesh has 2720 nodes and the average edge length error is $e_{avg} = -1.79\%$. The simulation is converged at 206 time steps. (B) The histogram of the dihedral angles of the mesh. The maximum angle is $\theta_{max} = 150.02^{\circ}$ and the minimum angle is $\theta_{min} = 20.06^{\circ}$

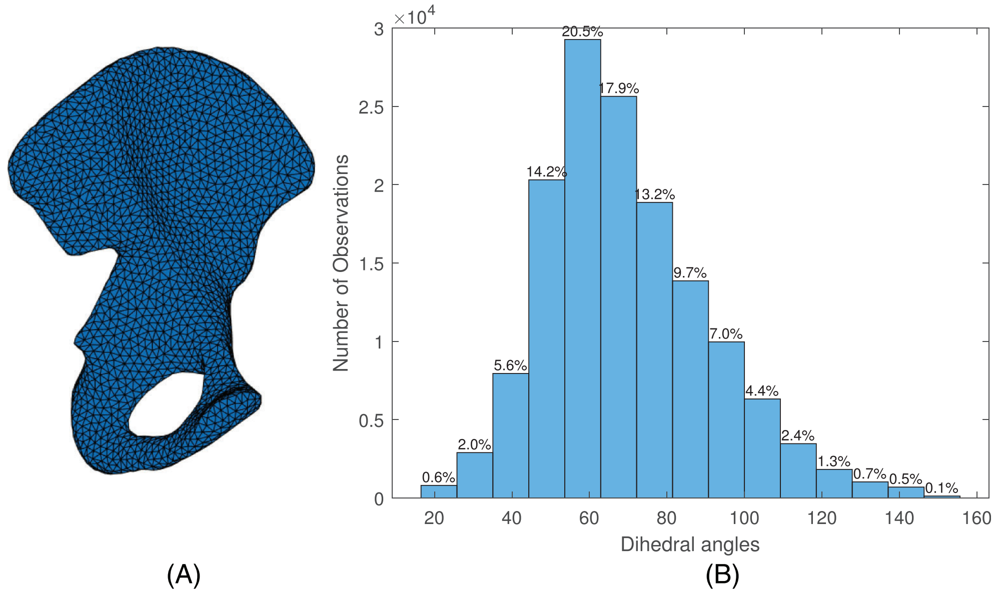

FIGURE 19 (A) A uniform mesh for the mesh domain shown in Figure 9A. The target length is set as $h = 5$ mm. The resulting uniform mesh has 5536 nodes and the average edge length error is $e_{avg} = 1.77\%$. The simulation is converged at 204 time steps. (B) The histogram of the dihedral angles of the mesh. The maximum angle is $\theta_{max} = 155.66^{\circ}$ and the minimum angle is $\theta_{min} = 16.51^{\circ}$

4 | CONCLUSION AND FUTURE WORK

The PIMesh proposed in this work can be used to generate node distributions and unstructured meshes for an object in any shape with minimum (or even no) user intervention. The performance of the PIMesh is demonstrated by generating node distributions and meshes for several 2D and 3D domains. As the kernel function in the PMD is essentially the same as the window function of meshless methods, the quality of the node distributions is guaranteed. The resulting meshes show that the qualities of the meshes are good and the accuracy of edge length can be guaranteed. Particularly, no post-processing procedure is required for generating high-quality 2D triangular meshes. The convergence of the PIMesh is almost independent of the total number of particles, and depends on the distributions of particle injection positions as well as the estimated number of particles at the initial stage. The results indicate that all the simulations can get converged within a few hundred time steps. Additionally, the motion simulation of the PMD and SPH share a

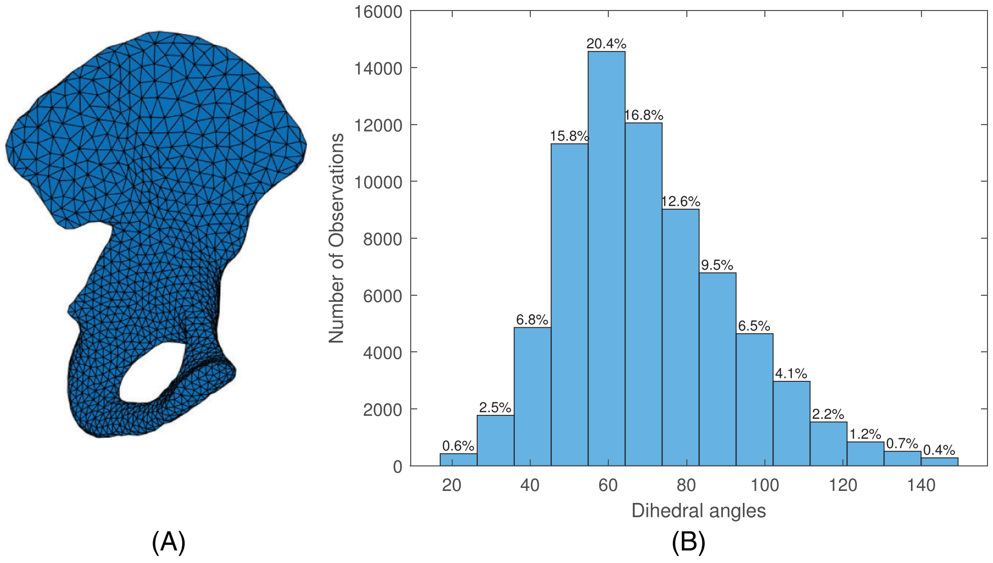

FIGURE 20 (A) A nonuniform mesh for the mesh domain shown in Figure 9A. The target edge length is set as 5 mm at the bottom part and 10 mm at the upper part. The resulting uniform mesh has 2925 nodes and the average edge length error is $e_{avg} = 2.56\%$. The simulation is converged at 350 time steps. (B) The histogram of the dihedral angles of the mesh. The maximum angle is $\theta_{max} = 149.42^{\circ}$ and the minimum angle is $\theta_{min} = 16.94^{\circ}$

lot of similarities in technical details, such as searching for neighbor particles and the use of kernel functions, etc. Therefore, the PIMesh has a very good potential to generate node distributions and meshes with millions of nodes efficiently if parallel computing techniques are employed.

Creating 3D models using CT scan images and 3D scanned images is common in the medical healthcare community nowadays. Therefore, it is completely possible that a healthcare educator can provide 3D models required in surgical training scenarios. In the future, we will develop a surgical simulator integrating the PIMesh which is used to automatically generate meshes and node distributions for the 3D models to run the surgical simulation. Besides, we will employ parallel computing techniques to improve the efficiency of the PIMesh. Furthermore, other triangulation algorithms and post-optimization algorithms will be investigated to link and move particles to generate tetrahedron meshes with better qualities. The ultimate goal is to publish the PIMesh as an alternative open source mesh generator.

ACKNOWLEDGEMENTS

The first and last authors gratefully acknowledge the support* of the program of Canada Research Chair in Health Care Simulation. The second and third authors acknowledge the support of their respective endowed professorships.

ORCID

Zhujiang Wang https://orcid.org/0000-0002-3628-9737
Junuthula N. Reddy https://orcid.org/0000-0002-9739-1639

ENDNOTE

* This project was led by Zhujiang Wang as a part of post-doctoral studies supported by Canada Research Chair in Health Care Simulation awarded to Adam Dubrowski

REFERENCES

1. Owen SJ. A survey of unstructured mesh generation technology. The 7th International Meshing Roundtable Sandia National Lab; 1998. p. 239–267.
2. Bommes D, Lévy B, Pietroni N, et al. Quad-mesh generation and processing: a survey. *Computer Graphics Forum*. Vol 32. Wiley Online Library; 2013:51-76.
3. Lo DSH. *Finite Element Mesh Generation*. CRC Press; 2014.
4. Shewchuk JR. Delaunay refinement mesh generation. PhD thesis, Carnegie-Mellon Univ Pittsburgh Pa School of Computer Science; 1997.

5. Audette MA, Chernikov AN, Chrisochoides NP. A review of mesh generation for medical simulators. *Handbook of Real-World Applications in Modeling and Simulation*. Wiley Online Library; 2012:261-297.
6. Mounoury V, Stab O. Automatic quadrilateral and hexahedral finite element mesh generation: review of existing methods. *Revue Européenne Des Éléments Finis*. 1995;4(1):75-102.
7. Ruppert J. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *J Algorithms*. 1995;18(3):548-585.
8. Shewchuk JR. Delaunay refinement algorithms for triangular mesh generation. *Comput Geom*. 2002;22(1–3):21-74.
9. Foteinos P, Chrisochoides N. Dynamic parallel 3D Delaunay triangulation. Proceedings of the 20th International Meshing Roundtable Springer; 2011.p. 3–20.
10. Schöberl J. NETGEN an advancing front 2D/3D-mesh generator based on abstract rules. *Comput Visualization Sci*. 1997;1(1):41-52.
11. Mohammadi F, Dangi S, Shontz SM, Linte CA. A direct high-order curvilinear triangular mesh generation method using an advancing front technique. International Conference on Computational Science Springer; 2020. p. 72–85.
12. Shephard MS, Georges MK. Automatic three-dimensional mesh generation by the finite octree technique. *Int J Numer Methods Eng*. 1991;32(4):709-749.
13. Yerry MA, Shephard MS. Automatic three-dimensional mesh generation by the modified-octree technique. *Int J Numer Methods Eng*. 1984;20(11):1965-1990.
14. Mavriplis DJ. An advancing front Delaunay triangulation algorithm designed for robustness. *J Comput Phys*. 1995;117(1):90-101.
15. Borouchaki H, Laug P, George PL. Parametric surface meshing using a combined advancing-front generalized Delaunay approach. *Int J Numer Methods Eng*. 2000;49(1–2):233-259.
16. Shimada K, Gossard DC. Bubble mesh: automated triangular meshing of non-manifold geometry by sphere packing. Proceedings of the third ACM symposium on solid modeling and applications; 1995. p. 409–419.
17. Monaghan JJ. Smoothed particle hydrodynamics. *Rep Prog Phys*. 2005;68(8):1703-1759.
18. Fu L, Han L, Hu XY, Adams NA. An isotropic unstructured mesh generation method based on a fluid relaxation analogy. *Comput Methods Appl Mech Eng*. 2019;350:396-431.
19. Ji Z, Fu L, Hu X, Adams N. A feature-aware SPH for isotropic unstructured mesh generation. *arXiv*. 2020;200301061.
20. Persson PO, Strang G. A simple mesh generator in MATLAB. *SIAM Rev*. 2004;46(2):329-345.
21. Geuzaine C, Remacle JF. Gmsh: a 3-D finite element mesh generator with built-in pre-and post-processing facilities. *Int J Numer Methods Eng*. 2009;79(11):1309-1331.
22. Chen L, Jc X. Optimal Delaunay triangulations. *J Comput Math*. 2004;22(2):299-308.
23. Du Q, Faber V, Gunzburger M. Centroidal Voronoi tessellations: applications and algorithms. *SIAM Rev*. 1999;41(4):637-676.
24. Mullen P, Memari P, de Goes F, Desbrun M. HOT: Hodge-optimized triangulations. *ACM Trans Graph*. 2011. 10.1145/1964921.1964998.
25. Desbrun M, Kanso E, Tong Y. Discrete differential forms for computational modeling. *Discrete Differential Geometry*. Vol 38. Springer; 2008:287-324.
26. Reddy JN. A dual mesh finite domain method for the numerical solution of differential equations. *Int J Comput Methods Eng Sci Mech*. 2019;20(3):212-228. doi:10.1080/15502287.2019.1610987
27. Reddy JN, Nampally P, Srinivasa AR. Nonlinear analysis of functionally graded beams using the dual mesh finite domain method and the finite element method. *Int J Non-Linear Mech*. 2020;127:103575.
28. Kobbelt LP, Botsch M. An interactive approach to point cloud triangulation. *Computer Graphics Forum*. Vol 19. Wiley Online Library; 2000:479-487.
29. Bern M, Eppstein D. Mesh generation and optimal triangulation. *Comput Euclidean Geom*. 1992;1:23-90.
30. Fornberg B, Flyer N. Fast generation of 2-D node distributions for mesh-free PDE discretizations. *Comput Math Appl*. 2015;69(7):531-544.
31. Shankar V, Kirby RM, Fogelson AL. Robust node generation for mesh-free discretizations on irregular domains and surfaces. *SIAM J Sci Comput*. 2018;40(4):A2584-A2608.
32. Slak J, Kosec G. On generation of node distributions for meshless PDE discretizations. *SIAM J Sci Comput*. 2019;41(5):A3202-A3229.
33. Monaghan JJ, Lattanzio JC. A refined particle method for astrophysical problems. *Astron Astrophys*. 1985;149(1):135-143.
34. Ericson C. *Real-Time Collision Detection*. CRC Press; 2004.
35. Violeau D, Leroy A. On the maximum time step in weakly compressible SPH. *J Comput Phys*. 2014;256:388-415.
36. Dinh VQ, Marechal Y. GPU-based parallelization for bubble mesh generation. *COMPEL-Int J Comput Math Electr Electron Eng*. 2017;36(4):1184-1197.
37. Freitag LA, Ollivier-Gooch C. Tetrahedral mesh improvement using swapping and smoothing. *Int J Numer Methods Eng*. 1997;40(21):3979-4002.

How to cite this article: Wang Z, Srinivasa AR, Reddy JN, Dubrowski A. PIMesh: An automatic point cloud and unstructured mesh generation algorithm for meshless methods and finite element analysis–with applications in surgical simulations. *Int J Numer Meth Biomed Engng*. 2022;e3615. doi:10.1002/cnm.3615

APPENDIX A: CALCULATING THE PARTICLE INJECTION POSITIONS FOR 2D MESH DOMAINS

The algorithm to calculate particle injection positions **S** for a 2D domain is shown in Table A1. N_{tri} is the total number of triangles of the 2D domain. A_i and $\mathbf{C}_i$ are the area and centroid of the i_{th} triangle of the 2D domain (see Figure A1). Using the algorithm shown in Table A1, we can obtain the particles' injection positions, $\mathbf{S} = \{\mathbf{S}_1, \mathbf{S}_2, \cdots, \mathbf{S}_{N_s}\}$ (N_s is the number of the injection positions).

TABLE A1 The algorithm to calculate particle injection positions for a 2D domain

1	Let $A_{sum} = 0, j = 1, A_0 = h^2$.	
2	For $i = 1, 2, \cdots, N_{tri}$:	
	2.1	$A_{sum} = A_{sum} + A_i$:
	2.2	If $A_{sum} \geq A_0$: Calculate the centroid of the i_{th} triangle $\mathbf{C}_i$. The centroid $\mathbf{C}_i$ is the j_{th} particle resource, $\mathbf{S}_j = \mathbf{C}_i$. Let $A_{sum} = 0, j = j + 1$.
3	$\mathbf{S} = \{\mathbf{S}_1, \mathbf{S}_2, \cdots, \mathbf{S}_{N_s}\}$ are the $N_s = j - 1$ particle resources.	

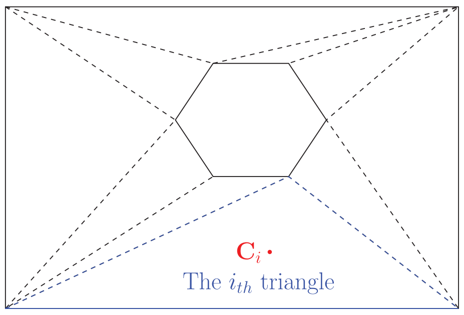

FIGURE A1 The triangular mesh following the OBJ format represents a 2D domain. The solid line segments are the boundary edges. A_i and $\mathbf{C}_i$ are the area and centroid of the i_{th} triangle of the domain respectively

APPENDIX B: CALCULATING THE PARTICLE INJECTION POSITIONS FOR 3D MESH DOMAINS

The algorithm to calculate particle injection positions **S** for a 3D mesh domain is shown in Figure B1. A tetrahedron mesh M_{3d} can be generated based on the vertices of the triangular surface mesh representing the 3D mesh domain using 3D Delaunay triangulation. N_{tet} is the total number of tetrahedrons of the tetrahedron mesh M_{3d}. V_i^{tet} is the volume of the i_{th} tetrahedron. The volume summary of the tetrahedrons of M_{3d} is calculated iteratively. When the summation $\mathrm{V_s}$ is greater equal than $18V_{t0}$ after adding the volume of the i_{th} tetrahedron V_i^{tet}, the centroid $\mathbf{C}_i$ of the tetrahedron is added to the injection positions **S** and $\mathrm{V_s}$ is then set zero ($\mathrm{V_s} = 0$). Repeat this process for all the tetrahedrons, the injection positions $\mathbf{S} = \{\mathbf{S}_1, \mathbf{S}_2, \cdots, \mathbf{S}_{N_s}\}$ is obtained and N_s is the total number of the injection positions of the particles.

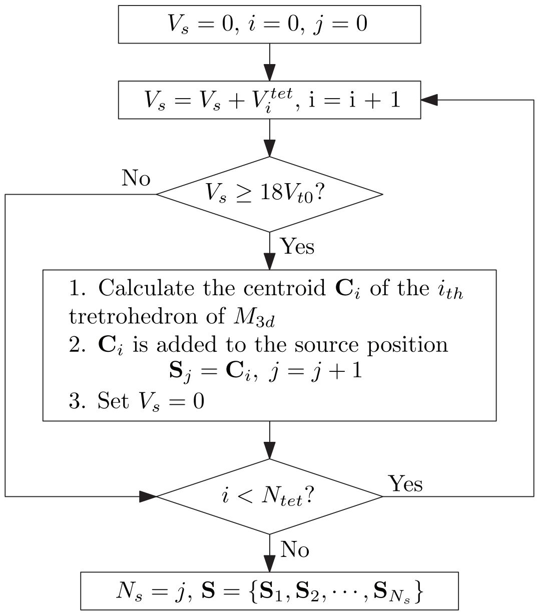
$V_s = 0,\ i = 0,\ j = 0$
$V_s = V_s + V_i^{tet}$, i = i + 1
$V_s \geq 18V_{t0}$?
No
Yes
1. Calculate the centroid $\mathbf{C}_i$ of the i_{th} tretrohedron of M_{3d}
2. $\mathbf{C}_i$ is added to the source position $\mathbf{S}_j = \mathbf{C}_i,\ j = j + 1$
3. Set $V_s = 0$
$i < N_{tet}$?
Yes
No
$N_s = j,\ \mathbf{S} = \{\mathbf{S}_1, \mathbf{S}_2, \cdots, \mathbf{S}_{N_s}\}$

FIGURE B1 The algorithm to calculate particle injection positions. V_i^{tet} is the volume of the i_{th} tetrahedron. V_{t0} is set to equal $V_{t0} = \frac{h^3}{6\sqrt{2}}$. **S** are the injection positions and N_s is the total number of **S**

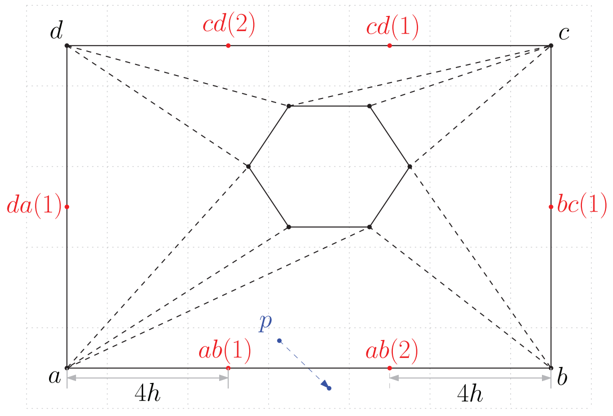

FIGURE C1 The background uniform grid drawn in dotted line segments is the uniform cells for the FCD technique. Extra vertices *ab*(1), *ab*(2), *bc*(1), *cd*(1), *cd*(2), and *da*(1) are added on the boundary edge. Since the boundary vertices *ab*(1) and *ab*(2) are in the neighbor cells of the particle *p*, we can detect that the particle *p* is near the boundary edge E_{ab} of the 2D mesh domain. FCD, fast collision detection

APPENDIX C: ADDING EXTRA BOUNDARY VERTICES FOR 2D AND 3D MESH DOMAINS

C.1 | Adding boundary vertices for 2D mesh domains

The method to add boundary vertices for a 2D mesh domain is demonstrated through an example shown in Figure C1. The background uniform grid drawn in dotted line segments is the uniform cells for the FCD technique. A particle *p* is motioning across the boundary edge E_{ab} of the 2D mesh domain. To detect whether the particle *p* is near the boundary of the 2D mesh domain, we search for the boundary vertices of the 2D mesh domain in the particle *p*'s neighbor cells. However, since there is no boundary vertex in the neighbor cells, we cannot detect that the particle *p* is near the boundary of the mesh domain. Therefore, the particle *p* can motion outside the mesh domain without detection. To solve this kind of problem, extra boundary vertices are added at

$$v_{ab(i)} = v_a + \frac{v_b - v_a}{N_e + 1} \cdot i, N_{ab} = \left\lceil \frac{||v_b - v_a||}{4h} - 1 \right\rceil, \tag{C1}$$

along the edge E_{ab}, where $i = 1, 2, ..., N_{ab}$, N_{ab} the total number of extra vertices added along the edge E_{ab}. As a result, two extra vertices ($N_{ab} = 2$) $ab(1)$ and $ab(2)$ are added on the boundary edge E_{ab}. When the particle p is crossing the edge E_{ab}, the vertices $ab(1)$ and $ab(2)$ are in the particle's neighbor cells and therefore we can use the two extra boundary vertices $ab(1)$ and $ab(2)$ to detect that the particle p is near the boundary of the mesh domain.

Repeating this process for all the boundary edges of which the length is greater than 4 h, extra boundary vertices bc (1), $cd(1)$, $cd(2)$, and $da(1)$ are added onto the boundary edges E_{bc}, E_{cd}, and E_{da}. After adding the extra boundary vertices, since the maximum distance between two boundary vertices on an edge is less equal than 4 h and the uniform cell size for FCD is equal to 2 h, whenever a particle is near the boundary of the mesh domain, there must be at least one boundary vertices in the particle's neighbor cell. Thus, it is guaranteed that a particle can be detected when crossing a boundary edge.

C.2 | Adding boundary vertices for 3D mesh domains

Similar to the 2D case, a particle can motion across the boundary of a 3D mesh domain through a boundary edge or triangle without detection. For example, a particle p can crossing T_{abc} without detection (see Figure C2A). Therefore, extra boundary vertices are added on the boundary edges and boundary triangles.

The method to add extra boundary vertices on the boundary of a 3D mesh domain is discussed through the example shown in Figure C3. E_{ab} is the longest edge and E_{bc} is the shortest. First, extra boundary vertices, such as $\mathbf{v}_{ab(1)}$, $\mathbf{v}_{ab(2)}$ on edge E_{ab}, and $\mathbf{v}_{bc(1)}$ on edge E_{bc}, are added on the boundary edges of the 3D domain using the same method for the 2D case (see Figure C3B). Extra vertices

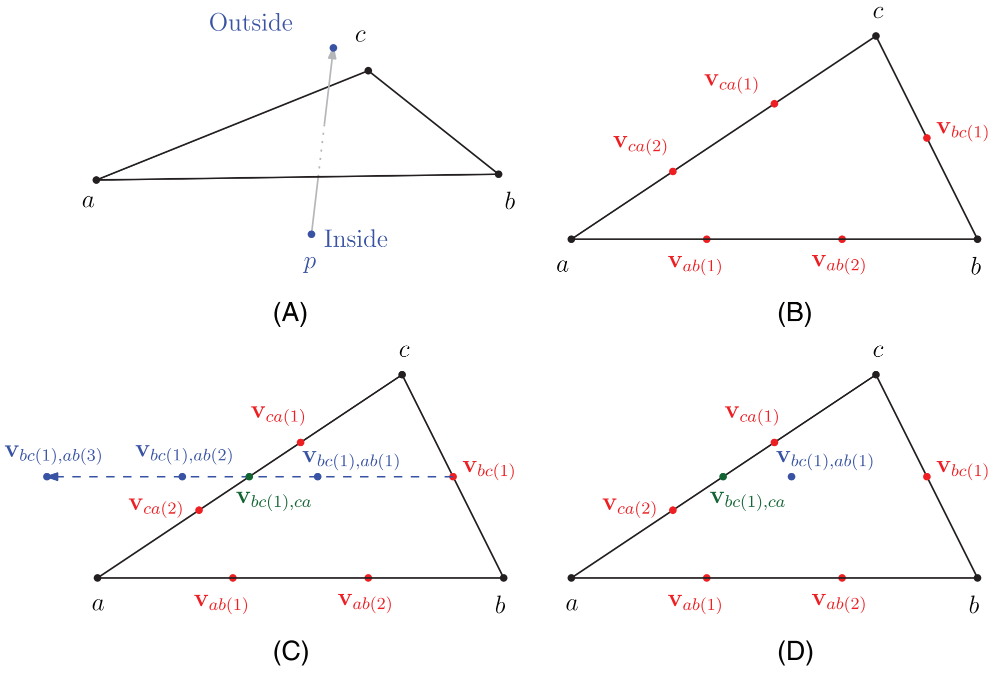

FIGURE C2 The triangle T_{abc} is a boundary triangle of a 3D mesh domain. E_{ab} is the longest edge and E_{bc} is the shortest. (A) A particle p can motion outside the 3D object through the triangle T_{abc} without detection. (B) Extra boundary vertices $\mathbf{v}_{ab(1)}$, $\mathbf{v}_{ab(2)}$, $\mathbf{v}_{bc(1)}$, $\mathbf{v}_{ca(1)}$, and $\mathbf{v}_{ca(2)}$ are added on the edges. (C) The line segment $L_{\mathbf{v}_{bc(1)},\mathbf{v}_{bc(1),ab(3)}}$ are parallel to E_{ab} and $\mathbf{v}_{bc(1),ab(1)}$ and $\mathbf{v}_{bc(1),ab(2)}$ separate $L_{\mathbf{v}_{bc(1)},\mathbf{v}_{bc(1),ab(2)}}$ into three equal pieces. $\mathbf{v}_{bc(1),ca}$ is the intersection vertex between line segment $L_{\mathbf{v}_{bc(1)},\mathbf{v}_{bc(1),ab(3)}}$ and edge E_{ca}. (D) Removing the vertices $\mathbf{v}_{bc(1),ab(2)}$ and $\mathbf{v}_{bc(1),ab(3)}$ that are outside the triangle T_{abc}, the remaining vertices are the extra boundary vertices added on the boundary triangle.

$$v_{bc(i),ab(j)} = v_{bc(i)} + \frac{v_a - v_b}{N_{ab}+1} j, \tag{C2}$$

are then added on the plane of the triangle T_{abc}, where $i = 1, 2, \cdots, N_{bc}$ and $j = 1, 2, \cdots, N_{ab} + 1$; $N_{ab} = 2$ and $N_{bc} = 1$ are the number of extra boundary vertices added on edges E_{ab} and E_{bc} respectively. Therefore, $\mathbf{v}_{bc(1),ab(1)}$, $\mathbf{v}_{bc(1),ab(2)}$, and $\mathbf{v}_{bc(1),ab(3)}$ are added (see Figure C3C). Additionally, the intersection points on edge E_{ca},

$$v_{bc(i),ca} = x_c + \frac{v_a - v_c}{N_{bc}+1} i, \tag{C3}$$

where $i = 1, 2, \cdots, N_{bc}$ (in the case of Figure C3C, there is only one intersection point $\mathbf{v}_{bc(1),ca}$). Removing the vertices outside the triangle T_{abc} (such as the $\mathbf{v}_{bc(1),ab(1)}$ and $\mathbf{v}_{bc(1),ab(2)}$) and the repeated vertices, the remaining vertices include three types: (1) vertices on edges such as $\mathbf{v}_{ab(1)}$, $\mathbf{v}_{ab(2)}$, $\mathbf{v}_{bc(1)}$, $\mathbf{v}_{ca(1)}$, and $\mathbf{v}_{ca(2)}$; (2) vertices inside the triangle, such as $\mathbf{v}_{bc(1),ab(1)}$; (3) intersection points on edge E_{ca}, such as $\mathbf{v}_{bc(1),ca}$ (see Figure C3D). Repeating this process for all the boundary triangles, we can obtain all the extra boundary vertices on the boundary edges and surfaces of the 3D mesh domain. In this way, whenever a particle is crossing the boundary surface of the 3D mesh domain, it can be detected since there is at least one boundary vertex located in the particle's neighbor cells.

APPENDIX D: DETERMINING THE LOCATION OF A PARTICLE

D.1 | 2D mesh domain

The algorithm to determine whether a particle inside the 2D mesh domain is illustrated through an example shown in Figure D1. $\mathbf{n}_{ab}$ and $\mathbf{n}_{bc}$ are the normal vectors of the boundary edges E_{ab} and E_{bc} of a 2D mesh domain. Points a, b, c, and d are the boundary vertices. $\mathbf{n}_b$ is the normal vector of the boundary vertex b. particles p_i ($i = 1, 2, 3, 4$) motion near the boundary of domain. To determine whether the particle p_i inside the mesh domain, we first obtain the projection points p_i' onto the mesh domain boundary, and then the projection points' normal vector, which is equal to the normal vector of the boundary edges or vertices where the projection points locate. For example, since p_1' is on the edge E_{bc}, the normal vector on p_1 is $\mathbf{n}_{p_1'} = \mathbf{n}_{bc}$. Similarly, we can obtain $\mathbf{n}_{p_3'} = \mathbf{n}_{bc}$, $\mathbf{n}_{p_4'} = \mathbf{n}_{ab}$, and $\mathbf{n}_{p_2'} = \mathbf{n}_b$. Then we compare the projection points' normal vector $\mathbf{n}_{p_i'}$ and the vector $\mathbf{v}_{p_i'p_i} = \mathbf{x}_{p_i} - \mathbf{x}_{p_i'}$ from the projection point p_i' to the particle p_i.

- If $\mathbf{v}_{p_i'p_i} \cdot \mathbf{n}_{bc} < 0$, the particle p_i (e.g., the particle p_1) is inside the mesh domain.
- If $\mathbf{v}_{p_i'p_i} \cdot \mathbf{n}_{bc} > 0$, the particle p_i (e.g., the particle p_2 and p_3) is outside the mesh domain.
- If $\mathbf{v}_{p_i'p_i} \cdot \mathbf{n}_{bc} = 0$, the particle p_i (such as the particle p_4) is on the boundary of the mesh domain.

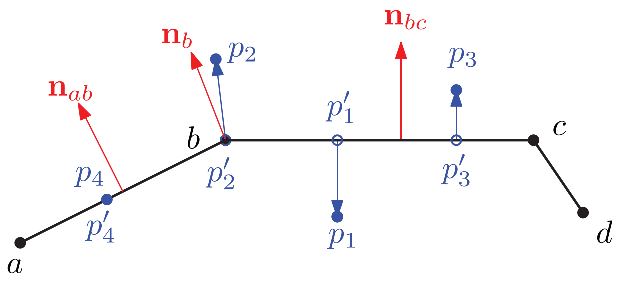

FIGURE D1 The edges E_{ab}, E_{bc}, and E_{cd} are boundary edges of a 2D mesh domain. Points a, b, c, and d are the boundary vertices. p_1 is a particle inside the domain; p_2 and p_3 are two particles outside the domain; p_4 is a particle on the boundary of the mesh domain. p_1', p_2', p_3', and p_4' are the four projection points of p_1, p_2, p_3, and p_4 on the domain boundary respectively. The position of p_2' is the same as the position of the vertex b. Since p_4 is on the domain boundary, p_4' and p_4 are at the same location. $\mathbf{n}_{ab}$ and $\mathbf{n}_{bc}$ are the normal vectors of the boundary edges E_{ab} and E_{bc} respectively. $\mathbf{n}_b$ is the normal vector of the vertex b

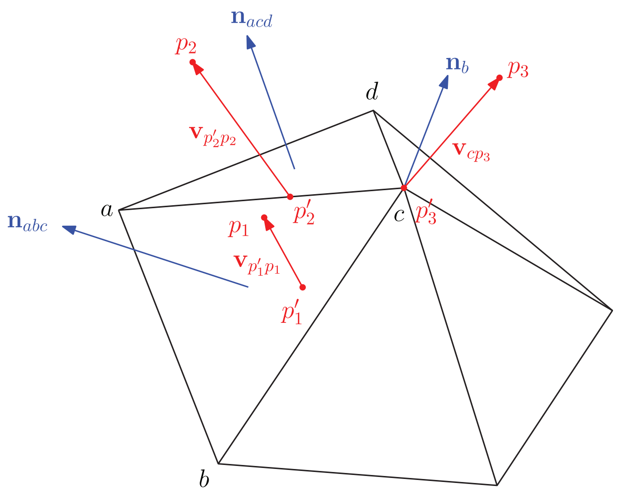

FIGURE D2 The triangles are parts of a 3D object boundary. p_1, p_2, and p_3 are three particles outside the object. p'_1, p'_2, and p'_3 are the three projection points of p_1, p_2, and p_3 on the object respectively (the position of p'_3 is the same as the position of the vertex c). $\mathbf{v}_{p'_ip_i} = \mathbf{x}_{p_i} - \mathbf{x}_{p'_i}$ ($i = 1, 2, 3$) are three vectors from the projection points to the particles. $\mathbf{n}_{abc}$ and $\mathbf{n}_{acd}$ are the normal vector of triangles T_{abc} and T_{acd} respectively. $\mathbf{n}_c$ is the normal vector of the vertex c on the object

D.2 | 3D mesh domain

The method to project particles outside the 3D mesh domain onto its boundary is similar to the method for the 2D mesh domain in the previous part. When a particle p_i is near the boundary, we should obtain the projection point p'_i on the 3D mesh domain and the projection points' normal vector. As shown in Figure D2, a particle's projection point can be on a triangle surface, a boundary edge, or a boundary vertex of the mesh domain. p'_1 is the projection point of the particle p_1, and the normal of p'_1 is the surface normal vector of triangle T_{abc} ($\mathbf{n}_{p'_1} = \mathbf{n}_{abc}$). p'_2 on the edge E_{ac} is the projection point of the particle p_2, and the normal vector of p'_2 is set as the edge normal vector $\mathbf{n}_{ac} = \frac{\mathbf{n}_{abc}+\mathbf{n}_{acd}}{\|\mathbf{n}_{abc}+\mathbf{n}_{acd}\|}$ ($\mathbf{n}_{p'_2} = \mathbf{n}_{ac}$). The boundary vertex c is the projection p'_3 of the particle p_3, and normal vector of the p'_3 is equal to the normal vector of the boundary vertex c ($\mathbf{n}_{p'_3} = \mathbf{n}_c$).

Then we compare the vector $\mathbf{v}_{p'_ip_i} = \mathbf{x}_{p_i} - \mathbf{x}_{p'_i}$ to the normal $\mathbf{n}_{p'_i}$ assigned for the projection point p'_i.

- If $\mathbf{v}_{p'_ip_i} \cdot \mathbf{n}_{p'_i} < 0$, the particle p_i is inside the 3D mesh domain.
- if $\mathbf{v}_{p'_ip_i} \cdot \mathbf{n}_{p'_i} > 0$, the particle p_i is outside the 3D mesh domain. For example, p_1, p_2, and p_3 are outside the mesh domain.
- if $\mathbf{v}_{p'_ip_i} \cdot \mathbf{n}_{p'_i} = 0$, the particle p_i is on the boundary surface of the 3D mesh domain.

APPENDIX E: CONSTRUCT DISCRETE MESH SIZE FUNCTIONS

To easily make the algorithms work for a mesh domain in any shape, the mesh size function is constructed in a discrete format and illustrated through an example shown in Figure E1. The shape drawn in solid line segments is the boundary of a 2D mesh domain. Assume that the mesh element size at the points 1, 2, 3, 4, and 5 are set as h_i ($i = 1, 2, 3, 4, 5$) respectively. h_{min} is the minimum mesh element size, $h_{min} = \min\{h_1, h_2, h_3, h_4, h_5\}$. To construct the discrete mesh size function, a bounding box $B_{b_1b_2b_3b_4}$ covering all the 2D mesh domain is set and discretized into a uniform mesh grid (represented by dashed line segments) with gird size equal to $h_{min}/4$. The element size at the four vertices b_1, b_2, b_3, and b_4 are set as h_{min}. A background triangular mesh drawn in dotted line segments is generated based on the positions of the four vertices of the bounding box $B_{b_1b_2b_3b_4}$ and the positions of points (point 1, 2, 3, 4, and 5) where element sizes are assigned.

With the background triangular mesh, the target mesh element size can be assigned for each cell of the uniform mesh grid. For example, the element size of the cell $C_{c_{56}c_{57}c_{67}c_{66}}$ is set as h_5 since the point 5 is inside the cell $C_{c_{56}c_{57}c_{67}c_{66}}$.

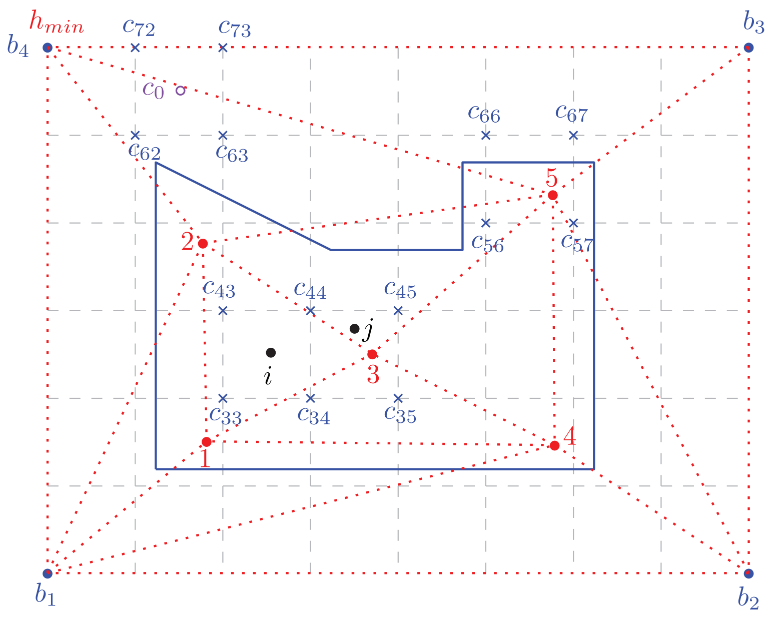

FIGURE E1 The shape drawn in solid line segments is the 2D object for which the mesh will be generated. The box $B_{b_1b_2b_3b_4}$ is the bounding box for the 2D object. h_i where $i = 1, 2, 3, 4, 5$ are the target edge length at the points 1, 2, 3, 4, and 5 respectively. $h_{min} = \min\{h_1, h_2, h_3, h_4, h_5\}$ is set as the target edge length at the four vertices of the bounding box $B_{b_1b_2b_3b_4}$. The dashed line segments slice the bounding box $B_{b_1b_2b_3b_4}$ into uniform cells, and the c_0 is the centroid of the cell $C_{c_{56}c_{57}c_{67}c_{66}}$. A triangular mesh drawn in dotted line segments is generated based on the positions of points where target edge length are given. Points i and j are two particles at $\mathbf{x}_i$ and $\mathbf{x}_j$ respectively.

For other cells, such as $C_{c_{62}c_{63}c_{73}c_{72}}$, the element size is not directly assigned. We can calculate the element size for these cells using linear interpolation. c_0 is the centroid of the cell $C_{c_{62}c_{63}c_{73}c_{72}}$ and is inside the triangle $T_{b_4h_2h_5}$ of the background triangular mesh. Since $\mathbf{x}_{c_0} = \alpha\mathbf{x}_{b_4} + \beta\mathbf{x}_{h_2} + \gamma\mathbf{x}_{h_5}$, the element size for the cell $C_{c_{62}c_{63}c_{73}c_{72}}$ is set as

$$h(c_{62}c_{63}c_{73}c_{72}) = \alpha h_{min} + \beta h_2 + \gamma h_5.$$

Repeating this process, we can obtain the target mesh element size for the uniform mesh grid. The discrete mesh size function can then be defined as

$$h_d(x) = h(C_{id}(x)), \tag{E1}$$

where $C_{id}(\mathbf{x})$ is the ID of the cell where $\mathbf{x}$ locates. For example, the target mesh size for the i_{th} and j_{th} particle in Figure E1 is $h_d(\mathbf{x}_i) = h(c_{33}c_{34}c_{44}c_{43})$ and $h_d(\mathbf{x}_j) = h(c_{34}c_{35}c_{45}c_{44})$ respectively.