

RESEARCH ARTICLE

An algorithm for triangulating smooth three-dimensional domains immersed in universal meshes

Ramsharan Rangarajan¹  | Hardik Kabaria² | Adrian Lew³

¹Department of Mechanical Engineering, Indian Institute of Science Bangalore, Bengaluru, India

²Carbon, Inc, Redwood City, California

³Department of Mechanical Engineering, Stanford University, Stanford, California

Correspondence

Ramsharan Rangarajan, Department of Mechanical Engineering, Indian Institute of Science Bangalore, Bengaluru-560 012, India.

Email: rram@iisc.ac.in

Funding information

Science and Engineering Research Board, Grant/Award Number: ECR/2017/000346; National Science Foundation's Civil, Mechanical and Manufacturing Innovation, Grant/Award Number: 1301396 and 1662452; Army Research Grant, Grant/Award Number: W911NF-07-2-0027

Summary

We describe an algorithm to recover a boundary-fitting triangulation for a bounded C^2 -regular domain $\Omega \subset \mathbb{R}^3$ immersed in a nonconforming background mesh of tetrahedra. The algorithm consists in identifying a polyhedral domain ω_h bounded by facets in the background mesh and morphing ω_h into a boundary-fitting polyhedral approximation Ω_h of Ω . We discuss assumptions on the regularity of the domain, on element sizes and on specific angles in the background mesh that appear to render the algorithm robust. With the distinctive feature of involving just small perturbations of a few elements of the background mesh that are in the vicinity of the immersed boundary, the algorithm is designed to benefit numerical schemes for simulating free and moving boundary problems. In such problems, it is now possible to immerse an evolving geometry in the same background mesh, called a universal mesh, and recover conforming discretizations for it. In particular, the algorithm entirely avoids remeshing-type operations and its complexity scales approximately linearly with the number of elements in the vicinity of the immersed boundary. We include detailed examples examining its performance.

KEYWORDS

background meshes, evolving boundary, immersed boundary, mesh relaxation, moving mesh, 3D meshing

1 | INTRODUCTION

The vagaries involved in meshing complex three-dimensional (3D) geometries have led to a proliferation of semi-manual workflows using open source and commercial software packages alike. These difficulties are only exacerbated when simulating problems with evolving/moving domains, where the evolution of the geometry may be dictated by the physics of the phenomenon being approximated,¹ or as realizations in a shape/topology optimization procedure,² for example. The large volume of literature on meshless methods,³ phase field methods,⁴ immersed boundary methods,⁵⁻⁷ fictitious domain methods,⁸ extended finite element methods,⁹ and arbitrary Lagrangian-Eulerian methods¹⁰ is representative of the emphasis on ideas aimed at circumventing the meshing problem.

Here, we describe a fast, simple and ostensibly robust algorithm for discretizing 3D domains realized when simulating problems with evolving boundaries. The algorithm, henceforth referred to as “um” or as the “um algorithm,” consists in mapping a specific collection of elements in a tetrahedral background mesh onto a boundary-conforming triangulation for each domain realized in a simulation. Notably, um computes a boundary-conforming mesh by only perturbing background mesh vertices lying in a narrow neighborhood of the immersed boundary. Unwieldy operations such as cutting/trimming of cells, mesh untangling and local remeshing are prudently bypassed.

Transforming the task of meshing a domain (Ω) into one of constructing a bijective mapping (M_h) over a subset of elements (ω_h) in a background mesh (\mathcal{T}) provides an appealing perspective to the meshing problem. Although not new,^{11,12} designing robust methods for constructing such mappings remains challenging. In particular, injectivity of M_h is essential to avoid degenerate, inverted and overlapping elements but is difficult to establish a priori. Lack of injectivity is, however, easily detected a posteriori and is usually rectified using intricate mesh repair operations.^{13,14} The key choices in the mapping that we identify are (i) the collection of elements ω_h in the background mesh selected to be mapped onto the immersed domain, (ii) the prescription for M_h over $\partial\omega_h$ as the closest point projection onto $\partial\Omega$ and (iii) the extension of M_h to the interior of ω_h using an optimization-based vertex relaxation algorithm.

There is considerable freedom in choosing each of the three ingredients mentioned above. For instance, Laplacian smoothing and its variants¹⁵ or physics-based algorithms^{16–18} are widely used for the purpose of mesh relaxation. Mesh moving methods,¹⁹ mesh adaption methods²⁰ and optimization-based smoothing algorithms²¹ provide more computationally intensive alternatives. The directional vertex relaxation algorithm (dvr) adopted here provides a clear advantage—a guarantee of monotonic improvement in mesh quality with each vertex perturbation.²² The dvr algorithm only requires the solution of robustly resolvable one-dimensional max-min problems for vertex updates and is amenable to efficient parallelization.

The more challenging aspects of the um algorithm lie in selecting ω_h and defining an injective correspondence between the boundaries $\partial\omega_h$ and $\partial\Omega$. While constructive methods to define homeomorphisms between regular surfaces are available,²³ the lack of any conformity of the background mesh to the immersed boundary and the faceted constitution of $\partial\omega_h$ render our task particularly challenging. The idea of setting the correspondence between $\partial\omega_h$ and $\partial\Omega$ to equal the closest point projection of the latter, while concurrently restricting certain angles and element sizes in the background mesh, is inspired by the two-dimensional (2D) analog of the um algorithm^{24,25} and a preliminary examination in three dimensions.²⁶ The conditions we require of the background mesh are hardly restrictive owing to the conspicuous absence of any conformity requirements and should be contrasted with algorithms explicitly tailored for background meshes with specific structure.^{27–30} We conjecture, based on the discussion in Section 5 and an extensive set of numerical experiments²⁶ including ones shown here that the restrictions identified on the regularity of the domain, and on dihedral angles and sizes of elements in the background mesh near the immersed boundary suffices for the closest point projection $\pi : \partial\omega_h \rightarrow \partial\Omega$ to be a homeomorphism. Such a result is required to ensure a priori, the injectivity of M_h over ω_h . We briefly discuss intermediate results proved toward this end in Section 5.

We emphasize that um is contrived to serve numerical schemes for simulating moving boundary problems where remeshing an evolving domain entirely at each solution iteration is impractical even with a good mesh generator. The um algorithm facilitates automatic, fast, local and ostensibly robust mesh updates for evolving domains, while permitting flexible boundary representations with the caveat of assuming sufficient smoothness. In principle and in practice, the algorithm is as easy to use for meshing a complex domain (ie, domains whose boundaries have varied geometric features), as it is for meshing a simple one. Unlike arbitrary Lagrangian-Eulerian methods or mesh morphing techniques,^{31,32} accommodating large boundary motions requires no special considerations. The examples included in Section 4, all involving domains with evolving boundaries, amply showcase this feature of um. The domain considered in the example in Section 4.2 additionally undergoes topological changes. In Section 4.3, we demonstrate the application of um for computing the motion of a deformable self-propelling object, ie, a swimmer, where a finite element method is used to compute the flow solution in a fluid domain that changes at each time step. In such calculations, the fact that um retains the connectivity of the background mesh enables using data structures with fixed sparsity patterns. These advantages of um are unparalleled by other commonly used mesh generators such as TetGen,³³ Gmsh,³⁴ CGAL,³⁵ and HyperMesh,³⁶ to name a few.

On the other hand, restrictions on the regularity of the immersed boundary renders um less versatile than many existing meshing algorithms.^{28,34,35,37,38} Hence, for the purpose of meshing a *given* domain, when the aforementioned features of um are unlikely to be of concern, Delaunay-based meshing algorithms are likely to be more suitable.^{39,40} Although there is a large volume of literature on this topic, we highlight the work of Oudot et al⁴¹ in this context for the guarantees it provides for meshing domains bounded by smooth surfaces with tetrahedral elements. In Section 3.5, we compare the performance of um with TetGen, Gmsh, CGAL, and HyperMesh for triangulating smooth domains. Details of the comparison, namely, the input provided to each algorithm and the values of specific algorithmic parameters chosen are important to scrutinize before interpreting the results and are discussed in detail in Section 3.5. Here, we only mention that the surface mesh (skin) of the triangulations computed with um and those generated with TetGen, Gmsh, and HyperMesh are all identical. The table in Figure 1 lists the poorest element quality in the meshes computed with each mesh generator. The performance of um clearly stands out. In this sense, even for the purpose of meshing a given smooth domain, um is a useful alternative to existing meshing algorithms.

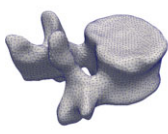
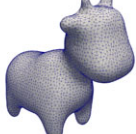
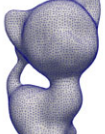
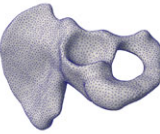
Mesh generator					Remarks
UM	0.660	0.663	0.669	0.634	See Algorithm 1.
TetGen	0.0755	0.0764	0.0861	0.0712	Input: surface mesh of UM. Options: <code>-q</code> command line switch.
Gmsh	0.340	0.262	0.327	0.253	Input: surface mesh of UM. Options: <code>NetGen</code> mesh improvement.
HyperMesh	0.268	0.369	0.242	0.248	Input: surface mesh of UM. Options: <code>Volume Tetra</code> , <code>mesh size</code> .
CGAL	0.205	0.193	0.202	0.209	Input: identical implicit function as UM. Options: <code>facet_angle</code> , <code>facet_size</code> , <code>cell_radius_edge_ratio</code> , <code>cell_size</code> , <code>facet_distance</code> .

FIGURE 1 A comparison of the poorest element quality, measured with the mean ratio metric, in meshes computed with TetGen, Gmsh, CGAL, HyperMesh, and the proposed um algorithm. Specific parameters used for the comparison are mentioned in the table and additional details are discussed in Section 3.5. In each of the four examples, we find that um returns meshes of better quality than the alternative mesh generators [Colour figure can be viewed at wileyonlinelibrary.com]

The algorithm introduced here shares some similarities with octree-based methods for meshing,⁴²⁻⁴⁷ where leaves of the octree intersected by the boundary of the domain are warped or deformed to conform to the domain. These leaves are then tetrahedralized using stencils or Delaunay triangulations. These methods can generate adaptively refined meshes for very general domains. On the other hand, for problems with moving domains, the connectivity of the mesh can change drastically between successive time steps due to the local tetrahedralization near the boundary as the domain moves.

The restriction of the closest point projection of $\partial\Omega$ to the boundary of ω_h and the dvr algorithm have been previously studied.^{22,26} Among our main contributions here is an integration of these two ideas to realize a meshing algorithm that is arguably more useful than either one of its ingredients themselves. The alliance is nontrivial in part because even though both define vertex perturbations, one is an explicit prescription for vertices on the boundary of ω_h while the other is an optimization-based iterative computation for vertices in the interior of ω_h . More precisely, the um algorithm projects boundary vertices of ω_h onto $\partial\Omega$ in multiple passes, while relaxing nearby interior vertices to accommodate the resulting element perturbations during each pass. Additional constrained relaxation of boundary vertices further improves element qualities in the final mesh. The result is an algorithm that is straightforward to implement, that is devoid of any case-by-case analysis of element-boundary intersections and one that is agnostic to the complexity of the immersed geometry. The set of numerical experiments included provides compelling evidence of the robustness of the um algorithm and of its applicability in numerical schemes for simulating moving boundary problems.

2 | THE UM ALGORITHM

We devote this section to providing a description of the um algorithm. We begin with a succinct and intuitive summary in Section 2.1 to convey the main ideas and to introduce the terminology that we will need. The detailed step-by-step algorithm follows in Section 2.2. The dvr algorithm is a crucial ingredient in um and is described in Section 2.2.3. Detailed examples follow in Section 3.

2.1 | The algorithm in a nutshell

To simplify visualizing the sequence of mesh perturbations involved in um, we show a 2D example of a circular domain immersed in a nonconforming mesh of equilateral triangles in Figure 2. The steps to recover a conforming discretization for the domain depicted in the figure apply verbatim to the 3D case. Specifically, let $\Omega \subset \mathbb{R}^3$ be a bounded open set with a nonempty interior and boundary $\partial\Omega$. We assume that Ω is immersed in a background mesh \mathcal{T} of tetrahedra (tets),

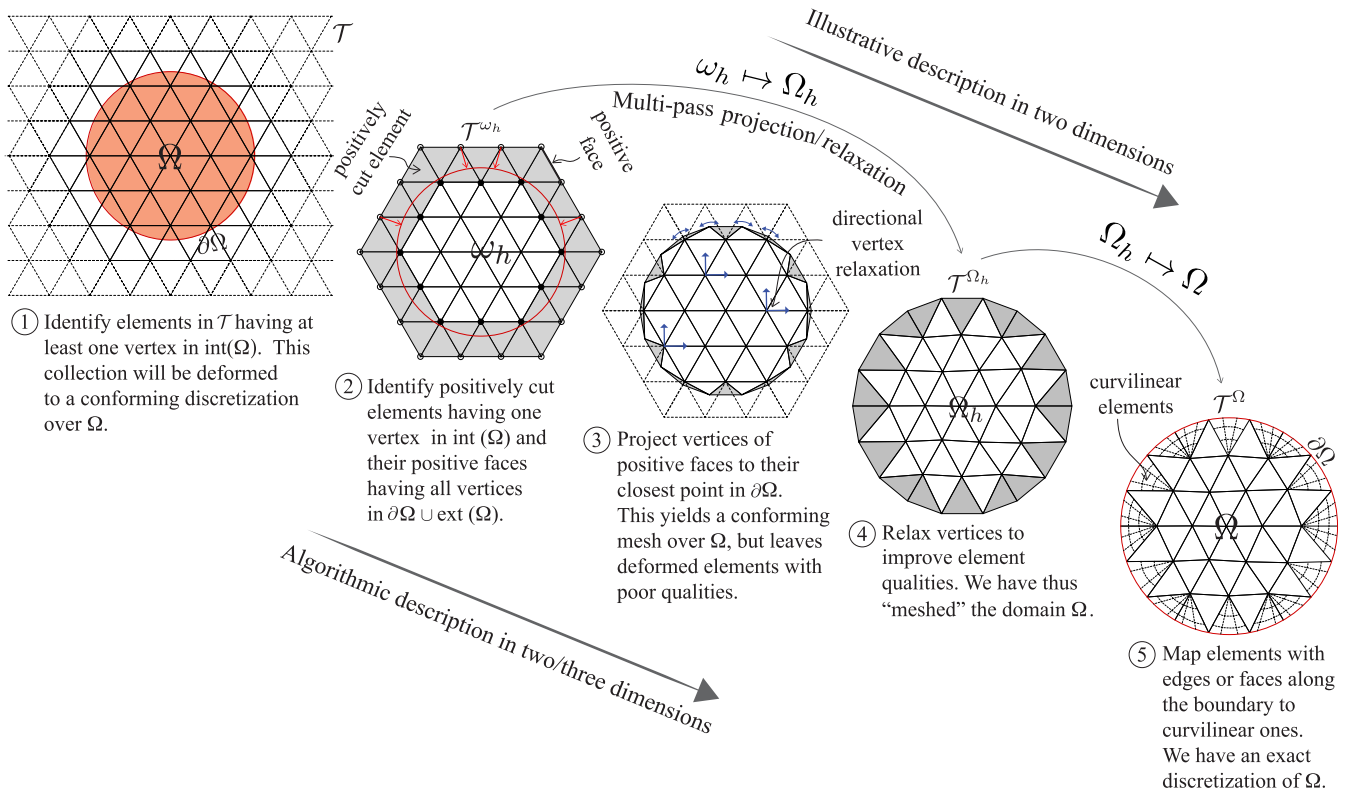


FIGURE 2 An illustration of the main ideas in the um algorithm for recovering a conforming discretization for a curved domain immersed in a background mesh. The images show the two-dimensional version of the algorithm²⁵ for the sake of easy visualization. The description applies verbatim to the three-dimensional case, which is the subject of this article [Colour figure can be viewed at wileyonlinelibrary.com]

by which we mean that the set triangulated by \mathcal{T} strictly contains the closure of Ω . The um algorithm consists of the following steps.

- (i) Identify tets in \mathcal{T} having at least one vertex in Ω . Label this collection of elements as the mesh \mathcal{T}^{ω_h} , which triangulates a (open) domain ω_h having a faceted boundary. In subsequent steps, we map ω_h onto a boundary fitting approximation of Ω .
- (ii) A tet is *positively cut* if it has precisely one vertex in Ω ; its face opposite to the interior vertex is called a *positive face*. Identify the collection of positive faces in \mathcal{T}^{ω_h} and denote their union by the set Γ_h^+ . The collection of indices of vertices of positive faces is denoted by I^+ . By definition, these vertices lie in the complement $\mathbb{R}^3 \setminus \Omega$. We shall refer to these vertices as *positive vertices*.
- (iii) Over multiple passes, incrementally project positive vertices to their respective closest points on the boundary $\partial\Omega$. To accommodate the motion of positive vertices in each pass, relax nearby interior vertices of ω_h with the dvr algorithm. The cumulative result of projecting and relaxing vertices in \mathcal{T}^{ω_h} yields a mesh $\hat{\mathcal{T}}^{\Omega_h}$ over a faceted approximation Ω_h of Ω . We have thus “meshed” the domain Ω with tets. The perturbations of vertices in \mathcal{T}^{ω_h} effectively defines a continuous, piecewise affine function that maps $\omega_h \mapsto \Omega_h$.
- (iv) To further improve qualities of elements near $\partial\Omega$, relax the boundary vertices in $\hat{\mathcal{T}}^{\Omega_h}$ while constraining them to remain on $\partial\Omega$. The final mesh computed by the algorithm is labeled \mathcal{T}^{Ω_h} .

In summary, the um algorithm is a composition of small and local vertex perturbations, each intended to enforce conformity of \mathcal{T}^{Ω_h} to $\partial\Omega$ and to maintain good element qualities. The algorithm is deemed to have succeeded if during the process of transforming \mathcal{T}^{ω_h} to a boundary-conforming mesh over Ω , no intermediate mesh iterate realized has degenerate, inverted or overlapping elements. In such a case, the final conforming mesh \mathcal{T}^{Ω_h} computed by the algorithm does not require any “repairs.” As indicated in Algorithm 1, two assumptions appear to be crucial to this end. The first is that \mathcal{T} is sufficiently refined along the immersed boundary. Second, specific dihedral angles in each positively cut tet, called its conditioning angles and illustrated using an example in Figure 3C, are strictly acute. We postpone a discussion of these conditions until Section 5. We will, however, pay heed to these assumptions when choosing the background mesh \mathcal{T} in numerical experiments using Algorithm 1. In particular, a simple way to satisfy the requirement on conditioning angles

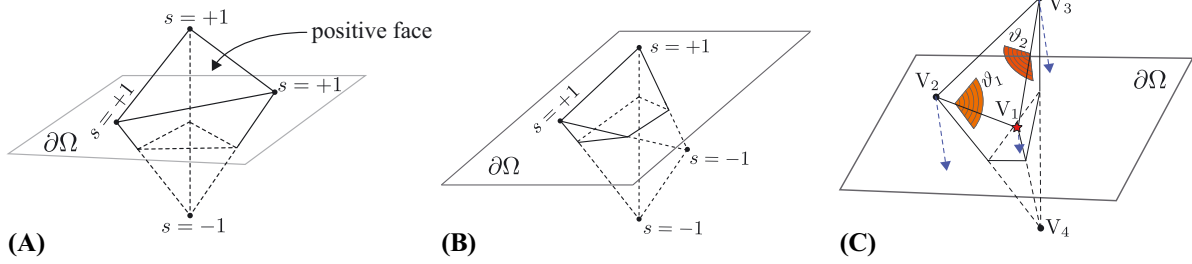


FIGURE 3 A positively cut tet and its positive face are shown in (A). The tet in (B) is not positively cut because it has two vertices in $\text{int}(\Omega)$. Both of the tets in (A) and (B) belong to \mathcal{T}^{ω_h} because they have at least one vertex in $\text{int}(\Omega)$. The variable s indicated in the figure has value -1 in Ω and equals $+1$ elsewhere. It can be associated, for instance, with the sign of the signed distance function to $\partial\Omega$. Figure (C) shows the proximal vertex and conditioning angles of a positively cut tet K having positive face $\overline{V_1V_2V_3}$. Since V_1 is closer to $\partial\Omega$ than V_2 and V_3 , V_1 is called the proximal vertex of K . The interior dihedral angles ϑ_1, ϑ_2 between the positive face and the faces $\overline{V_1V_2V_4}, \overline{V_1V_3V_4}$ respectively are called the conditioning angles of K [Colour figure can be viewed at wileyonlinelibrary.com]

is to ensure that all dihedral angles in positive cut tets in \mathcal{T} are smaller than 90° . This, in turn, is automatic if tets in the vicinity of the immersed boundary have all strictly acute dihedral angles (henceforth acute tets). In fact, in all our examples, we adopt background meshes having all acute tets. Consequently, the conditioning angle requirement on \mathcal{T} is satisfied a priori, independently of the domain immersed in it and of the specific placement of the domain relative to \mathcal{T} .

Figure 2 additionally shows a step mapping elements along the boundary in the rectilinear mesh \mathcal{T}^{Ω_h} to curved ones in order to achieve exact conformity with Ω . For brevity sake, we omit a discussion on constructing mappings to curved elements, which can be accomplished using transfinite interpolation methods.^{48–50} Alternatively, isoparametric maps can be adopted to systematically improve the approximation of curved boundaries.⁵¹

2.2 | Detailed description of the algorithm

Closely following the visual outline in Figure 2, Algorithm 1 provides a step-by-step description of um. The algorithm takes as input a background mesh \mathcal{T} and the domain Ω immersed in it. The mesh \mathcal{T} is a collection of closed tets, say, N of them $\{K_i\}_{i=1}^N$, with the usual properties that, for each $1 \leq i \leq N$, $\text{int}(K_i) \neq \emptyset$, and for each j different than i , $\text{int}(K_i) \cap \text{int}(K_j) = \emptyset$ and $K_i \cap K_j$ is either empty, a vertex, an edge or a face.⁵² It is convenient to identify \mathcal{T} with the triplet (V, I, C) , where I is the collection of nodal indices, V is the collection of Cartesian coordinates of the nodes and C is the collection of element connectivities of the form (i_1, i_2, i_3, i_4) modulo cyclic permutations where $i_1, \dots, i_4 \in I$. An element K in \mathcal{T} is denoted by $K \in \mathcal{T}$ and the Cartesian coordinates of a node with index $i \in I$ by $V_i \in \mathbb{R}^3$.

2.2.1 | The set ω_h

The first task in the algorithm is to identify the submesh \mathcal{T}^{ω_h} and the set I^+ of positive vertices. This is accomplished between steps 1 and 22 by looping over the set of all tets in \mathcal{T} and identifying each element's vertices that lie inside Ω and ones lying in the complement. Tets having at least one vertex in Ω belong to \mathcal{T}^{ω_h} , and among them, tets having precisely one vertex in Ω are positively cut, see Figure 3. Vertices of positive faces in positively cut tets are appended to I^+ . At this stage of the algorithm, the relationship between \mathcal{T} and \mathcal{T}^{ω_h} is exemplified by the fact that $I^{\omega_h} \subseteq I$, $C^{\omega_h} \subseteq C$ and $V^{\omega_h} \subseteq V$, with $V_i^{\omega_h} = V_i$ for each $i \in I^{\omega_h}$. In this sense, \mathcal{T}^{ω_h} is a “submesh” or a “subtriangulation” of \mathcal{T} . In Section 3.8, we mention an alternative “crawling” algorithm to identify \mathcal{T}^{ω_h} that avoids examining each element in \mathcal{T} as done in Algorithm 1.

The choice of ω_h is reminiscent of domain approximation schemes in immersed boundary methods.⁶ To help compare/distinguish the definition of ω_h from such schemes, we highlight a few observations.

- (i) A tet K in \mathcal{T} that is intersected by the boundary $\partial\Omega$ may or may not be included in \mathcal{T}^{ω_h} . For instance, it is possible that none of the vertices of K lie in Ω and yet $K \cap \partial\Omega \neq \emptyset$. While $K \in \mathcal{T}^{\omega_h} \Rightarrow K \cap \Omega \neq \emptyset$ by definition of \mathcal{T}^{ω_h} , $K \cap \partial\Omega \neq \emptyset$ does not imply that $K \in \mathcal{T}^{\omega_h}$ in general.
- (ii) Tets in \mathcal{T}^{ω_h} can be identified without ever computing intersections of tets/faces/edges with the boundary $\partial\Omega$. For instance, if Ω is specified implicitly, say as $\Omega = \{x \in \mathbb{R}^3 : \psi(x) < 0\}$, then it suffices to evaluate the sign of ψ at the vertices of \mathcal{T} to identify \mathcal{T}^{ω_h} .
- (iii) In general, it is not possible to guarantee either that $\Omega \subseteq \omega_h$ or that $\omega_h \subseteq \Omega$.
- (iv) When \mathcal{T} is sufficiently refined in the vicinity of $\partial\Omega$, ω_h can be thought of as an $\mathcal{O}(h)$ approximation of Ω , where h is a representative mesh size of \mathcal{T} . This is because the Hausdorff distance between Ω and ω_h is bounded by the

Hausdorff distance between their boundaries $\partial\omega_h$ and $\partial\Omega$, which, in turn, is bounded by $\max\{\text{diameter}(K) : K \in \mathcal{T}, K \cap \partial\Omega \neq \emptyset\}$.

Algorithm 1: UM: Mesh domain Ω immersed in a background tet mesh \mathcal{T}

Input:

$\mathcal{T} = (V, I, C)$: Tet mesh ▷ Assume: sufficient refinement, acute conditioning angles
 Ω : Domain immersed in \mathcal{T} , bounded open set in \mathbb{R}^3 ▷ Assume: C^2 -regularity
 N_R : Number of projection passes ▷ Assume: Sufficiently many
 N_P : Number of relaxation iterations ▷ Assume: Sufficiently many
rdirfunc: User-defined function specifying relaxation directions

```

1  $I^+, V^{\omega_h}, I^{\omega_h}, C^{\omega_h} \leftarrow \emptyset$  ▷ Identify submesh  $\mathcal{T}^{\omega_h}$  and positive vertices  $I^+$ 
2 for  $K \in \mathcal{T}$  ▷ See Section 3.8 for an alternative algorithm to identify  $\mathcal{T}^{\omega_h}$  without a
   loop over all tets in  $\mathcal{T}$ 
3 do
4    $\{i_1, i_2, i_3, i_4\} \leftarrow$  vertices of  $K$ 
5    $n_K^\pm \leftarrow 0$  and  $I_K^+ \leftarrow \emptyset$ 
6   for  $j = 1$  to 4 do
7     if  $V_{i_j} \in \Omega$  then
8        $n_K^- = n_K^- + 1$ 
9     else
10       $n_K^+ = n_K^+ + 1$  and  $I_K^+ = I_K^+ \cup \{i_j\}$ 
11    end
12  end
13  if  $n_K^- \geq 1$  then
14     $I^{\omega_h} = I^{\omega_h} \cup \{i_1, i_2, i_3, i_4\}$  ▷  $K$  is in  $\mathcal{T}^{\omega_h}$ 
15     $C^{\omega_h} = C^{\omega_h} \cup \{(i_1, i_2, i_3, i_4)\}$  ▷ Append it to  $\mathcal{T}^{\omega_h}$ 
16    if  $n_K^+ = 3$  then
17       $I^+ = I^+ \cup I_K^+$  ▷  $K$  is positively cut. Update positive vertices
18    end
19  end
20 end
21  $V^{\omega_h} \leftarrow$  subset of  $V$  listing coordinates of all vertices in  $I^{\omega_h}$ 
22 Set  $\mathcal{T}^{\omega_h} = (V^{\omega_h}, I^{\omega_h}, C^{\omega_h})$ 
   ▷ Multi-pass boundary projection with interior relaxation
23 Identify  $I_R \subseteq I^{\omega_h} \setminus I^+$  in the vicinity of  $\partial\Omega$  ▷ Interior vertices to relax
24 for  $\lambda = 1/N_P, 2/N_P, \dots, 1$  do
25   for  $i \in I^+$  do
26      $V_i^{\omega_h} \leftarrow \lambda \pi(V_i^{\omega_h}) + (1 - \lambda)V_i^{\omega_h}$  ▷ Positive vertices are perturbed towards their closest
       points in  $\partial\Omega$ 
27   end
28    $\text{dvr}(\mathcal{T}^{\omega_h}, I_R, \text{rdirfunc}, N_R)$  ▷ See Algorithm 2 in Section 2.2.4
29 end
30 Set  $\hat{\mathcal{T}}^{\Omega_h} = (V^{\omega_h}, I^{\omega_h}, C^{\omega_h})$ 
   ▷ Improve qualities of boundary elements
31 for  $r = 1$  to  $N_R$  do
32    $\partial \text{dvr}(\hat{\mathcal{T}}^{\Omega_h}, I^+, \partial\Omega, 1)$ 
       ▷ Constrained relaxation, see Algorithm 3 in Section 2.2.5
33    $\text{dvr}(\hat{\mathcal{T}}^{\Omega_h}, I_R, \text{rdirfunc}, 1)$  ▷ Relaxation of interior vertices
34 end
35 Set  $\mathcal{T}^{\Omega_h} \leftarrow \hat{\mathcal{T}}^{\Omega_h}$ 
36 return  $\mathcal{T}^{\Omega_h}$ 

```

2.2.2 | Projecting positive vertices onto the boundary $\partial\Omega$

With the submesh $\mathcal{T}^{\omega_h} = (\mathbf{V}^{\omega_h}, \mathbf{I}^{\omega_h}, \mathbf{C}^{\omega_h})$ at hand, Algorithm 1 proceeds to compute perturbations for vertices in \mathbf{I}^{ω_h} between steps 23-29. Let us examine how positive vertices are altered. With $\delta = 1/N_P$, step 26 implies that the trajectory of a vertex $i \in \mathbf{I}^+$ over N_P passes between steps 23-29 is given by

$$\underbrace{\mathbf{V}_i^{\omega_h}}_{\mathbf{V}_i} \rightarrow \underbrace{(1 - \delta)\mathbf{V}_i + \delta\pi(\mathbf{V}_i)}_{\mathbf{V}_i^1} \rightarrow \underbrace{(1 - 2\delta)\mathbf{V}_i^1 + 2\delta\pi(\mathbf{V}_i^1)}_{\mathbf{V}_i^2} \rightarrow \cdots \rightarrow \underbrace{(1 - N_P\delta)\mathbf{V}_i^{N_P-1} + N_P\delta\pi(\mathbf{V}_i^{N_P-1})}_{\mathbf{V}_i^{N_P} = \pi(\mathbf{V}_i)}. \quad (1)$$

Prior to the first pass, the location $\mathbf{V}_i^{\omega_h}$ of i coincides with its location \mathbf{V}_i in the mesh \mathcal{T} . In Equation (1), we have denoted its trajectory over the course of N_P passes by $(\mathbf{V}_i^1, \mathbf{V}_i^2, \dots, \mathbf{V}_i^{N_P})$. Therein, as well as in step 26 in Algorithm 1,

$$\pi(x) \triangleq \arg \min_{y \in \partial\Omega} d(x, y) \quad (2)$$

is the closest point projection onto $\partial\Omega$ and $d(\cdot, \cdot)$ in Equation (2) is the Euclidean distance in \mathbb{R}^3 . The assumption of C^2 -regularity for the boundary $\partial\Omega$ becomes crucial in tracing the trajectory of i and in justifying why $\mathbf{V}_i^{N_P} = \pi(\mathbf{V}_i)$ in Equation (1). To this end, introduce the signed distance function to $\partial\Omega$

$$\phi(x) \triangleq s(x) \min_{y \in \partial\Omega} d(x, y), \quad (3)$$

where the inclusion map $s : \mathbb{R}^3 \rightarrow \{-1, 1\}$, defined as $s(x) = -1$ if $x \in \Omega$ and equal to $+1$ otherwise, endows an orientation to $\partial\Omega$. Exploiting the regularity assumed of $\partial\Omega$, we can find a sufficiently small neighborhood of $\partial\Omega$ in which ϕ is C^2 , π is single-valued and C^1 , and we have the useful relationship^{53,54}

$$\pi(x) = x - \phi(x)\nabla\phi(x). \quad (4)$$

Equation (4) implies that if x is sufficiently close to $\partial\Omega$, then the segment joining x and $\pi(x)$ coincides with the direction of the unit normal to $\partial\Omega$ at $\pi(x)$. In particular, it shows that if \mathbf{V}_i is sufficiently close to $\partial\Omega$, then each point \mathbf{V}_i^k for $k = 1, \dots, N_P$ in Equation (1) lies in the segment $\overline{\mathbf{V}_i\pi(\mathbf{V}_i)}$, implying that $\pi(\mathbf{V}_i^k) = \pi(\mathbf{V}_i)$. In fact, Equation (1) now simplifies to the sequence

$$\mathbf{V}_i^k = \alpha^k \mathbf{V}_i + \beta^k \pi(\mathbf{V}_i) \quad \text{for } k = 1, \dots, N_P,$$

where: $\alpha^k = f_\delta(1, k)$, $\beta^k = k\delta + \sum_{\ell=1}^k (\ell\delta)f_\delta(\ell + 1, k)$, and $f_\delta(s, t) = \prod_{q=s}^t (1 - q\delta)$.

It is straightforward to verify that α^k decreases monotonically with increasing k and equals zero for $k = N_P$, β^k increases monotonically with k attaining value one for $k = N_P$, and that $\alpha^k + \beta^k = 1$. In this sense, the trajectory $(\mathbf{V}_i, \mathbf{V}_i^1, \dots, \mathbf{V}_i^{N_P})$ for a positive vertex i realized over multiple passes in steps 23-29 simply represents its march from \mathbf{V}_i to $\pi(\mathbf{V}_i)$ along the normal to $\partial\Omega$ at $\pi(\mathbf{V}_i)$.

2.2.3 | Vertex relaxations

To accommodate projections of positive vertices toward $\partial\Omega$, it is necessary to relax vertices of \mathcal{T}^{ω_h} in the vicinity of $\partial\Omega$ to ensure that tets in the computed mesh have good quality. To this end, we relax vertices in the subset \mathbf{I}_R of \mathbf{I}^{ω_h} using the `dvr` algorithm. Specifically, the invocation `dvr`($\mathcal{T}^{\omega_h}, \mathbf{I}_R, \text{rdirfunc}, N_R$) in step 28 consists in performing N_R relaxation iterations for vertices in \mathbf{I}_R in the mesh \mathcal{T}^{ω_h} along directions specified by the user-defined routine `rdirfunc`, while leaving the remaining vertices in the mesh fixed. At the end of N_P passes of incrementally projecting positive vertices toward $\partial\Omega$ and relaxing vertices in \mathbf{I}_R with `dvr` over steps 23 to 30, \mathcal{T}^{ω_h} is transformed into a mesh that conforms to Ω and is therefore relabeled as $\hat{\mathcal{T}}^{\Omega_h}$ in step 30. A description of the `dvr` algorithm is provided in Section 2.2.4. We note that there are no restrictions on the choice of relaxation directions; the direction for each vertex at each iteration can even be generated at random. A particularly simple choice is to relax vertices along the three Cartesian coordinate directions during successive iterations.

In our numerical experiments, we consistently find that the poorest element quality in the mesh $\hat{\mathcal{T}}^{\Omega_h}$ computed at the end of step 30 is realized over tets having one or more vertices in \mathbf{I}^+ . This is perhaps to be expected, since the trajectories of positive vertices are constrained to follow the local normal direction to $\partial\Omega$. In contrast, vertices in \mathbf{I}_R can be relaxed along any desired direction (as defined by `rdirfunc`). The final section of the algorithm from steps 31-34 is meant specifically to

improve the qualities of elements along the boundary of $\hat{\mathcal{T}}^{\Omega_h}$ by relaxing vertices in I^+ while constraining them to remain on $\partial\Omega$. For this purpose, we introduce a constrained variant of dvr called ∂dvr that is outlined in Algorithm 3 and discussed in Section 2.2.5. The routine $\partial\text{dvr}(\mathcal{T}^{\omega_h}, I^+, 1)$ invoked in step 31 consists in performing one boundary-constrained relaxation of positive vertices. By iterative relaxing vertices in I^+ with ∂dvr and vertices in I_R with dvr , the mesh $\hat{\mathcal{T}}^{\Omega_h}$ is transformed to the final mesh \mathcal{T}^{Ω_h} .

Notice that the mesh \mathcal{T}^{ω_h} is altered with each vertex perturbation during steps 23–29. Similarly, the mesh $\hat{\mathcal{T}}^{\Omega_h}$ is altered because of relaxations performed between steps 31 and 34. Favoring notational simplicity, we have not bothered to explicitly assign a new label for these intermediate mesh iterates, which differ from each other only in the coordinates of their vertices in $I^+ \cup I_R$. Such labels are, in any case, irrelevant in an implementation.

2.2.4 | Directional vertex relaxation

The dvr algorithm helps to accommodate the motion of positive vertices to their closest points on the boundary by relaxing vertices in I_R , so that elements in the computed mesh \mathcal{T}^{Ω_h} have good qualities. In principle, the choice of the mesh relaxation algorithm is not critical, provided that good element qualities are maintained. DVR is especially well suited for Algorithm 1 because it is known that perturbations of positive vertices are small and comparable to the mesh size. In addition, the guarantees of mesh improvement dvr provides (see Theorem 4.2 in the work of Rangarajan and Lew²²) set it apart from alternative relaxation algorithms in the literature, see Section 5.4. In the following, we briefly explain how vertex updates are computed in dvr and provide an outline for its implementation in Algorithm 2.

DVR perturbs a vertex i in a given mesh $\mathcal{T} = (V, I, C)$ along a prescribed direction \mathbf{d}_i in order to maximize the qualities of elements in the 1-ring of i . Specifically, let $\{K_{i_j}\}_{j=1}^{n_i}$ be the collection of tets in \mathcal{T} in the 1-ring of a vertex i being relaxed, and let $\{K_{i_j}^\lambda\}_{j=1}^{n_i}$ denote the corresponding collection of tets that result from perturbing i from its location V_i to $V_i + \lambda\mathbf{d}_i$. The optimal perturbation for i in the dvr algorithm is defined as

$$\lambda^{\text{opt}} \triangleq \arg \max_{\lambda \in \mathbb{R}} \min_{1 \leq j \leq n_i} Q(K_{i_j}^\lambda), \quad (5)$$

where Q is an element quality metric. In all our discussions and examples in this article, we adopt the mean ratio metric⁵⁵

$$Q(K) \triangleq 12\sqrt[3]{9} \frac{|\mu(K)|^{2/3} \text{sign}(\mu(K))}{\sum_{1 \leq p < q \leq 4} \ell_{pq}^2(K)}, \quad (6)$$

Algorithm 2: DVR : Relax vertices in I_R of the mesh \mathcal{T} along prescribed directions

Input:

$\mathcal{T} = (V, I, C)$: Tet mesh

I_R : List of vertices in I that to be relaxed

rdirfunc : Function assigning relaxation directions for vertices in I_R

N_R : Number of relaxation iterations

▷ Assume: $Q(K) > 0$ for each $K \in \mathcal{T}$

▷ $I_R \subseteq I$

37 **for** $r = 1, 2, \dots, N_R$ **do**

38 **foreach** $i \in I_R$ **do**

39 $\{K_{i_j}\}_{j=1}^{n_i} \leftarrow$ tets in the 1-ring of vertex i

40 Get relaxation direction $\mathbf{d}_i = \text{rdirfunc}(i)$

▷ \mathbf{d}_i can be iteration dependent

41 Compute $\lambda^{\text{opt}} = \arg \max_{\lambda \in \mathbb{R}} \min_{1 \leq j \leq n_i} Q(K_{i_j}^\lambda)$

▷ See⁵⁶ for resolution algorithms

42 Update $V_i = V_i + \lambda^{\text{opt}} \mathbf{d}_i$

43 **end**

44 **end**

45 **return** \mathcal{T}

where $\mu(K)$ is equal to the signed volume of the tet K and ℓ_{pq} is equal to the length of the edge joining its p th and q th vertices. As evident from Equation (5), relocating i from V_i to $V_i + \lambda^{\text{opt}} \mathbf{d}_i$ maximizes the minimum among the qualities of elements in its 1-ring with relaxation restricted to the direction \mathbf{d}_i .

The dvr relaxation algorithm simply consists in iteratively and sequentially relaxing vertices along prescribed directions, with each vertex update computed by resolving a scalar max-min problem of the form Equation (5). The steps involved

are summarized in Algorithm 2. The mesh returned by the algorithm differs from the input mesh only in the coordinates of vertices in the set I_R . In particular, the input and returned meshes have identical sets of nodal indices and element connectivities.

We refer to the work of Rangarajan and Lew²² for a detailed analysis of Algorithm 2. There, it is proved that, by adopting the mean ratio metric to define element qualities and by ensuring that elements in the input mesh \mathcal{T} have strictly positive qualities, vertex updates in Algorithm 2 are guaranteed to be well defined. In particular the max-min problem to be resolved in step 41 has a unique solution for each $i \in I_R$, at each relaxation iteration and irrespective of the choice of the relaxation direction. In addition to knowing that vertex updates in the algorithm are well defined, it is of course crucial to examine how the optimizer can be computed in practice. This is not a trivial issue because the function $\lambda \mapsto \min_{1 \leq j \leq n_i} \{Q(K_{i_j}^\lambda)\}$ to be maximized to compute λ^{opt} is in general only continuous but not differentiable everywhere. Hence, λ^{opt} cannot be computed using simple Newton-based algorithms. However, exploiting certain properties of the mean ratio metric (quasiconcavity in particular) and the fact that the problem is one dimensional helps in constructing simple and robust resolution methods. We have used the algorithms introduced in the work of Rangarajan⁵⁶ in all our examples in this article. For the sake of brevity, we have not reproduced them here.

2.2.5 | Constrained relaxation for boundary vertices

We introduce a variant of dvr , labeled ∂dvr , for relaxing vertices lying on the boundary $\partial\Omega$ in a mesh \mathcal{T} . For describing the steps involved in Algorithm 3, some additional notation is helpful. Given a vertex i in \mathcal{T} lying on the boundary $\partial\Omega$, let \mathbf{d}_i denote a unit vector in the tangent plane to $\partial\Omega$ at V_i , ie, $\mathbf{d}_i \in T_{V_i}(\partial\Omega)$. Retaining the definitions for $\{K_{i_j}\}_{j=1}^{n_i}$ and $\{K_{i_j}^\lambda\}_{j=1}^{n_i}$ introduced previously, let $\{K_{i_j}^{\lambda,\partial}\}_{j=1}^{n_i}$ denote the corresponding set of tets that result from perturbing vertex i from V_i to the location $\pi(V_i + \lambda\mathbf{d}_i) \in \partial\Omega$. An example distinguishing the tets K_{i_j} and $K_{i_j}^{\lambda,\partial}$ is shown in Figure 4.

Algorithm 3: ∂dvr : Relax boundary vertices in mesh \mathcal{T} lying on $\partial\Omega$

Input:

$\mathcal{T} = (V, I, C)$: Tet mesh

▷ Assume: $Q(K) > 0$ for each $K \in \mathcal{T}$

I^+ : List of vertices lying on $\partial\Omega$ to be relaxed.

$\partial\Omega$: C^2 -regular boundary

N_R : Number of relaxation iterations

N_S : Number of sampling points

```

46 for  $r = 1, 2, \dots, N_R$  do
47   foreach  $i \in I^+$  do
48     Choose a relaxation direction  $\mathbf{d}_i \in T_{V_i}\partial\Omega$            ▷  $\mathbf{d}_i$  can be iteration dependent
49     Identify  $\{K_{i_j}\}_{j=1}^{n_i} \leftarrow$  tets in the 1-ring of vertex  $i$ 
50      $h \leftarrow$  representative mesh size at  $i$ 
51      $\Lambda = \{-h, -h + h/N_S, \dots, h - h/N_S, h\}$            ▷ Sample coordinates, includes  $\lambda = 0$  for convenience
52     Identify  $\lambda^{\text{opt}} \in \arg \max_{\lambda \in \Lambda} \min_{1 \leq j \leq n_i} Q(K_{i_j}^{\lambda,\partial})$ 
53     if  $\min_{1 \leq j \leq n_i} Q(K_{i_j}^{\lambda^{\text{opt}},\partial}) > \min_{1 \leq j \leq n_i} Q(K_{i_j})$    ▷ Update  $V_i$  if the poorest element quality is improved
54     then
55       Update  $V_i = \pi(V_i + \lambda^{\text{opt}} \mathbf{d}_i)$ 
56     end
57   end
58 end
59 return  $\mathcal{T}$ 

```

The ∂dvr algorithm is a variant of dvr , wherein the coordinate λ^{opt} is computed as

$$\lambda^{\text{opt}} \in \arg \max_{\lambda \in \Lambda} \min_{1 \leq j \leq n_i} Q(K_{i_j}^{\lambda,\partial}), \quad (7)$$

with $\Lambda \subset \mathbb{R}$ being a finite set of sample points, following which the vertex i is relocated to $\pi(V_i + \lambda^{\text{opt}} \mathbf{d}_i)$. Such an update can be interpreted as constructing a curve $\lambda \mapsto \pi(V_i + \lambda \mathbf{d}_i)$ on $\partial\Omega$ and sampling it at the points in Λ to decide the best location

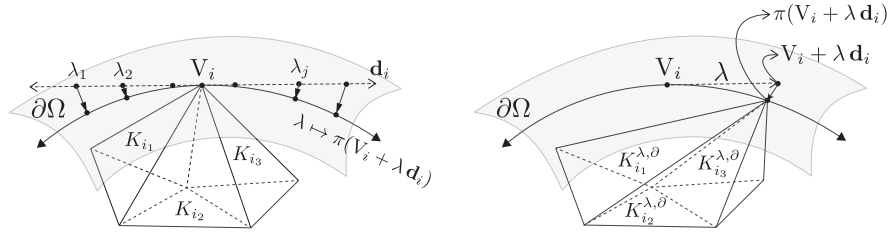


FIGURE 4 Illustration of the ∂dvr algorithm to relax a vertex i that is constrained to lie on the boundary $\partial\Omega$

for i . A direct comparison of Equations (5) and (7) reveals a few key differences between the dvr and ∂dvr algorithms. First, the optimizer λ^{opt} is sought in a predefined finite set of points Λ in ∂dvr —the best location for i is chosen from among a finite number of possibilities and determining λ^{opt} reduces to comparing element qualities at these candidate locations. In Algorithm 3, we have set Λ to be $\{-h, -h + \delta, \dots, h - \delta, h\}$, where h is a representative mesh size at the vertex i and $\delta = h/N_S$ is a small fraction of h . This choice defines a set of uniformly-spaced points along the tangent \mathbf{d}_i that is centered at V_i . In contrast, the set of candidate optimizers is the entire set \mathbb{R} in dvr . The second difference arises from the fact that, while ∂dvr evaluates qualities of tets $K_{i_j}^{\lambda,\theta}$, dvr evaluates qualities of the tets $K_{i_j}^\lambda$. The third difference is a subtle one and stems from the restricted set of candidate optimizers chosen in ∂dvr . In step 53, notice that an explicit check for improvement in the poorest element quality at the identified coordinate λ^{opt} is required. This check is required despite $\lambda = 0$ being included in the sampling set Λ . This is because, in the case that (7) has multiple maximizers including $\lambda = 0$, updating V_i to $\pi(V_i + \lambda^{\text{opt}}\mathbf{d}_i)$ with $\lambda^{\text{opt}} \neq 0$ may not guarantee improvement in the mesh quality. Specifically, it is possible in such a case that the mesh quality that is realized at $\lambda^{\text{opt}} \neq 0$ may be worse than that at $\lambda = 0$. We will revisit this point in Section 3.4.

3 | NUMERICAL EXPERIMENTS

We devote this section to presenting numerical experiments to examine the performance of the um algorithm. Through the first example, which is discussed in some detail, we explain aspects of representing domains with C^2 -regular boundaries, constructing background meshes with acute dihedral angles and introduce a vector-valued mesh quality metric. In the examples that follow in Section 3.5, we include comparisons of the meshes computed by um with alternative mesh generators (TetGen, Gmsh, CGAL, and HyperMesh). In Section 3.7, we show an example triangulating the same domain using different background meshes to emphasize the independence of the algorithm from the choice of the background mesh. Finally, in Section 3.8, we mention a few remarks related to implementing the algorithm and examine its scaling with the number of tets/vertices in the background mesh. Examples demonstrating the application of um to meshing evolving domains are presented subsequently in Section 4.

3.1 | The domain and the background mesh

We begin with the example of a “duckie”-shaped domain Ω immersed in a background mesh of tets shown in Figure 5A. The domain is defined implicitly, as the set

$$\Omega \triangleq \{x \in \mathbb{R}^3 : \psi(x) < 0\}, \quad (8)$$

where ψ is a level set function that is constructed following the procedure discussed in Section 3.7 to fit a given point cloud sampling of the surface. In keeping with the terminology introduced for similar constructions of implicit functions from point cloud representations, we shall henceforth refer to the implicit function ψ as an “SSD” function.⁵⁷ For the current discussion, it suffices to note that ψ defined by following the recipe in Section 3.7 is twice continuously differentiable and that Ω is a C^2 -regular domain as required in the um algorithm. The boundary $\partial\Omega$ shown in Figure 5A is equal to the zero level set of ψ and is a closed surface with genus 1.

Figure 5B shows the stencil of tets that serve as building blocks used to construct the background mesh \mathcal{T} . A special feature of this stencil is that all dihedral angles in the tets constituting it are strictly acute. Consequently, the background mesh constructed by tiling these stencils automatically satisfies the acute conditioning angle requirement in um .

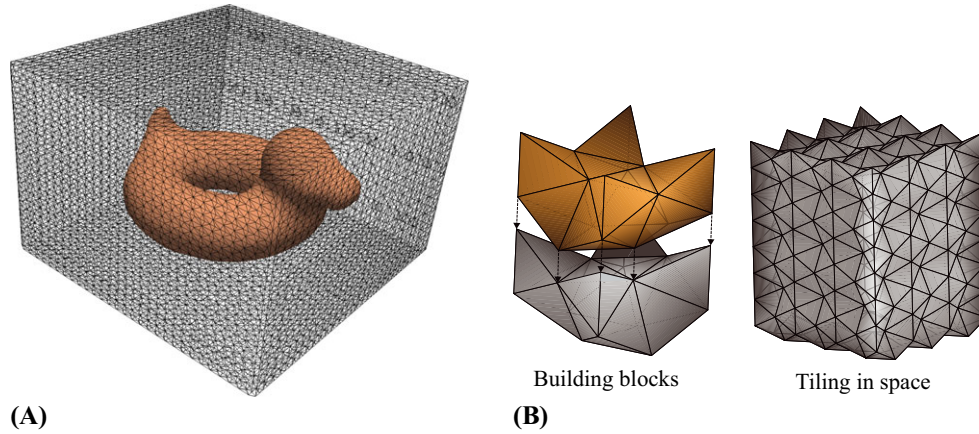


FIGURE 5 A duckie-shaped domain is shown immersed in a background mesh of tets in (A). By using the stencil of acute tets in (B) to construct the background mesh, we ensure a priori that the acute conditioning angle assumption in um is satisfied. With the exception of Section 3.7, we have used the stencil in (B) to construct background meshes in all the examples discussed in this article. Details of the stencil can be found under the construction “A15 tetrahedrally-closed packed structures” in the works of Eppstein et al⁵⁸ and Sullivan,⁵⁹ along with alternate choices for stencils with acute tets [Colour figure can be viewed at wileyonlinelibrary.com]

Although the um algorithm only requires specific dihedral angles in positively cut tets to be acute, such a construction for \mathcal{T} helps to avoid the tedium of explicitly identifying proximal vertices and inspecting the magnitude of conditioning angles. Besides, such a construction for the background mesh is simple and fast, with uniform refinement of the mesh achieved by simply stacking sufficiently small and sufficiently many of the stencils. Alternative constructions for tet meshes with all acute dihedral angles can be found in the works of Eppstein et al⁵⁸ and Sullivan,⁵⁹ some of which are used in the example in Section 3.7.

3.2 | Algorithmic parameters and mesh iterates

We set the number of passes $N_P = 5$ and the number of relaxation iterations $N_R = 25$ in Algorithm 1. For simplicity, we relax the entire collection of interior vertices in \mathcal{T}^{ω_h} , ie, $I_R = I^{\omega_h} \setminus I^+$, with vertices ordered in increasing order of their index assigned in \mathcal{T} . The relaxation directions, defined by the function `rdirfunc`, are chosen to alternate between the three coordinate directions $\{\mathbf{e}_x, \mathbf{e}_y, \mathbf{e}_z\}$ during successive iterations. Hence, in Algorithm 2, all vertices in I_R are relaxed along \mathbf{e}_x during the first iteration, along \mathbf{e}_y during the second iteration, along \mathbf{e}_z during the third, and so on. Finally, we set $N_S = 20$ in Algorithm 3 to compute perturbations of vertices constrained to lie on $\partial\Omega$. At each iteration and for each vertex in I^+ relaxed by Algorithm 3, the relaxation direction is set to be the projection of a randomly generated vector onto the local tangent plane of $\partial\Omega$.

Figure 6A shows the submesh \mathcal{T}^{ω_h} realized at step 22 in Algorithm 1. The key step in identifying \mathcal{T}^{ω_h} is the query in step 7 that distinguishes vertices in \mathcal{T} as lying in Ω or in its complement. Due to the implicit representation adopted for Ω in (8), this only requires inspecting the sign of the level set function ψ at the nodes of \mathcal{T} . Then, the algorithm simply retains the tets having $\psi < 0$ at one or more of its vertices. Notice from the figure that, although \mathcal{T}^{ω_h} appears only to crudely resemble Ω , its boundary is topologically equivalent to $\partial\Omega$. By this, we mean that $\partial\omega_h$ is a closed surface with genus 1, just as $\partial\Omega$. As we discuss in Section 5, this is likely not a coincidence but a consequence of the restrictions on $\partial\Omega$ and \mathcal{T} imposed in the algorithm. Figure 6B shows the mesh realized at the end of the second boundary projection-interior relaxation pass and clearly reveals the perturbation of positive vertices towards the boundary. The mesh shown in Figure 6C is realized at the end of step 30, at which stage, each positive vertex has been projected onto its respective closest point in $\partial\Omega$ while remaining vertices have been relaxed using the `dvr` algorithm. Figures 7A to 7C show further details of the final mesh \mathcal{T}^{Ω_h} computed by the algorithm.

3.3 | Vector-valued mesh qualities

By adopting the mean ratio metric to determine element qualities in the `dvr` and `∂dvr` algorithms, tet quality values closer to 1 indicate more regularly shaped elements, a value of 0 implies a degenerate (collapsed) tet and values less than 0

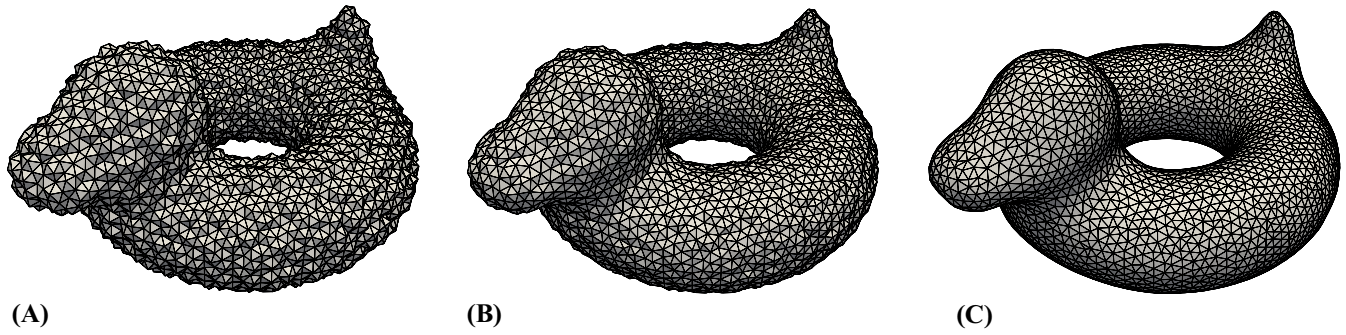


FIGURE 6 Intermediate meshes realized in Algorithm 1 when meshing the duckie-shaped domain shown in Figure 5A. A, \mathcal{T}^{ω_h} realized at step 22 in Algorithm 1; B, \mathcal{T}^{ω_h} after 2 passes of boundary projection and relaxation, ie, $\lambda = 2/N_p$, in Algorithm 1; C, $\hat{\mathcal{T}}^{\Omega_h}$ realized at step 30 in Algorithm 1

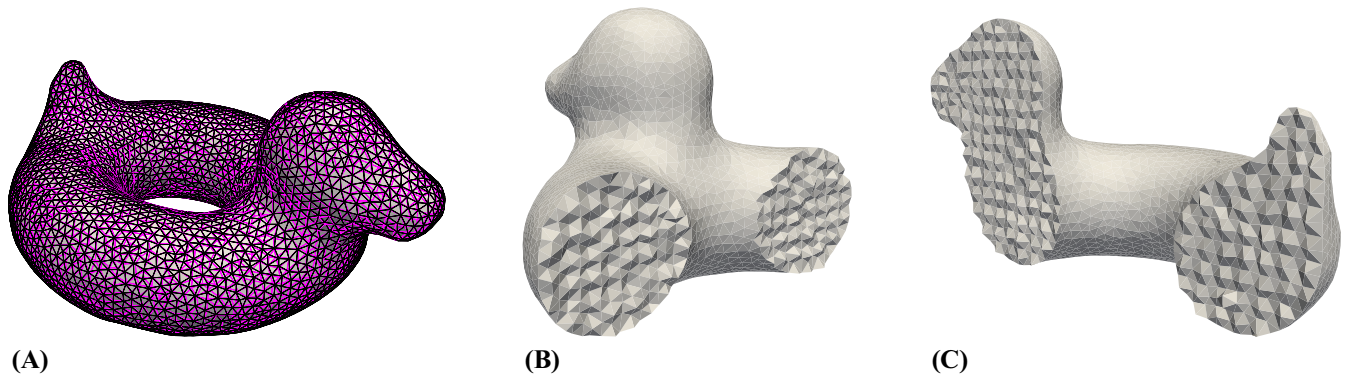


FIGURE 7 Details of the final mesh \mathcal{T}^{Ω_h} computed by um for the duckie. Figure (A) shows the effect of relaxing boundary nodes with Algorithm 3 (∂dvr). Boundaries of the meshes $\hat{\mathcal{T}}^{\Omega_h}$ and \mathcal{T}^{Ω_h} are rendered in pink and black colors, respectively. Figures (B) and (C) show detailed cut sections of \mathcal{T}^{Ω_h} [Colour figure can be viewed at wileyonlinelibrary.com]

correspond to inverted elements. For the purpose of examining the evolution of the mesh quality during the various steps in the um algorithm, just the element quality metric Q does not suffice. Instead, we introduce a vector-valued metric \mathbf{Q} that is comprised of a specific list of element qualities in a mesh.

Given a mesh S , let $q_i(S) = \min_{1 \leq j \leq n_i} Q(K_{i_j})$ denote the minimum among the qualities of tets $\{K_{i_j}\}_{j=1}^{n_i}$ in the 1-ring of vertex i in S . Let \mathbf{asc} be a permutation that rearranges components of a vector in ascending order, ie,

$$\mathbf{asc}(u_1 \leq u_2 \leq \dots \leq u_\alpha) \triangleq (u_1, u_2, \dots, u_\alpha), \quad \alpha \in \mathbb{N}.$$

We define the quality of S to be the vector

$$\mathbf{Q}(S) = \mathbf{asc}(q_1(S), q_2(S), \dots, q_\alpha(S)),$$

where $\{1, 2, \dots, \alpha\}$ is the index set of S . A couple of aspects of the definition of $\mathbf{Q}(S)$ are worth highlighting. Its first component $Q_1(S)$ is equal to the minimum among the qualities of all elements in S and is therefore significant in finite element calculations. The vector $\mathbf{Q}(S)$ is not simply a lexicographical ordering of element qualities in the mesh—the quality of the same element generally appears multiple times in $\mathbf{Q}(S)$. In this sense, $\mathbf{Q}(S)$ naturally accentuates the influence of elements with poor qualities. We will extensively use plots of quality vectors to inspect mesh qualities. Specifically, to inspect the quality of S , we plot the components $Q_1(S), Q_2(S), \dots$ along the vertical axis and the corresponding indices $1, 2, \dots$ along the horizontal axis. We choose a logarithmic scale along the horizontal axis to facilitate a closer examination of poorer element qualities.

Figure 8 inspects the mesh qualities realized at three different stages in the um algorithm while computing a mesh for the example in Figure 5A. The mesh \mathcal{T}^{ω_h} realized at step 22 is a subset of the background mesh and its quality plotted in Figure 8A simply reflects the quality of the background mesh. The quality of the mesh $\hat{\mathcal{T}}^{\Omega_h}$ realized after projecting positive vertices onto $\partial\Omega$ while concurrently relaxing interior vertices is noticeably poorer than that of \mathcal{T}^{ω_h} . Crucially though, none of its element qualities are zero or negative, indicating that no tet in $\hat{\mathcal{T}}^{\Omega_h}$ is collapsed or inverted. This feature helps the um algorithm bypass tedious mesh repair and local retriangulation operations.

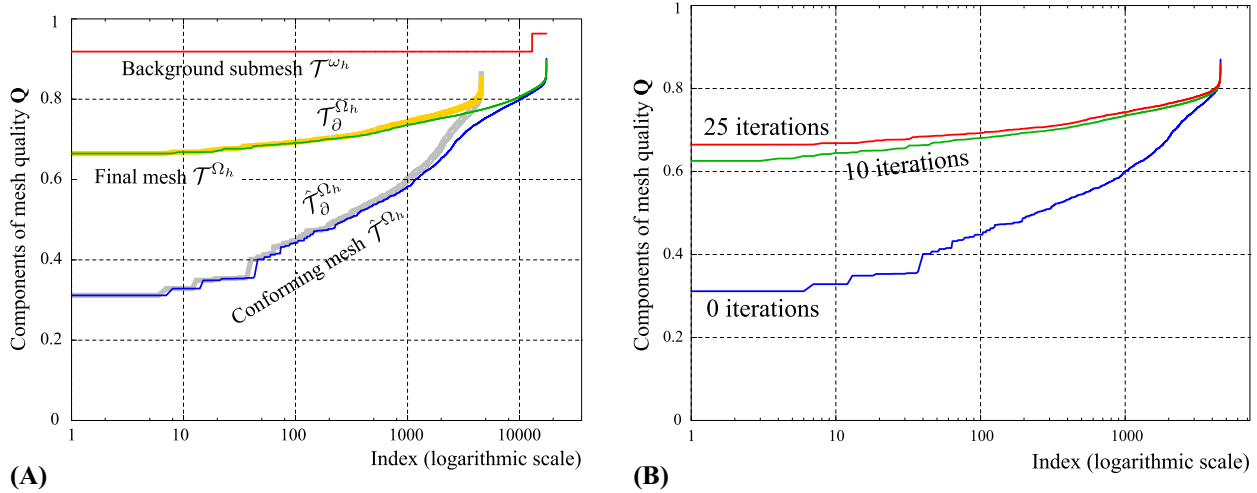


FIGURE 8 Inspecting the mesh qualities realized at various stages of Algorithm 1 while meshing the duckie-shaped domain. In (A), notice that the poorest element qualities in the boundary-conforming mesh $\hat{\mathcal{T}}^{\Omega_h}$ are realized over elements having one or more vertices in I^+ , which constitute the mesh $\hat{\mathcal{T}}_{\partial}^{\Omega_h}$. These element qualities are improved using boundary-constrained vertex relaxations with ∂dvr . Improvement in the quality of $\hat{\mathcal{T}}_{\partial}^{\Omega_h}$ after a representative number of iterations of ∂dvr is shown in (B). Vertex perturbations computed by ∂dvr that materialize in the mesh improvement observed in (B) are visualized in Figure 7A [Colour figure can be viewed at wileyonlinelibrary.com]

3.4 | Constrained relaxation of boundary vertices

The meshes $\hat{\mathcal{T}}_{\partial}^{\Omega_h}$ and $\mathcal{T}_{\partial}^{\Omega_h}$ referred to in Figure 8 are submeshes of $\hat{\mathcal{T}}^{\Omega_h}$ and \mathcal{T}^{Ω_h} , respectively, consisting of tets that have at least one vertex in I^+ . Comparing the blue and gray curves in Figure 8A reveals that the elements with poorest qualities in $\hat{\mathcal{T}}^{\Omega_h}$ in fact belong to its submesh $\hat{\mathcal{T}}_{\partial}^{\Omega_h}$. This is to be expected, because one or more vertices in each tet in $\hat{\mathcal{T}}_{\partial}^{\Omega_h}$ have been projected onto $\partial\Omega$ and consequently, these tets suffer vertex perturbations that are comparable to the local mesh size. It is precisely this realization that warrants additional boundary-constrained relaxation with ∂dvr . The quality of the final mesh \mathcal{T}^{Ω_h} shown in green in Figure 8A is significantly better than its counterpart at step 22, and this is achieved predominantly due to the improvement of qualities of tets in the submesh $\hat{\mathcal{T}}_{\partial}^{\Omega_h}$. The improvement in the mesh quality of $\mathcal{T}_{\partial}^{\Omega_h}$ compared to that of $\hat{\mathcal{T}}_{\partial}^{\Omega_h}$ is highlighted in Figure 8B.

At the end of Section 2.2.5, we noted a subtle distinction between the dvr and ∂dvr algorithms consisting in whether or not it is necessary to explicitly check for improvement in the poorest element quality at a vertex prior to updating it. Such a check is required in ∂dvr (step 53), but not in dvr . The distinction is based on the relationship between improvement in the poorest element quality and the mesh quality vector following a vertex update. Specifically, improvement in the poorest element quality guarantees improvement in the mesh quality as well.⁵⁶ It can be shown^{22,56} that Equation (5) underlying vertex updates in dvr has a unique solution when adopting the mean ratio metric, which automatically implies that its solution improves the poorest element quality around a vertex, which, in turn, improves the mesh quality. On the contrary, multiple candidates from the sample set Λ in Equation (7) may yield identical values for the optimal poorest element quality. In such a case, it becomes necessary to examine additional components of the mesh quality vector to distinguish between these candidates. Step 53 of the ∂dvr algorithm is included to avoid such involved calculations by explicitly checking for improvement in the minimum element quality instead.

3.5 | More examples and comparisons with Delaunay-based algorithms

Figures 9 and 10 show more examples of meshes computed by the um algorithm for domains represented implicitly using SSD functions. The background meshes in these examples are constructed using the same stencil shown in Figure 5B, with mesh sizes that are commensurate with geometric features of the immersed domains (eg, large curvatures and thin sections). The algorithmic parameters used for these examples are identical to those mentioned in Section 3.2.

In each example shown, we also compare the quality of the mesh computed by um with those from four popular tetrahedral mesh generators—TetGen,³³ Gmsh,³⁴ CGAL³⁵ and HyperMesh.³⁶ The first three algorithms are Delaunay-based. We do not speculate about which algorithm is used in the commercial package HyperMesh. Regardless, the specifics of

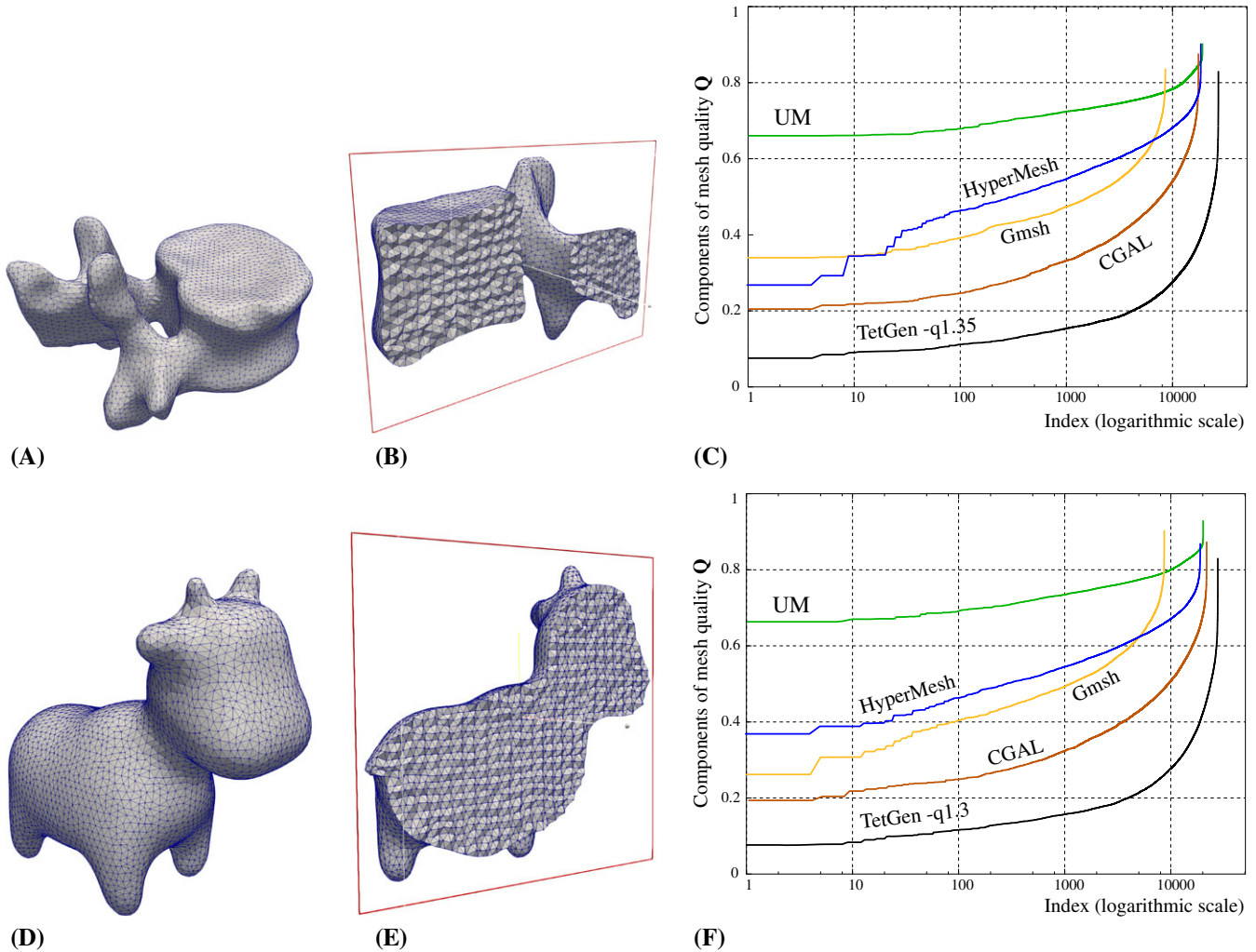


FIGURE 9 Examples of meshes computed by *um* are shown in (A), (B), (D), and (E). Qualities of these meshes are compared with those computed using alternate mesh generators in (C) and (F). Details of the inputs and options used with TetGen, Gmsh, CGAL, and HyperMesh are mentioned in Section 3.5 [Colour figure can be viewed at wileyonlinelibrary.com]

each mesh generator are quite different and for the purpose of their comparison with *um*, it is important to mention the inputs and options that we have used with each one.

TetGen: TetGen requires a surface mesh of triangles as input rather than an implicit surface description we adopt for *um*. To facilitate a fair comparison, we provide the boundary (ie, the skin) of the final mesh \mathcal{T}^{Ω_h} computed by *um* as the input to TetGen. With all other algorithmic options in TetGen set to default values, we only use the “-q” command line switch that controls the addition of new points in the mesh to improve element qualities by bounding the radius-edge ratio. Specifying a smaller value yields a mesh with better quality. The default value of the parameter is 2.0. In all the examples in Figures 9 and 10, we choose a lower value than the default, while also ensuring that the size of the final mesh (number of nodes and elements) is comparable to that computed by *um*. The value chosen in each example is indicated alongside the plots of mesh qualities.

Gmsh: We follow a similar workflow using Gmsh. The input mesh provided to Gmsh is the skin of \mathcal{T}^{Ω_h} computed by *um* and all algorithmic parameters are set to default values. It is worth mentioning that Gmsh uses the TetGen library for volume mesh generation. We then improve the mesh quality using NetGen’s mesh optimization routines⁶⁰ directly from the interface provided by Gmsh. Unlike *dvr* or *∂dvr* that involve only vertex perturbations, NetGen’s mesh optimization routines involve a large pool of operations including connectivity alterations (eg, face swapping). For this reason, it not surprising

to see in Figures 9 and 10 that the qualities of meshes computed by Gmsh are consistently better than those of TetGen—the additional set of mesh optimizations performed with the NetGen library improves element qualities noticeably.

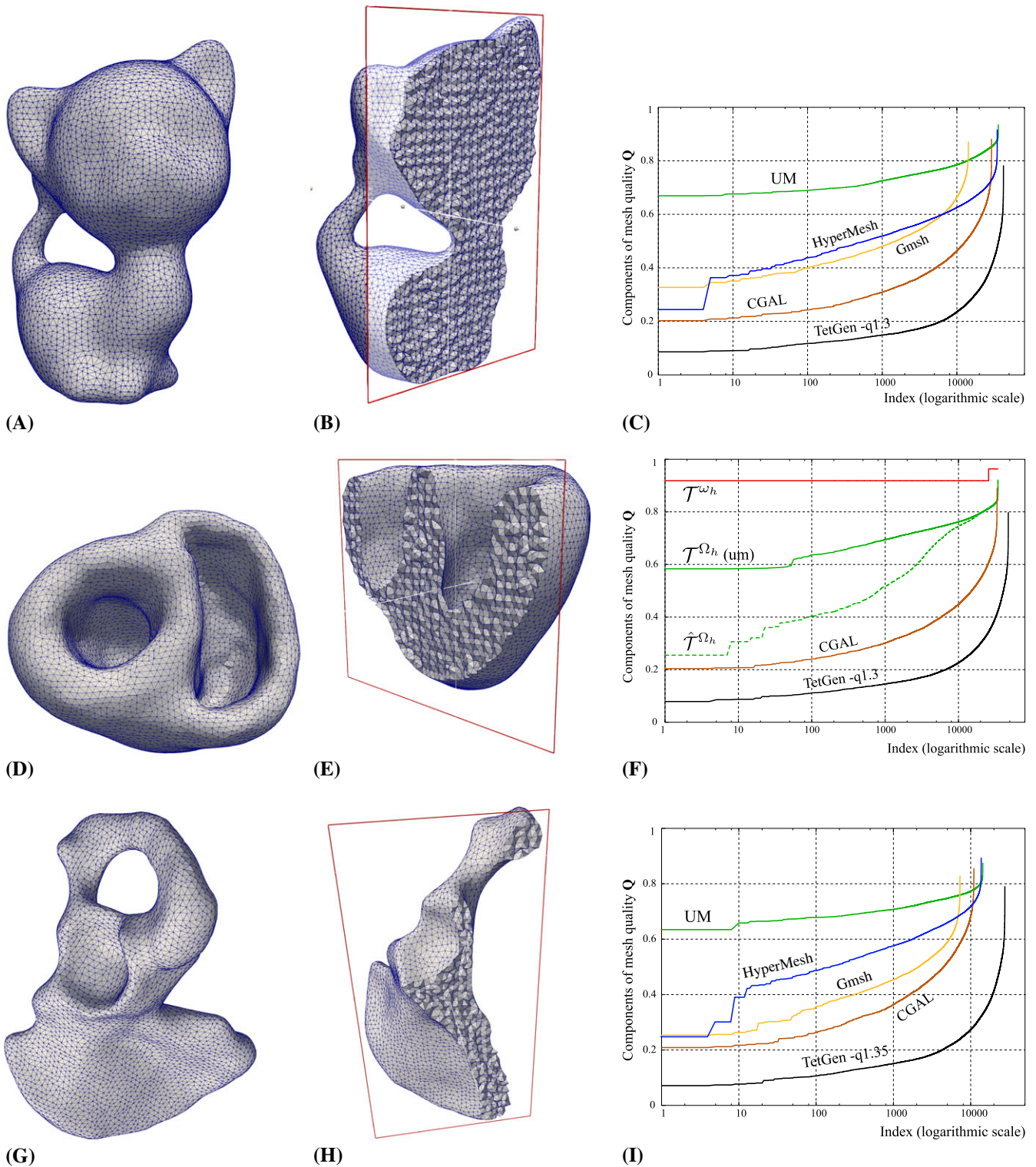


FIGURE 10 Examples of meshes computed by um are shown in (A), (B), (D), (E), (G), and (H). Qualities of these meshes are compared with alternate algorithms in (C), (F), and (I). See the text in Section 3.5 for further details of the comparison [Colour figure can be viewed at wileyonlinelibrary.com]

CGAL: The algorithm in work of Oudot et al⁴¹ implemented in CGAL directly accommodates an implicit representation for domains of the form (8). Hence, we provide the same SSD function as input to CGAL as we do for `um` in each example. The mesh-quality-related options required in CGAL, namely, the parameters `facet_angle` and `cell_radius_edge_ratio`, are set to 30 and 2, identical to the ones used in the examples accompanying the library. The first parameter is an angle and controls the shapes of triangles on the boundary. The chosen value additionally ensures that the algorithm terminates. The value of the parameter `cell_radius_edge_ratio` controls the qualities of tets generated by bounding the ratio of the circumradius of a tet and its shortest edge. Remaining size-related parameters (`facet_size`, `facet_distance`, and `cell_size`) are set to be commensurate with the mesh size of the background mesh used in `um`, so that the meshes computed by `um` and by CGAL have similar number of vertices/elements in each example.

HyperMesh: Similar to the workflow followed with TetGen and Gmsh, we provide the skin of the mesh \mathcal{T}^{Ω_h} computed by `um` as the input to HyperMesh. We use the `Volume Tetra` option for meshing and specify the domain to be meshed as the volume enclosed by the skin of \mathcal{T}^{Ω_h} . In each example, the mesh size parameter required by the software is chosen such that the resulting mesh has similar number of elements as that computed by `um`. All other meshing-related parameters are set to their preassigned default values.

In summary, we mention that the skin of the meshes computed by TetGen, Gmsh, HyperMesh and `um` are identical in each example. On the other hand, the implicit function specifying the boundary of the domain to be meshed is common to the CGAL and `um` algorithms in each case.

A cursory inspection of the qualities reported from the five meshing algorithms in Figures 9C, 9F, 10C, 10F, and 10I shows that `um` computes meshes that are consistently of better quality. In each example, the poorest element quality in the mesh computed by `um` is noticeably better than in the meshes computed by the remaining algorithms, see Figure 1. We remark that the only condition we used in selecting the examples is that $\partial\Omega$ be smooth and not the outcome of the comparison. Nevertheless, we caution that it is unwise to infer broad conclusions from these comparisons alone—the `um` algorithm is very different from the remaining four, and so are the classes of domains that can be meshed by it.

Figure 10F includes plots of the qualities of the intermediate meshes \mathcal{T}^{ω_h} and $\hat{\mathcal{T}}^{\Omega_h}$, besides that of the final mesh \mathcal{T}^{Ω_h} . As we observed previously in Figure 8A, the quality of $\hat{\mathcal{T}}^{\Omega_h}$ is poorer compared to the background mesh (\mathcal{T}^{ω_h}) but remains bounded away from zero. In particular, projecting positive vertices in \mathcal{T}^{ω_h} onto $\partial\Omega$ does not result in any element becoming degenerate or inverted. We also find, without exception, that elements with poor qualities in $\hat{\mathcal{T}}^{\Omega_h}$ have one or more vertices on the boundary $\partial\Omega$ and that their qualities are substantially improved by relaxation with ∂dvr .

Sources for point clouds used in the examples: In all the examples shown thus far in Figures 6, 9, and 10, we have defined the domain implicitly using SSD functions computed by following the procedure discussed in Section 3.8. The input data for these calculations are oriented point clouds, ie, collections of points in \mathbb{R}^3 and corresponding sets of unit normals. These point cloud datasets are chosen to be vertices of surface meshes accessed from a repository of 3D models⁶¹ for the examples in Figure 6 and Figure 9D, from a repository of sample meshes accompanying the TetGen library⁶² for the example in Figure 9A, from sample data⁶³ accompanying the CGAL library for the example in Figure 10A and from the repository at the VisionAir Project website⁶⁴ for the example in Figure 10D.

3.6 | A variety of background meshes

We have adopted the stencil in Figure 5B to construct background meshes in all the examples discussed thus far. For this reason, it may seem that the `um` algorithm and its performance observed in the examples are correlated to the specific stencil chosen. Indeed, it is possible to design meshing algorithms by assuming a specific structure for the background mesh.^{27,28} This is, however, not the case in `um`. As we mentioned previously in Section 2.2, the only assumptions on background meshes are the restrictions on sizes and dihedral angles in elements close to the immersed boundary. In this sense, the `um` algorithm is agnostic to how the background mesh is constructed, provided that the mesh size and conditioning angle assumptions are satisfied. To wit, the construction of the background mesh is *not* a part of the `um` algorithm but is simply an input to it. We use the following example to emphasize this point further.

We consider meshing the domain Ω defined to be the zero sublevel set of the implicit function

$$\psi(x, y, z) = 2y(y^2 - 3x^2)(1 - z^2) + (x^2 + y^2)^2 - (9z^2 - 1)(1 - z^2), \quad (9)$$

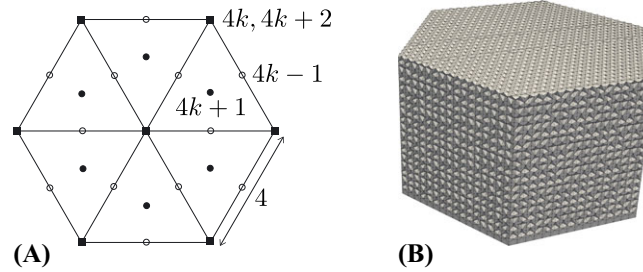


FIGURE 11 To construct stencil 2, we start by tiling \mathbb{R}^2 with equilateral triangles of side 4. Then, for each integer k , we add points at heights $4k, 4k + 2$ over the vertices, $4k - 1$ over midpoints of the edges and $4k + 1$ over triangle centroids as indicated in (A). A Delaunay triangulation of the point cloud defined this way yields a mesh of acute tets. Figure (B) shows a mesh constructed in this way to serve as the background mesh for triangulating the domain defined by (9). The mesh is a tetrahedrally close packed structure, labeled as “Z” in the work of Eppstein et al⁵⁸

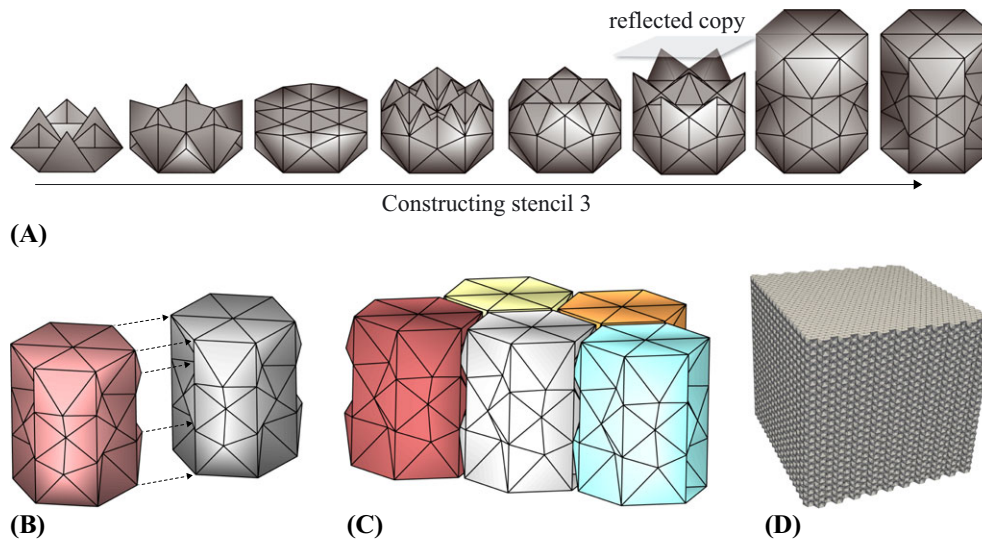


FIGURE 12 Stencil 3 is defined by following the steps outlined in Section 5.3 in Eppstein et al⁵⁸ and is visualized in (A). The three-fold rotational symmetry about the vertical axis and the translational symmetry along the same axis enable stacking together these stencils as shown in (B) and (C). A background mesh constructed this way for the purpose of triangulating the domain defined by (9) is shown in (D) [Colour figure can be viewed at wileyonlinelibrary.com]

where (x, y, z) are the Cartesian coordinates in \mathbb{R}^3 . The boundary of Ω is a closed surface with genus 2. We use four different background meshes for triangulating Ω , which are shown in Figures 5A, 11B, 12D, and 13B. Since each one of these meshes is constructed using a stencil consisting of all acute angled tets, the conditioning angle assumption is satisfied automatically. On the other hand, the mesh size is set by stacking together sufficiently many stencils. We shall refer to the stencils used in Figures 5, 11, 12, and 13 as stencil 1, 2, 3, and 4, respectively. Details of their construction are shown in the respective figures. The stencils in Figures 5 and 11 are tetrahedrally close packed structures.⁵⁸ The rather intricate details involved in defining Stencil 3 are illustrated in Figure 12A. A notable feature of the stencil is that since its end faces are flat, it can be used to tile domains enclosed between parallel planes, ie, to tile “slabs.” Stencil 4 shown in Figure 13 is a construction from the work of VanderZee et al⁶⁵ consisting of 277 vertices and 1370 (well-centered) tetrahedra. Its method of construction using a combination of advancing front surface triangulation, Delaunay triangulation, mesh optimization and manual vertex addition is illustrative of the difficulty in constructing acute tet meshes over bounded domains. The cut section of the stencil shown in Figure 13B reveals a very heterogenous distribution of element sizes.

With Ω immersed in each one of the four background meshes, we invoke the um algorithm to compute conforming meshes in each case. The algorithmic parameters are retained unchanged from Section 3.5. The computed meshes and their cut sections are shown in Figure 14. The first row of images in the figure show the subset of elements identified in each background mesh to be mapped onto Ω , ie, the triangulation \mathcal{T}^{ω_h} determined in Algorithm 1 with each background

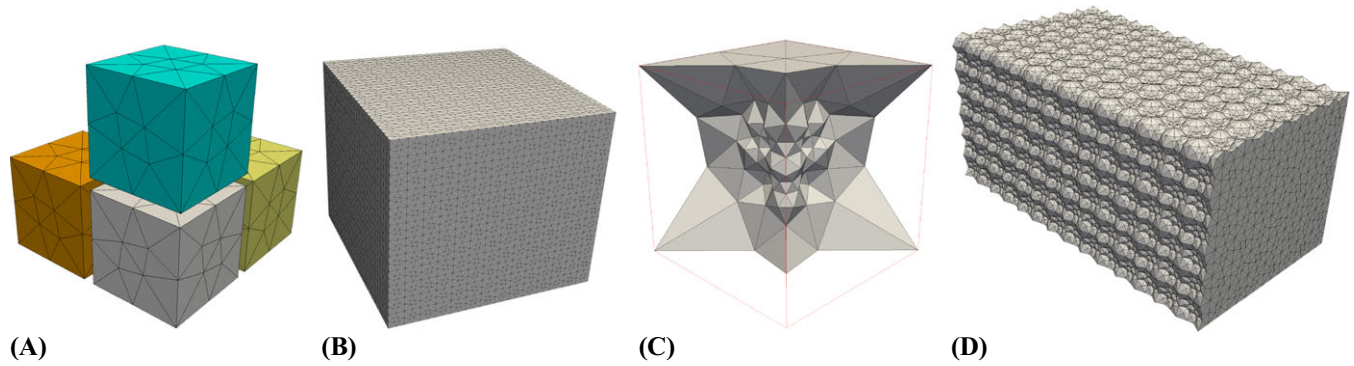


FIGURE 13 An acute triangulation of a cube from the work of VanderZee et al,⁶⁵ which we call stencil 4 in the text, is stacked as shown in (A) to construct the background mesh in (B). The stencil has rotational and translational symmetries, which permits such a stacking. A cut section of the stencil shown in (C) reveals a heterogeneous distribution of element sizes. As a consequence, elements in the mesh in (B) has a very nonuniform distribution of shapes and sizes, which is evident from the cut section shown in (D) [Colour figure can be viewed at wileyonlinelibrary.com]

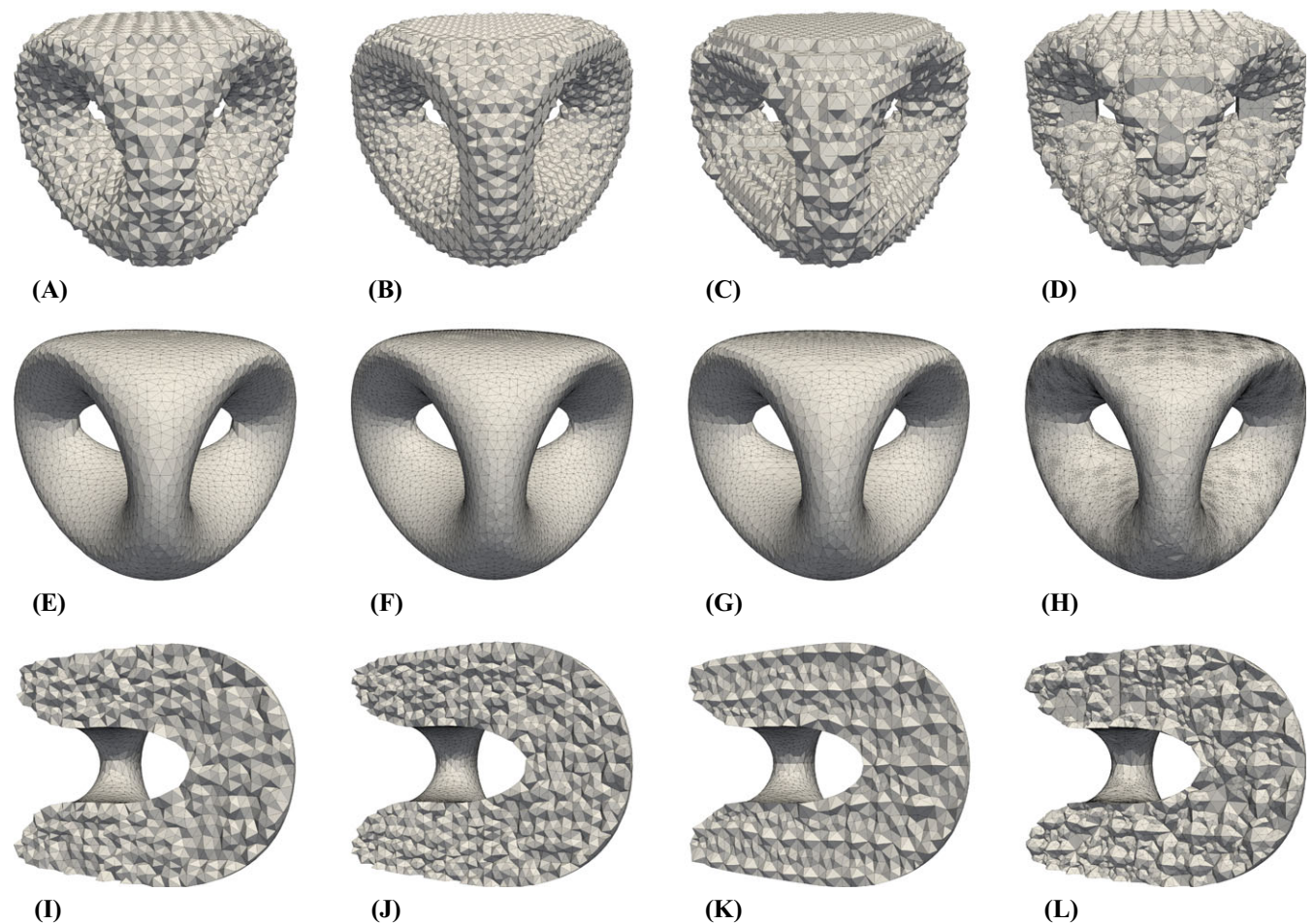


FIGURE 14 The first row of images show subsets of elements in background meshes constructed using stencils 1, 2, 3, and 4, respectively, that are identified in the um algorithm to be mapped onto Ω . The second and third rows show the conforming meshes computed by the algorithm and their cut sections, respectively. Element shapes and sizes in these images are directly correlated to element sizes/shapes in the respective background meshes. In particular, notice the nonuniform distribution of element sizes and shapes in (D), (H), and (L), which reflects those of the elements in stencil 4 depicted in Figure 13

mesh. The relatively uniform distribution of element sizes in stencils 1, 2, and 3 and the nonuniform distribution in stencil 4 are evident from the images. Qualities of these meshes are plotted in Figure 15A. Evidently, the quality of the

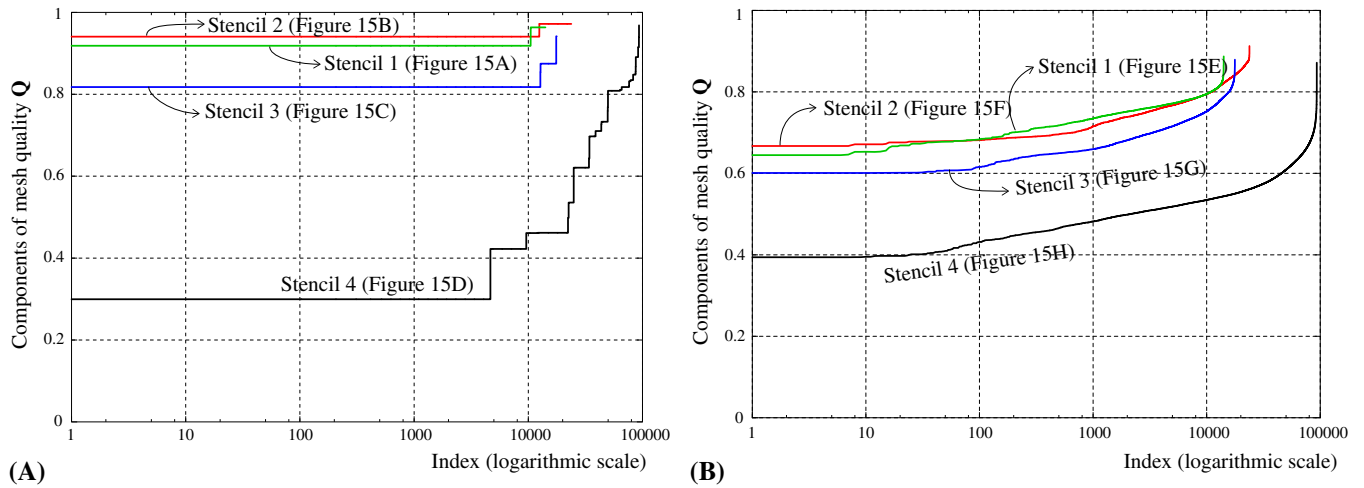


FIGURE 15 With the domain Ω immersed in each of the background meshes constructed using stencils 1, 2, 3, and 4, (A) shows plots of the qualities of subsets of elements in each background mesh identified to be mapped onto Ω . Corresponding qualities of the final meshes computed by um are plotted in (B). References to the specific meshes whose qualities are inspected are indicated in the Figure [Colour figure can be viewed at wileyonlinelibrary.com]

background mesh constructed using stencil 4 is noticeably poorer than the remaining ones, which is a direct consequence of the qualities of tets in the stencil used to construct the background mesh in the first place.

The final mesh computed for Ω with each background mesh is shown in the second row of Figure 14. Cut sections of these meshes are shown in the third row and their qualities are plotted in Figure 15B. We observe, as we did in the previous examples in this section, that the qualities of the computed meshes remain bounded away from zero. The poorest element qualities in the meshes computed using stencils 1, 2, 3, and 4 are equal 0.645, 0.667, 0.6, and 0.394, respectively.

We end this example mentioning that it is not necessary that background meshes in um be constructed using stencils. The reason behind using tessellations in our examples stems simply from the lack of a better alternative for computing background meshes with all acute tets.⁵⁸

3.7 | Implicit domain representation with SSD functions

In all the examples presented in Figures 6, 9, and 10, the domain Ω was C^2 -regular and was implicitly specified by an SSD function. Here, we discuss the details of computing such representations, which are particularly useful when data for the boundary is available not as a surface, but as a point cloud instead. In numerical simulations for instance, such a point cloud may consist of a collection of particles tracking a moving boundary/interface/crack. Alternatively, a point cloud representation for the boundary may be the result of surface reconstruction algorithms or, perhaps, represent data acquired using 3D laser-based scanners. In the ensuing discussion, we assume that in addition to the coordinates of the sample points, we also know the orientation (unit outward normal) of the boundary at these points. Hence, we consider an oriented point cloud \mathcal{P} with n points to be given, with the i th element in \mathcal{P} being a pairing (x_i, n_i) of the Cartesian coordinates x_i and the normal n_i to the boundary at x_i . Figure 16A shows the oriented point cloud consisting of $n = 5344$ points used to define the boundary for the example introduced in Section 3.1.

Evidently, the problem of defining a surface that “fits” the given data in \mathcal{P} is ill posed—uniqueness of the solution has to be contrived through an additional set of constraints. To this end, we identify the surface approximating \mathcal{P} as the zero level set of a SSD function.⁵⁷ Specifically, let $\mathcal{V} = \text{Span}\{N_a : \mathcal{D} \rightarrow \mathbb{R}\}_{a=1}^m$ be a finite dimensional function space spanned by local max-ent basis functions, where the nonempty open domain $\mathcal{D} \subset \mathbb{R}^3$ is the set over which we would like to compute the SSD representation. Consider the functional $E_{\mathcal{P}} : \mathcal{V} \rightarrow \mathbb{R}$ defined as

$$E_{\mathcal{P}}[f] \triangleq \frac{1}{2} \sum_{i=1}^n f^2(x_i) + \frac{1}{2} \sum_{i=1}^n \|\nabla f(x_i) - n_i\|^2 + \frac{\alpha}{2} \int_{\mathcal{D}} \mathbf{H}(f) : \mathbf{H}(f) dV, \quad (10)$$

where $\mathbf{H}(f) = \nabla \nabla f$ denotes the Hessian of f and α is a positive parameter. The SSD function $\psi : \mathcal{D} \rightarrow \mathbb{R}$ fitting the cloud \mathcal{P} is defined as

$$\psi \triangleq \arg \min_{f \in \mathcal{V}} E_{\mathcal{P}}[f]. \quad (11)$$

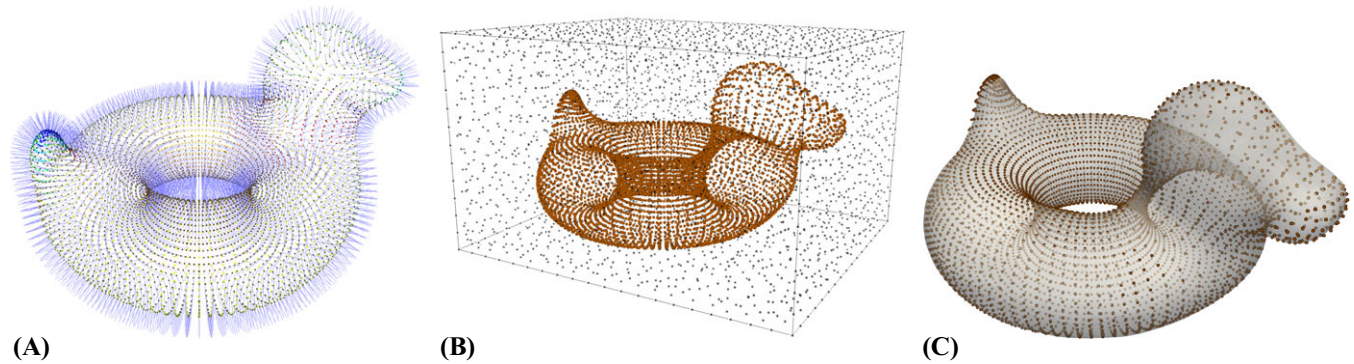


FIGURE 16 An oriented point cloud representation of the surface of a duckie-shaped domain is shown in (A). To compute an SSD representation for the domain, the point cloud is immersed in a distribution of max-ent nodes as shown in (B). In (C), we see a comparison of the computed implicit representation of the boundary (zero level set of the SSD) and its point cloud sampling [Colour figure can be viewed at wileyonlinelibrary.com]

In effect, Equation (11) transforms the discrete data \mathcal{P} provided for the boundary to an implicit domain representation ψ .

The dependence of $E_{\mathcal{P}}[f]$ on the Hessian $\mathbf{H}(f)$ reveals the reason behind choosing smooth (max-ent) basis functions to compute SSD functions. Observing the quadratic dependence of $E_{\mathcal{P}}[f]$ on f , it is straightforward to show that $E_{\mathcal{P}}$ has a unique minimizer in \mathcal{V} for any $\alpha > 0$. Consequently, ψ in (11) is well defined. Furthermore, computing ψ reduces to solving a linear system of equations for its coefficients $\{\psi_a\}_a$ in the representation $\sum_{a=1}^m \psi_a N_a$.

Implementation: We refer to the work of Arroyo and Ortiz⁶⁶ for a precise definition of local maximum entropy approximation schemes, discussions on their properties and details of their evaluation. Here, we only mention that the parameters β and ε appearing in the definitions of these functions that serve to control their rate of decay and their support are set to 1.6 and 10^{-4} , respectively, in all our examples. The choice of the node set \mathcal{N} defining the local max-ent basis is independent of the input data \mathcal{P} ; the two sets need not have any point in common. However, we ensure that \mathcal{P} is strictly contained in the interior of \mathcal{D} so that all the given data in \mathcal{P} are used. The domain \mathcal{D} is set to be the convex hull of \mathcal{N} so that all max-ent basis functions are well defined in \mathcal{D} . We use a triangulation over \mathcal{D} to compute integrals appearing in the matrix-vector system realized when resolving Equation (11). Since the basis functions $\{N_a\}_a$ have compact support, the matrix to be inverted is sparse. In our implementation, we employ an octree data structure to predetermine pairs of basis functions with overlapping support to limit the assembly of the matrix-vector system to just the nonzero entries.

Figure 16 illustrates details used to compute the SSD function for the duckie-shaped domain shown in Figure 5. The input oriented point cloud data is shown in Figure 16. The basis functions for \mathcal{V} are defined using a node set \mathcal{N} containing $m = 3550$ points that are shown in Figure 16B. The domain \mathcal{D} in this case is a parallelepiped. Figure 16C compares the data \mathcal{P} and the computed implicit representation $\psi^{-1}(0)$. Figure 17 similarly compares the input point clouds and the computed SSD representations for some of the examples appearing in Figures 9 and 10.

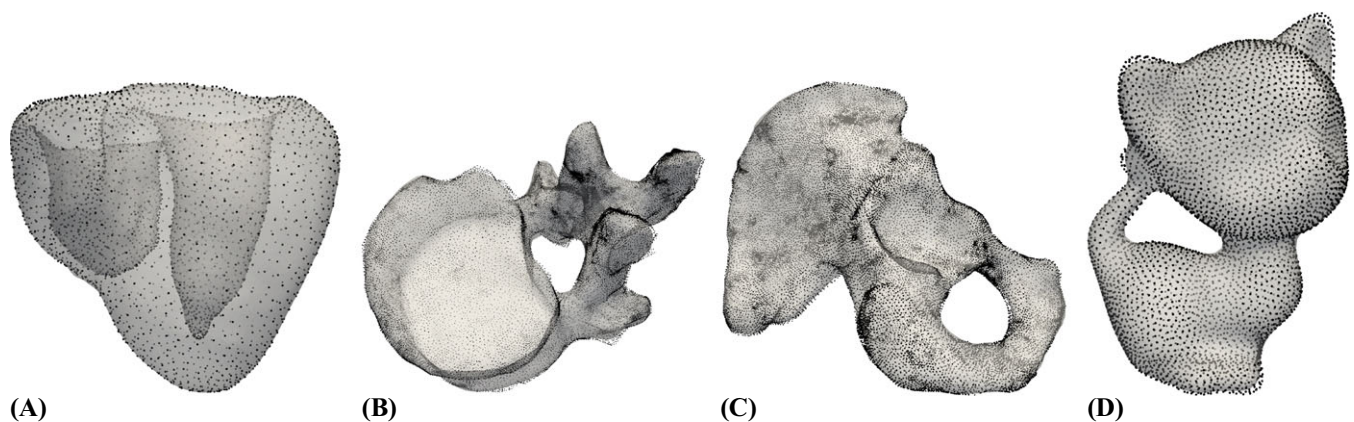


FIGURE 17 Comparisons of input point cloud data and computed SSD representations for domains considered in the examples in Section 3.5

3.8 | Practical considerations

We conclude this section mentioning a few important considerations that arise while implementing the um algorithm.

- (a) *Computing closest point projections*: To map positive vertices onto the boundary, we need to compute the closest point projection map π of $\partial\Omega$. While the specifics of this calculation depend on the representation adopted for $\partial\Omega$, we briefly mention the iterative procedure we use for computing π in the case in which $\partial\Omega$ is the zero level set of an implicit function ψ , since this is directly relevant to all our examples. Given a point x close to $\partial\Omega$, we set $t_0 = 0$ and $y_0 = x$. At the $(k + 1)$ th iteration, we do the following.

- (i) Compute the descent direction $\mathbf{w}_k = \nabla\psi(y_k)/\|\nabla\psi(y_k)\|$.
- (ii) Compute $J_{k+1} = \nabla\psi(y_k) \cdot \mathbf{w}_k$.
- (iii) Update $t_{k+1} = t_k - \psi(y_k)/J_{k+1}$.
- (iv) Update $y_{k+1} = x + t_{k+1}\mathbf{w}_k$.

These iterations are terminated if and when the values of either $|t_{k+1} - t_k|$ or $|\psi(y_k)|$ fall below a preset tolerance. The idea behind these iterations is that if the sequence $\{t_k\}_k$ converges to a limiting value t , then $t = \phi(x)$ and the limiting value y of $\{y_k\}_k$ is equal to $\pi(x)$. This follows from the observation that $y = x + t\nabla\psi(y)/\|\nabla\psi(y)\|$ by virtue of the relationship in step (iv). We remind that positive vertices at whose locations we require closest point projection calculations in um are necessarily closer to $\partial\Omega$ than the local mesh size. Therefore, $x \mapsto \pi(x)$ is well defined at these vertices when the background mesh is sufficiently refined in the vicinity of $\partial\Omega$.

- (b) *Computing vertex perturbations in dvr*: We have followed the algorithms discussed in the work of Rangarajan⁵⁶ for resolving the max-min problem in Equation (5) that defines vertex perturbations in dvr. These algorithms essentially require finding a limited set of maxima and intersections of curves of the form $\lambda \mapsto Q(K_j^\lambda)$, where j runs over indices of tets in the 1-ring of the vertex being relaxed. For the specific case of the mean ratio metric in (6), these calculations are surprisingly simple—identifying maxima reduces to computing roots of quadratic polynomials and computing intersections requires resolving polynomials of order 8 or lower. We use the GSL library⁶⁷ for polynomial root finding in our implementation.

- (c) *Identifying the submesh \mathcal{T}^{ω_h}* : In principle, the submesh \mathcal{T}^{ω_h} of \mathcal{T} can be identified by computing the inclusion map s at all the vertices in \mathcal{T} . At the outset, this is an expensive computation simply because of its direct scaling with the number of nodes in \mathcal{T} . Practical difficulties may arise in computing s particularly with parametric surface representations for $\partial\Omega$ (eg, NURBS surfaces). Even when adopting an implicit representation, evaluating the level set function to infer s may involve nontrivial computations, for instance, when ψ is defined using radial basis functions having nonlocal support.⁶⁸

To efficiently identify elements in \mathcal{T}^{ω_h} , a “crawling” algorithm discussed in the work of Kabaria and Lew²⁶ exploits the assumption that the set of positive faces Γ_h^+ and the boundary $\partial\Omega$ have identical topology. The crawling algorithm requires identifying just one positively cut tet for each connected component of Ω . Then, a recursive aggregation of edgewise neighbors of positive faces identifies the entire collection positive faces, while restricting all computations of s (and hence of ϕ and π if required) to a small neighborhood of $\partial\Omega$. Once the collection of positive faces are known, identifying the elements in \mathcal{T}^{ω_h} does not require any inspection of s ; just a recursive traversal of the neighbor list data structure of \mathcal{T} suffices.

- (d) *User-defined parameters*: Algorithms 1, 2, and 3 require a few user-defined parameters—the number of projection passes N_P , the number of relaxation iterations N_R , the number of sampling points N_S and the set of relaxation directions. These choices certainly affect the qualities of meshes computed by um, but based on extensive numerical experiments, not the general observations made in the examples we have presented here. It is to emphasize this fact that we have used an identical set of parameters in all our examples— $N_P = 5$, $N_R = 25$, $N_S = 20$ and a fixed set of relaxation directions. The most significant among these choices appears to be that for relaxation directions. Choosing an informed set of relaxation directions (eg, along the local normal and tangential directions for vertices close to the boundary) or even along randomly generated directions often yields significant improvement in mesh qualities over very few relaxation iterations.

Figure 18 highlights the performance of an implementation of um incorporating the ideas mentioned above. We draw attention to the linear scaling between the compute times and the *number of positive faces* in the background mesh. In comparison, a naïve implementation can scale linearly with the number of nodes/elements in the background mesh. The meshes shown in the figure are labeled by the genus of the bounding surfaces. The domains in the images \mathcal{G}_1 , \mathcal{G}_{18} , and

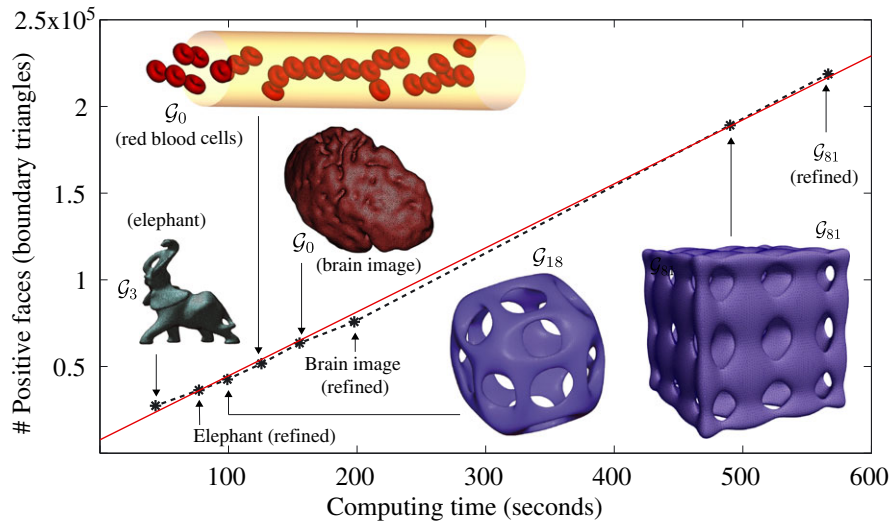


FIGURE 18 A serial implementation of the um algorithm showing approximately linear scaling of the compute time in the um algorithm with the number of positive faces in the background mesh. The times reported correspond to our C++ code executed on a 2.9 GHz Intel Core i7 processor with the GNU g++ compiler. The number of tets in the final meshes in these examples range from 273 thousand to 8.75 million. The examples shown are labeled with the genus of the bounding surface. The details of the images shown and computing times reported are discussed in the text [Colour figure can be viewed at wileyonlinelibrary.com]

\mathcal{G}_{81} are implicitly specified and have C^2 -regular boundaries. In \mathcal{G}_0 and \mathcal{G}_3 , a triangulated surface is provided as input to the meshing algorithm and we use the implementation provided by Calakli and Taubin⁵⁷ to compute their SSD functions. Unlike the calculation of SSD functions with max-ent functions we have used in the previous examples, SSD functions defined for \mathcal{G}_0 and \mathcal{G}_3 in this way are *not* C^2 functions. Nevertheless, in each of these examples, we have verified that element qualities in the computed meshes remain bounded away from zero. The times reported in the figure includes the time taken to identify the submesh \mathcal{T}^{ω_h} using the crawling algorithm, to compute closest point projections of positive vertices and to map them on to the boundary in $N_P = 5$ projection-relaxation passes while using $N_R = 5$ relaxation iterations in *dvr*. For the sake of efficiency, we use 10 iterations of a golden search algorithm for finding (suboptimal) perturbation distances to resolve (5) instead of the more involved algorithms from the work of Rangarajan.⁵⁶

4 | UNIVERSAL MESHES FOR EVOLVING DOMAINS

The examples presented in Section 3 provide encouraging evidence that um is a useful meshing algorithm in its own right. However, its advantages become most evident when meshing domains with moving or evolving boundaries. For it is now possible to immerse an evolving domain in a fixed background mesh and adopt um to compute a boundary conforming mesh at each desired instant of its evolution. Of course, it is possible that one or more of the assumptions in the algorithm are violated during the course of the evolution. For instance, the domain may fail to be C^2 -regular at or beyond certain times because of the physics governing the problem being simulated. Such scenarios may include incipient pinch-off in droplets or the kinking of cracks in solids. It is also possible that the background mesh needs to be locally refined because of the appearance of larger curvatures or smaller features in the boundary. Nevertheless, the same background mesh suffices for discretizing the geometry over time intervals that are significantly larger than the time step used in numerical schemes that advance the domain. We term such a background mesh as a *universal mesh* for the entire family of domains it can be used to triangulate with the um algorithm.

In the following examples, we demonstrate the applicability of um to meshing not one, but *entire families of domains* immersed in the same background mesh. These examples help to convey the effectiveness of um for discretizing domains in moving boundary problems.

4.1 | Meshing a deforming domain: bird flight in a universal mesh

The example in Figure 19 demonstrates an application of um for meshing a domain undergoing large deformations.

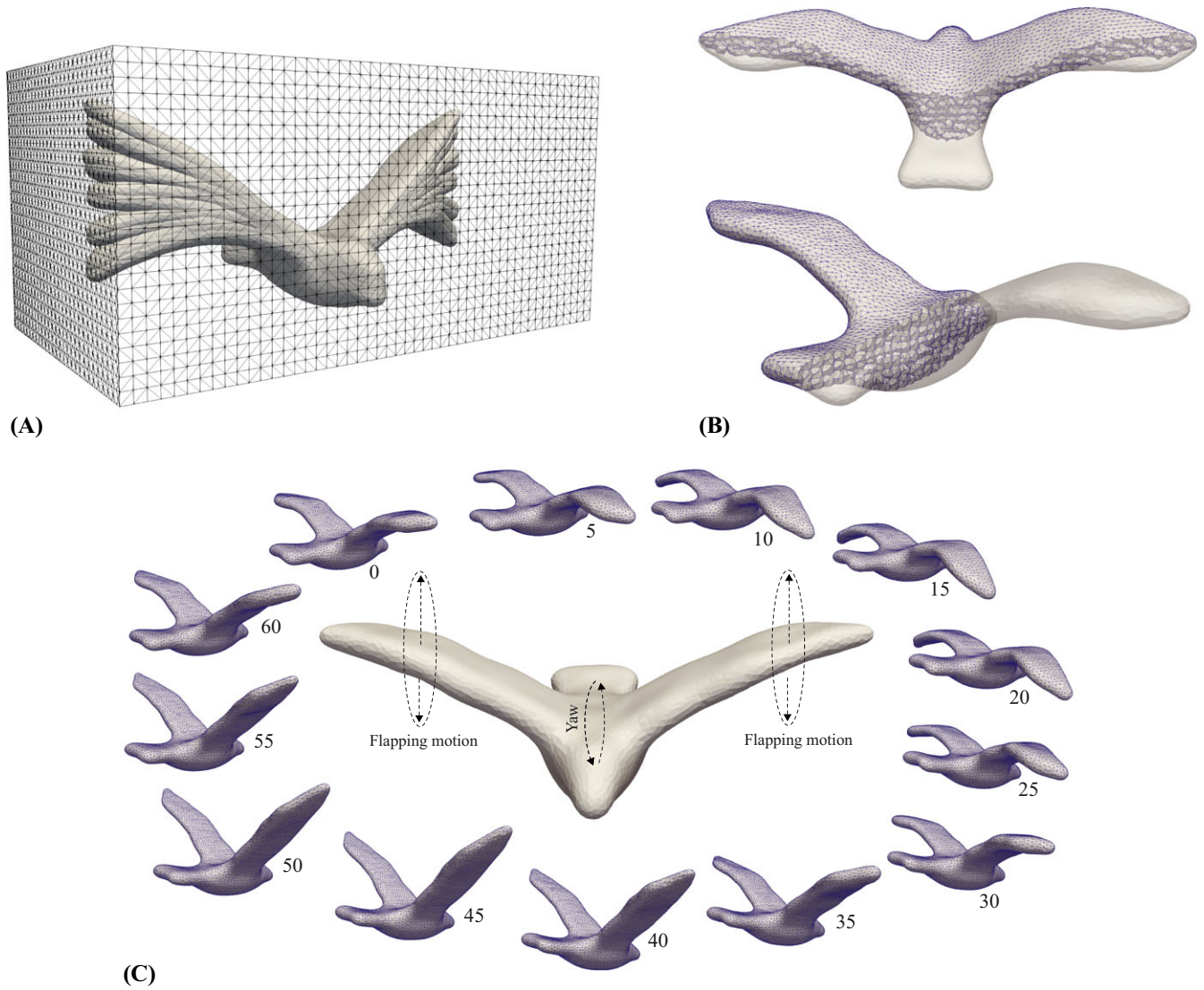


FIGURE 19 An example demonstrating the application of um to meshing evolving domains immersed in the same background mesh. The family of domains shown is defined implicitly with SSD functions to point clouds that are mapped according to the wave-like motion defined in (12). The time instants chosen are indicated in (C). Qualities of the computed meshes are plotted in Figure 21A [Colour figure can be viewed at wileyonlinelibrary.com]

Although the resemblance of the motion depicted to bird flight is entirely contrived, with some latitude, it helps to convey that the algorithm can serve as a useful tool in simulating problems involving fluid-structure interactions, for instance. For defining the evolving domain in the example, we construct a 1-parameter family of point clouds $\{\mathcal{P}_t\}_t$ by imposing a wave-like motion on an initial cloud \mathcal{P}_0 that is a sampling of the boundary of a bird-shaped domain.⁶⁹ In the point cloud \mathcal{P}_0 , the axes along the x and z coordinates are aligned with the wing span and the body of the bird, respectively, while the y coordinate is aligned with the wing flapping direction. For each point (x, y, z) in \mathcal{P}_0 , we define the corresponding point (x_t, y_t, z_t) in \mathcal{P}_t through the transformation

$$\begin{cases} x_t = x, \\ y_t = y + 0.8(\cos x - 1) \sin t + 0.1 \sin z \cos t, \\ z_t = z. \end{cases} \quad (12)$$

In Equation (12), the displacement along the y coordinate is composed of two parts—the first term proportional to $(\cos x - 1) \sin t$ mimics a flapping motion of the wings, while the second term proportional to $\sin z \cos t$ represents a yaw motion of the body. The specific constants appearing in Equation (12) are chosen to realize a reasonable motion of \mathcal{P}_0 . The family of domains $\{\Omega_t\}_t$ to be meshed is now defined implicitly using the SSD functions to the point clouds $\{\mathcal{P}_t\}_t$. Qualities of each

of the computed meshes are plotted in Figure 21A. We observe from the figure that the poorest element quality realized among all the meshes computed is about 0.657. This number should be contrasted with the minimum element qualities realized with other mesh generators in Figure 1, albeit for a different set of domains.

For realistic simulation of fluid-structure interaction problems, meshing the structure as we have done in this example is just one among many ingredients necessary for designing numerical algorithms. Directly relevant to our discussion here is the question of meshing the fluid domain around the structure to evaluate, for instance, the lift and drag forces on the wings. This can be achieved with um as well, by simply by switching the sign of the implicit function defining Ω_t .

4.2 | Meshing a domain undergoing topological changes

In the example shown in Figure 20, we consider the 1-parameter family of domains $\{\Omega_t\}_{t \geq 0}$ defined implicitly as

$$\Omega_t \triangleq \{x \in \mathbb{R}^3 : \psi_t(x) < 0\}, \quad \text{where } \psi_t(x) \triangleq \sum_{i=1}^3 (x_i^2 + \sin(tx_i)) - 1, \quad (13)$$

(x_1, x_2, x_3) represents the Cartesian components of $x \in \mathbb{R}^3$ and $t \geq 0$ is the parameter defining the evolution. In particular, the domain Ω_0 realized at time $t = 0$ is the unit sphere. Similar to the previous example, the domain undergoes large deformations as seen from the images shown in the figure. Additionally, the domain also undergoes topological changes at specific time instants, ie, the number of connected components in Ω_t changes with time as well. Notice for instance that while Ω_3 has one connected component, $\Omega_{3.2}$ has four.

Choosing a fixed background mesh (not shown), we adopt the um algorithm to mesh the sequence of domains realized at times $t = 0, 0.1, \dots, 4.5$ with the exception of $t = 3.1$ and 3.4 . A few snapshots of the computed meshes are shown in Figure 20 and the qualities of all 43 computed meshes are plotted in Figure 21B. From the plots, we find, as we did in the previous example, that the element qualities in the intermediate meshes $\hat{\mathcal{T}}^{\Omega_{t,h}}$ are bounded away from zero and that relaxing boundary nodes with ∂dvr significantly improves mesh qualities. The poorest element quality among all the meshes $\{\mathcal{T}^{\Omega_{t,h}}\}_t$ computed is 0.615.

At times close to 3.1, 3.4 and 4.5, the number of connected components of the domain change. At t close to 3.1, three new components nucleate, while at t close to 3.4, four connected components form narrow necks and fuse into one. Similarly, at t close to 4.5, additional components nucleate. In temporal intervals close to such instants at which the domain undergoes changes in topology or its boundary changes curvature drastically, the assumptions on regularity for the domain and/or on refinement of the background mesh are destined to fail. In practice, closest point projection calculations for positive vertices may also fail to converge resulting in termination of the algorithm. Even if the algorithm executes successfully, the computed meshes may be incorrect in such cases—the domain meshed may have an incorrect number of connected components for example. We caution that this should *not* be misconstrued as a failure of the algorithm—um imposes

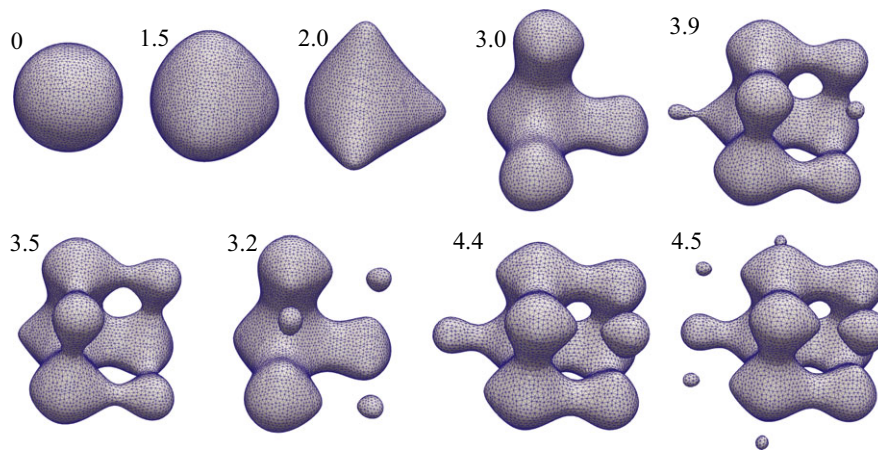


FIGURE 20 An example showing the um algorithm being used to mesh a domain that evolves according to (13). In addition to large deformations, notice that the domain undergoes frequent topological changes, as reflected by the number of its connected components. The um algorithm requires no special considerations that depend on the topology of the domain. An important caveat, however, is that close to instants when the domain undergoes topological changes, the assumptions of C^2 -regularity for it and/or sufficient refinement for the background mesh may not be satisfied [Colour figure can be viewed at wileyonlinelibrary.com]

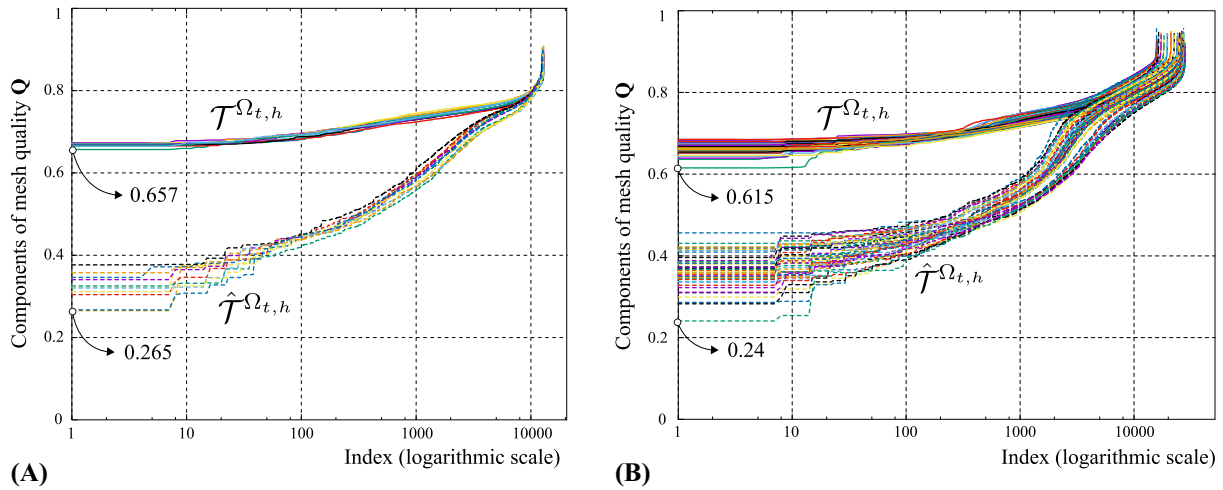


FIGURE 21 Qualities of meshes computed by um for the families of domains shown in Figures 19 and 20 are plotted in (A) and (B), respectively [Colour figure can be viewed at wileyonlinelibrary.com]

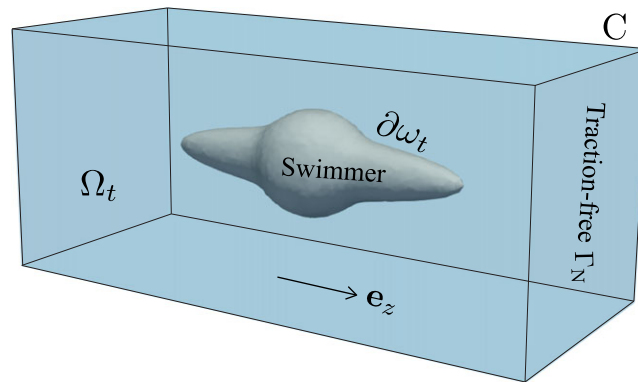


FIGURE 22 An illustration depicting the setup for the problem of simulating the motion of a swimmer in a creeping flow that is discussed in Section 4.3. A self-propelling object, whose time-dependent boundary is denoted by $\partial\omega_t$, is immersed in a viscous fluid in a rectangular channel C . The length of the channel is 29.16 units while its cross section is a square of side 12 units. The domain occupied by the fluid in the channel is denoted by Ω_t . The fluid is subject to no-slip boundary conditions on all but one wall of the channel, where a traction-free boundary condition is imposed instead. No slip boundary conditions are similarly imposed along the fluid-swimmer interface [Colour figure can be viewed at wileyonlinelibrary.com]

no restrictions on the topology of the domain. Rather, these simply represent scenarios in which the assumptions in the algorithm are not satisfied. Special techniques are invariably needed to mesh domains with singular (ie, infinitesimally small) features.^{70,71}

4.3 | Swimmer in a creeping flow

Next, we present an example simulating the motion of a self-propelling object (eg, a microorganism) in a creeping flow. Besides reinforcing the point made in the previous two examples on the applicability of the um algorithm for meshing evolving geometries, this example serves to bridge its connection to numerical methods for simulating problems involving moving boundaries.

The setup for the problem, which is illustrated in Figure 22, consists of a swimmer immersed in a fluid of viscosity μ contained in a rectangular channel C . The length of the channel is oriented parallel to the \mathbf{e}_z axis and the origin of coordinates coincides with the center of the channel. Five of the six faces of the channel are sealed, where the fluid is subject to no-slip boundary conditions. The remaining end face of the channel, indicated by Γ_N in the figure, allows intake and expulsion of the fluid and we impose a traction-free boundary condition for the fluid along this face.

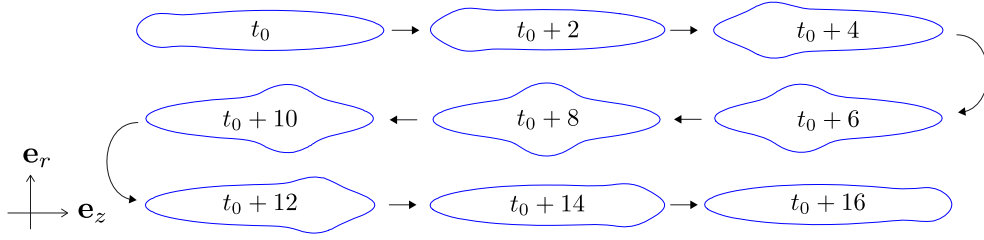


FIGURE 23 Snapshots of surface deformations of the swimmer that is assumed in (14). Noting that the deformations are axisymmetric, we visualize the swimmer's boundary in the $r - z$ plane. The time scale of surface deformations is set implicitly by (14) and t_0 represents a reference time instant. Observe that an approximately circular “bulge” simultaneously grows/shrinks in size and propagates along the length of the swimmer over the depicted time interval [Colour figure can be viewed at wileyonlinelibrary.com]

We begin with an ansatz for the motion of the surface of the swimmer. We assume that the geometry of the swimmer at time t , modulo rigid body motions, is the zero sublevel set of the implicit function

$$\phi_t(r, z) \triangleq 1 - \frac{2}{\pi} \arctan\left(\frac{r^2}{2} + \frac{z^2}{64} - 1\right) - \frac{2}{\pi} \arctan\left(r^2 + (z - t)^2 - \frac{25}{4} \cos^2 \frac{\pi t}{16}\right), \quad (14)$$

where (x, y, z) is the usual Cartesian system of coordinates in \mathbb{R}^3 and $r = \sqrt{x^2 + y^2}$. While we simply assume Equation (14) here, such motions can be reverse engineered from experimental observations of active soft matter.⁷² The pulsatile motion represented by Equation (14) is visualized at discrete time instants in Figure 23. We additionally assume that the motion in Equation (14) is directed along the normal to the surface at each time instant. Hence, particles on the surface of the swimmer do not undergo any tangential motions. Consequently, the spatial velocity field $\mathbf{v}_t(r, z)$ of the surface of the swimmer is given by

$$\mathbf{v}_t = -\frac{\partial \phi_t}{\partial t} \frac{\nabla \phi_t}{\|\nabla \phi_t\|^2}. \quad (15)$$

Evidently, the surface motion of the swimmer induces flow around it in the channel, which we assume to be of negligibly small Reynolds number. Hence, the velocity \mathbf{u}_t and the pressure p_t of the fluid satisfy the momentum balance and incompressibility conditions. The dependence of the flow solution (\mathbf{u}_t, p_t) on time, as indicated by the subscript t , arises from kinematic continuity (no-slip boundary condition) imposed along the moving surface of the swimmer and, in particular, is not due to any inertial effects in the fluid.

Fluid flow around the swimmer, in turn, exerts a net force on the swimmer, which, based on symmetry considerations, we assume to be directed along the \mathbf{e}_z axis. Our goal in this example is to determine the spatially uniform axial velocity that the swimmer needs to generate in addition to its pulsatile motion so that it remains at force-free instantaneous equilibrium with the surrounding fluid at all times. That is, we seek to compute the motion that the swimmer should maintain along the axis of the channel so that it does not experience any external forces due to the flow around it.

Description of the problem: We are now ready to concisely state the problem we solve. Let ω_t denote the closed region occupied by the swimmer at time t , with its initial configuration ω_0 assumed to be given. Denote the region occupied by the fluid at time t by $\Omega_t \triangleq C \setminus \omega_t$. We seek a function $t \mapsto z_0(t) \in \mathbb{R}$ such that

- (i) we have the initial condition $z_0(t_0) = 0$,
- (ii) ω_t is the zero sublevel set of ϕ_t translated by $z_0(t)\mathbf{e}_z$, ie, $\omega_t \triangleq \{(x, y, z) \in \mathbb{R}^3 : \phi_t(\sqrt{x^2 + y^2}, z - z_0(t)) \leq 0\}$,
- (iii) (\mathbf{u}_t, p_t) satisfy the balance equations

$$\begin{cases} \mu \Delta \mathbf{u}_t - \nabla p_t = 0 \\ \nabla \cdot \mathbf{u}_t = 0 \end{cases} \quad \text{in } \Omega_t, \quad (16)$$

- (iv) (\mathbf{u}_t, p_t) satisfies

$$\begin{cases} \mathbf{u}_t = 0 & \text{on } \partial C \setminus \Gamma_N, \\ \boldsymbol{\sigma}_t \mathbf{e}_z = 0 & \text{on } \Gamma_N, \end{cases} \quad (17)$$

where $\boldsymbol{\sigma}_t = \mu(\nabla \mathbf{u}_t + \nabla \mathbf{u}_t^T) - p_t \mathbb{I}$ is the stress in the fluid. Along the swimmer boundary, we impose no-slip boundary conditions, so that

$$\mathbf{u}_t(\cdot, z) = \mathbf{v}_t(\cdot, z - z_0(t)) + \dot{z}_0(t)\mathbf{e}_z \quad \text{along } \partial \omega_t, \quad (18)$$

(v) and the net force on the swimmer along the \mathbf{e}_z direction is zero, ie,

$$\mathbf{F}_t \triangleq \int_{\partial\omega_t} \mathbf{e}_z \cdot \left(\frac{\mu}{2} (\nabla \mathbf{u}_t + \nabla \mathbf{u}_t^T) - p_t \mathbb{I} \right) \mathbf{n} d\Gamma = 0, \quad (19)$$

where \mathbf{n} denotes the unit outward normal to $\partial\omega_t$.

In effect, the system in Equations (16) to (19) defines the translatory motion $t \mapsto z_0(t)$ of the swimmer along the axial direction.

The motion of the swimmer consists of two parts—a pulsatile motion that is assumed and a translatory motion that is computed to satisfy (16) to (19).

Solution: At first sight, it may appear that Equations (16) to (19) represent a coupled system of equations for $(\mathbf{u}_t, p_t, \dot{z}_0(t))$ at each instant t . However, we exploit the linearity of the problem in (16), the homogeneous nature of boundary conditions along the channel walls in (17), the affine dependence of the no-slip boundary condition along $\partial\omega_t$ on the unknown velocity $\dot{z}_0(t)$ in (18) and the linear dependence of \mathbf{F}_t on (\mathbf{u}_t, p_t) in (19) to follow a simple solution strategy. Specifically, these observations suggest the splitting

$$(\mathbf{u}_t, p_t) = \left(\mathbf{u}_t^\phi, p_t^\phi \right) + \dot{z}_0(t) \left(\mathbf{u}_t^z, p_t^z \right), \quad (20)$$

where both $(\mathbf{u}_t^\phi, p_t^\phi)$ and (\mathbf{u}_t^z, p_t^z) satisfy Equations (16) and (17). Along the boundary $\partial\omega_t$ of the swimmer, however, we set $\mathbf{u}_t^\phi(\cdot, z) = \mathbf{v}_t(\cdot, z - z_0(t))$ and $\mathbf{u}_t^z = \mathbf{e}_z$. Notice that the fields $(\mathbf{u}_t^\phi, p_t^\phi)$ and (\mathbf{u}_t^z, p_t^z) computed this way depend on the domains ω_t, Ω_t and therefore on $z_0(t)$ but, crucially, are independent of the unknown velocity $\dot{z}_0(t)$. Then, from (19) and (20), we get

$$\mathbf{F}_t = \int_{\partial\omega_t} \mathbf{e}_z \cdot \left(\frac{\mu}{2} (\nabla \mathbf{u}_t^\phi + \nabla \mathbf{u}_t^{\phi,T}) - p_t^\phi \mathbb{I} \right) \mathbf{n} d\Gamma + \dot{z}_0(t) \int_{\partial\omega_t} \mathbf{e}_z \cdot \left(\frac{\mu}{2} (\nabla \mathbf{u}_t^z + \nabla \mathbf{u}_t^{z,T}) - p_t^z \mathbb{I} \right) \mathbf{n} d\Gamma \triangleq \mathbf{F}_t^\phi + \dot{z}_0(t) \mathbf{F}_t^z, \quad (21)$$

where \mathbf{F}_t^ϕ represents the net force exerted by the flow on the swimmer due to its pulsatile surface motion, whereas \mathbf{F}_t^z represents the force exerted when the swimmer translates with a uniform velocity directed along \mathbf{e}_z . In this way, we find from Equation (20) that $\dot{z}_0(t) = -\mathbf{F}_t^\phi / \mathbf{F}_t^z$.

Discretization and simulation: Realizing the solution strategy discussed above requires approximating two pairs of solutions at each instant t , namely, $(\mathbf{u}_t^\phi, p_t^\phi)$ and (\mathbf{u}_t^z, p_t^z) , over the fluid domain Ω_t . The two solutions differ from each other only in the boundary condition imposed along $\partial\omega_t \subset \partial\Omega_t$. One of the main challenges in approximating these fields lies in the fact that the fluid domain Ω_t evolves with time—both due to the prescribed surface motion of the swimmer and due to its translation computed along the axis of the channel. The um algorithm addresses precisely this issue and is ideally suited to triangulate the evolving fluid geometry in this problem. We simply choose a universal mesh that fits the geometry of the channel and recover a mesh conforming to the fluid domain at each discrete time instant sampled in the numerical simulation.

The simulation scheme is now straightforward to describe. Let $t_0 < t_1 < t_2 < \dots$ be a sequence of discrete time instants in strictly increasing order. At a specific instant t_k , given $z_0(t_k)$, we compute $\dot{z}_0(t_k)$ as follows.

- (i) Triangulate the fluid domain Ω_{t_k} with the um algorithm and the fixed universal mesh.
- (ii) Compute (approximate) the fields $(\mathbf{u}_{t_k}^\phi, p_{t_k}^\phi)$ and $(\mathbf{u}_{t_k}^z, p_{t_k}^z)$ with a Stokes flow solver.
- (iii) Compute the forces $\mathbf{F}_{t_k}^\phi$ and $\mathbf{F}_{t_k}^z$ exerted on the swimmer using the fields computed in step (ii), see (21).
- (iv) Set $\dot{z}_0(t_k) = -\mathbf{F}_{t_k}^\phi / \mathbf{F}_{t_k}^z$.
- (v) (Forward Euler) Update $z_0(t_{k+1}) = z_0(t_k) + (t_{k+1} - t_k) \dot{z}_0(t_k)$ and proceed to the next time instant.

Noting that $z_0(t_0)$ is given, simulating the motion of the swimmer simply involves iterating over steps (i)-(v). In our simulations, we choose the time step to be approximately one unit.

We adopt the well-known stabilized finite element method^{73,74} as the Stokes solver to approximate $(\mathbf{u}_t^\phi, p_t^\phi)$ and (\mathbf{u}_t^z, p_t^z) at each instant. Specifically, each component of the fluid velocity and the pressure are interpolated with continuous piecewise affine functions. This choice is commonly referred to as the stabilized P1/P1 element in the literature. The stabilization constant is set to be $h^2/12\mu$, which, notably, scales with the square of the mesh size. We mention that the stabilized method introduces a small consistency error in the formulation but retains optimal convergence properties. We refer to the work of Brezzi and Fortin⁷⁵ for alternative formulations and stable element pairs suitable for the Stokes problem.

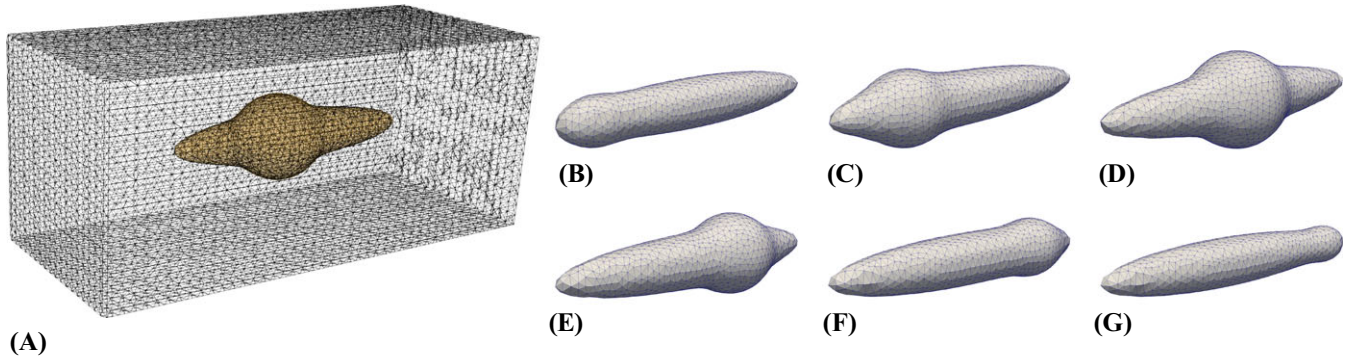


FIGURE 24 An outline of the mesh conforming to the fluid domain at a representative time instant is shown in (A). Figures (B) Δt ; (C) $4\Delta t$; (D) $8\Delta t$; (E) $11\Delta t$; (F) $13\Delta t$; (G) $16\Delta t$ show snapshots of the discretized swimmer surface realized at different time instants during the simulation [Colour figure can be viewed at wileyonlinelibrary.com]

An outline of the mesh conforming to the boundaries of the channel walls and the swimmer surface at a representative time instant is shown in Figure 24A. The figure also shows the triangulated swimmer surfaces realized at different time instants in the simulation. A minor caveat in our simulation is that the universal mesh constructed over the channel using the stencil shown in Figure 5B has a small bevel (side approximately equal to the mesh size) along two edges of the face Γ_N . This occurs because of the geometric structure/shape of the stencil in Figure 5B used to construct the background mesh over the channel. We presume that the influence of such a small geometric defect along the channel wall on the flow and, in turn, on the swimmer itself will be negligible.

With the viscosity of the fluid set to equal 1 unit, Figure 25 shows snapshots of the flow solutions $(\mathbf{u}_t^\phi, p_t^\phi)$ and (\mathbf{u}_t^z, p_t^z) over a section of the fluid domain computed at a representative time instant during the simulation. The only difference between the fields shown in Figures 25A and 25B is the boundary condition imposed along the swimmer wall. In the former, the imposed velocity is exclusively due to the pulsatile motion described in Equation (14) while in the latter, we impose an axial velocity with unit magnitude. Figure 26A shows the plot of the velocity $\dot{z}_0(t)$ computed by the simulation that renders the swimmer free from any axial forces due to the fluid around it. Figure 26B shows the translation of the swimmer $z_0(t)$ along the axis the channel. Each data point in the plots represents the result of resolving a pair of Stokes problems over a fluid domain that evolves with time. The um algorithm enables all these simulations to be performed using a fixed tetrahedral mesh of the geometry of the channel. In fact, since the connectivity of the mesh computed by um for the fluid domain at each time instant is a subset of that of the universal mesh, sparsity patterns of data structures used in the Stokes solver (ie, the finite element stiffness matrices and preconditioners) can be retained throughout the simulation.

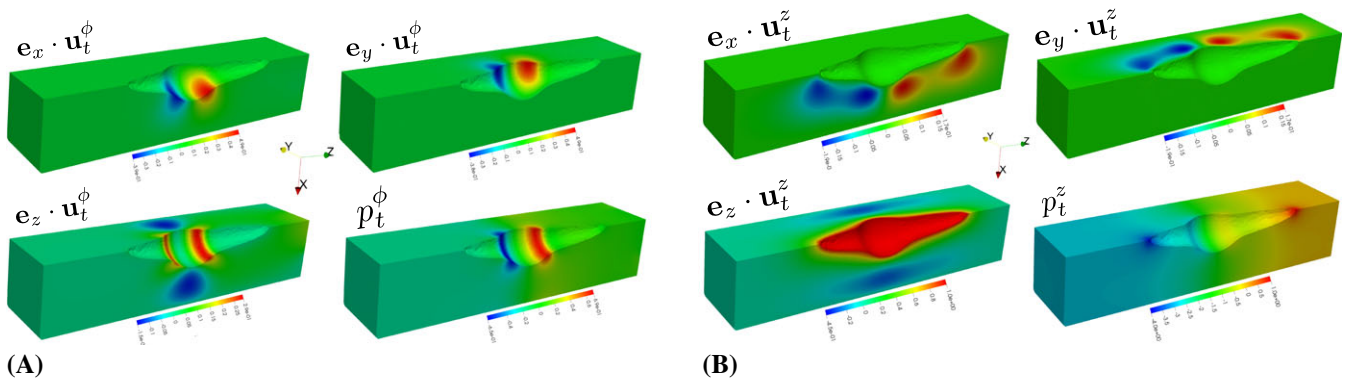


FIGURE 25 Snapshots of the flow fields around the swimmer computed at a specific time instant ($t = t_0 + 7\Delta t$) in the numerical simulation. The images show contours of the computed flow fields $(\mathbf{u}_t^\phi, p_t^\phi)$ and (\mathbf{u}_t^z, p_t^z) over a section of the fluid domain [Colour figure can be viewed at wileyonlinelibrary.com]

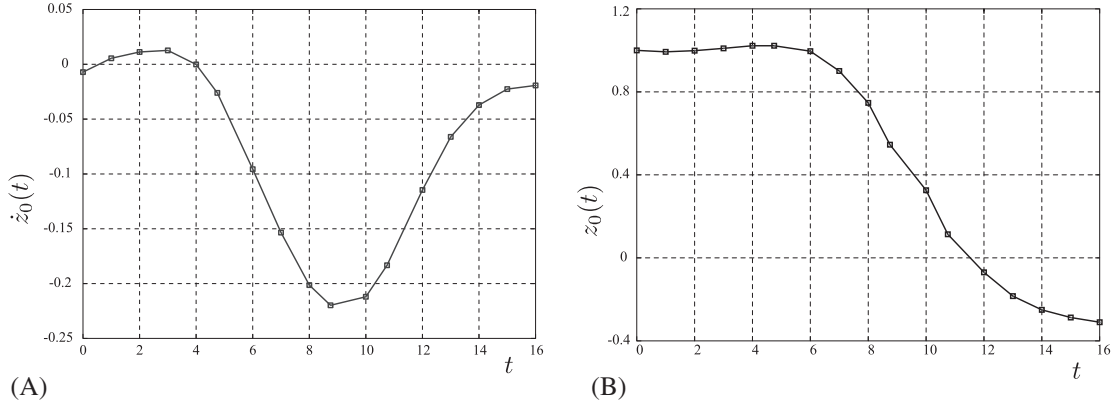


FIGURE 26 The main result of the simulation of the swimmer is the velocity profile $t \mapsto \dot{z}_0(t)$ shown in (A), representing the axial velocity that the swimmer needs to generate in order to remain at instantaneous equilibrium with the ambient fluid despite the pulsatile motion of its surface. The computed translation $z_0(t)$ is plotted in (B)

5 | A DISCUSSION OF THE RATIONALE BEHIND THE UM ALGORITHM

A rigorous analysis of the um algorithm is pending at this time. Our goal in this section is simply to convey the intuition underlying the algorithm. Although the discussion is necessarily a qualitative one, we mention a few intermediate results that corroborate the algorithm's performance observed in the numerical experiments presented here. We highlight important questions whose resolution would help shed light on the issue of the algorithm's robustness. For convenience in explanations, we shall assume in the remainder of this section that Ω is a connected domain; the discussions apply verbatim for domains with multiple components as well.

5.1 | The algorithm as a mapping between domains

As suggested in Figure 2, um can be interpreted as an algorithm that constructs a mapping between the domains ω_h and Ω_h . We introduce this notion solely for the purpose of discussing the um algorithm and note that it is not required anywhere in its implementation. We start by recognizing that the sequence of vertex perturbations in um defines a corresponding sequence in which successive pairs of meshes differ in the location of a single vertex. Let us label these meshes as $\{\mathcal{T}^{\omega_h^k}\}_{k=0}^N$ and denote the corresponding sequence of domains they triangulate by $\{\omega_h^k\}_{k=0}^N$, so that $\mathcal{T}^{\omega_h^0} = \mathcal{T}^{\omega_h}$, $\mathcal{T}^{\omega_h^N} = \mathcal{T}^{\Omega_h}$, $\omega_h^0 = \omega_h$ and $\omega_h^N = \Omega_h$. Hence, N is equal to the cumulative number of vertex perturbations performed in the algorithm. Then, following the steps in the algorithm, we are led to examining the sequence of transformations

$$\omega_h \triangleq \omega_h^0 \xrightarrow{F_1} \omega_h^1 \xrightarrow{F_2} \omega_h^2 \cdots \omega_h^{N-1} \xrightarrow{F_N} \omega_h^N \triangleq \Omega_h, \quad (22)$$

where the map F_{k+1} computes ω_h^{k+1} by perturbing a single vertex, say, that with index i_{k+1} , in the mesh $\mathcal{T}^{\omega_h^k}$. Consequently, each map $F_{k+1} : \omega_h^k \rightarrow \omega_h^{k+1}$ is continuous and piecewise affine, and differs from the identity just over the elements in the 1-ring over vertex i_{k+1} in the mesh $\mathcal{T}^{\omega_h^k}$. With this interpretation in mind, analyzing the um algorithm consists in examining the properties of the domain ω_h and the maps $\{F_k\}_{k=1}^N$.

From Equation (22) and Algorithm 1, it is clear that the collection of maps $\{F_k\}_k$ fall into one of three categories—ones that perturb positive vertices towards their closest point projections on $\partial\Omega$, ones that relax vertices in I_R along prescribed directions with dvr and ones that relax positive vertices constrained to lie on the boundary with ∂dvr .

5.2 | General observations

By virtue of Equation (22), we have

$$\Omega_h = \underbrace{F_N \circ F_{N-1} \circ \cdots \circ F_1}_{M_h}(\omega_h), \quad (23)$$

where the composition of the sequence of vertex perturbations is denoted by $M_h : \omega_h \rightarrow \Omega_h$. We can immediately identify a few requirements on ω_h and M_h from Equation (23). For the mesh \mathcal{T}^{Ω_h} to be devoid of any inverted, degenerate or overlapping elements, it is necessary that M_h be an injective map. For this, it suffices to ensure that each map $F_k, k = 1, \dots, N$, is injective, which, in turn, translates to saying that no intermediate mesh realized in the um algorithm can have inverted, degenerate or overlapping elements. In such a case, \mathcal{T}^{Ω_h} is a valid mesh over the polyhedral domain Ω_h in the usual sense.⁵²

The next observation concerns the question of whether the final mesh \mathcal{T}^{Ω_h} is a “good/reasonable” discretization of Ω . At the very least, we can request that $\Omega_h(\partial\Omega_h)$ have identical topology as $\Omega(\partial\Omega)$ (they are homeomorphic) and that vertices of $\partial\Omega_h$ lie on $\partial\Omega$. Of course, these are in addition to the requirement that \mathcal{T}^{Ω_h} be a valid triangulation of Ω_h and that its elements have good qualities. Then, in particular, since M_h is continuous (being a composition of continuous maps), we deduce that a necessary condition for um to compute a good mesh for Ω is that $\omega_h(\partial\omega_h)$, in turn, have identical topology as $\Omega(\partial\Omega)$.

The above remarks translate to clear questions in the context of the um algorithm. Are each of the mappings $F_k, k = 1, \dots, N$, realized in um injective? Having defined \mathcal{T}^{ω_h} to be the collection of all tets in \mathcal{T} having at least one vertex in Ω , do $\omega_h(\partial\omega_h)$ and $\Omega(\partial\Omega)$ have identical topology? Recognizing that only positive vertices are projected onto $\partial\Omega$ in the algorithm, do the set of positive faces Γ_h^+ coincide with $\partial\omega_h$? Fortunately, these questions can be verified a posteriori in numerical experiments. We can therefore affirm that each of these requirements on the algorithm are indeed satisfied in all the numerical experiments discussed in Sections 3 and 4. We are of course interested in identifying conditions on the background mesh \mathcal{T} and on the domain Ω that will help us establish these claims a priori. In the following, we discuss a couple of intermediate results available to this end. Specifically, we discuss conditions under which the maps $\{F_k\}_k$ are injective.

5.3 | Closest point projections and conditioning angles

Suppose that the mesh $\hat{\mathcal{T}}^{\Omega_h}$ in Algorithm 1 is realized after n vertex perturbations, so that we have $\mathcal{T}^{\omega_h^n} = \hat{\mathcal{T}}^{\Omega_h}$ and

$$\omega_h^n = \underbrace{F_n \circ F_{n-1} \circ \dots \circ F_1}_{\hat{M}_h}(\omega_h).$$

At this stage of the algorithm, each positive vertex in the mesh \mathcal{T}^{ω_h} has been projected onto $\partial\Omega$. For the composite map \hat{M}_h to be injective (so that the mesh $\hat{\mathcal{T}}^{\Omega_h}$ is a valid triangulation), it is necessary then that the restriction of \hat{M}_h to the set of positive faces Γ_h^+ be injective in particular. Noticing that the values of \hat{M}_h at the positive vertices equals their closest point projection on $\partial\Omega$, \hat{M}_h is a piecewise affine interpolant of the closest point projection map π onto $\partial\Omega$. We are hence led to the question of understanding whether the restriction π to Γ_h^+ is injective. Indeed, demonstrating $\pi : \Gamma_h^+ \rightarrow \partial\Omega$ to be a homeomorphism is a crucial step in analyzing the um algorithm.

Observe that for $\pi : \Gamma_h^+ \rightarrow \partial\Omega$ to be injective, it is necessary that the restriction of π to each positive face be injective. The restriction on conditioning angles in positively cut tets is motivated to ensure precisely this requirement. Referring to the positively cut tet K shown in Figure 3C having positive face $\overline{V_1V_2V_3}$ with its vertex $V_4 \in \Omega$, we call the vertex of the positive face closest to $\partial\Omega$ as the *proximal vertex* of K . In the figure, vertex V_1 is closer to $\partial\Omega$ than V_2 and V_3 and is therefore indicated to be the proximal vertex of K . Three faces of a positively cut tet meet at its proximal vertex. With each face oriented such that its normal points into the element, we call the dihedral angles between the positive face and the remaining two faces incident at the proximal vertex as the *conditioning angles* of the element. Referring to the figure, the *conditioning angles* of K are the pair of dihedral angles labeled as ϑ_1 and ϑ_2 , measured between the positive face $\overline{V_1V_2V_3}$ and the faces $\overline{V_1V_2V_4}$ and $\overline{V_1V_3V_4}$, respectively.

It is proved in Theorem 3.2 in the work of Kabaria and Lew²⁶ that if the local mesh size in the vicinity of $\partial\Omega$ is sufficiently small and if conditioning angles are strictly acute, then the restriction of π to each positive face in Γ_h^+ is injective. Figure 27A shows an example in which π fails to be injective on the positive face of a tet having an obtuse conditioning angle. Extending the claim of injectivity of π to the set Γ_h^+ is nontrivial, as the analogous 2D case²⁴ shows. Nevertheless, numerical experiments such as the ones presented here provide encouraging evidence supporting this conjecture.

5.4 | Dvr iterations

The second category of vertex perturbations in the um algorithm consists of relaxations performed with dvr and ∂dvr . Let us first consider the former. For definiteness, let us suppose that the $(k + 1)$ th vertex perturbation in the algorithm,

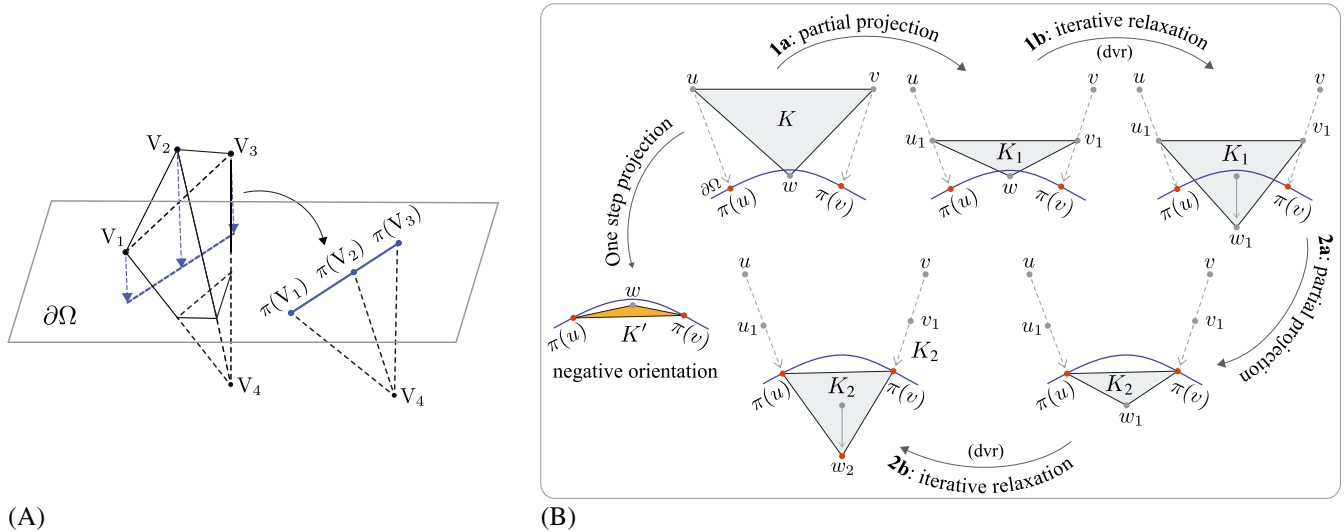


FIGURE 27 The example in (A) shows that without restricting conditioning angles in a positively cut tet to be strictly acute, projecting its positive vertices onto $\partial\Omega$ can result in degenerate or inverted elements. Projecting the positive vertices V_1, V_2 , and V_3 onto $\partial\Omega$ results in a degenerate face. Figure (B) illustrates the possibility of element inversion that is discussed in Section 5.5. For visualization sake, we depict the problem with a triangle rather than with a tet. Notice that projecting positive vertices u and v of K onto $\partial\Omega$ in one go yields the inverted triangle K' . Instead, projecting u and v onto $\partial\Omega$ over multiple steps, while also relaxing the interior vertex w with dvr, helps to alleviate the problem and yields the well-shaped triangle K_2 [Colour figure can be viewed at wileyonlinelibrary.com]

represented by the transformation $F_{k+1} : \omega_h^k \rightarrow \omega_h^{k+1}$, consists in relaxing the vertex $i_{k+1} \in \mathbb{I}_R$ with dvr. The question of injectivity of F_{k+1} is addressed by Theorem 4.2 in the work of Rangarajan and Lew,²² where it is shown that if tets in the mesh $\mathcal{T}^{\omega_h^k}$ all have positive qualities, then so do the tets in $\mathcal{T}^{\omega_h^{k+1}}$. In other words, if the mesh $\mathcal{T}^{\omega_h^k}$ is a valid triangulation, then so is the relaxed mesh $\mathcal{T}^{\omega_h^{k+1}}$ computed by dvr. This result is perhaps to be expected, since the vertex i_{k+1} is relaxed in such a way that the minimum among the qualities of tets in its 1-ring is improved, see Equation (5). It is, however, important to recognize that relaxing i_{k+1} in this way does not simply improve the qualities of all elements around it. Rather, a few element qualities are improved at the expense of others. It is therefore necessary to examine the effect of vertex relaxations on the quality vector \mathbf{Q} of the mesh as a whole. Indeed, the result cited shows that the quality vector $\mathbf{Q}(\mathcal{T}^{\omega_h^{k+1}})$ of the relaxed mesh is better than $\mathbf{Q}(\mathcal{T}^{\omega_h^k})$ according to a specific ordering relation. A similar justification also follows based on Theorem 4.10 in the work of Rangarajan and Lew²² for mesh improvement with ∂dvr . In summary, vertex perturbations with dvr and ∂dvr necessarily improve the mesh quality and, in particular, preserve the validity of the meshes they take as input.

5.5 | Qualities of intermediate meshes

Let us revisit the set of perturbations of positive vertices in step 26 of the um algorithm. Let us suppose that the sequence of these perturbations correspond to the realization of the transformed domains $\omega_h^n \mapsto \dots \mapsto \omega_h^{n+m}$, where $m = \#\mathbb{I}^+$. Even if positive faces remain nondegenerate during these perturbations as discussed in Section 5.3, it is still possible that a tet in $\mathcal{T}^{\omega_h^{n+m}}$ having one or more positive vertices gets inverted. This possibility is illustrated in Figure 27B using a 2D example of a positively cut triangle K for easy visualization. The positive face \overline{uv} remains nondegenerate when the positive vertices u and v are projected onto $\partial\Omega$ but the resulting triangle K' is inverted, ie, has the opposite orientation of K . The figure suggests that this happens because of the large perturbations of the positive vertices compared to the size of K . The analogous issue persists in the case of tets as well, since at least one vertex in each tet being perturbed during the transformation $\omega_h^n \mapsto \omega_h^{n+m}$ remains fixed. Then, realizing that the magnitude of perturbations of positive vertices can be comparable to the local mesh size reveals that some elements in $\mathcal{T}^{\omega_h^{n+m}}$ can indeed get inverted. Such inversions would of course be detected during the subsequent call to the dvr algorithm in step 28, where the input mesh is checked for $\mathbf{Q} > 0$, resulting in termination of the algorithm itself.

It is precisely to alleviate the aforementioned possibility of element inversions during positive vertex projections that we resort to a multiple pass projection strategy in Algorithm 1, which is further exemplified by (1). The underlying heuristic

is conceptually illustrated in Figure 27B. Rather than projecting positive vertices u and v of K onto $\partial\Omega$ in a single step, we subdivide the perturbations $u \mapsto \pi(u)$ and $v \mapsto \pi(v)$ into small increments, while also intermittently relaxing the interior vertex w with dvr . This strategy helps to alleviate the inversion of K to K' and yields the well-shaped triangle K_2 shown in the figure. We adopt this idea in Algorithm 1, where small perturbations of positive vertices in step 26 are immediately followed by dvr iterations with the anticipation that the qualities of elements in the intermediate meshes realized remain positive. It is possible, especially when dvr iterations do not yield sufficient mesh improvement, that such a strategy can fail to prevent mesh entanglement.

6 | CONCLUDING REMARKS

We have described an algorithm for efficiently triangulating complex 3D domains immersed in nonconforming background tetrahedral meshes. We perceive it not as yet another meshing algorithm but as a meshing algorithm devised specifically for moving boundary problems. We envision it becoming a useful tool in challenging applications including fluid-structure interactions, phase transformations, fracture mechanics and shape/topology optimization. For this, however, a diverse set of challenges lie ahead. Establishing the algorithm's robustness a priori is uncompromisable, since there is little scope for interactive mesh improvement in simulations where the geometry and consequently the mesh is updated at each time step or solution iteration. A rigorous analysis of the algorithm will also help identify ways to relax the assumptions on the regularity of domains and on restrictions on angles in background meshes, as well as provide estimates for the refinement required of the background mesh to help make the algorithm fully automatic. A parallel implementation of the algorithm will also help expedite its integration into existing simulation codes. We refer to the work of Rangarajan and Lew²² for an outlook on the directional vertex relaxation algorithm.

ORCID

Ramsharan Rangarajan  <http://orcid.org/0000-0001-7403-7728>

REFERENCES

1. Radovitzky R, Ortiz M. Lagrangian finite element analysis of Newtonian fluid flows. *Int J Numer Methods Eng*. 1998;43(4):607-619.
2. Allaire G, Dapogny C, Frey P. Shape optimization with a level set based mesh evolution method. *Comput Methods Appl Mech Eng*. 2014;282:22-53.
3. Belytschko T, Krongauz Y, Organ D, Fleming M, Krysl P. Meshless methods: an overview and recent developments. *Comput Methods Appl Mech Eng*. 1996;139(1-4):3-47.
4. Borden MJ, Verhoosel CV, Scott MA, Hughes TJR, Landis CM. Phase-field description of dynamic brittle fracture. *Comput Methods Appl Mech Eng*. 2012;217-220:77-95.
5. Höllig K. *Finite Element Methods With B-Splines*. Philadelphia, PA: Society for Industrial and Applied Mathematics; 2003.
6. Lew AJ, Buscaglia GC. A discontinuous-Galerkin-based immersed boundary method. *Int J Methods Eng*. 2008;76(4):427-454.
7. Peskin CS. The immersed boundary method. *Acta Numer*. 2002;11:479-517.
8. Burman E, Hansbo P. Fictitious domain finite element methods using cut elements: I. A stabilized Lagrange multiplier method. *Comput Methods Appl Mech Eng*. 2010;199(41-44):2680-2686.
9. Sukumar N, Dolbow JE, Moës N. Extended finite element method in computational fracture mechanics: a retrospective examination. *Int J Fract*. 2015;196(1-2):189-206.
10. Codina R, Houzeaux G, Coppola-Owen H, Baiges J. The fixed-mesh ALE approach for the numerical approximation of flows in moving domains. *J Comput Phys*. 2009;228(5):1591-1611.
11. Kovalev K. *Unstructured Hexahedral Non-conformal Mesh Generation* [PhD thesis]. Brussel, Belgium: Vrije Universiteit Brussel; 2005.
12. Yang FL, Chen CH, Young DL. Novel mesh regeneration algorithm for 2D FEM simulations of flows with moving boundary. *J Comput Phys*. 2011;230(9):3276-3301.
13. Freitag LA, Plassmann P. Local optimization-based simplicial mesh untangling and improvement. *Int J Numer Methods Eng*. 2000;49(1-2):109-125.
14. Shontz SM, Vavasis SA. Analysis of and workarounds for element reversal for a finite element-based algorithm for warping triangular and tetrahedral meshes. *BIT Numer Math*. 2010;50(4):863-884.
15. Freitag LA. On combining Laplacian and optimization-based mesh smoothing techniques. ASME applied mechanics division-publications-amd; 1997:37-44.
16. Blom FJ. Considerations on the spring analogy. *Int J Numer Methods Fluids*. 2000;32(6):647-668.
17. De Almeida VF. Domain deformation mapping: application to variational mesh generation. *SIAM J Sci Comput*. 1999;20(4):1252-1275.

18. Farhat C, Degand C, Koobus B, Lesoinne M. Torsional springs for two-dimensional dynamic unstructured fluid meshes. *Comput Methods Appl Mech Eng*. 1998;163(1-4):231-245.
19. Budd CJ, Huang W, Russell RD. Adaptivity with moving grids. *Acta Numer*. 2009;18:111-241.
20. Frey PJ, Alauzet F. Anisotropic mesh adaptation for CFD computations. *Comput Methods Appl Mech Eng*. 2005;194(48-49):5068-5082.
21. Knupp PM. Achieving finite element mesh quality via optimization of the Jacobian matrix norm and associated quantities. Part II: a framework for volume mesh optimization and the condition number of the Jacobian matrix. *Int J Numer Methods Eng*. 2000;48(8):1165-1185.
22. Rangarajan R, Lew AJ. Provably robust directional vertex relaxation for geometric mesh optimization. *SIAM J Sci Comput*. 2017;39(6):A2438-A2471.
23. Chazal F, Lieutier A, Rossignac J, Whited B. Ball-map: homeomorphism between compatible surfaces. *Int J Comput Geom Appl*. 2010;20(03):285-306.
24. Rangarajan R, Lew AJ. Analysis of a method to parameterize planar curves immersed in triangulations. *SIAM J Numer Anal*. 2013;51(3):1392-1420.
25. Rangarajan R, Lew AJ. Universal meshes: a method for triangulating planar curved domains immersed in nonconforming meshes. *Int J Numer Methods Eng*. 2014;98(4):236-264.
26. Kabaria H, Lew AJ. Universal meshes for smooth surfaces with no boundary in three dimensions. *Int J Numer Methods Eng*. 2017;110(2):133-162.
27. Börgers C. Triangulation algorithm for fast elliptic solvers based on domain imbedding. *SIAM J Numer Anal*. 1990;27(5):1187-1196.
28. Labelle F, Shewchuk JR. Isosurface stuffing: fast tetrahedral meshes with good dihedral angles. *ACM Trans Graph*. 2007;26(3).
29. Doran C. *Isosurface Stuffing Improved: Acute Lattices and Feature Matching* [PhD thesis]. Vancouver, Canada: The University Of British Columbia; 2013.
30. Bagley B, Sastry SP, Whitaker RT. A marching-tetrahedra algorithm for feature-preserving meshing of piecewise-smooth implicit surfaces. *Procedia Eng*. 2016;163:162-174.
31. Baker TJ. Mesh movement and metamorphosis. *Eng Comput*. 2002;18(3):188-198.
32. Staten ML, Owen SJ, Shontz SM, Salinger AG, Coffey TS. A comparison of mesh morphing methods for 3D shape optimization. In: *Proceedings of the 20th International Meshing Roundtable*. Berlin, Germany: Springer-Verlag GmbH Berlin Heidelberg; 2011:293-311.
33. Si H. TetGen, a Delaunay-based quality tetrahedral mesh generator. *ACM Trans Math Softw*. 2015;41(2):11.
34. Geuzaine C, Remacle J-F. 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.
35. The CGAL Project. CGAL User and Reference Manual. CGAL Editorial Board, 4.8 edition. 2016.
36. Altair Engineering Inc. HyperWorks help manual and HyperMesh documentation. 2017. <https://altairhyperworks.com>
37. Alliez P, Cohen-Steiner D, Yvinec M, Desbrun M. Variational tetrahedral meshing. *ACM Trans Graph*. 2005;24(3):617-625.
38. Shewchuk JR. Tetrahedral mesh generation by Delaunay refinement. In: *Proceedings of the fourteenth annual symposium on Computational geometry*; 1998; Minneapolis, MN.
39. Cheng S-W, Dey TK, Levine JA. A practical Delaunay meshing algorithm for a large class of domains. In: *Proceedings of the 16th International Meshing Roundtable*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2008:477-494.
40. Du Q, Wang D. Tetrahedral mesh generation and optimization based on centroidal Voronoi tessellations. *Int J Numer Methods Eng*. 2003;56(9):1355-1373.
41. Oudot S, Rineau L, Yvinec M. Meshing volumes bounded by smooth surfaces. In: *Proceedings of the 14th International Meshing Roundtable*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2005:203-219.
42. Shewchuk JR. Unstructured mesh generation. In: *Combinatorial Scientific Computing*. Boca Raton, FL: CRC Press; 2012:257.
43. Shephard MS. Approaches to the automatic generation and control of finite element meshes. *Appl Mech Rev*. 1988;41(4):169-185.
44. Shephard MS, Georges MK. Automatic three-dimensional mesh generation by the finite octree technique. *Int J Numer Methods Eng*. 1991;32(4):709-749.
45. Yerry MA, Shephard MS. Automatic three-dimensional mesh generation by the modified-octree technique. *Int J Numer Methods Eng*. 1984;20(11):1965-1990.
46. Mitchell SA, Vavasis SA. Quality mesh generation in higher dimensions. *SIAM J Comput*. 2000;29(4):1334-1370.
47. Mitchell SA, Vavasis SA. Quality mesh generation in three dimensions. In: *Proceedings of the Eighth Annual Symposium on Computational Geometry*; 1992; Berlin, Germany.
48. Barnhill RE, Birkhoff G, Gordon WJ. Smooth interpolation in triangles. *J Approx Theory*. 1973;8(2):114-128.
49. Gordon WJ, Hall CA. Transfinite element methods: blending-function interpolation over arbitrary curved element domains. *Numer Math*. 1973;21(2):109-129.
50. Mansfield L. Interpolation to boundary data in tetrahedra with applications to compatible finite elements. *J Math Anal Appl*. 1976;56(1):137-164.
51. Lenoir M. Optimal isoparametric finite elements and error estimates for domains involving curved boundaries. *SIAM J Numer Anal*. 1986;23(3):562-580.
52. Ern A, Guermond J-L. *Theory and Practice of Finite Elements*. New York, NY: Springer Science+Business Media; 2013.
53. Henry D. *Perturbation of the Boundary in Boundary-Value Problems of Partial Differential Equations*. Cambridge, UK: Cambridge University Press; 2005.

54. Gilbarg D, Trudinger NS. *Elliptic Partial Differential Equations of Second Order*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2015.
55. Liu A, Joe B. On the shape of tetrahedra from bisection. *Math Comp*. 1994;63(207):141-154.
56. Rangarajan R. On the resolution of certain discrete univariate max-min problems. *Comput Optim Appl*. 2017;68(1):163-192.
57. Calakli F, Taubin G. SSD: smooth signed distance surface reconstruction. *Comput Graph Forum*. 2011;30(7):1993-2002.
58. Eppstein D, Sullivan JM, Üngör A. Tiling space and slabs with acute tetrahedra. *Comput Geom*. 2004;27(3):237-255.
59. Sullivan JM. New tetrahedrally close-packed structures. In: Proceedings of the Eurofoam; 2000; Delft, The Netherland.
60. Schöberl J. NETGEN an advancing front 2D/3D-mesh generator based on abstract rules. *Comput Vis Sci*. 1997;1(1):41-52.
61. Crane K. <https://www.cs.cmu.edu/~kmc Crane/Projects/ModelRepository>. Accessed January 06, 2018.
62. TetGen, <http://wias-berlin.de/software/tetgen/examples.spine.html>. Accessed January 06, 2018.
63. CGAL, https://github.com/CGAL/cgal/blob/master/Poisson_surface_reconstruction_3/examples/Poisson_surface_reconstruction_3/data/kitten.xyz. Accessed January 06, 2018.
64. VisionAir Project. Aim@Shape: Digital Shape WorkBench v5.0. http://visionair.ge.imati.cnr.it:8080/ontologies/shapes/view.jsp?id=1464-Human_heart. Accessed January 06, 2018.
65. VanderZee E, Hirani AN, Zharnitsky V, Guoy D. A dihedral acute triangulation of the cube. *Comput Geom*. 2010;43(5):445-452.
66. Arroyo M, Ortiz M. Local maximum-entropy approximation schemes: a seamless bridge between finite elements and meshfree methods. *Int J Numer Methods Eng*. 2006;65(13):2167-2202.
67. Galassi M, Davies J, Theiler J, et al. *GNU Scientific Library Reference Manual*. 3rd ed. London, UK: Network Theory Ltd; 2009.
68. Carr JC, Beatson RK, Cherrie JB, et al. Reconstruction and representation of 3D objects with radial basis functions. In: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques; 2001; Los Angeles, CA.
69. VisionAir Project. Aim@Shape: Digital Shape WorkBench v5.0. http://visionair.ge.imati.cnr.it:8080/ontologies/shapes/view.jsp?id=2714-_255_wrl. Accessed January 06, 2018.
70. Alberti L, Comte G, Mourrain B. Meshing implicit algebraic surfaces: the smooth case. *Math Methods Curves Surf Tromso*. 2004;4:11-26.
71. Plantinga S, Vegter G. Isotopic meshing of implicit surfaces. *Vis Comput*. 2007;23(1):45-58.
72. Arroyo M, Heltai L, Millán D, DeSimone A. Reverse engineering the euglenoid movement. *Proc Natl Acad Sci*. 2012;109(44):17874-17879.
73. Brezzi F, Pitkäranta J. On the stabilization of finite element approximations of the Stokes equations. In: *Efficient Solutions of Elliptic Systems: Proceedings of a GAMM-Seminar Kiel, January 27 to 29, 1984*. Wiesbaden, Germany: Springer Fachmedien Wiesbaden; 1984:11-19.
74. Brezzi F, Douglas Jr J. Stabilized mixed methods for the Stokes problem. *Numer Math*. 1988;53(1-2):225-235.
75. Brezzi F, Fortin M. *Mixed and Hybrid Finite Element Methods*. New York, NY: Springer Science+Business Media; 2012.

How to cite this article: Rangarajan R, Kabaria H, Lew A. An algorithm for triangulating smooth three-dimensional domains immersed in universal meshes. *Int J Numer Methods Eng*. 2018;1-34. <https://doi.org/10.1002/nme.5949>