INTERNATIONAL JOURNAL OF
NUMERICAL ANALYSIS AND MODELING
Volume 19, Number 6, Pages 887–906

EFFICIENT GENERATION OF MEMBRANE AND SOLVENT TETRAHEDRAL MESHES FOR ION CHANNEL FINITE ELEMENT CALCULATION

ZHEN CHAO, SHENG GUI, BENZHUO LU, DEXUAN XIE

Abstract. A finite element solution of an ion channel dielectric continuum model such as Poisson-Boltzmann equation (PBE) and a system of Poisson-Nernst-Planck equations (PNP) requires tetrahedral meshes for an ion channel protein region, a membrane region, and an ionic solvent region as well as an interface fitted irregular tetrahedral mesh of a simulation box domain. However, generating these meshes is very difficult and highly technical due to the related three regions having very complex geometrical shapes. Currently, an ion channel mesh generation software package developed in Lu's research group is the only one available in the public domain. To significantly improve its mesh quality and computer performance, in this paper, new numerical schemes for generating membrane and solvent meshes are presented and implemented in Python, resulting in a new ion channel mesh generation software package. Numerical results are then reported to demonstrate the efficiency of the new numerical schemes and the quality of meshes generated by the new package for ion channel proteins with ion channel pores having different geometric complexities.

Key words. Finite element method, Poisson-Nernst-Planck equations, ion channel, membrane mesh generation, tetrahedral mesh.

1. Introduction

The Poisson-Boltzmann equation (PBE) [1, 2, 3, 4] and a system of Poisson-Nernst-Planck (PNP) equations [5, 6, 7] are two commonly-used dielectric continuum models for simulating an ion channel protein embedded in an membrane and immersed in an ionic solvent. While PBE is mainly used to calculate electrostatic solvation and binding free energies, PNP is an important tool for computing membrane potentials, ionic transport fluxes, conductances, and electric currents, etc. Both PBE and PNP have been solved approximately by typical numerical techniques such as finite difference, finite element, and boundary element methods. Among these techniques, finite element techniques can be more suitable to deal with the numerical difficulties caused by the complicated interface and boundary value conditions of PBE and PNP. Due to using unstructured tetrahedral meshes, they allow us to well retain the geometry shapes of protein, membrane, and solvent regions such that we can obtain a PBE/PNP numerical solution in a high degree of accuracy.

However, generating an irregular tetrahedral mesh for PBE/PNP finite element calculation can be very difficult and highly technical because a membrane region can cause a solvent region to have a very complicated geometrical shape, not mention that how to generate a membrane mesh remains a challenging research topic. In fact, because of the lack of membrane molecular structural data, generating a membrane mesh can become very difficult. To avoid this difficulty, the membrane

Received by the editors January 15, 2022 and, in revised form, April 21, 2022; accepted April 26, 2022.
2000 *Mathematics Subject Classification.* 65M50, 65M60, 92-08, 68N01.

can be simply treated as a piece of rectangular slab to separate a simulation box domain into the inner and outer solution regions. An ion channel protein region is then embedded in the slab to let ions flow across the membrane through the ion channel pore. Even so, the generation of a solvent mesh is still very difficult since a solvent region can have very complex interfaces with protein and membrane regions. To avoid this difficulty, a novel two-region approach is usually adopted to the development of an ion channel mesh generation scheme. That is, an ion channel simulation box is first divided into an ion channel protein region surrounded by an expanded solvent region without involving any membrane, where a mesh of this expanded solvent region is supposed to have been properly constructed such that it contains both membrane and solvent meshes that are to be constructed for an ion channel PBE/PNP finite element calculation; thus, the next work to be done is to develop a numerical scheme for extracting these two membrane and solvent meshes from the expanded solvent mesh. Based on this novel two-region approach, an ion channel mesh software package was developed in Lu's research group with more than seven-years efforts (2011 to 2018) [8, 9, 10, 11, 12, 13]. After four years, this package remains the unique one available in the public domain and applicable for PBE/PNP ion channel finite element calculation. For clarity, we will refer to it as ICMPv1 (i.e., Ion Channel Mesh Package version 1).

Clearly, the quality of membrane and solvent meshes extracting from an expanded solvent region strongly depends on the construction of a mesh extraction numerical scheme. In 2014 [9], cylinders (or spheres) were suggested to use in the separation of the membrane and pore regions but a separation process was mainly done manually. The first numerical extraction scheme was reported in 2015 [10], which significantly improved the usage and performance of ICMPv1. In this scheme, a walk-detect method was adapted to detect the inner surface of an ion channel pore numerically, making it possible for us to generate the solvent, membrane, and protein region meshes and an interface fitted mesh of a simulation box domain without involving any manual effort. However, the mesh quality and the performance of ICMPv1 rely on the selection of the walk step size, the number of searching layers, and the other mesh generation parameters. A proper selection of the values of these parameters turn out to be difficult and very time-consuming for an ion channel protein having an ion channel pore with a complicated geometrical shape.

Recently, ICMPv1 was adapted to the implementation of the new PBE/PNP ion channel finite element solvers developed in Xie's research group [6, 7, 14, 15]. During these applications, ICMPv1 was found to occasionally produce a membrane mesh that contains the tetrahedra belonging to a solvent mesh. It is possible to remove these false tetrahedra manually. For example, via a visualization tool (e.g., ParaView [16]), we may identify these false tetrahedra and then remove them by reconstructing a new mesh. We may also adjust the related parameters repeatedly until none of false tetrahedra occur in a membrane mesh, but doing so may be very time-consuming and may twist a protein, membrane, or solvent mesh due to using improper parameter values, causing the numerical accuracy of a PBE/PNP finite element solution to be reduced significantly. These cases motivated Xie's research group to develop more effective and more efficient numerical schemes than those used in ICMPv1. Eventually, the second version of ICMP, denoted by ICMPv2, has been developed by Xie's research group through a close collaboration with Lu's research group. The purpose of this paper is to present the new schemes implemented in ICMPv2 and report numerical results to demonstrate that ICMPv2 can generate membrane and solvent meshes much more efficiently and in much higher quality than ICMPv1, especially for an ion channel protein having an ion channel pore with a complicated geometric shape even for an ion channel protein with multiple ion channel pores — a case in which ICMPv1 would not work.

The rest of the paper is organized as follows. Section 2 introduces a general framework for ion channel mesh generations. Section 3 presents a new scheme for generating a triangle surface mesh of the box domain boundary. Section 4 presents a new scheme for selecting mesh points to be set as the vertices of an interface triangular mesh between the membrane and solvent meshes. Section 5 presents a new numerical scheme for extracting the membrane and solvent tetrahedral meshes from an expanded solvent tetrahedral mesh. Section 6 reports numerical results to demonstrate that ICMPv2 can generate membrane and solvent meshes in higher quality and better computer performance than ICMPv1. Conclusions are made in Section 7.

2. A framework for ion channel mesh generations

As required to implement a continuum dielectric model such as PBE or PNP for an ion channel system consisting of an ion channel protein, a membrane, and an ionic solvent, a sufficiently large simulation box, Ω, is selected to satisfy the domain partition

$$\Omega = D_p \cup D_m \cup D_s, \tag{1}$$

where D_p, D_m, and D_s denote a protein region, a membrane region, and a solvent region, respectively. Specifically, we construct Ω by

$$\Omega = \{(x,y,z) | L_{x_1} < x < L_{x_2}, L_{y_1} < y < L_{y_2}, L_{z_1} < z < L_{z_2}\}, \tag{2}$$

where L_{x_1}, L_{x_2}, L_{y_1}, L_{y_2}, L_{z_1}, and L_{z_2} are real numbers. We then assume that the center of an ion channel pore is at the origin of the rectangular coordinator system and the location of D_m is determined by two parameters $Z1$ and $Z2$ between L_{z_1} and L_{z_2}. An illustration of our box domain construction is given in Figure 1.

However, in practice, the subdomains D_p, D_m, and D_s are approximated by their tetrahedral meshes $D_{p,h}$, $D_{m,h}$, and $D_{s,h}$ since it is too difficult to obtain them directly due to their complex geometric shapes. As soon as $D_{p,h}$, $D_{m,h}$, and $D_{s,h}$ are obtained, an interface fitted tetrahedral mesh, Ω_h, of Ω can be constructed according to the following mesh domain partition

$$\Omega_h = D_{p,h} \cup D_{m,h} \cup D_{s,h}. \tag{3}$$

One key step to obtain $D_{p,h}$, $D_{m,h}$, and $D_{s,h}$ is to construct their triangular surface meshes $\partial D_{p,h}$, $\partial D_{m,h}$, and $\partial D_{s,h}$ since from these three triangular surface meshes the tetrahedral volume meshes $D_{p,h}$, $D_{m,h}$, and $D_{s,h}$ can be generated routinely by using a volume mesh generation software package such as TetGen [17, 18].

It has been known that $\partial D_{p,h}$ can be generated from one of the current software packages TMSmesh [8], NanoShaper [19], GAMer [20], MSMS [21], and MolSurf [22] when a molecular structure of an ion channel is known. However, how to generate $\partial D_{m,h}$ remains a difficult research topic. In fact, a membrane consists of a double layer of phospholipid, cholesterol, and glycolipid molecules, making it very difficult to derive a boundary mesh of $\partial D_{m,h}$.

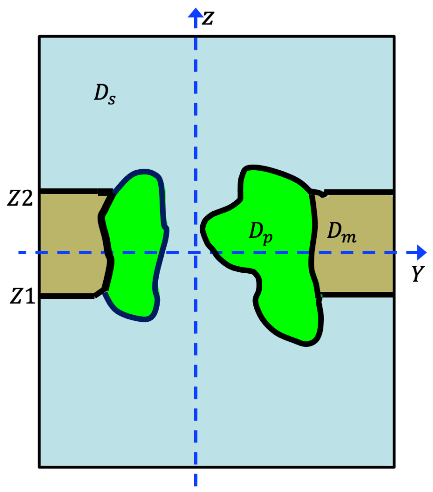


FIGURE 1. An illustration of box domain partition (1).

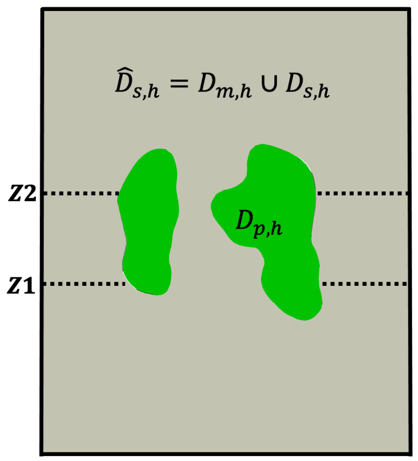


FIGURE 2. An illustration of the mesh partitions of (4).

To avoid these difficulties, we follow the strategy used in [9, 10] to construct an expanded solvent tetrahedral mesh, $\hat{D}_{s,h}$, satisfying

$$\Omega_h = D_{p,h} \cup \hat{D}_{s,h}, \quad \hat{D}_{s,h} = D_{m,h} \cup D_{s,h}. \tag{4}$$

An illustration of the above partition is given in Figure 2, where the dotted lines represent a set of the prior mesh vertices selected from the membrane bottom surface at $z = Z1$ and top surface at $z = Z2$. Such a set will be selected by a numerical scheme to be presented in Section 4.

Clearly, the boundary surface mesh $\partial\hat{D}_{s,h}$ of $\hat{D}_{s,h}$ can be constructed by

$$\partial\hat{D}_{s,h} = \partial D_{p,h} \cup \partial\Omega_h,$$

where $\partial D_{p,h}$ is a given ion channel protein triangular surface mesh and $\partial\Omega_h$ is a triangular surface mesh of the boundary $\partial\Omega$ of the box domain Ω, whose construction will be done by a numerical scheme to be presented in Section 3. We then can use current mesh software packages (e.g., TetGen) to generate the tetrahedral meshes $D_{p,h}$ and $\hat{D}_{s,h}$. As soon as $\hat{D}_{s,h}$ is known, we need a numerical scheme to extract $D_{m,h}$ and $D_{s,h}$ from $\hat{D}_{s,h}$. Such a scheme will be presented in Section 5.

3. Construction of a box triangular surface mesh

In this section, we present a numerical scheme for constructing a triangular surface mesh, $\partial\Omega_h$, to ensure that the mesh domain partition (3) holds. In this scheme, a triangular surface mesh of $\partial D_{p,h}$ is supposed to be given. Thus, we can find the smallest rectangular box $[a_1, b_1] \times [a_2, b_2] \times [a_3, b_3]$ that holds $\partial D_{p,h}$ using the formulas:

$$\begin{aligned} a_1 &= \min_{1\le i\le N} x_i, & b_1 &= \max_{1\le i\le N} x_i, \\ a_2 &= \min_{1\le i\le N} y_i, & b_2 &= \max_{1\le i\le N} y_i, \\ a_3 &= \min_{1\le i\le N} z_i, & b_3 &= \max_{1\le i\le N} z_i, \end{aligned} \tag{5}$$

where (x_i, y_i, z_i) denotes the position vector of the i-th mesh point of $\partial D_{p,h}$ and N is the total number of mesh points of $\partial D_{p,h}$. We then can construct a box domain, Ω, of (2) in terms of three parameters, η_i for $i = 1, 2, 3$, according to the following formulas:

$$\begin{aligned} L_{x_1} &= a_1 - \eta_1, & L_{x_2} &= b_1 + \eta_1, \\ L_{y_1} &= a_2 - \eta_2, & L_{y_z} &= b_2 + \eta_2, \\ L_{z_1} &= a_3 - \eta_3, & L_{z_2} &= b_3 + \eta_3. \end{aligned} \tag{6}$$

The default values of η_1, η_2, and η_3 are set as 20 but can be adjusted by users as needed. In this way, a selection of a box domain satisfying the partition (1) is greatly simplified.

We next split the six boundary surfaces of $\partial\Omega$ by

$$\partial\Omega = \Gamma_D \cup \Gamma_N, \tag{7}$$

where Γ_D consists of the bottom and top surfaces and Γ_N consists of the four side surfaces of $\partial\Omega$. We further divide a solvent mesh, $D_{s,h}$, into three portions, $D^b_{s,h}, D^t_{s,h}$, and $D^p_{s,h}$, by

$$D_{s,h} = D^b_{s,h} \cup D^t_{s,h} \cup D^p_{s,h}, \tag{8}$$

where $D^b_{s,h}$, $D^t_{s,h}$, and $D^p_{s,h}$ are defined by

$$D^b_{s,h} = \{\mathbf{r} \in D_{s,h} \mid \mathbf{r} = (x, y, z) \text{ with } z < Z1\},$$

$$D^t_{s,h} = \{\mathbf{r} \in D_{s,h} \mid \mathbf{r} = (x, y, z) \text{ with } z > Z2\},$$

and

$$D^p_{s,h} = \{\mathbf{r} \in D_{s,h} \mid \mathbf{r} = (x, y, z) \text{ with } Z1 \le z \le Z2\}.$$

An illustration of partition (8) is given in Figure 3.

Since a uniform triangular mesh of Γ_D can be constructed easily, we only describe the construction of a triangular surface mesh, $\Gamma_{N,h}$, of Γ_N. According to the solvent mesh partition (8), we can split $\Gamma_{N,h}$ into three sub-meshes, $\Gamma^{s,b}_{N,h}$, $\Gamma^m_{N,h}$, and $\Gamma^{s,t}_{N,h}$, such that

$$\Gamma_{N,h} = \Gamma^{s,b}_{N,h} \cup \Gamma^m_{N,h} \cup \Gamma^{s,t}_{N,h}, \tag{9}$$

where $\Gamma^{s,b}_{N,h}$, $\Gamma^m_{N,h}$, and $\Gamma^{s,t}_{N,h}$ are defined by

$$\Gamma^{s,b}_{N,h} = \Gamma_N \cap D^b_{s,h},\ \Gamma^{s,t}_{N,h} = \Gamma_N \cap D^t_{s,h},\ \Gamma^m_{N,h} = \Gamma_N \cap D_{m,h}.$$

In ICMPv2, these three sub-meshes are constructed as uniform triangular meshes, respectively, and are allowed to have different mesh sizes. In particular, we set the mesh size h_m of $\Gamma^m_{N,h}$ as an input parameter of ICMPv2. We then set $\Gamma^{s,b}_{N,h}$ and $\Gamma^{s,t}_{N,h}$ to have the same mesh size h_s. By default, we set $h_s = 2h_m$. In this case, $\Gamma^{s,b}_{N,h}$ and $\Gamma^{s,t}_{N,h}$ can be split as follows:

$$\Gamma^{s,b}_{N,h} = \Gamma^{s,1}_{N,h} \cup \Gamma^{s,2}_{N,h}, \quad \Gamma^{s,t}_{N,h} = \Gamma^{s,3}_{N,h} \cup \Gamma^{s,4}_{N,h}, \tag{10}$$

where $\Gamma^{s,2}_{N,h}$ and $\Gamma^{s,3}_{N,h}$ are defined by one mesh layer of $\Gamma^{s,b}_{N,h}$ and $\Gamma^{s,t}_{N,h}$, respectively. An illustration of surface mesh partitions (9) and (10) is given in Figure 4, where $\Gamma^{s,2}_{N,h}$ and $\Gamma^{s,3}_{N,h}$ are colored in pink, $\Gamma^{s,1}_{N,h}$ and $\Gamma^{s,4}_{N,h}$ in grey, and $\Gamma^m_{N,h}$ in green.

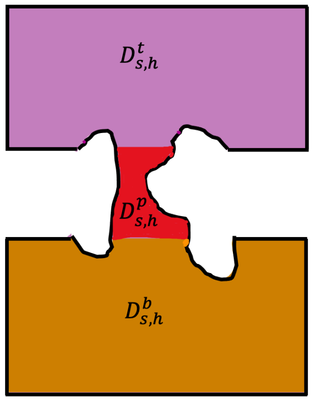


FIGURE 3. An illustration of the partition (8) of a solvent mesh $D_{s,h}$.

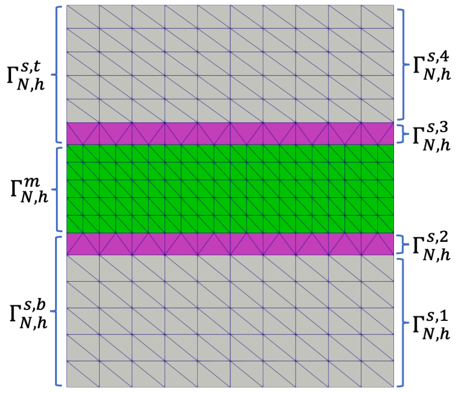
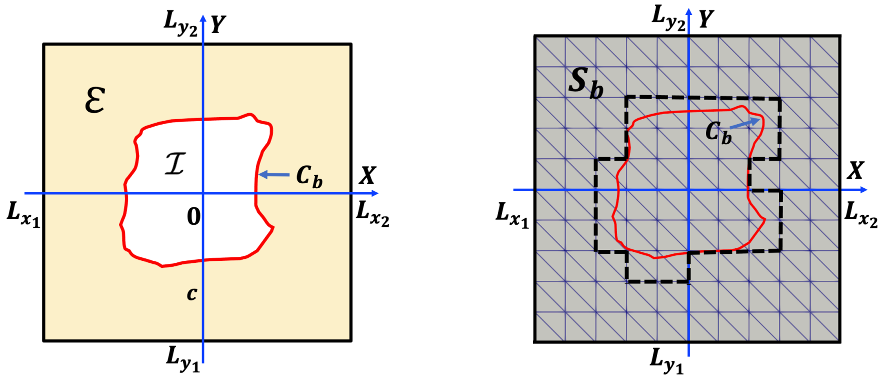


FIGURE 4. An example of box surface mesh partitions (9) and (10).



(A) A partition of rectangle $[L_{x_1}, L_{x_2}] \times [L_{y_1}, L_{y_2}]$ into an exterior region, $\mathcal{E}$, and an interior region, $\mathcal{I}$

(B) S_b consists of the mesh points outside the dash line plus the mesh points on the dash line.

FIGURE 5. An illustration of a partition of rectangle $[L_{x_1}, L_{x_2}] \times [L_{y_1}, L_{y_2}]$ by a cross section curve, denoted by $\mathcal{C}_b$, of $\partial D_{p,h}$ with the plane $z = Z1$ and a mesh point set S_b defined in (11).

4. A numerical scheme for selecting membrane surface mesh points

In this section, we present a numerical scheme for selecting a membrane surface mesh point set, $\mathcal{S}$, as needed in the construction of an expanded solvent tetrahedron mesh, $\hat{D}_{s,h}$, satisfying (4). In ICMPv2, the mesh $\hat{D}_{s,h}$ is generated by TetGen. By using one option, `-i`, of TetGen, the mesh points of $\mathcal{S}$ can be added to the TetGen input node file. To this end, the mesh points of $\mathcal{S}$ become the additional vertices of $\hat{D}_{s,h}$. On the other hand, they will be set as the vertices of a membrane mesh $D_{m,h}$. Hence, they are very helpful for us to extract $D_{m,h}$ from $\hat{D}_{s,h}$.

For clarity, we only describe a mesh point selection from the bottom membrane surface Γ^b_m since a selection from the top membrane surface can be done similarly.

With a given mesh size, h_m, of a a membrane boundary mesh $\partial D_{m,h}$, we construct a uniform rectangular mesh of the rectangle $[L_{x_1}, L_{x_2}] \times [L_{y_1}, L_{y_2}]$ and define a set, $\mathcal{T}$, of its mesh points by

$$\mathcal{T} = \{(x_i, y_j) \mid x_i = L_{x_1} + ih_m, y_j = L_{y_1} + jh_m \text{ for } i = 0, 1, \dots, m, j = 0, 1, \dots, n\},$$

where $m = (L_{x_2} - L_{x_1})/h_m$ and $n = (L_{y_2} - L_{y_1})/h_m$. We also obtain a curve, $\mathcal{C}_b$, of the cross section of $\partial D_{p,h}$ with the plane $z = Z1$. Clearly, this curve splits the rectangle $[L_{x_1}, L_{x_2}] \times [L_{y_1}, L_{y_2}]$ into an exterior region, denoted by $\mathcal{E}$, and an interior region, denoted by $\mathcal{I}$, as illustrated in Figure 5A. We then can obtain a set, S_b, of mesh points from the bottom membrane surface by

$$S_b = \{(x_i, y_i, Z1) \mid (x_i, y_i) \in \mathcal{T} \cap \mathcal{E}\} \tag{11}$$

as illustrated in Figure 5B.

From Figure 5B it can also be seen that reducing the mesh size h_m can improve the approximation of the dash line to the cross section curve $\mathcal{C}_b$, enabling us to extract a higher quality membrane mesh from the expanded solvent mesh $\hat{D}_{s,h}$. Due to this reason, we use $h_m = h_s/2$ (by default) in the selection scheme.

Similarly, we can obtain a set, S_t, of mesh points from the top membrane surface at $z = Z2$. Consequently, the membrane surface mesh point set $\mathcal{S}$ is derived by

$$\mathcal{S} = S_b \cup S_t.$$

The mesh points of $\mathcal{S}$ will be set as the prior mesh points for the generation of $\hat{D}_{s,h}$.

5. Extraction of membrane and solvent meshes

In this section, we present a numerical scheme for extracting membrane and solvent meshes, $D_{m,h}$ and $D_{s,h}$, from an expanded solvent mesh, $\hat{D}_{s,h}$. In this scheme, we assume that the tetrahedra of $D_{p,h}$ and $\hat{D}_{s,h}$ have label numbers 1 and 2, respectively, while the tetrahedra of $D_{p,h}$, $D_{s,h}$, and $D_{m,h}$ have label numbers 1, 2, and 3, respectively.

For clarity, we describe our new numerical scheme in six steps as follows:

Step 1: Construct a rectangle, $[a, b] \times [c, d]$, that contains a portion, $\mathcal{P}$, of triangular surface mesh $\partial D_{p,h}$ intercepted by the two planes $Z = Z1$ and $Z = Z2$ by

$$\begin{aligned} a &= \min\{x_i, \mid (x_i, y_i, z_i) \text{ on } \mathcal{P}\} - \tau, \\ b &= \max\{x_i, \mid (x_i, y_i, z_i) \text{ on } \mathcal{P}\} + \tau, \\ c &= \min\{y_i, \mid (x_i, y_i, z_i) \text{ on } \mathcal{P}\} - \tau, \\ d &= \max\{y_i, \mid (x_i, y_i, z_i) \text{ on } \mathcal{P}\} + \tau, \end{aligned}$$

where (x_i, y_i, z_i) denotes the ith vertex of $\mathcal{P}$ and τ is a positive parameter to ensure that the rectangle $[a, b] \times [c, d]$ is not to touch any part of $\mathcal{P}$. By default, we set $\tau = h_m$.

Step 2: Construct three submeshes, $\mathcal{B}_h$, $D_{ms,h}$, of $\hat{D}_{s,h}$, and $D^o_{m,h}$ by

$$\begin{aligned} \mathcal{B}_h &= \hat{D}_{s,h} \cap [L_{x_1}, L_{x_2}] \times [L_{y_1}, L_{y_2}] \times [Z1, Z2], \\ D_{ms,h} &= \hat{D}_{s,h} \cap [a, b] \times [c, d] \times [Z1, Z2], \end{aligned} \tag{12}$$

and

$$D^o_{m,h} = \mathcal{B}_h - D_{ms,h}. \tag{13}$$

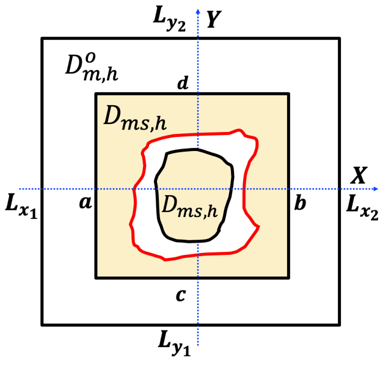


FIGURE 6. A partition (13).

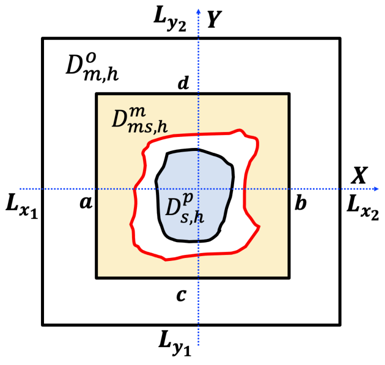


FIGURE 7. A partition (14).

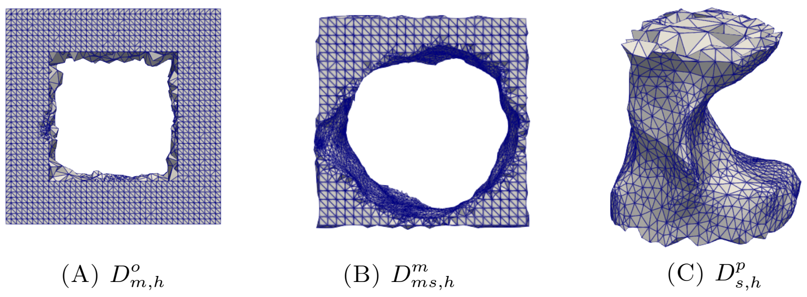


FIGURE 8. Submeshes $D^o_{m,h}$, $D^m_{ms,h}$, and $D^p_{s,h}$ defined in the partitions (13) and (14) for the case of a VDAC (PDB ID: 3EMN).

In other words, we have divided $\mathcal{B}_h$ into two submeshes satisfying

$$\mathcal{B}_h = D_{ms,h} \cup D^o_{m,h}$$

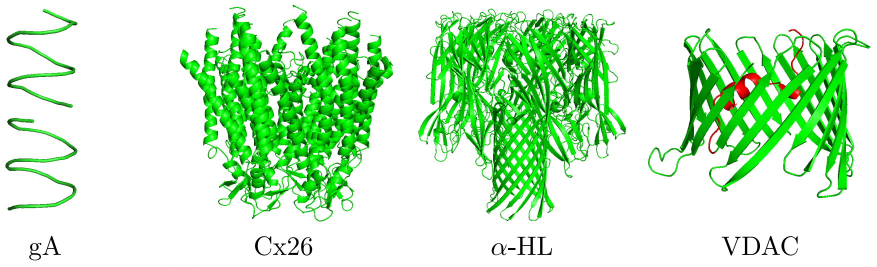
as illustrated in Figure 6. From this figure we can see that $D_{ms,h}$ consists of two non-overlapped parts — one part is nothing but the solvent portion $D^p_{s,h}$ of the solvent mesh partition (8) and the other part, denoted by $D^m_{ms,h}$, belongs to the membrane region D_m. Thus, $D_{ms,h}$ can be expressed as

$$D_{ms,h} = D^m_{ms,h} \cup D^p_{s,h}. \tag{14}$$

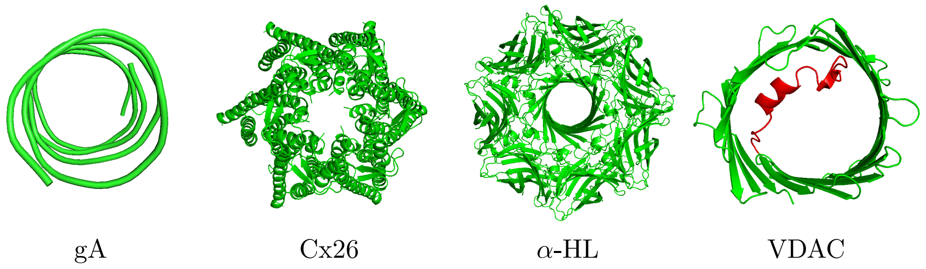
For the purpose of illustrating the partitions (13) and (14), we obtained the submeshes $D^o_{m,h}$, $D^m_{ms,h}$, and $D^p_{s,h}$ for an ion channel protein (VDAC) and displayed them in Figure 8.

Step 3: Separate the tetrahedra of $D_{ms,h}$ as two non-overlapped sets, one set leads to $D^m_{ms,h}$ and the other set to $D^p_{s,h}$, such that the partition (14) holds. In ICMPv2, this separation is done by a numerical scheme implemented in the Python function `split()` from the Python library *Trimesh*[1]. To do so, we need to obtain a boundary triangular mesh $\partial D_{ms,h}$ of $D_{ms,h}$ since it is a required input mesh of this Python function. We obtain $\partial D_{ms,h}$ through finding the boundary surface meshes of $D^m_{ms,h}$ and $D^p_{s,h}$, respectively.

[1] *https://github.com/mikedh/trimesh*



(A) Side view of four ion channel proteins



(B) Top view of four ion channel proteins

FIGURE 9. Molecular structures of the four ion channel proteins (gA, Cx26, α-HL, and VDAC) to be used for numerical tests in Section 6 in cartoon backbone representation — one common way to represent a three-dimensional protein secondary structure (e.g., α-helices in flat helical sheets and β-sheets in flat level sheets).

Step 4: Identify the tetrahedra of $D^p_{s,h}$ by doing ray tests via the ray-triangle intersection method (see [23] for example). To do so, we first calculate the centroids of all the tetrahedra of $D_{ms,h}$ and then use ray tests to check if they are inside the volume region enclosed by a boundary triangular surface mesh of $D^p_{s,h}$ or not. In the true case, we store the tetrahedron indices to an index set, $\mathcal{S}_s$; otherwise, the tetrahedron indices are stored to another index set, $\mathcal{S}_m$. We then change the label numbers of the tetrahedra with indices in $\mathcal{S}_m$ from 2 to 3 to obtain the first part of $D_{m,h}$, which is denoted by $D^1_{m,h}$. In ICMPv2, a ray test is done by calling the Python function `contains_points()` from a class object,

`trimesh.ray.ray_pyembree.RayMeshIntersector()`,

of the Python library *Trimesh*.

Step 5: Identify the tetrahedra of $D^o_{m,h}$ by using partition (13) and change their label numbers from 2 to 3 to obtain the second part of $D_{m,h}$, which is denoted by $D^2_{m,h}$.

Step 6: Obtain the membrane and solvent meshes $D_{m,h}$ and $D_{s,h}$ by

$$D_{m,h} = D^1_{m,h} \cup D^2_{m,h}, \quad D_{s,h} = \hat{D}_{s,h} - D_{m,h}.$$

Remark: *Another way to extract $D_{m,h}$ from $\hat{D}_{s,h}$ is to use $\mathcal{B}_h$ since $\mathcal{B}_h$ contains $D_{m,h}$ too. The reason why we use $D_{ms,h}$, instead of $\mathcal{B}_h$, is to further reduce the computational cost since $D_{ms,h}$ contains a much smaller number of tetrahedra than $\mathcal{B}_h$.*

6. Numerical results

We implemented the three new schemes of Sections 3, 4, and 5 in Python based on our recent mesh work done in [6, 7] and using some mesh functions from the FEniCS project[2] and Trimesh. We then used them to modify ICMPv1 as ICMPv2. Note that ICMPv2 retains the part of ICMPv1 in the generation of an ion channel protein molecular surface mesh $\partial D_{p,h}$ (i.e., doing so via the TMSmesh software packages developed in Lu's research group [8]). Hence, both ICMPv2 and ICMPv2 are expected to generate the same ion channel protein mesh $D_{p,h}$ and expanded solvent mesh $\hat{D}_{s,h}$ when they use the same TMSmesh and TetGen parameters and the same boundary surface mesh $\partial\Omega$. Their differences mainly occur in a process of extracting membrane and solvent meshes $D_{m,h}$ and $D_{s,h}$ from an expanded solvent mesh of $\hat{D}_{s,h}$. Therefore, in this section, we mainly report the numerical results related to this extraction process.

Table 1. Values of box domain dimensions and main mesh parameters used in our numerical tests. Here, d, c, and e are three TMSmesh parameters — d is the decay rate in the Gaussian surface, c is isovalue in the Gaussian surface, and e is an approximation precision between trilinear surface and Gaussian surface.

Ion channel protein	Dimensions of box domain Ω $[L_{x_1}, L_{x_2}; L_{y_1}, L_{y_2}; L_{z_1}, L_{z_2}]$	$Z1$	$Z2$	h_m	d	c	e
gA	$[-31, 31; -30, 29; -33, 33]$	-11	11	1.1	0.5	0.9	0.9
Cx26	$[-67, 67; -63, 63; -60, 62]$	-16	16	1.6	0.2	0.2	0.8
α-HL	$[-71, 71; -71, 68; -39, 104]$	-11	11	1.1	0.2	0.9	0.9
VDAC	$[-46, 53; -46, 43; -44, 40]$	-12	12	1.2	0.2	0.5	0.75

In particular, we did numerical tests on four ion channel proteins: (1) A gramicidin A (gA), (2) a Connexin 26 gap junction channel (Cx26), (3) a staphylococcal α-hemolysin (α-HL), and (4) a voltage-dependent anion channel (VDAC). Their crystallographic three-dimensional molecular structures can be downloaded from the Protein Data Bank[3] with the PDB identification numbers 1GRM, 2ZW3, 7AHL, and 3EMN, respectively. But, in this work, we downloaded them from the Orientations of Proteins in Membranes (OPM) database[4] since these molecular structures have satisfied our assumptions made in the partition (1). That is, the protein structure has been transformed such that the normal direction of the top membrane surface is in the z-axis direction and the membrane location numbers $Z1$ and $Z2$ are given. See Table 1 for their values.

Figure 9 displays these four ion channel proteins in cartoon views. The α-helix of VDAC has been colored in red to more clearly view it in Figure 9. From these plots we can see that the ion channel pores of these four proteins have different geometrical complexities. Thus, these four proteins are good for numerical tests on the efficiency of our new numerical schemes and a comparison study between ICMPv1 and ICMPv2.

[2] *https://fenicsproject.org*

[3] *https://www.rcsb.org*

[4] *https://opm.phar.umich.edu*

Table 1 lists the values of box domain dimensions, three parameters $Z1, Z2$, and h_m from our new schemes, and three parameters h, d, and c from the molecular surface software TMSmesh. In the numerical tests, we fixed the other parameter values of our schemes as follows:

$$\eta_1 = 20, \quad \eta_2 = 20, \quad \eta_3 = 20, \quad h_s = 2h_m, \quad \tau = h_m.$$

We also fixed the command line switches of TetGen as `'-q1.2aVpiT1e-10AAYYCnQ'`, whose definitions and usages can be found in TetGen's manual webpage[5]. Thus, we did not list them in Table 1. Actually, all the box domain dimensions of Table 1 can be produced from the formulas of (6). Even so, we have listed them in Table 1 for clarity.

TABLE 2. A comparison of mesh data generated by ICMPv1 with those by ICMPv2.

Channel protein	Number of vertices					
	$\hat{D}_{s,h}$	$D^p_{s,h}$	Ω_h	$D_{s,h}$	$D_{m,h}$	$D_{p,h}$
Mesh data generated by ICMPv1						
gA	17711	662	24155	9359	10653	11012
Cx26	74185	1091	156182	55869	24304	106354
α-HL	109911	1615	230657	101212	13582	160141
VDAC	32842	2323	52640	23877	12981	28863
Mesh data generated by ICMPv2						
gA	27075	690	33612	15254	15488	11105
Cx26	73984	1096	158396	55255	22645	108769
α-HL	116649	1545	237535	105027	16757	160281
VDAC	35681	2426	56584	25538	13440	29968

Channel protein	Number of tetrahedra					
	$\hat{D}_{s,h}$	$D^p_{s,h}$	Ω_h	$D_{s,h}$	$D_{m,h}$	$D_{p,h}$
Mesh data generated by ICMPv1						
gA	95145	2488	149558	44092	51053	54413
Cx26	387042	4221	979905	274291	112751	592863
α-HL	563949	6471	1448649	501402	62547	884700
VDAC	176335	9788	328992	116287	60048	152657
Mesh data generated by ICMPv2						
gA	143799	2579	198776	70787	73012	54977
Cx26	375132	4282	983421	270715	104417	608289
α-HL	597027	6403	1482295	519312	77715	885268
VDAC	183414	10334	343072	159658	60685	159658

We did all the numerical tests on a MacBook Pro computer with one 2.6 GHz Intel core i7 processor and 16 GB memory. We listed the mesh data generated from ICMPv1 and ICMPv2 in Table 2 and reported other test results in Table 3 and Figures 10 to 13.

[5]*https://wias-berlin.de/software/tetgen/1.5/doc/manual/manual.pdf*

Table 2 shows that the meshes generated by ICMPv2 have more mesh vertices and more tetrahedra than those by ICMPv1. Since both ICMPv1 and ICMPv2 used the same triangular ion channel surface meshes and the same TetGen mesh generation parameters, such mesh differences are caused mainly by their using different box domain surface meshes. In ICMPv2, the box domain surface mesh is constructed by the numerical scheme presented in Section 3. It can be denser than that by ICMPv1, resulting in a significant improvement on the qualities of membrane and solvent meshes $D_{m,n}$ and $D_{s,h}$ (see Figure 13 for examples).

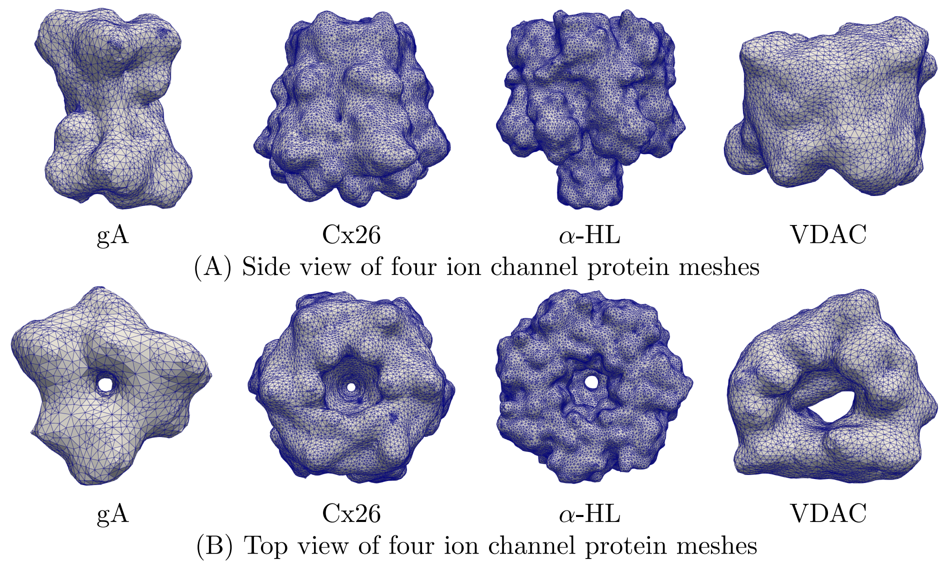
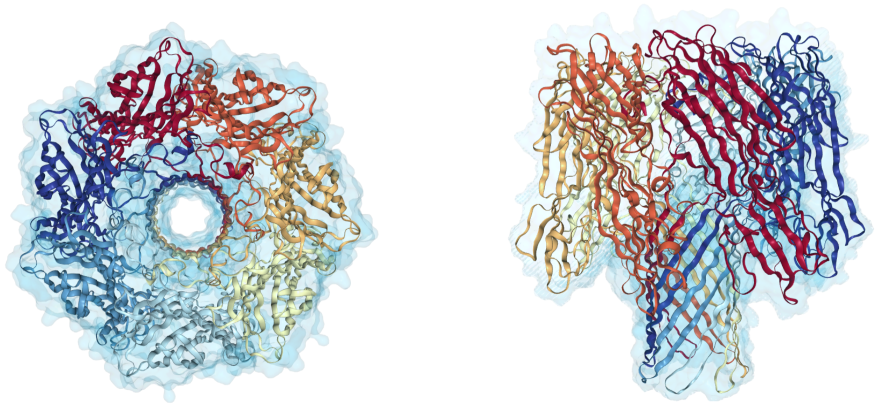

gA
Cx26
α-HL
VDAC
(A) Side view of four ion channel protein meshes
gA
Cx26
α-HL
VDAC
(B) Top view of four ion channel protein meshes

FIGURE 10. Top views of the four ion channel protein meshes $D_{p,h}$ generated by ICMPv2 according to the four molecular structures displayed in Figure 9.

Figure 10 displays the triangular surface meshes of the four ion channel proteins generated from ICMPv2 through using the software package TMSmesh. Note that VDAC has a much larger channel pore than the others.

Because α-HL has a complex molecular structure, we take it as an example to show how an ion channel protein mesh $D_{p,h}$ generated by ICMPv2 to fit a molecular structure. From Figure 11 it can be seen that the protein mesh $D_{p,h}$ can hold the molecular structure very well. We further display the box domain mesh Ω_h, solvent mesh $D_{s,h}$, and membrane mesh $D_{m,h}$ as well as a cross section view of solvent mesh $D_{s,h}$ in Figure 12. From this figure it can be seen that the unstructured tetrahedral meshes generated from ICMPv2 can catch well the main features of the complicated geometric shapes of the protein, membrane, and solvent regions D_p, D_m and D_s and the complex interfaces of the box domain mesh Ω_h.

Figure 13 presents a comparison of the membrane and solvent meshes $D_{m,h}$ and $D_{s,h}$ generated by ICMPv1 and ICMPv2 for VDAC. From it we can see that ICMPv2 can generate either $D_{m,h}$ or $D_{s,h}$ in higher quality than ICMPv1. We should point out that we spent a lot of time on the adjustment of mesh parameters to let ICMPv1 be able to extract $D_{m,h}$ and $D_{s,h}$ from the given expanded solvent mesh $\hat{D}_{s,h}$ successively in the sense that the membrane mesh $D_{m,h}$ does not contain



(A) A top view (B) A side view

FIGURE 11. A comparison of a protein mesh $D_{p,h}$ generated by ICMPv2 with a molecular structure of α-HL (PDB ID: 7AHL). Here the protein structure is depicted in cartoon backbone representation and colored in red, blue, yellow, and cyan.

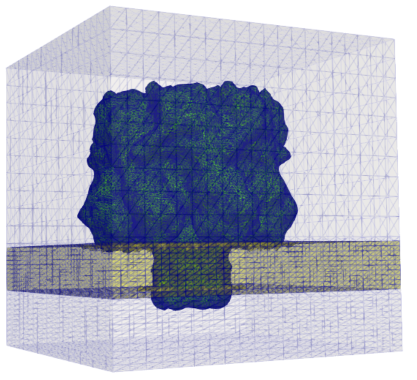
any tetrahedron from the solvent mesh $D_{s,h}$. Even so, the numerical schemes of ICMPv1 for constructing a box surface mesh and for selecting a set of mesh points from the bottom and top membrane surfaces still caused extraction problems, which decayed mesh quality.

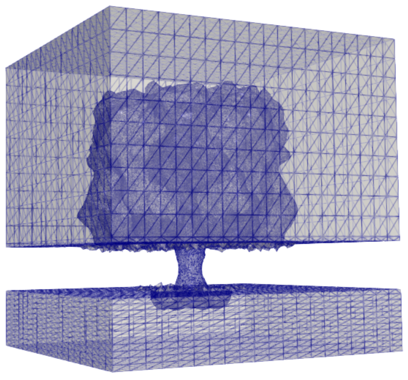
TABLE 3. A comparison of the computer performance of ICMPv1 with that of ICMPv2.

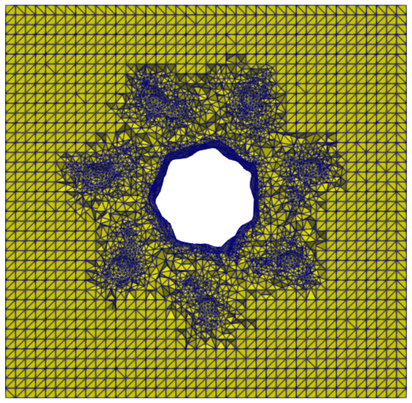
Mesh package	CPU time (in seconds)			
	gA	Cx26	α-HL	VDAC
ICMPv1	12.3	90.4	94.5	50.1
ICMPv2	1.1	2.2	2.5	1.5

Table 3 reports the performance of ICMPv1 and ICMPv2 in computer CPU time spent on the extraction of membrane and solvent meshes $D_{p,h}$ and $D_{s,h}$ from a given expanded solvent mesh $\hat{D}_{s,h}$. From Table 3 it can be seen that our new numerical algorithms reported in Sections 3, 4, and 5 can be much more efficient than the corresponding algorithms in ICMPv1 — about 11 times faster for gA and at least 30 times faster for others.

Finally, we did tests on an ion channel molecular structure with two ion channel pores to test the robustness of ICMPv2. We created this test case through rotating the α-helix of VDAC by 20° along the z axis at the hinge region *(Gly-21Tyr-22Gly-23Phe-24Gly-25)*. A view of the molecular structure of this modified VDAC, denoted by mVDAC, is displayed in Figure 14. With the values of parameters $Z1, Z2$, and h_m and the box domain Ω used in the case of VDAC (see Table 1), ICMPv2 produced both membrane and solvent meshes in about 4.1 seconds only, showing the efficiency of our new schemes and the robustness of ICMPv2. A view of the protein, membrane, and solvent meshes of mVDAC is displayed in Figure 14.


(A) A view of Ω_h


(B) Side view of $D_{s,h}$


(C) Top view of $D_{m,h}$ at $z = Z2$

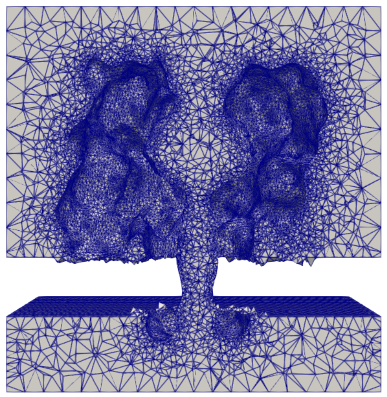
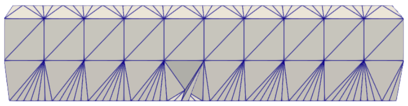

(D) Cross section of $D_{s,h}$ on the yz-plane

FIGURE 12. The whole domain mesh Ω_h, solvent mesh $D_{s,h}$, and membrane mesh $D_{m,h}$ generated by ICMPv2 for the ion channel protein α-HL.

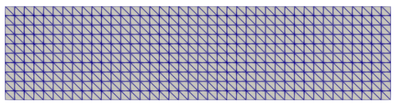
Here, the TMSmesh parameters $d = 0.4$ and $c = e = 0.9$; the numbers of vertices are 80630, 16001, 121943, 55779, 29185, and 66988 and the numbers of tetrahedra are 413086, 72646, 752618, 274675, 138411, and 339532, respectively, for the meshes $\hat{D}_{s,h}$, $D^p_{s,h}$, Ω_h, $D_{s,h}$, $D_{m,h}$, and $D_{p,h}$.

As comparison, we did tests on mVDAC using ICMPv1 too. However, ICMPv1 was found not to work on this test case. The best meshes that we produced from ICMPv1 (in the sense of containing a small number of false tetrahedra) were displayed in Figure 14. From Plot (C) it can be seen that the membrane mesh

A side view of a membrane mesh $D_{m,h}$

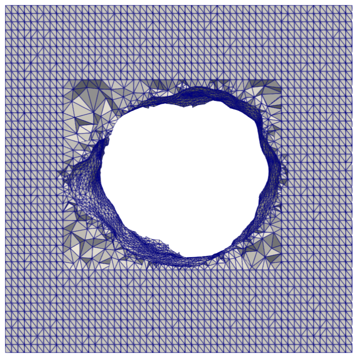


(A) By ICMPv1

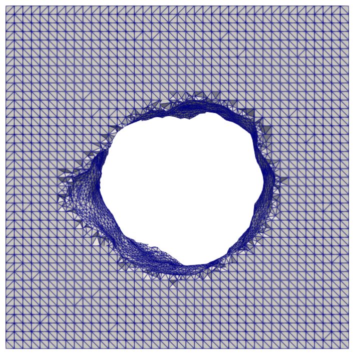


(B) By ICMPv2

A top view of a membrane mesh $D_{m,h}$

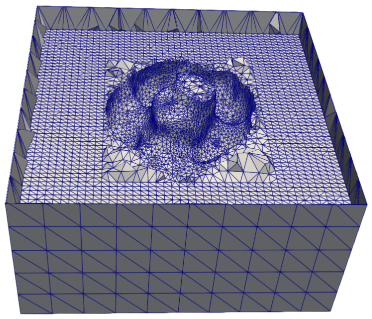


(C) By ICMPv1

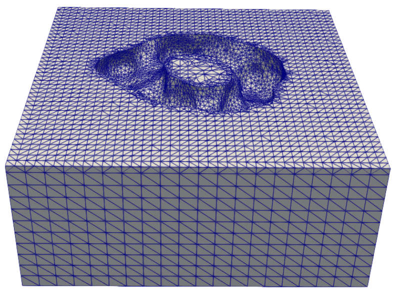


(D) By ICMPv2

A view of the bottom portion $D_{s,h}^{b}$ of a solvent mesh $D_{s,h}$



(E) By ICMPv1

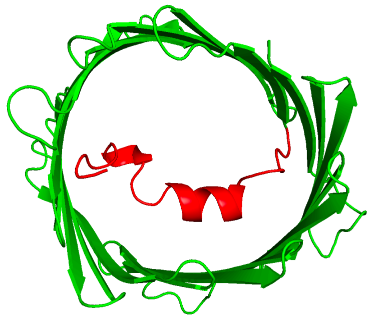


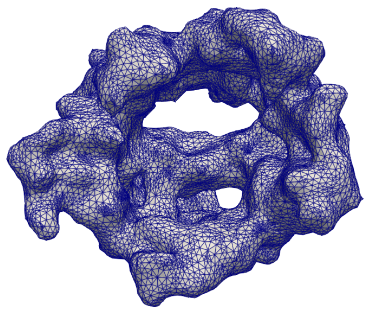
(F) By ICMPv2

FIGURE 13. A comparison of the membrane and solvent meshes $D_{m,h}$ and $D_{s,h}$ generated by ICMPv1 with those by ICMPv2 for VDAC. Here $D_{s,h}^{b}$ is defined in (8).

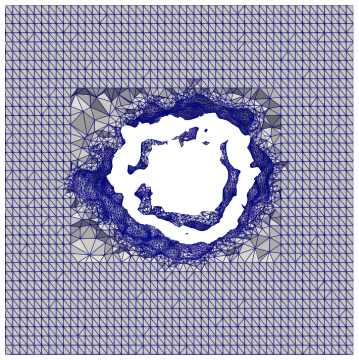
$D_{m,h}$ still contains many tetrahedra that belong to the solvent mesh $D_{s,h}$. Thus, the solvent mesh $D_{s,h}$ losses many tetrahedra so that its geometric shape has been twisted. Such poor membrane and solvent meshes may affect the approximation accuracy of a PBE/PNP finite element solution.

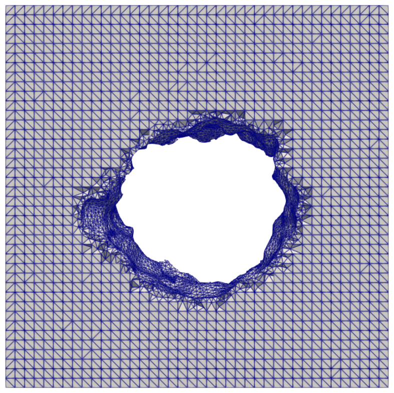
To illustrate it, we did numerical tests using the PNP finite element software package reported in [6] for the mVDAC in a salt solution of 0.1 mole potassium chloride (KCl). Since the ICMPv1 meshes reported in Figure 14 failed to work


(A) Top view of molecular structure

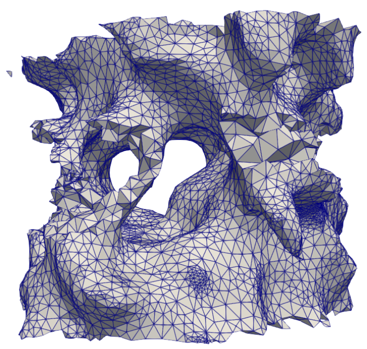

(B) Top view of $D_{p,h}$

A top view of a membrane mesh $D_{m,h}$


(C) By ICMPv1


(D) By ICMPv2

A side view of a portion $D^p_{s,h}$ of $D_{s,h}$ within the ion channel pore


(E) By ICMPv1

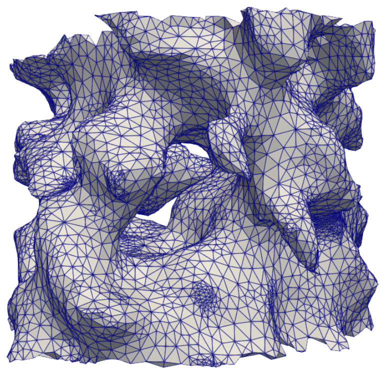
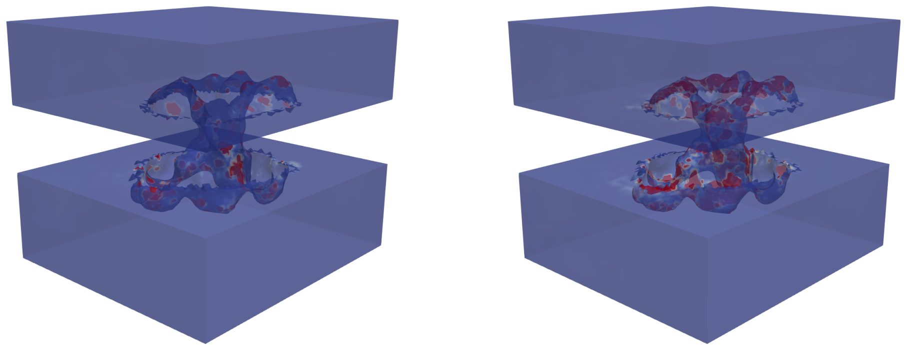

(F) By ICMPv2

FIGURE 14. (A,B): A molecular structure of mVDAC and a protein mesh of mVDAC generated by ICMPv2. (C) to (F): A comparison of the tetrahedral meshes $D_{m,h}$ and $D^p_{s,h}$ generated by ICMPv2 with those by ICMPv1 for mVDAC. Here $D^p_{s,h}$ is defined in (8) and mVDAC is a modified VDAC generated by rotating the α-helix of VDAC at the hinge region.



(A) (B)

FIGURE 15. (A) A color mapping of the potassium concentration difference $|c_1^{(1)} - c_1^{(2)}|$ on the surface of a solvent mesh, $D_{s,h}^{(2)}$, generated by ICMPv2. (B) A color mapping of the chloride concentration difference $|c_2^{(1)} - c_2^{(2)}|$ on the surface of $D_{s,h}^{(2)}$. Here the range of color mapping is from 0 (in blue) to 5 (in red) moles per liter.

for the PNP finite element solver, we generated a smoother ion channel molecular triangular surface mesh through changing the TMSmesh parameter d from 0.4 to 0.2. We then obtained the ICMPv1 meshes, which are similar to the ones reported in Figure 14 (C, E) but work for the PNP software package (in the sense that all the involving iterative schemes are convergent). With the same ICMPv1 mesh parameters, we generated the ICMPv2 meshes. We then derived the potassium concentration $c_1^{(1)}$ and chloride concentration $c_2^{(1)}$ from the ICMPv1 meshes via the PNP software package. We also similarly got the potassium concentration $c_1^{(2)}$ and chloride concentration $c_2^{(2)}$ by using the ICMPv2 meshes. In these calculations, we used the default parameter values of the PNP model and software package. Since $c_1^{(1)}$ and $c_2^{(1)}$ are defined on the solvent region mesh (with 40,212 vertices) generated by ICMPv1, which is different from the solvent region mesh $D_{s,h}^{(2)}$ (with 40,554 vertices) generated by ICMPv2, they were interpolated to $D_{s,h}^{(2)}$ so that the potassium concentration difference $|c_1^{(1)} - c_1^{(2)}|$ and chloride concentration difference $|c_2^{(1)} - c_2^{(2)}|$ were done on $D_{s,h}^{(2)}$.

Figure 15 displays the values of differences $|c_1^{(1)} - c_1^{(2)}|$ and $|c_2^{(1)} - c_2^{(2)}|$ in color mapping on the surface of the solvent mesh $D_{s,h}^{(2)}$ from the range 0 (in blue) to 5 (in red) moles per liter (i.e. all the values greater than 5 are colored in red). From the figure it can be seen that the ICMPv1 meshes significantly disturbed the PNP finite element solution generated by using the ICMPv2 meshes. We further found that the maximum values of $c_1^{(1)}$ and $c_2^{(1)}$ were about 17,001 and 12,990, respectively, which are unreasonably larger than the maximum values 615 and 299 of $c_1^{(2)}$ and $c_2^{(2)}$. These test results demonstrate that meshes in low quality can significantly affect PNP finite element solutions. Hence, updating ICMPv1 to ICMPv2 is important and necessary.

7. Conclusions

A PBE/PNP ion channel finite element solver can effectively handle the complicated geometries of a protein region, a membrane region, and a solvent region as well as an interface between two of these three regions within a simulation box domain. This remarkable feature make it a much more powerful ion channel simulation tool than the corresponding finite difference solver. However, its numerical accuracy depends on the quality of an unstructured tetrahedral mesh to be used in its implementation. The application of a PBE/PNP ion channel finite element solver can be greatly enhanced and extended through developing and improving meshing algorithms and software. Currently, the ion channel mesh software package developed in Lu's research group a few years ago, which is referred to as ICMPv1 in this paper, is the only one that works for PBE/PNP ion channel finite element solvers. To further improve it, we have presented three new numerical schemes and implemented them in Python. This work has resulted in the second version of ICMPv1, called ICMPv2 in this paper. Numerical test results on four ion channel proteins with different geometric complexities are reported in this paper, confirming that ICMPv2 can significantly improve the mesh quality and computer performance of ICMPv1 in the generation of an expanded solvent mesh and in the extraction of membrane and solvent meshes from a given expanded solvent mesh.

We need to point out that this work purely aims to generate unstructured tetrahedral meshes for the classical PBE and PNP ion channel models and their variants (such as ion size modified PBE/PNP ion channel models and a nonlocal modified PBE model [24]) that involve a piecewise constant permittivity function (i.e. treating the protein, membrane, and solvent regions of a simulation box domain as three different continuum dielectrics with distinct permittivity constants). These models have been widely applied to various ion channel simulations and will remain important and valuable in the fields of computational biochemistry and mathematical biology even though new dielectric continuum ion channel models may be developed from other modeling methodologies (e.g. smooth interface methods and continuous dielectric permittivity functions [25, 26]). Hence, in the future, we will do more numerical tests on various complex ion channel proteins and further improve the robustness and performance of ICMPv2.

Acknowledgements

This work was partially supported by the Simons Foundation, USA, through research award 711776. Chao's work was supported by 2021 AMS-Simons Travel Grant, USA. Gui and Lu's work was supported by the National Natural Science Foundation of China (Grant numbers 11771435 and 22073110).

References

[1] B. Lu, Y. Zhou, M. Holst, J. McCammon, Recent progress in numerical methods for the Poisson-Boltzmann equation in biophysical applications, Communications in Computational Physics 3 (5) (2008) 973–1009.

[2] M. J. Holst, The Poisson-Boltzmann equation: analysis and multilevel numerical solution, Online.

[3] L. Chen, M. J. Holst, J. Xu, The finite element approximation of the nonlinear Poisson-Boltzmann equation, SIAM journal on numerical analysis 45 (6) (2007) 2298–2320.

[4] D. Xie, New solution decomposition and minimization schemes for Poisson-Boltzmann equation in calculation of biomolecular electrostatics, Journal of Computational Physics 275 (2014) 294–309.

[5] T.-L. Horng, T.-C. Lin, C. Liu, B. Eisenberg, PNP equations with steric effects: a model of ion flow through channels, The Journal of Physical Chemistry B 116 (37) (2012) 11422–11441.

[6] D. Xie, Z. Chao, A finite element iterative solver for a PNP ion channel model with Neumann boundary condition and membrane surface charge, Journal of Computational Physics 423 (2020) 109915.

[7] Z. Chao, D. Xie, An improved Poisson-Nernst-Planck ion channel model and numerical studies on effects of boundary conditions, membrane charges, and bulk concentrations, Journal of Computational Chemistry 42 (27) (2021) 1929–1943.

[8] M. Chen, B. Tu, B. Lu, Triangulated manifold meshing method preserving molecular surface topology, Journal of Molecular Graphics and Modelling 38 (2012) 411–418.

[9] B. Tu, S. Bai, M. Chen, Y. Xie, L. Zhang, B. Lu, A software platform for continuum modeling of ion channels based on unstructured mesh, Computational Science & Discovery 7 (1) (2014) 014002.

[10] T. Liu, S. Bai, B. Tu, M. Chen, B. Lu, Membrane-channel protein system mesh construction for finite element simulations, Computational and Mathematical Biophysics 1 (2015) 128–139.

[11] T. Liu, M. Chen, B. Lu, Efficient and qualified mesh generation for Gaussian molecular surface using adaptive partition and piecewise polynomial approximation, SIAM Journal on Scientific Computing 40 (2) (2018) B507–B527.

[12] M. Chen, B. Lu, TMSmesh: a robust method for molecular surface mesh generation using a trace technique, Journal of Chemical Theory and Computation 7 (1) (2011) 203–212.

[13] T. Liu, M. Chen, Y. Song, H. Li, B. Lu, Quality improvement of surface triangular mesh using a modified laplacian smoothing approach avoiding intersection, PLoS One 12 (9) (2017) e0184206.

[14] D. Xie, B. Lu, An effective finite element iterative solver for a Poisson-Nernst-Planck ion channel model with periodic boundary conditions, SIAM Journal on Scientific Computing 42 (6) (2020) B1490–B1516.

[15] D. Xie, An efficient finite element iterative method for solving a nonuniform size modified Poisson-Boltzmann ion channel model, arXiv:2108.13616 and to be published on Journal of Computational Physics.

[16] J. Ahrens, B. Geveci, C. Law, Paraview: An end-user tool for large data visualization, The visualization handbook 717 (8) (2005).

[17] H. Si, K. Gärtner, Meshing piecewise linear complexes by constrained Delaunay tetrahedralizations, Proceedings of the 14th International Meshing Roundtable (2005) 147–163.

[18] H. Si, TetGen, a Delaunay-based quality tetrahedral mesh generator, ACM Transactions on Mathematical Software 41 (2) (2015) 1–36.

[19] S. Decherchi, W. Rocchia, A general and robust ray-casting-based algorithm for triangulating surfaces at the nanoscale, PloS One 8 (4) (2013) e59744.

[20] Z. Yu, M. J. Holst, J. A. McCammon, High–fidelity geometric modeling for biomedical applications, Finite Elements in Analysis and Design 44 (11) (2008) 715–723.

[21] M. F. Sanner, A. J. Olson, J. C. Spehner, Reduced surface: an efficient way to compute molecular surfaces, Biopolymers 38 (3) (1996) 305–320.

[22] P. Sjoberg, MolSurf-A generator of chemical descriptors for QSAR, Computer-Assisted Lead Finding and Optimization (1997) 83–92.

[23] T. Möller, B. Trumbore, Fast, minimum storage ray-triangle intersection, Journal of graphics tools 2 (1) (1997) 21–28.

[24] D. Xie, Y. Jiang, A nonlocal modified Poisson-Boltzmann equation and finite element solver for computing electrostatics of biomolecules, Journal of Computational Physics 322 (2016), 1–20.

[25] B. Lu, J. A. McCammon, Molecular surface-free continuum model for electrodiffusion processes, Chemical physics letters, 451(4-6) (2008) 282–286.

[26] T. Hazra, S. A. Ullah, S. Wang, E. Alexov, S. Zhao, A super-Gaussian Poisson-Boltzmann model for electrostatic free energy calculation: smooth dielectric distribution for protein cavities and in both water and vacuum states, Journal of Mathematical Biology 79 (2) (2019) 631–672.

Department of Mathematics, University of Michigan, Ann Arbor, MI 48109, USA.

LSEC, NCMIS, Key Laboratory of Systems and Control, Institute of Systems Science, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, China. School of Mathematical Sciences, University of Chinese Academy of Sciences, Beijing 100049, China.

LSEC, NCMIS, Key Laboratory of Systems and Control, Institute of Systems Science, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, China. School of Mathematical Sciences, University of Chinese Academy of Sciences, Beijing 100049, China.

Department of Mathematical Sciences, University of Wisconsin-Milwaukee, Milwaukee, WI 53201, USA. Corresponding author.
E-mail: `dxie@uwm.edu`.